Životni ciklus testiranja softvera

Pleše, Dario

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:761950

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

Image not found or type unknown

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski studij Informatike – jednopredmetni

Dario Pleše

Životni ciklus testiranja softvera

Završni rad

Mentor: izv. prof. dr. sc. Sanja Čandrlić

Rijeka, rujan, 2020.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 08.06.2020.

Zadatak za završni rad

Pristupnik: Dario Pleše

Naziv završnog rada: Životni ciklus testiranja softvera

Naziv završnog rada na eng. jeziku: Software Testing Life Cycle

Sadržaj zadatka:

Zadatak završnog rada je pojasniti proces testiranja softvera te predstaviti životni ciklus testiranja softvera kao dio razvojnog ciklusa testiranja softvera.

Mentor:

Izv. prof. dr. sc. Sanja Čandrlić

Voditelj za završne radove:

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 9.06.2020.

Dario Pleše

(Dario Pleše)

SADRŽAJ

1. SAŽETAK 3
2. UVOD 4
3. OSNOVE TESTIRANJA 5
3.1 PROPUST ILI ZATAJENJE (*FAULT OR FAILURE*) 5
3.2 SOFTVERSKA POUZDANOST 6
4. STLC KAO DIO SDLC 7
4.1 FAZE STLC – A 7
4.2 RAZINE TESTIRANJA 9
4.2.1 TESTIRANJE JEDINICA (UNIT TESTING) 9
4.2.2 INTEGRACIJSKO TESTIRANJE 9
4.2.3 SISTEMSKO TESTIRANJE 10
4.2.4 REGRESIJSKO TESTIRANJE 11
5. OGRANIČENJA SOFTVERSKOG TESTIRANJA 12
6. VAŽNOST TESTIRANJA 13
6.1 AUTONOMNA VOŽNJA AUTOMOBILA 13
6.2 TOYOTIN PROBLEM UBRZAVANJA 14
6.3 KONTROLA ZRAČNOG PROMETA LOS ANGELES 14
7. ZAKLJUČAK 16
8. LITERATURA 17

1. SAŽETAK

Testiranje softvera je vrlo važno područje sofverskog inženjerstva jer može služiti kao odlučujući faktor u plasiranju uspješno dovršenog projekta te direktnom odlučivanju razmjera kvalitete i pouzdanosti. Veliki dio istraživanja u ovom polju je već obavljen, no prema svemu sudeći, taj trend će se nastaviti i u budućnosti, s obzirom da je testiranje postalo sastavni dio svakog životnog ciklusa razvoja softvera. Testiranje softvera smatra se najvažnijom fazom razvoja jer bez faze testiranja ne bi bilo moguće ili bi bilo jako otežano otkriti moguće pogreške. S razvojem tehnologije, povećavaju se i metode i tehnike verifikacije prije odlaska proizvoda na tržište ili na produkciju. Uzevši u obzir sve veći broj kompleksnih sustava na kojima se svakodnevno radi, testiranja se razlikuju s obzirom na razinu sustava na kojoj se izvode. Kako bi testiranje bilo izvedeno i isplanirano pravilno, potrebno je poznavati ograničenja testiranja.

Ključne riječi: životni ciklus testiranja softvera, testiranje softvera, pouzdanost, softver, razvoj, testiranje.

2. UVOD

Računalni softver čini veliku komponentu informacijskog sustava čija je pouzdanost kritična za performanse i efikasnost organizacije sustava. Pouzdanost informacijskog sustava je vezana uz ljude, hardver, softver i podatke. Bez obzira na vrstu softvera o kojem se radi, pouzdanost softvera mora biti postignuta kroz kontinuirano planiranje, analizu, dizajn, programiranje, testiranje, instalaciju i održavanje samog softvera.

