Propisi i standardi verifikacije softvera u automobilskoj industriji

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Andrea Pilipović
andrea.pilipovic@yahoo.com
Vojkan Cvijović
vojkancvijovic@gmail.com
Strahinja Stanojević
strahinjastanojevic@rocketmail.com
Saša Cvetković
sasa.cvetkovic@yahoo.com

18. april 2018.

Sadržaj

1 Uvod

U velikoj većini različitih industrija se koriste embeded sistemi kako bi se podigla pouzdanost različitih sistema. Obično su to sistemi koji zahtevaju visoku pouzdanost i bezbednost. Sistemi od kojih mogu zavisiti i ljudski životi. Jedna od takvih industrija je i automobilska industrija. Potreba za korišćenjem softvera u ovoj industriji kao i sama njihova upotreba je značajno porasla u poslednjoj deceniji [1] [2]. Većina funkcionalnosti u modernim automobilima, posebno one funkcije koje su vezane za bezbednost, kao što su automatske kočnice ili automatski asistenti pri vožnji [3], kontrolišu se pomoću softvera [4]. Predviđa se da će oko 90% automobila u budućnosti biti voženo softverski, bez ljudske pomoći [2].

2 Softver u automobilskoj industriji

Automobilska industrija je tradicionalno pretežno bazirana na konceptima mašinskog inženjerstva [5][6]. Baš kao i neke druge industrije koje su tradicionalno ne-softverske, kao što su avionska, odbrambena ili raketna, i automobilska industrija biva pogodjena softverskom revolucijom. Sve veći broj, kako kritičnih tako i ne kritičnih, funkcionalnosti automobila počinje da bude kontrolisano pomoću softvera. Stoga je razumevanje softvera koji se koriste u automobilskoj industriji sve važnije, i mnogo se više pažnje pridaje nego u pre par decenija. Broj (eng. Broy) u [6] opisuje softver u ovoj industriji na sledeći način:

veličina: Za 40 (od 1976. [7]) godina broj linija koda u automobilskoj industriji se povećao sa nula na nekoliko miliona.

uloga: Funckije u vozilima, kako kritične kao što su kočioni sistem, tako i ne kritične (radio, klima i slično) danas se kontrolišu pomoću softvera.

interakcija i rasprostranjenost: U automobilima postoje ECU sistemi, koji su u suštini mikrokontroleri koji odgovaraju softverskim komponentama [**?**]. ECU sistemi rade zajedno kako bi izvršili potrebnu funkciju automobila. Te funkcije mogu biti veoma raznorodne, kao na primer kontrolisanje motora, vazdušnih jastuka ili prikaz nivoa goriva [8]. U prošlosti, svaki ECU sistem je izvršavao posebnu funkcionalnost. Zbog toga su softverski mogle da se izvršavaju samo funkcije koje je mogao da izvršava jedan ECU sistem. Ranih devedesetih, ECU sistemi su bili povezani lokalnom mrežom CAN (eng. Control Area Network) pomoću koje su mogli da komuniciraju i dele informacije. Ovo je dovelo do funkcija koje su bile kompleksnije i bilo je potrebno da ih zajedno izvršava više ECU sistema. U kasnim devedesetim, funkcije su bile toliko kompleksne da je bilo potrebno da postoji komunikacija izmedju više CAN mreža. Danas, sem komunikacije medju CAN mrežama unutar samog automobila, vrši se i njihova komunikacija izmedju CAN mreža i spoljasnjeg okruženja pomoću radio veza [2].

Do sada navedene karakteristike su samo dokaz da je upotreba softvera jako velika u automobilskoj industriji danas. Takodje, pokazuje da je softver neizostavni deo današnjih automobila. Bojl dalje u [7] opisuje karakteristike softvera

u automobilskoj industriji. Naredne stavke govore o kompleksnosti samog softvera.

heterogenost podsistema: Pri izradi softvera za automobil koristi se modularnost, kao i uopste pri proizvodnji automobila. Tako se rade softveri za kontrolisanje različitih sistema, kočnica, menjača, motora, a zatim se oni pojedinačno testiraju. Nakon uspešnog testiranja, sistemi se objedinjavaju u jedna jedinstven sistem vozila.

proivođači originalne opreme (eng. Original Equipment Manufacturer (OEM)): Proces integracije heterogenih sistema je sam po sebi zahtvean, međutim ovaj zadatak je daleko kompleksniji. To je tako zbog činjenice da veliki broj sistema za vozila dolazi od različitih proizvođača. Kasnije je zadatak automobilskih kompanija da ove sisteme i softvere dobijene od različitih proizvođača objedine i integrišu.

visoko konfigurabilni softver: Softveri u automobilskoj industriji su visoko konfigurabilni, s obzirom na to da postoji veliki broj različitih funkcija koje automobil obavlja. S toga, veliki broj različitih varijanti vozila može biti proizveden od modularnih delova. Na primer, autori [7] navode primer vozila sa 80 različitih softverskih jedinica, gde jednostavan izbor da li se funkcionalnost svakog od njih uključuje ili ne vodi do (2 na 80) mogućnosti. Iz ovoga se zaključuje da je moguće napraviti više varijanti vozila koji se razlikuju po funkcionalnostima koje su uključene i dobijene su kombinovanjem različitih softverskih jedinica.

3 Značaj verifikacije u automobilskoj industriji

Verifikiacija softvera, i testiranje kao jedna od oblasti verifikacije je doživela ekspanziju poslednjih godina, posebno u osetljivim oblastima, kao što su avionska, raketna, automobilska i slične. Ipak, što se same automobilske industrije tiče, ne postoji previše studija koje se fokusiraju na testiranje i verifikaciju na višim nivoima, pri integraciji pojedinačnih komponenti. Međutim, postoji potreba za fokusom i akademskih i industrijskih istraživača u ovoj oblasti.

S obzirom na sve veći broj funkcionalnosti koje pružaju današnji automobili, softveri bivaju sve kompleksniji. Jedan od glavnih zadataka u automobilskoj industriji je dizajnirati efektivne i efikasne procese izrade i testiranja softvera [9]. S obzirom na sve kompleksnije i veće softvere sa sve većim brojem mogućnosti i funkcionalnosti, njihova verifikacija i testiranje postaje dosta kompleksan zadatak [10]. Neotkrivene greške u softveru u ovoj industriji mogu imati jako ozbiljne posledice. Može doći do saobraćajnih nesreća, povreda, pa čak i do gubitka ljudskih života. Zbog ovoga je izuzetno važno izvršiti detaljno testiranje svih delova softvera. Samo jedna greška može dovesti do katastrofalnih posledica [11]. Upravo iz ovog razloga, oko 50% od ukupnog vemena koje se utroši na tehničke aktivnosti (vezane za izradu softverskog dela) u razvoju vozila se potroši na testiranje samog softvera [12].

