Testiranje kao ključni dio razvoja programskog proizvoda

Kalaica, Ana

SVEUČILIŠTE SJEVER
SVEUČILIŠNI CENTAR VARAŽDIN

DIPLOMSKI RAD br. 086-MMD-2022

TESTIRANJE KAO KLJUČNI DIO RAZVOJA PROGRAMSKOG PROIZVODA

Ana Kalaica

Varaždin, rujan 2022.

SVEUČILIŠTE SJEVER

SVEUČILIŠNI CENTAR VARAŽDIN

Studij Multimedija

DIPLOMSKI RAD br. 086-MMD-2022

TESTIRANJE KAO KLJUČNI DIO RAZVOJA PROGRAMSKOG PROIZVODA

Student:
Ana Kalaica, 1627/336

Mentor:
doc. art. dr. sc. Robert Geček

Varaždin, rujan 2022.

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

UNIVERSITY NORTH

Prijava diplomskog rada

Definiranje teme diplomskog rada i povjerenstva

ODJEL Odjel za multimediju

STUDIJ diplomski sveučilišni studij Multimedija

PRISTUPNIK Ana Kalaica | JMBAG 0336014300

DATUM 27.09.2022. | KOLEGIJ Web dizajn i produkcija

NASLOV RADA Testiranje kao ključni dio razvoja programskog proizvoda

NASLOV RADA NA ENGL. JEZIKU Testing as a key part of software product development

MENTOR Robert Geček | ZVANJE doc.art.dr.sc.

ČLANOVI POVJERENSTVA

1. doc.dr.sc. Andrija Bernik - predsjednik
2. doc.dr.sc. Domagoj Frank - član
3. doc.art.dr.sc. Robert Geček - mentor
4. izv.prof. dr.sc. Emil Dumić - zamjenski član
5.

Zadatak diplomskog rada

BROJ 086-MMD-2022

OPIS

S kontinuiranim razvojem programskih tehnologija, dolazi do sve kompliciranijih programskih rješenja. Kako bi programski proizvodi mogli konkurirati ostatku tržišta, danas je norma u razvoju programskog proizvoda osiguravanje kvalitete (eng. Quality assurance; QA).
Testiranje je jedna od metoda QA-a koja osigurava kvalitetu programskog proizvoda tako što testira njene specifikacije, funkcionalnosti i planirane ishode koje su dogovorene pri planiranju razvoja samog programa.

U radu će se opisati povijest testiranja, životni ciklus testiranja te ciljevi testiranja.
Zatim, obraditi će se metodologija testiranja koja obuhvaća metode testiranja i razine testiranja.
Nakon toga, objasniti će se tipičan primjer ručnog i automatskog testa.
Za kraj, provesti će se istraživanje u kojemu će se analizirati i usporediti podaci iz ručnih i automatskih testova za BizDataX program. Cilj ovog rada je naglasiti važnost testiranja u procesu izrade programskog proizvoda.

REPUBLIKA HRVATSKA 4 SVEUČILIŠTE SJEVER KOPRIVNICA

ZADATAK URUČEN 28.09.2022. | POTPIS MENTORA

Predgovor

Testiranje bilo kojeg proizvoda je temeljni aspekt osiguravanja kvalitete proizvoda. Iz vlastite želje da povežem svoju trenutnu poslovnu poziciju kao QA tester i diplomski studij, posvetila sam ovaj diplomski rad testiranju kao ključnim dijelom razvoja programskog proizvoda.

Zahvaljujem se svom mentoru, doc. art. dr. sc. Robertu Gečeku, na pruženoj pomoći tijekom studiranja i tijekom izrade ovog rada.

Na kraju, zahvaljujem se svojoj obitelji i svom dečku, koji su mi pružali podršku i razumijevanje tijekom studiranja.

Sažetak

Ovaj rad fokusira se na važnosti testiranja u razvoju programskog proizvoda. Na početku rada objašnjava se pojam testiranja, kako izgleda životni ciklus u razvoju programskog proizvoda te koji su to ciljevi testiranja. Nakon toga, navode se i obrađuju metode i razine testiranja. Zatim, izrađeni su jednostavni primjeri ručnog i automatskog testa koji su dodatno objašnjeni. U istraživačkom dijelu rada, prvo se opisuje program BDX i princip razvoja programa na projektu. Zatim, izvršavaju se ručni i automatski testovi na jednoj funkcionalnosti programa, kako bi donesli zaključak o uporabljivosti svake od navedenih metoda.

Ključne riječi: QA, osiguravanje kvalitete, testiranje, razvoj programa, pogreška, greška, kvar, ručni, automatiziran, test

Summary

This paper focuses on the importance of testing in the development of a software product. At the beginning of the paper, the concept of testing is explained, what the life cycle looks like in the development of a software product and what are the goals of testing. After that, testing methods and levels are listed and discussed. Then, simple examples of manual and automatic tests were created, which were then further explained. In the research part of the paper, the BDX program and the principle of program development on the project are described. Then, manual and automatic tests are performed on one functionality of the program, in order to reach a conclusion on the usability of each of the mentioned methods.

Keywords: QA, quality assurance, testing, software development, bug, failure, defect, error, fault, manual, automatized, test

Popis korištenih kratica i oznaka

QA Quality Assurance – Osiguravanje kvalitete

SDLC Software Development Life Cycle – Životni ciklus razvoja programa

STLC Software Testing Life Cycle – Životni ciklus testiranja programa

BDX BizDataX – program za maskiranje osjetljivih podataka

GDPR General Dana Protection Regulation - Opća uredba o zaštiti podataka

UI User Interface – Korisničko sučelje

Sadržaj

1. Uvod

Testiranje je neizostavan dio razvojnog procesa bilo koje vrste proizvoda. Kako bi proizvođač osigurao kvalitetu svog proizvoda i zadovoljio potrebe korisnika, primoran je testirati svoj proizvod prije plasiranja na tržište. Stvari koje se moraju testirati definirane su vrstom proizvoda i njegovim funkcionalnostima. U slučaju da testiramo žarulju prioritet bi postao testiranje svijetli li žarulja ili ne. Kod programskih proizvoda testovi se mogu provoditi na više razina razvojnog procesa koristeći razne metode. Metode testiranja se odabiru sukladno sa funkcionalnostima koje se žele testirati na proizvodu. Odabrana metoda testiranja može imati veliki utjecaj na količinu vremena i novca koje je potrebno utrošiti u testiranje proizvoda. Iz ovog razloga krucijalno je odabrati adekvatnu metodu testiranja. Svaka metoda testiranja ima svoje prednosti i mane. Zbog toga se tijekom razvojnog procesa najčešće koristi kombinacija metoda testiranja kako bi se što temeljitije testiralo programski proizvod.

Cilj ovoga rada je definirati važnost testiranja u razvojnom procesu programskog proizvoda i ustanoviti u kojim slučajevima je optimalnije koristiti ručne testove a u kojima automatske testove. Kako bi se naglasila važnost testiranja obrađuje se životni ciklus razvoja programskog proizvoda i objašnjava se uloga testiranja u njemu. Nakon toga se kreće sa obradom metoda testiranja i njihovih karakteristika. Objašnjava se način provođenja metoda testiranja i navodi se dijelovi razvoja programskog proizvoda u kojima se mogu koristiti. Nakon objašnjavanja metoda testiranja kreće se sa obradom pojma razine testiranja. Razine testiranja govore na kojim razinama proizvoda se mogu provesti testovi tokom razvojnog procesa. Svaka razina ima svoje prednosti i posebne karakteristike. U radu se zatim daje primjer ručnog i automatskog testa. Testni slučaj, testni scenarij i preduvjeti testa su definirani pri vrhu prikaza testova, a neposredno ispod toga su objašnjeni svi koraci testa. U istraživačkom dijelu rada prvo se govori o programu BDX i o njegovim razvojnim procesima. Potom se objašnjava cilj istraživanja i opisuje se njegov sadržaj. Rezultati istraživanja se iznose i definiraju se prednosti i mane ručnog i automatskog testiranja. Na kraju rada se donosi zaključak baziran na rezultatima provedenog istraživanja.

2. Testiranje programskog proizvoda

S kontinuiranim razvojem programskih tehnologija, dolazi do sve kompliciranijih programskih rješenja. U industriji razvoja programa, manje kompanije često imaju krivi pristup kako bi se probili na tržište, takozvana kvantiteta naspram kvalitete. Cilj takvih kompanija je što brža isporuka proizvoda kako bi konkurirali tržištu, no to prouzrokuje greške koje mogu uvelike umanjuju kvalitetu programskog proizvoda. Kako bi programski proizvodi mogli konkurirati ostatku tržišta, danas je norma u razvoju programskog proizvoda osiguravanje kvalitete (*eng. Quality Assurance*; *QA*). Testiranje je jedna od metoda QA-a koja osigurava kvalitetu programskog proizvoda tako što testira njene specifikacije, funkcionalnosti i planirane ishode koje su dogovorene pri planiraju razvoja samog programa [1].

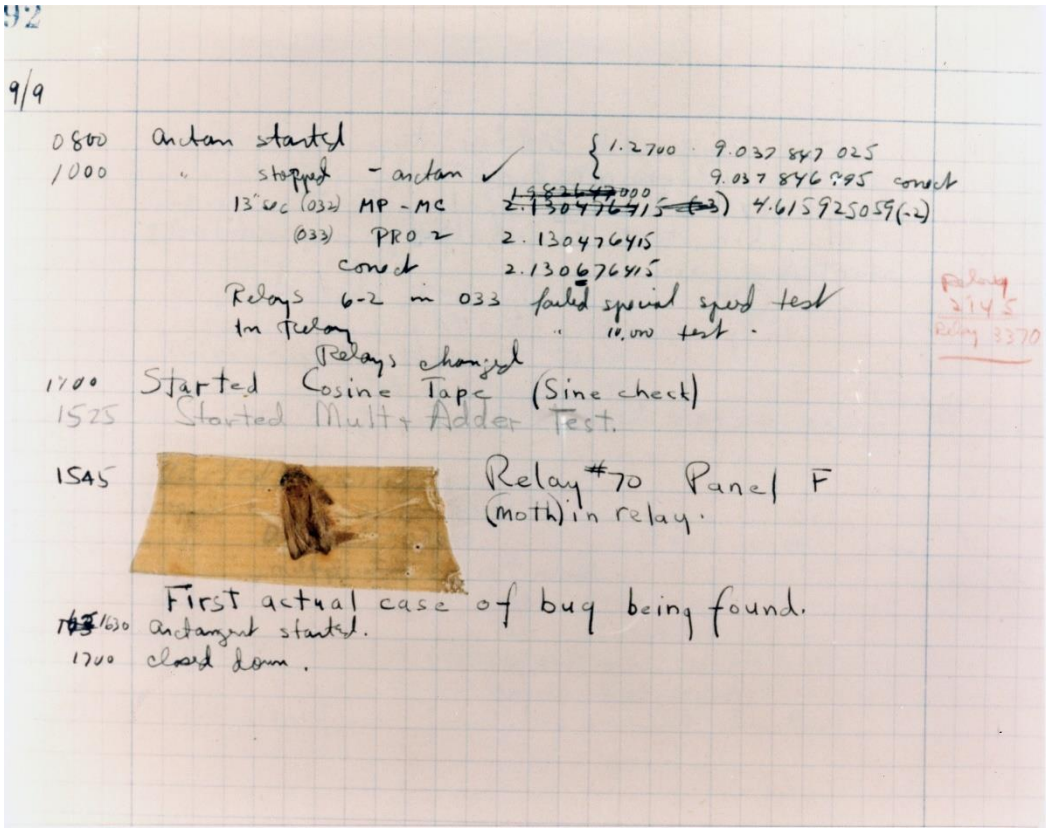
Testiranje programskog proizvoda je proces izvršavanja programa sa ciljem pronalaska skrivenih pogrešaka u programu [2]. Osim pronalaska pogreški, testiranje omogućava daljnje napredovanje funkcionalnosti i upotrebljivosti programa.

Pozicija testera u razvoju programskog proizvoda zahtjeva, osim tehničkih predispozicija, kritičko i analitičko razmišljanje iz perspektive korisnika programa. Kao tester, potrebno je pratiti da li program ispunjava zadane funkcionalnosti i obrasce ponašanja koje su prethodno definirane dokumentacijom. Također, ako neka funkcionalnost nema konkretnu pogrešku, tester može savjetovati razvojni tim kako unaprijediti tu određenu funkcionalnost, kako bi se korisnik lakše snalazio u programu.

2.1. Povijest testiranja

Testiranje se razvijalo kroz povijest zajedno sa potrebama tadašnjih tehnologija. David Gelperin i Bill Hetzel podijelili su povijest testiranja na pet značajnijih razdoblja. Prvo razdoblje je nazvano razdobljem koje je orijentirano na otklanjanje pogrešaka. Glavni cilj u tom razdoblju je bilo testiranje i otklanjanje pogrešaka isključivo iz hardvera. Jedan od prvih digitalnih računala je razvijen 1947. godine na Harvard sveučilištu, naziva Mark II.