Razvoj softvera, iako relativno mlad u usporedbi s ostalim industrijama, prošao je dug put od vremena kada nisu postojale jasne metode i principi rada što je dovodilo do velikog broja projekata koji su propadali s obzirom da ljudi nisu mogli na vrijeme i unutar okvira dogovorenog budžeta razviti odgovarajući proizvod i prenijeti korisničke zamisli i potrebe u proizvod koji služi svojoj namjeni. Rezultat toga bili su softveri koji su bili nepouzdani i nerealno skupi za održavanje. Uz to, sve se više poduzeća odlučivalo na računalnu podršku u svom poslovanje što dovodi do prevelike potražnje u odnosu na mogućnosti industrije da zadovolji korisnike. Sve navedeno dovelo je do krize zkoja je znana pod nazivom “Softverska Kriza” (*Software Crisis*) [8] kasnih 60 - ih godina prošlog stoljeća. Rješenje je bilo transformiranje neorganiziranog programiranja u jasno posloženu i definiranu disciplinu čime su *de facto* udareni temelji softverskog inženjerstva kakvog danas poznajemo.

Tema završnog rada je Životni ciklus testiranja softvera. Odabrana je kako bi se podsjetilo na važnost testiranja proizvoda, aktivnosti koja se danas prilično često uzima zdravo za gotovo no predstavlja kamen temeljac za pouzdani proizvod konkurentan na tržištu.

3. OSNOVE TESTIRANJA

Testiranje je zahtjevna aktivnost koja uključuje nekoliko visoko zahtjevnih zadataka. U prvom planu je zadatak određivanja adekvatnog seta testova (*test cases*) po tehnici odabira testova koji se uklapaju u budžet. No, odabir testova je tek prvi zadatak u nizu jako bitnih zadataka s kojima se suočavaju testeri kao što su mogućnost pokretanja tih testova (u kontroliranom okruženju), odlučivanje jesu li rezultati prihvatljivi a ako nisu, otkrivanje posljedica neuspjeha te njihovog direktnog (pogreška) ili indirektnog uzroka, odlučivanje je li testiranje bilo dovoljno temeljito te može li se obustaviti što bi za uzvrat zahtijevalo izmjerenu efektivnost testova. Svaki od tih zadataka predstavlja prepreku i svojevrsan izazov za testere za čije će im rješavanje njihova stručnost biti najbitnija stavka.

3.1 PROPUST ILI ZATAJENJE (*FAULT OR FAILURE*)

Kako bi se u potpunosti razumjeli aspekti sofverskog testiranja, potrebno je razjasniti termine "pogreška" (*error*), "propust" (*fault*) i "zatajenje" (*failure*) [6]. Iako su njihova značenja strogo gledano povezana, među tim konceptima postoje jasne i važne razlike. Zatajenje je nemogućnost programa da izvede traženu funkciju odnosno sistemski kvar koji se događa zbog neispravnog unosa, nenormalni prekid (prislino zaustavljanje zbog nemogućnosti daljnjeg izvođenja) ili povrede postavljenih ograničenja. Uzrok zatajenja, primjerice dio koda koji je neispravan ili nedostaje nazivamo pogreškom. Propust može biti neotkriven dugo vremena, ovisno o veličini propusta i mjestu na kojem se nalazi, sve dok ga neki događaj ne aktivira. Kada se to dogodi, program se prvo dovodi u nestabilno stanje koje nazivamo *error* koji, kad se pretvori u izlaz, uzrokuje neuspjeh. Proces neuspjeha možemo sumirati na sljedeći način: Propust -> Pogreška -> Zatajenje koji se može manifestirati i rekurzivno u smislu da propust može biti uzrokovan zatajenjem nekog drugog dijela sustava. Međutim, pojam propusta je dvosmislen i težak za shvatiti jer ne postoji precizan kriterij po kojem možemo odrediti uzrok promatranog zatajenja. [3]

