**TEHNIČKO VELEUČILIŠTE U ZAGREBU**

**POLITEHNIČKI SPECIJALISTIČKI DIPLOMSKI STRUČNI STUDIJ**

**Specijalizacija informatika**

Igor Stanković

Primjena ispitivanja temeljenog na svojstvima unutar integracijskog ispitivanja sustava

DIPLOMSKI RAD

Zagreb, veljača, 2023.

**TEHNIČKO VELEUČILIŠTE U ZAGREBU**

**POLITEHNIČKI SPECIJALISTIČKI DIPLOMSKI STRUČNI STUDIJ**

**Specijalizacija informatika**

Primjena ispitivanja temeljenog na svojstvima unutar integracijskog ispitivanja sustava

DIPLOMSKI RAD

Povjerenstvo:

Zagreb, veljača, 2023.

**Sažetak**

U ovome radu objašnjena je važnost testiranja programskog koda kao i važnost testiranja programskog sustava kao cjeline. Navedeno je koji razvojni okviri i koji programski jezik omogućavaju takvu vrstu testiranja te njihove prednosti u odnosu na druge. Također je naglašena važnost detaljnog testiranja temeljenog na svojstvima gdje se prolazi kroz sve moguće slučajeve koji se lako mogu propustiti tijekom testiranja. U praktičnom djelu demonstrirana je provedba integracijskog testiranja temeljnog na svojstvima nad sustavom kojeg čini logiRECORDER uređaj i *Dashboard* aplikacija kojom se uređaj upravlja. Cilj rada je prikazati uspješnost takve vrste testiranja, objasniti arhitekturu projekta koja se temelji na objektno orijentiranom načinu programiranja te demonstrirati testiranje temeljno na svojstvima za testiranje parsiranja podataka CAN sabirnice koja predstavlja vrlo bitnu značajku *Dashboard* aplikacije.

Ključne riječi: test, integracija, PyTest, Hypothesis, parametar, strategija, CAN

Sadržaj

[1. Uvod 1](#_Toc126163510)

[2. Integracijsko testiranje 2](#_Toc126163511)

[2.1. Zašto integracijsko testiranje? 2](#_Toc126163512)

[2.2 Strategije za testiranje integracije 3](#_Toc126163513)

[2.3 Kako provesti integracijsko testiranje? 4](#_Toc126163514)

[3. PyTest razvojni okvir 5](#_Toc126163515)

[3.1 Kako i zašto koristiti PyTest 5](#_Toc126163516)

[3.2 Grupacija testova 7](#_Toc126163517)

[3.3 PyTest metoda učvršćenja 9](#_Toc126163518)

[3.4 Parametrizacija pomoću parametara 10](#_Toc126163519)

[3.5 Preskakanje testova 12](#_Toc126163520)

[4. Testiranje temeljeno na svojstvima 13](#_Toc126163521)

[4.1 Korištenje hipototeze 15](#_Toc126163522)

[4.2 Uspješnost testiranja temeljnog na svojstvima 20](#_Toc126163523)

[5. Ispitni sustav i alati za izradu 21](#_Toc126163524)

[5.1 Alati za izradu 24](#_Toc126163525)

[5.2 Virtualno okruženje 24](#_Toc126163526)

[5.3 Biblioteke unutar virtulanog okruženja 25](#_Toc126163527)

[6. Praktični prikaz testiranja unutar PyTest okvira 27](#_Toc126163528)

[6.1 Temelji projekta 28](#_Toc126163529)

[6.2 CAN protokol 31](#_Toc126163530)

[6.2.1 Struktura CAN okvira 32](#_Toc126163531)

[6.2.2 CAN baze podataka 33](#_Toc126163532)

[6.2.3 CAN Motbus baza podataka 34](#_Toc126163533)

[6.3 Testiranje parsiranja CAN baza unutar PyTesta 37](#_Toc126163534)

[7. Zaključak 61](#_Toc126163535)

[9. Literatura 62](#_Toc126163536)

**Popis oznaka, kratica**

**Popis slika**

[Slika 1. Primjer testne metode [5] 6](#_Toc126184276)

[Slika 2. Neuspješan ishod testa [5] 6](#_Toc126184277)

[Slika 3. Nazivi PyTest datoteka [6] 7](#_Toc126184278)

[Slika 4. PyTest marker [6] 8](#_Toc126184279)

[Slika 5. PyTest Fixture[6] 9](#_Toc126184280)

[Slika 6. Parametri testne metode [6] 10](#_Toc126184281)

[Slika 7. Rezultat testa [6] 11](#_Toc126184282)

[Slika 8. Kobinacije parametara [7] 11](#_Toc126184283)

[Slika 9. Dekoratori za markere [6] 12](#_Toc126184284)

[Slika 10. Testna funkcija [9] 15](#_Toc126184285)

[Slika 11. Funkcije unutar testa [9] 16](#_Toc126184286)

[Slika 12. Test temeljen na svojstvima [9] 16](#_Toc126184287)

[Slika 13. Pronalazak greške [9] 17](#_Toc126184288)

[Slika 14. Kompozitna strategija [9] 18](#_Toc126184289)

[Slika 15. Funkcija unutar testa [9] 18](#_Toc126184290)

[*Slika 16. Koirštenje prepostvke [9]* 19](#_Toc126184291)

**Popis tablica**

[Tablica 1. Usporedba dva načina testiranja 15](#_Toc112580259)

# 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 istestirati 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 čovjek je drugačiji tako i 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 automatizacijom testova koji se po potrebi mogu izvršavati svakoga dana kako bi se odmah uočila potencijalna opasnost u izmjenjivanju postojećeg koda. 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 testiranje temeljno na svojstvima. Navedene tvrdnje ispitane su unutar šestog poglavlja gdje je provedena takva vrsta testiranja nad sustavom za snimanje i reproduciranje podataka video kamera i sabirnica unutar vozila opisanog u petom poglavlju s ciljem pronalaženja teško uočljivih grešaka i rasterećivanjem testera koji sustav ručno testiraju.

# Integracijsko testiranje

Integracijsko testiranje je vrsta testiranja koja provjerava ispravan način rada pojedinih jedinica softvera kada se one povežu u jednu cjelinu za razliku od Unit testiranja kod kojeg se pojedine jedinice testiraju zasebno neovisno jedna o drugoj.

Svrha integracijskog testiranje je provjera funkcionalnosti sustava kada se svi njegovi moduli povežu u cjelinu te radi li sustav onako kako se to od njega očekuje.Takvo testiranje se postiže tako da se najprije provjere zasebni moduli ili jedince pojedinog sustava te se zatim provjerava ispravnost modula kada sustav radi s više modula u isto vrijeme. [1]

## Zašto integracijsko testiranje?

Svaki gotov sustav trebao bi raditi bez grešaka i onako kako je zamišljeno, no često se kod dizajniranja ili redizajniranja sustava ne uviđaju moguće pogreške ili nestabilnosti u radu. Na dizajniranju većih sustava radi više programera koji pišu zasebne module kod kojih se logika i razumijevanje međusobno razlikuje tako da se njihova cjelina svakako treba provjeriti. Često se zahtjevi klijenata mijenjaju pa je potrebno ažurirati pojedine cjeline u skladu sa zahtjevima. Mijenjanjem pojedinih cjelina ne smije se narušavati rad ostalih sustava i ostalih modula koji se nalaze u njemu. Svakako je važno pripaziti da softverska sučelja rade u skladu s hardverskim što integracijski testovi dobro pokrivaju te su se pokazali vrlo korisnim. [1]

## 2.2 Strategije za testiranje integracije

Kod integracijskog testiranja postoje dvije osnovne strategije za provođenje integracijskog testiranja Pristup Velikog praska i Inkrementalni pristup. Inkrementalni pristup se još dijeli na Pristup odozgo prema dolje, Pristup odozdo prema gore i kombinacija ta dva pristupa. [2]

Pristup velikog praska je tehnika kod kojeg se pojedini koraci ne provjeravaju dok nisu ispunjeni koraci niže razine koji su ključni za obavljanje koraka više razine.Npr. pojedini test više razine ne može nikako proći provjeru jer znamo da je prije njega pao test niže razine koji neophodan parametar za njegovu provjeru.

Test integracije odozdo prema gore je tehnika testiranja bazirana na strategiji inkrementalnog pristupa gdje se pojedini moduli koji se testiraju postepeno i logički nadopunjuju i povezuju u cjelinu sve dok svi moduli nisu testirani. Kod tehnike odozdo prema gore prvo se testiraju moduli niže razine te se postepeno napreduje prema modulima viših razina. Prednosti takve tehnike je što je lakše pronaći greške i ne gubi se vrijeme čekajući da svi moduli budi kodirani, no kod takve tehnike rani prototip nije moguć.

Test integracije odozgo prema gore je još jedna od tehnika inkrementalnog pristupa samo što se sada prvo testiraju moduli najviših razina te se postepeno spušta prema modulima nižih razina. Prednosti ovo tehnike je rani prototip i brže otkrivanje grešaka koje su prve uočljive kod korisnika dok je mana to što su moduli na nižoj razini slabije istestirani. [2]

## 2.3 Kako provesti integracijsko testiranje?

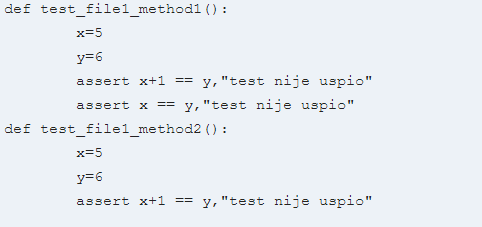
Kao prvi korak za provedbu integracijskog testiranje potrebna je izrada plana testiranja i integracije. To znači odabrati ispravnu strategiju i pristup testiranja, odrediti uloge i odgovornosti ispitivača, odrediti zahtjeve za ispitivanje, odabrati ispitno okruženje te odrediti razlike i mjere. Nakon toga potrebno je dizajnirati testni scenarij, test slučajeve i testne skripte. Kada su svi potrebni zahtjevi dizajnirani kreće se u izvođenje testnih slučajeva nakon čega se analiziraju dobivani rezultati i prijavljuju se greške i nedostaci. Izvođenje testnih slučajeva i analiziranje se provodi sve dok se integracija ne uspostavi u potpunosti. Integracijsko testiranje sustava se ne može provesti prije negoli su sve komponente sustava podvrgnute jediničnom testu. Svi nedostaci koji su od visokog prioriteta moraju biti riješeni i zatvorni, a svi moduli moraju biti spremni i mogu se integrirati. Testni plan integracije, testni scenariji kao i testni slučajevi trebaju biti dizajnirani i validirani. Kriterij kod kojeg se integracijsko testiranje zaustavlja mora zadovoljiti određene zahtjeve. Integriran sustav uspješno je testiran i svi testni slučajevi su dokumentirani. Svi nedostaci s visokim prioritetom su riješeni i zatvoreni, a tehnička dokumentacija je isporučena. [2]

# PyTest razvojni okvir

Jedan od razvojnih okvira za testiranje aplikacija je PyTest koji omogućuje testiranje programskog koda odnosno pisanje jednostavnih i skalabilnih testnih slučajeva za testiranje baze podataka, API-a i korisničkog sučelja. Razvojni okvir PyTest uglavnom se koristi za testiranje programskog sučelja aplikacije u kojem se nudi široki raspon kompleksnosti testova od unit testova do kompleksnih funkcionalnih testova unutar programskog jezika python. Python je programski jezik temeljen na orijentiranom objektnom programiranju koji teži jednostavnosti. Njegova sintaksa je vrlo jednostavna te u odnosu na ostale jezike sadrži mnogo manje informacija o varijablama, funkcijama i samim objektima. Nastoji izbjeći referencirane implementacije koji inače štede memoriju i brže obavljaju svoje funkcije po cijeni jednostavnosti i sažetosti. Python je vrlo moćan jezik koji je dizajniran da bude vrlo proširiv s velik brojem specijaliziranih biblioteka koje je moguće uključiti u vlastito programsko okruženje, neke od biblioteka koje se koriste za izradu rada bit će detaljnije opisane u nastavku. [3]

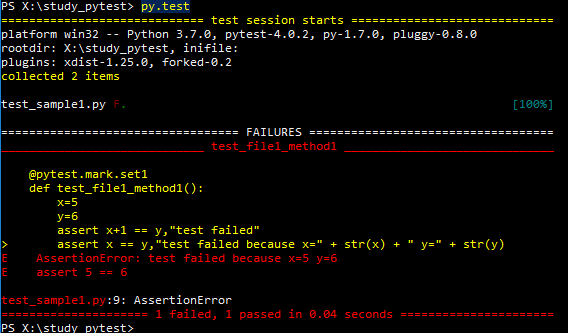
## 3.1 Kako i zašto koristiti PyTest

PyTest je vrlo jednostavan besplatan razvojni okvir koji nudi varijaciju mogućnosti kod pisanja vlastitih testnih slučajeva. Jedne od brojnih mogućnosti uključuju pokretanje specifičnog testa ili grupu testova unutar skupine testova na temelju zadanog uvjeta, automatsko prepoznavanje testova, preskakanje pojedinih testova, izvršavanje više testnih slučajeva istovremeno čime se smanjuje trajanje izvršenja itd. Glavna smisao testnih slučajeva su PyTest tvrdnje koje vračaju status istine ili laži. Ako tvrdnja u testnoj metodi ne uspije odnosno ako je lažna tada se izvršenja 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 testnih metoda može se vidjeti na slici 1. [5]



Slika 1. Primjer testne metode [5]

Pokretanjem testnih metoda prikazanih na slici 1 kao rezultat na terminalu dobit sljedeće prikazano na slici 2. [5]

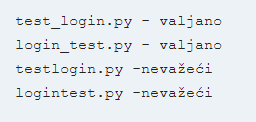


Slika 2. Neuspješan ishod testa [5]

Na slici 2 vidljivo je da su prikupljena 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 jednako broju 6 lažna, što uzrokuje izlaz iz testne metode i vračanjem pogrešne tvrdnje (engl. *AssertionError*) uz dodatnu poruku o informacijama pogreške. Vrlo bitan dio kod postavljanja tvrdnji unutar testnih metoda je poruka koja će se ispisati uz lažnu tvrdnju kako bi korisnik testova imao što više informacija o tome koji dio koda nije zadovoljio tvrdnju i zašto. [5]