Sa pojavom prvih digitalnih računala, pojavila se i prva pogreška. Naime, računala tog doba su bila puno većih proporcija nego danas, stoga i nije tako čudno da je jedan moljac pronašao put unutar računala i uzrokovao grešku, upravo to je razlog zašto dan danas nazivamo greške unutar programa *Bug*-ovima (eng. *Bug*, hrv. *kukac*, *pogreška*) [3]. Slika 2.1 prikazuje izvještaj (eng. *Logbook*) u kojem su Howard Aiken i Grace Hopper 9. rujna 1947. zapisali bilješku: „Prvi pravi slučaj pronalaska pogreške (kukca)“. 1949. godine, izdan je članak o provjeri programa, koji opisuje potrebu za dokazivanjem kvalitete samog programa, a napisao ga je Alan Turing [4].



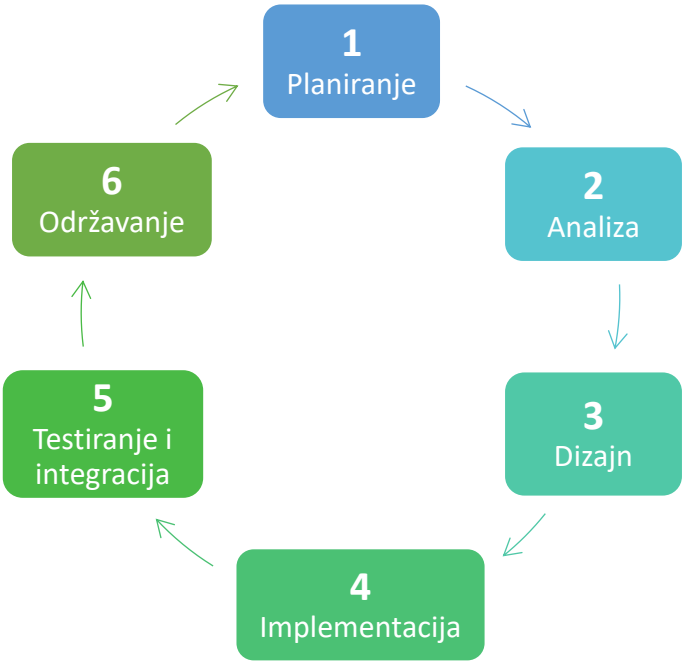
Slika 2.1 Bilješka o prvom bug-u, Howard Aiken i Grace Hopper

Od 1957. do 1978. godine odvija se drugo razbolje testiranja, takozvano demonstracijski orijentirano razdoblje. U tom periodu je utemeljena razlika između testiranja i uklanjanja pogrešaka, kao dvije zasebne aktivnosti. Glavni cilj testiranja je bio osigurati zadovoljenje programskih zahtjeva i specifikacija. Od 1979. do 1982. godine proteže se treće razdoblje testiranja, razdoblje orijentirano na uništavanje. Točnije, cilj testiranja u ovom razdoblju je rastavljanje koda na manje cjeline kako bi se pronašle pogreške. U tom razdoblju, Glenford J. Myers je opisao testiranje kao proces izvršavanja programa sa ciljem pronalaženja pogrešaka [5].

Od 1983. do 1987. godine nalazi se četvrto razbolje, takozvano razbolje orijentirano na evaluaciju. Glavni cilj testiranja je bilo na definiranje smjernica za procjenu i mjerenju kvalitete programa. Od 1988. do 2000. godine odvija se i posljednje razdoblje testiranja, takozvano preventivno orijentirano razdoblje. Ovo razdoblje je definirano sa pripremom najbolje tehnike testiranja prije samog testiranja, potrebno je razumjeti rad programa kako bi se što više grešaka pronašlo. Poslije 2000. godine, pojam testiranja dobiva novi značaj sa razvojem alata za automatizaciju testova. Danas, sa sve bržim rastom programskih tehnologija, umjetna inteligencija kreće zauzimati svoje mjesto u procesu testiranja [6].

2.2. Životni ciklus razvoja programskog proizvoda

Životni ciklus razvoja programskog proizvoda je naziv koji opisuje životni vijek i faze razvoja programskog proizvoda. Faze životnog ciklusa se odvijaju od početka razvoja do implementacije te faze održavanja proizvoda (eng. *Software Development Life Cyle; SDLC*). Slika 2.2 opisuje navedeni proces. SDLC je zapravo sustavna metoda razvoja programa, koja osigurava bržu isporuku i kvalitetu programa.



Slika 2.2 Faze životnog ciklusa u razvoju programskog proizvoda

Razvoj programskog proizvoda je podijeljen u 6 faza:

1. Planiranje
2. Analiza zahtjeva
3. Dizajn arhitekture programa
4. Implementacija
5. Testiranje i integracija
6. Održavanje

U prvim fazama razvoja programskog proizvoda odvija se planiranje i analiza zahtjeva. Važno je imati dobre temelje na početku samog razvoja kako ne bi došlo do kasnijih komplikacija tokom razvoja. Tijekom faze planiranja i analize zahtjeva potrebno je definirati sve potrebne faktore kako bi obuhvatili projekt u cijelosti, poput neophodnih funkcionalnosti, vremenskih rokova te sam opseg projekta. Specifikacije samog projekta se upisuju u projektnu dokumentaciju. Dizajniranje arhitekture programa se razvija prema projektnoj dokumentaciji koja je napravljena u prethodne dvije faze [7]. Dokumentacija sa dizajnom arhitekture programa sadrži ključne informacije kao što su:

- tehničke specifikacije
- odnosi između različitih dijelova programa
- definiranje funkcionalnosti programa
- definiranje baze podataka

U fazi implementacije počinje razvoj programa, konkretnije kodiranje ili programiranje. Implementacija određene funkcionalnosti zahtjeva suzbijanje većeg modula u više manjih zadataka, kako bi razvojni tim mogao paralelno raditi te time biti efikasniji u svom poslu. Ova faza je ujedno i najduža faza u razvoju programskog proizvoda.

Faza testiranja i integracije je tema ovoga rada stoga će se obraditi detaljnije u slijedećim poglavljima. U samom početku faze testiranja potrebno je izraditi testne scenarije, stoga sa fazom testiranja se može započeti i prije same implementacije funkcionalnosti, točnije odmah nakon što je napisana projektna dokumentacija. Nakon što je funkcionalnost implementirana, može se započeti sa izvršavanjem testova. Tijekom testiranja potrebno je pronaći pogreške i nedostatke u programu te ih onda prijaviti razvojnom timu. Ciklus testiranja i prijavljivanja pogrešaka se ponavlja dokle god se pogreške i dalje ponavljaju. Nakon što su uklonjene sve greške te je program u skladu sa projektnom dokumentacijom i specifikacijama, faza testiranja je završena [8].

Zatim počinje integracija funkcionalnosti u proizvodno okruženje programa. Faza održavanja je ključna faza za održavanje kvalitete programskog proizvoda. Prilikom korištenja programskog proizvoda, korisnik može imati raznih problema koje mogu biti uzrokovane okruženjem, nadogradnjom sustava, krivim korištenjem, itd. U ovoj fazi se izvršavaju poboljšanja i nadogradnje programa prema zahtjevima naručitelja.

2.3. Životni ciklus testiranja programskog proizvoda

Životni ciklus testiranja programskog proizvoda je naziv koji opisuje životni vijek i faze testiranja programskog proizvoda. Faze životnog ciklusa se odvijaju od analize zahtjeva do izvršavanja testova te faze zatvaranja testiranja (eng. *Software Testing Life Cyle; STLC*). Slika 2.3 opisuje navedeni proces. STLC je proces testiranja koji sadrži aktivnosti za osiguravanje kvalitete programskog proizvoda.

Slika 2.3 Faze životnog ciklusa u testiranju programskog proizvoda

Testiranje programskog proizvoda je podijeljeno u 6 faza:

1. Analiza zahtjeva
2. Planiranje testiranja
3. Izrada testnih scenarija
4. Testiranje okruženja
5. Izvršavanje testova
6. Zatvaranje testiranja

Faza analize zahtjeva služi kako bi se u početku testiranja odmah definirale nejasnoće i spriječili budući problemi tokom razvoja programskog proizvoda. Potrebno je analizirati zahtjeve u dokumentaciji zajedno sa naručiteljima programa, poslovnim analitičarima, razvojnim timom, itd. Zahtjevi mogu biti poslovni iz projektne dokumentacije, arhitektonski i dizajnerski te sistemski i integracijski. Kod arhitektonskih i dizajnerskih zahtjeva nalaze se detaljnije informacije o cjelokupnom dizajnu od poslovnih zahtjeva [7]. Sistemski i integracijski zahtjevi imaju još detaljnije opise, i to u obliku korisničkih priča (eng. *User Story*; *US*).

Preduvjet ovoj fazi je specifikacija zahtjeva:

- prikupljanje podataka putem dokumentacije
- određivanje prioriteta
- analiza i planiranje testova
- definiranje mogućnosti automatiziranih testova

Kod faze planiranja razvija se strategija i tijek testiranja, a to se postiže sa sljedećim aktivnostima:

- definiranje željenoga rezultata testiranja
- definiranje preduvjeta za testno okruženje
- definiranje preduvjeta i ograničenja u testiranju
- definiranje rasporeda izvršavanja testiranja

U fazi izrade testnih scenarija potrebno je pokriti što više mogućih testnih slučajeva. Testni scenariji su sastavljeni se od preduvjeta, testnih koraka i očekivanih rezultata. Testni koraci su doslovne upute kojih se treba pridržavati kako bi svaki test bio izvršen na konzistentan i korektan način. Prilikom izvršavanja određenog testnog koraka, potrebno je obratiti pažnju na sve funkcionalnosti i opisati u testnom scenariju koji je očekivani rezultat za svaki korak. Što je testni scenarij detaljniji, lakše će se uočiti i spriječiti buduće pogreške prilikom ponavljanja testa. Kada se testni scenarij sastoji od jednostavnijih koraka, onda taj testni scenarij može poslužiti u budućnosti kao predložak za automatski test [9].

Testiranje okruženja je faza u kojoj se vodi briga o stanju okruženja u kojem je potrebno izvršiti test. Potrebno je osigurati da su sve komponente, od hardvera do softvera, spremne za rad u realnom okruženju. Ako se testira program, potrebno je pratiti dokumentaciju prilikom instalacije programa na računalo, kako bi provjerili usput da li je dokumentacija razumljiva korisniku. Nakon što je testno okruženje postavljeno, testiranje može započeti [9].

Faza izvršavanja testova je ključni dio testiranja. Prilikom izvršavanja testa, uspoređuju se očekivani i dobiveni rezultati. Testni slučaj je prošao (eng. *Passed*) ako su očekivani i dobiveni rezultat u svakom testnom koraku jednaki. Ako postoji razlika između očekivanog i dobivenog rezultata na bilo kojem testnom koraku, označavamo pad (eng. *Failed*) na testu. U slučaju da se test ne može izvršiti radi neispunjenog preduvjeta za testiranje, tada je testni slučaj blokiran (eng. *Blocked*) [5]. Kada je pronađena greška, ona se prijavljuje programeru koji je implementirao tu funkcionalnost te se detaljno raspisuju koraci i komponente koje su korištene tokom izvršavanja samog testa, kako bi programer mogao imati istu reprodukciju te greške u svom okruženju. Nakon što je greška ispravljena, testiranje se ponavlja sa svim testnim scenarijima kako bi utvrdili da li je ispravak greške potencijalno utjecao na rad drugih funkcionalnosti.

Zatvaranje testiranja je posljednja faza. Testiranje može biti završeno samo kada su sve greške ispravljene. Razvojni tim procjenjuje jesu li sve stavke pokrivene i jesu li ciljevi testiranja ispunjeni. Također, pišu se i izvještaji rezultata testova kako bi se testiranje poboljšalo za buduće iteracije.

Tablica 2.1 prikazuje osnovne razlike između životnog ciklusa razvoja programa i životnog ciklusa testiranja programa. Životni ciklusi u konačnici predstavljaju tijek aktivnosti kroz koje određeni entitet prolazi.

Životni ciklus razvoja programa	Životni ciklus testiranja programa
Nadređeni proces	Dio je životnog ciklus razvoja programa
Temelji se na projektnoj dokumentaciji	Temelji se na testnim scenarijima
Cilj je razviti kvalitetan program	Cilj je pronaći pogreške

Tablica 2.1 Osnovne razlike između SDLC-a i STLC-a [10]

2.4. Ciljevi testiranja

Osnovni cilj testiranja je pronalazak pogrešaka u programu. No, osim pogrešaka, potrebno je utvrditi da li je program u skladu sa predefinirani zahtjevima. WTestiranje je postupak evaluacije programa ili pojedine komponente programa ručnim ili automatiziranim alatima s ciljem utvrđivanja zadovoljava li taj program određene zahtjeve.“ [7].