3.2 SOFTVERSKA POUZDANOST

Bez obzira na količinu i kvalitetu testova, nekoliko ili više pogrešaka će se ipak provući kroz testiranje. No pogreška može biti manje ili više zabrinjavajuća s obzirom na to koliko se često pojavljuje krajnjem korisniku te koliko su ozbiljne njezine posljedice. U skladu s tim, jedina važna mjera u odlučivanju je li softverski proizvod spreman za plasiranje na tržište je upravo njegova pouzdanost. Strogo gledano, softverska pouzdanost je probabilistička procjena te mjeri vjerovatnost da će softver raditi bez neuspjeha u danom okruženju za određeno vremensko razdoblje. Stoga vrijednost softverske pouzdanosti ovisi o tome koliko često će krajnji korisnik unositi ulaze koji uzrokuju neuspjeh. Procjene softverske pouzdanosti mogu biti napravaljene pomoću testiranja. Zato, s obzirom da je pojam pouzdanosti specifičan za dano okruženje, testovi moraju biti izvučeni iz skupa koji je što sličniji mogućim ulazima koji će se koristiti kada softver jednom uđe u produkciju. Taj proces zove se operativna distribucija [3].

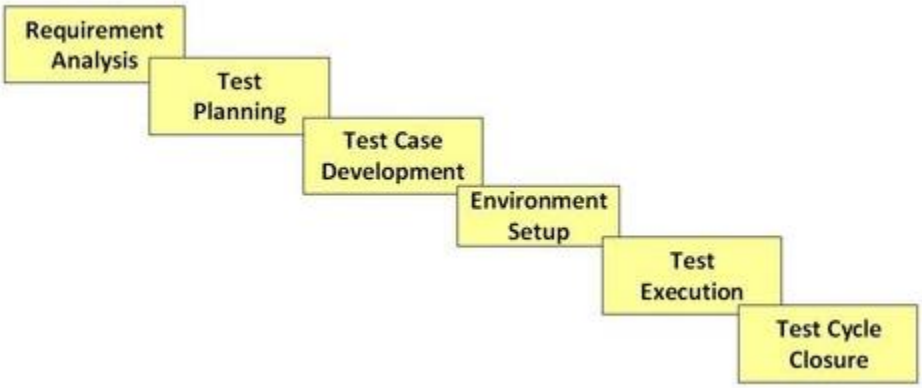
4. STLC KAO DIO SDLC

Životni ciklus razvoja softvera (*Software development life cycle*, SDLC) je proces planiranja, izrade, testiranja te isporuke proizvoda [8]. Njegov glavni cilj je smanjenje cijene uz maksimalno povećavanje kvalitete u što je mogućem kraćem roku. SDLC to uspijeva tako što prati plan koji uklanja probleme u koje inače zapadaju projekti razvoja softvera. Također je važno naglasiti kako je ovdje snažan fokus stavljen na životni ciklus testiranja softvera. Važnost testiranja unutar SDLC-a je da poveća pouzdanost, performance te ostale faktore su definirani u softverskim zahtjevima.

4.1 FAZE STLC – A

Životni ciklus testiranja softvera (STLC) je razlomljen u nekoliko faza koje sofver mora proći kako bi bio označen kao potpuno testiran proizvod. U prvoj fazi, analizi zahtjeva, odvija se pregled korisničkih zahtjeva od strane tima zaduženog za provjeru kvalitete u kojem se želi utvrditi koji se točno zahjevi trebaju testirati. U slučaju pojave problema u smislu da nešto nije dovoljno jasno ili nešto nedostaje, tim se mora posavjetovati i koordinirati s razvojnim timom koji je radio na tome kako bi se problem rješio. Druga i najbitnija faza životnog ciklusa testiranja softvera je planiranje testiranja iz razloga što se u ovoj fazi definiraju sve strategije te planovi testiranja. Voditelj tima zaduženog za testiranje kalkulira omjer procijenjenog troška i truda koji će biti potrebno uložiti kako bi se testiranje uspješno privelo kraju. U ovoj fazi priprema se plan testiranja koji je obavezan dokument bez kojeg proces testiranja ne bi bio moguć. Ova faza ne može krenuti bez da je prethodna faza, analiza zahtjeva, završena. Sljedeća, treća faza započinje tek nakon što se finalizira faza planiranje testiranja. To je faza razvoja testova u kojoj se razvijaju testovi te faza u kojoj aktivnost planiranje testiranja prestaje. Testovi se pišu od strane tima za osiguranje kvalitete ili se automatski generiraju. Test specificira skup ulaznih podataka, uvjeta te očekivanih rezultata. Određeni skup podataka bi trebao biti izabran tako da proizvodi očekivane rezultate, ali i sadržavati namjerno pogrešne podatke koji će uzrokovati problem tijekom testiranja. Na taj način se provjerava koje uvjete softver prestaje izvoditi. Kada završi faza razvoja