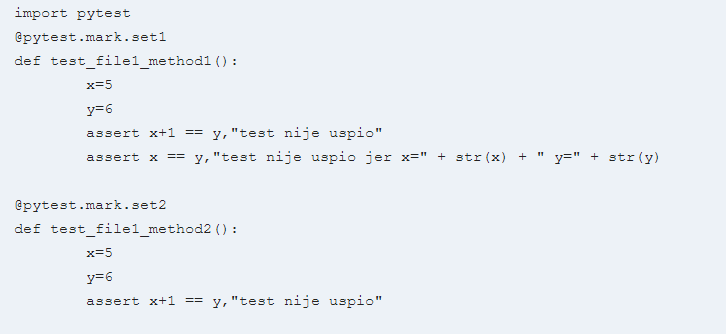
## 3.2 Grupacija testova

Prema već zadanim postavkama PyTest identificira nazive datoteka koji počinju na test ili završavaju na test kao testne datoteke, no eksplicitno se mogu spomenuti i druga imena datoteka. Isto tako PyTest zahtjeva da nazivi testnih metoda počinju na test jer svi nazivi koji ne počinju na test bit će zanemareni čak i ako je izričito zatraženo pokretanje tih metoda. Na slici 3. prikazani su neki važeći i nevažeći nazivi PyTest datoteka. [6]



Slika 3. Nazivi PyTest datoteka [6]

Naredbom *pytest* pokrenuti će se svi nazivi datoteka koji počinju na test ili završavaju na test u toj mapi i podmapama u toj mapi. PyTest nudi opciju grupiranja naziva testova prema podnizu podudaranja i grupiranje po markerima. Da bi se pokrenuli testovi koji u nazivu imaju method1 potrebno je pokrenuti naredbu *pytest –k method1 –v*. Kako bi se koristili markeri potrebno ih je definirati dekoratorima iznad testne metode kao na slici 4. [6]

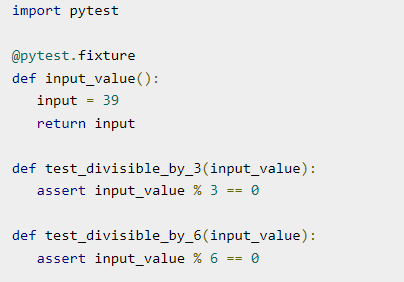


Slika 4. PyTest marker [6]

Naredbom *pytest –m set1* pokrenuti će se samo test *test\_file1\_method* koji ima dekorator *pytest.mark.set1* dok će test *test\_file1\_method2* biti zanemaren. [6]

## 3.3 PyTest metoda učvršćenja

Unutar PyTests okvira postoje posebne funkcije koje se nazivaju funckije učvršćenja (engl, *Fixture functions*). Funkcije učvrščenja se pokreću prije testnih funkcija na koje se primjenjuju. One služe za unos podataka u testove kao što se URL-ovi za testiranje, veze i informacije vezane za bazu podataka i neke druge vrste ulaznih podataka. Tako sprječavamo pokretanje istog koda za svaki test jer jednom kada se funkcija učvršćenja izvrši njezini izlazni podaci dostupni su svim testovima u istom modulu. Primjer uporabe funkcije učvršćenja prikazan je na slici 5. [6].



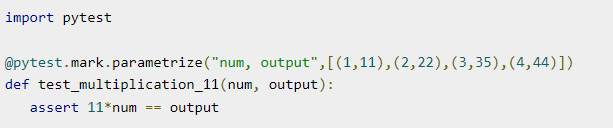
Slika 5. PyTest Fixture[6]

Funkcija *input\_value()* je funkcija učvršćenja koja vraća tip podataka koji je cijeli broj . Testovi *test\_divisible\_by\_3* i *test\_divisible\_by\_6* kao ulazni parametar definiraju ime varijable koja odgovara imenu funkcije učvršćenja te tako PyTest prepoznaje unutar testne metode mora proslijediti izlaznu vrijednost funkcije učvršćenja. Kako je ranije spomenuto“ funkcija učvrščenja koja je definirana u određenom modulu ima opseg vidljivosti samo tom modulu. Kako bi bila vidljiva i u drugim modulima potrebno ju je definirati u modulu pod nazivom *conftest.py*. Tada će pokrenuti testovi tražiti funkcije

učvršćenja u istom modulu u kojem se nalaze, no ako ne uspiju naći traženu funkciju nastavit će tražiti u unutar *conftest.py* modula. [6]

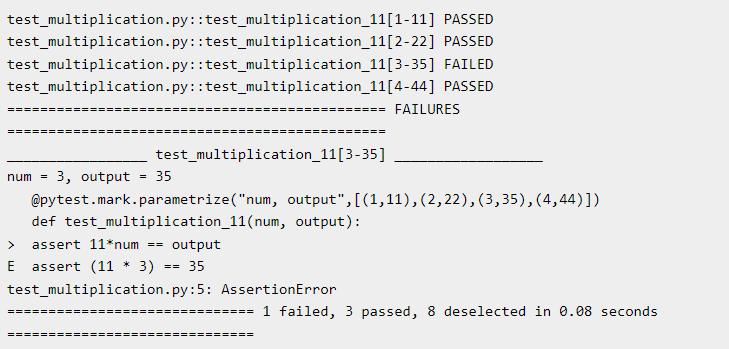
## 3.4 Parametrizacija pomoću parametara

Parametrizacija testa je dodatak unutar Pytest okvira koji omogućuje izvođenje pojedinog testa za više skupova ulaznih podataka. Kako bi se testnoj metodi definirali ulazni parametri potrebno je koristi marker dekorator *@PyTest.mark.parametrize.* Primjer korištenja parametara prikazano je na slici 6.



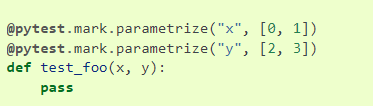
Slika 6. Parametri testne metode [6]

Na slici je prikazan test koji prima kao ulazne parametre dva broja i provjerava jednakost prvog broja pomnoženog s brojem 11 s drugim brojem. Nakon pokretanja gore prikazanog koda test će izgenerirati sljedeće prikazano na slici 7.



Slika 7. Rezultat testa [6]

Na slici je vidljivo kako se isti test *test\_multiplication\_11* izvršio četiri puta. Test je pokrenut za svaki predani tuple u listi parametara od kojih je treći tuple po redu izvršavanja uzrokovao grešku koja ne ispunjava pretpostvku da je jednakost broja 3 pomnoženo s brojem 11 jednako broju 35. Također PyTest nudi nizanje više danih parametara za isti test. Tada će se test izvršiti sve moguće kombinacije predanih parametara. Da bi se dobile sve kombinacije više parametriziranih argumenata, moguće je složiti parametrizacijske markere kao na slici 8. [6, 7]

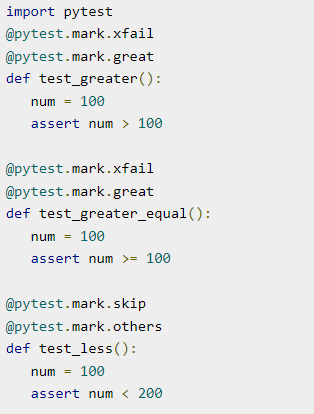


Slika 8. Kobinacije parametara [7]

U slučaju postavljanja parametara kao na slici 8 test će se izvršiti sa sljedećom kombinacijom ulaznih parametara: x=0/y=2, x=1/y=2, x=0/y=3 i x=1/y=3.

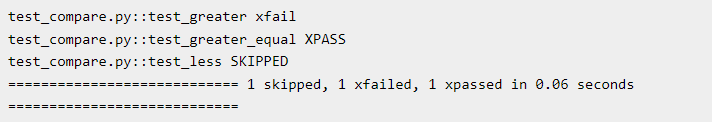
## 3.5 Preskakanje testova

Ponekad pojedine testove koji su napisani potrebno je preskočiti zbog što test nije relevantan neko duže vrijeme ili je neka nova značajka u implementaciji a već je dodana test koji ispituje tu značajku. U navedenim situacijama postoji opcija dekoratora za markere markera *xfail* koji će omogućiti izvršenje testa, ali ga neće smatrati neuspjelim ako ne prođe i neće ispisivati detalje vezene uz neuspjelu pretpostavku. Druga opcija je dekorator *skip* koji će u potpunosti prekositi test koji ima definiran takav marker. Primjer korištenja navedenih markera prikazano je na slici 9. [6]

****

Slika 9. Dekoratori za markere [6]

Nakon pokretanje koda prikazanog na slici 9 dobije se sljedeći rezultat prikazan na slici 10.



Slika 10. Rezultat testova s marker dekoratorima [6]

Za testove *test\_grater* i *test\_greater\_equal* prikazan je uspješnost izvršenja testova s naznakom da se radi o test s markerom *xfail*. Test *test\_less* s markerom skip je preskočen i za njega nije poznat ishod izvršenja.

# Testiranje temeljeno na svojstvima

Kako bi se izbjeglo testiranje sa konstantno istim ulaznim parametrima i unaprijedili testovi napisani u programskom jeziku python koristi se razvojni okvir hipoteza (engl, Hypothesis). Hypothesis je Pythonova biblioteke za izradu testova temeljenih na svojstvima koji su po strukturi jednostavniji, ali u mnogim slučajevima snažniji i detaljniji kod izvođenja. Vrlo stabilan i moćan radni okvir kojeg je vrlo lako ukomponirati unutar PyTest razvojnog okvira. Testiranje hipotezom se još naziva i testiranje temeljno na svojstvima koje može otkriti i najtvrdokornije bugove pomoću pametnih algoritama. Radi tako da testira rubne slučajeve i sve one slučajeve kojeg se autor testa ne bi mogao sjetiti prilikom pisanja testa. Razlika u izvoženju standardnih testova i testova temeljnih na svojstvima prikazano je u tablici 1.[8]

Tablica 1. Usporedba dva načina testiranja

|  |  |
| --- | --- |
| **Standarni test** | **Test temeljen na svojstvima** |
| Postavljanje ulaznih parametara | Generiranje nasumičnih ulaznih parametra koji odgovaraju nekoj strategiji |
| Izvršavanje testnih metoda nad parametraima | Izvršavanje testnih metoda nad parametraima |
| Provjera pretpostavki o ulaznim parametrima | Provjera pretpostvki o ulaznim parametrima |

Radi tako da generira proizvoljne podatke koji odgovaraju postavljenim specifikacijama i provjerava vrijede li pretpostavke postavljene u testu za generirani slučaj. Ako pronađe primjer tamo gdje ne vrijede, uzima taj primjer i pojednostavljuje ga dok ne pronađe mnogo manji primjer koji i dalje uzrokuje problem. Zatim sprema taj primjer za kasnije, tako da jednom kada pronađe problem u testiranom kodu ne zaboravi ga u budućnosti. [8]

Pisanje testova ovog oblika obično se sastoji od odlučivanja o jamstvima koja bi testni kod trebao dati. Svojstva koja bi uvijek trebala biti istinita, bez obzira na to što se prosljedi kao ulazni parametar. Primjeri takvih jamstava mogu biti:

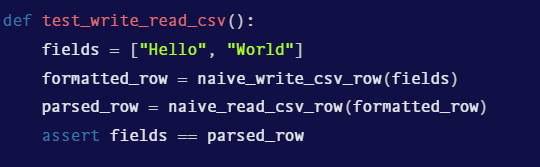
* Kod ne bi trebao izbaciti iznimku ili bi trebao izbaciti samo određenu vrstu iznimke (ovo funkcionira posebno dobro ako postoje mnogo internih tvrdnji).
* Ako se izbriše objekt, on više nije vidljiv.
* Ako se serijalizira, a zatim deserijalizira vrijednost, tada će se dobiti ista vrijednost natrag.

Jedne od značajki koje krase testove temeljenih na svojstvima su:

* Iako koriste slučajnost generiranja podataka koristi prethodno izgenerirane podatke koji su rezultirali uzrok greške
* Dobiva detaljan odgovor kako i zašto je test pao te nudi opciju reproduciranja ulaznih parametara koja su uzrokovala pad testa
* Nudi opciju proširivanja strategija koje se koriste za generiranje ulaznih parametara. [8] [9]

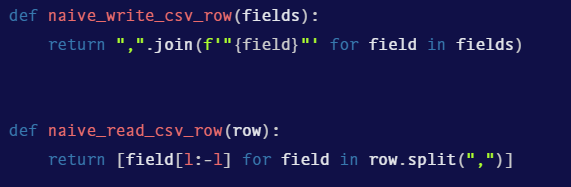
## 4.1 Korištenje hipototeze

Hypothesis razvojni okvir pruža strategije za veliki broj ugrađenih tipova s argumentima za ograničavanje ili prilagođavanje izlaza, kao i kompleksnije strategije koje se mogu sastaviti za generiranje složenijih tipova. Jedan od najkorištenijih tipova u programiranju je niz znakova. Hypothesis nudi strategiju za kreiranje teksta s ulaznim parametrima minimalnog i maksimalnog broja znakova u nizu te Alphabeta od kojeg se želi kreirati tekst. Na slici 10. prikazan je *test write\_read\_csv().* [8] [9] [10]



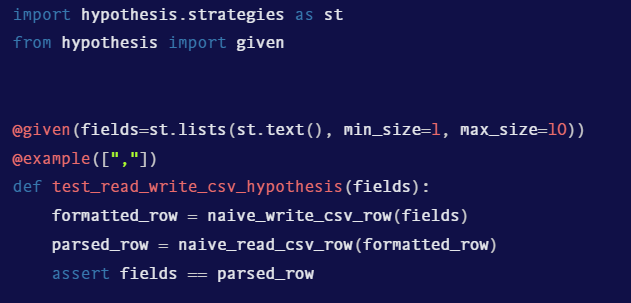
Slika 10. Testna funkcija [9]

Test kao ulazni parametar sadrži listu stringova i testira dvije funkcije od kojih jedna kao ulazni parametar prima listu nizova i vraća jedan niz sastavljen od svih elemenata liste odvojenih zarezom, dok druga funkcija radi suprotno. Kod funkcija prikazan je na slici 11. [9]