U testiranju, često se spominje pojam pogreška. No, pogreška je zapravo univerzalni pojam kojega možemo svesti na nekolicinu konkretnijih pojmova:

- greška (eng. *Error*)
- kvar (eng. *Bug*/*Fault*/*Defect*)
- vanjska pogreška (eng. *Failure*)

Pojam greška koristi se u slučajevima kada je greška uzrokovana ljudskim utjecajem. Greška može biti uzrokovana kodom, ali moguće je da je nastala tokom ranijih faza razvoja programa.

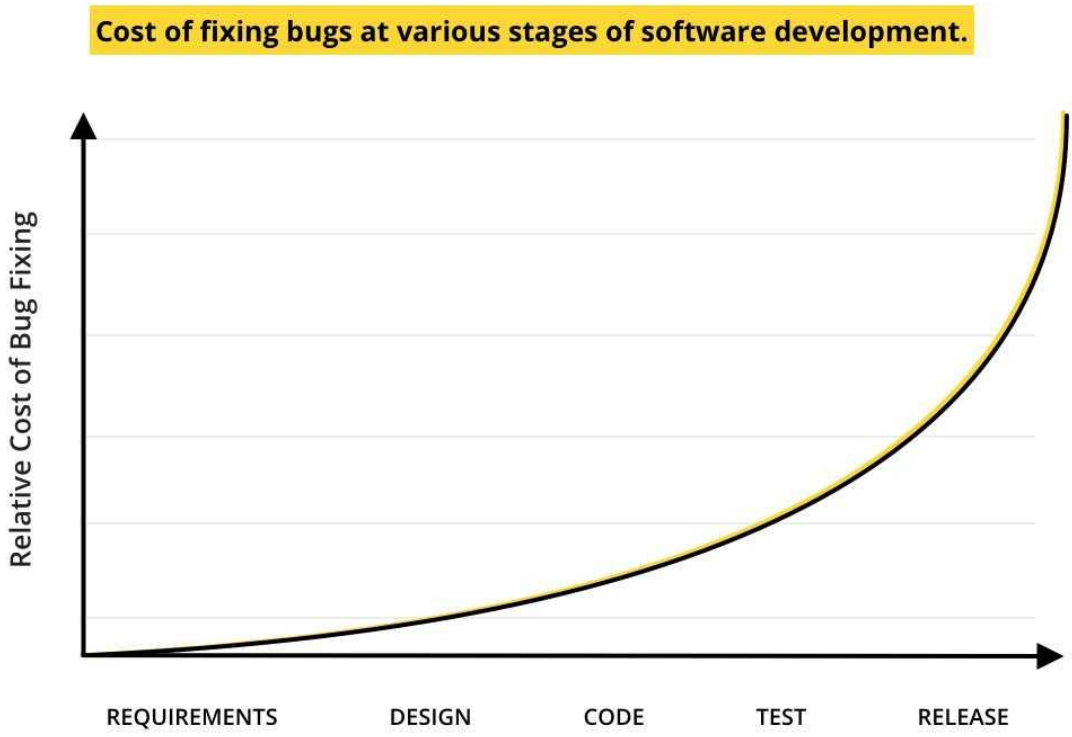
Pojam kvar označava situaciju kada program ne izvršava željene funkcije te su dobiveni rezultati različiti od očekivanih rezultata. Kada je u pitanju vanjski kvar, onda se opisuje kvar u obliku krivog korištenja programa ili krivog okruženja, to jest hardvera koji nije prilagođen programu.

Pojam vanjska pogreška definira pogrešku koja je uzrokovana kvarom. U konačnici, vanjska pogreška predstavlja nepravilnosti u radu programa koje nisu u skladu sa specifikacijama.

Najvažniji ciljevi testiranja su:

- prevencija pogrešaka
- otkrivanje pogrešaka
- osiguranje kvalitete
- zadovoljstvo krajnjeg korisnika

Cilj prevencije pogrešaka je predviđanje mogućih grešaka koje bi se mogle pojaviti tokom razvoja. S prevencijom pogrešaka uvelike se smanjuju troškovi testiranja. Prilikom testiranja, bitno je pronaći pogreške u što ranijoj fazi razvoja [11]. Kako bi osigurali kvalitetu programskog proizvoda, potrebno je da program ispunjava zadane zahtjeve. Na kraju je najbitnije zadovoljstvo krajnjeg korisnika, stoga treba voditi brigu o korisničkom iskustvu (eng. *User experience*) prilikom korištenja programa. Slika 2.4 (vlasništvo: *mcro.tech.com*) prikazuje relativni rast cijene ispravka pogrešaka, ovisno u kojoj fazi su pogreške otkrivene prilikom razvoja programa.



Slika 2.4 Cijena ispravka pogrešaka u različitim fazama razvoja programa

Postoje 7 načela testiranja:

1. „Testiranjem se prikazuje prisutnost pogrešaka, a ne njihova odsutnost. Testiranjem se smanjuje vjerojatnost pojave pogrešaka, no to ne znači da je program u potpunosti pouzdan ako pogreške nisu pronađene prilikom testiranja.
2. Kompletno testiranje je nemoguće. Testiranjem se ne mogu pokriti sve moguće varijacije svih testnih scenarija. Iz tog razloga, potrebno je kvalitetno isplanirati testiranje i postaviti jasne prioritete testnim scenarijima.
3. Testiranje u ranoj fazi razvoja štedi vrijeme i novac. Pogreške otkrivene u kasnijim fazama razvoja programskog proizvoda mogu uvelike povećati troškove i vrijeme isporuke programskog proizvoda.
4. Pogreške su povezane. Pareto princip, odnosno 80-20 pravilo opisuje kako 80% rezultata proizlazi iz 20% svih uzoraka. U prijevodu, testiranjem se mogu otkriti osjetljiva područja na pogreške te se potom treba testiranje usmjeriti u smjeru tog osjetljivog područja.
5. Testovi degradiraju sa vremenom. Kako se program razvija, tako se i testni scenariji trebaju razvijati s njim, odnosno potrebno je nadograđivati testne scenarije sa novim funkcionalnostima kako bi bili relevantni u daljnjem otkrivanju pogrešaka.
6. Kontekst je bitan u testiranju. Svaki program ima svoje zahtjeve i ciljeve stoga nijedan program ne smije biti testiran sa istim tehnikama i metodama.
7. Odsutnost greška je zabluda. Poput načela koje govori da kompletno testiranje nije moguće, tako nije moguće da su sve pogreške otklonjene. Gotovo uvijek postoji vjerojatnost da postoje neotkrivene pogreške.“ [10].

Provođenjem navedenih načela se uvelike povećava učinkovitost testiranja te osigurava kvaliteta programskog proizvoda. Sa kvalitetnim testiranjem, sprječavaju se daljnje komplikacije u razvoju, što rezultira u štednji vremena i novca [4].

3. Metodologija testiranja programskog proizvoda

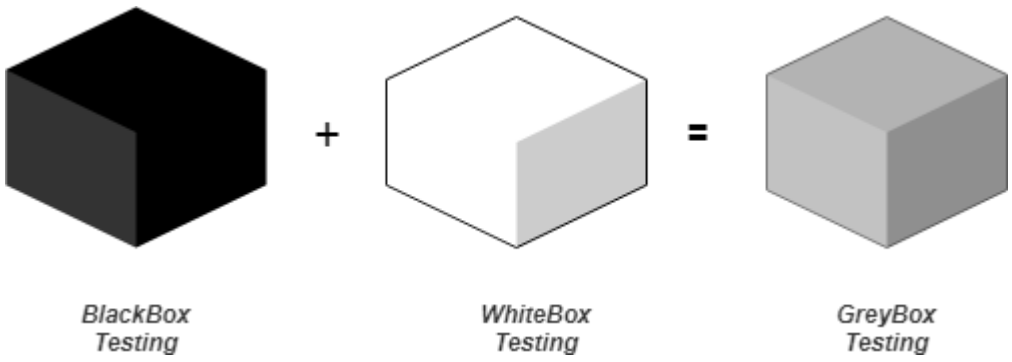
Programski proizvodi su neizbježni u današnjem svijetu. Pronalazimo ih svugdje oko nas, u svakodnevnim uređajima, pametnim kućama (eng. *Smart Home*), automobilima, itd. Kako se tehnologija konstantno ubrzano razvija, tako i potreba za održavanjem njene kvalitete.

Metodologija se definirana kao sustavna metoda za rješavanje istraživačkih problema prikupljanjem podataka pomoću raznih tehnika, interpretiranjem prikupljenih podataka i zatim dolaženja do zaključka. U području testiranja programskog proizvoda, metodologija je pristup i strategija koju tester koristi za vrijeme testiranje proizvoda.

Kako bi se osigurala kvaliteta programskog proizvoda, potrebno je procijeniti koja će vrsta, metoda ili razina testiranja biti najoptimalnija za određeni program, kako bi se postigli željeni ciljevi u što kraćem roku.

3.1. Metode testiranja

Kod testiranja programskog proizvoda, tester se može koristiti različitim metodama i pristupima. Postoje različiti pristupi i vrste testiranja. Testiranje prema pristupima i vrstama možemo podijeliti na statičko, dinamičko, testiranje crne kutije (eng. *Black-box*), testiranje bijele kutije (eng. *White-box*), testiranje sive kutije (eng. *Grey-box*), testiranje sučelja komponenti i vizualno testiranje [12]. Slika 3.1 (vlasništvo: *javatpoint.com*) prikazuje odnos između metoda testiranja crne, bijele i sive kutije.



Slika 3.1 Odnos metoda testiranja između crne, bijele i sive kutije

3.1.1. Black-box testiranje

Metoda Black-box testiranja fokusira se na očekivan ishod i/ili ponašanje programskog proizvoda bez informacija o internim procesima i radu softvera. Iz razloga što testeri ne znaju koji se procesi događaju unutar programskog proizvoda tokom testiranja ova metoda je dobila ime Black-box. Ova metoda se najčešće koristi kod provjere definiranih specifikacija programskih proizvoda.

Tokom korištenja ove metode testiranja krucijalno je definirati skup testnih slučajeva koji sadrži gotovo sve moguće inpute (hrv. *Unos*). Sve inpute nije moguće testirati zbog vremena kojeg iziskuje, stoga se definiraju skupovi potencijalnih inputa. Iz definiranih skupova, određeni podaci se odabiru i postave kao input u testu [13].

3.1.2. White-box testiranje

Za razliku od Black-box metode testiranja koja uopće nema znanja o internim procesima proizvoda, White-box metoda se usredotočuje upravo na rad programa iznutra. Kako bi mogli koristiti ovu metodu testiranja potrebno je biti upoznat sa unutrašnjom strukturom, dizajnom i implementacijom programskog proizvoda koji se testira.

Specifikacije proizvoda nisu toliko bitne kod ovog tipa testiranja pošto se tokom provođenja testa provjerava njegov kod. Poznavanje strukture i svih procesa unutar proizvoda je ovoj metodi nadjenulo ime staklena kutija (eng. *Glass-box*). Iako ova metoda ima dva naziva, češće se koristi izraz White-box. Ova metoda testiranja se može provesti na svim razinama testiranja, no najčešće se koristi nakon provođenja testova metodom crne kutije [10].

3.1.3. Grey-box testiranje

Metoda koja u svom pristupu spaja karakteristike White-box i Black-box testiranja zove se Grey-box metoda. Kao što je rečeno, Grey-box metoda koristi aspekte iz White-box i Black-box testiranja, pa se tako kod ove metode tokom testiranja djelomično zna kako programski proizvod funkcionira iznutra.

Ova metoda omogućava provođenje testiranja na kodu i na strani web aplikacije, odnosno onako kako ju korisnik vidi. Grey-box metoda se koristi kako bi se pronašle greške koje su uzrokovane nepravilnom strukturom koda ili nepravilnim rukovanjem [10].

Black-box	White-box	Grey-box
Nije potrebno znanje o programu niti o njegovom radu	Potrebno je relevantno znanje o programu i njegovom radu	Potrebno je detaljno razumijevanje koda i baze podataka
Korisno za testiranje funkcionalnosti i dizajna korisničkog sučelja	Korisno za detaljno testiranje funkcionalnosti	Korisno za jedinično testiranje
Jednostavno za izvedbu, izvode ga korisnici i testeri	Malo teže za izvedbu, izvode ga testeri i programer	Najteže za izvedbu, izvode ih najčešće programer

Tablica 3.1 Usporedba metoda testiranja crne, bijele i sive kutije

3.1.4. Regresijsko testiranje

Regresijsko testiranje je vrlo bitno u procesu razvijanja programskog proizvoda. Ova metoda se koristi nakon provedenog testiranja funkcionalnosti pojedinih komponenti i sa njom se testira kompletan programski proizvod. Dodavanje novih funkcionalnosti, promjene u kodu sukladno sa zahtjevima, saniranje greški i problema performansi su sve slučajevi u kojima se koristi regresijsko testiranje. Sukladno s programskim proizvodom i njegovim specifikacijama definiraju se testovi koje se treba provesti.