Neki od primera kada su greške u softveru mogle da dovedu do katastrofalnih posledica:

Proizvodjač automobila Tojota (eng. Toyota) je 2009. i 2010. godine imao prijave vozila koja su ubrzavala sama od sebe bez ikakvog razloga. Ispostavilo se da je problem bio u softveru koji je kontrolisao ubrzavanje vozila. Isti proizvođač je imao jos prijava Nakon prijave jos jednog problema na nekoliko vozila kompanija je odlučila da napravi novi hibridni softver. Drugi problem je bio vezan za kočioni sistem, što je moglo rezultovati čak i ljudskim žrtvama.

14. februara 2016. godine, Guglov (eng. Google) samovozeći automobil (self-driving car) je izazvao nesreću na semaforu. Kompanija je priznala da je problem bio greška softvera u predviđanju kretnje autobusa, kao i u odluci šta sam automobil treba da uradi, odnosno kuda da ide. Nakon otvranja zelenog svetla za automobil, on je uspešno detektovao autobus, međutim pretpostavka koju je napravio je da će autobus usporiti, tako da je krenuo dalje. Nesreća nije bila velikih razmera.

4 Izazovi u verifikaciji softvera u automobilskoj industriji

U ovom delu dajemo kratak pregled izazova koji se mogu sresti u verifikaciji softvera u automobilskoj industriji [13][14][15].

merenje napora: Poteskoće u određivanju važnosti testiranja na različitim nivoima [14].

znanje osoblja: Različita mišljenja različito stručnih ljudi [14].

distribuirane funkcionalnosti: Visoka kompleksnost testova usled distribuiranosti softvera [14].

metrika pokrivenosti: Nedostatak podrške za merenje testova [14].

različite varijante: Kombinatorna eksplozija testiranja zbog visokog stepena prilagođavanja [14].

zahtevi i mogućnost praćenja: Problemi vezani za zahteve kao što su nedostatak jasnih zahteva za testove visokog nivoa i nedostatak praćenja napretka kao prepreka za verifikaciju [13].

proces testiranja: Odsustvo jedinstvenog procesa testiranja [13].

alati za verifikaciju: Nedostatak adekvatnih alata i tehnika za verifikaciju [13].

testiranje i praćenje nedostataka: Visoka cena popravljanja nedostataka kao i neotkrivenih nedostataka [13].

dokumentacija: Nedostatak odgovarajuće i ažurne dokumentacije [13].

dokumentacija: Preklapanje testova na različitim nivoima testiranja dovodi do gubitka vremena i resursa [15].

5 Efikasno testiranje u automobilskoj industriji

Softverski sistemi postaju sve više kompleksniji i sve više se izvršavaju u okruženjima gde je bezbednost od izuzetne važnosti, na primer sistemi za pomoć pri upravljanju vozilom. Stoga kvalitet softvera je vrlo bitna stavka kod automobilskih softverskih sistema. Testiranje softvera je jedna od najbitnijih stavki prilikom razvoja softvera za automobile. Bilo kakve greške na softveru koje se ne otkriju prilikom testiranja, mogu da izazovu ogromne finansijske probleme i čak da rizikuju ljudske živote.

Proces testiranja softvera je, prema Spillner i Linz, podeljen u pet faza, kao što je prikazano u figuri 1. Počinje se sa fazom planiranja. Koncept testa i plan testa su definisani u ovoj fazi. Resursi moraju biti raspoređeni i ciljevi svakog testa moraju biti definisani. Posle početnog planiranja, sledi faza analize i dizajna. U toj fazi svi slučajevi se opisuju na apstraktan način i definišu se prema specifikaciji softvera. Nakon toga svi test slučajevi se moraju implementirati, dalje se vrši odabir testova, pokreću se na sistemu za testiranje, ručno ili automatski. Kada se izvršavanje testova završi, rezultati se sakupljaju i dokumentuju za dalju analizu. Osoba odgovorna za projekat mora da odluči koji su dalji testovi neophodni. Ovo se može utvrditi na primer, posmatranjem učestalosti dešavanja kvarova i popravki na sistemu. Novi testovi se izvršavaju ako je to potrebno.

Generalno testiranje je u velikoj meri ograničeno resursima stoga je cilj da testiranje bude efikasnije i da u isto vreme bude i efektivnije. U automobilskom softverskom testiranju, postoje posebni izazovi. Prvo, u automobilskoj industriji postoji posebna veza između proizvođača, dobavljača i testera. Pošto su softverske komponente obično razvijene od strane dobavljača i testirane lokalno, integraciju obavlja proizvođač ili je izvršena od strane drugog dobavljača. Usled ugovorenih ograničenja, izvorni kod nije dostupan za integraciono i sistemsko testiranje. To dovodi do metode testiranja crnom kutijom gde je testiranje zasnovano na specifikaciji sistema. Drugo, proizvodi testiranja su često naznačeni u prirodnom jeziku [20]. Ovo otežava pravljenje pretpostavki u vezi progresa testiranja i pokrivenosti sistema na kome se testira i automatizacije izbora test slučaja i njihovog izvršavanja. Treće, u većini slučajeva, svi naznačeni test slučajevi ne mogu biti izvršeni zbog ograničenja resursa. Stoga, mora biti izabran podskup test slucajeva. Ovo je veoma kritičan zadatak, pošto nije jednostavno da se izabere pravi test slučaj, koji će čverovatno uočiti grešku. U metodi testiranja crnom kutijom, koja je greška za automobilske softvere i sisteme, izvorni kod nije dostupan. To dovodi do nedostatka popularne pokrivensti koda. Umesto toga, drugi heuristike moraju da budu uvedene u automobilsko područije za izbor test slučajeva. Četvrto, u automobilskom softverskom razvoju, dosta pod-procesa od celokupnog procesa testiranja se izvršavaju ručno. Automatizacija testiranja procesa, uključujući izbor i prioritet test slučaja, je željeni cilj, pošto može da smanji napor testiranja u poređenju na ručno izvršenje[16].

Kako bi automobilsko softversko testiranje napravili više efikasnim, u ovom papiru, predstavićemo šest različitih izazova testiranja i prema rešenjima u automobilskom razvoju softvera uzimajući u obzir određena ograničenja i izazove

u ovom područiju. Ovaj pristup pokriva test slučajeve u prirodnom jeziku, automobilizaciju izbora i prioritizacija test slučaja. Prikazaćemo zadati koncept pomoću primera, Body Comfort System [17].

