Testiranje programskih proizvoda na primjeru poduzeća Tech Resources d.o.o.

Vidaković, Matea

SVEUČILIŠTE U SPLITU

EKONOMSKI FAKULTET

ZAVRŠNI RAD

Testiranje programskih proizvoda na primjeru poduzeća Tech Resources d.o.o.

Mentor:

prof. dr. sc. Mario Jadrić

Student:

Matea Vidaković

Split, srpanj 2023.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, MATEA VIDAKOVIĆ,
(ime i prezime)

izjavljujem i svojim potpisom potvrđujem da je navedeni rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja na objavljenu literaturu, što pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio navedenog rada nije napisan na nedozvoljeni način te da nijedan dio rada ne krši autorska prava. Izjavljujem, također, da nijedan dio rada nije korišten za bilo koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Split, 2023. godine

Vlastoručni potpis : Matea Vidaković

Sadržaj

1. UVOD

1.1. PROBLEM I PREDMET ISTRAŽIVANJA

Testiranje programskih proizvoda je ključni proces u razvoju softvera. Ono ima iznimnu važnost kako za klijente tako i za tvrtku koja razvija i pruža softverske proizvode. Za klijente, testiranje osigurava visoku kvalitetu i pouzdanost softvera koji koriste. Kroz detaljno testiranje, identificiraju se i otklanjaju greške prije nego što softver dođe u ruke korisnika. Rezultat toga je poboljšano korisničko iskustvo. Također, tvrtka može identificirati i ispraviti greške u ranoj fazi razvoja, što smanjuje troškove i vrijeme potrebno za naknadno ispravljanje grešaka. Kvalitetan softver poboljšava ugled tvrtke, gradi povjerenje kod klijenata i može rezultirati povećanjem prodaje i rasta poslovanja.

Ova važna tematika detaljno će se obraditi u radu, te također istražiti će se tipovi i primjeri testiranja programskih proizvoda. Navedeni primjeri su projekti tvrtke Tech Resources te odlično prikazuju proces testiranja na realnim problemima.

1.2. CILJ ISTRAŽIVANJA

Cilj ovog rada je istražiti i naglasiti važnost testiranja u razvoju softverskog proizvoda, s posebnim fokusom na ručno i automatsko testiranje. U radu će se teorijski opisati različiti načini i tipovi testiranja koji se primjenjuju u razvoju softvera, kao što su jedinično testiranje, integracijsko testiranje, dinamičko testiranje, sistemsko testiranje i ostali. Također, bit će pruženi praktični primjeri testiranja u razvoju web aplikacija, s naglaskom na popularne metode i alate koji se koriste u tom području. Kroz analizu teorijskih koncepta i primjene u stvarnim scenarijima, rad će ilustrirati koliko je testiranje ključno za osiguravanje kvalitete, funkcionalnosti i pouzdanosti web aplikacija te kako doprinosi uspješnom razvoju i isporuci kvalitetnih softverskih proizvoda.

1.3. METODE ISTRAŽIVANJA

Metode korištene u radu:

Induktivno-deduktivna metoda: Induktivno-deduktivna metoda kombinira procese indukcije (izvođenje općih zaključaka na temelju specifičnih primjera) i dedukcije (izvođenje specifičnih

zaključaka iz općih postavki). To omogućuje da se na temelju konkretnih primjera dolaze do općih zaključaka, koristeći pritom logičko razmišljanje.

Metoda deskripcije: Metoda deskripcije se koristi za jednostavno opisivanje činjenica ili karakteristika određenog fenomena i pojmova.

Metoda analize i sinteze: Metoda analize i sinteze se koristi za proučavanje složenih koncepata. Analiza uključuje razdvajanje tih objekata na jednostavnije dijelove kako bi se detaljnije istražili. Sinteza uključuje ponovno spajanje tih dijelova kako bi se razumjela njihova međusobna povezanost i funkcioniranje kao cjelina.

Metoda komparacije: Metoda komparacije uključuje uspoređivanje različitih pojava, stvari ili pravila kako bi se identificirale sličnosti i zajednička obilježja među njima.

Metoda kompilacije: Metoda kompilacije uključuje prikupljanje i preuzimanje rezultata, opažanja i zaključaka iz različitih izvora (znanstveni radovi, istraživanja ili knjige) te njihovu sintezu u smislen i koristan pregled.

Studija slučaja: Studija slučaja je metoda istraživanja koja se usredotočuje na detaljno proučavanje pojedinačnog slučaja kako bi se steklo dublje razumijevanje specifičnog procesa.

1.4. STRUKTURA ZAVRŠNOG RADA

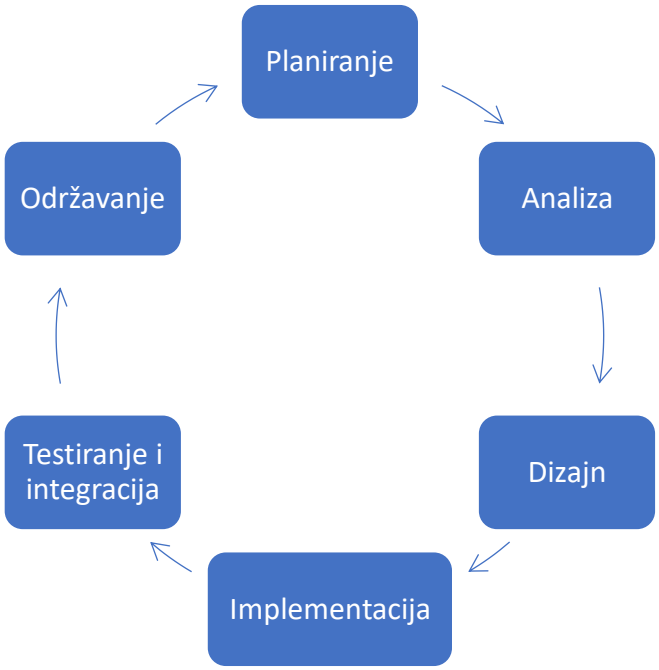
Rad je podijeljen na dvije sadržajne cjeline. U teorijskom dijelu obrađuju se osnove testiranja kao uvod u temu testiranja programskih proizvoda. Opisuju se ciljevi i načela testiranja, ciklus razvoja softvera te najvažniji tipovi testiranja. U praktičnom dijelu prikazuju se primjeri testova ručnog i automatskog testiranja programskog proizvoda, njihove dobre i loše strane karakteristike i zaključak u kojem se navodi kada je primjereno koristiti određeno testiranje i koje je prikladnije u različitim situacijama.

2. OSNOVE TESTIRANJA PROGRAMSKIH PROIZVODA

2.1. ŽIVOTNI CIKLUS RAZVOJA SOFTVERA

Prema Ranjan, R. (2019.) životni ciklus razvoja softvera (SDLC) je strukturirani pristup razvoju softvera koji uključuje niz faza, uključujući analizu softverskih zahtijeva, dizajn softverske arhitekture, implementaciju softvera, testiranje softvera i održavanje softvera. SDLC proces pomaže osigurati da se softver razvija učinkovito, da ispunjava zahtjeve korisnika i da se isporučuje na vrijeme i unutar proračuna. Trenutačno se koriste dvije metodologije životnog ciklusa razvoja softvera (SDLC), to su tradicionalni razvoj i agilni razvoj. Faze tradicionalne metodologije ovog životnog ciklusa prikazane su na slici.

Slika 2.1. Faze životnog ciklusa u razvoju programskog proizvoda



Izvor: Istraživanje autora

Generiranje ideja/pokretanje projekta: U ovoj fazi analiziraju se ideje kako bi se pronašlo rješenje za određeni problem s kojim se susreću mnogi korisnici.

Analiza zahtijeva: Ovo je najvažnija faza životnog ciklusa razvoja softvera (SDLC), u kojoj se prikupljaju sve vrste dokumenata o zahtjevima za bilo koji projekt nakon komunikacije s korisnicima.

Dizajn: U ovoj fazi se izrađuje visoko i nisko razinski dizajn softvera koji će se razvijati.

Implementacija/razvoj: U ovoj fazi se vrši stvarno kodiranje softvera.

Testiranje: Ova faza uključuje testiranje softvera kako bi se osiguralo da ispunjava zahtjeve i da je bez grešaka.

Implementacija: Nakon što je softver testiran i odobren, implementira se u produkcijsko okruženje.

Održavanje: Ova faza uključuje održavanje softvera kako bi se osiguralo da i dalje ispunjava potrebe korisnika i da ostaje ažuran s promjenjivim zahtjevima.

S druge strane, Yu i sur. (2012.) tvrde da se agilni razvoj temelji se na inkrementalnom i iterativnom pristupu, gdje se faze životnog ciklusa razvoja neprestano ponavljaju radi poboljšanja softvera kroz povratne informacije primljene od korisnika. U agilnom razvoju, umjesto jednog velikog procesnog modela kao u konvencionalnom SDLC-u, životni ciklus podijeljen je na manje dijelove nazvane "inkrementi" ili "iteracije", svaki uključuje tradicionalne faze razvoja. Prema Agilnom manifestu, ključni čimbenici agilnog pristupa uključuju:

1. Rani uključivanje korisnika
2. Iterativni razvoj
3. Samoorganizirajući timovi
4. Prilagodba promjenama

Trenutno postoji šest prepoznatih metoda agilnog razvoja: Crystal metodologije, dinamička metoda razvoja softvera, razvoj vođen svojstvima, lean razvoj softvera, scrum i ekstremno programiranje.

2.2. ŽIVOTNI CIKLUS RAZVOJA TESTIRANJA

Testiranje kao ključna faza ovog ciklusa zauzima poseban položaj. Postoje tvrdnje da bi se testiranje trebalo odvijati vrlo rano u životnom ciklusu razvoja softvera, po mogućnosti i prije programiranja. Testiranje softvera može se uključiti u životni ciklus razvoja softvera (SDLC) na nekoliko načina kako bi se osigurala maksimalna učinkovitost. Navaei & Tabrizi (2021.) tvrda da jedan od uobičajenih pristup je izvršavanje testiranja paralelno s procesom razvoja, što omogućuje rano otkrivanje i rješavanje nedostataka. Drugi pristup je korištenje automatiziranih alata za testiranje radi pojednostavljivanja procesa testiranja i smanjenja vremena i napora potrebnog za ručno testiranje. Osim ovih pristupa, testiranje se može integrirati u svaku fazu SDLC-a. Potražnja za softverskim aplikacijama podigla je standard kvalitete razvijenog softvera te uloga testiranje postaje još značajnijom.

Stalne promjene u kodu sustava mogu rezultirati pojavom pogrešaka, kvarova i defekata u ponašanju sustava. Kako bi se otkrile te pogreške prije implementacije sustava, važno je provesti testiranje. Potrebno je napraviti izbor između ručnog i automatskog testiranja. Pronalaženje nedostataka u

infrastrukturi sustava korištenjem automatskog testiranja može rezultirati značajnim smanjenjem troškova, čak do trećine ukupnih troškova.

Kako bismo stvorili razumna očekivanja o prednostima testiranja softvera, važno je prepoznati neka od njegovih ograničenja. Testiranje softvera ima sljedeće nedostatke prema Quadri & Farooq (2010.) iako je najpopularnija metoda provjere:

1. Testiranje se može koristiti za isticanje prisutnosti grešaka, ali nikada ne može pokazati njihovu odsutnost! Ograničen je na otkrivanje poznatih problema ili grešaka. Testiranjem se ne može jamčiti nepostojanje grešaka u sustavu koji se testira.

2. Testiranje je beskorisno kada se odlučuje hoće li se "kasno objaviti proizvod bez bugova zbog kršenja roka" ili "objaviti proizvod s pogreškama radi ispunjavanja roka".