Broj testova može dostizati ogromne brojke, stoga se pokušava uštediti na vremenu kako je god moguće. Vrlo je bitno definirati koje je testove najvažnije provesti kako bi se što brže i efikasnije obavilo testiranje. Testovima se pridodaju prioriteti te se tim redoslijedom izvršavaju. U nekim slučajevima se testovi nižih prioriteta mogu u potpunosti izostaviti kako bi se skratilo vrijeme testiranja. Prioritetni testovi se mogu definirati po raznim kriterijima. Testeri mogu odlučiti da su najvažniji testovi: oni koji se odnose na područje sa najviše grešaka, oni koji obuhvaćaju što veći dio programskog proizvoda, oni koji su vremenski najoptimalniji, i slično. Bez obzira na kriterij definiranja prioriteta, broj testova i dalje može biti prevelik i zato se u tim slučajevima dio regresijskog testiranja može automatizirati [14].

3.1.5. Ad Hoc testiranje

Ad Hoc testiranje je karakteristično po tome što se provodi bez prethodno definiranog plana i dokumentacije. Ova vrsta testiranja se izvodi nasumično i bez očekivanih rezultata. Pošto ne postoji razrađeni plan testiranja, u ovoj metodi testeri samostalno istražuju programski proizvod i njegovo ponašanje u raznim situacijama.

Zbog svojih obilježja, ova metoda testiranja se smatra naj ne formalnijom metodom testiranja. Iako je tako ova metoda ima više prednosti u odnosu na druge metode. Ad Hoc testiranjem se može naići na greške koje prethodno nisu bile navedene u dokumentaciji i do kojih se ne bi moglo doći formalnijim metodama testiranja [15], [16].

3.1.6. Statičko testiranje

Metoda statičkog testiranja izvodi se bez pokretanja programskog proizvoda i izvršavanja njegovih funkcija. Ovakva vrsta testova se može provesti vrlo rano u razvojnom procesu programskog proizvoda. Cilj ove metode je pravovremeno nalaženje greška u proizvodu. Statičko testiranje se može vršiti na različite dijelove programskog proizvoda kao što su: kod, dizajn, funkcionalnost, struktura, modeli, specifikacija zahtjeva i sl. [17].

Pristup testera kod statičkog testiranja može varirati ovisno o tome što i kako se treba testirati. Tester može izabrati neformalni pregled, formalni pregled, kolegijalni pregled, demonstracija prolaska (eng. *Walkthrough*) ili inspekciju za provođenje statičkog testa.

Neformalni pregled se koristi u fazi razvoja projektne dokumentacije. Službeni dokumenti se prezentiraju timu i voditeljima projekta koji potom mogu sugerirati nove načine za poboljšanje proizvoda [10], [17].

Formalni pregled zahtjeva dokumentaciju kako bi se mogao provesti. Ova vrsta pregleda je sačinjena od šest glavnih faza: planiranje, započinjanje (eng. *Kick-off*), formalni sastanak, korekcija grešaka i kontrola (eng. *Follow-up*) [10].

Kolegijalni pregled je karakterističan po tome što evaluaciju vrše autor i jedan ili više njegovih kolega. Ova vrsta pregleda isključuje menadžere i klijente iz procesa evaluacije.

Walkthrough je sastanak kojeg vodi autor proizvoda i uglavnom je neformalne prirode. Na ovom sastanku autor objašnjava sve o svom proizvodu i učesnicima nastoji približiti temu kako bi se mogle prikupiti povratne informacije od učesnika. Ovim procesom se nastoji prenijeti znanje, evaluirati sadržaj proizvoda i pokrenuti diskusiju o predloženim rješenjima.

Inspekcija zahtjeva definiranu dokumentaciju i postupak kako bi se mogla provesti, zbog toga se inspekcija smatra najformalnija vrstom pregleda. Ovu vrstu pregleda provode stručni moderatori kako bi se osigurala ispravnost pregleda. Ciljevi inspekcije su nalaženje i detaljno dokumentiranje grešaka, utvrđivanje mogućih unaprjeđenja, evaluacija kvalitete i evaluacija performansi.

3.1.7. Dinamičko testiranje

U dinamičkom testiranju testira se ponašanje izvedenog programa sa dinamičkim varijablama. Dinamičko testiranje se može provesti prije finalne verzije proizvoda. Funkcija ovih testova je pronalaženje slabih točki i nedostataka proizvoda u stvarnim okruženjima.

Rezultati dinamičkog testiranja se uspoređuju sa očekivanim rezultatima i na temelju toga se definiraju mogući nedostatci proizvoda. Funkcionalno i nefunkcionalno testiranje su kategorije tehnika dinamičkog testiranja. Fokus nefunkcionalnog testiranja je sigurnost, performanse, dostupnost i slično. S druge strane, funkcionalno testiranje se bavi funkcionalnim zahtjevima [10], [18].

3.1.8. Agilno testiranje

Proces agilnog testiranja prati i poštuje principe agilnog razvoja programskog proizvoda. Ti principi definiraju karakteristike koje sustav mora imati. Prilagodljivost, fleksibilnost i kooperativnost su određene karakteristike koje sustav mora posjedovati.

Agilna okolina u kojoj se provodi testiranje zahtjeva blisku suradnju između testera i programera. Kako bi se uspješno provelo testiranje u takvim okolinama potrebno je odrediti prioritetna područja za testiranje, automatizirati što više testova, više koristiti istraživačke testove i prilagoditi se izazovima.

3.1.9. Smoke test

Smoke test je vrsta testiranja koja nije detaljna no obuhvaća većinu glavnih funkcija (eng. *Build*) pred-izdane verzije programa. Ime ove vrste testiranja potiče od tehnike testiranja hardverskih komponenti. U ovoj tehnici hardver se uključio i promatrao hoće li se krenuti dimiti. Ukoliko dima ne bi bilo, komponenta bi prošla test.

Ovom vrstom testiranja se provodi osnovna provjera build-a u kojoj se donosi odluka je li build dovoljno stabilan za nastavak testiranja. Smoke testom se također provjerava ispravnost osnovnih funkcionalnosti programskog proizvoda. Iako je smoke test poprilično površan, ušteda vremena, lociranje pogrešaka u ranoj fazi i mali broj potrebnih testova tokom njegovog provođenja ga čini konkurentnim. Smoke testovi se također mogu automatizirati što dodatno ubrzava njihovo provođenje [14], [17].

3.1.10. Testiranje ispravnosti

Testiranje ispravnosti (eng. *Sanity Test*) se koristi kako bi se provjerila ispravnost promjena unutar koda. Ova vrsta testiranja se najčešće provodi nakon smoke testa. Kao i smoke test, testiranje ispravnosti obuhvaća većinu ključnih funkcionalnosti i pritom ne ulazi u detalje. Testiranje ispravnosti ne koristi definiran plan i dokumentaciju. Manjak dokumentacije drastično skraćuje vrijeme potrebno za provođenje testiranja ispravnosti. Još jedna prednost ove vrste testiranja je njena mogućnost brzog identificiranja grešaka unutar ključnih funkcionalnosti programskog proizvoda Ciljevi testiranja ispravnosti su provjera implementirane funkcionalnosti i definiranje ugroženosti osnovnih funkcija programskog proizvoda. Ukoliko proizvod ne prođe testiranje ispravnosti na njemu se neće vršiti detaljnije metode testiranja. Ovaj tip testiranja može se provesti ručno ili ga se može automatizirati pomoću specijaliziranih alata [14], [17].

3.1.11. Ručno testiranje

Najstarija vrsta testiranja je ručno (eng. *Manual*) testiranje. Istovremeno, zahtjeva da se tester razumije u program i njegove specifikacije, zato je i najzahtjevnija metoda testiranja. Za provođenje ručnih testova potrebna je strpljivost i oko za detalje. Ručno testiranje je kompleksno i monotono, tako da tester treba imati upornost i intuiciju za rješavanje problema (eng. *Problem solving*).

S obzirom da ručno testiranje dolazi prije svih drugih tehnika testiranja, ono ostaje imperativ u testiranju programa. Kako bi se testovi mogli automatizirati, prvo je potrebno izvršiti ručne testove, od njih sve počinje. Slika 3.2 prikazuje faze ručnog testiranja.

Slika 3.2 Faze ručnog testiranja

Kod faze analize zahtjeva, testeri detaljno proučavaju zahtjeve, specifikacije i projektnu dokumentaciju. Na temelju tih podataka, tester kreće sa fazom izrade testnog plana. Testni plan sadrži informacije o uvjetima i ciljevima testiranja [15]. Također, sadrži upute o načinu testiranja određenih funkcionalnosti, raspored izvođenja testiranja te prioritete testova.

Nakon izrade testnog plana, izrađuju se testni slučajevi. Testni slučaj sastoji se od preduvjeta, testnih koraka i očekivanih rezultata. Kada se izvršava testiranje, potrebno je detaljno slijediti testne slučajeve te uspoređivati dobivene i očekivane rezultate kako bi mogli uočiti pogreške.

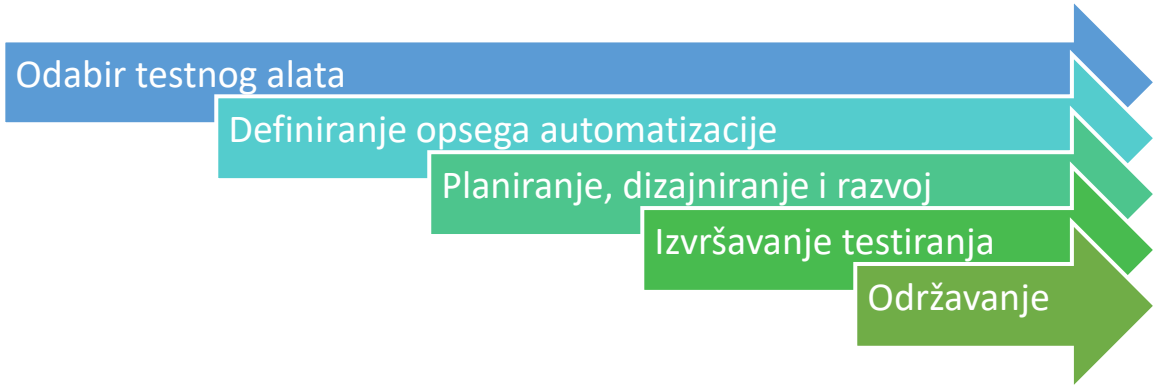
Nakon što je pogreška otkrivena, ona se prijavljuje programeru tako da se detaljno opišu koraci i testno okruženje kako bi programer mogao reproducirati pogrešku. Poslije toga, pogreška je ispravljena i potrebno je ponovno izvršavanje testiranja sa svim testnim slučajevima kako bi provjerili da li je ispravak jedne pogreške možda uzrokovao drugu. Kada je pogreška ispravljena, zatvara se testiranje i testirana funkcionalnost prelazi u okruženje produkcije [19].

3.1.12. Automatsko testiranje

Automatsko testiranje kao što i sam naziv opisuje, izvodi se uz pomoć programerskih alata za automatizaciju testova. Za razliku od ručnog, automatsko testiranje podrazumijeva postojanje koda kako bi se testovi izvršavali automatski. Napisani kod se naziva testna skripta. Automatsko testiranje se najčešće koristi kod testova kojih je potrebno konstantno ponavljati, točnije osigurati se da manja ali vitalna komponenta programa funkcionira.

Postoje dva pojma koja se koriste prilikom automatskog testiranja:
„Automatizirano testiranje je čin provođenja određenog niza testova i provjere njihovih rezultata umjesto da se to radi ručno. S druge strane automatizacija testiranja predstavlja širi koncept. Odnosi se na automatizaciju cjelokupnog procesa upravljanja različitim testiranjima unutar neke organizacije.“ [10] .

Automatski testovi se izvršavaju na kompleksnim i dugoročnim projektima, gdje je potrebno održavanje i ubrzavanje procesa razvoja. S automatskim testovima umanjuju se troškovi i vrijeme testiranja osnovnijih funkcionalnosti što uvelike olakšava posao testeru. Nemoguće je automatizirati svaki testni scenarij, zato je bitno kvalitetno planirati koje će se testovi izvršavati na koji način. No, postoje i slučajevi kada je nemoguće ručno testirati određeni testni scenarij, a tada se svakako isplati uložiti u automatsko testiranje [18]. Slika 3.3 prikazuje faze automatskog testiranja.



Slika 3.3 Faze automatskog testiranja

Faza odabira testnog alata je temelj prije samog početka razvoja automatskih testova. Potrebno je odabrati testni alat koji je odgovara potrebama programskog proizvoda. Danas je moguće koristit testne alate u gotovo svakom programskom jeziku.