6 Analiza i dizajn

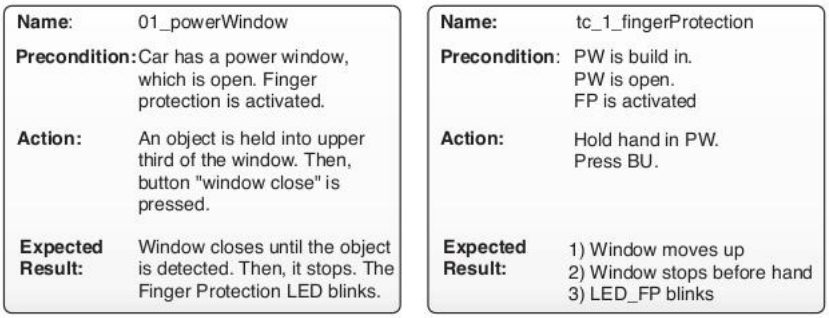
U ovoj fazi glavni zadatak je kreiranje slučajeva za testiranje, kako to predstavlja osnos samog procesa testiranja. Ova faza je važna za bilo koju formu testiranja, kako se u njoj definiše koje informacije su dostupne i kako im pristupiti prilikom testiranja. Greške iz ove faze mogu biti jako skupe u narednim fazama, na primer, nedovoljan opis testa, redudanran test slučaj. Što se pre otkriju takve greške, to ih je lakše rešiti. U deljem teksu se opisuju dva načina na koji mogu da se unaprede testovi u ovoj fazi testiranja. Prvo se opisuje specifikacija za opis test slučajeva, dalje se predlaze koncept za uklanjanje redudantnih test slučajeva.

6.1 Specifikacija test slučajeva koji se testiraju metodom crne kutije

U fazi analize, test slučajevi su neretko formulisani u nekom govoornom jeziku. Uobičajno je da se test slučaj sastoji od preduslova, akcije i očekivanog rezultata. Akcije su podeljene na veći broj koraka koji tester mora da izvrši. Test slučajevi su obično povezani određenim uslovnostima, što je obično zadatko u specifikaciji to test slučaja. I preduslovi i koraci test slučaja se uglavnom čuvaju u odrađenim alatima poput, HP Quality Center, IBM DOORS. Takvi alati omogućavaju praćenje test slučajeva, njihovih razlika i preduslova. Cilj u testiranju metodom crne kutije je da se postigne zacrtana pokrivenost zahteva testovima. Na primer, test slučaj za zaštitu prstiju prilikom zatvaranja prozora, može biti opisan u govornom jeziku. Specifikacija ove funkcionalsnosti se sastoji od, u slučaju da je omogućena zaštita prstiju, detektuje se objekat, prsti, prilikom zatvaranja prozora i zatvaranje prozora se preikida kako ne bi doslo do povrede. Test slučaj ove funkcionalnosti moze se definisati na dva različita načina kao što je pokazano na slici 1. Najčešći problem prilikom definisanja test slučaja je sam opis test slučaja. Kako se govorni jezik koristi za opis, od testera zavisi kako će test slučcaj biti opisan.

Na primer, u prikazanim testovima na slici 1, u opisu jednog testa se nalaze pune rečenice dok kod drugog se nalaze samo stavke. Skraćenice mogu da izazovu probleme, moraju se definisat. U suprotnom test slučaj postaje nedovoljano jasan drugim saradnicima. Postoje mnogi slučajevi gde defincije test slučajeva mogu da se razlikuju ali da se odnose na istu funkcionalnost.

U slučaju da se test slučajevi analiziraju, sledeći aspekti se moraju razmotriti: da li je neka komponenta ili funkcionalnost već pokrivena pokrivena više nego ostale funkcionalnosti, da li postoje redudantnosti među testovima, da li su neki test slučajevi previše komplikovani, da li određeni testovi mogu da se iskombinuju. Ako bi se koristio alat za analiziranje test slučajeva, moglo bi da



Slika 1: Test slučaj otiska prsta

doje do problema u slučaju da opisi testtova nisu konzistentni. Ovo zahteva da se testovi dizajniraju korišćenjem određenog vokabulara, da sadrže samo određene fraze i unapred definisane strukture. Na primer, potrebno je da test dizajner piše PowerWindow i definise redom preduslove, akcije i očekivani rezultat. Može biti definisano, koliko najviše koraka može jedan test slučaj da sadrži. Potrebo je definisati sve neophodne prekidače, pojlja, parametre i način kakko se oni referišu u opisu test slučaja. Ovo dovodi do uniformisanja test slučajeva. Na kraju, test slučajevi moraju biti podržani od strane korišćenih alata. Ti alati mogu da predlože dizajneru testova, koje parametre da refereiše, gde da ih piše...

6.2 Uklanjanje redudansi

Izaziv je rukovati velikim skupom testova prilikom razvoja softvera. Broj test slučajeva se uvećava vremenom, kako novi projekti u kompanije se često baziraju na prethodnim verzijana softvera i koriste proizvode koje su kreirane u drugim projektima [18]. Takvi proizvodi mogu biti uslovni, softverske komponente, test slučajevi. Ali ponovno korišćenje može da dovede do redudantnosti među proizvodima. Na primer, razvija se nova verzija sistema, za novi model. Prethodna verzija je imala nekoliko hiljada testova, od kojih većina može biti ponovo korišćena. Testovi se kopiraju u novi projekat ali potrebno je testirati nove funkcionalnosti. Ovo dovodi do toga da se dizajniraju novi testovi. Kako tester ne može da proveri opis i cilj hiljadu starih testova, moguće je da pojedini novi testovi su već pokriveni starim testovima. Ovo može lako da se dogodi u slučaju da više od jednog dizajnera radi na istoj funkcionalnosti.

Redudantnost može da dovede do velike neefikasnosti testiranja kako se jedan isti test slučaj testira nekoliko puta bez promene rezultata. Jedno rešenje jeste da se ne koriste gotovi proizvodi. To bi značilo da testeri moraju da napišu svaki test slučaj od početka. Pa sledi da ovo nije bas efikasan način, zbog velikog broja testova. Potreban je pametniji pristup, pristup koji može da detektuje i otkloni redudantnosti dok u isto vreme omogućava veliku iskorišćenost gotovih proizvoda. Testovi se porede kako bi se našli isti delovi. Ovo zahteva da test slučajevi su napisani na taj način da mogu da se porede. Nakon procesiranja

specifikacije test slučaja, redudantni test slučajevi mogu biti obeleženi ili u potpunosti uklonjeni. Kada se piše novi test slučaj, test dizajner može biti direktno upozoren ako je novi test slučan postojećem.

7 Implementacija i egzekucija

Kada su svi testovi opisani, potrebno je da budu i izvršeni. Moguće je da se za testove visokog nivoa moraju implementirati i neke dodatne test skripte. Iako su test slučajevi sistematično opisani i bez redudantnosti, mora se pronaći balans izmedju potencijalno velikog broja dizajniranih testova i veoma ograničenih resursa za testiranje. Glavni izazovi su selekcija test slučajeva koji će biti izvršeni i njihovo sortiranje. Prioritizacija test slučajeva [21][19] je neophodna i uglavnom ne mogu svi odabrani testovi biti izvršeni zato što neki testovi moraju biti ručno izvršeni. U proiritizaciji testova, važniji testovi se izvršavaju prvo. Zbog toga se neki ciljevi testiranja postižu ranije, što je poželjno, pogotovu ako ne mogu si odabrani testovi biti testirani. U automobilskoj industriji odobir testova i njihova prioritizacija predstavljaa izazov s obzirom da izvorni kod sistema koje se testira je nepoznat. Takvo testiranje metodom crne kutije omogućava vrlo ograničene informacije, poput pokrivenosti koda, za razliku od tehnika metode bele kutije.