3. Testiranje može samo pokazati da su kôd, funkcionalnost i zahtjevi pokriveni do određene mjere; ne može dokazati da proizvod radi ispravno u svim okolnostima.

Životni ciklus testiranja softvera (STLC) je proces u kojem se izvodi sedam temeljnih koraka u svrhu testiranja. Koraci u životnom ciklusu testiranja softvera uključuju analizu zahtijeva, planiranje testiranja, razvoj testnih slučajeva, postavljanje okruženja, izvršavanje testova, te na kraju, nakon završetka procjene, izvješćivanje i zatvaranje aktivnosti testiranja. Ovi koraci prikazani su na slijedećoj slici.

Slika 2.2. Faze životnog ciklusa u testiranju programskog proizvoda

Izvor: Istraživanje autora

Prva faza je analiza zahtijeva, u kojoj tim za testiranje prikuplja informacije o zahtjevima softvera. Također, aktivno se komunicira s klijentima, članovima tima i ostalim zainteresiranim stranama kako bi detaljno razumjeli zahtjeve. Nakon toga slijedi faza planiranja testiranja, u kojoj se detaljno izrađuje plan koji opisuje pristup testiranju, testne slučajeve i testne podatke. U trećoj fazi, tim za testiranje razvija testne slučajeve prema zahtjevima i planu testiranja, pazeći da se obuhvate svi mogući scenariji i da softver zadovoljava specificirane zahtjeve. Nakon toga, tim postavlja testno okruženje u kojem se provodi testiranje. U fazi izvršenja testiranja, tim provodi testne slučajeve, bilježi rezultate i prijavljuje sve nastale nedostatke ili probleme timu za razvoj radi daljnjeg rješavanja. Konačno, u fazi evaluacije, tim procjenjuje rezultate testiranja i provjerava zadovoljava li softver specificirane zahtjeve. Ukoliko su identificirani nedostaci ili problemi, isti se prijavljuju timu za razvoj kako bi se poduzeli potrebni koraci za njihovo rješavanje.

2.3. CILJEVI TESTIRANJA

Ciljevi testiranja softvera prema Sawant i sur. (2012.) mogu se sažeti na sljedeći način:

1. Provjera: Osiguravanje da softver zadovoljava specificirane zahtjeve i funkcionira prema namjeri.
2. Validacija: Potvrda da softver radi onako kako je željeno i ispunjava postavljene uvjete.
3. Pokrivenost prioriteta: Izvršavanje testiranja na učinkovit i djelotvoran način unutar određenog budžeta i vremenskih ograničenja.
4. Prevencija grešaka: Identifikacija i rješavanje potencijalnih problema i grešaka u sustavu kako bi se spriječile poteškoće u softveru.
5. Upotrebljivost: Osiguravanje da softver bude korisnički prijateljski i zadovoljava potrebe krajnjih korisnika.
6. Pouzdanost: Testiranje softvera kako bi se osigurala njegova stabilnost, robusnost i sposobnost dosljednog izvršavanja pod različitim uvjetima.
7. Sigurnost: Provoditi testiranje sigurnosti kako bi se identificirale i riješile ranjivosti i zaštitilo od neovlaštenog pristupa.
8. Performanse: Procjena performansi softvera, uključujući brzinu, odzivnost i upotrebu resursa.
9. Uslužnost: Osiguravanje da softver zadovoljava relevantne industrijske standarde, propise i smjernice.
10. Zadovoljstvo korisnika: Isporuka visokokvalitetnog softverskog rješenja koje zadovoljava zahtjeve i očekivanja klijenta.

Quadri & Farooq (2010.) smatraju da testiranje softvera ima implicitne i eksplicitne ciljeve. Eksplicitni ciljevi testiranja softvera su provjeriti i potvrditi da softverska aplikacija ili program zadovoljava poslovne i tehničke zahtjeve koji su postavljeni pred njega te da radi kako se očekuje. Implicitni ciljevi testiranja softvera su identificirati važne pogreške ili nedostatke kategorizirane prema razini ozbiljnosti u aplikaciji koje se moraju popraviti.

2.4. ZAHTJEVI

Proces određivanja zahtjeva (specifikacija) je aktivnost unutar softverskog procesa u kojoj se identificiraju i analiziraju zahtjevi budućeg sustava. Kao rezultat procesa nastaje dokument koji opisuje funkcionalnosti koje sustav treba imati, ali bez navođenja načina njihovog postizanja. Tim sastavljen od programera i budućih korisnika je odgovoran za proces određivanja zahtijeva.

Burnstein (2002.) dijeli zahtjeve na dvije vrste: funkcionalni i nefunkcionalni.

Funkcionalni zahtjevi: Ovi zahtjevi definiraju što softver treba raditi i kako se treba ponašati. Oni opisuju funkcionalnosti i značajke softvera ključne za korisnikovo zadovoljstvo. U slučaju Android telefona, primjer funkcionalnog zahtjeva mogu bi bio obavljanje telefonskih poziva. Funkcionalni zahtjevi dalje se dijele na eksplicitne i implicitne zahtjeve. Eksplicitni zahtjevi izričito su navedeni od strane klijenta ili korisnika, dok implicitni zahtjevi se očekuju da su prisutni u softveru i bez izravnog spominjanja.

Nefunkcionalni zahtjevi: Ovi zahtjevi fokusiraju se na kvalitete i karakteristike softvera, a ne na njegove specifične funkcionalnosti. Oni uključuju aspekte poput performansi, upotrebljivosti, pouzdanosti i sigurnosti. Na primjer, u slučaju Android telefona, nefunkcionalni zahtjevi mogu uključivati rad aplikacija bez bugova ili moderan dizajn.

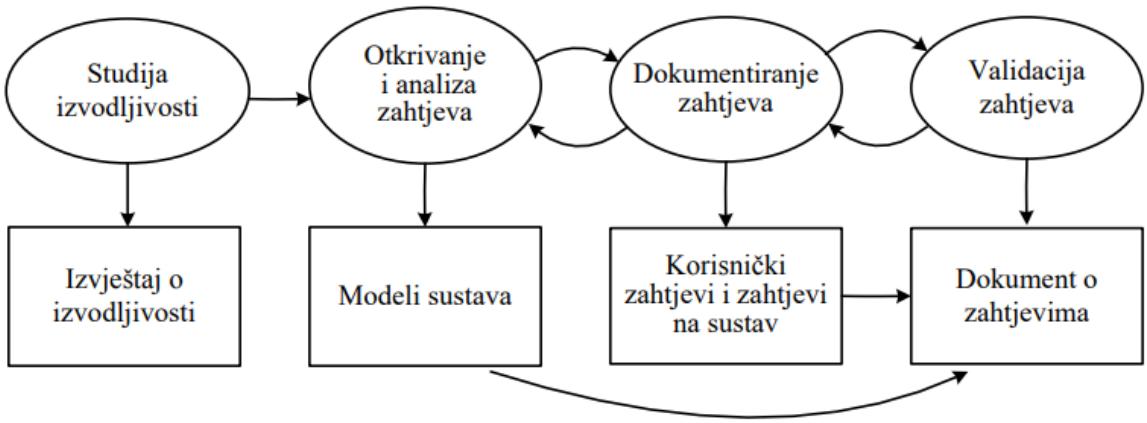
Manger (2016.) ukazuje da se cijeli proces utvrđivanja zahtjeva sastoji od sljedećih koraka:

1. Studija izvodljivosti: Vrši se analiza kakvo je trenutno stanje sustava, kakva je njegova ekonomska isplativost i može li se razviti unutar određenog budžeta.
2. Identifikacija i analiza zahtijeva: Analiziraju se otkriveni zahtjevi kako bi se izradili modeli sustava, koji omogućuju bolje razumijevanje korisničkih potreba i preciznije formuliranje zahtjeva.
3. Dokumentiranje zahtijeva: Prikupljene informacije pretvaraju se u dokumente koji definiraju zahtjeve. Postoje najmanje dvije razine opisivanja zahtjeva: "korisnički zahtjevi" i "zahtjevi sustava".

4. Validacija zahtjeva: Provjerava se jesu li zahtjevi ispravno opisani i postoje li greške u njihovoj specifikaciji.

Konačni rezultat ovog procesa je dokument koji sadrži jasno definirane zahtjeve, osiguravajući tako lako razumijevanje svih članova ovoga procesa.

Slika 2.3. Pod-aktivnosti unutar utvrđivanja zahtjeva



Izvor: Manger, R. (2016.): Softversko inženjerstvo, Element, Zagreb

2.5. VERIFIKACIJA I VALIDACIJA

Testiranje je povezano s dva procesa nazvana verifikacija i validacija. Burnstein (2002.) te procese opisuje na slijedeći način.

Verifikacija se odnosi na provjeru ispravnosti izgradnje softvera i usklađenost s specifikacijom tijekom ili na kraju razvojnog ciklusa. To uključuje provjeru pravilne implementacije zahtjeva, ispunjenje tehničkih i funkcionalnih zahtjeva, te ispravno funkcioniranje softvera prema očekivanjima. Verifikacija se može provesti putem različitih metoda, kao što su statička verifikacija koja uključuje pregled koda ili dokumentacije, ili testiranje koje obuhvaća izvršavanje softvera i provjeru rezultata.

Validacija, s druge strane, odnosi se na provjeru zadovoljavanja stvarnih potreba korisnika i korisnost softvera u stvarnom okruženju. Validacija se fokusira na provjeru pružanja očekivanih funkcionalnosti i performansi softvera, te zadovoljstva korisničkim iskustvom. Ova provjera se obično provodi putem testiranja softvera u stvarnim uvjetima korištenja.

Metode verifikacija i validacije koje ističe Marinčić i sur. (2005.) su:

1. Validacija konceptualnog modela: Ova metoda uključuje pregled projektne specifikacije kako bi se provjerila prikladnost konceptualnog modela. Sudionicima se šalje izvještaj, a zatim se prikupljaju povratne informacije o prikladnosti modela.
2. Validacija podataka: Ova metoda se fokusira na provjeru pouzdanosti i točnosti podataka. Provjerava se izvor podataka, otkrivaju se eventualne netočnosti i razvija se postupak za prikupljanje i pretvorbu podataka u ispravan format.
3. Verifikacija i strukturna validacija: Ove dvije metode se provode tijekom faze kodiranja modela. Verifikacija uključuje provjeru koda kako bi se osigurala ispravna primjena podataka i logike. Strukturna validacija obuhvaća vizualnu provjeru modela kroz razne tehnike kao što su prolazak kroz model događaj po događaj, predviđanje i provjera sljedećeg događaja, izolacija dijelova modela…
4. Validacija cjelokupnog modela: Ova metoda uključuje usporedbu modela s stvarnim sustavom ili drugim modelima. Ukoliko nema dostupnih podataka, koriste se metode usporedbe s matematičkim modelima, deterministčkim modelima ili već verificiranim simulacijskim modelima.

Navedene metode imaju za cilj osigurati verifikaciju i validaciju softvera i modela te provjeriti njihovu točnost, prikladnost i usklađenost s ciljevima i stvarnim sustavom.

2.6. PRINCIPI TESTIRANJA

Quadri & Farooq (2010.) navode slijedeće principe testiranja:

1. Testiranje programa s ciljem pronalaženja grešaka - Cilj testiranja je pronaći greške, a najbolji način postizanja toga je testirati program s namjerom da ga se dovede do neuspjeha. Što više grešaka pronađemo, to je testiranje učinkovitije.
2. Potpuno iscrpno testiranje nije moguće - Nemoguće je testirati sve moguće kombinacije ulaza, stoga se prioritizira na temelju rizika i najkritičnijih dijelova sustava.
3. Rano testiranje štedi vrijeme i novac - Što se ranije otkrije greška, to je jeftinije i lakše njihovo otklanjanje. Testiranje treba započeti što je prije moguće u životnom ciklusu razvoja softvera.
4. Klasteriranje grešaka - Mali broj modula sadrži većinu grešaka otkrivenih tijekom testiranja. Ovaj princip sugerira da se napori trebaju usmjeriti u testiranju na najkritičnija i najsklonija pogreškama područja sustava.
5. Paradoks pesticida – Ako se isti testovi ponavljaju iznova, s vremenom ovi testovi više ne pronalaze nove greške. Potrebno je redovito pregledavati i ažurirati testne slučajeve.