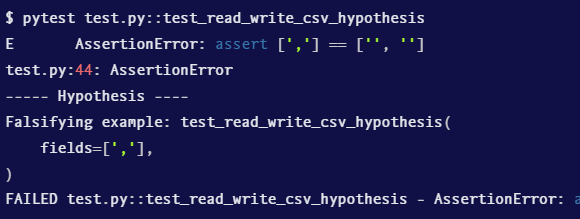
Slika 11. Funkcije unutar testa [9]

Prema tome pretpostavka da ulazna lista mora biti jednaka listi koju vraća funkcija *naive\_read\_csv\_row()* vrijedi za listu koja je inicijalizirana u primjeru. Ako test prikazan na slici 11. napišemo kao test temeljen na svojstvima onda bi ona izgledao kao na slici 12. [9]



Slika 12. Test temeljen na svojstvima [9]

Test kao ulazni parametar prima strategiju koja generira listu nizova koji mogu biti od veličine 1 do 10 znakova. Testu još dodan i dekorator *@example* u kojem se može specificirati uzorak kojeg želimo da strategija sigurno izgenerira. Kada se pokrene test sa slike 13. dobije se rezultat prikazan na slici 13. [9]

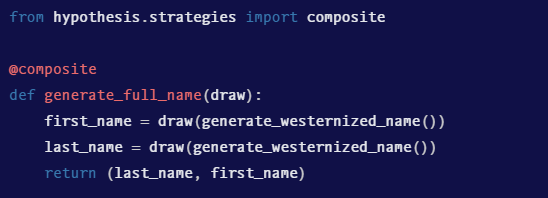


Slika 13. Pronalazak greške [9]

Za ulazni parametar koji se uzima iz primjerka prikazanog na slici 13. ishod testa je neuspješan kod provjere pretpostavke da je izlazna lista koju je vratila metoda *naive\_read\_row* ista ulaznoj listi koja je proslijeđena testu kao parametar. [9]

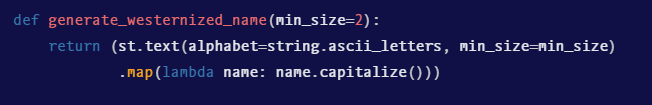
#### Kreiranje vlastitih strategija

Iako biblioteka hipototeze nudi veliki broj gotovih strategija za razne tipove podataka nudi i opciju kreiranje vlastitih strategija. Dekorator *@composite* omogućuje pisanje imperativnog koda pomoću kojeg se točno određuje što će se uzimati za predaju parametara. Primjer *@composite* dekoratora prikazan je na slici 14. [9] [10]



Slika 14. Kompozitna strategija [9]

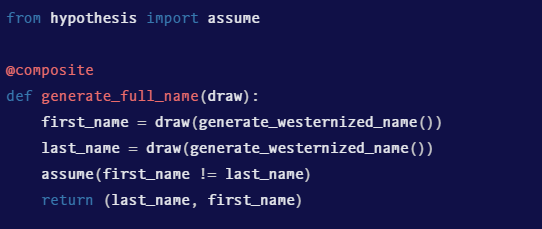
Funkciji koja ima dekorator *@composite*  potrebno je definirati parametar *draw* pomoću koje se dobivaju generirane vrijednosti strategija. Funkcija generira varijable *first\_name* i *last\_name* preko funkcije *generate\_westernied\_name* koja vraća strategiju prikazanu na slici 14. Zatim se dvije izgenerirane strategije spremaju u tuple koji se potom predaju testu prikaznom na slici 15. [9] [10].



Slika 15. Funkcija unutar testa [9]

Ukratko definirana je strategija koja u sebi u komponira više strategija koje potom vračaju generiran tip podataka koji potreban za izvršavanje testa. Kada se koristi strategija koja ima *@composite* dekorator za predaju parametara nije moguće koristi *exemple* metodu kao na primjeru sa slike 13.

Jer tada se prekida sposobnost da pravilno sintetiziraju testni slučajevi. Ako strategija izgenerira neki primjer koji ne odgovara potrebama testa moguće je koristiti funkciju *assume* pomoću koje se definira koji primjeri ne zadovoljavaju potrebu. Primjer funkcije *assume* prikazan je na slici 16. [9] [10]



*Slika 16. Koirštenje prepostvke [9]*

Funkcija *assume* navodi pretpostavku koju hipoteza mora ispuniti ako se želi generirati primjer iz strategije *generate\_full\_name*. Poput *assert* izjave unutar PyTest okvira funkcija *assume()* podučava hipotezu što je, a što nije zadovoljavajući primjer. [9]

#### Pravila kreiranje vlastitih strategija

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

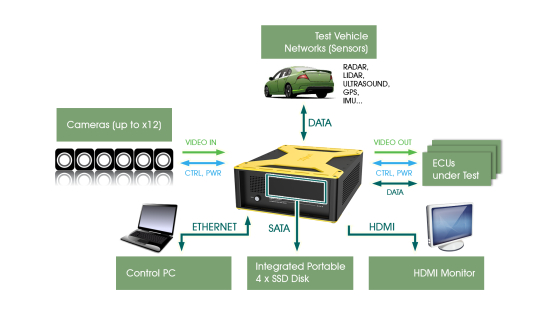
To je korisno kada su strategije neuspješne ili ne generiraju primjere kakvi su očekivani zato što se onda vrlo lakše otkrivaju greške kod pisanja strategija jer se one mogu provjeravati redom. Velike ugniježđene strategije teže je razriješiti i još teže rasuđivati. Važno je napomenuti da vlastite strategije pišu samo kad se moraju imperativnim strategijama koje koristite „assume“ treba više vremena da se konvergiraju u valjnom primjeru. [9]

## 4.2 Uspješnost testiranja temeljnog na svojstvima

Mnoge IT tvrtke testiraju svoj programski kod s testovima koji su bazirani na nekakvom primjeru odnosno testnom slučaju kao što su Unit testovi, UI testovi i QA testiranje. Kod takvog testiranja bolji je uvid u to što se testira no potrebno je potrošiti više vremena svaki puta kada se testira neka značajka. Isto važno napomenuti da takvo testiranje pokriva najmanje mogućih ulaznih parametar koji bi mogli prouzročiti nekakav kvar. Neke tvrtke su otišle korak ispred te koriste testove koji su složeniji i bazirani su na statističkoj analizi. Testovi takve strukture pokrivaju veći broj mogućih ulaza, no pružaju slabiji uvid u ono što se testira. Tvrtka Quviq AB osnovna je 2006. Godine te je danas vodeća tvrtka za testiranje programskog koda koja je među prvim počela s 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. Quviq je općenito primijenio testiranje temeljno na svojstvima za 5G radio bazne stanice i AUTOSAR programe pokazujući deset puta veću učinkovitost testiranja čak 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] [12]

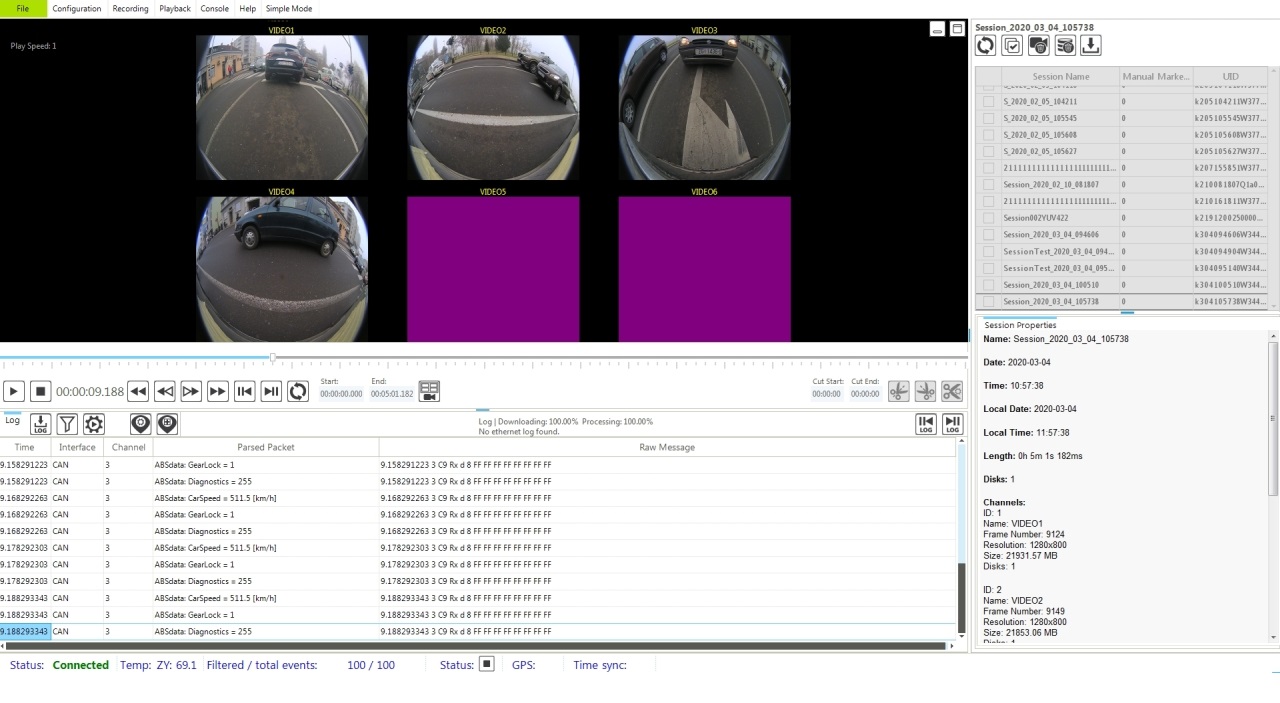
# Ispitni sustav i alati za izradu

U nastavku rada provedeno je integracijsko testiranje temeljno na svojstvima nad sustavom proizvedenim od strane tvrtke Xylon. Sustav se sastoji od logiRECORDER 3 uređaja, pomoćnih komponenti i desktop aplikacije kojom se upravlja uređajem. Razvoj i testiranje vrhunskih sustava autonomne vožnje i ADAS sustava koji se temelje na vizualnom pregledu zahtijeva mnogo testiranja različitih situacija vožnje u različitim uvjetima na cesti. Dugotrajni, skupi i ograničeni testni scenarij za testiranje na razvoju sustava u prometu može se u velikoj mjeri zamijeniti s logiRECORDER uređajem. logiRECORDER omogućuje snimanje podataka s niskom latencijom neobrađenih višekanalnih video i mrežnih podataka, analizu podataka i sinkronu reprodukciju zapisanih podataka u realističnim Hardware-in-the-Loop (HIL) simulacijama koje reproduciraju video sustav testnog vozila s više kamera i podataka sučelja u vozilu. Paralelno je moguće koristiti 20 različitih sućelja ulazno/izlaznih jedinca, uključujući video module (FPD-Link 3, GMSL 2 itd), automobilske mrežne module (CAN, LIN, FlexRay itd.) i Internet module (Smart Ethernet, BroadR-Reach). [13]



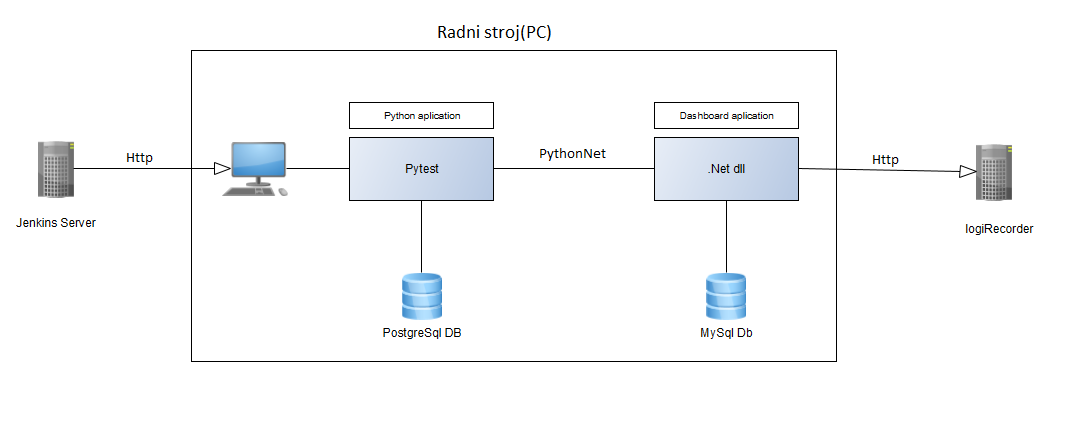
17. logRecorder uređaj [13]

Kako bi korištenje logiRECORDER uređaja bilo jednostavnije napravljena je Windows desktop aplikacija *Dashboard* kojom je omogućeno upravljene uređajem kao i vizualizacija svog prometa i video kamera spojenih u uređaj. Aplikacija omogućuje potpunu kontrolu nad svim značajkama uređaja. Pruža mogućnost mrežnom načinu rada na osobnom računalu ili tablet računalu povezanom s rekorderom putem kabela ili bežične veze ili na računalu za izvanmrežnu analizu zapisa testnih podataka bez povezanog logiRECORDERa . Prikaz prozora za reprodukciju snimaka koje sadržavaju video i CAN promet prikazan je na slici 18. [14]



*Slika 18. Dashboard aplikacija [14]*

Svrha projekta je svakodnevno testiranje logiRECORDER sustava i Dashboard aplikacije, a to se postiže automatskim pokretanjem testova koji testiraju trunk verzije firmware logiRECORDERa i Dashboard projekta te njihovu međusobnu komunikaciju. Komunikacija između Dashboard aplikacije i logiRECORDER firmwarea provodi se http protokolom preko kojeg se šalju http zahtjevi na uređaj kojima se pokreću željene naredbe. Integracijski testovi koristeći pythonNet biblioteku testiraju klase i metode iz prethodno buildanog Dashboard projekta. Za automatiziranje testova koristi se Jenkins server koji pokreće Jenkins poslove koji izvršavaju testove u postavljenom vremenu i određenim redoslijedom. Unutar Jenkinsa postoje izvješća koja nakon završetka Jenkins posla nude pregled ishoda svakog pojedinog testa. Za detaljniju analizu testova i mogućnost personalizacije rezultata po potrebi te slaganje pojedinih upita rezultati testova kao i svi koraci unutar testova spremaju se u PostgreSQL bazu podataka. Na slici 19 prikazana je shema sustava.