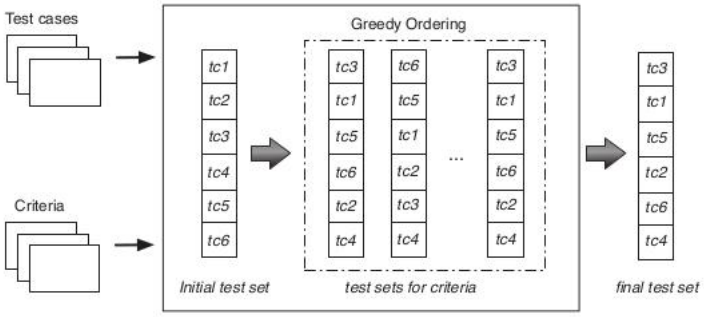
7.1 Efektivna selekcija testnih slučajeva i prioritizacija u testovima crne kutije

Prvo pitanje, na koji treba dati odgovor, jeste šta zapravo predstavlja jedan dobar testni slučaj. Ovo pitanje nas dovodi do činjenice da je neophodno uložiti veliki napor pri izboru slučajeva testiranja. Kaner [22] je istraživao ovo pitanje i opisao kvalitet testnog slučaja koristeći nekoliko faktora. U opštem testiranju, dobar testni slučaj se može definisati prema nekoliko različitih kriterijuma. Neki od mogućih primera su slučajevi kada test ima veliku verovatnoću da pronađe grešku, test testira funkcije koje se verovatno koriste u stvarnosti itd. Zbog toga, za odabir test slučajeva je važno odabrati one testne slučajeve koji odgovaraju trenutnom testnom cilju.

Pretpostavljajući korišćenje slučajeve testiranja na visokom nivou u slučaju scenariju testiranja crne kutije, kreiranje test slučajeva je obično ručni zadatak, koji podrazumeva veliku količinu stručnog znanja i iskustva u vezi sa sistemom koji se testira. To dovodi do dva glavna problema. Prvo, poznavanje kreiranih testova slučajeva i njihovog dizajna se često gubi kada osoba napusti projekat ili kompaniju. Baš zbog te činjenice, druge osobe ne mogu da ponove testiranje. Zadatak kreiranja novih testnih slučajeve je proces koji može dugo traja zbog potrebe za razvijanjem potrebne veštine za kreiranje đobrih"testnih slučajeva. Drugo, ručna selekcija je neefikasna u pogledu raspodele testnih slučajeva, vremena za odabir i neuspeha. Kako procedura izbora nije formalno definisana, rezultati nisu predvidljivi, pa zavise od lične procene osobe koja kreira test slučaj. Podrška u vidu alata može dosta da pomogne u kreiranju stvaranja testova na mnogo strožiji i sistematizovaniji način. Neiskusnim testeri može

biti vodilja pri izboru i kreiranju probnih test slučajeve sa ciljem da prvo njih testiraju sa većom verovatnoćom pronalaženja neuspeha.

Da bismo to omogućili, predstavljamo poželjnu prioritizaciju testa i tehniku selekcije koja je pogodna za testiranje crne kutije. Ovo zahteva definisan skup testnih slučajeva i skup kriterijuma za odabir / prioritizaciju kao ulaz. Kao rezultat se dobija uređena lista testnih slučajeva. Šematski opis ovog pristupa je prikazan na 2. Pristup koristi unete test slučajeve koji su sortirani na neki proizvoljan način, abecedno ili nakon datuma nastanka. Odabran je skup unapred definisanih kriterijuma, kao što je vreme merenja, prioritet, testirana funkcionalnost ili istorija testnih slučajeva. Na primer, vreme izvršavanja test slučaja može se koristiti da bi se izabrali oni koji se izvršavaju najbrže. Ovo bi bilo korisno u vrlo ograničenom okruženju. Još jedan od mogući kriterijum je istorija testiranja, koja obuhvata broj neuspeha pronađenih testom i njihovim prioritetom. Takvi kriterijumi su korisni u scenariju testiranja regresije, gde se testni predmeti moraju izvršiti nakon nekog ažuriranja softvera. Ranije neuspešna funkcionalnost bi se trebala ponovo testirati. Pohlepni algoritam izračunava različite uređene testne grupe za svaki od izabranih kriterijuma. Najbolji testni slučaj u odnosu na izabrani kriterijum uvek se postavlja u prvi plan. Da biste dobili rezultat test slučaja, uređenja su kombinovana. Kako bi se ovaj korak učinio boljim za različite projektne domene i faze projekta, testovi mogu biti kombinovani



Slika 2: Pohlepna regresije strategije selekcije

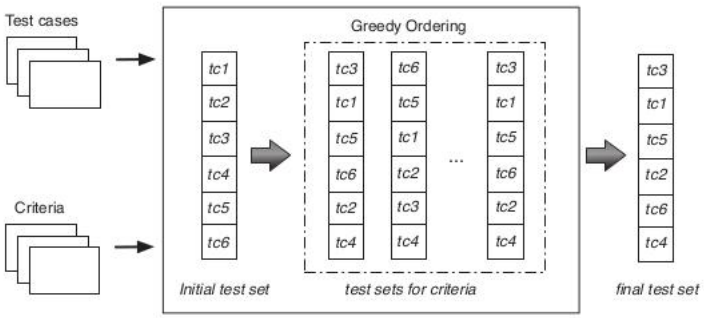
na neku funkciju težine između kriterijuma. Težine za svaki kriterijum definisani su od strane testera ili se mogu automatski podešavati. Na primer, ako se izda novi softver , gde je važna istorija testa, ali su resursi za testiranje ograničeni, vreme provere testa i istorija testova se koriste kao kriterijumi. Istorija testa može biti 75%, a vreme izvršenja sa 25% da utiče na rezultat. Testni slučajevi se uređuju u obliku, da je prvi test slučaj najvažniji, a zatim sledeći drugi najvažniji i sl. Ograničeno u vremenu i resursima, tester pokušava da testira što je moguće više testova u datom porudžbini.

Ovaj pristup je veoma pogodan za testiranje crno-kutije u automobilskoj industriji, pošto definisani kriterijimi nemaju. zahteve za bilo kojim znanjem o softveru i samom izvornom kodu, već samo zavise od testnih slučajeva. Ovo rešava problem nedostatka metoda kodova, koji bi se primarno koristio u testiranju u beloj kutiji. Nije garantovan pronalazak optimalnog test slučaja, ali ovaj pristup može se koristiti kao brz način da se pronađe prilično dobar rezultat. Ključni faktor za uspeh ovog pristupa su definisani kriterijumi i njihove težine.