6. Testiranje ovisi o kontekstu - Učinkovitost testiranja ovisi o kontekstu projekta, poput zahtjeva, tehnologije, proračuna i rasporeda. Testiranje treba biti prilagođeno specifičnim potrebama projekta.
7. Pogreška odsutnosti grešaka - Testiranje treba usredotočiti ne samo na pronalaženje grešaka, već i na osiguravanje da sustav ispunjava zahtjeve i očekivanja korisnika.
8. Proces testiranja treba biti planiran i kontroliran - Testiranje treba biti dobro planiran i kontroliran proces koji se nadzire i prilagođava prema potrebi kako bi se osigurala njegova učinkovitost.

3. TIPOVI TESTIRANJA PROGRAMSKIH PROIZVODA

3.1. TESTIRANJE BIJELE, CRNE I SIVE KUTIJE

Autori Sawant i sur. Navode da je White Box ispitivanje izuzetno učinkovita metoda ispitivanja koja ne samo da testira funkcionalnost softvera, već također ispituje unutarnju strukturu aplikacije. Tester selektira različite pristupe za testiranje koda i kreiranje scenarija testnog procesa. Pri dizajniranju testnih slučajeva za provođenje White Box ispitivanja, potrebne su programerske vještine za oblikovanje testnih slučajeva. Arumugam (2019.) tvrdi da se White Box ispitivanje također naziva Clear Box ili Glass Box ispitivanje. Ova vrsta ispitivanja može se primijeniti na sve razine, uključujući jedinično, integracijsko ili sustavno ispitivanje. Ova vrsta ispitivanja također se naziva sigurnosno ispitivanje jer zadovoljava potrebu za utvrđivanjem štiti li informacijski sustav podatke i održava li namijenjenu funkcionalnost. Prema Eskić (2019.) neke od prednosti "bijele kutije" metode je mogućnost rane inicijacije testiranja, neovisno o dostupnosti korisničkog sučelja (GUI). Ova metoda omogućava dublje i temeljitije testiranje, obuhvaćajući većinu funkcionalnosti. S druge strane, postoje i određeni izazovi. Složenost testova zahtijeva stručne resurse s dubokim razumijevanjem programiranja i tehnika testiranja. Održavanje testnih skripti može postati opterećenje, posebno ako se implementacija često mijenja. Također, alati za testiranje možda neće uvijek biti lako dostupni.

S druge strane, Black Box ispitivanje je tehnika ispitivanja koja se uglavnom fokusira na funkcionalnost aplikacije bez ulaska u detalje implementacije. Tester ne poznaje unutarnju strukturu programskog rješenja koje se testira. Ova tehnika se može primijeniti na svaku razinu ispitivanja u životnom ciklusu razvoja softvera. Glavni cilj prema Manger (2016.) je izvršiti ispitivanje na način koji obuhvaća sve funkcionalnosti aplikacije kako bi se utvrdilo zadovoljava li početno specificirane zahtjeve korisnika. Eskić (2019.) konstatira da tester nije obvezan poznavati programski jezik ili tehničke detalje

implementacije softverskog rješenja. Testiranje može provoditi neovisna osoba, izvan programerskog tima, čime se omogućava objektivna analiza bez programerske pristranosti. Testni slučajevi se mogu razviti nakon završetka specifikacija proizvoda. S druge strane, postoje i određeni nedostaci. Metoda može ograničiti testiranje samo na ograničen broj mogućih scenarija, dok mnogi drugi mogući scenariji mogu ostati neistraženi. Kvalitetno definiranje testnih slučajeva može biti izazovno bez jasnih specifikacija. Također, testovi mogu postati nepotrebni ako su programeri već proveli isti testni slučaj.

Arumugam (2019.) ukazuje da Grey Box ispitivanje je kombinacija tehnika White Box ispitivanja i Black Box ispitivanja koja kombinira prednosti oba pristupa. Potreba za ovakvim vrstom ispitivanja proistekla je iz činjenice da ispitivač ima djelomično znanje o unutarnjoj strukturi i funkcioniranju aplikacije, što omogućava testiranje funkcionalnosti na bolji način uzimajući u obzir unutarnju strukturu aplikacije.

3.2. ALPHA I BETA TESTIRANJE

Alpha i Beta testiranje prihvatljivosti spada u testiranje prihvatljivosti s obzirom na okolinu. Ivandić (2017.) ističe kako je Alpha testiranje vrsta internog testiranja kojom se pokušavaju identificirati sve pogreške u sustavu prije nego što se preda krajnjim korisnicima. Glavni cilj ovog testiranja je simulirati potrebe krajnjih korisnika koristeći tehnike testiranja "White box" i "Black box". Alpha testiranje se provodi u internom okruženju i izvode ga zaposlenici poduzeća kako bi osigurali da sustav ispunjava sve zahtjeve prije izlaska na tržište.

Beta testiranje, poznato i kao eksterno testiranje, provodi se izvan poduzeća. Dudaš (2019.) navodi da testeri nisu zaposlenici tvrtke niti su sudjelovali u razvoju programa. Također, ovo testiranje se obično provodi nakon alfa testiranja. U alfa testiranju sudjeluje manji broj ljudi u usporedbi s beta testiranjem, ali je važno prvo obaviti alfa testiranje kako bi se smanjio broj pogrešaka prije nego što proizvod postane dostupan. Slično kao i kod alfa testiranja, svaki beta tester (krajnji korisnik) pruža povratne informacije o primijećenim pogreškama. Razvojni tim potom ispravlja i poboljšava programski proizvod.

3.3. REGRESIJSKO TESTIRANJE

Regresijsko testiranje ima ključnu ulogu u procesu razvoja programske aplikacije. Ova metoda se primjenjuje nakon testiranja pojedinih komponenti kako bi se testirao cjelokupni programski proizvod.

Regresijski se testira kada se sustavu doda nova funkcionalnost, a kod je modificiran da apsorbira i integrira tu funkcionalnost s postojećim kodom. Također, kada je neki nedostatak identificiran u softveru te kada se kod modificira kako bi se optimizirao njegov rad.

Kalaica (2022.) tvrdi da se u skladu s programskim proizvodom i njegovim specifikacijama, moraju definirati testovi koji se trebaju provesti. S obzirom na veliki broj mogućih testova, nastoji se uštedjeti vrijeme koliko god je moguće. Stoga je važno odrediti prioritet, koji su testovi najvažniji kako bi se testiranje obavilo što brže i učinkovitije. U nekim slučajevima, testovi nižeg prioriteta se mogu potpuno izostaviti kako bi se skratilo vrijeme testiranja. Unatoč definiranju prioriteta, broj testova može biti i dalje prevelik, pa se u tim slučajevima dio regresijskog testiranja može automatizirati. Automatizacija omogućuje brže izvršavanje testova i smanjenje ljudskih resursa potrebnih za testiranje.

Prema Hamiltonu (2023.) prednosti regresijskog testiranja leže u njegovoj sposobnosti poticanja efikasnog softverskog razvoja i unaprjeđenja njegove kvalitete. Ovaj pristup osigurava usuglašenost s korisničkim zahtjevima i održava stabilnost softvera u razvoju. Također, pojavljuju se i neki izazovi. Regresijsko testiranje zahtijeva izvođenje nakon svake, pa čak i najsitnije promjene, a ručno izvođenje može biti zahtjevno i dugotrajno jer podrazumijeva često ponavljanje postupaka.

3.4. AD HOC TESTIRANJE

Ovo testiranje je vrsta beta testiranja koje obično vrši krajnji korisnik. Bhatti i sur. (2019.) tvrde da se Ad hoc testiranje može provoditi tijekom bilo koje faze SDLC (Software Development Lifecycle) testnog procesa, ali samo pod uvjetom da je sustav potpuno razvijen i funkcionalan. Vrlo je važno da tester posjeduje duboko razumijevanje funkcionalnosti sustava. Ad hoc testiranje može se koristiti i kada postoji ograničeno vrijeme u STLC (Software Testing Lifecycle) procesu. Međutim, Ad hoc testiranje ne bi trebalo biti provedeno ako prethodni bug još uvijek nije uklonjen, jer to može uzrokovati pojavu ili skrivanje još jednog defekta. Također bi trebalo izbjegavati ad hoc testiranje tijekom testiranja builda/verzije od strane klijenata.

Postoje tri vrste Ad hoc testiranja:

1. Buddy Testing
2. Pair Testing
3. Monkey Testing

3.5. STATIČKO TESTIRANJE

Statičko testiranje je jedna vrsta softverskog testiranja u kojem se softverska aplikacija testira bez izvršavanja koda ili programa. Testiranje se može izvesti ručno ili pomoću alata za automatsku obradu. Glavni cilj statičkog testiranja je poboljšanje kvalitete proizvoda identificiranjem i dokumentacijom grešaka u ranim fazama procesa razvoja. Mamić (2019.) konstatira da postoje nekoliko tehnika statičkog testiranja, kao što su pregled (review), walkthrough i inspekcija (inspection). Pregled može biti formalan ili neformalan. U neformalnom pregledu, često se prezentiraju službeni dokumenti projektnom timu i voditeljima projekta, poput specifikacija zahtjeva, detaljnog dizajna i tehničkih specifikacija. Tijekom prezentacije dokumentacije, članovi tima mogu izražavati mišljenja i predlagati poboljšanja softvera. S druge strane, formalni pregled je strukturiran proces koji uključuje planiranje, kick-off sastanak, pripremu, formalni sastanak, korekcije i follow-up, a sve faze su dokumentirane.

3.6. DINAMIČKO TESTIRANJE

Dinamičko testiranje je vrsta testiranja softvera koje se provodi radi analize dinamičkog ponašanja koda. Uključuje testiranje softvera za unesene vrijednosti i izlazne vrijednosti koje nisu konstante te se mijenjaju s vremenom. Testovi se koriste kako bi se identificirale potencijalne slabosti softvera u stvarnom okruženju. Oni uključuju unošenje ulaznih podataka te uspoređivanje dobivenih rezultata s očekivanim rezultatima. Cilj je otkriti eventualne razlike i provjeriti ispravnost softvera.

Postoje različite razine dinamičkog testiranja. To su:

1. Testiranje jedinica (Unit Testing)
2. Testiranje integracije (Integration Testing)
3. Testiranje sustava (System Testing)
4. Testiranje prihvaćanja (Acceptance Testing)

3.7. RUČNO TESTIRANJE

Ručno testiranje je vrsta softverskog testiranja u kojem se testni slučajevi izvršavaju ručno, umjesto korištenja automatiziranog alata. Ono je jedna od osnovnih metoda testiranja, jer može otkriti kako očite, tako i skrivene greške u softveru. Greška predstavlja razliku između očekivanog rezultata i rezultata koji pruža program. Prije nego što softver ili proizvod bude podvrgnut automatiziranom

testiranju, on mora proći kroz ručno testiranje. Tijekom ručnog testiranja, razvojni tim ispravlja pronađene greške prije nego što se proizvod predstavi testeru za ponovno testiranje. Cilj je utvrditi ispunjava li aplikacija zahtjeve navedene u dokumentu s zahtjevima.