*Slika 19. Shema ispitnog sustava*

## 5.1 Alati za izradu

Kod izrade projekta i prikaz rezultat testiranja potrebna instalacija određenih alata i paketa koje pružaju podršku kreatoru testova i njihovim korisnicima. Za izradu programski koda projekta izabran je programski jezik Python verzije 3.8.10. Python je odabran zbog svoje dinamike i velike količine otvorenih paketa koje olakšavaju izradu vlastitog proizvoda. Dvije najvažnije biblioteke koje se koriste za izradu ovoga projekta su PyTest i Hypothesis biblioteka opisani u ranijem poglavlju. Tu su i ostale biblioteke koje će biti opisane nešto kasnije kroz seminar. Što se tiče izrade projekta u programskom jeziku Python preporučuje se koristiti razvojno okruženje PyCharm kao profesionalan alat za izradu Python projekata koji pruža određene značajke za lakšu I bržu izradu pojedinih zadataka u svakodnevnom radu. Za prikaz i pohranjivanje rezultata testova korist se relacijska baza podataka PostgreSQL, a za automatizirano pokretanje testove i dobivanje izvješća nakon završetka izvršavanja testa koristiti se poslužitelj za automatizaciju otvorenog koda Jenkins. Za lakše snalaženje i jednostavniji prikaz projekta u obliku dijagrama koristiti se yED program za dijagrame opće namjene.

## 5.2 Virtualno okruženje

Od verzije Pythona 3.5 moguće je koristiti ugrađen modul zvan venv za stvaranje Python virtualnih okruženja. Modul služi za stvaranje izolirane mape ili direktorija u kojem se mogu pohraniti sve pip biblioteke koji se koriste u nekome projektu. Nakon što je kreirano virtualno okruženje svaka nova instalirana biblioteka bit će pohranjena unutar izolirane mape i nigdje drugdje, a nakon što se završi s korištenjem okruženja može se jednostavno deaktivirati te će tada biti dostupne samo globalne pip biblioteke. Kreiranjem virtualnog okruženja izbjegavaju se sukobi s pogrešno instaliranim verzijama biblioteka i nepotrebnim instalacijama na osobnom računalu te se osigurava da će naš projekt uvijek imati sve potrebne biblioteke s odgovarajućim verzijama.

Za potrebe ovog projekta napravljene su dvije skripte env\_create.bat i env\_create.bat koje služe za stvaranje virtualnog okruženja i instalaciju svih potrebnih biblioteka iz popisa biblioteka pohranjenih u tekstilni dokument *env\_pakages.txt*. Pokretanjem prvo skripte env\_create.bat kreira se virtualno okruženje na lokalnom računalu.

py -3.8 -m venv C:\\Tools\\Python\\IntegrationTestsEnvironment

Pokretanjem druge *skripte env\_create.bat* ažuriramo python instaler i pokrećemo instalaciju biblioteka s popisa paket unutar *env\_pakages.txt*.

call C:\Tools\Python\IntegrationTestsEnvironment\Scripts\activate.bat

python -m pip install --upgrade pip

python -m pip install --upgrade pip-tools

python -m pip install --upgrade setuptools

pip install -r ./requirements.txt

## 5.3 Biblioteke unutar virtulanog okruženja

U nastavku će se malo detaljnije pojasnit pojedina biblioteka koja se nalazi na popisu biblioteka pohranjenih unutar requirements.txt. Za svaki dodanu biblioteku potrebno je staviti i verziju biblioteke kako bi pip uspješno instalirao biblioteku unutar virtualnog okruženja. U nastavku će biti nabrojani i pojašnjeni neki od najvažnijih biblioteka koji se koriste za izradu projekta i zašto se koriste.

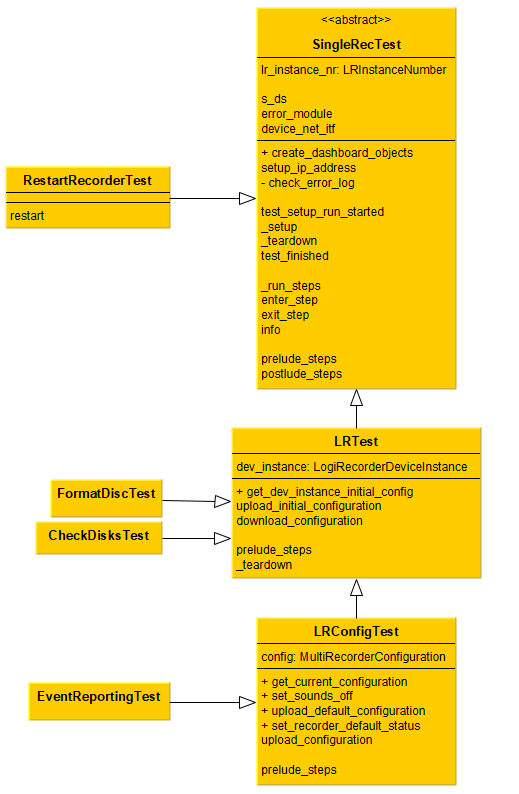
* **PyTest** se smatra radnim okvirom zbog kompleksnosti i kontrole nad konstrukcijom programa. Kada se koristi PyTest potrebno se pridržavati određenih pravila i strukture koju radni okvir nameće. Projekt je konstruiran da se pridržava tih pravila, a detaljnije o Py Testu već je opisano je u poglavlju 3.
* **Hypothesis** se također smatra radnim okvirom iz prijašnje navedenih razloga te je njega moguće ukomponirati s PyTest radnim okvirom. Takva struktura gdje se zajedno koriste PyTest i Hypothesis radni okvir demonstrirat će kasnije kroz rad. Detaljnije o hypothesis random okviru opisano je u poglavlju 4.
* **Pyseria**l bibloiteka omogučuje pristup serijskom priključku računala. Pruža backend podrušku Windowsima i ostalim sustavima kontrolu nad postavkama serijskih portova te omogučava slanje i čitanje prometa koji prolaze kroz seriske portove. Kako je logiRecorder uređaj povezan s računalom preko serijskog porta za slanje pojednih komandi i čitanje povratnih informacija koje nam uređaj vraća Pyserial paket biti će od velike koristi za interakciju i automatizaciju u testovima. [15]
* **PythonNet** biblioteka omogućuje pristup serijskom priključku računala. Pruža backend podršku Windowsima i ostalim sustavima kontrolu nad postavkama serijskih portova te omogućava slanje i čitanje prometa koji prolaze kroz serijske portove. Kako je logiRecorder uređaj povezan s računalom preko serijskog porta za slanje pobjednih komandi i čitanje povratnih informacija koje nam uređaj vraća Pyserial paket bit će od velike koristi za interakciju i automatizaciju u testovima. [16]
* **Requests** je standardna biblioteka u python za izradu HTTP zahtjeva. Smanjuje složenost kreiranja zahtjeva u obliku jednostavnog API-a koje osim kreiranja zahtjeva nudi opcije konfiguracije i prilagodbe zahtjeva kao i čitanje povratnih informacija koje zaprimljene kroz zahtjev. Osim povezanosti logiRECORDER uređaja i računala preko serijskog porta povezani su i Ethernet kabelom preko kojeg je moguće slati HTTP zahtjeve. Pomoću requests biblioteke automatizira se slanje zahtjeva na logiRECORDER uređaj kada se želi poslati neka određena komanda ili ako nas zanima neko određeno stanje samog uređaja. [17]
* **Pyfunctional** biblioteka olakšava kreiranje podatkovnih protoka pomoću uglačanih funkcionalnih operatora. Kako testovi koriste mnogo podataka koji su pohranjeni jedan unutar drugoga potrebno je mnogo vremena i korištenje kvadratnih jednadžbi kako bi se prošlo kroz sve kolekcije podataka.Kako bi se ubrzala i olakšala transformacija nad podacima koristi se Pyfunctional biblioteka koja nudi niz gotovih operacija koje na učinkovit i efikasan način izvršavaju transformaciju. Jedna od osnovnih funkcija unutar Pyfunctional biblioteke je filter funkcija koja filtrira ulazne podatke na temelju zadanog uvjeta . [18]

# Praktični prikaz testiranja unutar PyTest okvira

Arhitektura projekta bazirana je na objektno orijentiranom programiranju uz pridržavanje određenih pravila PyTest radnog okvira. Kako se radi veliki broj testova unutar sustava potrebno je izdvojiti zajedničke korake za pojedine grupe testova unutar baznih klasa kako bi se smanjila količina koda i nepotrebno ponavljanje. Tada je refaktoriranje koda olakšano jer mijenjanjem pojedine komponente unutar bazne klase automatski mijenjamo tu komponentu i za sve klase koje ju nasljeđuju. Drugi važan princip unutar projekta je nasljeđivanje unutar klasa. Dashboard aplikacija nudi niz istih mogućnosti (konfiguriranje, snimanje prometa, reproduciranje snimki itd.) za više različitih sučelja (CAN, Video, Ethernet, UART, LIN itd.). Iz toga se može zaključiti kako će testovi za pojedina sučelja imati iste korake unutar testa, ali će implementacija tih koraka za svako sučelje biti nešto drugačija. Recimo da se želie kreirati testovi koji će testirati snimljen promet za LIN i testove koji će testirati snimljen promet za CAN tada će se napraviti bazni test koji će sadržavati zajedničke korake kao što su postavljenje 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.

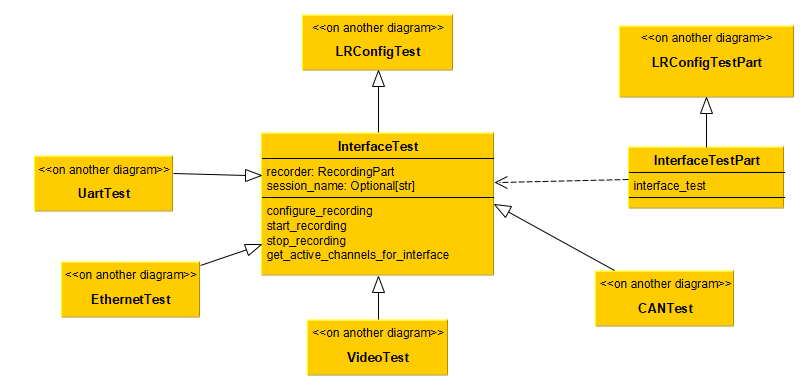
## 6.1 Temelji projekta

Svi testovi unutar projekta imaju zajedničku klasu *SingleRecTest* u kojoj su odrađeni svi potrebni koraci za svaki test unutar projekta. 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 u informacija unutar baze 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 određene komponente kao što je resetiranje logiRecorder. Iz *SingleRecTest* klase izvedena je *LRTest* klasa koja nadodaje korake dohvaćanje osnovnog konfiguracijskog dokumenta preko kojeg se stvara inicijalna konfiguracija za logiRECORDER. Iz LRTesta kreirani su testovi kao što su nadogradnja firmwarea, provjera logiRECORDER diskova itd. Također definirana je i *teardown()* metoda koja se uvijek izvršava nakon završetka testa neovisno o njegovom ishodu. Ona vrlo važna kao bi se osiguralo izvršavanje pobjednih operacija. Na primjer što ako je pokrenuto snimanje i test u sljedećem koraku kod slanja prometa ne uspije izvršavanje? Tada će PyTest prekinut izvršavanje testa i svi sljedeći koraci neće biti izvršeni što uključuje i prekidanje snimanja. Metodom *teardown()* osiguravamo da će se snimanje uvijek prekinuti nakon što se test izvrši. Sljedeća bazna klasa koje naslijeđena iz LRTest klase je *LRConfigTest* koja implementira korake vezane uz slanje konfiguracije na logiRECORDER kao i postavljanje logiRECORDERa u očekivano stanje. Dijagram navedenih klasa prikazan je na slici 20.



*Slika 20. Dijagram klasa*

Klasa koja se nasljeđuje je *IneterfaceTest* koja predstavlja baznu klasu za sva sučelja koje se testiraju testovima. Klasa sadrži korake zadužene za konfiguraciju snimanja i dohvačanje aktivnih kanala sa logiRecorder uređaja. Diagram Interface klase i njezinih podklasa prikazano je na slici 21.



*Slika 21. Dijagram klasa*

Kako je rečeno *teardown()* metodom osiguramo izvršavanje pojedinih zadataka nakon što se testovi završe, no isto tako je bitno da se određeni uvjeti ispune i prije izvršavanja samih testova. Za to se koristi PyTest značajka kojom je moguće izvršiti učvršćene funkcije prije ulaska u testnu metodu. Kako se isti testovi izvršavaju više puta za drugačije parametre dobro bi bilo numerirati izvršavanje testa tj. dodijeliti im invokacije kako bi znali koliko se taj test puta izvršio, na kojoj invokaciji je ona pao i kako bi dobili mogućnost manipulacije nad testovima. Zato je potrebno prije ulaska u testnu metodu učvršćenom metodom proslijediti potrebne informacije o tome koje invokacije želimo pokrenuti. Druga metoda učvršćivanja koju je potrebno izvršiti prije je kreiranje privremenog foldera u kojeg će se spremati snimke i drugi podaci koji se koriste i unutar testa.Zajedničke učvršćene metode moguće je staviti na jednom mjestu unutar conftest.py datoteke, PyTest će tada proslijediti parametre iz metoda unutar svakog testne metode koja ju pozive nebitno o tome u kojem se modulu testna metoda nalazi.