testova, započinje faza postavljanja testnog okruženja. To je izuzetno bitna faza životnog ciklusa testiranja softvera. U suštini, testno okruženje odlučuje softverske uvjete pod kojim se proizvod testira. Tim zadužen za testiranje nije uključen u postavljanje samog okruženja, već to radi ili razvojni tim ili klijent. No, nakon toga, tim koji provodi testiranje mora napraviti provjeru spremnosti (*smoke test*) danog testnog okruženja. Nakon toga slijedi faza izvođenja testa koju provode testeri na temelju plana testiranja te pripremljenih testova. Proces se sastoji od izvršavanja te održavanja testnih skripti te izvještavanja o problemima. Ukoliko je neki problem pronađen te prijavljen, razvojni tim mora riješiti problem te je potrebno napraviti ponovno testiranje kako bi se utvrdilo da je problem uistinu riješen i da rješenje ovog problema nije uzrokovalo neke nove probleme. Posljednja, no ne i manje važna faza životnog ciklusa testiranja softvera je zatvaranje ciklusa testiranja. To je faza završetka izvršavanja testova koja uključuje nekoliko aktivnosti, između ostalih i izvještavanje o završetku testiranja te prikupljanje rezultata testiranja. Članovi tima koji je proveo testiranje analiziraju provedene postupke te strategije koje bi trebale biti implementirane u buduće životne cikluse kako bi se izbjegla moguća usporavanja (*bottlenecks*), imajući na umu usvojeno znanje iz prijašnjih testiranja [4], [7].



Slika1 Schema životnog ciklusa testiranja softvera[7]

4.2 RAZINE TESTIRANJA

Tijekom razvojnog ciklusa softvera, testiranje se provodi na različitim razinama i može uključivati cijele sustave ili samo dijelove sustava. Bez obzira na vrstu modela koji se koristi, moguće je razlikovati testiranje jedinica (unit testing), integracijsko testiranje (integration testing) te sistemsko testiranje. Ovo su 3 faze testiranja tradicionalnog pristupa podijeljenog u faze (primjer toga bio bi vodopadni model) [3]. Međutim, razmatrajući različite, modernije modele, razlika između ove 3 testne razine ostaje korisna kako bi se naglasila tri različita trenutka u verifikaciji kompleksnog sustava. Važno je naglasiti da niti jedna od ove tri razine nije važnija od druge dvije.

4.2.1 TESTIRANJE JEDINICA (UNIT TESTING)

Jedinica, u ovom konteksu, predstavlja mali komadić softvera koji se može testirati te se može sastojati od nekoliko linija koda pa sve do nekoliko stotina ili tisuća linija koda te, u globalu, predstavlja rad jednog programera. Svrha ove razine je osigurati da jedinica zadovoljava njezine funkcionalne specifikacije i / ili da implementirana struktura odgovara predviđenoj strukturi.
Testiranje jedinica se također može primijeniti kako bi se provjerila sučelja (ako su parametri proslijeđeni u pravom redoslijedu, ako je broj parametara jednak broju argumenata), struktura podataka (pogreške u imenima varijabli, nedosljednost tipova podataka), itd. [3].