Zatim, potrebno je definirati sam opseg automatizacije. Kao što je već spomenuto, nije moguće automatizirati sve testove, stoga je potrebno pomno planirati koji testovi su pogodni za automatizaciju. Treba računati da se automatski testovi trebaju konstantno održavati, zato se izrađuju najčešće za ključne funkcionalnosti.

Faza planiranja, dizajniranja i razvoja je daleko najkompleksnija faza jer uključuje kodiranje. Općenito, testeri ne moraju znati programirati, ali kada su u pitanju testeri koji izrađuju automatske testove, potrebno je znati barem osnove kodiranja.

Tokom faze izvršavanja testiranja, testni alat najčešće u pregledniku (eng. *Browser*) izvršava testove. Nakon završetka testiranja, testni alat isporučuje analizu prolaznosti testova. Ako su testovi međusobno povezani, pad jednog vitalnog testa može uzrokovati pad svih ostalih. Stoga je bitno konstanto pratiti i održavati automatske testove [12].

Postoje razni alati za automatizaciju testova, a neki od njih su:

- Selenium
- Eggplant
- Katalon Studio
- Cucumber
- IntelliJ
- JMeter

3.2. Razine testiranja

Jedinično testiranje, integracijsko testiranje, testiranje sustava i provjera prihvatljivosti su četiri glavne razine na kojima se testiranje provodi tokom razvojnog procesa programskog proizvoda.

Slika 3.4 Razine testiranja

3.2.1. Jedinično testiranje

Jedinično testiranje (eng. *Unit testing*) testira najmanje izvodive dijelove programa. Izvodivi djelić aplikacije može sadržavati od nekoliko linija koda do tisuća linija koda. Odabrani dio testira se odvojeno do ostatka proizvoda. Programski proizvod testira se na ovoj razini prije testiranja kompletnog proizvoda. Prilikom izvođenja ovakvih testova, pogreške se saniraju neposredno nakon njihovog pronalaska.

3.2.2. Integracijsko testiranje

Integracijsko testiranje (eng. *Integration testing*) testira interakciju modula s ostalim dijelovima programa i povezanost među različitim modulima. Ova vrsta testiranja se može izvršiti na više načina. Big Bang (hrv. *Veliki Prasak*) je naziv jednog od tih načina. U njemu se sve komponente simultano integriraju u programski proizvod te se potom testiraju. Drugi način integracijskog testiranja zove se inkrementalno testiranje. Kod inkrementalnog testiranja komponente se jedna po jedna integriraju u programski proizvod. Ova vrsta testiranja se može provesti na dva načina: od dolje prema gore i od gore prema dolje. U pristupu od dolje prema gore se testiraju niže komponente u sustavu a potom više, dok se u pristupu od gore prema dolje testira od kompletnog programskog proizvoda prema nižim komponentama [8].

3.2.3. Sistemsko testiranje

Sistemsko testiranje (eng. *System testing*) je proces koji je sačinjen od: definiranja testnih planova i slučajeva, generiranja i odabira testnih podataka, izvršavanja testnih slučajeva, korekcije programskih grešaka, regresijskog testiranja i ponavljanja aktivnosti testnog ciklusa. Ova vrsta testiranja koristi u svrhu evaluacije usklađenosti programskog proizvoda sa korisničkim, funkcionalnim i poslovnim zahtjevima. Kako bi se uspjele detektirati karakteristične pogreške u produkcijskom okruženju, potrebno je prilagoditi okruženje u kojem se odvija testiranje da bude što sličnije produkcijskom okruženju [19].

3.2.4. Testiranje prihvatljivosti

Testiranje prihvatljivosti (eng. *Acceptance testing*) se pokreće na zahtjev naručitelja proizvoda. Ovi testovi se najčešće provode s timom naručitelja, a cilj im je demonstrirati funkcionalnost i spremnost proizvoda za plasiranje na tržište. Proces provjere prihvatljivosti sastoji se od dvije faze. Alfa faza je naziv prve faze u kojoj odabrani korisnici testiraju proizvod u kontroliranim uvjetima. Druga faza se zove beta faza. U ovoj fazi proizvod se učini dostupnim određenom broju pravih korisnika koji ga potom testiraju. Probleme na koje nailaze i svoja mišljenja prijavljuju programerima koji zatim ispravljaju prijavljene probleme [11].

4. Primjeri testova

U ovom poglavlju, izraditi će se tri ručna i jedan automatski test. Testiranje će se provoditi na unin.hr početnoj stranici, koja je zapravo WordPress web stranica. Ručni testovi će biti izrađeni u tablicama sa tipično raspisanim testnim koracima i očekivanim rezultatima. Automatski test će biti izrađen sa testnim alatom Selenium, uz Chrome Webdriver.

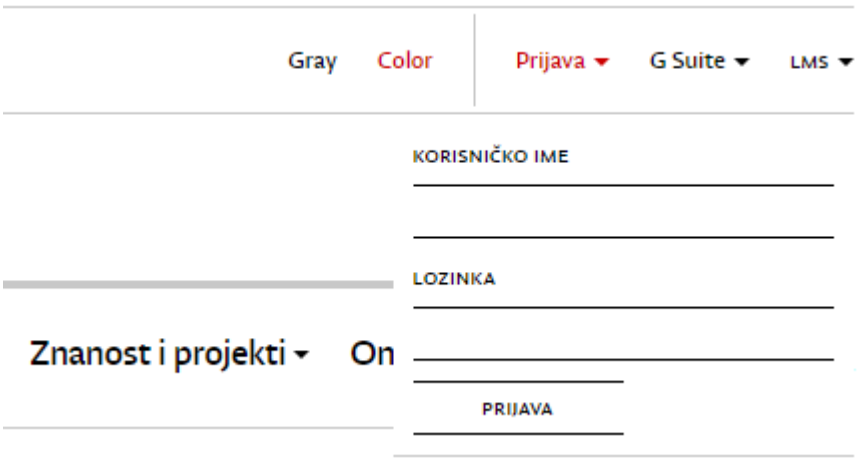
Happy path (hrv. *Sretan put*) predstavlja testni scenarij u kojemu se testiraju funkcionalnosti na način da se očekuje pozitivan rezultat na kraju testa. Unhappy path (hrv. *Nesretan put*) predstavlja testni scenarij u kojemu se testiraju funkcionalnosti na način da se očekuje negativan rezultat na kraju testa. Prvi test je Happy path, dok druga dva testa su kombinacije Happy i Unhappy path-a.

Slika 4.1 Početna stranica unin.hr

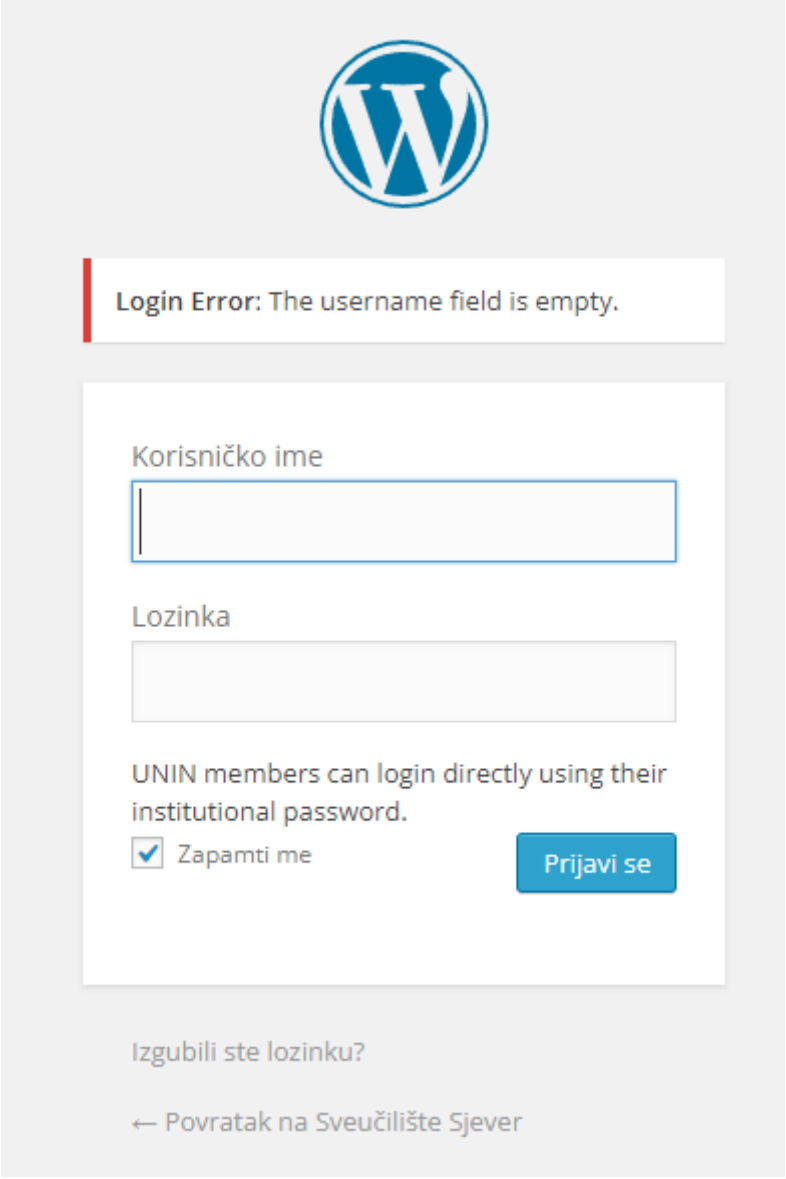
4.1. Primjer ručnog testa

Prvi primjer ručnog testa sadrži testni scenarij koji opisuje korak po korak, uspješnu prijavu korisnika na unin.hr stranicu.

Testni ID	1.		
Testni slučaj	Prijava korisnika na unin.hr		
Testni scenarij	Uspješna prijava registriranog korisnika (*Happy path*)		
Preduvjeti	• Registriran profil na unin.hr • Otvorena početna stranica unin.hr		
Testni koraci	**Očekivani rezultati**	**Status**	**Komentari**
1. Korisnik klikne na ***Prijava*** padajući izbornik u navigacijskoj traci	1. ***Prijava*** padajući izbornik je prikazan, vidljiva su polja za unos ***Korisničko ime*** i ***Lozinka*** te gumb ***Prijava***	Prolaz/ Pad/ Blokirano	
2. Korisnik klikne na polje za unos ***Korisničko ime*** unutar padajućeg izbornika za prijavu	2. Polje za unos ***Korisničko ime*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
3. Korisnik upisuje **„ankalaica“** u polje za unos ***Korisničko ime*** unutar padajućeg izbornika za prijavu	3. Polje za unos ***Korisničko ime*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
4. Korisnik klikne na polje za unos ***Lozinka*** unutar padajućeg izbornika za prijavu	4. Polje za unos ***Lozinka*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
5. Korisnik upisuje **„lozinka“** u polje za unos ***Lozinka***	5. Polje za unos ***Lozinka*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
6. Korisnik prelazi sa mišem (eng. *Hover*) iznad gumba ***Prijava***	6. Gumb ***Prijava*** sadrži crveni tekst i obrub kada se prelazi sa mišem i kada je aktivno	Prolaz/ Pad/ Blokirano	
7. Korisnik klikne na gumb ***Prijava*** unutar padajućeg izbornika za prijavu	7. Učitana je početna stranica unin.hr, ***Ime*** i ***Prezime*** prijavljenog korisnika je prikazano u navigacijskoj traci	Prolaz/ Pad/ Blokirano	
8. Korisnik klikne na ***Ime*** i ***Prezime*** padajući izbornik u navigacijskoj traci	8. ***Ime*** i ***Prezime*** prijavljenog korisnika postaje crvene boje. ***Odjava*** padajući izbornik je prikazan koje sadrži samo gumb ***Odjava*** sa sivom pozadinom i crnim tekstom	Prolaz/ Pad/ Blokirano	
9. Korisnik prelazi sa mišem (eng. *Hover*) iznad gumba ***Odjava***	9. Gumb ***Odjava*** sadrži crvenu pozadinu i bijeli tekst kada se prelazi sa mišem i kada je aktivno	Prolaz/ Pad/ Blokirano	
10. Korisnik klikne na gumb ***Odjava*** unutar padajućeg izbornika za odjavu	10. Učitana je početna stranica unin.hr, ***Prijava*** padajući izbornik je prikazan u navigacijskoj traci	Prolaz/ Pad/ Blokirano	