7.2 Kombinacija test slučajeva

Jedno od najčešćih ograničenja u testiranju je raspoloživo vreme. Jedna od mogućnosti je da se povećava broj izvršenih testnih slučajeva pod vremenskim ograničenjima je kombinacija testnih slučajeva. Slično pristupu opisanom u delu 8.2, uređenje testnih slučajeva može dovesti do povećane efikasnosti testiranja. Prvo izvršavanje važnih probnih slučajeva je samo dobra pretpostavka, propusti su skriveni i nepoznati i testeru i njegovom alatu. Smanjenje vremena izvršavanja testa može dovesti do poboljšanih rezultata u poređenju sa ručnim izborom testa. Ovo se može postići njihovim uređenjem prema njihovim pre – uslovima i post - uslovima.

Za studiju slučaja BCS postoje dva testna slučaja za funkciju kontrole prozora električne energije, koja omogućava korisniku da pomera prozor pomoću dugmadi na daljinskom upravljačkom tasteru. Test slučajevi su prikazani na slici 3. Vidimo da se ova dva slučaja mogu povezati sa logičkim redosledom, pri čemu se prvi prozor pomera prema gore, a potom se pomeri dole. Post - stanje prvog testnog slučaja je preduslov drugog.



Slika 3: Dva primera testnih slučajeva za kontrolu električnih prozora

Iako je dat primer prilično trivijalan, pronalazak takvih struktura i koherencija u nekoliko hiljada testnih predmeta, koji su napisani na prirodnom jeziku, nije. Jedna mogućnost bi bila istražiti pojmove i rečenice prirodnog jezika kako

bi se pronašle moguće kombinacije testnih slučajeva u testne sekvence. Ovo zahteva jedinstvenu specifikaciju testnih slučajeva kako je opisano u delu [23]. Druga mogućnost je da se koriste pristupi zasnovani na modelima [23] kako bi se napravile testne sekvence iz nekoliko testnih slučajeva.

Algoritmi zasnovani na grafikonu omogućavaju brzu obradu i upoređivanje u velikim skupovima testnih modela. Iako je testiranje zasnovano na modelima široko razmatrana i istražena tema, jedan poznati problem je nedostatak takvih modela u industrijskom kontekstu. Ovo je dovelo do različitih pristupa za testne modele koji su zasnovani na "testnim slučajevima"[24][25]. Mogu se analizirati ovi modeli i videti koji delovi ovog modela su pokriveni testnim slučajevima. Moguće je kombinovanje pri čemu je konačno stanje jednog testnog slučaja početno stanje drugog. Pored toga, sprovedeni modeli se mogu koristiti za izvođenje novih testnih slučajeva za sistem. Što je bolji model to su detaljniji i različitiji testovi koji mogu da se generišu.

8 Rezultat evaluacije i izveštaj testnih rezultata

Nakon implementacije i izvršenja testnih slučajeva, rezultati testova moraju biti ocenjeni. Bez evaluacije, ne može se izvući znanje, niti se dobijaju nove informacije. Osim toga, važno je prilagođavanje procesa testiranja i testnih slučajeva. Ovo uključuje promenu odabira testnih predmeta, prioritizaciju testiranja, raspodelu resursa itd. U testu regresije, Kaner [22] definiše cilj da napiše testni slučaj ko bi mogao ponovno da se iskoristi. Dobar test za regresiju bi verovatno trebalo da propadne, ako promene izazivaju greške u testiranoj oblasti programa koji se testira. U regresijskom testiranju, ti testni predmeti su odabrani za izvršavanja jer najverovatnije otkrivaju neuspehe uvedene promenama na već testiranom sistemu. Ovaj proces se trenutno visoko zasniva na stručnom znanju i sposobnosti testera da imaju uvid u trenutni status testa. U nastavku ćemo prikazati dva moguća pristupa zasnovana na mašiniskom učenje za automatizaciju ovog procesa.

8.1 Mašinsko učenje u regresionom testiranju

Slučajevi testiranja visokog nivoa često se označavaju meta-podacima, npr. prioritet testnih slučajeva, testirana funkcionalnost, istorija izvršavanja testa. Meta-podaci se mogu koristiti za odabir testnih predmeta i određivanje prioriteta, kako je opisano u delu . Na primer, u okviru studije slučaja BCS može se tvrditi da je test slučaj vezan za funkcionalnošću alarmnog sistema ima viši prioritet od testnog slučaja koji testira odgovarajuću LED indikaciju za ovu funkciju. Metapodatke definiše dizajner testova, i predstavlja skup proces u njegovom stvaranju, jer nije trivijalno odlučiti, na primer, koji je prioritet test-slučaja. Ovaj problem se povećava sa svakom testnom iteracijom, jer se pridruženi meta-podaci trebaju menjati tokom vremena, jer se sistem koji se testira veoma menja (npr., Popravke grešaka, nove funkcije itd.). To dovodi do pitanja kako se meta-podaci mogu menjati nakon testne kampanje bez oslanjanja na ručnu intervenciju i stručno znanje.

Jedno od mogućih rešenje za izbegavanje ručne intervencije prilikom prilagođavanja meta-podataka nakon testnih kampanja jeste primena tehnike mašinskog učenja. Mašinsko učenje se već koristi u testiranju softvera [26][27] . Mašinsko učenje pristupa identifikovanju uzoraka u podacima, koji se mogu koristiti za izradu novih znanja i mogu se primeniti za donošenje odluka. Mašinske tehnike mogu analizirati specifikaciju testa, zahteve, rezultate testa i dostupnost testiranja njegove istorije. Na osnovu nalaza, ML pristup može prilagoditi meta-podatke povezane sa testnim slučajevima za sledeću testnu iteraciju i proces selekcije. Na primer, može se zamisliti da klasifikator, koji odlučuje da li će probni slučaj verovatno otkriti neuspeh ili ne, i stoga mu dodeliti viši prioritet za sledeće izvršenje testiranja.

Predstavljeni test je opisan na slici 1. Pretpostavimo da je test slučaj izabran za test slučaj i izvršen je. Test slučaj je otkrio značajan neuspeh. Automatski mašinsko učenje odlučuje da prioritet testnog slučaja mora biti promenjeno sa prethodne vrednosti od 2 na 4, jer je grešku potrebno proveriti.