4.2.2 INTEGRACIJSKO TESTIRANJE

Općenito govoreći, integracija je proces u kojem se dijelovi ili komponente softvera agregiraju, odnosno spajaju u veću cijelinu. Integracijsko testiranje cilja otkrivanje problema koji se mogu pojaviti u ovoj fazi. Iako jedinice, kada se testiraju pojedinačno, zadovoljavaju testove, kada su dio jedne veće cjeline one bi i dalje mogle rezultirati pogrešnim ili nekonzistentnim ponašanjem. Stoga integracijsko testiranje cilja verifikaciju da svaka komponenta komunicira u skladu sa

specifikacijama koje su definirane u ranijim fazama razvoja softvera. U literaturi nema mnogo formaliziranih pristupa integracijskom testiranju pa se stoga u praksi najviše oslanja na osjećaj za dobar dizajn i intuiciju testera. Kod tradicionalnih sustava koristi se ili inkrementalni ili neinkrementalni pristup. Kod neinkrementalnog pristupa komponente su povezane te se testiraju sve odjednom (*big bang testing*). Kod inkrementalnog pristupa koristi se *top-down* strategija u kojem se moduli integriraju jedan po jedan, počevši od glavnoga pa sve do njemu podređenih, ili *bottom-up* strategija u kojoj se testovi kreiraju od modula na najnižoj razini hijerarhije te postepeno prema vrhu. U praksi se najboljim pokazao hibridni pristup u kojem se kombiniraju obje strategije, no sve ovisi o raznim vanjskim faktorima poput dostupnosti testera, dostupnosti modula i dr. [3].

4.2.3 SISTEMSKO TESTIRANJE

Sistemsko testiranje uključuje cijeli sistem ugrađen u hardver te se cilja na verifikaciju da se sustav ponaša u skladu sa korisničkim zahtjevima. Detaljanije to bi značilo kako se želi otkriti probleme koji ne mogu biti pripisani pojedinim komponentama, nekonzistencijama među komponentama te ostalim objektima (koji su subjekt integracijskog testiranja). Glavne ciljeve sistemskog testiranja možemo sažeti na tri primarna: otkrivanje problema koji se manifestiraju samo na sistemskoj razini te ih stoga nije bilo moguće otkriti tijekom testiranja jedinica ili integracijskog testiranja, povećavanje samopouzdanja da proizvod uspješno i pravilno implementira tražene sposobnosti te skupljanje informacija koje mogu biti korisne u odlučivanju o plasiranju proizvoda na tržište.

Sistemsko testiranje bi stoga trebalo osigurati da svaka funckija sustava radi kao što je predviđeno te da je svaka pogreška otkrivena i analizirana. Sistemsko testiranje omogućava dostupnost informacija o stanju razvoja koje druge metode verifikacije ne mogu omogućiti. Općenito, sistemsko testiranje uključuje testiranje performansi, sigurnosti, pouzdanosti, otpornosti na stres te oporavljanje te se često dodaje i nova razina testiranja, testiranje prihvaćanja. No testiranje prihvaćanja je više proširenje sistemskog testiranja nego zapravo prava razina testiranja[3].Test

prihvaćanja možemo okarakterizirati kao testnu sesiju provedenu kroz cijeli sustav koja se najviše fokusira na zahtjeve za uporabljivost (*usability requirements*). Namjera je zapravo verificirati da je trud koji se zahtijeva od krajnjih korisnika kako bi naučili koristiti i iskoristiti funkcionalosti sustava u potpunosti prihvatljiv [3].