Slika 4.2 Padajući izbornik Prijava



Slika 4.3 Dijalog za prijavu na WordPress stranici

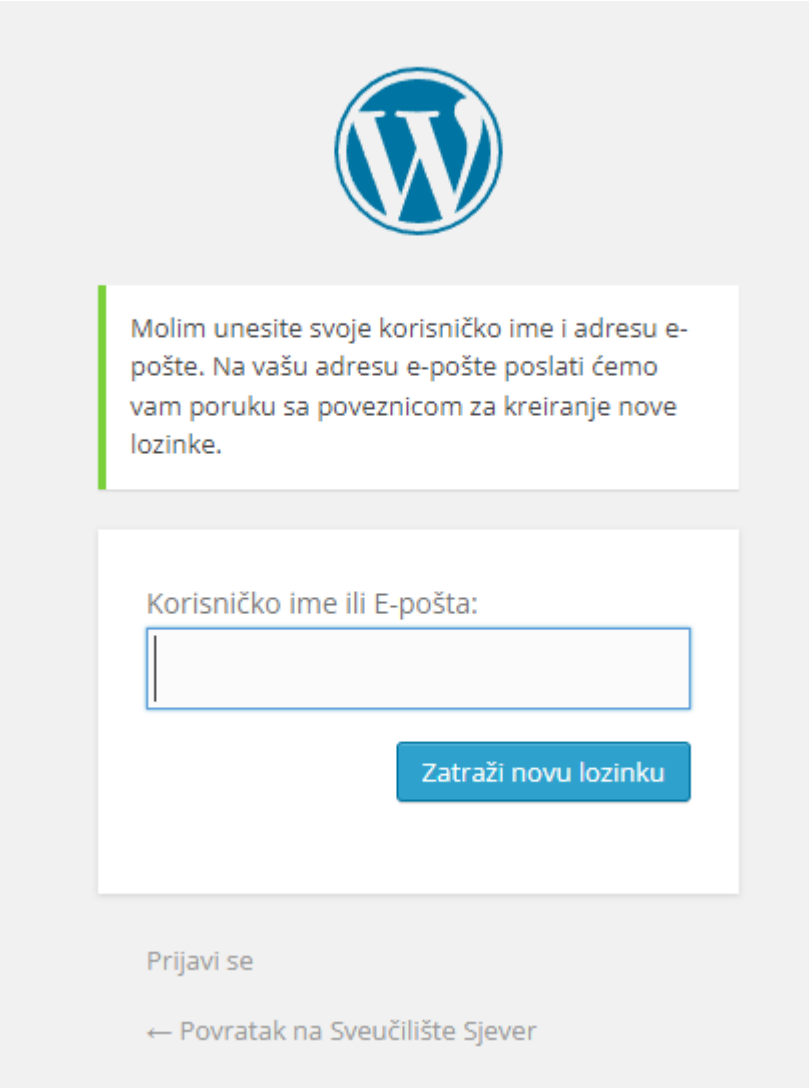
Drugi primjer ručnog testa sadrži testni scenarij koji opisuje korak po korak, neuspješnu prijavu korisnika na unin.hr stranicu. Nakon 8. koraka, testni scenarij opet ima Happy path jer završava test sa pozitivnim očekivanim rezultatom.

Testni ID	2.		
Testni slučaj	Neuspješna prijava korisnika na unin.hr		
Testni scenarij	Neuspješna prijava registriranog korisnika, prazna polja za unos (*Unhappy path*)		
Preduvjeti	• Registriran profil na unin.hr • Otvorena početna stranica unin.hr		
Testni koraci	**Očekivani rezultati**	**Status**	**Komentari**
1. Korisnik klikne na ***Prijava*** padajući izbornik u navigacijskoj traci	1. ***Prijava*** padajući izbornik je prikazan, vidljiva su polja za unos ***Korisničko ime*** i ***Lozinka*** te gumb ***Prijava***	Prolaz/ Pad/ Blokirano	
2. Korisnik klikne na polje za unos ***Korisničko ime*** unutar padajućeg izbornika za prijavu	2. Polje za unos ***Korisničko ime*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
3. Korisnik ostavlja polje za unos ***Korisničko ime*** prazno	3. Polje za unos ***Korisničko ime*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
4. Korisnik klikne na polje za unos ***Lozinka*** unutar padajućeg izbornika za prijavu	4. Polje za unos ***Lozinka*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
5. Korisnik ostavlja polje za unos ***Lozinka*** prazno	5. Polje za unos ***Lozinka*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
6. Korisnik prelazi sa mišem (eng. *Hover*) iznad gumba ***Prijava***	6. Gumb ***Prijava*** sadrži crveni tekst i obrub kada se prelazi sa mišem i kada je aktivno	Prolaz/ Pad/ Blokirano	
7. Korisnik klikne na gumb ***Prijava*** unutar padajućeg izbornika za prijavu	7. Učitana je WordPress login stranica, greška je prikazana „***Login Error: The username field is empty.***“, polja za unos ***Korisničko ime*** i ***Lozinka*** su vidljiva, zajedno sa kvačicom u potvrdnom okviru ***Zapamti me***	Prolaz/ Pad/ Blokirano	
8. Korisnik upisuje **„ankalaica“** u polje za unos ***Korisničko ime*** unutar padajućeg izbornika za prijavu	8. Polje za unos ***Korisničko ime*** sadrži plavi obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
9. Korisnik upisuje **„lozinka“** u polje za unos ***Lozinka***	9. Polje za unos ***Lozinka*** sadrži plavi obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
10. Korisnik klikne na potvrdni okvir ***Zapamti me***	10. Potvrdni okvir ***Zapamti me*** je sada prazan	Prolaz/ Pad/ Blokirano	
11. Korisnik klikne na gumb ***Prijavi se***	11. Učitana je početna stranica unin.hr, ***Ime*** i ***Prezime*** prijavljenog korisnika je prikazano u navigacijskoj traci	Prolaz/ Pad/ Blokirano	

Treći primjer ručnog testa sadrži testni scenarij koji opisuje korak po korak, neuspješnu prijavu korisnika na unin.hr stranicu. Nakon 8. koraka, testni scenarij opet ima Happy path jer korisnik postavlja novu lozinku i tako završava test sa pozitivnim očekivanim rezultatom.

Testni ID	3.		
Testni slučaj	Zaboravljena lozinka korisnika na unin.hr		
Testni scenarij	Neuspješna prijava registriranog korisnika, resetiranje lozinke (*Happy + Unhappy path*)		
Preduvjeti	• Registriran profil na unin.hr • Otvorena početna stranica unin.hr		
Testni koraci	**Očekivani rezultati**	**Status**	**Komentari**
1. Korisnik klikne na ***Prijava*** padajući izbornik u navigacijskoj traci	1. ***Prijava*** padajući izbornik je prikazan, vidljiva su polja za unos ***Korisničko ime*** i ***Lozinka*** te gumb ***Prijava***	Prolaz/ Pad/ Blokirano	
2. Korisnik klikne na polje za unos ***Korisničko ime*** unutar padajućeg izbornika za prijavu	2. Polje za unos ***Korisničko ime*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
3. Korisnik upisuje **„ankalaica“** u polje za unos ***Korisničko ime*** unutar padajućeg izbornika za prijavu	3. Polje za unos ***Korisničko ime*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
4. Korisnik klikne na polje za unos ***Lozinka*** unutar padajućeg izbornika za prijavu	4. Polje za unos ***Lozinka*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
5. Korisnik upisuje **„kriva-lozinka“** u polje za unos ***Lozinka***	5. Polje za unos ***Lozinka*** sadrži crveni obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
6. Korisnik prelazi sa mišem (eng. *Hover*) iznad gumba ***Prijava***	6. Gumb ***Prijava*** sadrži crveni tekst i obrub kada se prelazi sa mišem i kada je aktivno	Prolaz/ Pad/ Blokirano	
7. Korisnik klikne na gumb ***Prijava*** unutar padajućeg izbornika za prijavu	7. Učitana je WordPress login stranica, greška je prikazana „***Directory Authentication Error: Incorrect password.***“, polja za unos ***Korisničko ime*** i ***Lozinka*** su vidljiva, zajedno sa kvačicom u potvrdnom okviru ***Zapamti me***	Prolaz/ Pad/ Blokirano	
8. Korisnik klikne na poveznicu ***Izgubili ste lozinku?*** ispod dijaloga za prijavu	8. Prikazana je poruka „***Molim unesite svoje korisničko ime i adresu e-pošte. Na vašu adresu e-pošte poslati ćemo vam poruku sa poveznicom za kreiranje nove lozinke.“,*** polje za unos ***Korisničko ime ili E-pošta*** je vidljivo, zajedno sa gumbom ***Zatraži novu lozinku***	Prolaz/ Pad/ Blokirano	
9. Korisnik upisuje **„ankalaica“** u polje za unos ***Korisničko ime ili E-pošta***	9. Polje za unos ***Korisničko ime ili E-pošta*** sadrži plavi obrub kada je aktivno	Prolaz/ Pad/ Blokirano	

10. Korisnik klikne na gumb ***Zatraži novu lozinku***	10. Učitana je WordPress login stranica, prikazana je poruka „***Please Login Provjerite svoju e-poštu, za poveznicu za potvrdu registracije***.“, polja za unos ***Korisničko ime*** i ***Lozinka*** su vidljiva, zajedno sa kvačicom u potvrdnom okviru ***Zapamti me***	Prolaz/ Pad/ Blokirano	
11. Korisnik otvara e-poštu i klikne na link za ponovo postavljanje lozinke koji se nalazi ispod teksta „***Za novu lozinku, posjetite ovu adresu:***“	11. Učitana je WordPress login stranica, prikazana je poruka „***Ispod upišite vašu novu lozinku.***“, polja za unos ***Nova lozinka*** i ***Potvrdi novu lozinku*** su vidljiva, zajedno sa ***Pokazateljem jačine***	Prolaz/ Pad/ Blokirano	
12. Korisnik upisuje **„Nova-lozinka123“** u polje za unos ***Nova lozinka***	12. Polje za unos ***Nova lozinka*** sadrži plavi obrub kada je aktivno, prilikom upisivanja lozinke ***Pokazatelj jačine*** mijenja statuse iz ***Jako slabo***, ***Slabo***, ***Srednja*** i ***Jako***	Prolaz/ Pad/ Blokirano	
13. Korisnik upisuje **„Nova“** u polje za unos ***Potvrdi novu lozinku***	13. Polje za unos ***Potvrdi novu lozinku*** sadrži plavi obrub kada je aktivno, prilikom upisivanja lozinke ***Pokazatelj jačine*** ima status ***Neslaganje***	Prolaz/ Pad/ Blokirano	
14. Korisnik upisuje **„Nova-lozinka123“** u polje za unos ***Potvrdi novu lozinku***	14. Polje za unos ***Potvrdi novu lozinku*** sadrži plavi obrub kada je aktivno, prilikom upisivanja lozinke ***Pokazatelj jačine*** mijenja statuse iz ***Jako slabo***, ***Slabo***, ***Srednja*** i ***Jako***	Prolaz/ Pad/ Blokirano	
15. Korisnik klikne na gumb ***Reset lozinke***	15. Poruka je prikazana „***Vaš lozinka je resetirana. Prijavi se***“	Prolaz/ Pad/ Blokirano	
16. Korisnik klikne na poveznicu ***Prijavi se***	16. Poruka je prikazana „***Please Login***“, polja za unos ***Korisničko ime*** i ***Lozinka*** su vidljiva, zajedno sa praznim poljem u potvrdnom okviru ***Zapamti me***	Prolaz/ Pad/ Blokirano	
17. Korisnik upisuje **„ankalaica“** u polje za unos ***Korisničko ime***	17. Polje za unos ***Korisničko ime*** sadrži plavi obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
18. Korisnik upisuje **„Nova-lozinka123“** u polje za unos ***Lozinka***	18. Polje za unos ***Lozinka*** sadrži plavi obrub kada je aktivno	Prolaz/ Pad/ Blokirano	
19. Korisnik klikne na potvrdni okvir ***Zapamti me***	19. Potvrdni okvir ***Zapamti me*** sada sadrži kvačicu	Prolaz/ Pad/ Blokirano	
20. Korisnik klikne na gumb ***Prijavi se***	20. Učitana je WordPress profil stranica, ***Ime*** i ***Prezime*** prijavljenog korisnika je prikazano u navigacijskoj traci	Prolaz/ Pad/ Blokirano	
21. Korisnik prelazi mišem preko poveznice ***Sveučilište Sjever*** u navigacijskoj traci	21. ***Sveučilište Sjever*** padajući izbornik je prikazan sa tekstom plave boje kada je aktivan, vidljiv je gumb ***Visit Site*** koji također ima plavi tekst kada je aktivan	Prolaz/ Pad/ Blokirano	
22. Korisnik klikne na ***Visit Site*** gumb unutar ***Sveučilište Sjever*** padajućeg izbornika	22. Učitana je početna stranica unin.hr, ***Ime*** i ***Prezime*** prijavljenog korisnika je prikazano u navigacijskoj traci	Prolaz/ Pad/ Blokirano	



Slika 4.4 Dijalog za zaboravljenu lozinku na WordPress stranici

4.2. Primjer automatskog testa

„Selenium je alat za testiranje softvera koji služi za automatsko testiranje web aplikacija. Podržan je u sljedećim programskim jezicima: Java, C#, Perl, PHP, Python i Ruby. Takvi testovi se zatim mogu pokretati na bilo kojem novijem web pregledniku. Dostupan je na Windows, Linux i Mac OS X operativnim sustavima, a predstavlja open-source proizvod, tj. u potpunosti je besplatan i može ga preuzeti bilo tko.“ [16]. Primjer automatskog testa napisan u Java programskom jeziku za Selenium i Chrome WebDriver. Iz razloga što se automatski testovi koriste za jednostavnije testne slučajeve, slijedeći primjer je također jednostavan.