Prema Džale (2021.) ručno testiranje je najstariji oblik testiranja. U ovom pristupu, tester samostalno sastavlja testne scenarije koji provjeravaju funkcionalnosti aplikacije i njezine rubne slučajeve. Tester mora imati dobro poznavanje aplikacije i njezinih mogućnosti. Nakon sastavljanja testnih scenarija, tester izvršava testove i bilježi rezultate - jesu li testovi prošli (očekivani rezultati) ili pali (nepoželjni rezultati). U slučaju neuspjelih testova, bitno je zabilježiti sve korake koji su doveli do tog neuspjeha i detaljno opisati što se dogodilo. Na temelju tih zabilješki, programeri mogu ponoviti korake, identificirati pogreške i ispraviti ih. Nakon toga, testeri ponovno provode testiranje kako bi potvrdili da su greške ispravljene.

Prema Sharma (2014.) glavni problemi ručnog testiranja su:

1. Testiranje je zahtjevno i monotono
2. Značajna ulaganja u ljudske resurse: Ručno izvođenje testnih slučajeva zahtijeva veći broj testera.
3. Niža pouzdanost: Ručno testiranje je manje pouzdano jer se testovi ne izvode uvijek precizno zbog ljudskih pogrešaka.
4. Nemogućnost programiranja: Nemoguće je koristiti programiranje za izradu složenih testova koji bi otkrili skrivene informacije.
5. Ručno testiranje može postati dosadno i time povećati mogućnost pogrešaka.

3.8. AUTOMATSKO TESTIRANJE

Prije nego što se softver pusti u proizvodnju, funkcioniranje proizvoda se provjerava i osigurava njegova usklađenost sa zahtjevima korištenjem automatiziranog testiranja. Tvrtka može izvršiti određena testiranja softvera brže i bez potrebe za ljudskim testerima zahvaljujući automatiziranom testiranju. Veliki ili testni slučajevi koji se ponavljaju prikladniji su za automatizirano testiranje. Skripta za automatizirani test može se pripremiti jednom i koristiti više puta te zato povratna informacija do testera dolazi puno brže. Tvrtka može koristiti automatizirane testove u različitim situacijama, uključujući testiranje regresije, jedinica i sučelja za programiranje aplikacija (API).

Stankić (2021.) navodi da automatizacija ima mnoge prednosti, ali također ima nedostatke i probleme koje treba riješiti. Ono je točnije i pouzdanije od ručnog jer ne podliježe ljudskih greškama te uvijek

daje isti rezultat. Međutim, budući da uvođenje automatskih testova zahtijeva značajne financijske izdatke i opsežnu obuku osoblja, a ne nudi nikakvu sigurnost u uspjeh, organizacije se često suočavaju s neizvjesnošću u tom pogledu. Automatsko testiranje nije namijenjeno da u potpunosti zamijeni ručno testiranje, niti je to realistična pretpostavka. Činjenica da se automatsko testiranje ne može koristiti za prepoznavanje odstupanja u dizajnu, ilustrira njegove nedostatke. Osim toga, tijekom testiranja s alatima za automatizaciju neće biti obuhvaćeni svi scenariji, niti se svaki uvjet može automatizirati. Zbog čestih promjena u sustavu koji se testira, IN automatizacija se ne isplati ni kada su projekti manjeg opsega.

4. RAZINE TESTIRANJA PROGRAMSKIH PROIZVODA

Razine testiranja softvera predstavljaju različite faze životnog ciklusa razvoja softvera u kojima se provodi testiranje. Svaka razina testiranja ima svoju svrhu, specifične izazove i ciljeve. Glavna svrha razina testiranja je organiziranje testiranja i olakšavanje prepoznavanja testnih scenarija na svakoj pojedinoj razini.

Razine testiranja softvera uključuju:

1. Jedinično testiranje

2. Integracijsko testiranje

3. Sistemsko testiranje

4. Testovi prihvaćanja

4.1. JEDINIČNO TESTIRANJE

Testiranje jedinica samo je jedna od razina testiranja koja zajedno čine cjelokupnu sliku testiranja sustava. Prema Milin (2023.) ono se temelji na provjeri najmanjih samostalnih komponenata unutar softvera, uključujući funkcije, metode, procedure i module. Cilj ove faze testiranja je osigurati da svaka pojedinačna jedinica softverskog koda funkcionira prema očekivanjima. Jedinično testiranje općenito se smatra testiranjem bijele kutije.

Prednosti testiranja jedinica prema Sawant i sur. (2012.) su sljedeće:

1. Testiranje na razini jedinica je vrlo ekonomično.

2. Pruža značajno veće poboljšanje pouzdanosti u odnosu na testiranje na razini sustava. Posebno otkriva greške koje su inače suptilne i često katastrofalne, poput neočekivanih rušenja sustava koja se događaju u stvarnom radnom okruženju kada se dogodi nešto neobično.
3. Omogućava testiranje dijelova projekta bez čekanja da drugi dijelovi budu dostupni.
4. Paralelizam u testiranju omogućava istovremeno testiranje i rješavanje problema od strane više inženjera.
5. Omogućava otkrivanje i uklanjanje nedostataka uz znatno manje troškove u usporedbi s kasnijim fazama testiranja.
6. Pojednostavljuje postupak otklanjanja pogrešaka ograničavanjem područja pretraživanja koda na malu jedinicu.
7. Smanjuje ciklus kompiliranja, izgradnje i otklanjanja pogrešaka prilikom rješavanja složenih problema.

4.2. INTEGRACIJSKO TESTIRANJE

Autor Lučić (2019.) navodi da integracijsko testiranje predstavlja stupanj ispitivanja u kojem se pojedinačne funkcionalne jedinice ili moduli sustava kombiniraju i testiraju kao skup kako bi se istražile kritične točke povezivanja. Cilj ove faze je otkrivanje grešaka u procesu interakcije između spojenih funkcionalnih jedinica. U integracijskom testiranju često se koriste testni upravljački programi ili komponente kao pomoćni alati. Komponentno integracijsko testiranje ima za svrhu identifikaciju pogrešaka u sučeljima i interakciji između integriranih komponenata. Različite metode testiranja, kao što su crna, bijela i siva kutija, primjenjive su na ovoj razini ispitivanja.

Postoje različiti pristupi integracijskom testiranju koje navode Sawant i sur. (2012.):

1. Integracijsko testiranje odozgo prema dolje (Top-down Integration Testing)

Top-down integracija je pristup testiranju u kojem se integracijski testovi izvode počevši od vrha i prolaze kroz svaki sloj softverske arhitekture. Kontrolni tok testa kreće se od korisničkog sučelja (UI) i postupno se spušta prema donjim slojevima, sve do baze podataka softvera. Ovaj pristup testiranju integracije posebno je pogodan za web aplikacije i softverske arhitekture s više slojeva.

Glavna prednost pristupa testiranju integracije odozgo prema dolje je njegova relativna jednostavnost implementacije i minimalna ovisnost o drugim dijelovima vaše aplikacije. Ovaj pristup koristi stubove (engl. stubs), koji su općenito lakši za implementaciju od stvarnih upravljačkih programa. Jednostavna

i inkrementalna priroda ovog pristupa olakšava brzo otkrivanje grešaka u sučeljima, iako neki kritičari upozoravaju da može rezultirati nedovoljnim testiranjem modula niže razine.

2. Integracijsko testiranje odozdo prema gore (Bottom-up Integration Testing)

Testiranje integracije odozdo prema gore je proces u kojem se pojedinačne komponente testiraju i integriraju počevši od najnižeg modula u arhitekturi, te se postupno kreće prema višim slojevima. Ovaj pristup omogućuje timovima da započnu testiranje čak i kada su još u tijeku razvoj modula visokih razina. Najčešće se koristi kada timovi pokušavaju integrirati već gotove komponente s postojećim proizvodima.

Integracijsko testiranje odozdo prema gore obično postiže visoke stope uspjeha te predstavlja brz i učinkovit oblik testiranja integracije. Budući da se prvo testiraju niži moduli, timovi za testiranje mogu osigurati da ključni temeljni dijelovi aplikacije besprijekorno funkcioniraju zajedno prije nego što pređu na testiranje viših razina.

Jedan od glavnih nedostataka testiranja odozdo prema gore je nemogućnost promatranja funkcionalnosti na razini sustava sve dok se ne postavi konačni testni upravljački program.

Integracijsko testiranje odozdo prema gore započinje izgradnjom i testiranjem pojedinačnih modula. Budući da se komponente integriraju odozdo prema gore, uvijek je dostupna obrada potrebna za komponente niže razine, te nema potrebe za korištenjem zamjenskih modula (stubova).

3. Testiranje sendvič integracije

Sendvič integracija je metodologija testiranja koja kombinira pristupe testiranja odozgo prema dolje i odozdo prema gore.

U sendvič integraciji, sustav je podijeljen u tri sloja: srednji sloj, gornji sloj i donji sloj. Ispitivači započinju testiranje modula od srednjeg sloja i nastavljaju prema gore i dolje, s posebnim naglaskom na prioritiziranje modula na najvišoj i najnižoj razini. Sendvič integracija koristi priključke i upravljačke programe za testiranje modula na svim razinama.

Ova metodologija testiranja posebno je korisna u slučajevima velikih projekata koji se mogu podijeliti na više podprojekata ili kada se testiraju iznimno veliki softverski moduli.

Ipak, testiranje sendvič integracije može biti dugotrajno. Također, ova vrsta testiranja ne pruža priliku za testiranje modula koji čine podpotprojekte prije konačne integracije, što može dovesti do ozbiljnih problema ako se ti moduli zanemare.

4.3. SISTEMSKO TESTIRANJE

Sistemsko testiranje je proces koji se provodi na potpuno integriranom sustavu kako bi se procijenila usklađenost sustava sa svojim specificiranim zahtjevima. Sistemsko testiranje spada u kategoriju testiranja crne kutije (black box testing) i stoga ne zahtijeva poznavanje unutarnjeg dizajna koda ili logike. Prema Pleše (2020.) osnovne ciljeve sistemskog testiranja možemo sažeti u tri ključna aspekta:

1. Identificiranje problema koji se pojavljuju isključivo na razini sustava i nisu mogli biti otkriveni tijekom testiranja pojedinačnih komponenti ili integracijskog testiranja
2. Izgradnja povjerenja da proizvod uspješno i pravilno ostvaruje tražene sposobnosti
3. Prikupljanje informacija koje mogu biti korisne pri donošenju odluka o lansiranju proizvoda na tržište

Neke od različitih vrsta sistemskog testiranja su sljedeće:

1. Testiranje oporavka (Recovery testing)
2. Testiranje sigurnosti (Security testing)
3. Testiranje grafičkog korisničkog sučelja (Graphical user interface testing)
4. Testiranje kompatibilnosti (Compatibility testing)

4.4. TESTOVI PRIHVAĆANJA

Testiranje prihvaćanja je važna faza u razvoju softvera koja uključuje aktivno uključivanje krajnjeg korisnika. Mamić (2021.) navodi da se ovaj tip testiranja provodi kako bi se potvrdilo da softver ispunjava sve definirane značajke i da se ponaša u skladu s očekivanjima. Krajnji korisnik ima ulogu provjeriti da softver zadovoljava njihove zahtjeve i da radi kako je predviđeno prije nego što se pusti u rad.

Testovi prihvaćanja omogućuju identifikaciju potencijalnih grešaka koje nisu otkrivene tijekom jediničnog testiranja. Oni pružaju korisne povratne informacije o tome u kojoj mjeri je sustav dovršen i kako se ponaša u stvarnom okruženju. Ovi testovi se fokusiraju na provjeru funkcionalnosti, korisničkog sučelja i usklađenosti s definiranim zahtjevima.

Iako izvršavanje testova prihvaćanja može predstavljati izazov, pravilno definiranje tko će ih provesti, kada i kako će se provesti te kako će se mjeriti rezultati, ključno je za uspješnu implementaciju. Obično poslovni tim raspisuje ove testove, često čak i prije potpune implementacije od strane razvojnog tima.