4.2.4 REGRESIJSKO TESTIRANJE

Regresijsko testiranje zapravo nije zasebna razina testiranja, već se može poistovjetiti s testiranjem samo jedne jedinice, kombinacije komponenti ili cijelog sustava nakon učinjenih određenih promjena kako bi se potvrdilo da one nisu dovele do novih problema. S obzirom da su softveri u konstantnoj evoluciji, vođeni od strane tržišta te napretka u tehnologiji, regresijsko testiranje uzima premoćan dio cjelokupnog testiranja u industriji. S obzirom da se modifikacije izvode učestalo, ponovno pokretanje svih prijašnjih testova bilo bi veoma skupo. Stoga su razvijene razne tehnike kojima se želi smanjiti cijena regresijskog testiranja te učiniti ga efikasnijim.

Selektivno regresijsko testiranje je tehnika koja pomaže u selekciji podskupova već postojećih testova tako što se proučavaju modifikacije. Drugi pristupi prioritetiziraju testove prema određenom kriteriju (primjerice maksimiziranje moći otkrivanja pogrešaka) kako bi se testovi, koji su usvojeni prema određenom kriteriju izveli prvi, unutar granice budžeta [3].

5. OGRANIČENJA SOFTVERSKOG TESTIRANJA

Kako bi u potpunosti razumjeli sam proces testiranja, potrebno je poznavati i njegova ograničenja. Ne treba smetnuti s uma kako čak niti najrigorozniji testovi zapravo nisu garancija da će se konačni proizvod moći koristiti kada ga se plasira na tržište. Autori Sandeep Dalal i Rajender Singh Chhilar [5] navode najbitnija ogarničenja:

- Predefinirano vrijeme za testiranje nije dodijeljeno kada faza testiranja započne, odnosno u tradicionalnim pristupima, u većini slučajeva, datum isporuke je određen i objavljen prije početka samog projekta te u tom procesu najviše vremena dobivaju razvojni programeri što ostavlja manje vremena za testiranje što u konačnici može dovesti do propadanja projekta zbog nedovoljnog testiranja ili izbacivanje proizvoda na tržište može kasniti što za posljedicu povećava cijenu proizvoda
- Obavljanje 100% testiranja nije moguće kod kompleksnih sustava – u slučaju kompleksnih sustava koji uključuju mnogo komunikacije s aplikacijama postoji mogućnost da se testiranje ne odradi dovoljno temeljito
- Nedostatak formalnog testiranja i pregleda u fazama specifikacije zahtjeva i dizajna
- Omjer razvojnih programera u odnosu na testera je 1:3 što povećava cijenu testiranja kompletnog proizvoda
- Kompletno testiranje je neisplativo. Složenost jest korijen problema. U određenom trenutku testiranje mora biti zaustavljeno te proizvod mora biti plasiran na tržište
- Testiranje se može iskoristiti kako bi se otkrili problem no ne može se i dokazati da problema nema. Testiranjem ne možemo potvrditi da je sustav bez problema ili propusta
- S obzirom da u današnje vrijeme svaka aplikacija mora biti razvijena I za mobitele, tablete i ostale sustave, njihova kompatibilnost s različitim hardverom i za sve preglednike uvelike otežava stabilnost proizvoda u svim okruženjima te tako i cjelovito testiranje

6. VAŽNOST TESTIRANJA

U današnje moderno vrijeme i vrijeme kada tehnologiija diktira svaki naš korak, ne možemo niti zamisliti dio dana bez nje, a još manje i cijeli život bez nje. Ona je snažno isprepletena s našom svakodnevnicom te se sve više stvari zamjenjuje sa nekom vrstom softvera. No bez obzira na to koliko tehnologija pokuša pojednostaviti našu egzistenciju, ona je i dalje plod ljudske kreacije te je kao takva podložna ljudskim pogreškama. To je posebno slučaj u situacijama oko kojih se vrti veliki novac te u tim situacijama velike tvrtke možda ne žele uložiti još novca u ispravljanje problema ukoliko malo ili nimalo korisnika zna to njih. Tehnologija u isto vrijeme može biti vrlo zabavna ali i vrlo opasna i zastrašujuća no upravo faza testiranja, koja često biva zanemarena, ima mogućnost smanjiti taj osjećaj opasnosti na minimum. S obzirom da testiranje može biti skupo ono se često uzima zdravo za gotovo te se često zanemaruje, smatrajući kako su osnovne funkcionalnositi pokrivene te da korisnici neće primjetiti neke nedostatke. No testiranje u nekim situacijama može doslovno značiti razliku između života i smrti. U nastavku se navodi nekoliko primjera takvih situacija.