Test scenarij:

Uspješna prijava korisnika na unin.hr stranicu.

Testni koraci:

1. Upali preglednik Chrome
2. Otvori https://www.unin.hr/
3. Pronađi element **Prijava** i klikni na njega
4. Pronađi element **Korisničko ime**
5. Pronađi element **Lozinka**
6. Unesi „**ankalaica**“ u polje **Korisničko ime**
7. Unesi „**lozinka**“ u polje **Lozinka**
8. Klikni na element **Prijavi se**

```
import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.Assert;

import org.testng.annotations.Test;

public class LoginAutomation {

@Test

public void login() {

System.setProperty("webdriver.chrome.driver", "path of driver");

WebDriver driver=new ChromeDriver();

driver.manage().window().maximize();

driver.get("https://www.unin.hr/");

driver.findElement(By.name("Prijava")).click();

WebElement username=driver.findElement(By.id("user_login"));

WebElement password=driver.findElement(By.id("user_pass"));

WebElement login=driver.findElement(By.id("wp-submit"));

username.sendKeys("ankalaica");

password.sendKeys("lozinka");

login.click();

String actualUrl=" https://www.unin.hr/";

String expectedUrl= driver.getCurrentUrl();

Assert.assertEquals(expectedUrl,actualUrl);

}

}                                                                    [20]
```

5. Istraživanje i usporedba ručnog i automatskog testiranja u BizDataX programu

U ovom poglavlju, obraditi će se osnovni pojmovi vezani uz BizDataX program te kako izgleda razvoj programskog proizvoda na realnom projektu. Također, navesti će se opis samog istraživanja te će se povesti istraživanje sa definiranim parametrima. U sklopu istraživanja, izvršiti će se tri ručna i tri automatska testa u BizDataX programu kako bi dokazali slijedeće tvrdnje:

- ručno testiranje je isplativije na manjim projektima, dok je automatsko testiranje uporabljivije na većim projektima.

- ručno testiranje je pouzdanije u otkrivanju pogrešaka naspram automatskog testiranja.

Cilj ovoga istraživanja je usporediti ručno i automatsko testiranje, te navesti koje su to karakteristike koje opisuju svaku od ovih metoda testiranja. Obje metode zaslužuju svoje mjesto u razvoju programskog proizvoda, no potrebno je utvrditi koja od ovih metoda odgovara kojim situacijama kako bi lakše donijeli zaključak ovoga rada.

5.1. Općenito o BDX-u

„Ekobit d.o.o. je privatna hrvatska konzultantska i razvojna IT tvrtka osnovana 1992. koja nudi poslovna rješenja i usluge iz područja upravljanja životnim tijekom aplikacija (ALM), upravljanja testnim podacima i anonimizacije podataka, integracije te razvoja poslovnih aplikacija za PC i mobilne platforme. BizDataX je Ekobitovo rješenje za upravljanje testnim podacima i anonimizaciju podataka." [21].

Količine podataka kojima organizacije upravljaju rastu ubrzanim tempom. Podaci koji se koriste za razvoj programa, testiranje, treniranje i druge sekundarne svrhe rastu još i brže. Tvrtke imaju poteškoća u kontroli i zaštiti podataka, osobito u sekundarnom kontekstu gdje je teško postići implementaciju učinkovitih sigurnosnih mjera. Maskiranje podataka je najbolja praksa kada se pridržavate politike zaštite podataka. Također, služi kao dodatan sloj obrane u slučaju proboja sigurnosti. Slika 5.1 (vlasništvo: *BizDataX*) prikazuje BizDataX logo.

Slika 5.1 BizDataX logo

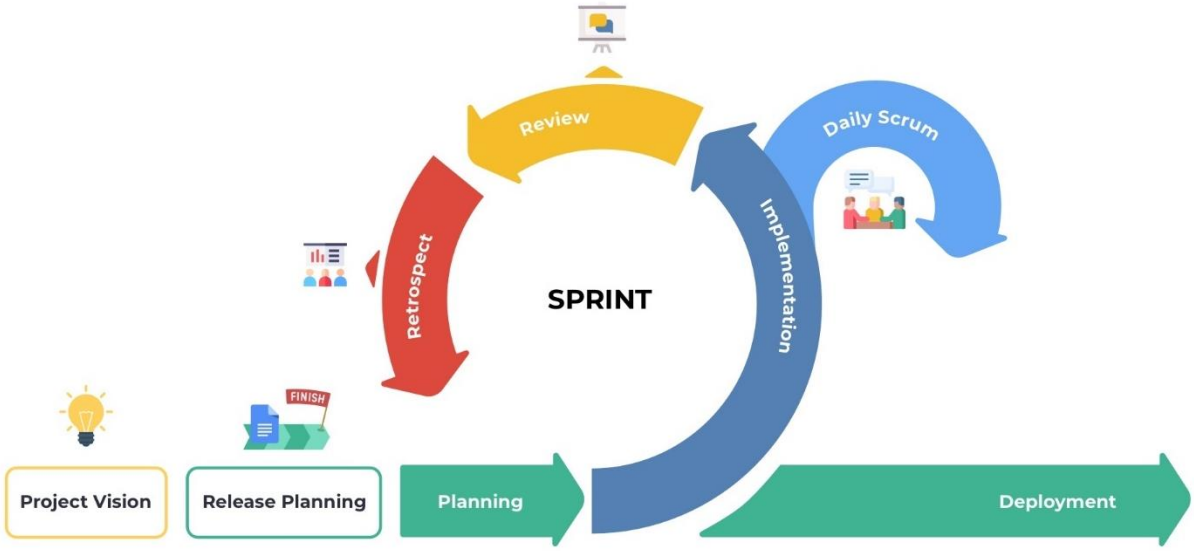
Podaci koji mogu identificirati osobu (eng. *Personally Identifiable Information*; *PII*) su svi podaci o korisniku koji bi mogli biti korišteni za identifikaciju. Sadrži imena, prezimena, adrese, matične brojeve, brojeve računa, medicinsku dokumentaciju, itd.

GDPR (eng. *General Data Protection Regulation*; *GDPR*) ili Opća uredba o zaštiti podataka, privlači veliku pozornost pri rukovanju osjetljivim podacima. Ljudi preuzimaju kontrolu nad svojim osobnim podacima, što znači da je potrebna privola svaki puta kada želite obraditi osobne podatke. Većina poslovnih klijenata ne bi odobrila korištenje njihovih osobnih podataka za razvoj i testiranje. Korištenjem alata za maskiranje, to jest anonimizaciju podataka, podaci su izvan dosega GDPR-a.

BizDataX program pruža prikladan alat za maskiranje podataka za rukovanje sa osjetljivim podacima na način da se prikrije identitet, a u isto vrijeme daje testerima i programerima podatke koji „izgledaju“ poput stvarnih [22].

5.2. Razvojni proces BDX-a

BDX projekt radi po Scrum principu razvoja programskog proizvoda. Scrum je radni okvir za upravljanje projektom koji koristi agilni način razmišljanja za razvoj, isporuku i održavanje složenih proizvoda, sa početnim naglaskom na razvoj programa. Scrum se pojavljuje i u drugim područjima, kao što su istraživanja, prodaja, marketing i ostale napredne tehnologije. Slika 5.2 (vlasništvo*: HYGGER.io*) prikazuje faze razvoja proizvoda po Scrum principu razvoja.



Slika 5.2 Faze razvoja proizvoda po Scrum principu razvoja

Sastanak dnevnog pregleda (eng. Daily) održava se svakodnevno. Svaka osoba u timu daje kratki osvrt na jučerašnje zadatke, s kojim problemima su se susretali te koji su im planovi rada za danas. Ukoliko postoje nekakve pogreške, bitno je obavijestiti ostatak tima kako bi svi bili sinkronizirani.

Iteracije se protežu tokom dva tjedna, a u tom periodu programeri i testeri pokušavaju implementirati i testirati određene funkcionalnosti kako bi postigli cilj iteracije. Prije početka iteracije, definira se cilj iteracije i konkretni zadaci. Jedna iteracija se sastoji od: daily-a, retrospektive (eng. *Retrospective*) i planiranja (eng. *Planning*). Prije zadnjeg dana iteracije, recenziraju se obavljeni zadaci. Zatim, tim se sastaje i dogovara koje će zadatke prikazati na

osvrtu iteracije (eng. *Iteration Review*). Samo testirani i zatvoreni zadaci mogu biti prezentirani. Netko iz tima priprema kratku demonstraciju (eng. *Demo*) kako bi se pokazale najbitnija rješenja te iteracije. Retrospektiva se održava se jednom mjesečno, nakon svake dvije iteracije. Tim razgovara o slijedećim stavkama:

- Što mi se svidjelo od zadnje retrospektive?
- Što sam naučila?
- Što bi moglo biti unaprijeđeno?
- Što bih htjela da je bilo napravljeno?

Nakon retrospektive, kreće planiranje slijedeće iteracije. Definira se cilj iteracije, zadatci koji nisu riješeni u prošloj iteraciji prebacuju se u novu. Ako je potrebno, novi zadaci su dodani u iteraciju. Tim zajedno odlučuje o novim zadacima te im se pridodaje vrijednost po bodovima (eng. *Story Point)*. Bodovi zadatka opisuju koliko je potrebno vremena i truda uložiti u određeni zadatak.

Kanban je metoda za razvoj programskih proizvoda i procesa sa naglaskom na just-in-time (hrv. *Točno na vrijeme*) isporuku, dok se pri tome ne opterećuju programeri. Stupci koji se nalaze na Kanban ploči, služe za opisivanje stanja zadataka:

- Aktivno (eng. *Active*)
- U tijeku (eng. *In Progress*)
- Recenzija koda (eng. *Code Review*)
- Testiranje (eng. *Test*)
- Odbijeno (eng. *Rejected*)
- Riješeno (eng. *Resolved*)
- Zatvoreno (eng. *Closed*)

5.3. Opis istraživanja

U ovom dijelu radu provodi se istraživanje u kojem se uspoređuje automatsko i ručno testiranje. Primjeri ručnog i automatskog testiranja odraditi će se u BizDataX programu. U istraživanju su zadana tri testna slučaja koji su potom testirani na ručni i automatski način. Tokom kompletnog procesa testiranja prate se određeni parametri po kojima će se kasnije uspoređivati ručna i automatska vrsta testiranja. Kako bi mogli pratiti parametre prvo ih treba definirati. Parametri koje se analizira u ovome istraživanju:

- Prolaznost testova
- Potrebno znanje o programiranju
- Potrebni dodatni alati
- Testiranje UI-a
- Vrijeme izrade testa
- Vrijeme izvršavanja testa
- Pokrivenost testova
- Mogućnost ponavljanja testa
- Točnost rezultata testa
- Potrebna radna snaga
- Visina troškova
- Održavanje testova

Opis ovog istraživanja je usporedba ručnog i automatskog testiranja te njihovih glavnih karakteristika. Istraživanje o prolaznosti testova se provodi na uzorku od tri ručna testa i tri automatizirana testa. Testira se jedna funkcionalnost u tri slučaja. Funkcionalnost koja se testira služi za instalaciju dodataka koji kasnije služe za analiziranje i otkrivanje osjetljivih podataka iz željene baze podataka.

Slijede tri testna scenarija koja će biti korištena za ručni i za automatski test:

1. Instalacija dodatka: korisnik treba instalirati dodatak putem dijaloga za registraciju dodataka (eng. *Happy path*).
2. Deinstalacija dodatka: korisnik treba deinstalirati dodatak putem padajućeg izbornika (eng. *Happy path*).
3. Instalacija postojećeg dodatka: korisnik treba instalirati postojeći dodatak putem padajućeg izbornika (eng. *Unhappy path*).

Temeljem ovoga istraživanja o prolaznosti testova, prikupljati će se podaci i o osnovnim karakteristikama putem jasno definiranih parametara. Tako će se moći usporediti općenito iskustvo i razlika između ručnih i automatskih testova.

5.4. Rezultati istraživanja

Na temelju provedenog istraživanja u BDX programu, obrađeni podaci te rezultati istraživanja nalaze se u Tablica 5.1.