5. ANALIZA I PRIMJER RUČNOG I AUTOMATSKOG TESTIRANJA

5.1. O PODUZEĆU

Tech Resources je privatna tvrtka za razvoj softvera i savjetovanje specijalizirana za pružanje ekonomičnih tehničkih resursa prilagođenih jedinstvenim potrebama svakog klijenta. Sjedište tvrtke nalazi se u Splitu, Hrvatska.

Sa više od 10 godina iskustva u radu s tvrtkama na razini poduzeća diljem Europske unije i Sjedinjenih Američkih Država, ova tvrtka radi na digitalizaciji procesa i rješavanju problema u mnogim različitim industrijama.

Najznačajniji projekti:

1. Rješavanje problema u Upravi teretne luke
2. Optimizacija rad u uslugama prijevoza
3. Smanjivanje troškove obrade plaća

Slika 5.1. Logo tvrtke Tech Resourced d.o.o.

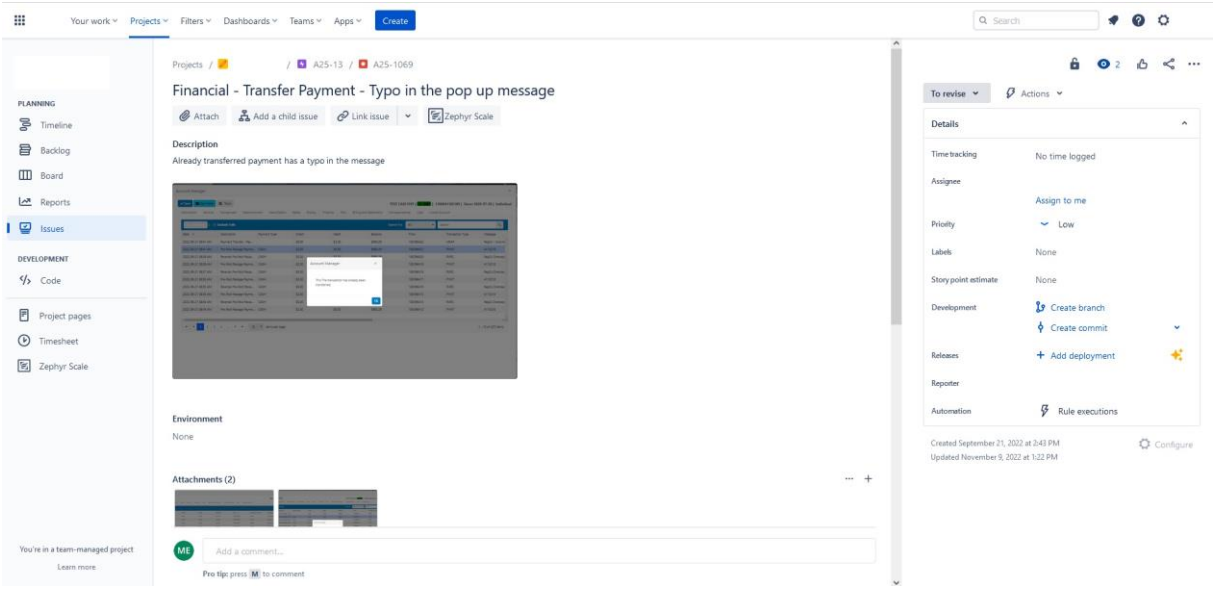
Izvor: Internet

5.2. PRIMJER RUČNOG TESTIRANJA

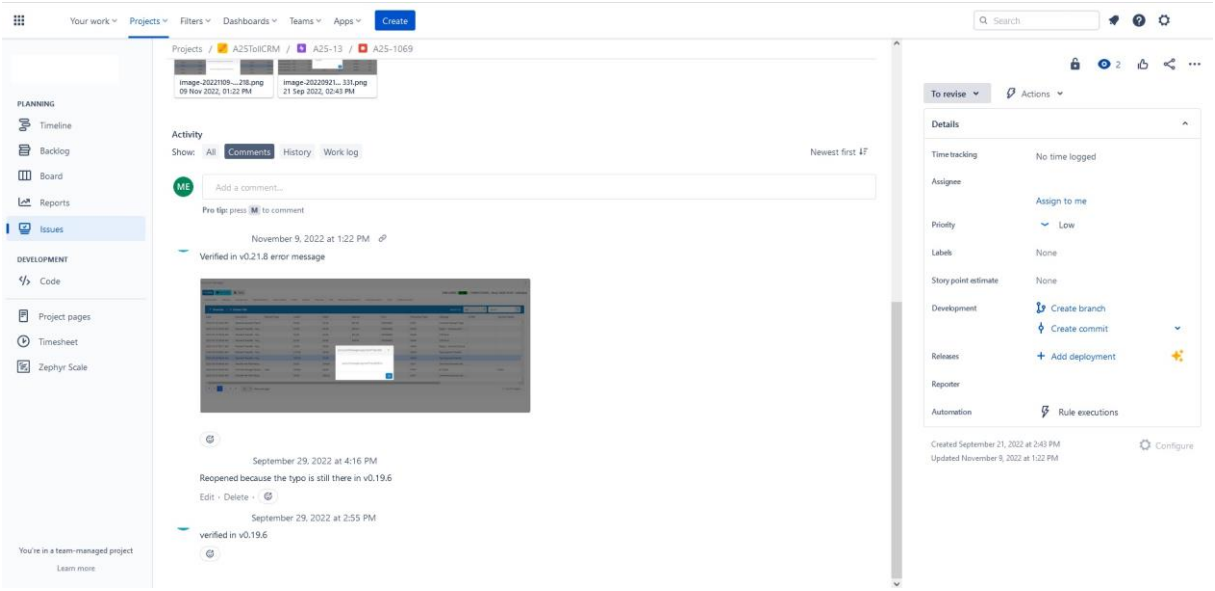
Tvrtka koristi Jiru kao alat za upravljanje i praćenje procesa testiranja softvera. Kroz upotrebu Jire, moguće je učinkovito stvoriti, organizirati i upravljati test slučajevima, pratiti rezultate testiranja i dokumentirati otkrivene greške ili probleme. Test slučajevi mogu biti temeljito opisani, prioritetizirani,

dodijeljeni odgovornim pojedincima te strukturirani u hijerarhijskom obliku. Nakon izvršenja testova, rezultati mogu biti evidentirani, uključujući status testa, relevantne komentare i priloge. U slučaju identifikacije grešaka, Jira omogućuje stvaranje novih bugova ili problema, uz detaljan opis i pridruživanje nadležnim osobama za daljnje rješavanje kao što je prikazano i na slici.

Slika 5.2. Primjer buga u programu Jira



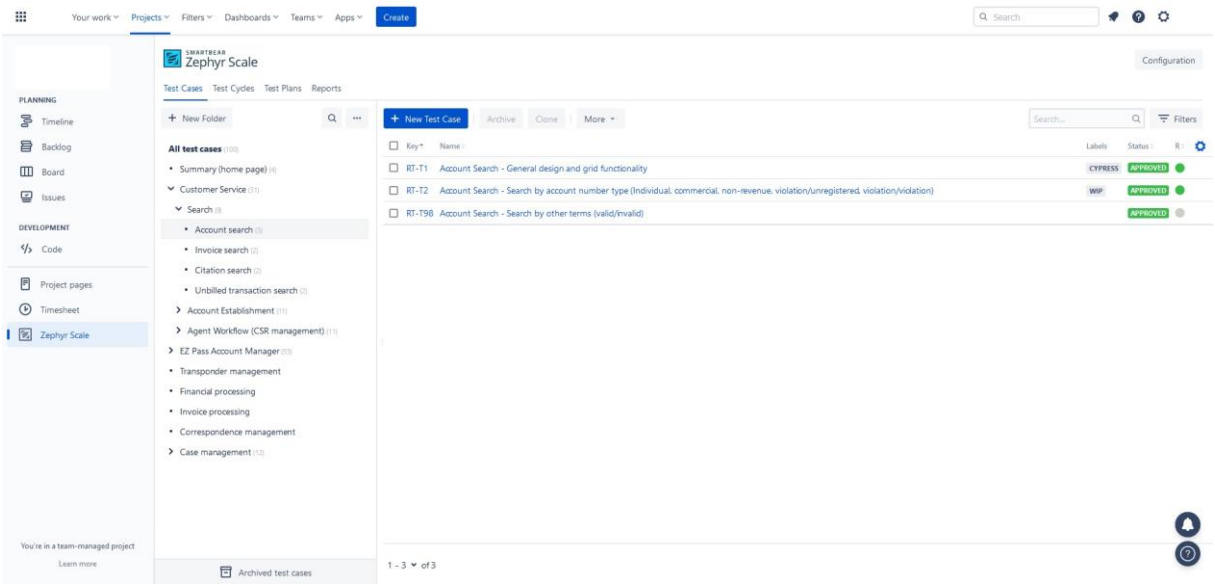
Slika 5.3. Komentari vezani za prijavljeni bug



Izvor: tvrtka Tech Resources

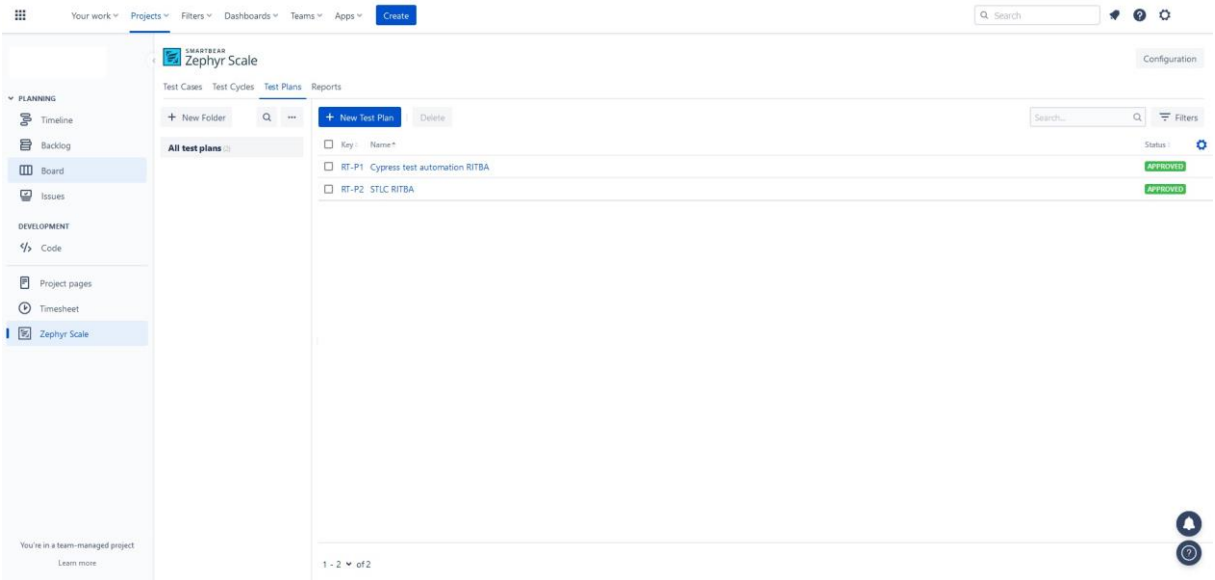
Slijedeći program koji tvrtka Tech Resources koristi u testiranju jest Zephyr. Zephyr je dodatak (plug-in) za Jiru i njegov homepage prikazan je na slici. On pruža napredne mogućnosti za upravljanje testovima unutar Jire i olakšava organizaciju, praćenje i izvršavanje testova.