6.1 AUTONOMNA VOŽNJA AUTOMOBILA

U ožujku 2018. automobil prijevozne tvrtke Uber testiran je koristeći autonomni oblik vožnje (oblik vožnje u kojem softver upravlja automobilom bez asistencije čovjeka). Tijekom jednog od testova u državi Arizoni, “softverski vozač” nije stigao na vrijeme reagirati te je udario jednog pješaka nanijevši mu smrtonose posljedice. Ta tragedija potresla je cijeli svijet i automobilsku industriju te stavila veliki upitnik iznad budućnosti autonomne vožnje. Pješak je prelazio cestu sa svojim biciklom izašavši iz mraka te se naglo pojavio pred svijetlima automobila koji nije na vrijeme zakočio/reagirao. Do ovoga je dovelo loše dizajniranje te osmišljavanje testova koji nisu pokrili sve slučajeve koji se mogu dogoditi na cesti jer bi se u suprotnom ova situacija mogla izbjeći te sačuvati život. Nakon ovog incidenta Uber više nije zatražio dozvolu za testiranje autonomne vožnje u Arizoni s obzirom na veliko nezadovoljstvo zajednice koja ne odobrava ovakva testiranja na njihovim cestama sve dok se sigurnost drastično ne poveća. No bez obzira

na to, veliki dio autoindustrije i dalje smatra da je budućnost vožnje automobila u autonomnoj vožnji te nastavljaju s testiranjem iste [2].

6.2 TOYOTIN PROBLEM UBRZAVANJA

Sredinom 2000–ih mnogo vozača automobila Toyota prijavilo je situaciju u kojoj njihov automobil počne ubrzavati bez da vozač dotakne papučicu ubrzavanja. Nakon serije nesreća, odlučeno je pokrenuti službenu istragu te su istražitelji otkrili kako su sve nesreće uzrokovane greškom u softveru. No u ovom slučaju nije "zaslužan" jedan već cijeli niz problema: oštećenje memorije uzrokovane lošim rukovanjem, onemogućavanje sigurnosnih sustava, tisuće globalnih varijabli te sustavi s pojedinačnom točkom pucanja (ukoliko jedna komponenta doživi neuspjeh, sustav se gasi). Nakon završene istrage te utvrđivanja da je Toyotin softver krivac za mnoge nesreće, uprava je bila prisiljena povući nekoliko milijuna vozila te se vrijednost Toyotinih dionica strmoglavila i pala za 20–ak %. U periodu nakon nemilih događaja Toyota se našla na udaru javnosti te se vrijednosti koju je Toyota izgubila uslijed tih događaja broji u stotinama milijuna dolara. Ovaj dokaz demonstrira posljedice nepridavanja dovoljno pažnje dobrom programiranju te dobrom i temeljitom testiranju u želji da se proizvod što prije izbaci na tržište [1].

6.3 KONTROLA ZRAČNOG PROMETA LOS ANGELES

Kontrola zračnog prometa je izuzetno važna stavka svake zrakoplovne luke te nosi veliku odgovornost informiranja pilota o relevantnim podacima o vremenu, udaljenosti od drugih zrakoplova, rutama i još mnogim drugim podacima. Ukoliko zakaže na nekoj od prethodno navedenih zadaća, može rezultirati katastrofom. 14 rujna 2004. kontrola tračnog prometa u zračnoj luci u LA – u izgubila je mogućnost komunikacije sa 400-tinjak zrakoplova koji su se u tom trenutku nalazili u jugozapadnom SAD–u te su se mnogi od njih nalazili na istim putanjama krećući se jedan prema drugome. Primarni sustav za verbalnu komunikaciju se iznenada isključio. No nesreća nikad ne dolazi sama pa je povrh toga i rezervni sustav otkazao nakon tek nekoliko