Parametri	Ručno testiranje	Automatsko testiranje
Prolaznost testova	2/3	3/3
Potrebno znanje o programiranju	Ne	Da
Potrebni dodatni alati	Ne	Da
Testiranje UI-a	Da	Ne
Vrijeme izrade testa	Kraće	Duže
Vrijeme izvršavanja testiranja	Duže	Kraće
Pokrivenost testova	Manja	Veća
Mogućnost ponavljanja testa	Manja	Veća
Točnost rezultata testa	Manja	Veća
Potrebna radna snaga	Veća	Manja
Visina troškova	U početku manji	U početku veći
Održavanje testova	Jednostavno	Komplicirano

Tablica 5.1 Rezultati istraživanja i usporedba ručnog i automatskog testiranja

Kod parametra prolaznosti testova, važno je primijetiti kako su gotovi svi testovi prošli bez greške. Kod trećeg ručnog testa, otkrivena je pogreška tijekom korištenja korisničkog sučelja.

Opis pogreške:

Kada korisnik odabere postojeći dodatak za instalaciju, poruka o pogrešci se ne prikazuje.

Očekivani rezultat:

Polje sa odabranim dodatkom je označeno crvenim obrubom, poruka o pogrešci je napisana ispod polja sa odabiranim dodatkom, gumb za potvrdu dijaloga je neaktivan.

Stvarni rezultat:

Polje sa odabranim dodatkom je označeno crvenim obrubom, gumb za potvrdu dijaloga je neaktivan. Nedostaje poruka o pogrešci ispod polja sa odabranim dodatkom.

Za razliku od ručnih testova, automatski testovi se ne koriste za otkrivanje grešaka korisničkog sučelja, njihova temeljna svrha je ponavljanje testova koji su ključni za rad programskog proizvoda.

5.4.1. Prednosti

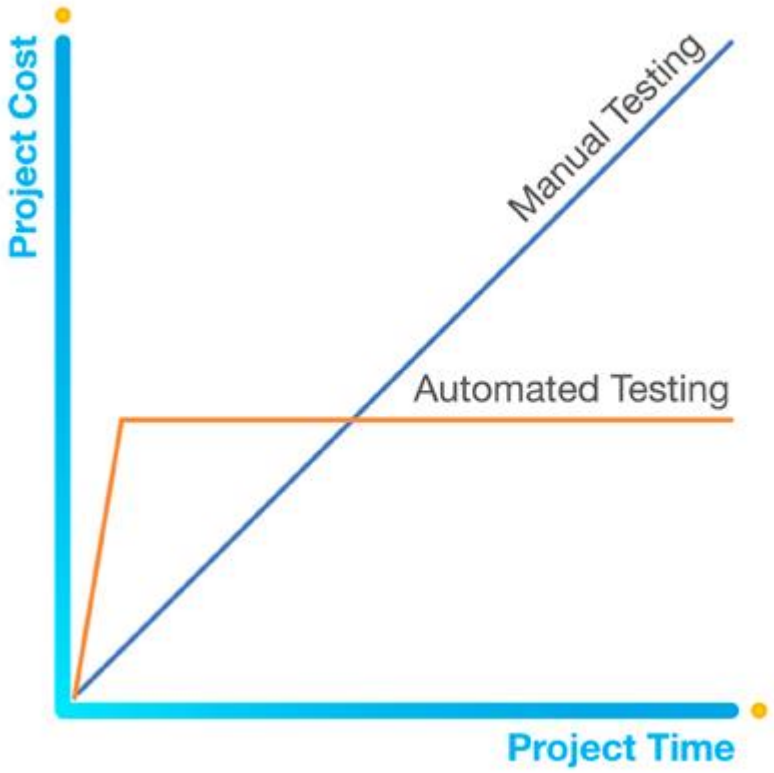
Ručno testiranje ima prednost naspram automatskog po pitanju jednostavnosti korištenja. Nije potrebno znati kodirati niti je potrebno koristiti dodatne alate. Vrijeme izrade testova je kraće. Također, ručno testiranje je idealno za korisničko sučelje kao i opće iskustvo korisnika. Održavanje i nadogradnja testova je jednostavna. Uz sve ovo navedeno, valja spomenuti i na manje troškove prilikom početka razvoja, no ako se sve prepusti ručnom testiranju, cijena testiranja će ubrzo rasti.

Najbitnija prednost automatskog testiranja je veliki opseg pokrivenosti testova i mogućnost svakodnevnog ponavljanja ključnih funkcionalnosti. To je ujedno i sigurnosna metoda provjere najvitalnijih komponenti programskog proizvoda. Također, točnost rezultata testova je veća iz razloga što ne postoji mogućnost ljudske pogreške, ne računajući na situacije kada je testna skripta razlog za pad testa. Treba spomenuti da nije potrebna velika količina radne snage u obliku testera, potrebna je osoba koja je dedicirana za automatizaciju testiranja. Što se tiče cijene testiranja, na početku je veća zbog kupnje i postavljanja testnog alata i okruženja za rad ali s vremenom, barem na većim projektima se isplati.

5.4.2. Nedostaci

Jedan od nedostataka kod ručnih testova je to što je pokrivenost testiranja malena, točnije testiranje se može obavljati na puno manjem opsegu naspram automatskih testova. Ako je potrebno pokriti testiranjem neku veću funkcionalnost, onda to oduzima puno vremena. Također, potrebno je konstantno ponavljati testiranje nad specifičnim funkcionalnostima, što je vrlo težak i monoton posao. Uz to, potrebno je više testera kako bi se brže i efikasnije testiralo.

Kod automatskih testova, nedostatak je to što tester mora znati programirati te je potreban testni alat za provedbu automatiziranih testova. Programiranje testnih skripti koje funkcioniraju oduzima puno vremena. Također, nemoguće je testirati korisničko sučelje i korisničko iskustvo kroz automatske testove. Manje greške će prije promaknuti prilikom automatskih testova naspram ručnih testova. Kompliciranо je održavati automatske testove, osim što je potrebno ponovno kodirati, potrebno je prilagoditi kod onom trenutnom koji je konstantno razvija i nadograđuje. Kod manjih projekata ne isplati se odmah ulagati u automatske testove. Kada se automatski testovi izvršavaju, pogotovo ako su u pitanju veći projekti, to može potrajati satima. Također, ako su testovi povezani, čim jedan padne – padaju svi. Slika 5.3 (vlasništvo: *itechcraft.com*) prikazuje odnos troškova i vremena sa ručnim i automatskim testiranjem.



Slika 5.3 Troškovi ručnog i automatskog testiranja kroz vrijeme

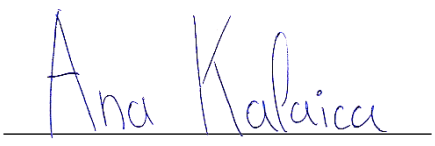
6. Zaključak

Ljudi nisu savršena bića i često griješe. Tijekom procesa razvoja proizvoda u kojem sudjeluju ljudi greške su neizbježne. Kako bi se minimizirali defekti i greške na proizvodu potrebno ga je testirati. Testiranje programskog proizvoda je proces ocjenjivanja i provjeravanja funkcionalnosti koje bi program ili aplikacija trebala imati. U procesu razvijanja programskog proizvoda greške se mogu javiti na svakom koraku. Greške koje se javljaju tijekom razvoja proizvoda ponekad su zanemarive, no u nekim slučajevima greške mogu biti strašne i skupe. Kada proizvođač plasira na tržište proizvod koji prethodno nije testiran postoji mogućnost da će se javiti greška koja može ugroziti dobrobit klijenata. Posljedice takvih grešaka su ozbiljne, zbog toga je testiranje neizostavan dio razvoja svih proizvoda.

Kako bi se programski proizvod brže i jeftinije finalizirao bitno je koristiti adekvatne metode testiranja tokom njegovog razvoja. Krivi pristup testiranju može dodatno poskupiti i produžiti kompletni proces razvoja. Automatizacijom testova se može ubrzati proces testiranja. Iako je tako, određene funkcionalnosti se ne mogu testirati automatskim testovima. Ručnim testiranjem se može preciznije testirati programski proizvod, no ovaj pristup zahtjeva puno strpljenja i truda što može biti presporo za današnji ubrzani način života.

Usporedbom ručnog i automatskog testiranja zaključila sam da je svaka od ovih metoda vrlo bitna u razvoju programskog proizvoda. Ako su u pitanju manji projekti, automatski testovi nisu nužni. No, ako su u pitanju projekti većeg raspona podataka i funkcionalnosti, potrebno je automatizirati proces testiranja ključnih elemenata programa kako bi ubrzali sveukupni tijek i isporuku proizvoda. Što se tiče ručnih testova, oni će uvijek biti temelj testiranja te bez ručnih testova nema niti automatskih. Smatram da će se u budućnosti još više automatizirati procesi testiranja, kako umjetna inteligencija polako zauzima svoje mjesto u procesu testiranja, tako će drastično nastaviti se razvijati i ubrzavati proces programskih proizvoda.

U Varaždinu, 27. rujna 2022.



Potpis studenta

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, ANA KALAICA *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica ~~završnog~~/diplomskog (*obrisati nepotrebno*) rada pod naslovom TESTIRANJE KAO KLJUČNI DIO RAZVOJA PROGRAMSKOG PROIZVODA (*upisati naslov*) te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)
Ana Kalaica
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, ANA KALAICA *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom ~~završnog~~/diplomskog (*obrisati nepotrebno*) rada pod naslovom TESTIRANJE KAO KLJUČNI DIO RAZVOJA PROGRAMSKOG PROIZVODA (*upisati naslov*) čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)
Ana Kalaica
(vlastoručni potpis)

7. Literatura

[1] J. Tian, Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement, Hoboken, New Jersey: John Wiley & Sons, Inc., 2005.

[2] H. v. Vliet , Software Engineering: Principles and Practice, Wiley, 2008.

[3] R. Patton, Software Testing, Sams Publishing, 2005.

[4] F. L. Morris i C. Jones, »An Early Program Proof by Alan Turing,« *IEEE Annals of the History of Computing,* br. Volume 6, Issue 2, p. 139–143, 1984.

[5] G. J. Myers , C. Sandler i T. Badgett, The Art of Software Testing, Wiley, 2011.

[6] D. Gelperin i B. Hetzel, The growth of software testing, New York, NY, United States: Association for Computing Machinery, 1988.

[7] G. D. Everett i R. McLeod, Software Testing: Testing Across the Entire Software Development Life Cycle, Wiley-IEEE Computer Society Pr, 2001.

[8] D. Graham , E. v. Veenendaal, I. Evans i R. Black, Foundations of Software Testing ISTQB Certification, Cengage Learning EMEA, 2012.

[9] A. Spillner, T. Linz i H. Schaefer, Software Testing Foundations: A Study Guide for the Certified Tester Exam, Rocky Nook, 2011.

[10] A. Mamić, »Metode i tehnike testiranja softvera,« 2021. [Mrežno]. Available: https://urn.nsk.hr/urn:nbn:hr:148:993225.

[11] D. Fundak, »Automatizacija testiranja moblinih aplikacija,« 2018. [Mrežno]. Available: https://urn.nsk.hr/urn:nbn:hr:200:112943.

[12] N. Kukuljan, »Testiranje web aplikacija,« 2022. [Mrežno]. Available: https://urn.nsk.hr/urn:nbn:hr:195:326338.

[13] J. Dudaš, »Testiranje programskih proizvoda,« 2019. [Mrežno]. Available: https://urn.nsk.hr/urn:nbn:hr:211:723018.

[14] K. Radić, »Važnost testiranja za kvalitetnu izradu web rješenja,« 2020. [Mrežno]. Available: https://urn.nsk.hr/urn:nbn:hr:225:568396.

[15] I. Jukić, »Automatsko testiranje programa,« 2018. [Mrežno]. Available: https://urn.nsk.hr/urn:nbn:hr:211:043593.

[16] M. Ivandić, »Važnost, tipovi i primjeri testiranja programskih proizvoda,« 2017. [Mrežno]. Available: https://urn.nsk.hr/urn:nbn:hr:124:385130.

[17] J. Osivčić, »Metode i metodologije u testiranju softvera,« 2019. [Mrežno].

[18] V. Šarić, »Izrada i testiranje mobilne aplikacije za pregled preostalog broja pacijenata u redomatskom redu,« 2019. [Mrežno].

[19] J. Viljevac, »Važnost osiguravanja kvalitete softvera manualnim i automatiziranim testiranjem,« 2021. [Mrežno]. Available: https://urn.nsk.hr/urn:nbn:hr:200:066090.

[20] N. Vaidya, »Login automation using Selenium Webdriver: Tutorial,« 2021. [Mrežno]. Available: https://www.browserstack.com/guide/login-automation-using-selenium-webdriver.

[21] BANKA.HR, »Ekobit implementira BizDataX u Intesa Sanpaolo Card,« 2015. [Mrežno]. Available: https://www.poduzetnistvo.org/news/ekobit-implementira-bizdatax-u-intesa-sanpaolo-card.

[22] Ekobit, »BizDataX,« [Mrežno]. Available: https://bizdatax.com/.

Popis slika

Popis tablica