## 6.2 CAN protokol

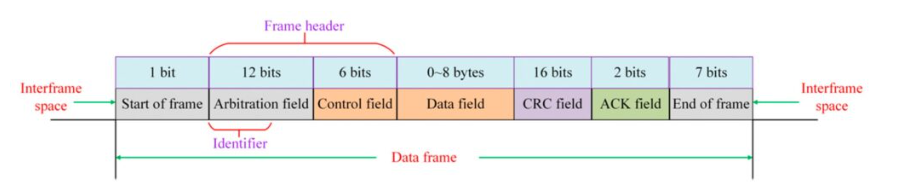
CAN skraćeno od Control Erea Network je najkorištenija sabirnica za umrežavanje velikog broja električnih upravljačkih jedinca unutar automobila. Za umrežavanje se koristi UTP kabel dizajniran da pruža otpornost u okruženjima s elektromagnetskim šumom. CAN se značajno razvijao od 1986. godine te su od tada razvijene sljedeće implementacije CAN sabirnice:

* CAN male brzine (LS-CAN)
* CAN velike brzine (HS-CAN)
* Fleksibilna brzina prijenosa podataka CAN (CAN FD)
* Treća generacija, poznata kao extra-long CAN (CAN XL)

CAN je protokol temeljen na okvirima koji se koristi na multipleksnim ožičenjima u automobilima kako bi se smanjila težina kabelskog opterećenja i podržale visoke brzine prijenosa podataka. Svaka poruka ili okvir podataka prenosi se sekvencijalno. Ako više od jednog čvora odašilje istovremeno, prijenosi podataka strukturirani su tako da osiguraju da čvor s najvećim prioritetom dobije pristup sabirnici. Ostali čvorovi zaustavljaju sve prijenose. Na primjer tijekom vožnje stalno se dobiva informaciju o tome koliko se krećemo na sat, no u trenutku naletavanje na vozilo ispred nas velikom brzinom sustav bi trebao zaustaviti sve sporedne signale i prvo poslati signal naglog zakočenja vozila kako bi reakcija bi što je moguće brža. [18] [19]

### 6.2.1 Struktura CAN okvira

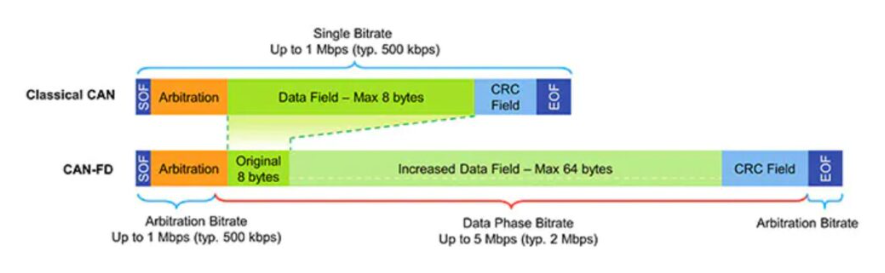
CAN podatkovni okvir sastoji se od sedam različitih bitnih polja: početak okvira, arbitražno polje, kontrolno polje, podatkovno polje, polje cikličke provjere redundantnosti (CRC), polje znaka potvrde (ACK) i kraj okvira.



*Slika 22. CAN okvir [19]*

* Početak okvira sastoji se od jednog dominantnog bita.
* Arbitražno polje uključuje identifikator i bitove RTR-a (engl, Radio Teletype Receiver).
* Kontrolno polje sastoji se od šest bitova, uključujući dva rezervirana bita i četiri bita duljine podataka s dopuštenim vrijednostima duljine podataka od nula do osam bajtova.
* Podatkovno polje šalje međuspremnik prema kodu duljine, označavajući duljinu polja. Kao i kontrolno polje može biti od nula do osam bajtova, a svaki bajt sadrži osam bitova.
* Polje CRC (engl, Cyclic redundancy check) koda sastoji se od CRC duljine 15 bitova i CRC graničnog znaka (engl, implicitni bit). Polinomi koji se koriste u CRC izračunu uključuju početnu domenu, arbitražnu domenu, kontrolnu domenu, podatkovnu domenu i tok bitova za uklanjanje.
* ACK polje sadrži prazninu odgovora i recesivne bitove definicije odgovora od pošiljatelja. Svi čvorovi koji prime ispravan CRC niz mijenjaju recesivni bit poslan u dominantne bitove u praznini odgovora čvora koji šalje.

CAN FD pruža značajno povećanje brzine u usporedbi s HS-CAN-om. Veća brzina, u kombinaciji s maksimalnim povećanjem veličine korisnog opterećenja od (osam bajtova na 64 bajta), osmišljena je za značajno poboljšanje propusnosti CAN FD mreža. [19]

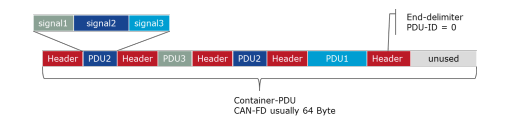


*Sika 23. CAN FD okvir [19]*

### 6.2.2 CAN baze podataka

CAN baza podataka je tekstilna datoteka koja sadrži informacije za dekodiranje neobrađenih podataka CAN sučelja iz heksadecimalnih vrijednosti u fizičke vrijednosti. CAN DBC baze podataka mogu se transformirati u xml ili složeniju ARXML datoteku. ARXML datoteka je konfiguracijska datoteka spremljena u AUTOSAR XML formatu. Koristi ga AUTOSAR koji predstavlja inicijativu proizvođača i dobavljača automobila osnovan 2003. za uspostavu softverske arhitekture za automobilske elektroničke upravljačke jedinice ECU-a. ARXML datoteke sadrže podatke o konfiguraciji i specifikacijama u XML formatu za ECU-ove, koji se koristi za kontrolu komponenti motora kako bi se osiguralo da motor postiže optimalne performanse.

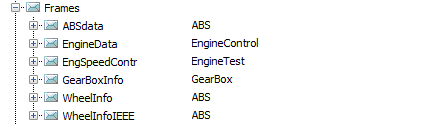
Okviri iz ARXML baze mogu biti samo CANFD okviri s veličinom 64 bajta koji u sebi mogu sadržavat PDU (engl. Protocol Data Unit) kontejner za pohranu dodatnih CAN okvira kako bi se omogućila veća brzina prijenosa. PDU kontejner nalazi se unutar podatkovnog sloja CANFD okvira. [21] [22]



*Slika 24. PDU kontejner [22]*

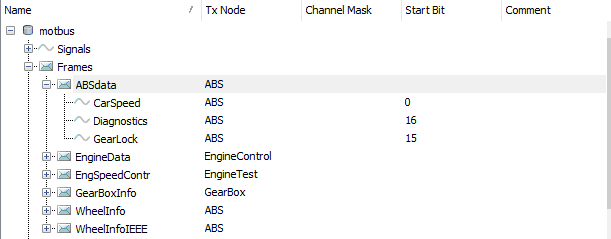
### 6.2.3 CAN Motbus baza podataka

Motbus je manja XML baza koja se sastoji od 6 CAN okvira kreirana od globalne tehnološke tvrtke koja isporučuje sustave za osobna vozila ZF (ger. Zahnradfabrik Friedrichshafen) . Kako se radi o XML formatu bazu je moguće otvoriti u jednom od editora ili koristi profesionalni alat Vector Logger Configurator pomoću kojega se jasno vidi koji se okviri nalaze u bazi prikazanoj na slici 25. [23]



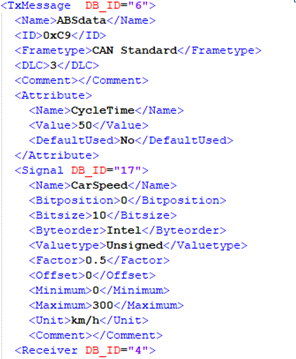
*Slika 25. CAN okviri Motbus DBC-a*

Svaki od prikazanih okvira može u sebi sadržavati više signala koja pripadaju istome okviru.



*Slika 26. Signali ABSdata okvira*

Ako pogledamo okvir ABSdata u unutar XML datoteke možemo naći sve informacije koje sadrži okvir i njegove signale. Prikaz ABSdata okvira i signala CarSpeed prikazano je na slici 27.

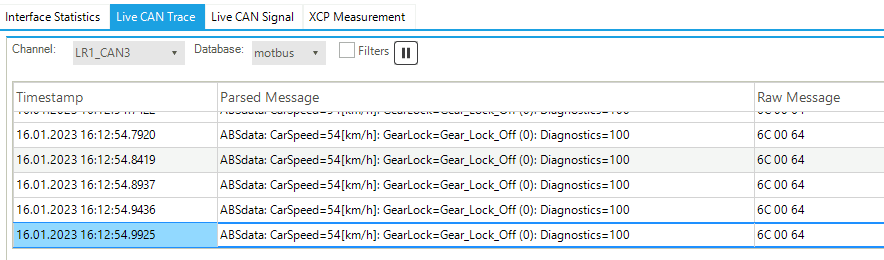


*Slika 27. ABSdata okvir unutar Notepad++ editora*

Iz baze očitava se (ID, identifikacijski ključ) okvira koji iznosi 0xC9 heksadecimalno, tip okvira je običan CAN, veličina okvira je 3 bajta, a ciklus poruke je 50 milisekundi. Startna pozicija za CarSpeed signal je 0, veličina signala je 10 bita, a tip vrijednosti podatka je „Unsigned“ te negativne vrijednosti nisu moguće što čini raspon vrijednosti od 0 do 1023. Vrijednost signala potrebno je pomnožiti s faktorom te ga zbrojiti s pomakom i tada se dobiva konačna vrijednost po formuli. [24]

Signal\_value = decimal\_value \* Factor + Offest

Moguče vrijednosti za tip podatka su *Unsigned*, *Signed* i IEEE *float*. Kod *Signed* vrijednosti mogući su negativni brojevi pa je raspon vrijednosti od do . Tip vrijednosti IEEE *float* omgučuje decimalne vrijednosti i njegova veličina je 32 bita. Unutar njega najznačajniji bit određuje predznak, sljedeća 8 bita koriste se za određivanje eksponenta, a preostlih 23 bita predstavljaju frakciju. Raspon mogućih vrijednosti je od do . Mogući poredak bitova je *litle endian* i *big endian*. Za signal CarSpeed prikazan na slici 28 poredak bitova je tipa Intel što znači da je najznačajniji bit podatka smještan na bit sa najvećom adresom. Ako se pomoću PCAN generatora prometa šalju okviri s ID-em 0xC9 s podatkovnim poljem 0x6c0064 heksadecimalne vrijednosti unutar *Dashboard* prozora *LIVE CAN Trace* moguće je vidjeti fizičke vrijednosti signala za taj okvir.



*Slika 28. Fizičke vrijednosti signala unutar Dashboard aplikacija*

Bitovi koji ulaze u CarSpeed signal su 00 0110 1100 što predstvlja vrijdnost 108 u decimalnom zapisu. Taj broj pomnožen s faktrom 0.5 i zbrjenim pomakom 0 za rezultat dobija se vrijednost 54 km/h.

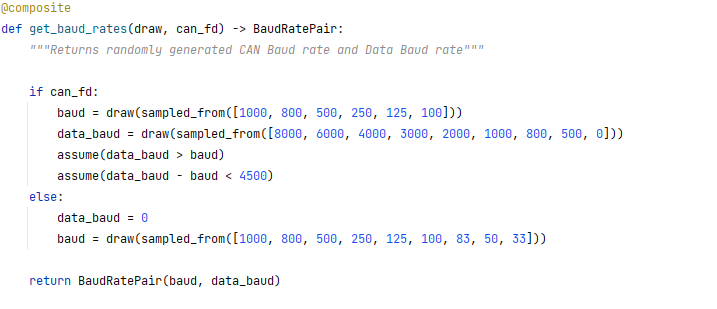
## 6.3 Testiranje parsiranja CAN baza unutar PyTesta

U nastavku je prikazan test koji pomoću PyTest i Hypothesis razvojnih okvira generira ulazna parametre za testiranje ispravnosti rada Dashboard aplikacije u radu s CAN bazama podataka. To uključuje testiranje snimljenog CAN prometa na svim brzinama u Ros-bag i Mdf4 formatu, testiranje CAN filtera i CAN okidača te provjere parsiranja CAN prometa za dobranu bazu podataka. Za kreiranje testa najprije je potrebno uvesti potrebne biblioteke kao što su PyTest, Hypothesis i Functional kao i sve kreirane klase koje se koriste unutar testa. Unutar testa dodane su dvije metode učvršćenja za kreiranje privremenog modula i predaju objekta koji kontrolira izvođenje invokacije testa. logiRECORDER omogućuje snimanje na 16 kanala koji se mogu raspodijelit na sljedeća sučelja CAN, Uart i LIN. Koji će se kanali koristi ovisi o korisniku na koje je kanale je spojio pripadajući generator prometa. Unutar ovog testnog scenarija koristit će se generator CAN prometa spojen na kanale 3 i 4.

Kako bi se izgenerirali potrebni parametri za test kreirati će se kompozitna funkcija get\_can\_database\_parameters() koja vraća strategiju parametara odnosno instancu klase *DatabaseHypotheisParameters*. Kreirana strategija se pomoću dekoratora @given predaje testnoj funkciji kao ulazni parametar. Kompozitna funkcija *get\_can\_database\_parametars()* poziva druge kompozitne funkcije koje su potrebne za dobivanje svih parametara. Najprije se generira varijabla *can\_fd* koja odlučuje hoće li generirati test koji će testirati CAN ili CANFD bazu podatka. Zatim se poziva *get\_baud\_rates()* funkcija koja vraća kombinaciju klasične brzine prijenosa i brzine prijenosa podatka.

Ako se radi o CAN sučelju tada je moguća samo klasična brzina prijenosa dok je za CANFD moguće dobiti bilo koju kombinaciju sve dok ona zadovoljava funkciju assume. Kompozitna strategija koja vraća kombinaciju brzina prikazana je na slici 29.

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



*Slika 29. Kompozitna funkcija za vračanje brzine prijenosa*

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 dohvaća se jedna od CAN ili CANFD XML baza podataka koja se nalaze u folderu *CAN-databse*. Pomoću *XmlExtractor* klase koristeći ElementTree XML biblioteku dohvaćaju se svi CAN okviri unutar baze, a zatim se uzima uzorak nasumično dohvaćenih okvira. Kreiranje prvih 6 parametara testa unutar kompozitne funkcije *get\_can\_database\_test\_parameters()* prikazano je na slici 30.