minuta rada. Uzrok je bio u činjenici što je komunikacijski sustav imao interni brojač koji je odbrojavao u milisekundama. Kad je odbrojeno do nule, nije se mogao nazad postaviti na traženo vrijeme stoga se je isključio. Ovaj prekid rada utjecao je na više od 800 letova diljem države te uzrokovao stotine milijuna dolara štete [1].

7. ZAKLJUČAK

Primarna zadaća testiranja softvera nije pokazati ispravnost konačnog proizvoda već otkriti skrivene probleme kako bi oni mogli biti ispravljeni. Ono također ima potencijal uštedjeti vrijeme i novac tako što može otkriti probleme u ranoj fazi razvoja kada rješenje još nije prekomplicirano te omogućava dostavu klijentu proizvoda koji nema/ima manje problema. Bez propisnog testiranja postoji veliki rizik da softver doživi neuspjeh. Stoga kako bi se testiranje provelo temeljito i efektivno, svi koji uključeni u razvoj trebali bi biti upoznati s osnovama softverskog testiranja.

Postoje mnoge aktivnosti tijekom testiranja koje čine životni ciklus testiranja softvera. One su od vitalne važnosti za svaki sustav s obzirom da njihova primjena može dramatično smanjiti rizik puštanja proizvoda s pogreškama u opticaj te tako osigurati kvalitetu samog softvera te njegovu dostavu na vrijeme. U pravilu ove testne aktivnosti traju između 30% te 50% cjelokupnog vremena koje je alocirano za izradu projekta. Stvarno dodjeljivanje vremena potrebnog za testiranje ovisi o razini rizika softvera kojeg se izrađuje. U razvoju softvera koji koristi osjetljive podatke koji ne bi smjeli pasti u krive ruke, testiranju i rješavanju problema bi trebalo biti dodjeljeno i više vremena.

Bez obzira na testne aktivnosti, testiranje softvera može samo pomoći u osiguravanju kvalitete proizvoda, a ne ugraditi kvalitetu u proizvod. Kako bi izgradili kvalitetu, treba se pažljivo analizirati zahtjeve nakon čega slijedi efektivni dizajn sustava, a ne oslanjati se samo na testiranje kako bi otkrili i “uhvatili” (moguće) probleme te otklonili iste koji potječu s početka projekta.

8. LITERATURA

1. Bits and Pieces portal, 2018. https://blog.bitsrc.io/software-is-not-perfect-cases-of-software-failure-and-their-consequences-f5fec39c038f, 10.07.2020.

2. Medium portal, 2019. https://blog.checkio.org/%EF%B8%8F-top-5-software-failures-of-2018-2019-5-is-pretty-alarming-2a5400b01658, 10.07.2020.

3. Bertolino A., Marchetti E., 'A Brief Essay on Software Testing', 1-14

4. Arif M., Abid Jamil M., Awang Abubakar N. S., Ahmad A., 'Software testing Techniques: A Literature Review', 2016., 6th *International Conference on Information and Communication Technology for The Muslim World*, 177 - 182

5. Dalal S., Chhilar R. S., 'Software Testing – Three P'S Paradigm and Limitations', 2012., *International Journal of Computer Applications,* volume 54, broj 12, 49 - 54

6. Veenedaal E. v., 2007., *Standard glossary of terms used in Software Testing,* Lokalizirano izdanje, verzija 2.0

7. Guru99 https://www.guru99.com/software-testing-life-cycle.html, 30.08.2020.

8. UKESSAYS https://www.ukessays.com/essays/computer-science/software-crisis.php, 30.08.2020.

9. Wikipedija https://en.wikipedia.org/wiki/Systems_development_life_cycle#Testing, 30.08.2020.