Slika 5.4. Zephyr home page

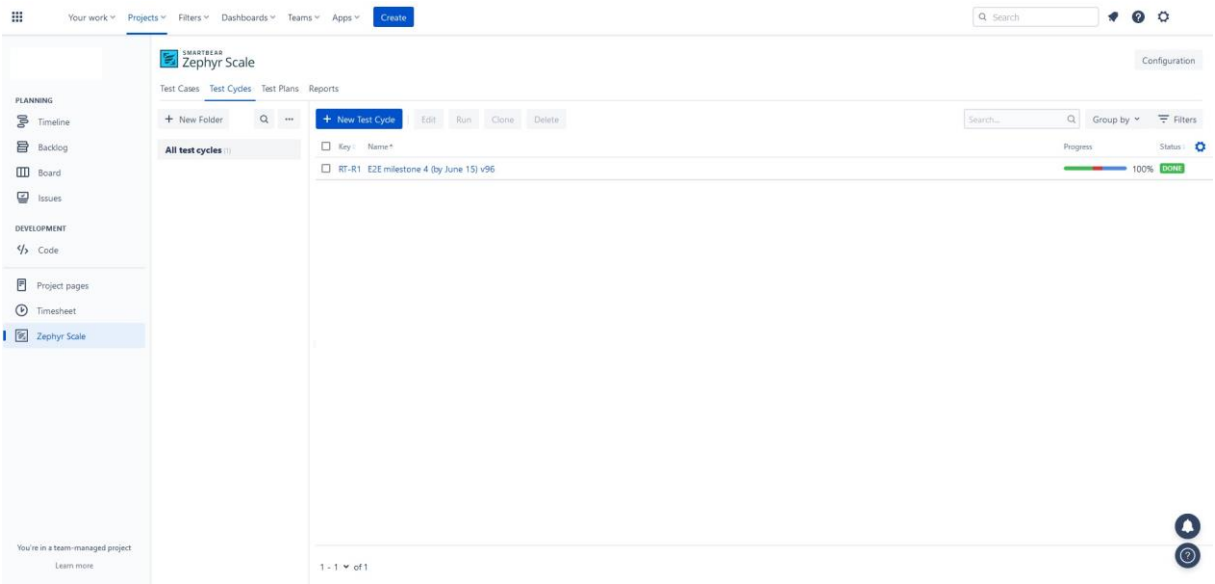


Kroz Zephyr Plugin, korisnici mogu stvarati, planirati i organizirati testove unutar Jire. Testovi se mogu grupirati u test planove i cikluse, što olakšava njihovu organizaciju i izvršavanje. Plugin omogućuje praćenje rezultata testiranja, uključujući evidenciju prolaznih i neuspješnih testova, kao i praćenje statusa testova tijekom vremena. Ovo je vidljivo na slijedećim slikama.

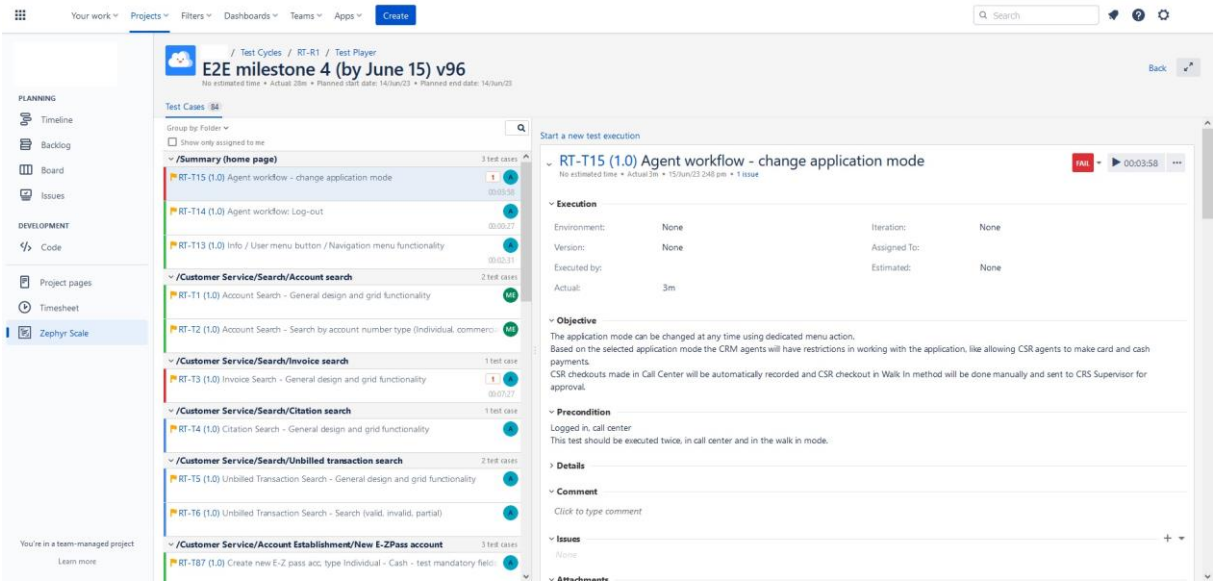
Slika 5.5. Zephyr test plan



Slika 5.6. Zephyr test cycle



Slika 5.7. Zephyr test cycle cases

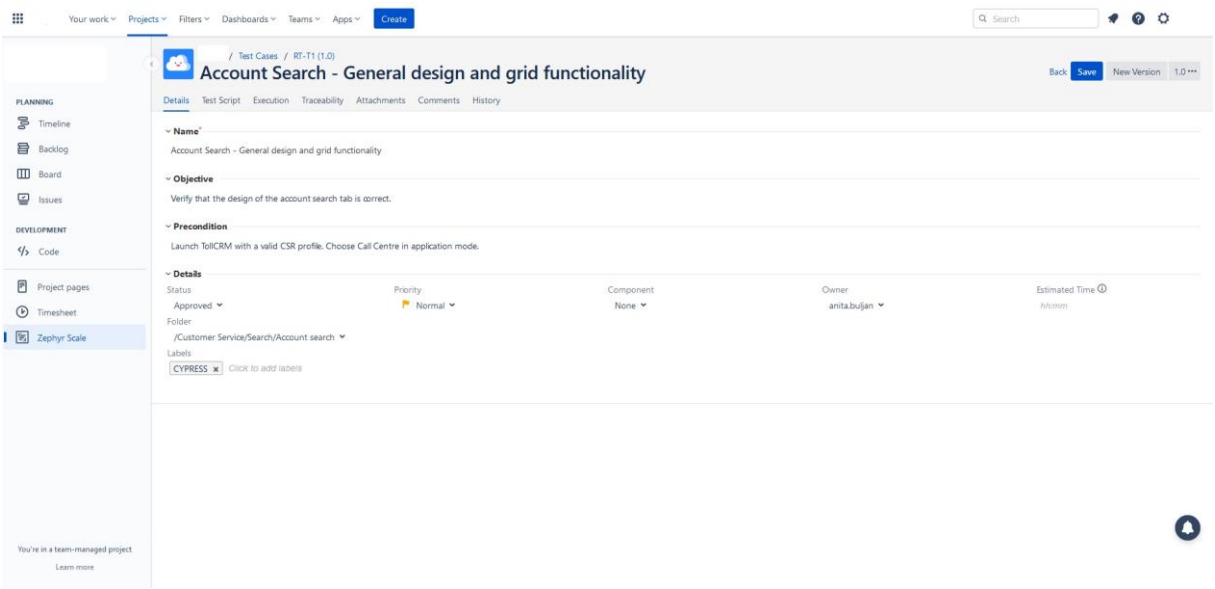


Izvor: tvrtka Tech Resources

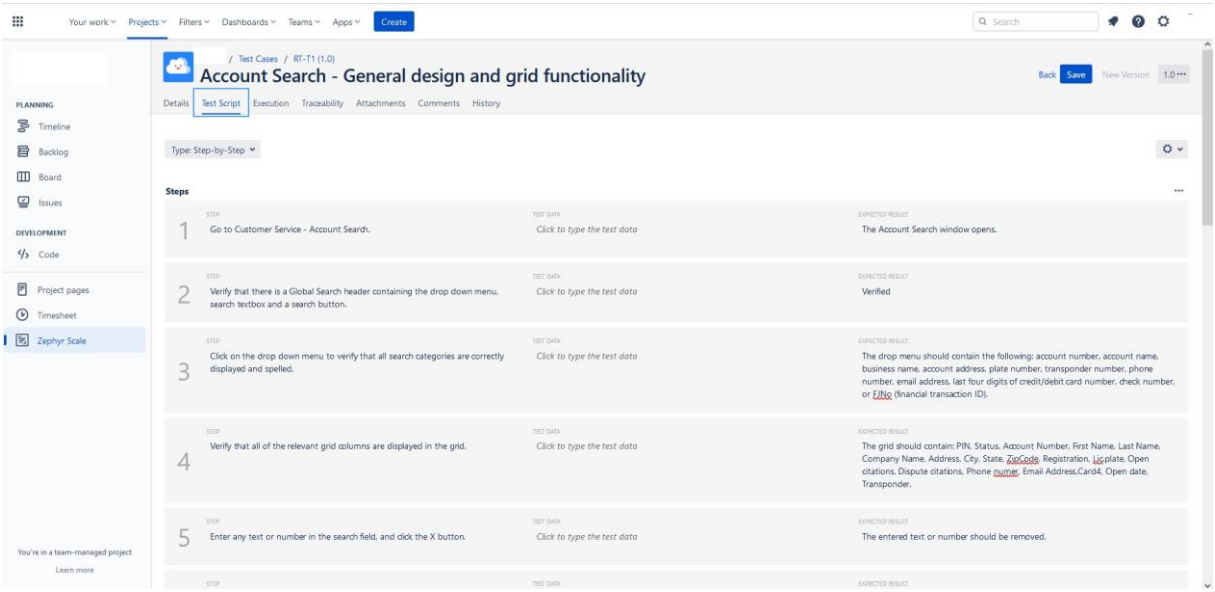
Zephyr Test Case je jedna od ključnih komponenti Zephyr alata za testiranje softvera koji se integrira s Jira platformom. Kroz Zephyr Test Case, korisnici mogu detaljno opisati korake koje je potrebno slijediti kako bi se izvršio test, kao i očekivane rezultate koji se trebaju postići. Test Case može sadržavati informacije poput naziva, opisa, prioritetne razine, odgovornih osoba i drugih relevantnih podataka. Test Case može biti organiziran u hijerarhijsku strukturu, gdje se mogu stvarati test planovi, ciklusi i iteracije kako bi se bolje organizirao proces testiranja. Tijekom izvršavanja testova, Zephyr

pruža funkcionalnost za praćenje statusa svakog Test Case-a, što omogućuje timovima da jasno vide napredak i rezultate testiranja. Slike prikazuju jedan primjer jednog Test Case-a.