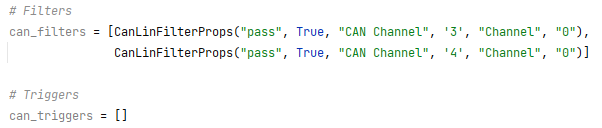
Copyright Xylon d.o.o 2023 All Rights Reserved, author Igor Stanković



*Slika 30. Kopozitna funkcija za kreiranje parametara testa*

Dashboard aplikacija omogućuje filtriranje CAN prometa po kanalu, ID-u ili rasponu ID-eva, PDU ID-u kontejnera ili njegovom rasponu. Filtere je moguće kombinirati tako je moguće postaviti filter koji će propustiti sav promet na jednom kanalu dok na drugom propustiti okvire s određenim ID-om. Kao i filtere okidače je isto moguće međusobno kombinirati. Unutar strategije za generiranje parametara testa moguće izgenerirane kombinacije su okidači koji se okidaju na neki ID, PDU ID ili neku vrijednost signala. Ponekad nije dobro krenuti s testiranjem svih mogućih komponenti na samo početku zbog otežanog i kompleksnijeg pronalaženja potencijalnih grešaka. Kako bi se najprije testiralo samo parsiranje CAN okvira iz baze podataka bez dodatnih filtriranja i okidača unutar strategija lista filtera postavit će se tako da kanal 3 i 4 propuštaju sve okvire, a lista okidača postavit će se na praznu listu kako je prikazano na slici 31.

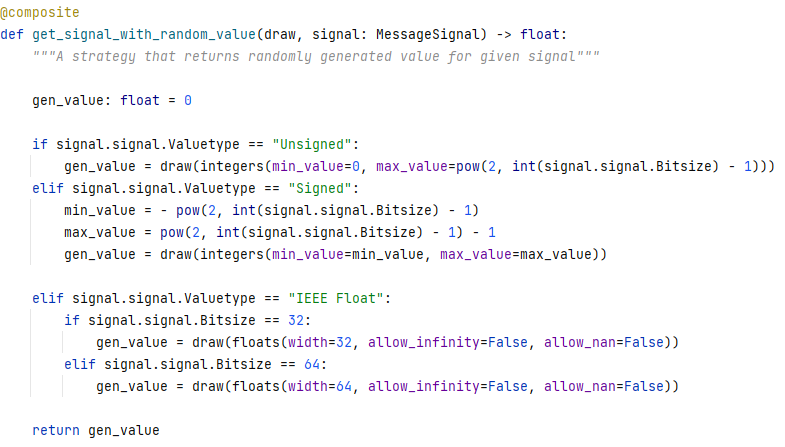
Copyright Xylon d.o.o 2023 All Rights Reserved, author Igor Stanković



*31. Liste filtera i okidača za prvo testiranje*

Kada se provede testiranje parsiranja okvira bez dodatnog filtera i okidača te se dokaže da ta komponenta radi može se krenuti s dodavanjem dodatnih komponenta unutar testa. Podatkovno poslije unutar CAN okvira sadrži informacije o signalima i njihovim vrijednostima u heksadecimalnom obliku Šaljući pobjedni okvir sa specifičnim ID-em specificiran unutar baze podakta Dashboard aplikacija trebala bi dekodirati o kojoj se CAN poruci radi i koje su njezine vrijednosti za signale unutar podatkovnog polja. Za nasumično generiranje mogućih vrijednosti signala pojedinog CAN okvira koristi se kompozitna funkcija get\_signal\_with\_random\_values() prikazana na slici 32.

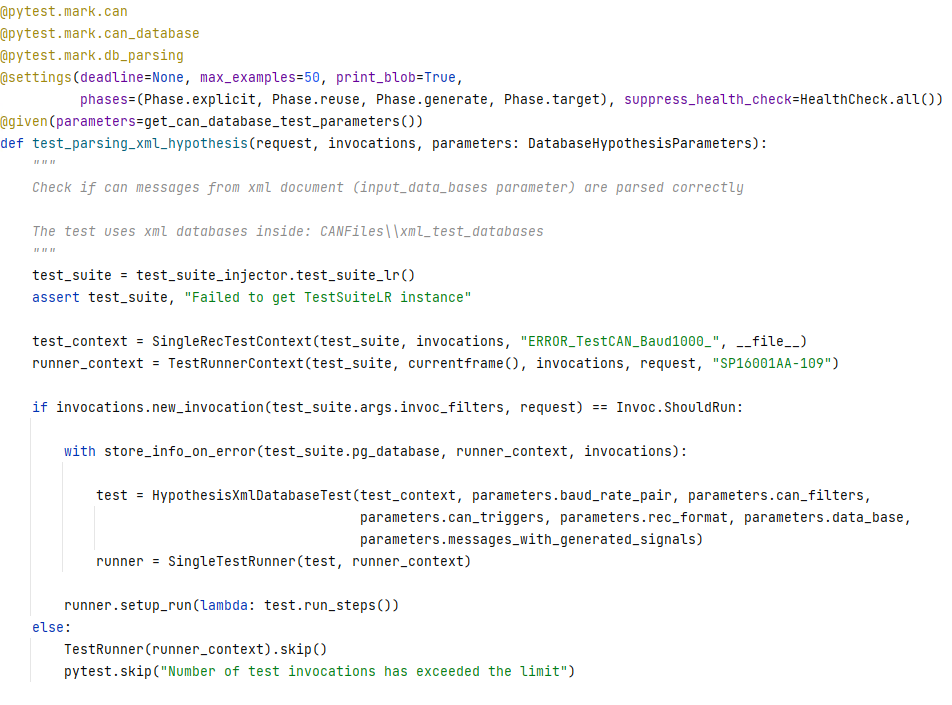
Copyright Xylon d.o.o 2023 All Rights Reserved, author Igor Stanković



*Slika 32. Dohvačanje nasumične vrijednosti signala*

Kako je rečeno u prijašnjem poglavlju signal može biti tipa *Unsigned* cijeli broj, *Signed* cijeli boroj i IEEE decimalni broj te za svaki od tipova napisan je uvjet koji pomoću strategije generira moguću vrijednost za taj signal. Kada su izgenerirani svi parametri potrebni za testnu metodu potrebno ih je proslijediti testnoj metodi pomoću dekoratora *@given*. Iznad testne metoda specificirane su neke od postavki Hypothesis razvojnog okvira kao i markeri za testnu metodu. Testna metoda *test\_parsing\_xml\_hypothesis()* prikazana je na slici 33.

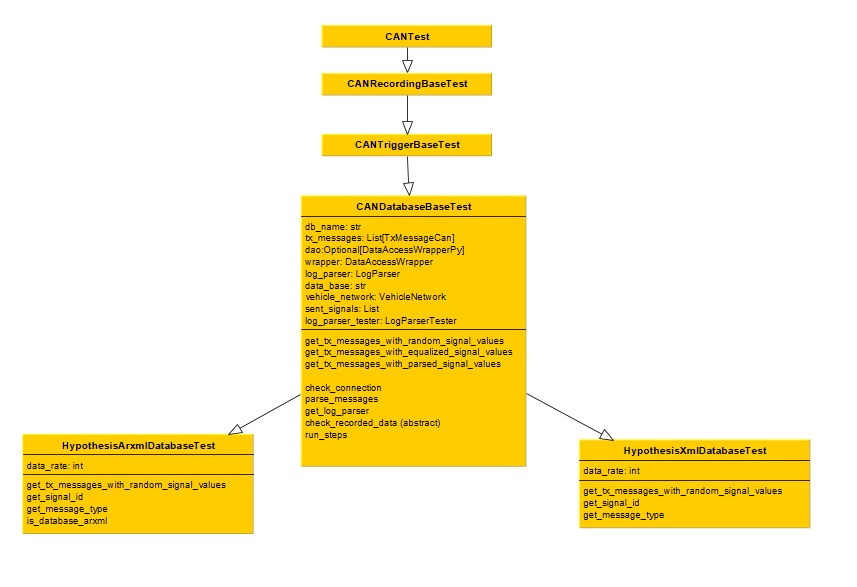
Copyright Xylon d.o.o 2023 All Rights Reserved, author Igor Stanković



*Slika 33. Testna metoda*

Iznad testne metode postavljeni su PyTest markeri can i db\_parsing. Pomoću Hypothesis dekoratora postavljene su postavke koje uključuje broj izvođenja od 50 uzoraka, isključen je rok za generiranje pojedinačnog primjera, omogućeno je reproduciranje neuspješnih testova, odabrane su željene faze koje se žele izvršiti na primjerima te su isključena upozorenja za predugo trajanje generiranja strategija zbog kompleksnosti strategije.

Ulaskom u testnu metodu kreira se *TestSuiteLR* instanca koji dohvaća argumente postavljene unutar *lunch.json* datoteke. Zatim se kreiraju *SingleRecTestContext* i *TestRunnerContext* istance koje pohranju potrebne argumente za svaki test. Nakon kreiranja navedenih instanci provjerava se uvjet treba li se izvršiti invokacija odnosno treba li *SingleTestRunner* objekt pokrenuti korake koje je prikupio. Ako uvjet nije ispunjen invokacija se preskače i kreće se na novu invokaciju s novim generiranim parametrima. Parametre koje je generirala kompozitna funkcija predaju se testu HypothesisXmlDatabaseTest koji definira svoje korake ispitivanja i pridružuje ih svim koracima koje je naslijedio. Kada je test kreiran predaje se SingleTestRunner objektu koji pokreće sve korake ispitivanja pozivom metode run\_steps(). Klasa HypothesisXmlDatabaseTest nastavlja arhitekturu prikazanu na slici 34.

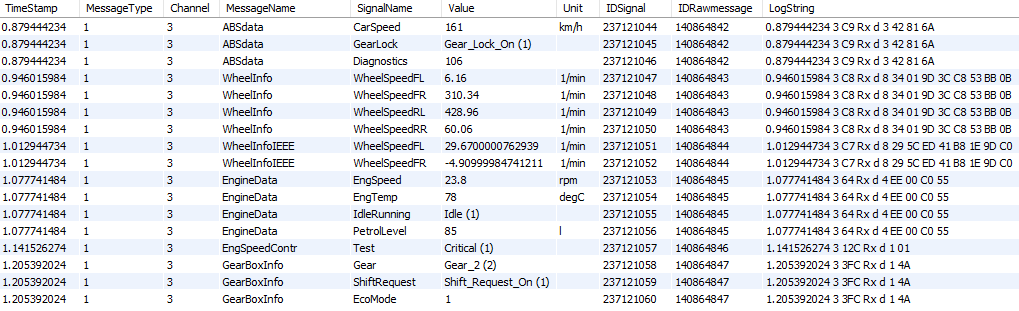


*Slika 34. Dijagram klasa*

#### 6.3.1 Ispitni koraci

Ispitni koraci su sve metode koje poziva funkcija run\_steps(). Takve metode sadrže dekorator @step koji omogućuje ispis imena metode unutar konzole debagiranja te njezin ishod. Ako su sve pretpostavke unutar ispitnog koraka zadovoljene izlazi se iz metode s ispisom da je taj korak uspješno prošao, no ako samo jedna pretpostavka bude pogrešna ispisuje da je korak neuspješan i prekida se izvođene testa. Kako bi se testna metoda test\_parsing\_xml\_hypothesis() smatrala uspješnom za svi 50 invokacija nasumično izgeneriranih parametara moraju biti uspješno izvedeni sljedeći koraci:

1. **create\_dashboard\_objects**: Kreiranje *Dashboard* instanci koje se koriste u unutar testa.
2. **get\_dev\_instance\_initial\_config**: Deserijalizacija inicijalne konfiguracije iz INIT\_LCF datoteke koja se pridružuje *LogiRecorderConfiguration* objektu
3. **convert\_hardware\_configuration**: kreiranje *DeviceHardwareConfiguration* instance
4. **get\_current\_configuration**: Dohvačanje trenutne konfiguracija na logiRecorderu
5. **upload\_default\_configuration**: postavljanje konfiguracija koja predstavlja početnu konfiguraciju za svaki test.
6. **set\_recorder\_default\_status**: Provjera stanja logiRecordera, ako je logiRecorder ostao u stanju snimanja ili reproduciranja postavlja se na početno stanje
7. **set\_sounds\_off:** Isključivanje zvuka bazera prilikom pokretanja i zaustavaljanja snimanja
8. **get\_active\_channels**: dohvaćanje aktivnih CAN kanala
9. **activate\_interfaces**: Konfiguracija sučelja aktivnih kanala i njezino aktiviranje
10. **setup\_filters**: Postavljanje filtera koji propuštaju promet sučelja
11. **configure\_recording**: Postavljanje željenih postavki snimanja. Odabire se standardno snimanje, snimanje na okidač koje može biti sa jednostruko ili višestruko.
12. **configure\_general**: Konfiguracija generalinih postavki kao što su format snimanja, dodjeljivanje imena snimke, ograničavanje veličine snimki itd.
13. **setup\_triggers**: Postavljanje okidača za aktivna sučelja, ako se triggeri ne koriste korak se preskače.
14. **upload\_configuration**: Nakon što su svi parametri konfiguracije postavljeni konfiguracija se šalje na logiRecorder *firmware* .
15. **start\_recording**: Pokreće se snimanje
16. **send\_can\_messages**: Slanje svih CAN okvira koji se nalaze u XML bazi podataka odvija se pomoću PCAN generatora prometa korištenjem PCAN API-a. Okviri se šalju u obliku sirovih heksadecimalnih podataka koji su složeni tako da sadržavaju vrijednosti signala generiranih *get\_signal\_with\_random\_values()* kompozitnom funkcijom.
17. **stop\_recording***:* Prekida se snimanje.
18. **check\_connection**: Dashboard aplikacija sprema logove sesije unutar MySql baze podataka. Kako bi se dohvatili podaci iz baze potrebno je najprije uspostaviti konekciju za što je zadužen ovaj ispitni korak.
19. **parse\_messages**: Kreira se *Dashboard* objekt *ControlThread* preko kojeg se poziva metoda *StartParsing()* zadužena za parsiranja logova i njihovo spremanje u MySql bazu podataka.
20. **get\_expected\_markers**: Dohvačanje okidača markera ako postoje aktivni okidači. Markeri za okidače nalaziti će se unutar parsiranih logova ako neki od CAN okvira sadrži ID, PDU ID ili vrijednost signala za koju je postavljen okidač unutar konfiguracije.
21. **filter\_can\_messages**: Filtriraju se okviri iz lokalne liste koja se sastoji od poslanih CAN okvira. Ako postoje više filtera okviri se filtriraju za svaki filter.
22. **check\_recorded\_data**: Unutar ovog ispitnog koraka provjerava se točnost parsiranih logova unutar MySQL baze. Unutar SignalView tablice nalaze vrijednosti koje su izračunate .Net objektom LogParser. To podrazumijeva provjeru broja snimljenih okvira, provjeru okidača markera, provjera broja signala i njihovih vrijednost. Ako se uzme scenarij u kojem se pošalju svi CAN okviri iz Motbus XML baze koja sadrži 6 okvira tablica SingleView unutar MySql baze podataka bi trebao izgledati kao na slici 35.



*Slika 35. SingalView tablica*

Unutar Singleview tablice trebaju se nalaziti svih 6 okvira unutar MessageName stupca i svi signali za pripadajuće okvire unutar SignalName stupca. Za stupac Channel ispisuje se kanal za koji je okvir snimljen dok MessageType stupac ispisuje kojeg je tipa poruka.

Broj jedan označuje da se radi o standardnom CAN okviru. Unutar Value stupca prikazuje se parsirana vrijednost signala za poslani CAN okvir i mjerna jedinca unutar Unit stupca ako je sadrži. LogString stupac prikazuje snimljeni log u kojem se može vidjeti okvir u heksadecimalnom obliku.

#### Izvršavanje testa

Test se pokreće naredbom:

PyTest -s -v --project=LR --invoc\_filters="count:any:-1" -m "db\_parsing and can“

-s : onemogućava hvatanje ispisa

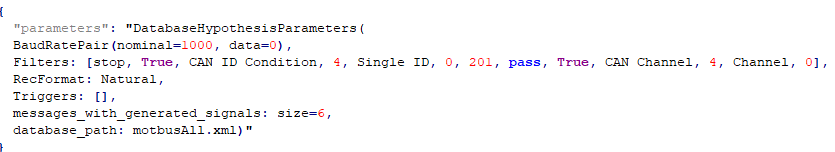
-v: povećava opširnost informacaija ishoda

--project==LR: zadaje se projket koji se testira

-- invoc\_filters="count:any:-1: odabrano je izvršavanje svih invokacija

-m "db\_parsing and can“: postavljeni su markeri koji selektiraju test

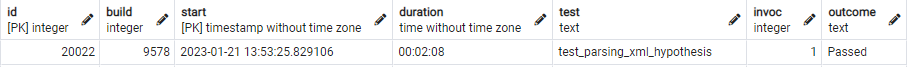
Ulazni parametri testa su generirani su strategijom prikazanoj na slikama 4 i 5, gdje se za inicijalizaciju listi filtera i okidača ne koristi strategija već su one postavljene na fiksnu vrijednost kao bi se test pojednostavio i kako bi fokus testiranja bio na parsiranju CAN okvira. Na slici 36 prikazano je koji su parametri izgenerirani za prvi invokaciju.



*Slika 36. Ulazni parametri testa*

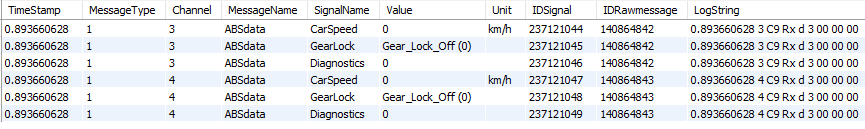
Brzina koja je generirana je klasična brzinu prijenosa 100 kbit/s dok je brzina prijenosa podataka 0 kbit/s kako se radi o standardnom CAN okviru, filteri i okidači su za sada fiksni, broj okvira koji je generiran je jedan okvir iz MotbusAll XML baze.

Prva invokacija je uspješno prošla za izgenerirane parametre, to nam pokazuje tablica invocations unutar PostgreSQL baze. Tablica invocations sadrži sljedeće kolone: broj testna integracije, vrijeme pokretanja invokacije, trajanje invokacije, ime testa, redni broj invokacije, ishod, ispitni korak na kojemu se dogodila greška, poruka uz grešku te ulazni parametri. Prve četiri kolone za prvu invokaciju prikazano je na slici 37.



*Slika 37. Ishod prve invokacije*

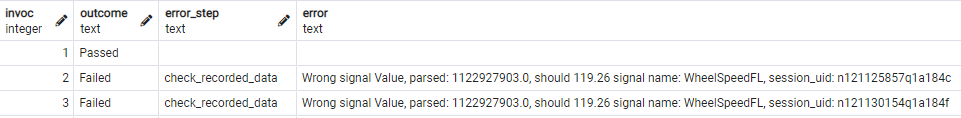
Okvir koji je generiran i njegovi signali mogu se pogledati u MySql bazi unutar tablice SingleView prikazana na slici 38.



*Slika 38. SignalView tablica*

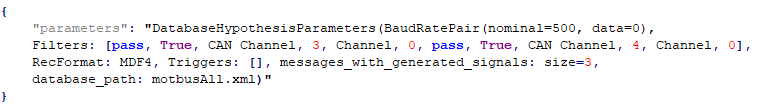
Okvir koji je poslan preko generatora na oba kanala je ABSdata s ID-em okvira 0xC9, a veličina podatkovnog polja je 3 bajta i ono iznosi 0x00 0x00 0x00. Kada se vrijednosti podatkovnog polja parsiraju dobivaju se vrijednosti prikazane pod stupcem Value.

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. Daljnjim izvođenjem testa invokacija broj 2 ne uspijeva zadovoljiti pretpostavku provjere vrijednosti signala unutar check\_recorder\_data() ispitnog koraka. Test radi sigurnosti ponovno pokušava izvrtjeti sljedeću invokaciju s istim parametrima, no njezin ishod je isti kao kod druge invokacije kao što je prikazano s posljednja 4 stupca tablicom Invocations.



*Slika 39. Ishod druge i treće invokacije*

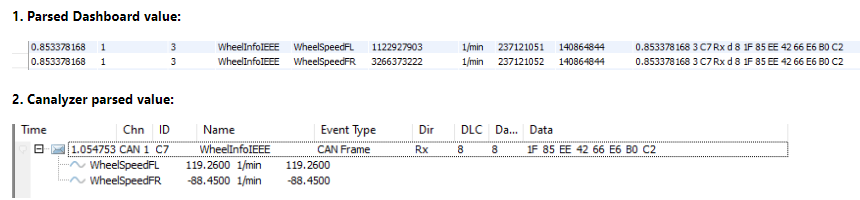
Parametri koji su generirani za dugu invokaciju prikazni su na slici 40.



*Slika 40. Parametri druge invokacije*

Testni kod za signal WheelSpeedFL izračunao je vrijednost signala 119.26 1/min dok je *LogParser* unutar *Dashboard* aplikacije izračunao vrijednost signala 1122937903.0 1/min. Kako bi se utvrdila točna vrijednost koristi se Vectorov alat CANalyzer pro pomoću kojeg je moguče utvrditi točnu parsiranu vrijednost za pojedini signal.

Na slici 41 prikazne su vrijednsti signala WheeLSpeedFL unutar tablice *SingleView* i vrijednosti unutar CANalyzer alata.



*Slika 41. Usporedba vrijednosti WheelSpeedFL signala*

#### Pronađene greške pomoću testova parsiranja

U nastavku će biti prikazna neke od pronađenih grešaka koje su pronađene izvođenjem testova test\_xml\_hypotheisis i test\_arxml\_hypotheisis prije dodavanja strategija za generiranje filtera i okidača unutar strategije za generiranje parametara spomenutih testova. Test\_arxml\_hypotheis koristi isti strukturu i odrađuje iste ispitne korake kao i test\_xml\_hypothesis, samo što se koriste ARXML baze podataka koje su kompliciranije pa je sam obrada okvira zahtjevnija.

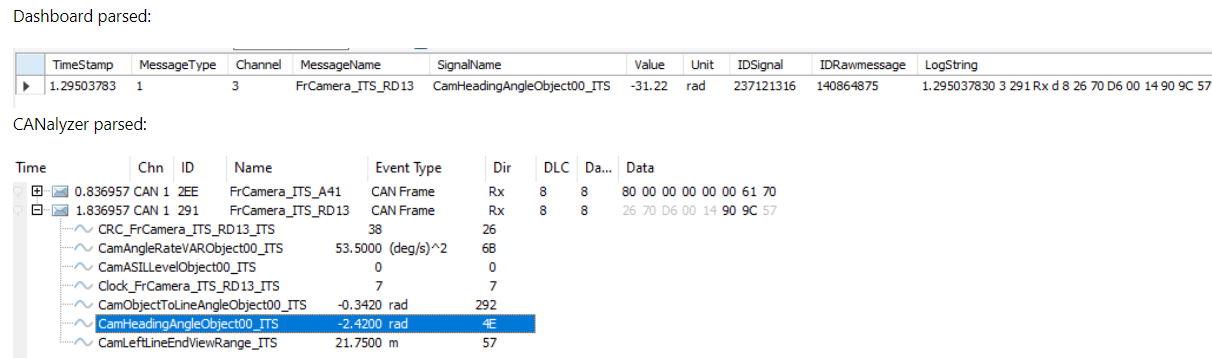
Kako je naglašeno u ranijem poglavlju okviri iz ARXML baze mogu sadržavati PDU kontejner koji sadrži dodatne okvire pa je na mjestima gdje je potrebna obrada okvira potrebno proći kroz dodatnu petlju koja obrađuje sve okvire unutar PDU kontejnera pojedinog okvira. Što se tiče najvažnijih pronađenih grešaka one su sljedeće:

* U jednoj od verzija aplikacije Dashboard parsiranje logova za Ros-bag format nikada ne završi s izvođenjem. Ispitni korak parse\_message() sadrži brojač koji ako vrijeme trajanja parsiranja prijeđe jednu minutu ispitni korak smatra se neuspjelim. Kod detaljnijeg debagiranja na .Net strani otkriveno je kako ulazak u „if“ granu koja prekida dretvu za parsiranje logova za Ros-bag format nije ostvaren.
* Izračunate pogrešne vrijednosti signala koji sadrže redoslijed bitova Motorola. MotbusAll baza ne sadrži CAN FD okvire no druge baze sadrže te pobjedni okviri sadrže signale kojima je poredak bitova od najznačajnijeg bita do najmanje značajnog. Kod izgeneriranih parametara pomoću Hypothesis strategija s takvim signalima otkrivena je greška kod koje su izračunate pogrešne vrijednosti za takve signale. Na slici 42 je prikazan ispis s konzole na kojoj se može vidjeti kako pretpostavka koja uspoređuje vrijednosti signala izračune od strane LogParser objekta i vrijednosti izračunate unutar testa nije uspjela.



*Slika 42. Provjera parsirane vrijednosti u konzoli*

Na slici 43 je prikazna je parsirana vrijednost unutar SingleView tablice i vrijednosti koju je prikazao CANalayzer alat. [24]



*Slika 43. Usporedba vrijednosti CamHeadingAngleObject00\_ITS signala*

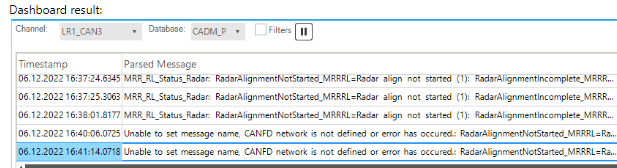
* Nemogućnost parsiranja imena okvira za 64 bitne vrijednosti signala: Kod generiranja vrijednosti za 64 bitne signale strategija generira velike vrijednosti koje su unutar Dashboard aplikacije izazvale iznimku kod produživanja tako velikih vrijednosti unutar varijabli koje su deklarirane kao decimalne vrijednosti umjesto *double*. Za takve slučajeve iznimka bi se uhvatila s ispisanom porukom „Unable to set message Name.“ koja bi se postavila kao vrijednost za ime okvira te bi pretpostavka unutar check\_recorder\_data() ispitnog koraka koja provjerava ime okvira završila neuspješnom.

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



*Slika 44. 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.

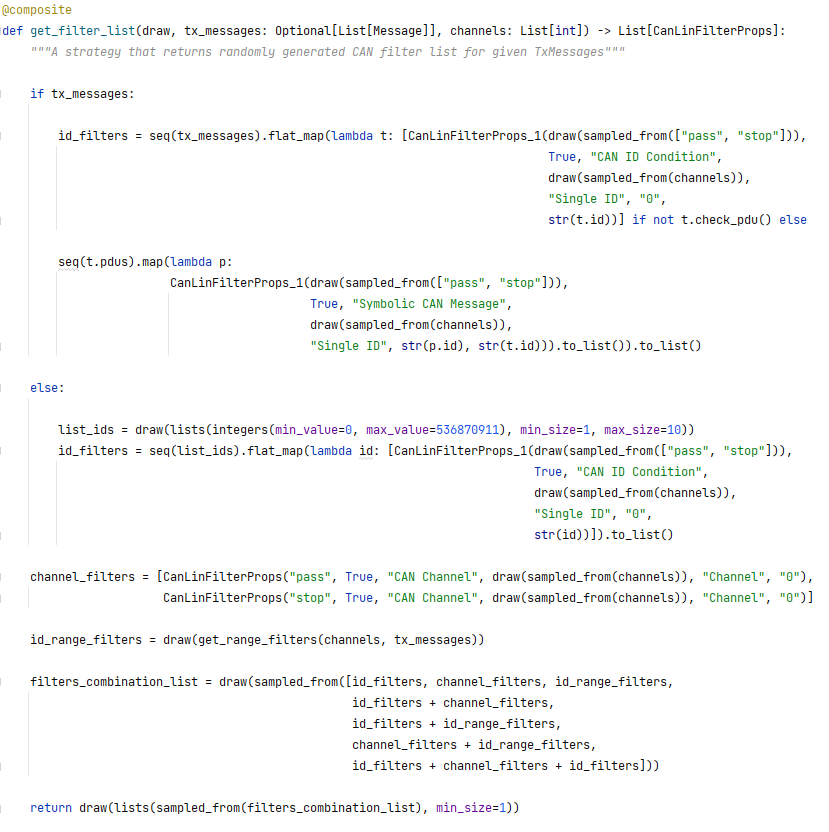


*Slika 45. Prikaz greške unutar Dashboard aplikacije*

#### Unaprjeđenje postojećih testova

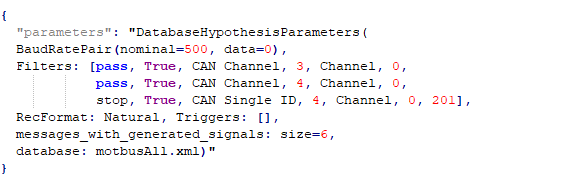
Kada se isprave sve greške koje su otkrivene i sve pretpostavke unutar testa su zadovoljene za sve izgenerirane parametre svih 50 invokacija poželjna je nadogradnja testova ako je ona moguća. Glavni cilj testova je pronalaženje grešaka, no ako testovi uvijek prolaze ne znači da greške ne postoje nego da možda samo one nisu pronađene. Za nadogradnju postojećih testova test\_xml\_hxpotheisis i test\_arxml\_hypotheis potrebno je dodati kompozitnefunkcije za generiranje nasumično generiranih filtere i nasumično generiranih okidača. Prvo će se dodati funkcija za generiranje liste filtera prikazan na slici 46.