Slika 4: Ilustruje primer studije BCS

8.2 Automatska selekcija kriterijuma za testiranje regresije

Selekcija test slučaja za regresiono testiranje je rekurzivni zadatak. Kao što je opsano u delu , može se uvesti koncept, pri čemu se specifični kriterijumi selekcije koriste za odabir uređene liste testnih slučajeva. Opisani metod i dalje zahteva ljudski ulaz, pošto meta-podaci dodeljuje testove za odabir, kriterijume selekcije i testeri moraju obezbediti odgovarajuće težine. Kao što je prikazano u prethodnom odeljku, tehnike mašinskog učenja se mogu koristiti za prilagođavanje meta-podataka u kontekstu testiranja crne kutije. Međutim, nakon svakog testiranja, tester bi i dalje morao da odluči kako treba da izmeni prethodno korišćene kriterijume i njihove težine. Zbog toga se predlaže proširenje pristupa dela kako bi ga učinili automatizovanijim. Umesto ručnog definisanja kriterijuma selekcije, dobijeni podaci iz prethodnih testova mogu se koristiti za prilagođavanje kriterijuma i težina za sledeću test kampanju. Na primer, pretpostavimo da postoji definisan kriterijum za testiranje da se određena funkcija, npr. zaštita prstiju, uvek testira. U okviru najnovije testne kampanje, nisu pronađeni nikakvi neuspesi za ovu funkciju. Nakon toga, testni sistem će automatski odbiti izbor ovog kriterijuma za izbor ili mu dodeliti nižu težinu u sledećem testiranju. Ovo dovodi do veoma fleksibilnog pristupa testiranju, gde se testeru pomaže u donošenju odluka, ali i dalje može ručno uneti podešavanja.

9 Slični radovi

U ovom delu pregledan je rad testiranjem crne kutije uopšte i specifikacijama slučajeva testiranja prirodnog jezika, odabirom testnih predmeta i prioritizacijom i testiranjem zasnovanih na mašinskom učenju.

Nesistematski specifični testovi mogu dovesti do velikog napora u uvođenju automatizacije. Barros et al. predstavljaja kontrolisani prirodni jezik za slučajeve upotrebe [28]. Strobbe i sar. definisao je jezik testiranja test slučajeva u obliku KSML strukture za opisivanje meta-podataka testnog slučaja [30]. Ova ograničena jezika pomažu u kreiranju uporedivih i obradivih specifikacija testnih predmeta. Takođe postavljaju osnovu za moguće alatiranje. Selekcija testnih predmeta i prioritizacija su obuhvaćeni u literaturi u kontekstu izbora regresijskog testa i efikasnog testiranja linija softverskih proizvoda. Evolucioni algoritmi su pokazali da su vrlo zgodni u prioritizaciji testnih slučajeva [29]. Regresiono testiranje dovodi do problema u izboru i prioritizaciji testnih slučajeva. Engstrom i sar. [35] su sproveli sistematski pregled empirijskih evaluacija pristupa selekcije regresionog testa. Oni primećuju da je upoređivanje pregledanih 28 tehnika prilično teško, pošto zavise od mnogih faktora, koji takođe intuitiraju odgovarajuće procene. Neki primeri za takve faktore su granularnost tehnike, koliko su česti regresijski ciklusi ili proizvodi koji se testiraju. Softverske linije proizvoda su veoma relevantne u automobilskom kontekstu, pošto automobile mogu da konfiguriše klijent na više načina. Engstrom i Runeson [31] su istraživali tehnike testiranja linija softverskih proizvoda. Oni su identifikovali tri glavna istraživačka izazova u ovoj oblasti: (i) A (previše) veliki broj testova, (ii) kompromis između komponenata koji se mogu ponovo upotrebiti i betonskih proizvoda i (iii) tretmana varijabilnosti. Cmirev i ReiBig [32] su predstavili tehniku za smanjenje probnog slučaja za sisteme bogate varijantama, gde je izabran podskup proizvodnih varijanti dok garantuje potpunu pokrivenost zahteva. Ovaj pristup zasniva se na pohlepnoj strategiji selekcije i vezi između karakteristika proizvoda i njegovih zahteva. Selekcija pokušava da pronađe minimalan skup konfiguracije proizvoda dok se još uvek pokriva 100% obaveznih zahteva. Johansen je predstavio algoritam za generisanje t-mudrih pokrivnih nizova za velike funkcionalne modele [33]. Ovo omogućava testiranje kombinatornih interakcija linija softverskih proizvoda.

Briand [26] je dao pregled primena mašina za učenje u softverskim testovima. On definiše ulogu mašinskog učenja u testiranju softvera i kaže da može pomoći u rešavanju nekih dugotrajnih problema sa softverom. Jedan od predstavljenih pristupa je nazvan MELBA (MachrnE učenje zasnovano na poboljšanju test specifikacije crne kutije) koji je predstavljen kao pristup zasnovan na mašinskom učenju, koji pomaže u prečišćavanju testnih slučajeva u crnoj kutiji [34]. Koristi se inkrementalni proces koji je poluautomatski. Pristup pomaže razumevanje granica testnih slučajeva i pronalaženje viškova. Međutim, ne razmatra slučajeve testiranja u bilo kom redosledu ili daje prognoze o rezultatima testnih slučajeva.

10 Standardi u automobilskoj industriji

Uvođenjem novih tehnoloških inovacija u takozvane "pametne automobile", koje su namenjene podršci i savetovanju vozača, imaju za cilj da olakšaju i obezbede veću sigurnost i udobnost u vožnji [36]. Istovremeno, ovakve inovacije nose sa sobom i mogućnost rizika od pogrešnog rada, zbog kontrolisanja i interakcije ugrađenih softvera sa važnim sistemima kao što su kočnice i upravljački sistem. Kompanija Fiat Chrysler u maju 2017. godine imala je grešku u softveru koja je dovela do tragičnih saobraćajnih nesreća. Greška je prouzrokovala to da bočni vazdušni jastuci kao i pojas u slučaju naglog kočenja ne rade. Sličan problem je imala i kompanija General Motors koja je grešku otkrila tek nakon saobraćajne nesreće koja je ubila jednog i ranila tri čoveka.

Standardi funkcionalne bezbednosti (eng. Functional Safety Standard) postoje kako bi se izbegle nesreće ovog tipa. Pridržavanjem ovih standarda održava se siguran i pravilan rad vozila, a fokus je prvenstveno na rizicima koji poizilaze iz slučajnih hardverskih kvarova, sistemskih grešaka u dizajnu sistema, hardveru ili razvoju softvera, proizvodnji, pa sve do funkcionalne aplikacije. Dva standarda imaju važnu ulogu u razvoju softvera automobilske industrije, MISRA i ISO 26262, zajedno sa još jednim koji ih polako ali sigurno prati: AUTOSAR. Iako većina programera u automobilskoj industriji prepoznaje ova imena, nevoljno se pridržavaju ovih standarda nadajući se da će neko drugi brinuti o njima. Kako su ovi standardi obavezni kod većine automobilskih dobavljača, bitno je razumeti namenu koja stoji iza njih i to kako se analiza statičkog koda uklapa u testiranje krajnjeg proizvoda zarad ispunjavanja ovih standarda.