Slika 5.8. Zephyr test case



Slika 5.9. Zephyr test case koraci



Izvor: tvrtka Tech Resources

5.3. PRIMJER AUTOMATSKOG TESTIRANJA

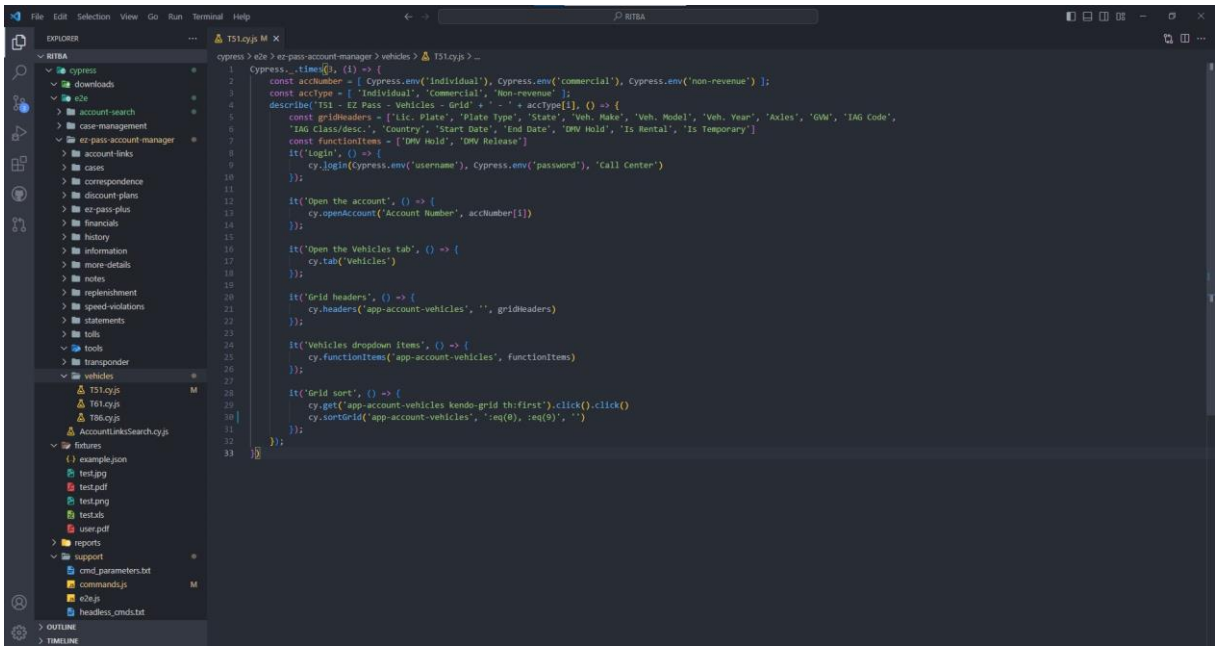
Automatsko testiranje vrši se u Cypressu. Cypress je alat za testiranje softvera otvorenog koda koji se koristi za automatizaciju testova web aplikacija. On nudi bogate funkcionalnosti koje olakšavaju pisanje, vođenje i održavanje testova. Cypress posjeduje jedinstvenu arhitekturu koja omogućuje izvršavanje testova izravno u pregledniku, što rezultira bržim i pouzdanijim izvedbama. S njegovim intuitivnim API-jem, programeri mogu jednostavno upravljati s elementima na stranici, provjeriti stanje i upravljati zahtjevima. API je skup definiranih pravila i protokola koji omogućuju komunikaciju između različitih softverskih komponenti. Real-time prikaz tijekom izvršavanja testova omogućuje vizualno praćenje događaja na stranici, olakšavajući pronalaženje i rješavanje problema.

Cypress omogućava:

1. Postavljanje testova
2. Pisanje testova
3. Pokretanje testova
4. Uklanjanje bugova u testovima

Kako bi testiranje moglo započeti, potrebno je napisati kod za test u Cypressu. Kroz pisanje koda, postiže se kontrola nad testiranjem, omogućujući precizno definiranje, izvršavanje i analiziranje testova. Slika prikazuje primjer koda za aplikaciju za plaćanje mostarina.

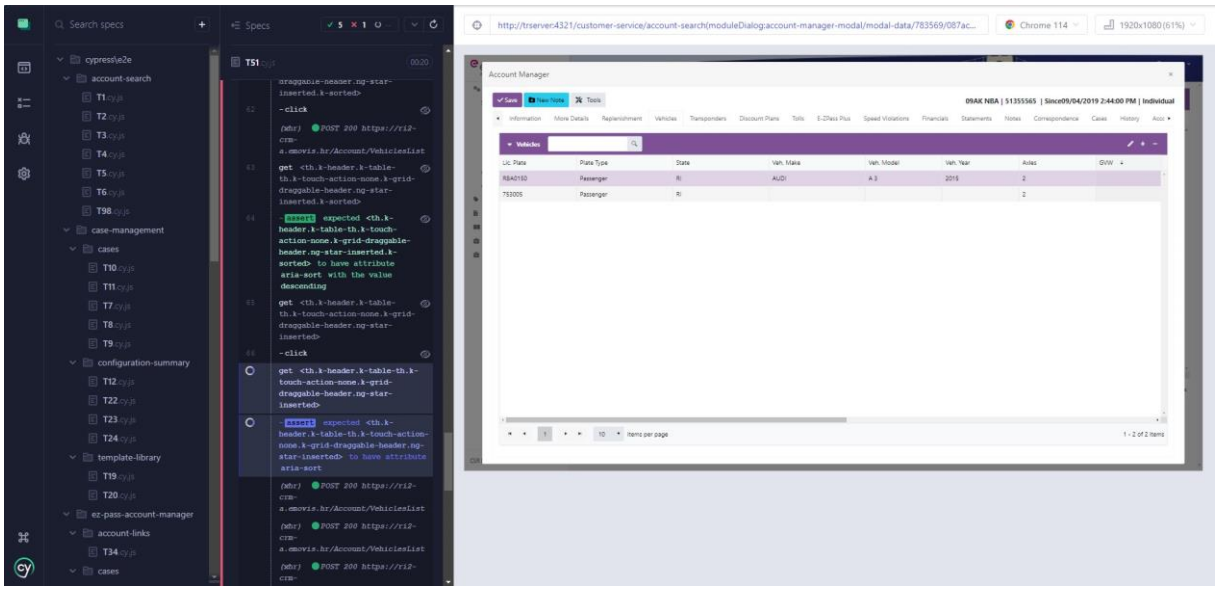
Slika 5.10. Cypress code



Izvor: Tvrtka Tech Resources

Cypress se često uspoređuje s Seleniumom; međutim, Cypress je temeljno i arhitektonski drugačiji. Cypress nije ograničen istim restrikcijama kao Selenium. Cypress ima potpunu kontrolu nad preglednikom i izvršava testove izravno u njemu. To omogućuje brže izvršavanje testova, precizniju kontrolu nad aplikacijom i lakše otkrivanje grešaka. Kada se pokrene test u Cypressu, on otvara preglednik i kontrolira ga putem svog vlastitog izvršnog procesa. To mu omogućuje potpunu kontrolu nad preglednikom tijekom izvršavanja testova. Cypress ima pristup cjelokupnom DOM-u web stranice koja se testira i omogućuje interakciju s elementima putem intuitivnog API-ja. Tijekom izvršavanja testova, Cypress pruža lako praćenje koraka testa u stvarnom vremenu na web stranici. To daje vizualni uvid u izvođenje testa i pomaže u pronalaženju mogućih problema. Cypress također ima funkcionalnost automatskog ponovnog pokretanja, što znači da će pokušati ponovno izvršiti akciju ako se dogodi pogreška, osiguravajući dosljedne rezultate. Integracija s DevTools preglednikom omogućuje korištenje naprednih alata za pregled, analizu i debugiranje testova. Slijedeća slika prikazuje program Cypress pokrenut u programu.

Slika 5.11. Cypress u pregledniku



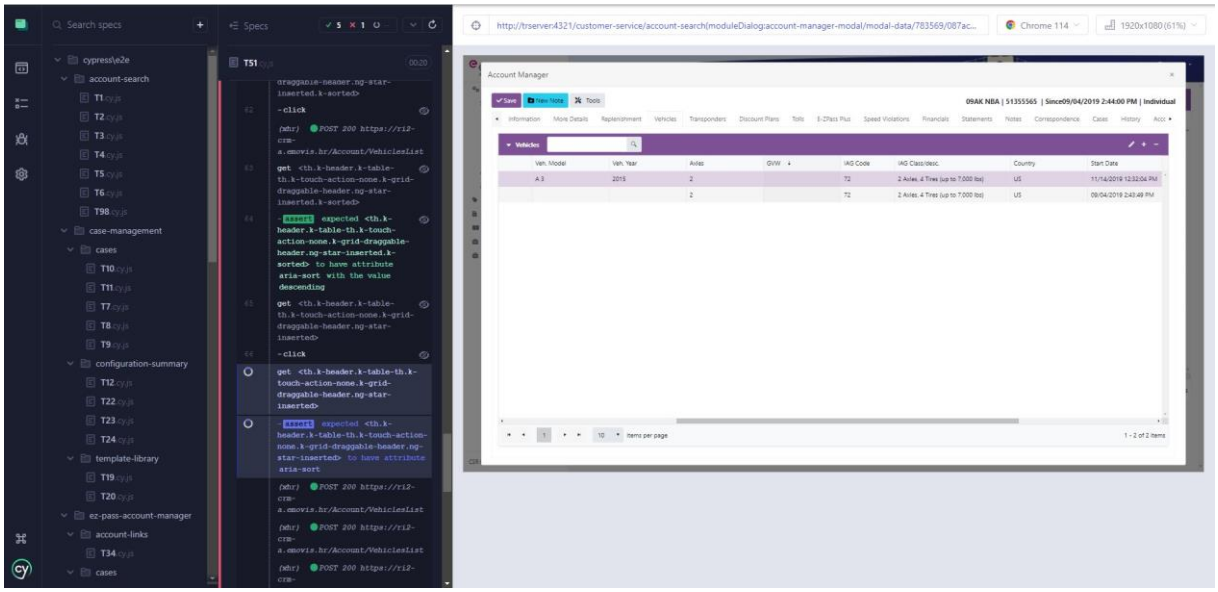
Izvor: tvrtka Tech Resources

Cypress koristi različite tehnike kako bi pronašao greške (bugove) na web stranici ili aplikaciji. Jedan od glavnih načina je kroz korištenje assertions (tvrdnji) koje provjeravaju očekivano ponašanje elemenata ili funkcionalnosti. Kada se test izvršava, Cypress će provjeriti da li se očekivani sadržaj prikazuje, da li se određeni elementi nalaze na stranici ili da li se događaju očekivane akcije. Ako tvrdnja ne uspije, Cypress će generirati grešku koja ukazuje na potencijalni bug. Također, Cypress koristi selektore za pronalaženje elemenata na web stranici. To mogu biti CSS selektori, XPath ili Cypress-ovi jedinstveni selektori poput cy.get(). Kroz selektore, Cypress može pronaći specifične

elemente s kojima želite interagirati ili koje želite testirati. Ako se selektor ne može pronaći ili ne odgovara očekivanom elementu, to može ukazivati na potencijalni bug. Dodatno, Cypress simulira korisničke akcije na web stranici, kao što su klikovi, unošenje teksta, slanje obrazaca i drugo. Tijekom ovih interakcija, Cypress prati rezultate i provjerava da li se događa očekivano ponašanje. Ako nešto ne ide kako je planirano, to može ukazivati na grešku ili bug.

Slika prikazuje primjer u kojem se pojavljuje bug jer zadana tvrdnja nije ispunjena.

Slika 5.12. Bug u Cypress-u



Izvor: tvrtka Tech Resources

5.4. USPOREDBA RUČNOG I AUTOMATSKOG TESTIRANJA

Sve veće razumijevanje i usvojenost automatizacije testiranja nerazmjerno je umanjila vrijednost ručnog testiranja. Sve prisutno je mišljenje da je ono zastarjela i nepotrebna praksa. Iako automatsko izvođenje testova ima svoje značajne prednosti, ručno testiranje i dalje ima neosporno važnu ulogu. Tulchak L.V. & Noskovenko Y.P. (2019.) navode da inicijalna automatizacija zahtjeva znatno više vremena i resursa. Pokušaj uvođenja automatizacije prema kraju ciklusa testiranja može biti beskoristan jer ostaje vrlo malo vremena za osiguranje pouzdanosti automatiziranog testiranja. U tom slučaju, resurse je bolje usmjeriti prema ručnom testiranju.