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



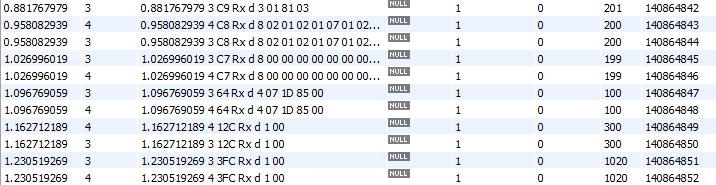
*Slika 46. Kopozitna funkcija za generiranje filtera*

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 u rasponu od 0 do maksimalne moguće vrijednosti 536870911. Filteri 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 što je dodana strategija za generirane filtera u postojeće testove ponovno se pokreću za svih 50 invokacija. Prva invokacija za test\_xml\_hypothesis() generirala je sljedeće parametre prikazane na slici 47.



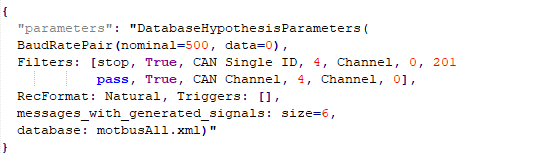
*Slika 47. Parametri prve invokacije*

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 što predstavlja Ros-bag format, a okidači su postavljeni na praznu listu. Generirani su svih 6 okvira iz MotbusAll 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 6 okvira dok kanal 4 treba propusti 5 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 prikazna na slici 48.



*Slika 48. Message tablica*

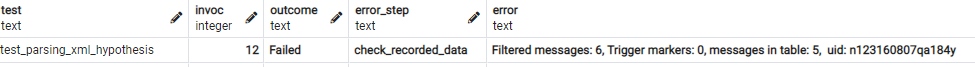
Broj snimljenih okvira je 11, a okvir koji nedostaje je okvir s ID-em 201 na četvrtom kanalu. To se može vidjeti ako se gleda drugi stupac koji prikazuje na kojem kanalu je snimljen okvir i 7. stupac u kojem je prikazan ID okvira. Ne postoji red koji sadržava vrijednost kanala 4 i ID okvira koji ima vrijednost 201. Nakon danjeg izvođenja invokacija 12 sa parametrima prikazanim na slici 49 nezadovoljava pretpostavku očekivanog broja filtriranih CAN okvira.



*Slika 49. Parametri invokacije 12*

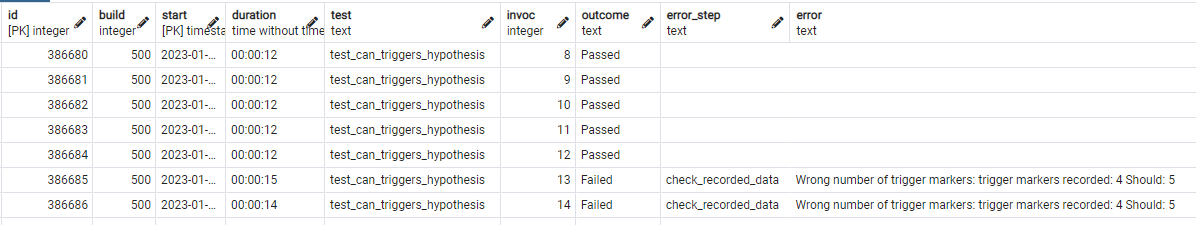
Kako je prvi element iz liste filtera filter koji stopira okvire s ID-em 201 na kanalu 4, a drugi i posljednji filter unutar liste propušta sve okvire na kanalu 4 očekivani broj okvira za sve okvire poslane iz XML baze trebao bi biti 6. Filter koji filtrira okvire po kanalu ima prednost po važnosti filtriranja 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 50.



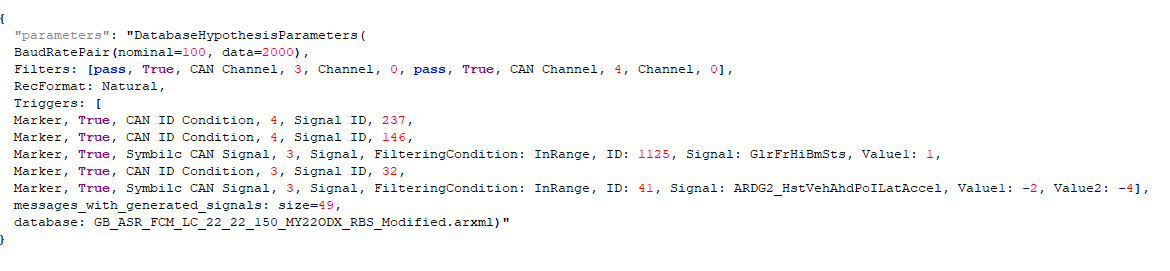
*Slika 50. Ishod invokacije 12*

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 otkrivenena slici 51 i prolaska svih preostalih invokacija testa dodaje se funkcija za generiranje liste okidača. Funkcija get\_trigger\_list() bazira se na istom principu kao i funkcija *get\_filter\_list()* uz dodatno generiranje nasumičnih vrijednosti signala za koje se 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 onaj okviri čije podatkovno polje kada se parsira sadrži vrijednost 100 km/h okinutiće se marker okidač. Pokretanjem test\_can\_triggers\_hypothesis testa sa dodanim funkcijama *get\_filter\_list()* i *get\_trigger\_list()* dobivaju se sljedeći rezultati prikazani u PostgreSQL bazi unutar tablice invocations.



*Slika 51. Ishod invokcija*

Na slici vidljivo je kako su prvih 12 invokacije bile uspješne dok trinaesta invokacija nije zadovoljila pretpostavku koja provjera broj markera okidača. Ulazni parametri invokacije 12 prikazano je na slici 52.



*Slika 52. Parametri invokacije* *12*

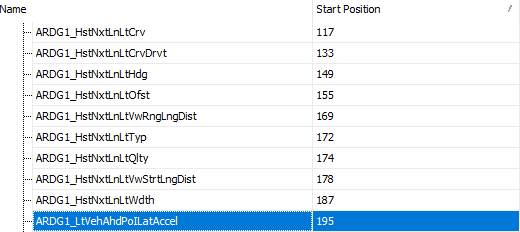
Unutar liste okidača nalazi se 5 okidača koji su postavljeni tako da se okidaju na jedan od CAN okvira iz ARXML baze. Poslani se svi okviri iz GR\_ASR\_FCM ARXML baze s takvim podatkovnim poljem da zadovoljavaju uvjet okidanja za postavljene okidače. Okidači koji imaju CAN ID Condition okidaju se na ID okvira dok okidači koji imaju “Symbolic CAN signal” kao uvjet okidanja okidaju se na vrijednost signala unutar podatkovnog polja. Kod detaljnijeg ispitivanja uviđa se kako okidač za signal ARDG2\_HstVehAhdPoILatAccel koji je trebao okinuti marker za vrijednosti signla između -i -za poslani CAN okvir čije podatkovno polje sadrži vrijednost signala ARDG2\_HstVehAhdPoILatAccel između -. Pasirana 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 mogule. Tako u ovom slučaju ako su dobro istestirani koraci parsiranja, a novi koraci koji provjeravaju okidače prouzrokuju greške inicijativna sumnja bi trebala ići prema ispitivanju komponente koje je dodana naknadno prouzrokovači neuspješan ishod testa koji je prije bio uspješan za svih 50 invokacija.

Zato se daljnje ispitivanje 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 prikazan je na slici 53.



*Slika 53. Konfiguracija okidača*

Početni bit za signal ARDG1\_LtVehAhdPoILatAccel postavljen je na vrijednost 203 dok je njegova prava vrijednost 195 kao što je prikazano na slici 54.



*Slika 54. Prikaz točne vrijednosti startnog bita*

Vrijednost je uzeta iz Autosar Vector Explorer View alata koji se koristi za prikaz CAN okvira, PDU kontejnera i signala iz ARXMLbaze podataka.

# Zaključak

Kada se uz valjanu strategiju i uspješan plan integracijskog testiranja ukomponiraju kvalitetni razvojni okviri kao što su PyTest i Hypothesis koji služe za izradu automatskog integracijskog testiranja sustava dobije se kvalitetan proizvod koji ima sposobnost detektiranja raznovrsnih grešaka programskog koda. Struktura integracijskog testiranja važna je zbog testiranja više programskih komponenti koje međusobno komuniciraju. Iako sve komponente koje se testiraju rade bez greške same za sebe to ne znači da će one raditi pravilno kada jedna komponenta ovisi o drugoj. Automatizacijom takvih testova pomoću skripti dobiva se na brzini izvođenja, uštedi vremena gdje računalo odrađuje većinu posla, učestalosti i smanjenom utjecaju ljudskog faktora na greške. Kako testovi koje 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 koju 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 testova. Testiranje koda jednako je bitno kao i izrada same programske aplikacije. Često razvojni programeri požuruju s isporukom aplikacija dok nisi istestirane sve komponente i svi mogući ulazni parametri koje korisnik može unijeti koji bi potencijalno uzrokovali nepravilan rad aplikacije. 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. Svakodnevnim izvođenjem testova pogreške se odmah mogu detektirati te nije potrebno čekanje razdvojih programera za izbacivanje nove verzije aplikacije na testiranje. Testiranje temeljno na svojstvima pokazalo se vrlo korisnim te je za daljnje unaprjeđenje projekta potrebno ukomponirati takvu vrstu testiranja za ostala sučelja logiRECORDERA.

# Literatura

1. „myservername.com“, s Interneta, https://hr.myservername.com/what-is- component-testing, 9.8.2022.
2. „ITpedia.com“, s Interneta, https://hr.itpedia.nl/2019/05/17/wat-is-integratietesten-en-waarom-doen-we-het/, 9.8.2022.
3. „tutorialspoint.com“, s Interneta, https://www.tutorialspoint.com/what-is-a-PyTest-framework, 9.8.2022.
4. „guru99t.com“, s Interneta, https://www.guru99.com/PyTest-tutorial.html#11, 9.8.2022.
5. „docs.PyTest.org“, s Interneta, https://docs.PyTest.org/en/7.1.x/history.html, 16.8.2022.
6. „tutorialspoint.com“, s Interneta, https://www.tutorialspoint.com/PyTest/PyTest\_grouping\_the\_tests.htm, 20.8.2022.
7. „docs.PyTest.org“, s Interneta, https://docs.PyTest.org/en/6.2.x/parametrize.html, 20.8.2022.
8. „hypothesis.readthedocs.io“, s interneta, 25.8.2022, https://hypothesis.readthedocs.io/en/latest/
9. „inspiredpython.com/“, s interneta, 25.8.2022, https://www.inspiredpython.com/course/testing-with-hypothesis/testing-your-python-code-with-hypothesis
10. „hypothesis.readthedocs.io“, s interneta, 25.8.2022, https://hypothesis.readthedocs.io/en/latest/data.html
11. „medium.com“, s interneta, 25.8.2022, https://medium.com/criteo-engineering/introduction-to-property-based-testing-f5236229d237
12. „itea4.org“, s interneta, 25.8.2022, https://itea4.org/organisation/11130-quviq-ab.html
13. [13] „logiRecorder 3.2 AUTOMOTIVE HIL VIDEO LOGGER “s interneta, 3.1.2023, https://xylon-lab.com/product/logirecorder/
14. [14] „logiRecorder DASHBOARD“, s interneta 3.1.2023, https://xylon-lab.com/product/logirecorder-dashboard-2/
15. [15] „Welcome to pySerial's documentation“, s interneta 6.1. 2023, https://pyserial.readthedocs.io/en/latest/
16. [16] “Python.Net”, s internet 6.1 2023, http://pythonnet.github.io/
17. [17] “Python’s Requests Library (Guide), s internet 6.1 2023, https://realpython.com/python-requests/
18. [18] “Pyfunctional”, s internet 6.1. 2023, https://github.com/EntilZha/PyFunctional
19. [19] “What is the controller area network (CAN) bus?“, s interneta 8.1.2023, https://www.microcontrollertips.com/what-is-the-controller-area-network-can-bus-faq/
20. [20] „Controller Area Network (CAN Bus) - Message Frame Architecture“, s interneta 8.1.2023, https://copperhilltech.com/blog/controller-area-network-can-bus-message-frame-architecture/
21. [21] „Understanding AUTOSAR ARXML for Communication Networks“ s interneta 9.1.2023,https://www.intrepidcs.net.cn/wp-content/uploads/2019/05/202.\_Understanding\_ARXML\_EEA\_COM\_TD\_USA\_2019.pdf
22. [22] “CAN FD with dynamic multi-PDU-to-frame mapping“, s interneta 9.1.2023, https://www.can-cia.org/fileadmin/resources/documents/proceedings/2015\_decker.pdf
23. [23] „Manual Vector Logger Configurator“, s interneta 10.1.2023, https://usermanual.wiki/m/46eb93b67b2628b5e07f77a57aba10d3d65cbb47439001898e04d5191dc72adb
24. [24] „CANalyzer-manual“, s interneta 11.1.2023, https://manuals.plus/m/8ad4bdef518565c653a90638fd5aa1178d81da2864233edc39a0faddd5d572ba.pdf