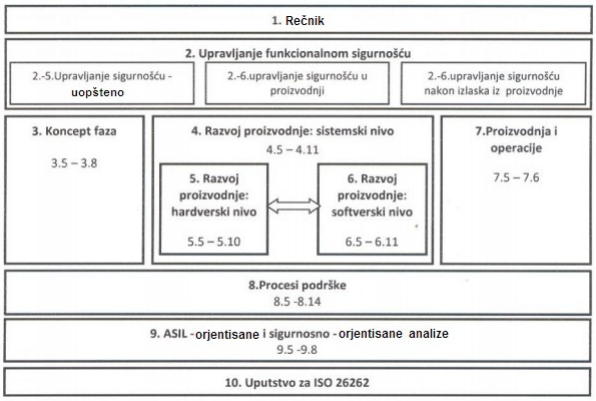
10.1 ISO 26262 standard

ISO (eng. International Organization for Standardization) je najveća svetska organizacija koja se bavi razvojem i izdavanjem internacionalnih standarda. ISO 26262 standard, poznat i pod nazivom Đrumska vozila - Funkcionalna bezbednost"(eng. Road vehicles - Functional safety) je rezultat potrebe za automobilsko specifičnim internacionalnim standardom koji bi se fokusirao na bezbednosno kritične komponente automobila. On je nastao kao adaptacija šire primenjivanog IEC 61508 standarda Međunarodne Elektrotehničke Komisije (eng. International Electrotechnical Commission - IEC), generičko funkcionalno bezbednosni standard za električne i elektronske sisteme, koji postavlja uslove za osiguravanje da su sistemi dizajnirani, implementirani, funkcionalni i napravljeni tako da obezbeđuju potreban nivo sigurnosti integriteta (SIL, eng. Safety integrity level).

Ovaj standard ima za cilj da smanji rizik na prihvatljiv nivo, pokrivajući sigurnost sistema u kompletnom životnom veku. Počiva na konceptu rizika, verovatnoći opasnih događaja i ozbiljnosti njihovih posledica kao i sigurnosnim funkcijama koje takav rizik smanjuju. Tri stava na kojima se zasniva standard IEC 61508 su: nulti rizik nikada ne može biti postignut, sigurnost se mora uzimati u obzir od samog početka i netolerisani rizici moraju biti reducirani [36]. ISO 26262 je standard funkcionalne bezbednosti koji se primenjuje na proizvodnju jednog ili više automobilskih električnih i elektronskih sistema koji se ugrađuju u putnička vozila do 3500 kilograma. Primeri ovakvih sistema

uključuju elektronske parkirne kočnice, napredni sistemi za pomoć vozačima (ADAS, eng. advanced driver assistance system), elektronski program stabilnosti, motorne kontrolne jedinice, prilagodiva kontrola brzine, itd.

Serija standard ISO 26262 sadrži 10 delova, 43 poglavlja, 180 inženjerskih metoda, 600 zahteva koji su sadržani u 450 stranica sa oko 750 rečenica [36]. Na slici 5 prikazana je struktura serije standarda ISO 26262 sa područjem primene svih delova od 1-10.



Slika 5: Struktura serije standarda ISO 26262 sa područjem primene njegovih delova

Kratak opis delova standarda [37]:

- ISO 26262-1 predstavljen na samom vrhu šematskog prikaza specificira pojmove, definicije i skraćenice za aplikaciju koje se koriste u svim ostalim delovima standarda
- ISO 26262-2 specificira zahteve za upravljanje funkcionalnom sigurnošću za automobilske aplikacije, uključujući zahteve nezavisne od projekta a koji se tiču organizacija uključene u projekat (sveobuhvatno upravljanje sigurnošću) i zahteve zavisne od projekta a koji se tiču aktivnosti upravljanja u sigurnosnom ciklusu (upravljanje tokom koncept faze i razvoja proizvoda, i nakon puštanja u rad za proizvodnju)
- ISO 26262-3 specificira zahteve za koncept fazu automobilskih aplikacija, uključujući: definiciju stavki (eng. item), iniciranje sigurnosnog ciklusa, analize opasnosti i procene rizika i koncept funkcionalne sigurnosti
- ISO 26262-4 specificira zahteve za razvoj proizvoda na sistemskom nivou, uključujući: uslovi za pokretanje razvoja proizvoda, specifikacija tehničkih sigurnosnih zahteva, tehnički koncept sigurnosti, dizajn sistema, integracija stavki i testiranje, sigurnosna provera, procena funkcionalne sigurnosti i puštanje proizvoda u rad

- ISO 26262-5 specificira zahteve za razvoj proizvoda na hardverskom nivou, uključujući: uslovi za pokretanje razvoja proizvoda na hardverskom nivou, specifikacija hardverskih sigurnosnih zahteva, dizajn hardvera, podaci hardverskog nivoa i procena kršenja bezbednosne granice usled slučajnih grešaka hardvera, njegova integracija i testiranje
- ISO 26262-6 specificira zahteve za razvoj proizvoda na softverskom nivou, uključujući: uslovi za pokretanje razvoja proizvoda na softverskom nivou, specifikacija softverskih sigurnosnih zahteva, dizajn softverske arhitekture, dizajn softverskih jedinica i njihovih implentacija, testiranje softverskih jedinica, softverska integracija i testiranje i provera softverskih sigurnosnih zahteva
- ISO 26262-7 specificira zahteve za proizvodnju, rad, usluge i razgradnju
- ISO 26262-8 specificira zahteve za prateće procese
- ISO 26262-9 specificira zahteve za ASIL (eng. Automotive Safety Integrity Level) orjentisanu i sigurnosno orjentisanu analizu
- ISO 26262-10 predstavlja sveobuhvatni prikaz ISO 26262 standarda, pruža dodatna obrazloženja u cilju razumevanja drugih delova ISO 26262

11 Zaključak i budući rad

U ovom radu dali smo pregled nekih izazova u testiranju softvera za automobile. Fokusirali smo se na scenarije testiranja crne kutije, pošto izvorni kod često nije dostupan u scenariju GEM-dobavljača u automobilskom domenu. Nakon opšteg procesa testiranja, predstavili smo šest pristupa za poboljšanje efikasnosti testiranja s obzirom na ograničenja razvoja softvera za automobile. Neki od predloženih koncepata su već ostvarljivi, na primer, sistematski opis slučaja testa. Drugi trebaju neke buduće radnje da bi bili od praktične upotrebe, npr., Pristupi u mašinskom učenju u testiranju u crnoj kutiji.

Literatura

[1] F. Saglietti, “Testing for dependable embedded software,” in 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2010, pp. 409–416.

[2] K. Grimm, “Software technology in an automotive company: major challenges,” in Proceedings of the 25th International Conference on Software Engineering. IEEE Computer Society, 2003, pp. 498–503.

[3] Umar Zakir Abdul, Hamid; et al. (2017). "Autonomous Emergency Braking System with Potential Field Risk Assessment for Frontal Collision Mitigation". 2017 IEEE Conference on Systems, Process and Control (ICSPC). Retrieved 14 March 2018.

[4] B. Katumba and E. Knauss, “Agile development in automotive software development: Challenges and opportunities,” in Product-Focused Software Process Improvement. Springer, 2014, pp. 33–47.

[5] F. Fabbrini, M. Fusani, G. Lami, and E. Sivera, “Software engineering in the European automotive industry: Achievements and challenges,” in 32nd Annual IEEE Computer Society International Conference on Computers, Software and Applications(COMPSAC), 2008, pp. 1039–1044.

[6] M. Broy, “Automotive software and systems engineering,” in Proceedings of the 3rd ACM and IEEE International Conference on Formal Methods and Models for Co-Design(MEMOCODE), 2005, pp. 143–149.

[7] M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann, “Engineering automotive software,” Proceedings of IEEE, vol. 95, no. 2, pp. 356–373, 2007.

[8] J. S. Her, S. W. Choi, J. S. Bae, S. D. Kim, and D. W. Cheun, “A component-based process for developing automotive ecu software,” in Product-Focused Software Process Improvement. Springer, 2007, pp. 358– 373.

[9] F. Franco, M. Mauro, S. Stevan, A. B. Lugli, and W. Torres, “Modelbased functional safety for the embedded software of automobile power window system,” in 11th IEEE/IAS International Conference on Industry Applications (INDUSCON), 2014, pp. 1–8.

[10] M. Conrad, "Verification and validation according to ISO 26262: A workflow to facilitate the development of high-integrity software"Proceedings to 6th European Congress on Embedded Real Time Software and Systems (ERTS2), 2012.

[11] S. S. Barhate, "Effective test strategy for testing automotive software"in International Conference on Industrial Instrumentation and Control (ICIC). IEEE, 2015, pp. 645–649.

[12] R. Awedikian and B. Yannou, "A practical model-based statistical approach for generating functional test cases: application in the automotive industry Software Testing, Verification and Reliability, vol. 24, no. 2, pp. 85–123, 2014.

[13] A. Kasoju, K. Petersen, and M. V. Mäntylä, “Analyzing an automotive testing process with evidence-based software engineering,” Information and Software Technology, vol. 55, no. 7, pp. 1237–1259, 2013.

[14] D. Sundmark, K. Petersen, and S. Larsson, "An exploratory case study of testing in an automotive electrical system release process"in 6th IEEE International Symposium on Industrial Embedded Systems (SIES). IEEE, 2011, pp. 166–175.

[15] A. M. Pérez and S. Kaiser, “Top-down reuse for multi-level testing,” in 17th IEEEInternationalConferenceandWorkshopsonEngineeringofComputer Based Systems (ECBS), 2010, pp. 150–159.

[16] J. Kasurinen, O. Taipale, and K. Smolander. Software Test Automation in Practice: Empirical Observations. Advances in Software Engineering, 2010: 571-579, 2010.

[17] S. Lity, R. Lachmann, M. Lochau, and I. Schaefer. Delta-oriented Software Product Line Test Models - The Body Comfort System Case Study. Technical report, TU Braunschweig, 2013.

[18] M. Lochau, S. Lity, R Lachmann, I. Schaefer, and U. Goltz. Delta-oriented model based integration testing of large-scale systems. The Journal of Systems and Software, 91: 63-84, 2014.

[19] L. Zhang, D.Han, L.Zhang, G.Rothermel, and H. Mei. Bridging the Gap between the Total and Additional Test-Case Prioritization Strategies. In International Conferenceon Software Engineering, ICSE 2013, 2013

[20] R. Lachmann and I. Schaefer. Herausforderungen beim Testen von Fahrerassistenzsystemen. In 11. Workshop Automotive Software Engineering (ASE), 2013.

[21] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing Test Cases For Regression Testing. IEEE Transactions on software engineering, Vol.27 No.10:929-948,2001.

[22] C. Kaner. What is a good test case? In Software Testing Analysis and Review Conference (STAR) East, 2003.

[23] M.Utting and B.Legeard. Practical Model-based Testing. Morgan Kaufmann, 2007.

[24] M. A. Sindbu and 1c. Meinkc. ms: An lncnemntal Lemming Algorithm for Finite Automata. CoRR, abs/1206.2691, 2012.

[25] H. Raffelt, B. Steffen, and T. Margaria. Dynamic Testing via Automata Learning. In Proceedings of the 3rd International Haifa Verification Conference on Hardware and Software: Verification and Testing, HVC'07, pages 136-152. Springer-Verlag, 2008

[26] L.C. Briand. Novel Applications of Machine Learning and Software Testing. In Quality Software, 2008 QSIC '08. The Eighth International Conference on, pages 3-10, Aug 2008.

[27] A R. Lenz, A. P020, and S. R. Vergilio. Linking software testing results with a machine learning approach. Engineering Applications of Artificial Intelligence, 266-6): 1631-1640, 2013.

[28] F. A. Barros, L. Neves, E. Hori, and D. Torres. The ucsCNL: A Controlled Natural Language for Use Case Specifications. In SEKE, pages 250-253. Knowledge Systems Institute Graduate School, 2011.

[29] J. Ferrer, P. M. Kruse, F. Chicano, and E. Alba. Evolutionary Algorithm for Prioritized Pairwise Test Data Generation. In Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO '12, pages 1213-1220. ACM, 2012.

[30] C.Strobbe, S.Herramhof, E.Vlachogiannis, and C.A. Velasco. Test Case Description Language (TCDL): Test Case Metadata for Conformance Evaluation. In ICCHP, pages 164-171, 2006.

[31] E. Engstrom and P. Runeson. Software product line testing - A systematic mapping study. Information and Software Technology, 53:2-13, 2011.

[32] A. Cmyrev and R. ReiBig. Optimierte Varianten- und Anforderungsabdeckung im Test. In Automotive Software Engineering Workshop. 43. GI Jahrestagung, 2013.

[33] M.F. Johansen, O.Haugen, and F. Fleurey. An algorithm for generating t-wise covering arrays from large feature models. In SPLC, pages 46-55, 2012.

[34] L.C. Briand, Y. Labiche, and Z. Bawar. Using the Machine Learning to Refine Black-Box Test Specifications and Test Suites. In Quality Software, 2008. QSIC'08. The Eighth International Conference, pages 135-144, Aug 2008.

[35] B. Engstrom, P. Runeson, and M. Skoglund. A systematic review on regression test selection techniques. Information and Software Technology, 52: 14-30, 2010.

[36] Sabira S. Salihović, Suada F. Dacić, Azra A. Ferizović, Funkcionalna sigurnost cestovnih vozila prema seriji standarda ISO 26262, 2015.

[37] International Organization for Standardization (ISO), ISO 26262-Part 1 - Part 9: Road vehicles - Functional safety, 2011.