Automatizacija je vrlo podobna da sadrži greške. Automatizacijski skript s greškama može dovesti do pogrešnog tumačenja ispravnosti testirane aplikacije ili čak pogrešno interpretirati grešku kao

ispravno funkcioniranje. Ručno testiranje osnovnih funkcionalnosti osigurava da testni slučaj uspješno prolazi iz perspektive korisnika, bez prostora za krivo tumačenje. Također, neke situacije jednostavno nije moguće automatizirati. Nekad su to stvarne tehničke nemogućnosti ili složenost scenarija, dok ponekad troškovi automatizacije znatno nadmašuju troškove jednostavnog ručnog testa.

Iako je krajnji cilj automatizacije olakšavanje, postavljanje okvira i izrada skripti nisu nimalo jednostavni zadaci. Učinkovit tester treba posjedovati osnove programerskih vještina i duboko razumijevanje dizajna testiranja. Ove vještine se razvijaju kroz dugotrajno iskustvo u QA i razvoju, pri čemu pronalazak osobe sa specifičnim vještinama, posebno u ograničenom vremenskom okviru, nije jednostavan proces. S druge strane, većina ručnih testova je jednostavna za izvođenje i brzo se može usvojiti. Prateći korake u testnim slučajevima, provjerava se usklađenost stvarnih rezultata s očekivanim.

Automatizirano testiranje pruža niz prednosti u odnosu na tradicionalno ručno testiranje. Dvije osnovne koristi su povećana učinkovitost i veća preciznost. Kada se učinkovitost testiranja poveća, to ima dvostruki pozitivan učinak: smanjuju se troškovi i postiže se bolji povrat ulaganja. Prema Asfaw (2015.) ova dva aspekta su od suštinske važnosti za svaku organizaciju koja razvija softverske proizvode. Drugi važan faktor je razlika u preciznosti između ručnog i automatiziranog testiranja. Podaci pokazuju da razlika u točnosti između najboljeg ručno testiranog vremena i vremena provedenog u automatiziranom testiranju može doseći čak 70%. Ovo ukazuje na to da je ručno testiranje inherentno podložno većem broju grešaka u usporedbi s automatiziranim pristupom.

Jedna od prednosti automatiziranog testiranja je i smanjenje utjecaja ljudske greške. Iako se vještine testera mogu poboljšati tijekom vremena, greške su i dalje neizbježne. S druge strane, automatizirane skripte dosljedno izvode testne korake bez varijacija, što minimizira potencijalne pogreške. Unatoč napretku u ručnom testiranju, potpuna preciznost se rijetko postiže. Bez obzira na intelektualne vještine i pažnju, ručno testiranje ostaje podložno subjektivnosti i promjenjivim uvjetima. Uvođenje automatiziranih skripti u proces testiranja donosi više prednosti. To ne samo da povećava učinkovitost testiranja, već i smanjuje potrebu za velikim brojem radne snage. Također, smanjuje troškove održavanja, jer je brže i jednostavnije provesti testiranje.

Kombinacija ovih faktora značajno doprinosi boljem povratu ulaganja. Kroz automatizirano testiranje, organizacije postižu visoku razinu efikasnosti, dosljedne preciznosti i smanjenja troškova. Sve ovo zajedno rezultira većim zadovoljstvom klijenata, bržim razvojem softvera i uspješnijim postizanjem poslovnih ciljeva.

6. ZAKLJUČAK

Testiranje ključni proces u razvoju softvera koji ima važnu ulogu u osiguravanju kvalitete i pouzdanosti softverskih sustava. Testiranje omogućuje otkrivanje grešaka u performansama softvera, čime se smanjuje rizik od neuspjeha i poboljšava korisničko iskustvo.

U ovom radu su razmotreni različiti koraci u životnom ciklusu testiranja softvera (STLC) kao i osnovne principe testiranja. Analiza zahtjeva, planiranje testiranja, razvoj testnih slučajeva, izvršenje testiranja, evaluacija i nadzor, kao i održavanje, predstavljaju ključne faze u STLC-u koje treba pažljivo provesti kako bi se proces odvijao uspješno.

Također su prikazane sve osnovne vrste testiranja. Postoji širok spektar tipova testiranja koji se mogu koristiti u različitim fazama razvoja softvera. Svaki od njih doprinosi postizanju kvalitetnog softvera koji zadovoljava zahtjeve korisnika i poslovne ciljeve. Glavni naglasak stavljen je na ručno i automatsko testiranje. Ručno testiranje uključuje ručno izvršavanje testova od strane testera koji provjeravaju funkcionalnosti i performanse softvera. Ovo testiranje je fleksibilno i omogućuje testiranje kompleksnih scenarija, ali može biti vremenski zahtjevno i skupo.

S druge strane, automatsko testiranje koristi posebne alate i skripte kako bi se testovi izvršavali automatizirano. Ovo testiranje ima prednost brzine izvršavanja, ponovljivosti testova i smanjenja ljudske pogreške. Kombinacija ručnog i automatskog testiranja pruža najbolje rezultate u osiguravanju kvalitete softvera.

Svjesni činjenice da ljudi nisu savršena bića te da tijekom procesa razvoja proizvoda greške su neizbježne, postaje jasno koliko je važno testiranje u cjelokupnom procesu. U procesu razvoja softvera, greške se mogu javiti na različitim koracima. Iako neke greške mogu biti zanemarive, postoje slučajevi u kojima greške mogu imati ozbiljne posljedice koje su i financijski i reputacijski pogubne. Stoga je testiranje neizostavan dio razvojnog procesa svih proizvoda kako bi se osigurala njihova kvaliteta i sigurnost te smanjio rizik od mogućih negativnih posljedica.

7. SAŽETAK

Ovaj rad naglašava važnost testiranja u procesu razvoja programskog proizvoda. U početku rada se objašnjava koncept testiranja, opisuje se životni ciklus razvoja programskog proizvoda te se ističu ciljevi testiranja. Nadalje, obrađuju se metode i razine testiranja kako bi se pružio cjelovit uvid u tematiku. U sklopu rada su također prikazani primjeri ručnog i automatskog testiranja tvrtke Tech Resources, te alati u kojima se testiranja izvršavaju. Ovi primjeri pružaju praktičan uvid u način na koji se provodi testiranje i kako se mogu koristiti različite metode za postizanje željenih rezultata. Kroz primjere i istraživanje, ističe se kako ručno i automatsko testiranje mogu biti korisni alati za osiguranje kvalitete i funkcionalnosti softvera.

Ključne riječi: testiranje, razvoj programskog proizvoda, životni ciklus, metode testiranja, ručno testiranje, automatsko testiranje, alati, kvaliteta.

ABSTRACT

This paper emphasizes the importance of testing in the software development process. It begins by explaining the concept of testing, describing the software development life cycle, and highlighting the objectives of testing. Furthermore, it discusses different methods and levels of testing to provide a comprehensive understanding of the topic. The paper also includes examples of manual and automated testing conducted by Tech Resources, along with the tools used for executing the tests. These examples offer practical insights into the testing process and demonstrate how various methods can be employed to achieve desired results. Through the examples and research, it emphasizes how manual and automated testing can serve as valuable tools for ensuring the quality and functionality of software.

Keywords: testing, software development, life cycle, testing methods, manual testing, automated testing, tools, quality.

8. LITERATURA

Knjige:

1. Burnstein, I. (2003.): Practical software testing; a process-oriented approach, Springer-Verlag, New York
2. Manger, R. (2016.): Softversko inženjerstvo, Element, Zagreb

Članci i studije:

1. Arumugam, A., (2019.): Software Testing Techniques & New Trends, International Journal of Engineering Research & Technology (IJERT), Vol. 8 Issue 12, December,
2. Asfaw, A., (2015.): Benefits of Automated Testing Over Manual Testing, (IJIRIS) Issue 1, Volume 2 (January 2015), str. 9.-13.
3. Bhatti, I., Siddiq, J., Moiz, A., Memon, Z., (2019.): Towards Ad hoc Testing Technique Effectiveness in Software Testing Life Cycle, Sukkur IBA University
4. Dudas, J., (2019.): Testiranje programskih proizvoda, Varaždin, Sveučilište u Zagrebu, str. 39-40.
5. Džale, D., (2021.): Automatizacija testiranja ogledne aplikacije, Split, Sveučilište u Splitu, str. 28.
6. Eskić, E., (2019.): Testiranje programskih rješenja u mrežnom okruženju, Sveučilište J.J. Strossmayera u Osijeku, str.16.
7. Ivandić, M., (2017.): Važnost, tipovi i primjeri testiranja programskih proizvoda, Split, Sveučilište u Splitu, str. 15.
8. Kalaica, A., (2022.): Testiranje kao ključni dio razvoja programskog proizvoda, sveučilišni centar Varaždin
9. Lučić, M., (2019.): Prakse testiranja programskih proizvoda, Sveučilište u Zagrebu, Fakultet organizacije i informatike
10. Mamić, A., (2021.): Metode i tehnike testiranja softvera, Sveučilište u Zagrebu
11. Milin, A., (2023.): Usporedba ručnog i automatiziranog testiranja na primjeru web aplikacije "Swag Labs" koristeći alat "Playwright", Split, Sveučilište u Splitu, str 42.
12. Navaei, M., Tabrizi, N., (2021.): Machine Learning in Software Development Life Cycle: A Comprehensive Review, East Carolina University, str. 344.-354.
13. Pleše, D., (2020.): Životni ciklus testiranja softvera, Sveučilište u Rijeci – Odjel za informatiku
14. Quadri, S.M.K., Farooq, S. (2010.): Software Testing – Goals, Principles, and Limitations, University of Kashmir (India), Volume 6– No.9, September, str. 7.-9.
15. Ranjan, R., (2019.): A Review Paper on Software Testing, Galgotias University, January, Volume 6, Issue 1, str. 25.-32.

16. Sawant, A., Bari, P., Chawan, P. (2012.): Software Testing Techniques and Strategies, University of Mumbai, Vol. 2, Issue 3, May-June, str. 980-986
17. Sharma, R. M., (2014.): Quantitative Analysis of Automation and Manual Testing, International Journal of Engineering and Innovative Technology (IJEIT). Volume 4, Issue 1, July, str. 252.-257.
18. Stankić, N., (2021.): Automatsko testiranje programskih rješenja na mreži, Osijek, Sveučilište Josipa Jurja Strossmayera u Osijeku, str 20.-22.
19. Tulchak, L.V., Noskovenko, Y.P., (2019.): Why manual testing remains relevant, Vinnytsia National Technical University
20. Yu B. L.,Wooi K. L., Wai Y. T., Soo F. T.: Software Development Life Cycle AGILE vs Traditional Approaches, IPCSIT vol. 37, str. 162.-167.

Internet izvori:

1. Cypress
   Dostupno na: https://www.cypress.io/ (datum pristupa: 12.7.2023.)
2. Hamilton, T., (2023): What is Regression Testing? Test Cases Example
   Dostupno na : https://www.guru99.com/regression-testing.html (datum pristupa 21.7.2023.)
3. Jira Software
   Dostupno na: https://www.atlassian.com/software/jira (datum pristupa: 12.7.2023.)
4. Software Testing | Dynamic Testing
   Dostupno na: https://www.geeksforgeeks.org/software-testing-dynamic-testing/ (datum pristupa: 11.7.2023.)
5. Što je integracijsko testiranje? Duboko zaronite u vrste, procese i implementaciju
   Dostupno na: https://www.zaptest.com/hr/sto-je-integracijsko-testiranje-duboko-zaronite-u-vrste-procese-i-implementaciju (datum pristupa 10.7.2023.)
6. What is Dynamic Testing? (Types and Methodologies)
   Dostupno na: https://www.browserstack.com/guide/dynamic-testing (datum pristupa 1.8.2023.)
7. Zephyr
   Dostupno na: https://zephyrproject.org/ (datum pristupa: 12.7.2023.)

Popis slika: