METODE I TEHNIKE TESTIRANJA SOFTVERA

Mamić, Andreja

Professional thesis / Završni specijalistički

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Economics and Business / Sveučilište u Zagrebu, Ekonomski fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:148:993225

Sveučilište u Zagrebu

Ekonomski fakultet Zagreb

Poslijediplomski specijalistički studij

Informatički menadžment

METODE I TEHNIKE TESTIRANJA SOFTVERA

Poslijediplomski specijalistički rad

Andreja Mamić

Zagreb, travanj 2021.

Sveučilište u Zagrebu

Ekonomski fakultet Zagreb

Poslijediplomski specijalistički studij

Informatički menadžment

METODE I TEHNIKE TESTIRANJA SOFTVERA

SOFTWARE TESTING METHODS AND TECHNIQUES

Poslijediplomski specijalistički rad

Student: Andreja Mamić

Matični broj studenta: PDS-10-2020

Mentor: Prof. dr. sc. Mario Spremić

Zagreb, travanj 2021.

Sažetak na hrvatskom jeziku

Razvoj softvera je kompleksan zadatak koji se sastoji od velikog broja različitih faza i koraka. Razvoj podrazumijeva korištenje različitih tehnologija, metoda i tehnika. Kako bi se osigurala isporuka dobrog softvera postoje određene zakonitosti, pravila i definicije koje je potrebno poznavati. Upravo poznavanje i primjena različitih metodologija u razvoju softvera omogućuju pouzdanost i kvalitetu. Kao dio rada bit će opisan životni ciklus razvoja softvera te metode i njihovi principi koji to omogućuju. Jedan od neophodnih koraka u životnom ciklusu razvoja softvera jest testiranje softvera koje je predmet analize ovog rada. Testiranje softvera je postupak pronalaska grešaka s ciljem procjene sposobnosti softvera za rad. Temeljni cilj testiranja softvera je isporuka kvalitetnog i pouzdanog softvera za upotrebu. Ovaj rad ukazuje na važnost provođenja testova te nastoji ukazati na potrebu za testiranjem softvera u njegovom razvoju. U radu je prikazana studija različitih metoda, tehnika, razina i tipova testiranja. Iako je tijekom testiranja ponekad teško pronaći greške neophodnim se pokazalo poznavanje metoda čiji odabir može biti ključ prave kombinacije. Na kraju samog rada provedeno je istraživanje usporedbom prednosti i nedostataka primjene pojedine metode testiranja tijekom razvoja platnog sustava. Istraživanje se zasniva na iskustvenoj metodi koja je omogućila uvid i zaključivanje o stanju procesa testiranja platnog sustava.

Ključne riječi: metodologije razvoja softvera, testiranje softvera, ciljevi i principi testiranja, metode, razine i tipovi testiranja softvera, platni sustavi

Sažetak na engleskom jeziku

Software development is a complex task that consists of a large number of different phases and steps. Development involves the use of different technologies, methods and techniques. To ensure the delivery of good software there are certain principles, rules and definitions that need to be known. It is the knowledge and use of different methodologies in software development that enables reliability and quality. The software development life cycle, methods and their principles will be described as part of this paper. One of the necessary steps in the software development life cycle is the software testing that is the subject of the analysis of this paper. Software testing is process of evaluating the software with intention to find out error in it. The basic goal of software testing is to deliver quality and reliable software for use. This paper shows the importance of software testing in software development. The paper presents a study of different methods, techniques, levels and types of testing. Although is sometimes difficult to find errors during test, it is proved as necessary to know different methods and selection of right method can be key of successful combination. At the end of the paper, a research was conducted comparing the advantages and disadvantages of applying individual testing methods during the development of the payment system. The research is based on experiential method that provided insight about state of payment system testing process.

Key words: software development methodologies, software testing, testing goals and principles, methods, levels and types of software testing, payment system

IZJAVA O AKADEMSKOJ ČESTITOSTI

Izjavljujem i svojim potpisom potvrđujem da je poslijediplomski specijalistički rad / seminarski rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu, a što pokazuju korištene bilješke i bibliografija.

Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog izvora te da nijedan dio rada ne krši bilo čija autorska prava.

Izjavljujem, također, da nijedan dio rada nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Marić

(vlastoručni potpis studenta)

Zagreb, 26.04.2021.

(mjesto i datum)

STATEMENT ON THE ACADEMIC INTEGRITY

I hereby declare and confirm by my signature that the final thesis is the sole result of my own work based on my research and relies on the published literature, as shown in the listed notes and bibliography.

I declare that no part of the thesis has been written in an unauthorized manner, i.e., it is not transcribed from the non-cited work, and that no part of the thesis infringes any of the copyrights.

I also declare that no part of the thesis has been used for any other work in any other higher education, scientific or educational institution.

Marić

(personal signature of the student)

Zagreb, 26.04.2021.

(place and date)

SADRŽAJ

1. UVOD

1.1. Definiranje predmeta istraživanja

Konstantni razvoj tehnologije nameće sve veće zahtjeve u razvoju softvera koji s vremenom postaje sve kompleksniji uz stalni porast njegove primjene. Kako softver postaje kompleksniji stvara se sve veća potreba za osiguranjem njegove kvalitete u smislu pouzdanosti, lakoće za održavanje, otpornosti na greške.

U razvoju softvera pogreške se mogu pojaviti u bilo kojoj fazi životnog ciklusa. Stoga se važnost osiguranja kvalitete nikako ne može zanemariti. Predmet istraživanja ovog rada je testiranje kojim se osigurava da softver bude pouzdan i siguran za korištenje.

Testiranje softvera je postupak procjene njegove funkcionalnosti s namjerom utvrđivanja udovoljava li razvijeni softver navedenim specifikacijama ili ne, te da se utvrde eventualni nedostaci kako bi se osiguralo da nema grešaka koje bi utjecale na rad sustava. Dakle, testiranje je proces izvođenja programa s namjerom pronalaska različitih vrsta pogrešaka (funkcionalne, sistemske, logičke, performansne, sigurnosne, pogreške koje onemogućavaju interakciju pojedinih dijelova informacijskih sustava, itd). Testiranje se često smatra zamornim poslom iz razloga jer je kompleksno i ponavljajuće, a svaki ispravak greške zahtijeva ponovo testiranja dijelova, a ponekad i cijelog sustava. Smanjenju vremena testiranja i povećanju produktivnosti svakako bi doprinijela automatizacija testiranja.

U današnje vrijeme automatiziranje testiranja postaje sve učestalija tema, a sve više tvrtki odlučuje se na barem djelomično automatiziranje testova. Prema istraživanju Sauce Labs iz 2018. godine koje je provedeno na 1091 informatičkih tvrtki 8% ispitanika provodi isključivo ručna testiranja, njih 36% djelomično provodi automatske testove ali se ipak više oslanjaju na ručno testiranje, 28% podjednako kombinira ručno i automatizirano testiranje, njih 26% većinom provodi automatizirane testove ali dio testova još uvijek provode ručno, dok svega 2% ispitanika koristi isključivo automatizirane testove.[1]

[1] Sauce Labs (2018). *Testing trends for 2018* [online]. Sauce Labs. Dostupno na: https://saucelabs.com/resources/white-papers/testing-trends-for-2018 [15. siječnja 2021.]

Do danas nije razvijena metodologija testiranja softvera koja bi garantirala da će programsko rješenje raditi bez grešaka, ali sistematiziranim i planiranim testiranjem značajno se može povećati povjerenje u rad samog softvera te skratiti vrijeme potrebno za testiranje.

U ovom radu će se kroz više poglavlja opisivati sam proces testiranja kao i njegov životni ciklus. Pritom će se poseban naglasak staviti na metodologije testiranja. S obzirom na to da je testiranje softvera kompleksan pojam i sastoji se od različitih metoda i tehnika, svakoj od tih metoda i tehnika pridat će se posebna pažnja kako bi se naglasila važnost testiranja kojim se osigurava kvalitetno programsko rješenje.

Na kraju samog rada napravit će se komparacija ručnog i automatiziranog testiranja prilikom faze razvoja platnog sustava. Studijom slučaja nastojat će se prikazati koje su prednosti i nedostaci ručnog testiranja te prednosti i nedostaci automatiziranog testiranja u platnom sustavu. Na kraju će biti izneseno zaključno mišljenje i preporuke za poboljšanje procesa testiranja. Cilj rada je ukazati kako bi se automatizacijom pojedinih dijelova testiranja poboljšala kvaliteta i ubrzao proces testiranja platnog sustava.

1.2. Ciljevi rada

Osnovni ciljevi istraživanja ovog rada su:

1. definirati proces testiranja kao i važnost testiranja u osiguranju kvalitete softvera
2. definirati razne metode i vrste testiranja softvera
3. istražiti prednosti i nedostatke ručnog testiranja u procesu razvoja platnog sustava
4. istražiti prednosti i nedostatke automatiziranog testiranja u procesu razvoja platnog sustava
5. usporediti rezultate istraživanja iz prethodne dvije točke te dati preporuke za poboljšanje testiranja na temelju dobivenih rezultata

Rezultat navedenih ciljeva trebao bi uputiti na važnost automatizacije ponavljajućih testova koji su se provodili prilikom razvoja platnog sustava. Očekivani rezultat istraživanja trebao bi pokazati kako bi se automatizacijom ponavljajućih testova smanjilo vrijeme testiranja te povećala produktivnost zaposlenika.

1.3. Izvor podataka i metode istraživanja

Za pisanje rada koristit će se metode koje obuhvaćaju istraživanje, proučavanje i analiziranje izvora podataka koji uključuju domaću i stranu znanstveno-stručnu literaturu i članke te relevantne izvore iz područja informacijsko komunikacijske tehnologije, umjetne inteligencije, razvoja softvera, testiranja softvera i osiguranje kvaliteta softvera.

Kao izvor primarnih podataka koristit će se analiza i usporedba učinkovitosti ručnog i automatiziranog testiranja platnog sustava.

Kroz rad će se izmjenjivati različite metode, kao što su: metoda analize i sinteze, metoda indukcije i dedukcije, metoda klasifikacije, komparativna metoda, metoda deskripcije, metoda apstrakcije i konkretizacije, povijesna metoda, metoda istraživanja studije slučajeva, grafičke metode.

1.4. Sadržaj rada

Specijalistički rad je strukturiran u šest poglavlja. Nakon prvog uvodnog poglavlja, u drugom poglavlju opisane su metodologije razvoja softvera, što uključuje opis i analizu klasičnih i agilnih metoda razvoja softvera. Osim toga u ovom poglavlju opisan je i životni ciklus razvoja softvera. U trećem poglavlju opisana je definicija, ciljevi i načela testiranja softvera, životni ciklus testiranja softvera kao i kako se procesom testiranja osigurava kvaliteta softvera. U četvrtom poglavlju detaljno su prikazane različite razine testiranja, metode, tipovi te metrike testiranja softvera. U petom poglavlju analiziran je proces testiranja u fazi razvoja instant platnog sustava. U ovom poglavlju opisan je platni sustav, prednosti i nedostaci ručnih i automatiziranih testova koji se korišteni prilikom testiranja platnog sustava. Osim toga u ovom poglavlju su iznesene preporuke za poboljšanje procesa testiranja platnog sustava. U zaključnom poglavlju su sumirani rezultati razrada iz prethodnih poglavlja.

2. METODOLOGIJE RAZVOJA SOFTVERA

Razvoj i implementacija softvera je opsežan, zahtjevan i dugotrajan proces. Podrazumijeva korištenje raznih tehnologija i metoda. Kvalitetan i pouzdan softver nije moguće razviti bez poštivanja pravila i redoslijeda korištenja tehnologija, tehnika i metoda. U ovo informacijsko doba softver se smatra jednim od velikih dostignuća ljudskog stvaralaštva. Razvoj softvera svakim danom postaje sve kompleksniji. Time se nameće zaključak da upravo taj softver mora biti isporučen na vrijeme s visokom kvalitetom i sa što manje grešaka. Ova problematika nameće korištenje odgovarajućih metoda za razvoj softvera, nasuprot neorganiziranom razvoju.

Metodologije razvoja softvera predstavljaju skup pravila, smjernica i načela koje se koriste u procesu istraživanja, planiranja, dizajniranja, razvoja, testiranja i održavanja softvera.[2]

Samo jedna metodologija razvoja softvera ne može djelovati na čitavom spektru različitih projekata. Sve metodologije i nisu primjenjive u svim projektima. Usprkos tomu, projektni menadžment treba identificirati specifičnu prirodu projekta te odabrati najprikladniju razvojnu metodologiju. Cilj razvoja softvera je proizvodnja kvalitetnog softvera, u zadanom vremenu, unutar predviđenog budžeta i uz zadovoljavanje stvarnih potreba naručitelja. Uspjeh projekta ovisi o dobrom upravljanju zahtjevima (engl. requirement management). Metodologija odabire metode, prilagođava ih konačnom cilju, propisuje redoslijed upotrebe metoda, propisuje proces modeliranja od početka do kraja životnog ciklusa softvera. Cjelovit pristup razvoju softvera podrazumijeva upotrebu metodologije koja u sebi sadrži metode primjenjive u pojedinim fazama razvoja softvera, tako da je svaka faza pokrivena s jednom ili više metoda.[3]

S obzirom na brojnost metodologija danas se najčešće spominju klasične i agilne metode razvoja softvera. U ovom poglavlju prikazna je njihova klasifikacija, razlike, prednosti i nedostaci.

---

[2] Liviu, M. (2014.) Comparative study on software development methodologies. *Database System Journal* [online], vol. V, no. 3/2014. Dostupno na: https://dbjournal.ro/archive/17/17_4.pdf [16. siječnja 2021.]

[3] Čubranić, D., Kaluža, M., Novak, J.(2013.) *Standardne metode u funkciji razvoja softvera u Republici Hrvatskoj*. Rijeka: ur. Zbornik Veleučilišta u Rijeci

Iako postoji trend u području razvoj softvera, niti jedna metodologija nije se pokazala primjenjiva na sva područja. U praksi svaka organizacija upravlja razvojem softvera na drugačiji način, koji se često razlikuje od projekta do projekta. Ipak, gotovo sve organizacije koriste neki podskup ili kombinaciju gore spomenutih metodologija.[4]

2.1. Klasične metode razvoja softvera

Klasičnim metodama razvoja softvera, ili kako se još češće u praksi nazivaju tradicionalne metode razvoja softvera, smatraju se one koje imaju slijedni (fazni) pristup odnosno pristup gdje se prelaskom na sljedeći proces više ne vraća na prethodni. Njihova osnovna karakteristika je da nisu prilagodljive niti podložne promjenama, vremenski okvir razvoja je jasno definiran, a faza razvoja se provodi i završava prema jasno utvrđenim aktivnostima i postupcima. Osim toga poznato je kako se u klasičnim metodama razvoja softvera svaka faza razvoja jasno i opsežno dokumentira. U nastavku će biti detaljnije opisani model vodopada i V-Model.

2.1.1. Model vodopada

Ovaj model izvorno je osmislio američki računalni znanstvenik Winston W. Royce 1970. godine. Pri opisivanju modela nije upotrijebio riječ 'vodopad' ali se zbog načina na koji je predstavio bit razvoja softvera, model nazvan upravo tako. Ključni detalj Royceovog modela bilo je pozicioniranje razvojnih faza prema slijedu njihova izvođenja. Ako se pretpostavi da je vrijeme u modelu vodopada raspoređeno po vodoravnoj osi, tada se može zaključiti da sljedeća faza razvoja slijedi tek nakon što je završena prethodna.[5]

Model vodopada (engl. Waterfall) glavni je predstavnik tradicionalnih modela vođenja projekata iz kojeg su prilagođavanjem nastale i ostale metodologije u tradicionalnom kontekstu. On predstavlja planski vođen proces jer se cijeli proces razvoja mora planirati i

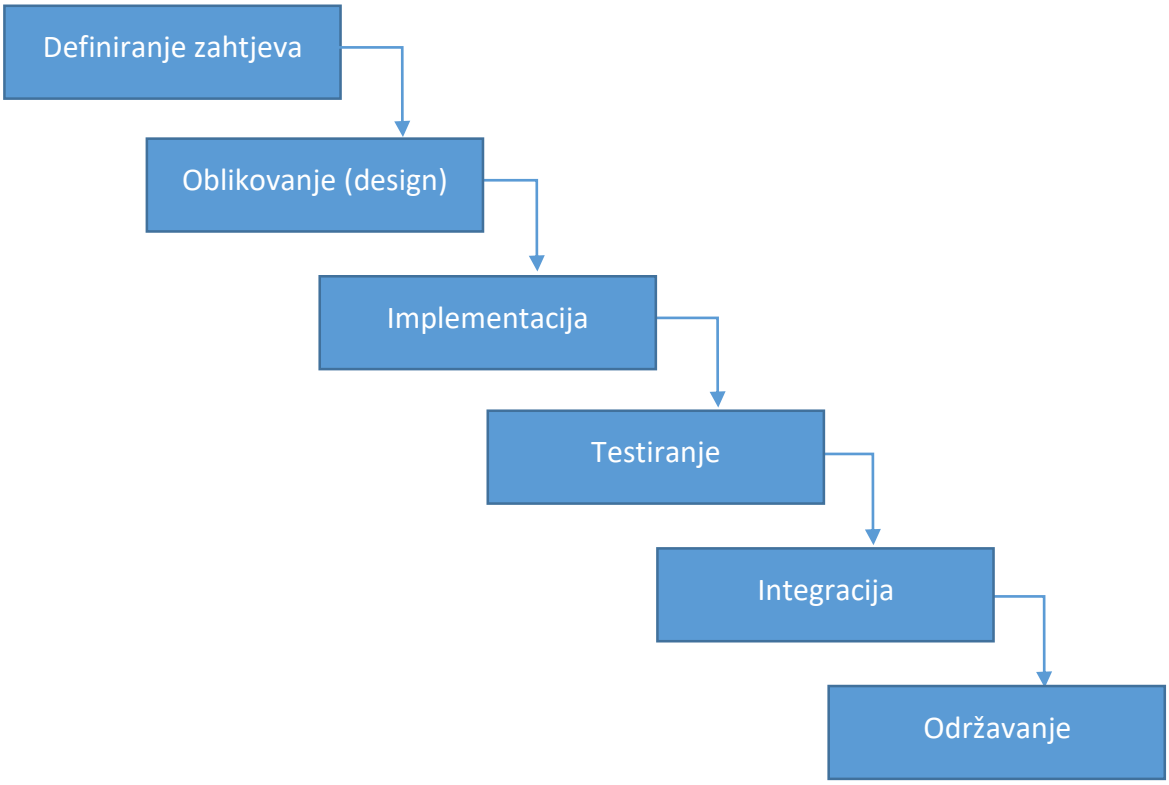
[4] Young, D. (2014). Software Development Metodologies, *ResearchGate* [online], Dostupno na: https://www.researchgate.net/publication/255710396_Software_Development_Methodologies [16. siječnja 2021.]

[5] Matković, P., Tumbas, P. (2010). A Comparative Overwiev of the Evolution of Software Development Models, *Journal of Industrial Engineering and Management* [online], 2(4). Dostupno na: https://www.researchgate.net/publication/267711880_A_Comparative_Overview_of_the_Evolution_of_Software_Development_Models [16.siječnja 2021.]

vremenski definirati što znači da se za sve aktivnosti moraju planirati termini izvođenja. Cijeli proces razvoja softvera podijeljen je u nekoliko zasebnih faza, a ishod jedne faze omogućuje ulaz u sljedeću fazu u nizu. Kao što je vidljivo na slici model vodopada sastoji se od nekoliko faza:

- analiza i definiranje zahtjeva (specifikacija)
- oblikovanje (engl. design)
- implementacija
- testiranje
- integracija sustava (eng. deployment)
- održavanje.

Slika 1. Faze modela vodopada



Izvor: Balaji, S., Murugaiyan Sundararajan, M. (2012). Waterfall vs V-Model vs Agile: A Comparative Study on SLDC, *International Journal of Information Technology and Business Management* [online], 2(1). Dostupno na: https://mediaweb.saintleo.edu/Courses/COM430/M2Readings/WATEERFALLVs%20V-MODEL%20Vs%20AGILE%20A%20COMPARATIVE%20STUDY%20ON%20SDLC.pdf [16.siječnja 2021.]

U fazi analize i definiranja zahtjeva uz konzultaciju s korisnicima sustava definiraju se ciljevi, ograničenja, zahtjevi s poslovne i tehničke strane. Svi zahtjevi sustava koji se trebaju razviti u ovoj fazi obuhvaćeni su i dokumentirani u dokumentu sa specifikacijama zahtjeva.

U drugoj fazi, oblikovanje ili dizajniranje, proučavaju se zahtjevi iz prve faze te se priprema dizajn sustava. Dizajn sustava pomaže u specificiranju hardverskih i sistemskih zahtjeva ali i pomaže u utvrđivanju cjelokupne arhitekture sustava čime se definira njihova međuovisnost.

Rezultatima iz faze dizajniranja projektno rješenja softvera realizira se skupom manjih programskih jedinica, a koje se integriraju u sljedećoj fazi. Svaka programska jedinica razvijena u ovoj fazi testira se jediničnim testovima (engl. Unit test). Ovim testovima utvrđuje se ostvaruje li svaka programska jedinica svoju planiranu funkciju. Sve programske jedinice razvijene u prethodnoj fazi integriraju se u sustav nakon čega se testiraju kako bi se provjerilo zadovoljava li softver zahtjeve, ali ovaj puta kao sustav. Nakon što se provedu sva funkcionalna i nefunkcionalna testiranja softver se isporučuje naručitelju.

Faza održavanja sustava je obično najduža faza. U ovoj fazi softver je već u upotrebi. Održavanje obuhvaća ispravljanje grešaka koje nisu otkrivene u ranijim fazama, poboljšanje softvera u smislu izdavanja novih verzija softvera koje će također biti skladu sa svim postavljenim specifikacijama.

Zahvaljujući načinu na koji je model definiran omogućeno je detaljno planiranje cijelog procesa, može se jednostavno ustanoviti trenutna faza projekta. Osim toga, ovaj model omogućuje čvrstu kontrolu nad projektom uz pomoć detaljne dokumentacije. S druge strane nedostatak ovog modela je da ne dozvoljava puno odmaka od planiranog. Jednom kada je aplikacija u fazi testiranja, vrlo je teško vratiti se i promijeniti nešto što u prethodnim fazama nije bilo dobro dokumentirano.

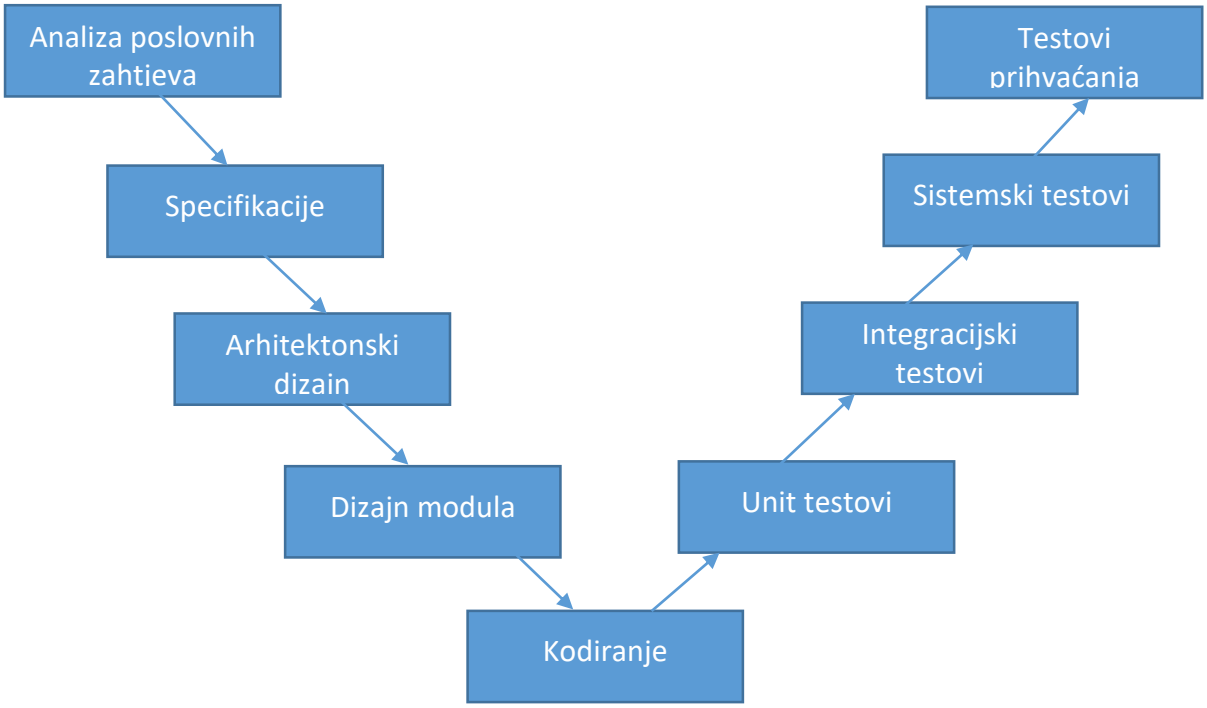
2.1.2. V- Model

V-Model (engl. Validation and Verification model) je modificirana verzija modela vodopada te predstavlja njegovu ekstenziju. Osnovna razlika između modela vodopada i V-Modela jest u tome što V-Model nije dizajniran u linearnoj osi. V-Model se sastoji od više aktivnosti te nudi više mogućnosti interakcija između pojedinih aktivnosti. Zbog svog dizajna ovaj model je jednostavan za korištenje i svaka od faza ima specifične rezultate. Razvojni proces je uravnotežen, oslanja se na provjeru rezultata iz prethodnih faza. Rezultat svake faze se pomno provjerava. To znači da je za svaku pojedinu fazu u razvojnom ciklusu izravno povezana faza

testiranja. Ovo je visoko discipliniran model i sljedeća faza započinje tek nakon završetka prethodne faze. Kao što je vidljivo na slici 2. V-Model sastoji se od nekoliko faza:

- analiza poslovnih zahtjeva
- definiranje specifikacija
- arhitektonski dizajn (engl. High-Level Design)
- dizajn modula (engl. Unit Low-Level Design)
- kodiranje/programiranje
- unit testovi
- integracijski testovi
- sistemski testovi
- testovi prihvaćanja.

Slika 2. Faze V-Modela



Izvor: Balaji, S., Murugaiyan Sundararajan, M. (2012). Waterfall vs V-Model vs Agile: A Comparative Study on SLDC, *International Journal of Information Technology and Business Management* [online], 2(1). Dostupno na: https://mediaweb.saintleo.edu/Courses/COM430/M2Readings/WATEERFALLVs%20V-MODEL%20Vs%20AGILE%20A%20COMPARATIVE%20STUDY%20ON%20SDLC.pdf [16. siječnja 2021.]

Prva faza, faza analize poslovnih zahtjeva, uključuje detaljnu komunikaciju s kupcem radi razumijevanja njegovih očekivanja i točnih zahtjeva. Ovo je vrlo važna aktivnost i njome treba dobro upravljati, a poslovni zahtjevi kasnije mogu poslužiti kao ulaz testovima prihvaćanja.

U sljedećoj fazi, fazi specifikacije detaljno se opisuje kompletna postavka hardvera, softvera i komunikacije. Prema ovoj fazi se izrađuje plan testiranja, što znači ako se ova faze ispravno definira ostaje više vremena za stvarno provođenje testova.

U trećoj fazi se definiraju i dizajniraju arhitektonske specifikacije odnosno jasno se definira komunikacija između unutarnjih modula i drugih sustava. Pomoću informacija iz ove faze mogu se dizajnirati i dokumentirati integracijski testovi.

U četvrtoj fazi, dizajniranje modula, definira se detaljan unutarnji dizajn za sve sistemske module. Ono što je bitno za ovu fazu je da dizajn mora biti kompatibilan s ostalim modulima u arhitekturi sustava i ostalim vanjskim sustavima.

U fazi kodiranja odnosno programiranja odabire se najprikladniji programski jezik na temelju arhitektonskih zahtjeva. Prije nego bude stvarno isporučen kod prolazi razne verifikacije i validacije.

Sljedeće četiri faze nazivaju se još i faze provjere valjanosti. Faze provjere valjanosti podrazumijevaju testiranja na razini koda (unit testovi), integracijske testove odnosno testove provjere rada između različitih komponenti sustava, sistemske testove kojima se provjerava cjelokupna funkcionalnost sustava i na kraju testove prihvaćanja kojima se otkrivaju problemi kompatibilnosti s drugim sustavima.

Treba istaknuti kako ova metoda nudi više mogućnosti što ju čini jednostavnijom i lakšom za upravljanje. V-Model pokazuje kako se sve aktivnosti provode od vrha prema dolje i svaka faza ima specifične rezultate. Planovi se u ovome modelu mogu unaprijed razvijati jer je omogućena verifikacija i validacija proizvoda u ranijim fazama razvoja. S druge strane nedostatak ovog modela je nefleksibilnost za promjene.

2.2. Agilne metode razvoja softvera

Agilne metode razvoja softvera predstavljaju njihovu suprotnost klasičnim metodologijama. Suvremena povijest agilne metodologije započinje 2001. godine kada je skupina stručnjaka iz područja softverskog inženjerstva odlučila pokušati pronaći rješenje koje će biti suprotno nefleksibilnim tradicionalnim metodama. Rezultat njihovog rada je Agilni manifest kao inicijalni dokument i vodič za iterativno (agilno) vođenje projekata. Zahvaljujući svojim novim principima čiji je temelj kontinuirana isporuka, kratki razvojni ciklusi, visoka razina komunikacije te velika prilagodljivost, agilne metode postaju vodeća metodologija u razvoju softvera.

Agilni razvoj softvera može se definirati kao skup metodologija temeljenih na iterativnom razvoju gdje se zahtjevi rješavaju suradnjom između više timova. Agilne metode razvoja softvera osmišljene su kako bi riješile problem isporuke visokokvalitetnog softvera na vrijeme u poslovnom okruženju i u uvjetima koji se brzo mijenjaju. Glavna prednost agilnog razvoja softvera je omogućavanje prilagodljivog procesa što znači da se promjene mogu napraviti čak i u kasnim fazama procesa razvoja. Korištenjem više iteracija, primjenom agilnih metoda razvoja softvera omogućava se stvaranje kvalitetnog, funkcionalnog softvera s malim timovima i ograničenim resursima.[6]

Postoji nekoliko različitih metoda agilnog razvoja softvera, a neke od najpopularnijih su Scrum, ekstremno programiranje (XP), lean razvoj softvera (Lean Software Development), FDD (Feature Driven Development), DSDM (Dynamic Systems Development Method), kristalna obitelj metodologija. U nastavku će biti opisane Scrum, ekstremno programiranje i lean metodologija kao vodeće agilne metodologije.

2.2.1. Scrum

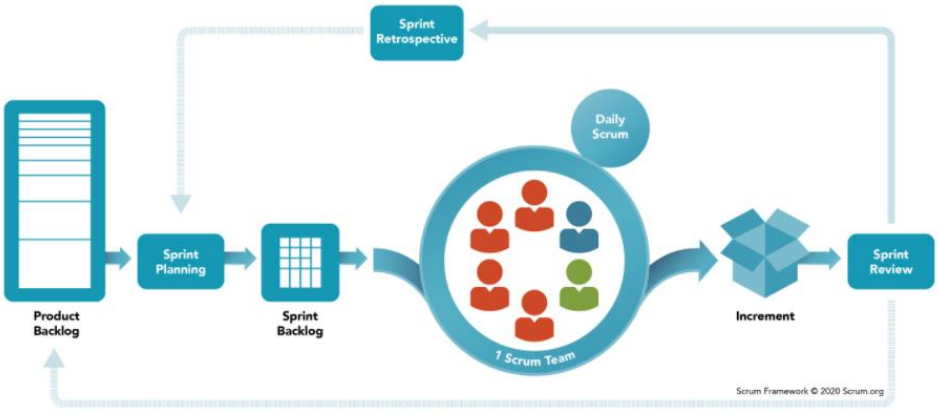
Ken Schwaber i Jeff Sutherlandom razvili su proces Scrum kako bi pomogli organizacijama koje se bore sa složenim razvojnim projektima. Scrum je ime zapravo dobio prema izrazu Scrum u ragbiju. Taj se izraz koristi za situaciju kada se nakon prekida protivnički timovi zbijaju na hrpe

[6] Mekni, M. , Buddhavarapu, G. , Chinthapatla, S. and Gangula, M. (2018). Software Architectural Design in Agile Environments. *Journal of Computer and Communications* [online], 6(1). Dostupno na: https://www.scirp.org/(S(czeh2tfqyw2orz553k1w0r45))/journal/paperinformation.aspx?paperid=81436 [20. siječnja 2021.]

i bore za posjed lopte. Svakim je prekidom (Scrum-om) tim sve bliže cilju i zauzima nove pozicije. Tim ne pokušava predvidjeti sve situacije koje se mogu dogoditi u igri već se prilagođava trenutnoj situaciji na terenu te nastoji ostvariti cilj.

Scrum je empirijski pristup koji se temelji na fleksibilnosti, prilagodljivosti i produktivnosti. Omogućuje programerima da u procesu implementacije odaberu određene tehnike, metode i prakse razvoja softvera. [7] Scrum se sastoji od Scrum timova i njihovih pridruženih uloga, događaja, artefakata i pravila. Na slici 3 je prikazan ciklus Scrum iteracije kao i njegovi sastavni dijelovi.

Slika 3. Ciklus Scrum iteracije



Izvor: Schwaber, K. (2021). *What is Scrum?* [online]. Scrum.org. Dostupno na: https://www.scrum.org/resources/what-is-scrum [20. siječnja 2021.]

Scrum tim sastoji se od vlasnika proizvoda (engl. Product Owner), razvojnog tima (engl. Development team) i Scrum mastera. Vlasnik proizvoda je odgovoran za rad razvojnog tima, dok je razvojni tim zadužen za isporuku inkrementacije proizvoda na kraju svakog Sprinta. Scrum muster osigurava da se timovi pridržavaju pravila i prakse Scruma.

Što se tiče događaja, Scrum koristi vremenski ograničene događaje i to tako da svaki događaj ima određeno vrijeme trajanja. Sprint se smatra središtem Scruma i predstavlja vremenski period od jednog mjeseca ili manje tijekom kojeg se mora isporučiti upotrebljiv inkrement proizvoda. Svakim Sprintom su definirana pravila kako će se obaviti zadaci kako bi se dobio končan inkrement proizvoda. Osim Sprinta kao događaj je bitan i dnevni Scrum koji

[7] Mekni, M. , Buddhavarapu, G. , Chinthapatla, S. and Gangula, M. (2018). Software Architectural Design in Agile Environments. *Journal of Computer and Communications* [online], 6(1). Dostupno na: https://www.scirp.org/(S(czeh2tfqyw2orz553k1w0r45))/journal/paperinformation.aspx?paperid=81436 [20. siječnja 2021.]

predstavlja petnaest minutni vremenski događaj koji služi kako bi razvojni tim uskladio aktivnosti i kako bi se razvio plan za sljedeća 24 sata.

Scrum artefakti omogućuju transparentnost i pregled ključnih informacija. Product Backlog predstavlja uređenu listu svih informacija potrebnih za proizvod kao i izvor zahtjeva za promjenama koje su potrebne za poboljšanje proizvoda. Sprint Backlog predstavlja plan i procjenu razvojnog tima te koji se zadaci trebaju odraditi tijekom Sprinta.

Budući da je platni sustav, o kojem će više biti riječi u petom poglavlju, podložan promjenama npr. promjena regulative ili poslovnih procesa, razvijan je prema Scrum metodologiji koja ostavlja dovoljno prostora i mogućnost prilagodbe za promjene. Razvoj i implementacija sustava vođeni su prema unaprijed definiranim obrascima koji osiguravaju kvalitetu i upravljivost procesa.[8]

2.2.2. Ekstremno programiranje

Ekstremno programiranje (engl. Extreme Programming) je metoda razvoja softvera čiji je cilj proizvodnja softvera visoke kvalitete. Jedna je od najčešće korištenih metoda među ostalim agilnim metodama, a naglasak stavlja na timski rad i suradnju između svih članova tima. Timovi planiraju malu količinu posla i izvršavaju ga u kratkim vremenskim intervalima koji se nazivaju iteracije. Ova metoda razvoja softvera temelji se na četiri osnovne vrijednosti:

- komunikacija
- jednostavnost
- povratne informacije
- hrabrost i poštovanje.[9]

Ekstremno programiranje sastoji se od 4 glavne faze koje su prikazane na slici ispod:

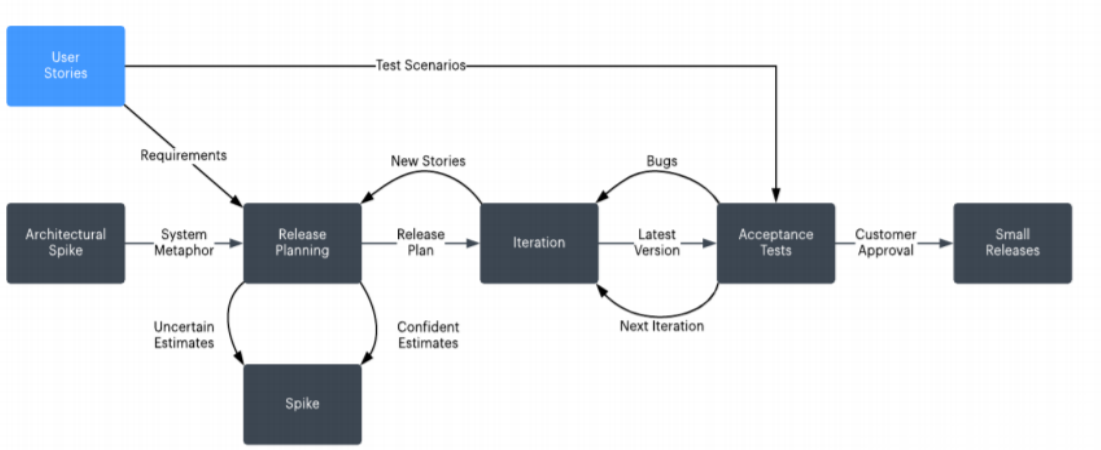
- planiranje
- dizajniranje
- kodiranje

[8] Autor rada prema internoj dokumentaciji

[9] Agility in Mind. (2020). *What is eXtreme Programming (XP)?: XP Values* [online]. Agility in Mind. Dostupno na: https://agility.im/frequent-agile-question/what-is-extreme-programming/ [26. siječnja 2021.]

- testiranje.[10]

Slika 4. Faze ekstremnog programiranja



Izvor: Poudel, R. (2019). Network Traffic Visualization with Scapy and ELK Stack*: Extreme Programming Methodology (Selected Methodology)* [online]. ResearchGate. Dostupno na: https://www.researchgate.net/publication/337304617_Network_Traffic_Visualization_with_Scapy_and_ELK_Stack [21. siječnja 2021.]

Prava faza je faza planiranja. Kako je cilj svakog sustava zadovoljiti potrebe klijenata, aktivnosti u ovoj fazi ponajviše ovise o klijentu i njegovim zahtjevima. U ovoj fazi klijent definira kako sustav treba raditi, nakon čega programeri pretvaraju korisničke priče (engl. user stories) u iteracije koje pokrivaju mali dio definirane funkcionalnosti. Komponente poput troškova, vremena i potrebnih resursa također se procjenjuju u ovoj fazi.

Nakon faze planiranja slijedi faza dizajniranja. Dizajniranje je način stvaranja strukture sustava. Bez pravilnog dizajna implementacija sustava postaje previše složena te je tako teško razumjeti i sam rad sustava. Prema tome glavni cilj faze dizajniranja je jednostavnost jer se jednostavnošću osigurava brža implementacija ali i održavanje sustava.

Glavna faza ove metodologije je kodiranje odnosno programiranje. Programiranje se temelji na pisanju visokokvalitetnog koda.

Testiranje je važna i presudna faza u ekstremnom programiranju. U ovoj fazi provode se osnovni testovi za provjeru ispravnosti implementiranih funkcija. Sav kod prolazi određena

[10] Rizwan Jameel Qureshi M., Ikram J.S. (2015) Proposal of Enhanced Extreme Programming Model: Introduction. *I.J. Information Engineering and Electronic Business* [online], 7(1). Dostupno na: http://www.mecs-press.org/ijieeb/ijieeb-v7-n1/IJIEEB-V7-N1-5.pdf [26.siječnja 2021.]

testiranja kako bi se eliminirale greške odnosno spriječilo njihovo pojavljivanje u drugim iteracijama. Neki od testova koji se provode u ekstremnom programiranju su jedinični testovi, testovi prihvaćanja i integracijski testovi koji će biti detaljnije opisani u sljedećim poglavljima.

2.2.3. Lean Software Development

Koncept Lean prvi put se pojavio pedesetih godina prošlog stoljeća u proizvodnom sektoru u Toyoti u Japanu. U Toyoti lean pristup značio je planiranje i optimizaciju proizvodnje tako da se smanje troškovi, resursi i skrati gubitak vremena. Krajem 19. stoljeća mnoge druge industrije počele su slijediti Toyotin primjer kada su uvidjele kako se ovim pristupom poboljšava učinkovitost, upravljanje vremenom i troškovima te maksimizira vrijednost kupca.

Lean razvoj softvera zamisao je Toma i Mary Poppendieck koji su koncepte lean proizvodnje prilagodili razvoju softvera. Ta zamisao rezultirala je razvojem lean softvera koji je postao široko prihvaćen u agilnim zajednicama. Mary i Tom Poppendieck definirali su sedam principa na kojima počiva lean razvoj:

1. otklanjanje otpada
2. pojačano učenje
3. odluči što je kasnije moguće
4. isporuči što brže je moguće
5. osnaži tim
6. izgradi integritet
7. sagledaj cjelinu.[11]

U nastavku teksta je ukratko opisan cilj i svrha pojedinog principa.[12]

Jedno od osnovnih principa koji upravo Lean čini uspješnim je otklanjanje otpada. Otklanjanje otpada odnosi se na uklanjanje viška zaliha, nepotrebnih napora, dupliciranih podataka i što je najvažnije troškova povezanih sa svim spomenutim.

[11] Poppendieck, M. i Poppendieck, T. (2003) *Lean Software Development: An Agile toolkit* : *Guided Tour*. Addison-Wesley Longman Publishing Co., Inc., USA.
[12] Svitis, C. (2013.) *Lean Software Development Theory validation in terms of cost-reduction and quality-improvement.* Bachelor of Science Thesis. University of Gothenburg.

Pojačano učenje podrazumijeva stvaranje novih znanja ali i dijeljenje naučenih znanja među članovima tima. Prenošenjem znanja među članovima tima povećava se produktivnost ali i fleksibilnost tima.

Princip 'odluči što je kasnije moguće' temelji se na ideji donošenja odluke u posljednjem odgovornom trenutku. Ovim principom se omogućuje da se velike i kritične odluke donose tek kada za njih postoji dovoljno valjanih informacija, a koje omogućuju donošenje ispravne odluke.

Princip 'isporuči što je brže moguće' odnosi se na isporuku manjih dijelova softvera. Manjim isporukama je lakše upravljati nego odjednom cijelim proizvodom. Također u manjim isporukama eventualne greške mogu biti primijećene prije, a samim time i njihov ispravak će biti implementiran ranije.

Principom 'osnaži tim' nastoji se skrenuti pažnja kako je neophodno stvoriti ozračje u kojem je svaki član tima važan i svakom članu tima se omogućuje da ostvari svoj puni potencijal.

Princip 'izgradi integritet' podupire potrebu za izgradnjom cjelovitog proizvoda s visokom kvalitetom. A kvaliteta se može postići ispravnim tokom informacija, povratnim informacijama od kupaca, revidiranjem koda, provođenjem testova, pravovremenim ispravljanjem grešaka.

Kod principa 'sagledaj cjelinu' je važno kontinuirano istraživati kako optimizirati cijeli sustav. Sustav se može optimizirati uvođenjem promjena, primjenom znanja s drugih projekata i primjenom iskustava naučenih na greškama.

Prema dosad navedenom može se zaključiti kako se lean razvoj softvera koncentrira na stvaranje vrijednosti, stvaranje kvalitete i uklanjanje gubitaka. Kontinuiranim usavršavanjem, međusobnim poštivanjem i uvažavanjem, stalnim naporima i ulaganjem u razvoj, lean razvoj softvera ističe se kao jedna od metoda koja potiče generiranje novih inovativnih ideja koje omogućuju stalni napredak.

2.3. Životni ciklus razvoja softvera

U početku se razvoj softvera svodio na računanje matematičkih zadataka nakon čega su programeri programirali cijeli postupak izračuna i pokušavali dobiti rezultat na temelju matematičkih izračuna. Sve složeniji računalni problemi potaknuli su programere na

automatizaciju zadataka te su tako stvoreni različiti pristupi u razvoju softvera. Ovakvi pristupi brzo su postali poznati kao modeli životnog ciklusa razvoja softvera (engl. Software Development Life Cycle - SDLC).

Prvi formalni model tzv. 'životni ciklus produkcije programa' je 1956. godine predstavio Herbert Bennington. Nekoliko godine kasnije, 1968. taj model je proširen tako da uključuje petlje povratnih informacija i iterativni razvoj.[13]

S razvojem tehnologije, sustavi su postali sve složeniji i kompleksniji te su samim time razvijeni modeli i okviri koji omogućuju organizirani razvoj softvera. U prilog tome ide činjenica da su i tradicionalni pristupi razvoja softvera prilagođeni kako bi udovoljili neprestano rastućim potrebama korisnika i organizacija.

Životni ciklus razvoja softvera (SDLC) predstavlja metodu kojom se softvera razvija na sustavan način čime se osigurava kvaliteta softvera ali i povećava vjerojatnost dovršenja projekta u roku. [14] Okvir životnog ciklusa razvoja softvera sastoji se od niza aktivnosti koje svi sudionici u razvoju softvera trebaju slijediti kako bi se razvio kvalitetan softver. Faze koje su prisutne u gotovo svakom razvoju softvera su:[15]

1. planiranje
2. analiza zahtjeva
3. dizajn arhitekture softvera
4. implementacija
5. testiranje i integracija
6. održavanje.

[13] Sherrill, C. (2017) *On The Sholders of Giants: A Brief History of SDLC Models* [online]. Business Analyst Coach. Dostupno na: https://businessanalystcoach.blog/2017/12/15/on-the-shoulders-of-giants-a-brief-history-of-sdlc-models/ [15. veljače 2021.]

[14] Apoorva, M., Deepty, D. (2013) A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios. *International Journal of Advance Research in Computer Science and Management Studies* [online], 1 (5). Dostupno na: https://www.researchgate.net/profile/Apoorva_Mishra2/publication/289526047_A_Comparative_Study_of_Different_Software_Development_Life_Cycle_Models_in_Different_Scenarios/links/59c00417458515e9cfd549a1/A-Comparative-Study-of-Different-Software-Development-Life-Cycle-Models-in-Different-Scenarios.pdf [08. veljače 2021.]

[15] Eby, K. (2017). *The Ultimate Guide to Understanding and Using a System Development Life Cycle* [online]. Smartsheet. Dostupno na: https://www.smartsheet.com/system-development-life-cycle-guide [08. veljače 2021.]

Na samom početku životnog ciklusa razvoja softvera nalaze se faza planiranja i faza analize zahtjeva. Ove faze izuzetno su važne jer predstavljaju temelj za dobar razvoj ostalih faza. Ove dvije faze podrazumijevaju pripremne radnje kao što su definiranje korisničkih zahtjeva, identificiranje neophodnih funkcionalnosti, planiranje troškova, planiranje rizika, definiranja snaga, slabosti sustava i ukupnih mogućnosti sustava. Ukratko tijekom faze planiranja i faze analize zahtjeva nastoji se utvrditi ukupni opseg projekta, što podrazumijeva ekonomske, operativne i ljudske čimbenike kao i jasno definirane vremenske rokove.

Nakon što je napravljeno planiranje i kompletna analiza svih zahtjeva slijedi faza dizajniranja arhitekture softvera. U ovoj fazi se pripremaju dokumenti za dizajn sustava prema dokumentima dobivenim iz prethodne dvije faze. Ti dokumenti najčešće sadrže informacije kao što su:

- definicije funkcionalnosti sustava
- odnosi i ovisnost među različitim dijelovima sustava
- definiranje baze podataka zajedno s njenim ključnim elementima
- tehničke specifikacije.

Ova faza postaje temelj za sljedeće faze koje slijede u životnom ciklusu razvoja softvera.

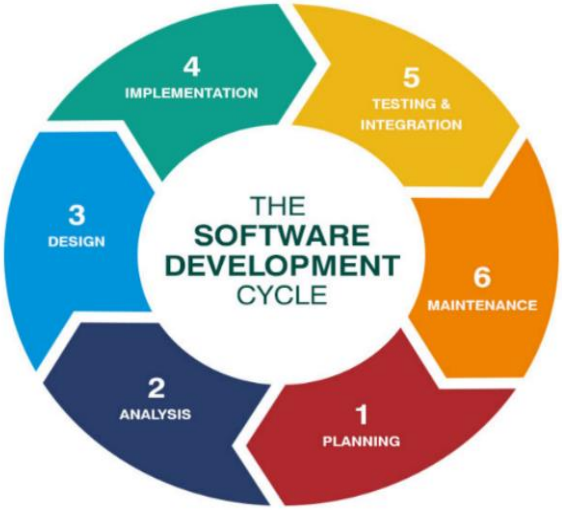
Nakon što je definiran dizajn sustava slijedi faza implementacije odnosno razvoja softvera. U ovoj fazi počinje razvoj programa odnosno pisanje koda odabranim programskim jezikom. Pisanje koda podijeljeno je na nekoliko modula odnosno manjih zadataka koji se zatim dodjeljuju timu za razvoj. Tim za razvoj čine programeri. Tijekom ove faze programeri koriste različite alate za pisanje koda kao što su kompajleri, interpreteri, programi za prepoznavanje grešaka, itd. Faza programiranja predstavlja vremenski najdulju fazu u životnom ciklusu razvoja softvera.

Sljedeća, ali ipak podjednako važna faza o kojoj će biti više riječi u sljedećim poglavljima je testiranje. Ova faza započinje kada je dovršen odnosno implementiran kompletan softver ili dijelovi softvera, ovisno o metodi koja se koristi prilikom razvoja softvera. Tijekom testiranja nastoje se utvrditi greške i nedostaci u radu softvera koji se zatim prijavljuju programerima na ispravak. Ovaj postupak ponavlja se tako dugo dok nije istestiran cijeli sustav i dok nisu uklonjene sve greške. Faza testiranja završava tek kada se testovima prihvaćanja potvrdi kako je sustav stabilan i radi u skladu s poslovnim potrebama i specifikacijama. Nakon potvrde da

je sustav stabilan i spreman za rad slijedi integracija sustava u proizvodno okruženje te njegovo svakodnevno korištenje.

Posljednja faza je faza održavanja. Ona služi rješavanju različitih problema koji se pojavljuju prilikom korištenja sustava. Osim toga može se reći kako ova faza također pruža mogućnost poboljšanja funkcionalnosti ili nadogradnje sustava prema različitim potrebama naručitelja sustava. Faze životnog ciklusa razvoja softvera prikazane su na slici 5.

Slika 5. Faze životnog ciklusa razvoja softvera



Izvor: DATAROB. (2019). *Software Development Life Cycle (SDLC) – Basics, Stages, Models: Stages* [online], Estonia, Datarob. Dostupno na: https://datarob.com/essentials-software-development-life-cycle/ [08.veljače 2021.]

Iako s fazom održavanja završava životni ciklus razvoja softvera kroz različitu literaturu može se primijetiti kako različiti autori gore spomenutim fazama dodaju i još dvije faze: evaluacija i zbrinjavanje. Faza evaluacije daje pregled zadovoljstva rada cjelokupnog sustava. Dakle, njome se nastoji utvrditi zadovoljava li sustav sve definirane zahtjeve i ciljeve te koje su to slabosti sustava. Za razliku od evaluacije, faza zbrinjavanja predstavlja sam kraj ciklusa, odnosno onaj trenutak kada sustav prestaje biti koristan ili potreban. U ovoj se fazi detaljno razrađuju planovi kako na ispravan način upravljati informacijama iz sustava, kako ih pohraniti i na ispravan način odbaciti, definirati metode njihovog arhiviranja i na kraju definirati sam plan prijelaza na novi sustav.

Iako živimo u dobu velikih tehnoloških napredaka i otkrića istraživanja su pokazala kako se još uvijek dio projekata ne izgrađuje prema agilnim ili standardnim metodama razvoja softvera. Prema nedavnom istraživanju koje je provodio Organize Agile 2020. godine među

kompanijama iz 19 različitih zemalja, svega gotovo polovica kompanija koristi agilne metode razvoja softvera već tri godine ili duže. Ove kompanije uglavnom su koristile agilne metode razvoja softvera za svoje programe promjena poznatije kao agilne transformacije. Te promjene zapravo obuhvaćaju okretan rad u malim timovima koji se usredotočuju na brzo i iterativno postizanje rezultata. Kao najpopularnija agilna metoda naveden je Scrum koji je zastupljen čak 80 % među ispitanicima na kojima je provedeno istraživanje.[16]

Brojne su prednosti dobro definiranog životnog ciklusa razvoja softvera kao što su uvid u cjelokupni projekt, precizno definirani ciljevi za svaku pojedinu fazu ciklusa, opsežna dokumentacija, iterativan pristup. Kao nedostaci životnog ciklusa razvoja softvera može se istaknuti nefleksibilnost pojedinih metoda, neprepoznavanje promjena koje je potrebno implementirati, nemogućnost definiranja svih detalja u početnim fazama razvoja.

Životni ciklus razvoja softvera predstavlja polaznu točku za razvoj softvera ali je još uvijek pojedinu fazu potrebno prilagoditi potrebama organizacije.

[16] Organize Agile. (2020). Half of companies applying Agile metodologies & practicws [online], Europe, Consultancy.eu. Dostupno na https://www.consultancy.eu/news/4153/half-of-companies-applying-agile-methodologies-practices [08.veljače 2021.]

3. TESTIRANJE SOFTVERA I OSIGURANJE KVALITETE

Testiranje softvera postaje sve važniji segment u životnom ciklusu razvoja softvera. Razlog tomu je razvoj sve kompleksnijeg i složenijeg softvera čiji korisnici na današnjem tržištu zahtijevaju da taj softver bude besprijekoran, što bi značilo bez grešaka koje mogu utjecati na rad i njegovu kvalitetu.

Kada se vratimo u povijest možemo zaključiti kako se testiranje softvera nije razvilo u jednom danu, trebalo je puno vremena, znanja i istraživačkog rada da bi se stiglo na današnju poziciju. Posljednja su četiri desetljeća podijeljena na nekoliko razdoblja. Razdoblja su dobila naziv utjecajem najdominantnijeg modela testiranja. Prema Davidu Gelperinu i Billu Hetzelu povijest testiranja može se podijeliti u pet značajnijih razdoblja koja će biti objašnjena u nastavku. [17]

Prvo razdoblje, aktivno tijekom ranih 1950-ih, poznato je i kao razdoblje orijentirano na otklanjanje pogrešaka. Ovo razdoblje uglavnom se fokusiralo na testiranje hardvera, a sam pojam otklanjanja grešaka i testiranja softvera nisu bili niti jasno definirani niti diferencirani. Prvi članak o provjeri softvera napisao je 1949. godine Alan Turing gdje ukazuje na potrebu dokazivanja valjanosti programa.

Demonstracijski orijentirano razdoblje proteže se od 1957. pa do 1978. godine. U ovom razdoblju napravljena je razlika između otklanjanja pogrešaka i testiranja s time da je testiranje izdvojeno kao zasebna aktivnost. Glavni cilj testiranja bio je osigurati zadovoljavanje softverskih zahtjeva i specifikacija.

Razdoblje orijentirano na uništavanje traje od 1979. pa do 1982. godine, a njegov fokus bio je na rastavljanju koda na manje cjeline te pronalaženje grešaka u tim cjelinama. Zanimljivo je da je u ovom razdoblju Meyers, u knjizi The Art of Software Testing, prvi definirao testiranje kao proces izvršavanja programa s namjerom pronalaženja grešaka.

Sljedeće razdoblje nazvano još i razdoblje orijentirano na evaluaciju traje od 1983. do 1987. godine. Fokus ovog razdoblja bio je na definiranju smjernica za procjenu i mjerenje kvalitete

[17]Gelperin, D., Hetzel, B. (1988) The Growth of Software Testing. *Communications of ACM* [online], 31(6). Dostupno na: http://understandingrequirements.com/resources/2.2%20%20Growth%20of%20SW%20Testing.pdf [08. veljače 2021.]

softvera. Testiranje se u ovom razdoblju provodilo do one točke gdje bi se broj grešaka sveo na minimum, a softver postao upotrebljiv.

Posljednje razdoblje tzv. preventivno orijentirano započinje od 1988. godine i traje sve do 2000. godine. Ovo je razdoblje koje doživljava istraživački procvat, definiranje tehnike testiranja postaje ključni segment procesa testiranja, a cilj je što bolje razumjeti softver kako bi se pronašlo čim više grešaka.

Poslije 2000. godine testiranje doživljava pravu evoluciju pojavom alata za automatizaciju. A sada svjedočimo dobu gdje umjetna inteligencija polako zauzima svoje mjesto u testiranju.

3.1. Definicija, ciljevi i načela testiranja

Proces testiranja se koristi kako bi se izbjegle pogreške, ali u svrhu provjere radi li softver u skladu s definiranim zahtjevima. Testiranje softvera je postupak procjene funkcionalnosti softvera s namjerom da se utvrdi udovoljava li razvijeni softver navedenim specifikacijama te da se utvrde eventualni nedostaci kako bi se osiguralo da nema grešaka koje bi utjecale na rad sustava. Jedna od najstarijih definicija testiranja jest da je testiranje proces izvođenja programa s namjerom pronalaska grešaka.[18] Još jedna definicija testiranja kaže kako je testiranje postupak evaluacije sustava ili pojedine komponente sustava ručnim ili automatiziranim alatima s ciljem utvrđivanja zadovoljava li taj sustav određene zahtjeve.[19]

Gotovo u svakoj definiciji testiranja spominje se pojam pogreške. Pogreška je univerzalan pojam ali problemi u radu sustava mogu se svesti na nekoliko preciznijih pojmova objašnjenih u nastavku:[20]

- greška (engl. error)
- kvar (engl. defect/fault/bug)
- vanjska pogreška (engl. failure).

[18] Myers, G.J. (2004) The Art of Software Testing. 2nd. ed. Hoboken, New Jersey: John Wiley & Sons, Inc.
[19] Everett, G. D., McLeod, R. (2007) *Software Testing:Testing Acros The Entire Software Development Lyfe Cycle*. IEEE
[20] Spillner, A., Linz, T., Schafer H. (2014) *Software testing foundations*. Santa Barbara: Rocky Nook Inc.

Greška je pojam koji se vezuje uz ljudsku pogrešku. Ne pojavljuje se samo zbog pogreške u kodu već može nastati tijekom različitih faza razvoja softvera (greška u poslovnoj analizi, nedovoljno precizne informacije od strane naručitelja, greške nastale zbog vremenskog pritiska).

Kvar nastaje kada softver ne izvršava tražene funkcije i daje rezultate koji nisu u skladu s očekivanim rezultatima. Razlog vanjskih kvarova može biti pogrešan način korištenja sustava, hardver koji nije prilagođen određenom softveru.

Vanjska pogreška (engl. failure) nastaje kao posljedica kvara (engl. defect). Predstavlja uočljivo neispravno i netočno ponašanje sustava odnosno predstavlja nepravilnosti u radu koje nisu u skladu sa specifikacijama.

Definiranjem gornjih pojmova lako je zaključiti kako su međusobno ovisni i povezani. Najčešće pojava jednog problema vodi ka pojavi drugog što u konačnici rezultira odstupanjima od očekivanih rezultata. U softverskoj industriji još uvijek postoje nesuglasice oko gore navedenih pojmova i upotreba istog izraza ponekad može dovesti do različitih tumačenja.

Kao najvažniji ciljevi u testiranju softvera ističu se prevencija, otkrivanje grešaka, zadovoljstvo krajnjeg korisnika i osiguranje kvalitete softvera.

Prevencijom se nastoje predvidjeti moguće greške te se tako sprječava njihovo pojavljivanje u budućnosti. Prevencijom se značajno mogu smanjiti troškovi održavanja kvalitete softvera. Postupkom pronalaženja i otklanjanja grešaka sprječava se pojavljivanje grešaka kod krajnjeg korisnika kada je softver već u upotrebi. A kada je softver već u upotrebi tada on mora ispunjavati određene korisničke zahtjeve. Sve to utječe na zadovoljstvo samog korisnika. Kvaliteta softvera zapravo podrazumijeva držanje softverskih grešaka na najnižoj mogućoj razini, a da je taj softver još uvijek pouzdan i validan.

U nastavku je definirano je 7 načela testiranja koja se koriste kao opća smjernica za sve vrste testiranja:[21]

1. Testiranjem se prikazuje prisutnost grešaka, ne njihova odsutnost. To bi značilo da se testiranjem smanjuje vjerojatnost pojave grešaka, ali to ne znači da je sustav 100% pouzdan ako testiranjem nisu pronađene greške.

[21] Spillner, A., Linz, T., Schafer H. (2014) *Software testing foundations*. Santa Barbara: Rocky Nook Inc.

2. Cjelovito (ili iscrpno) testiranje nije moguće. Nije realno očekivati da se testiranjem mogu pokriti sve moguće kombinacije svih scenarija. Stoga je neophodno planirati testiranje i tako dodijeliti prioritet pojedinim scenarijima.
3. Rano testiranje štedi vrijeme i novac. Aktivnosti testiranja trebale bi započeti što je moguće ranije u životnom ciklusu razvoja softvera. Troškovi ispravljanja pogrešaka u kasnijim fazama razvoja softvera mogu povećati troškove ali i uštedjeti vrijeme (npr. gubitak vremena na analizu greške, pronalaženje uzroka, ispravljanje greške, regresijski testovi, itd).
4. Greške se grupiraju zajedno. Ovdje vrijedi pravilo 80-20 poznato kao Pareto princip koji tvrdi da 80% rezultata proizlazi iz 20% svih uzoraka. To znači da testiranjem treba identificirati osjetljiva područja, a potom usmjeriti test na takva područja.
5. Korištenjem istih testova smanjuje se učinkovitost. Kako se softver razvija tako je potrebno prilagoditi i metode testiranja čime se povećava vjerojatnost pronalaska grešaka.
6. Testiranje ovisi o kontekstu. Različiti softveri imaju različite zahtjeve i svrhe te se prema tome različit softver ne treba biti testiran istim metodama i tehnikama.
7. Odsutnost pogrešaka je zabluda. Ovo načelo usko je povezano s pravilom kako cjelovito testiranje nije moguće. I nakon što je proces testiranja završen gotovo uvijek postoji određena vjerojatnost da postoje neotkrivene pogreške.

Navedeni ciljevi i načela kroz povijest su se pokazali kao točni i primjenjivi. Stoga je važno istaknuti kako se njihovim poštivanjem značajno može povećati kako učinkovitost testiranja tako i kvaliteta samog softvera.

Na kraju ovog poglavlja može se zaključiti kako je testiranje softvera važan dio razvoja softvera iz nekoliko razloga:

- ako se testiranje ne provede softver može sadržavati pogreške koje mogu narušiti rad sustava, a u najgorem slučaju mogu dovesti do prekida rada sustava
- ne provođenje testova može dovesti do velikih financijskih gubitaka
- ne provođenje testova može rezultirati pravnim tužbama i kaznenim progonima
- ne provođenje testiranja može utjecati na reputaciju organizacije što u konačnici može rezultirati gubitkom prihoda.

U prilog gore navedenim činjenicama govore i rezultati istraživanja koje je 2002. godine proveo NIST (National Institute of Standards and Technology). Istraživanjem je utvrđeno kako programske pogreške američko gospodarstvo godišnje koštaju 59,5 milijardi američkih dolara. Poboljšanjem infrastrukture moguće je smanjiti troškove za čak 22,2 milijarde američkih dolara.[22] Rezultati istraživanja prikazani su na slici ispod.

Slika 6. Troškovi američkog gospodarstva nastali zbog neadekvatnog testiranja

	The Cost of Inadequate Software Testing Infrastructure (billions)	Potential Cost Reduction from Feasible Infrastructure Improvements (billions)
Software developers	$21.2	$10.6
Software users	$38.3	$11.7
Total	$59.5	$22.2

Izvor: NIST (2002) *The Economic Impacts of Inadequate Infrastructure for Software Testing* [online], Gaithersburg: National Institute of Standards and Technology. Dostupno na: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf [09.veljače 2021.]

3.2. Životni ciklus testiranja softvera

Tijekom posljednjih nekoliko desetljeća testiranje je znatno evoluiralo i to toliko da se ne sastoji samo od prijavljivanja grešaka koje su pronađene tijekom testa već testiranje ima svoj vlastiti životni ciklus. Životni ciklus ustvari predstavlja slijed promjena kroz koje neki entitet prolazi.

U drugom poglavlju ovog rada opisan je životni ciklusa razvoja softvera. Testiranje predstavlja samo jednu fazu u životnom ciklusu razvoja softvera, s opet ima svoj vlastiti životni ciklus. U tablici 1. su prikazane osnovne razlike između životnog ciklusa razvoja softvera i životnog ciklusa testiranja softvera.

[22] NIST (2002) *The Economic Impacts of Inadequate Infrastructure for Software Testing* [online]. Gaithersburg: National Institute of Standards and Technology. Dostupno na: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf [09.veljače 2021.]

Tablica 1. Osnovne razlika između SDLC i STLC (autorski rad)

	Životni ciklus razvoja softvera	Životni ciklus testiranja softvera
1.	usredotočuje se na izgradnju cijelog softvera	usredotočuje se na testiranje softvera
2.	krajnji cilj je razviti visokokvalitetan softver	krajnji cilj je pronaći greške
3.	temelji se na projektnoj dokumentaciji	temelji se na testnim scenarijima
4.	nadređeni proces	dio je životnog ciklusa razvoja softvera

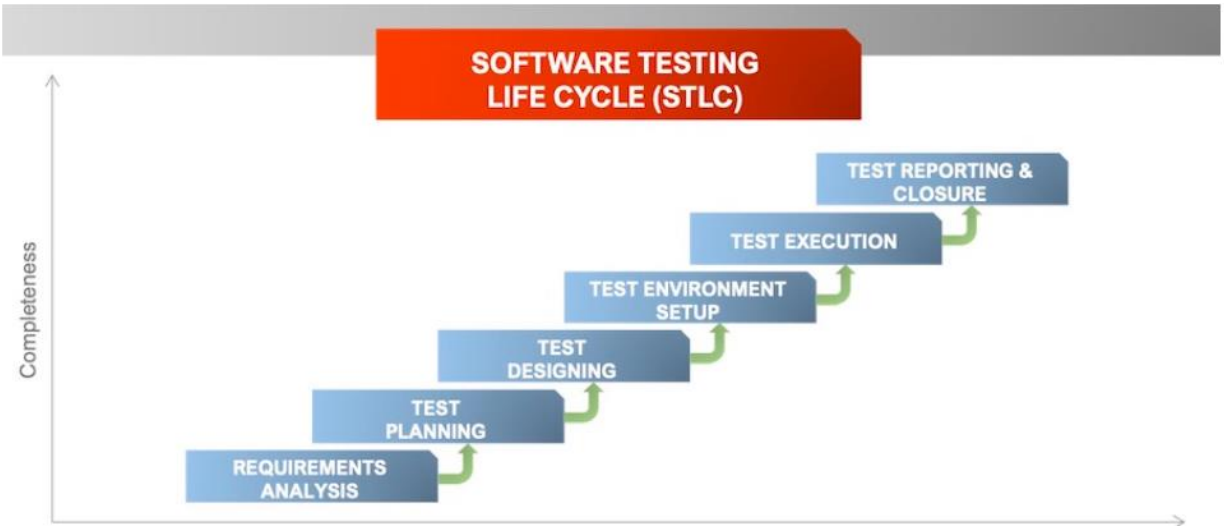
Životni ciklus testiranja softvera (engl. Software Testing Life Cycle) predstavlja slijed određenih aktivnosti koje se provode tijekom procesa testiranja kako bi se osiguralo da su ispunjeni ciljevi kvalitete softvera. Suprotno uvriježenom mišljenju testiranje nije samo pojedinačna aktivnost već se sastoji od niza aktivnosti koje se provode planirano i metodološki.[23] Iako se ne može reći kako postoji jedan univerzalni životni ciklus testiranja softvera, ali se može kako zaključiti postoje uobičajeni nizovi aktivnosti kojima se omogućuje da se postignu definirani ciljevi.

U nastavku su prikazane i opisane pojedine faze:[24]

1. analiza zahtjeva
2. planiranje testiranja
3. definiranje testnih scenarija
4. pripremanje testnih okolina
5. provođenje testova
6. izvještavanje i zatvaranje ciklusa/testiranja

[23] Guru99 (2021). *What is software testing life cycle (STLC)?* [online]. Guru99. Dostupno na: https://www.guru99.com/software-testing-life-cycle.html [10. veljače 2021.]
[24] Ibid

Slika 7. Aktivnosti u životnom ciklusu testiranja softvera



Izvor: Software Testing Fundamentals. (2020). *Software Testing Life Cycle (STLC)* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/software-testing-life-cycle/ [10.veljače 2021.]

Svaka spomenuta faza ima određene kriterije ulaska i izlaska. Kriteriji ulaska i izlaska predstavljaju određene preduvjete. Tako kriteriji za ulazak u pojedinu fazu predstavljaju preduvjete koji se moraju ispuniti prije nego se započne s nekom fazom. Dok kriteriji za izlaz predstavljaju određene rezultate koji se moraju ispuniti prije nego se može prijeći na sljedeću fazu.

3.2.1. Faza analize zahtjeva

Na samom početku životnog ciklusa testiranja nalazi se faza analize zahtjeva. U ovoj fazi testni tim raščlanjuje zahtjeve i definira što je potrebno testirati. Tijekom ove faze testni tim također proučava zahtjeve tako da im se dodijeli prioritet. Prilikom analiziranja zahtjeva članovi testnog tima komuniciraju s različitim sudionicima kao što su naručitelji softvera, poslovni analitičari, arhitekti sustava, programeri, itd. Postoji nekoliko vrsta zahtjeva:[25]

- poslovni zahtjevi visoke razine preuzeti iz poslovne dokumentacije projekta
- arhitektonski i dizajnerski zahtjevi su detaljniji od poslovnih zahtjeva i sadrže cjelokupni dizajn potreban za provedbu poslovnog zahtjeva

[25] Guru99 (2021). *What is software testing life cycle (STLC)?* [online]. Guru99. Dostupno na: https://www.guru99.com/software-testing-life-cycle.html [10. veljače 2021.]

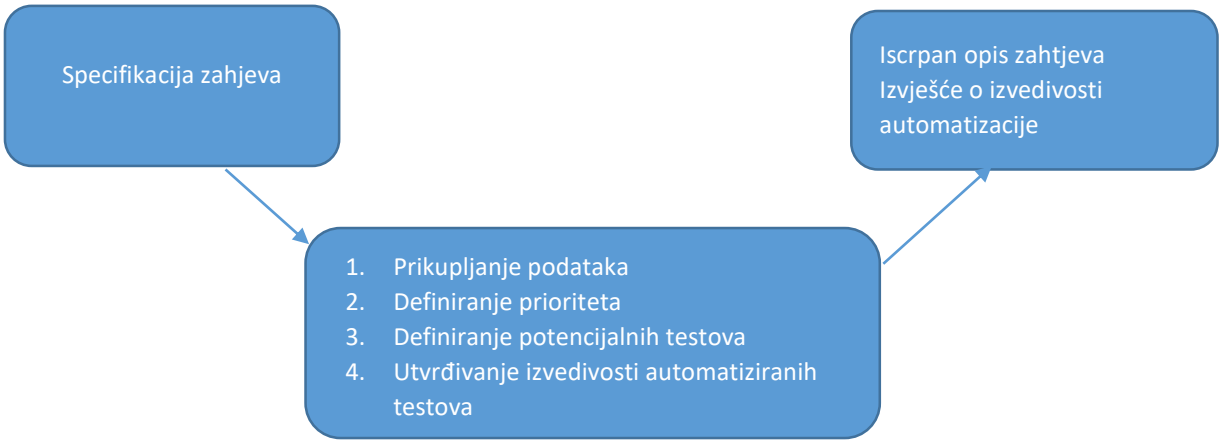
- sistemski i integracijski zahtjevi sadrže detaljan opis u obliku korisničkih priča, a obiluju detaljima koji su preduvjet za programiranje i testiranje.

Kao kriterij ulaska u ovu fazu potrebna je specifikacija zahtjeva. Aktivnosti koje se zatim provode su:

- prikupljanje podataka i informacija potrebnih za testiranje (od poslovne analize, arhitekata, naručitelja, programera)
- definiranje prioriteta testiranja
- analiza i definiranje testova koji se mogu izvoditi
- utvrđivanje izvedivosti automatiziranih testova.

Kao rezultat navedenih aktivnosti je dokument koji prati zahtjeve naručitelja koji sadrži iscrpan opis zahtjeva, te izvješće o izvedivosti automatizacije.

Slika 8. Prikaz aktivnosti unutar faze analize zahtjeva (autorski rad)



3.2.2. Faza planiranja testiranja

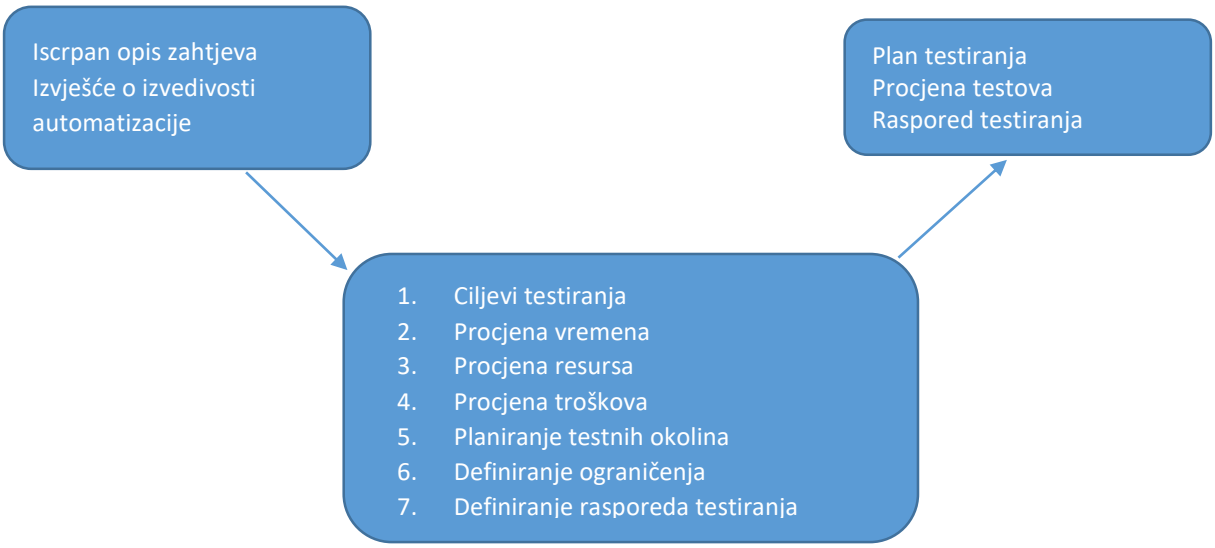
Kao ulazni kriterij ove faze uzimaju se dokumenti koji su dobiveni kao rezultat faze analize zahtjeva. Planiranjem testiranja određuje se strategija testiranja. Strategija testiranja podrazumijeva detalje kao što su:

- definiranje ciljeva testiranja
- procjena vremena koje se planira utrošiti
- procjena broja ljudi koji će biti uključeni u test

- procjena troškova testiranja
- planiranje testnih okolina
- definiranje ograničenja u testiranju
- definiranje rasporeda testiranja.

Rezultat svih prethodnih aktivnosti su dokumenti koji su važni za daljnji nastavak testiranja: plan testiranja, procjena testiranja i raspored testiranja.

Slika 9. Prikaz aktivnosti unutar faze planiranja testiranja (autorski rad)



3.2.3. Definiranje testnih scenarija

Nakon što je završena faza planiranja slijedi faza pisanja testnih scenarija. U ovoj fazi testni tim raspisuje testne skripte. Testne skripte sastoje se od testnih koraka koji predstavljaju najmanju jedinicu testiranja. Osim što u ovoj fazi testni tim raspisuje testne skripte, oni također pripremaju i testne uzorke koji su jedan od preduvjeta testiranja. Testni koraci opisuju kako točno treba provesti test.

Iako organizacije imaju različite pristupe u pisanju i dokumentiranju testnih scenarija ono što je zajedničko za sve jest da testni scenariji trebaju biti što detaljniji. Tako se olakšava određivanje kriterija uspješnosti provedenog testnog koraka. Osim toga, kada je testni scenarij detaljno raspisan to olakšava njegovo ponavljanje pogotovo ako se nakon ponovljenog testa planira uspoređivanje starijih i novijih rezultata testiranja. U nekim

slučajevima detaljno raspisan scenarij može poslužiti kao predložak za lakšu automatizaciju pojedinog scenarija.

3.2.4. Pripremanje testnih okolina za test

U ovoj fazi određuju se softverski i hardverski preduvjeti koji su potrebni za provođenje testova. Rezultati ove faze uvelike ovise o suradnji s drugim članovima projektnog tima. Podešavanje testnih okolina najčešće odrađuju timovi zaduženi za definiranje arhitekture softvera, timovi za razvoj, dizajneri, timovi za održavanje baze podataka, timovi zaduženi za mrežnu infrastrukturu, itd.

Dakle, gore navedeni timovi zaduženi su za:

- podešavanje testnih servera
- podešavanje mrežnih postavki
- podešavanje računala za testiranje (instalacija programa potrebnih za provođenje testa, alata za prijavu grešaka)
- podešavanje pristupa bazama podataka.

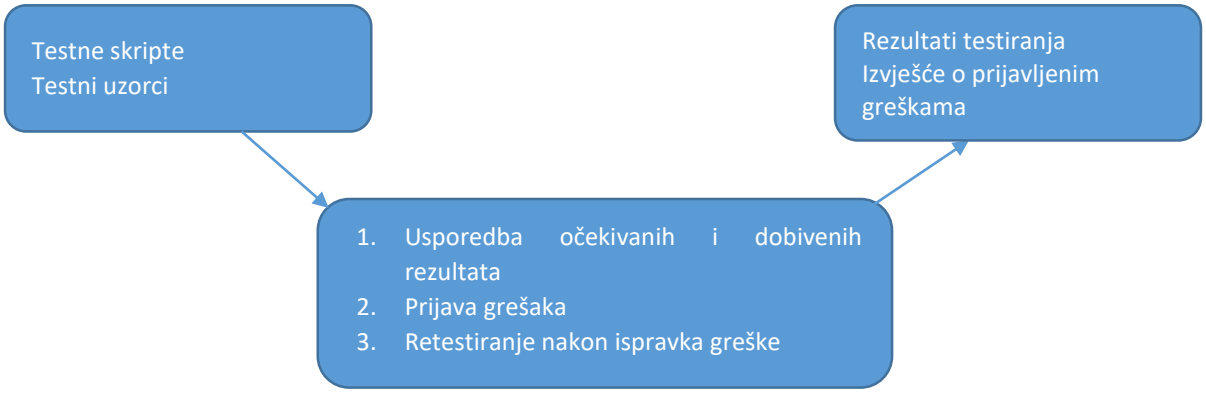
Nakon što je testna okolina podešena i spremna za testiranje, testni tim izvršava provjeru spremnosti okoline za rad.

3.2.5. Provođenje testova

Nakon što su završene sve prethodne faze slijedi faza testiranja. Testiranje predstavlja postupak izvršenja testnih koraka prema testnoj skripti i usporedbom očekivanih i dobivenih rezultata. Ako je dobiveni rezultat nekog testnog koraka jednak očekivanom testni korak se smatra uspješnim te se prelazi se na sljedeći.

Ako je dobiveni rezultat različit od očekivanog tada tester prijavljuje grešku te od programera traži njen ispravak. Nakon ispravka greške testeru preostaje još jednom provesti sve testne korake prema scenariju kako bi se utvrdilo da ispravak greške nije utjecao na rad druge funkcionalnosti.

Slika 10. Prikaz aktivnosti unutar faze testiranja (autorski rad)



3.2.6. Izvještavanje i zatvaranje ciklusa/testiranja

Ovo je posljednja faza u životnom ciklusu testiranja softvera. Ova faza uključuje sastajanje svih članova testnog tima te ocjenu kriterija završetka ciklusa na temelju rezultata dobivenih testom, pokrivenosti testa, ispunjavanje definiranih ciljeva.

Proces zatvaranja testa provodi se kroz nekoliko faza:

- definiranje verzije isporuke koja će se dati naručitelju
- provjera ima li prijavljenih kritičnih grešaka koje mogu dovesti do prekida rada sustava
- primopredaja sustava
- arhiviranje testne dokumentacije
- prikupljanje iskustava i ideja koje se mogu iskoristiti na drugim projektima.

Rezultat ove faze su dokumenti kao: testovi prihvaćanja, ispitne metrike, izvještaj o zatvaranju testa.

3.3. Osiguranje kvalitete softvera

Prije definiranja pojma osiguranje kvalitete softvera potrebno je definirati i razjasniti pojmove kvalitete softvera i kontrole kvalitete. Kako postoje različite definicije kvalitete softvera potrebno je razjasniti koja su to očekivanja od kvalitetnog softvera, a tek kasnije definirati sam pojam. Prema Jeffu Tian kvalitetan softver mora raditi ono što je predviđeno da radi

prema određenim specifikacijama, kvalitetan softver mora izvoditi zadatke ispravno i prema očekivanom. [26]

Kako je ovaj pojam dosta općenit iz vlastitog iskustva smatram kako zapravo kvalitetan softver predstavlja onaj softver koji je pogodan i ispravan za upotrebu te koji zadovoljava očekivanja naručitelja.

Osiguranje kvalitete softvera (engl. Software Quality Assurance) predstavlja skup aktivnosti za osiguranje kvalitete u procesima softverskog inženjerstva koji rezultira ili barem daje povjerenje u kvalitetu softvera.[27]

Osiguranje kvalitete softvera predstavlja skup planiranih i sistematiziranih aktivnosti čije je izvođenje neophodno kako bi se osiguralo odgovarajuće povjerenje da neki proizvod zadovoljava određene tehničke preduvjete.[28]

Osiguranje kvalitete softvera trebalo bi se provlačiti kroz sve faze procesa razvoja softvera jer je to jedini način da se osigura njegova kvaliteta. Ono je usredotočeno na poboljšanje procesa razvoja softvera kao i njegovo stvaranje prema definiranim standardima koji imaju cilj osigurati kvalitetu.

Osim osiguranja kvalitete kroz upravljanje kvalitetom često se provlači i pojam kontrola kvalitete. Iako se ova dva pojma često naizmjenično koriste ipak postoji određena razlika između njih.

Kontrola kvalitete predstavlja podskup aktivnosti u osiguranju kvalitete, a definira se kao dio upravljanja kvalitetom usmjeren na ispunjavanje zahtjeva kvalitete. Osiguranje kvalitete odnosi se na to kako se izvodi postupak osiguranja kvalitete, dok je kontrola kvalitete više usmjerena na to tko provodi i nadzire postupak upravljanja kvalitetom. To znači da se nad softverom provode određena mjerenja i ispitivanja radi ocjenjivanja jedne ili više karakteristika i jesu li te karakteristike usklađene s definiranim zahtjevima.[29]

---

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

[27] Software Testing Fundamentals (2020). *SoftwareQuality Assurance* [online]. Dostupno na: https://softwaretestingfundamentals.com/software-quality-assurance/ [22. veljače 2021.]

[28] Laporte, C.Y., April, A. (2018) Software Quality Assurance. USA: John Wiley & Sons.

[29] American Society for Quality (2021). *Quality Assurance & Quality Control* [online]. USA: ASQ. Dostupno na: https://asq.org/customer-services/contact-asq [18. veljače 2021.]

Slika 11. Prikaz odnosa između osiguranja kvalitete, kontrole kvalitete i testiranja

Izvor: Altexsoft. (2021). *Quality Assurance, Quality Control and Testing — the Basics of Software Quality Management* [online], Altexsoft. Dostupno na: https://www.altexsoft.com/whitepapers/quality-assurance-quality-control-and-testing-the-basics-of-software-quality-management/ [22.veljače 2021.]

Kroz proces upravljanja kvalitetom provlači se i testiranje. Testiranje softvera predstavlja centralnu ulogu u osiguranju kvalitete softvera. Pokretanjem softvera ili izvođenjem njegovih pojedinih funkcija testeri mogu utvrditi odgovara li uočeno ponašanje softvera definiranim specifikacijama. Usmjereno je na otkrivanje i rješavanje pogrešaka u kodu ali je usmjereno i na procjenu ukupne upotrebljivosti i sigurnosti softvera.

Iako u ovom kontekstu testiranje predstavlja centralnu aktivnost u osiguranju kvalitete softvera, ono ipak čini samo njegov podskup. Ova dva pojma mogu se usporediti prema određenim karakteristikama kao što su definicija, fokus, orijentacija, tip aktivnosti i cilj. U sljedećoj tablici će biti prikazane ključne razlike između testiranja softvera i osiguranja kvalitete softvera.

Tablica 2. Prikaz razlika između SQA i testiranja softvera

	SQA	Testiranje softvera
definicija	inženjerski postupak koji osigurava kvalitetu	otkrivanje grešaka prije nego softver postane aktivan
fokus	uključuje aktivnosti povezane s provedbom procesa, postupaka i standarda	uključuje aktivnosti povezane s pronalaženjem grešaka
orijentacija	procesno je orijentiran	orijentiran je na proizvod
tip aktivnosti	predstavlja preventivnu metodu	predstavlja korektivnu metodu
cilj	osigurati kvalitetu	kontrolirati kvalitetu

Izvor: Vasylyna, N. (2021). Difference Between QA and Testing [online]. QATestLab Blog. Dostupno na: https://blog.qatestlab.com/2011/04/07/what-is-the-difference-between-qa-and-testing/ [23. veljače 2021.]

Dakle, osiguranje kvalitete softvera, kontrola kvalitete i testiranje imaju hijerarhijsku prirodu. Ti su pojmovi međusobno povezani, ali nisu zamjenjivi. Iako se fokusiraju na različite stvari, metode i tehnike ipak teže zajedničkom cilju. Njihova međusobna povezanost trebala bi jamčiti određenu razinu kvalitete i isporuku pouzdanog softvera.

3.3.1. Tehnike osiguranja kvalitete softvera

Iako postoje različite tehnike u osiguranju kvalitete softvera, revizija je jedna od najprihvaćenijih. Ostale značajnije tehnike su navedene su i ukratko upisane u nastavku teksta:[30]

- pregledavanje
- funkcionalna testiranja
- standardizacija
- provjeravanje koda
- stress testovi
- statička analiza koda
- six sigma.

[30] Educba (2020). Software Quality Assurance [online]. Dostupno na: https://www.educba.com/software-quality-assurance/ [28. veljače 2021.]

Tehnika pregledavanja podrazumijeva sazivanje i održavanje sastanaka na kojima sudionici pregledavaju projekt razvoja softvera s ciljem da se utvrdi zadovoljava li softver definirane zahtjeve.

Revizija je proces procjene kvalitete informacijskih sustava kroz postupke analize njihova učinka na poslovanje i provjere točnosti, učinkovitosti, djelotvornosti i pouzdanosti.[31] Ona zapravo podrazumijeva provjeru cijelog softvera i svih podataka kako bi se utvrdilo slijedi li se standardni postupak.

Funkcionalni testovi fokusiraju se na testiranje značajki sustava. Značajke sustava definirane su specifikacijama. Prema tome, ovim testovima nastoji se utvrditi radi li sustav u skladu s definiranim specifikacijama.

Standardizacijom se osigurava da razvijeni softver bude u skladu sa zahtjevima standarda te tako smanji nedvosmislenost, ali i osigura kvalitetu.

Tehnika provjere koda je također jedna od korištenijih tehnika u osiguranju kvalitete. Najčešće ju provode programeri, obzirom da su oni ti koji pišu kod, a njezin cilj je pronalaženje grešaka koje bi na neki način mogle utjecati na prekid izvođenja softvera.

Stress testovima provjerava se kako sustav radi pod velikim opterećenjem odnosno kako sustav radi izvan normalnih uvjeta. Ovo je jedna od tehnika koja ima važnu ulogu u procjeni kvalitete softvera.

Statička analiza koda predstavlja proces provjere koda ali bez njegova izvršavanja. Ova tehnika može biti integrirana unutar različitih okruženja koda (razvojno, testno, produkcijsko). Postoje specijalizirani alati koji stalno prate kvalitetu programa, a upravno takvim načinom je moguće na vrijeme poduzeti mjere za poboljšanja kvalitete koda.

Six Sigma predstavlja tehniku osiguranja kvalitete koja teži savršenstvu. Temelji se na podacima i metodologijama čiji je zadatak uklanjanje nedostataka odnosno odstupanja u bilo kojoj fazi razvoja softvera. Široko se primjenjuje u različitim područjima, uključujući i razvoj softvera. Glavni cilj ove tehnike je poboljšanje svih procesa tako da konačna verzija softvera ima vrlo mali postotak nedostataka.

[31] Spemić, M. (2017) *Sigurnost i revizija informacijskih sustava u okruženju digitalne ekonomije*. Zagreb: Sveučilište u Zagrebu, Ekonomski fakultet.

Kroz proces razvoja platnog sustava korištene su različite tehnike osiguranja kvalitete. Među njima se svakako ističu funkcionalni testovi, standardizacija, stress testovi ali i statička analiza koda. Sve ove tehnike su provođene kroz različite faze razvoja softvera. Na primjer, statička analiza koda je tehnika koja se provodila na kraju svakog sprinta.Temeljem rezultata statičke analize provodile su se manje ili veće korekcije koda. Sve ove tehnike su na poseban način doprinijele tome da platni sustav u konačnici zaživi te postane sustav siguran za upotrebu.

3.3.2. Standardi osiguranja kvalitete softvera

U svrhu osiguranja kvalitete napisani su mnogi standardi. Tijekom razvoja softvera gotovo svaka organizacija suočava se s potrebom usklađivanja istog sa SQA standardima. Priprema bilo kojeg standarda dug je i ponavljajući postupak koji uključuje kompromis i sažimanje detalja. U mnogim slučajevima standardi opisuju tehnike i metode kojima se osigurava osnovna razina kvalitete.[32] Postoji nekoliko različitih instituta i profesionalno orijentiranih organizacija koje su uključene u razvoj standarda za osiguranje kvalitete softvera. Organizacije koje su zadužene za razvoj SQA standarda su:[33]

- IEEE Computer Society (Institute of Electrical and Electronics Engineers)
- ISO (International Organization for Standardization)
- DOD (US Department of Defense)
- ANSI (American National Standards Institute)
- IEC (International Electro Technical Commission)
- EIA (Electronic Industries Association)

Dakle, navedene organizacije razvijaju međunarodne standarde kvalitete, ali također izdaju i SQA certifikate koji se dodjeljuju nakon revizije kvalitete u organizaciji. Među navedenim institucijama ISO 9001 je trenutno najistaknutiji pružatelj SQA certifikata u Europi.

Prema konzultantskoj tvrtki Compliencehelp koja se specijalizirala za nekoliko različitih ISO standarda postoji nekoliko ključnih načela na kojima se temelji sustav osiguranja kvalitete ISO

[32] Rubey, J.R., Brewer, A. (1991). *Software quality assurance standards-a comparison and an integration* [online]. San Diego: IEEE Computer Society. Dostupno na: https://www.computer.org/csdl/proceedings-article/sesaw/1991/00151534/12OmNyv7mvm [23. veljače 2021.]

[33] Tutorialspoint (2021). Standards and Certificates [online]. Dostupno na: https://www.tutorialspoint.com/software_quality_management/software_quality_management_standards_and_certificates.htm [28. veljače 2021.]

9001, a to su usredotočenost na kupca, procesni pristup, sudjelovanje ljudi, stalno poboljšanje, donošenje odluka na činjenicama ili dokazima, upravljanje odnosima.[34]

Unutar platnog sustava koji je predmet istraživanja ovog specijalističkog rada, implementira se standard ISO 27001. Ovaj standard predstavlja međunarodni standard koji se orijentira na područje informacijske sigurnosti. Njime je definirana metodologija za primjenu upravljanja informacijskom sigurnosti unutar organizacije. Osim toga omogućuje organizacijama dobivanje certifikata koji je potvrda kako je organizacija implementirala informacijsku sigurnost sukladno standardu. Standard ISO 27001 u instant platnom sustavu usredotočen je na zaštitu povjerljivosti, cjelovitosti i raspoloživosti podataka.

Iako standard ISO 27001 nije direktno vezan uz osiguranje kvalitete i na prvi pogled se čini kao da nema mnogo zajedničkog sa standardom ISO 9001 ipak dobar dio zahtjeva u oba standarda su isti. Prema tome samom implementacijom standarda ISO 27001 na neki način se ipak osigurava kvaliteta platnog sustava. Implementacijom se jamči sprečavanje sigurnosnih incidenata i zaštita podataka što sudionicima platnog sustava garantira kako osjetljivi osobni podaci neće biti zlorabljeni. Osim gore spomenutih standarda platni sustav je usklađen i sa sljedećim standardima:

- sustav pohranjuje sve datume u skladu sa standardom ISO 8601
- sustav podržava rad sa skupom znakova UTF-8
- sustav podržava razmjenu poruka u XML formatu
- sustav podržava sigurnosne i kripto standarde (SSL enkripcija komunikacije, CAdES format digitalnog potpisa ...).

[34] Compliencehelp Consulting (2020). What are Management System Standards? [online]. Dostupno na: Dostupno na: https://quality-assurance.com.au/what-are-management-system-standards/ [28. veljače 2021.]

4. METODOLOGIJE TESTIRANJA I NAČINI PROVOĐENJA TESTOVA

U današnjem tehnološkom okruženju, softver je postao ključan dio mnogih uređaja i sustava koji su postali svakodnevica u našem društvu. Ponekad se softver uzima zdravo za gotovo i podrazumijeva kako će on raditi ispravno i ono za što je namijenjen, međutim iza takvih očekivanja stoji cijeli tim koji traži najbolje moguće metode i pristupe testiranju kako bi taj softver koji dođe do krajnjeg korisnika ispunio sve zahtjeve.

Metodologije se mogu smatrati skupom mehanizama za testiranje koje se koriste u životnom ciklusu razvoja softvera, od unit testova do sistemskih testiranja.[35] Postoje mnoge metode testiranja softvera ali izbor odgovarajuće metode i dalje ponekad predstavlja problem. Prema tome odabir odgovarajuće metodologije testiranja smatra se ključnim postupkom testiranja. Osnovni cilj brojnih metodologija u testiranju softvera je brza isporuka softvera, a da se pritom osigura što veća kvaliteta.

Povijesno gledajući, u testiranju softvera se približno koristi 40-50% ukupnih resursa i između 50 i 60% otpada na ukupne troškove u procesu razvoja softvera.[36] Prema ovim podacima može se zaključiti kako testiranje predstavlja veliki izazov u razvoju softvera, a njegovim pravilnim upravljanjem mogu se znatno optimizirati troškovi, kvaliteta kao i vrijeme testiranja.

U nastavku poglavlja biti će opisane razine, metode kao i tipovi testiranja. Da bi se osigurala ali i održala kvaliteta softvera potrebno je ispravno procijeniti koja razina, metoda ili tehnika će biti najprihvatljivija kako bi se postigli definirani ciljevi.

4.1. Razine testiranja softvera

Prema ISTQB razine testiranja softvera predstavljaju različite faze životnog ciklusa razvoja softvera u kojim se provodi testiranje. Svaka od razina testiranja ima određenu svrhu, svoje

[35] Software Testing Help (2021). Software Developmet And Testing Methodologies (With Pros And Cons) [online]. Software Testing Help. Dostupno na: https://www.softwaretestinghelp.com/software-development-testing-methodologies/ [01. ožujka 2021.]

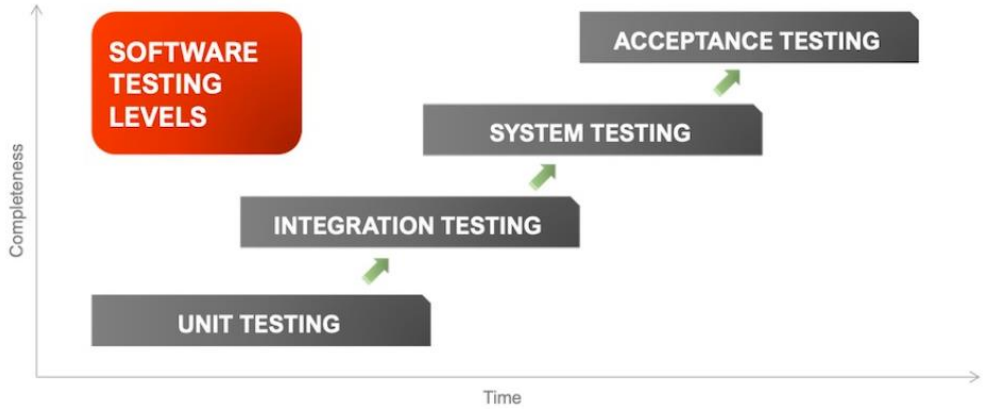
[36] Kumar, D., Mishra, K. K. (2016). *The Impact of Test Automation on Software's Cost, Quality and Time to Market. Procedia Computer Science* [online] *79*, Dostupno na: https://core.ac.uk/download/pdf/82601631.pdf [01.ožujka 2021.]

probleme i ciljeve. Svrha razina testiranja je sistematiziranje testiranja i lakše prepoznavanje testnih scenarija na pojedinoj razini.[37]

Kod testiranja softvera postoje četiri razine testiranja koje prikazani na slici ispod:

1. Jedinično testiranje
2. Integracijsko testiranje
3. Sistemsko testiranje
4. Testovi prihvaćanja.[38]

Slika 12. Kronološki redoslijed razina testiranja



Izvor: Software Testing Fundamentals. (2020). *Software Testing Levels* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/software-testing-levels/ [02. ožujka 2021.]

4.1.1. Unit testovi

Jedinično testiranje (engl. unit testing) je razina testiranja na kojoj se testiraju pojedinačne jedinice odnosno komponente sustava. Svrha ovih testova je potvrditi da svaka komponenta softvera radi onako kako je zamišljeno i prema definiranim specifikacijama.[39] Karakteristika ovih testova je izoliranje jedinice odnosno komponente. Jedinica predstavlja najmanji dio koda koji se testira, a može biti pojedinačna funkcija, metoda, postupak ili objekt.

[37] Software Testing Fundamentals. (2020). *Software Testing Levels* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/software-testing-levels/ [02. ožujka 2021.]
[38] Ammann, P., Offutt J. (2008) *Intraduction to Software Testing*. Cambridge: Cambridge University Press.
[39] Software Testing Fundamentals. (2020). *Unit Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/unit-testing/ [02. ožujka 2021.]

Provodi se tijekom razvoja aplikacije i najčešće ga provode programeri tako da izoliraju pojedinu jedinicu koda, a potom provjeravaju njenu ispravnost. Iako je uloga testera na ovaj razini testiranja minimalna, važno je da bez obzira na to poznaje proces i svrhu jediničnih testova.[40] Tijek izvođenja jediničnih testova izvodi se kroz pisanje plana, testnih slučajeva i pisanje koda za testiranje valjanosti pojedinih komponenti, a osim toga ovo testiranje se izvodi pomoću metode bijele kutije koja će biti detaljnije objašnjena u sljedećem poglavlju.

Postoje mnoge prednosti jediničnih testova, a prema Software Testing Fundamentals ovo su samo neke od njih:[41]

- kada se napišu dobri jedinični testovi olakšava se testiranje svake promjene koda
- jediničnim testovima smanjuju se troškovi testiranja jer se greške mogu otkriti u najranijoj fazi razvoja softvera
- ispravljanje pogrešaka je jednostavnije jer kada se testom otkrije greška potrebno je samo ispraviti odnosno ažurirati kod
- kvaliteta koda je bolja jer se greške otkrivaju u ranoj fazi
- dugoročno gledajući razvoj softvera je brži jer su napor i vrijeme utrošeni u jedinične testove puno manji u odnosu na napor i vrijeme utrošeno na pronalaženje grešaka tijekom sistemskih testova ili testova prihvaćanja.

Iako jedinični testovi pružaju velike koristi, postoje i određeni nedostaci:

- nije realno očekivati kako će se ovim testovima prepoznati svaka greška
- ova vrsta testiranja fokusira se na jedinicu koda stoga je teško otkriti greške na razini cijelog sustava te se iz tog razloga preporučuje jedinične testove kombinirati s drugim vrstama testiranja.

Jedinični testovi najčešće su automatizirani, iako se ne isključuje mogućnost njihova ručnog provođenja. Neki od alata koji nude više mogućnosti za provođenje ovih testova su Junit, NUnit, Jmockit, PHPUnit, itd. U posljednje vrijeme sve je više alata za jedinično testiranje, a posebno su popularni alati za programske jezike C i Java.

[40] Lučić, M. (2019) *Prakse testiranja programskih proizvoda*. Završni rad. Varaždin: Fakultet organizacije i informatike.

[41] Software Testing Fundamentals. (2020). *Unit Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/unit-testing/ [02. ožujka 2021.]

4.1.2. Integracijski testovi

Nakon završetka jediničnih testova, jedinice koje su najmanji dio ovakvih testova povezuju se u veće cjeline. Testiranje tako povezanih cjelina čini drugu razinu testiranja i naziva se integracijsko testiranje.

Ovo testiranje predstavlja vrstu testiranja gdje se pojedinačne komponente kombiniraju odnosno povezuju, a zatim testiraju kao cjelina. Svrha ove razine testiranja je otkrivanje grešaka u interakciji između integriranih jedinica, provjera funkcionalnosti, performansi među cjelinama ali i njihove pouzdanosti. Može se provoditi pomoću metoda i crne i bijele kutije.[42]

Integracijsko testiranje je ponekad faza na koju se ne obraća dovoljno pažnje, a koje se, u praksi, često zanemaruje i ne provodi na adekvatan način. Vrlo često se postavlja i pitanje zašto provoditi ovakva testiranja. Odgovor se nameće sam od sebe, a to je da u ovoj fazi komponente zapravo prvi puta rade zajedno i sasvim je očekivano da će se pojaviti novi problemi.[43]

Prije provođenja samog testiranja bitno je dobro pripremiti testove što uključuje razrađivanje testnog plana, raspisivanje testnih scenarija, detaljno definiranje interakcije među jedinicama, dokumentiranje detalja o samim jedinicama.

Softversko inženjerstvo definira nekoliko tehnika integracijskog testiranja koje su opisane u nastavku teksta:[44]

- Integracija velikog praska (engl. Big bang)
- Integracija od vrha prema dnu (engl. Top-down)
- Integracija od dna prema vrhu (engl. Bottom-up)
- Sendvič integracija (engl. Sandwich)

Kod integracije velikog praska sve se komponente odjednom integriraju, a zatim testiraju kao cjelina. Ako sve komponente nisu dovršene postupak integracije se neće moći izvršiti. Ovakav pristup i nije najpraktičniji za upotrebu kod kompleksnijih softvera jer je teško lokalizirati

[42] Software Testing Fundamentals. (2020). *Integration Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/integration-testing/ [02. ožujka 2021.]
[43] Živković, M. (2018) *Testiranje softvera*. Prvo izdanje. Beograd: Univerzitet Singidunum.
[44] Ibid

greške, a često se može dogoditi da se nakon spajanja svih jedinica u cjelinu propusti testirati poneki dio. S druge strane ovakav pristup se vrlo lako može primijeniti na softver koji se sastoji od tek nekoliko komponenti.

Integracija od vrha prema dnu i od dna prema vrhu spadaju u inkrementalnu integraciju. Testiranje se kod ovakvih pristupa provodi tako da se integriraju dvije ili više komponenti koje su međusobno logički povezane, a zatim se testira ispravnost sučelja. Komponente se dodaju postepeno, jedna po jedna i nakon svakog dodavanja sljedeće komponente vrši se detaljno testiranje, a povezane komponente se testiraju kao cjelina. Razlika između integracija od vrha prema dnu i od dna prema vrhu se može utvrditi i prema nazivu. Kod integracije od vrha prema dnu nakon testiranja glavnog modula dodaju se novi hijerarhijski niži moduli. Kod integracije od dna prema vrhu postupak dodavanja novih modula je suprotan, nakon testiranja hijerarhijski najnižeg modula postepeno se dodaju i testiraju novi.

Sendvič integracija predstavlja spoj prethodne dvije integracije pa se često naziva i hibridnim integracijskim testiranjem. On funkcionira tako da se moduli najviše razine testiraju s modulima niže razine dok se istovremeno niži moduli povezuju s hijerarhijski višim modulima te se zatim testiraju kao cjelina. Ovakav oblik testiranja vrlo je pogodan za velike projekte koji se sastoje od nekoliko različitih potprojekata.

4.1.3. Sistemsko testiranje

Kronološki nakon integracijskog testiranja slijedi sistemsko testiranje. Ovo testiranje se temelji na testiranju sustava kao cjeline. Njegova svrha je provjera radi li sustav kao cjelina u skladu za definiranim zahtjevima.[45]

Obično se kod sistemskog testiranja koristi metoda crne kutije kod koje tester ne poznaje unutarnju strukturu sustava, ali se testovi rade temeljem specifikacija zahtjeva. Testiranja uglavnom provode testeri najčešće ručno, ali u posljednjih nekoliko godina sve više raste interes za automatiziranim testovima.

Glavni fokus sistemskih testova je testiranje cijelog softvera kako bi se provjerilo da sve komponente sustava rade zajedno odnosno kao cjelina (engl. end-to end). Ovaj pojam end-to

[45] Software Testing Fundamentals. (2020). *System Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/system-testing/ [02. ožujka 2021.]

end testiranje zapravo u praksi znači da se sustav testira u realnom scenariju. Tijekom testiranja instant platnog sustava također su se provodili end-to-end testovi koji su podrazumijevali uključivanje u test svih sudionika, uspostavljanje mreže komunikacije sa svim sudionicima sustava, povezivanje na bazu, osiguranje rada svih hardverskih komponenti koje su potrebne za provođenje testova. Dakle, cilj ovog testa je bio provjera rada sustava ali s naglaskom na korisnikovu perspektivu.

Postoji više od 50 različitih vrsta sistemskog testiranja. Neki od ključnih i onih koji se u praksi najčešće provode su funkcionalna testiranja, regresijska testiranja, testovi opterećenja, performansni testovi, stress testovi, sigurnosni testovi, testovi oporavka. Neke od ovih vrsta testiranja bit će detaljnije objašnjene kroz sljedećih nekoliko poglavlja.

4.1.4. Test prihvaćanja

Test prihvaćanja je vrsta testiranja koju provodi krajnji korisnik kako bi potvrdio da softver radi prema očekivanjima prije nego se počne izvoditi na produkcijskoj okolini, odnosno prije nego se pusti u rad.

Prema Milleru i Collinsu test prihvaćanja daje krajnjem korisniku potvrdu da softver ima sve definirane značajke i da se ponaša u skladu s očekivanim. U teoriji tek kad su svi testovi prihvaćanja uspješno završeni bez pronađenih kritičnih grešaka projekt se smatra dovršenim.

Generalno, ispravno provođenje testova prihvaćanja može dati korisne konačne odgovore: [46]

- testovima prihvaćanja se provjeravaju korisnički zahtjevi i u kojoj mjeri sustav odgovara tim zahtjevima
- njihovim provođenjem dolaze do izražaja greške koje nisu otkrivene tijekom jediničnih testiranja
- daju povratnu informaciju u kojoj mjeri je sustav dovršen.

Iako izvršavanje ovih testova zvuči jednostavno zapravo može predstavljati veliki izazov ako se ne definira ispravno. Problemi koji se najčešće javljaju su definiranje tko će raspisivati ovakve testove, tko će ih provoditi, kada se trebaju provoditi, koliko vremenski vremena treba

[46] Miller, R., Collins, C. T. (2001). Acceptance testing. *Proc. XPUniverse* [online]. Dostupno na: http://www.dsc.ufcg.edu.br/~jacques/cursos/map/recursos/Testing05.pdf [03. ožujka 2021.]

izdvojiti ze testiranje i na kraju kako mjeriti rezultate. U praksi scenarije za testove prihvaćanja najčešće raspisuje poslovni tim i to vrlo često i prije nego je razvojni tim u potpunosti implementirao kod. Rezultat ovih testova obično je binaran što znači da je testni korak uspješno izvršen ili je pronađena greška.

Testovima prihvaćanja instant platnog sustava ispitivao se tijek procesa i funkcionalnosti sustava sa stanovišta korisnika i tipično se provode u suradnji s krajnjim korisnicima (HNB, banke), prije samog prihvaćanja sustava i početka rada.

Testni scenariji te rezultati i detaljna analiza svih provedenih testova evidentirani su putem propisanih obrazaca i dokumentacije sukladno normi ISO 9001.

4.2. Metode testiranja

Način testiranja softvera ima jednaku važnost kao i sam softver. Ta rečenica zapravo govori o činjenici koliko je važan odabir metode testiranja. Postoji mnogo različitih metoda i pristupa koji se mogu koristiti. U osnovi ne postoji jedna metoda koja bi se mogla izdvojiti kao najbolja. U praksi je odabir metode najčešće kombinacija nekoliko različitih metoda. Metode testiranja koje se najčešće koriste su:[47]

- statičko testiranje
- dinamičko testiranje
- testiranje crne kutije (engl. black box testing)
- testiranje bijele kutije (engl. white box testing)
- testiranje sive kutije (engl. gray box testing)
- agilno testiranje
- ad hoc testiranje
- ručno testiranje
- automatizirano testiranje.

[47] Software Testing Fundamentals. (2020). *Software Testing Methods* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/software-testing-methods/ [03. ožujka 2021.]

4.2.1. Statičko testiranje

Statičko testiranje je metoda u kojoj se softver testira bez izvršavanja koda odnosno bez njegova pokretanja, bilo ručno ili putem alata.[48] Ovakva vrsta testiranja može se izvoditi i u ranim fazama razvoja softvera, odnosno i prije provođenja dinamičkih testova. Provoditi se mogu na izvornom kodu, dizajnu, modelima, arhitekturi ili funkcionalnim zahtjevima, a tijekom provođenja testa je moguće otkriti greške kao što su odstupanja od standarda, neodrživi kod, odstupanja od specifikacije i zahtjeva.[49]

Postoji nekoliko tehnika statičkog testiranja: [50]

- Pregled (engl. review)
- Walkthrough
- Inspekcija (engl. inspection).

Pregled sam po sebi može biti formalan ili neformalan. Ako je riječ o neformalnom pregledu tada se najčešće radi o prezentiranju službenih dokumenata projektnom timu i voditeljima projekata (dokumenti kao što su specifikacije zahtjeva, detaljni dizajn, tehničke specifikacije). Tijekom prezentiranja dokumentacije članovi projektnog tima mogu davati mišljenja i predlagati načine za poboljšanje softvera. S druge strane, formalni pregled zahtjeva formalni proces što znači da je tijek procesa reguliran i strukturiran. Faze tijekom formalnog pregleda su planiranje, kick-off, priprema, formalni sastanak, korekcije i follow-up. Za razliku od neformalnog pregleda, formalni se obavezno dokumentira.

Cilj tehnike walkthrough je prezentiranje dokumentacije kako bi se prenijelo i generiralo novo znanje, dobile povratne informacije, razvila diskusija i predložila bolja rješenja. Može se pojavljivati u različitim oblicima formalno ili neformalno.

Inspekcija predstavlja najformalniju kategoriju jer se provodi po definiranom postupku i s definiranom dokumentacijom. Najčešće ju provode osobe stručne u određenom području. Cilj

[48] Software Testing Fundamentals. (2020). *Static Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/static-testing/ [03. ožujka 2021.]
[49] Živković, M. (2018) *Testiranje softvera*. Prvo izdanje. Beograd: Univerzitet Singidunum.
[50] Software Testing Fundamentals. (2020). *Static Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/static-testing/ [03. ožujka 2021.]

je pronalaženje grešaka koje se detaljno dokumentiraju, procjena kvalitete, procjena performansi, utvrđivanje područja u kojem su moguća poboljšanja.

Neke od prednosti statičkih testova su rano otkrivanje grešaka što će rezultirati i njihovim ranijim ispravljanjem, smanjenje vremena razvoja softvera, smanjenje troškova i vremena testiranja što će u konačnici rezultirati povećanjem produktivnosti.

4.2.2. Dinamičko testiranje

Za razliku od statičkog testiranja, dinamičko se provodi tako da se testira softver koji se izvodi, odnosno testira se ponašanje softvera s dinamičkim varijablama ili varijablama koje nisu konstantne. Testovima se nastoje pronaći slabe točke softvera u stvarnom okruženju gdje se nakon unošenja ulaznih podataka rezultati uspoređuju s očekivanim rezultatima.

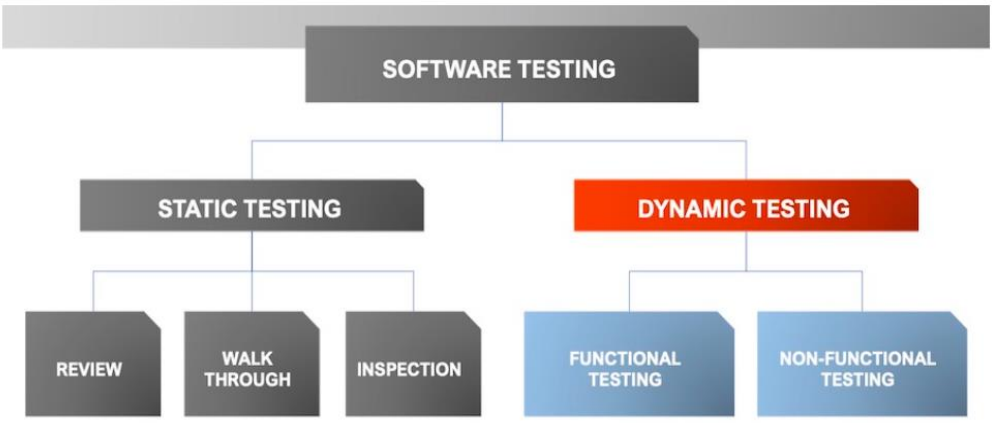
Tehnike dinamičkog testiranja klasificirane su u dvije kategorije: [51]

- funkcionalno testiranje
- nefunkcionalno testiranje.

Funkcionalno testiranje provodi se prema funkcionalnim zahtjevima, dok se nefunkcionalno testiranje usredotočuje na performanse, sigurnost, usklađenost, dostupnost, itd.

O funkcionalnom i nefunkcionalnom testu biti će više riječi u poglavlju Tipovi testiranja.

Slika 13. Shematski prikaz statičkog i dinamičkog testiranja



Izvor: Software Testing Fundamentals. (2020). *Dynamic Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/dynamic-testing/ [04. ožujka 2021.]

[51] Tutorialspoint (2021). Dynamic Testing [online], Tutorialspoint. Dostupno na: https://www.tutorialspoint.com/software_testing_dictionary/dynamic_testing.htm [04. ožujka 2021.]

Kao prednost dinamičkog testiranja svakako se ističe dubinsko testiranje što jamči veću kvalitetu, testiranje softvera iz korisničke i poslovne perspektive, a može se i automatizirati. Budući da je dinamičko testiranje temeljitije zahtjeva više vremena, više resursa, a i troškovi su veći.[52]

4.2.3. Testiranje metodom crne kutije

Prema ISTQB testiranje metodom crne kutije (engl. black box testing) ili poznata još kao funkcionalno testiranje, je postupak provođenja testa kod kojeg tester ne poznaje unutarnju strukturu, dizajn ili implementaciju softvera. Na softver se gleda kao na crnu kutiju koja obrađuje ulazne podatke te stvara izlazne. Ovom metodom mogu se otkriti greške kao što su pogrešna implementacija koda, greška u ponašanju sustava, performansni problemi.[53]

Može se primijeniti na svim razinama testiranja, počevši od jediničnih testova preko integracijskih pa sve do sistemskih. Za razliku od drugih testova testiranje metodom crne kutije je veće i kompleksnije.[54]

Glavna prednost testiranja metodom crne kutije je što testeri ne moraju imati znanje o određenim programskim jezicima kao niti znanje o samoj implementaciji, programeri i testeri su neovisni jedni o drugima ali je ipak bitna suradnja među njima.[55]

Najveći nedostatak ovakve metode testiranja je da uvijek postoji mogućnost da pojedine funkcionalnosti ne budu pokrivene testom. A kako postoji mogućnost da neki dijelovi ne budu pokriveni testom tako postoji određena vjerojatnost za ponavljanjem nekih testova ili barem dijelova testova koji su se već izvršili.

Tehnike testiranja crne kutije su: [56]

- klasa ekvivalencije

---

[52] Try QA (2021). Dynamic Testing Technique [online], Try QA. Dostupno na: http://tryqa.com/what-is-dynamic-testing-technique/ [04. ožujka 2021.]

[53] Software Testing Fundamentals. (2020). *Black Box Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/black-box-testing/ [03. ožujka 2021.]

[54] Živković, M. (2018) *Testiranje softvera*. Prvo izdanje. Beograd: Univerzitet Singidunum.

[55] Nidhra, S., Dondeti, J. (2012). Black box and white box testing techniques-a literature review [online]. *International Journal of Embedded Systems and Applications (IJESA)*, *2*(2). Dostupno na: https://www.researchgate.net/publication/276198111_Black_Box_and_White_Box_Testing_Techniques_-_A_Literature_Review [04. ožujka 2021.]

[56] Ibid

- analiza graničnih vrijednosti
- tablica odlučivanja
- uzročno posljedični dijagram.

Najkorištenija tehnika je analiza graničnih vrijednosti kod koje se greške pronalaze u granicama ulaznih vrijednosti. Klasa ekvivalencije se koristi za temeljita i iscrpna testiranja, provodi se podjelom ulaza na klase i dobivanjem vrijednosti iz svake klase. Tablica odlučivanja i uzročno posljedični dijagram su tehnike koje se fokusiraju na poslovnu logiku i pravila koja su definirana u specifikacijama. Pomoću tablice odlučivanja se mogu izraziti kompleksna poslovna pravila čime se značajno olakšava posao testerima. [57]Kako je testiranje svih mogućih kombinacija nemoguće tablica odlučivanja predstavlja dobru metodu za definiranje kombinacija koje će biti izostavljene, a koje će se testirati.

4.2.4. Testiranje metodom bijele kutije

Dok se metodom crne kutije tester usredotočuje na to što softver radi, kod metode bijele kutije (engl. white box testing) fokus se stavlja na to kako softver radi. Testiranje ovom metodom podrazumijeva da testeri poznaju unutarnju strukturu, dizajn i implementaciju softvera koji se testira.[58] Specifikacija ne igra veliku ulogu jer se tijekom testa provjerava sam kod, pa se često ova metoda naziva i metoda otvorenih kutija ili staklenih kutija. Može se primijeniti na svim razinama testiranja ali se najčešće primjenjuje nakon što su provedeni testovi metodom crne kutije.

Greške koje se ovom metodom mogu otkriti najčešće su: rupe u sigurnosti sustava, loše strukturirani kod, loše definirane petlje, nepredviđeni izlazi, logičke pogreške u kodu.
Neke od prednosti testiranja metodom bijele kutije su: [59]

- optimizira kod pronalaženjem skrivenih grešaka te pomaže u uklanjanju nepotrebnih, suvišnih linija koda
- mogu se lako automatizirati

[57] Živković, M. (2018) *Testiranje softvera*. Prvo izdanje. Beograd: Univerzitet Singidunum.
[58] Software Testing Fundamentals. (2020). *White Box Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/white-box-testing/ [04. ožujka 2021.]
[59] Khan, M. E., Khan, F. (2012). A comparative study of white box, black box and grey box testing techniques [online]. *International Journal of Advanced Computer Science and Applications (IJACSA)*, *3*(6). Dostupno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.685.1887&rep=rep1&type=pdf#page=22 [05. ožujka 2021.]

- maksimalna pokrivenost postiže se pisanjem testnih scenarija.

Kao nedostaci ističu se:

- kompleksno je i skupo, a od testera zahtjeva posebne vještine
- iako ima veliku pokrivenost, moguće je da dio koda ostane netestiran i s neprepoznatim kritičnim greškama
- oduzima puno vremena jer zahtjeva detaljno razumijevanje koda i načina implementacije.

4.2.5. Testiranje metodom sive kutije

Testiranje metodom sive kutije (engl. gray box testing) je kombinacija testiranja metodom crne i bijele kutije. Kod metode crne kutije tester ne poznaje unutarnju strukturu sustava koji testira, kod testiranja metodom bijele kutije struktura sustava je poznata dok je kod testiranja metodom sive kutije struktura sustava djelomično poznata.[60]

Ovakav način omogućuje da se testiranje provede i sa strane web aplikacije, ono što vidi korisnik, ali i na strani koda. Glavni cilj ove metode je pronaći greške koje nastaju zbog nepravilne strukture koda ili nepravilne upotrebe aplikacije.

Kao prednosti testiranja metodom sive kutije ističu se: [61]

- kombinirane prednosti metoda crne i bijele kutije
- nije nametljiva metoda jer se ne oslanja samo na kod već i na specifikacije, arhitekturu i dizajn aplikacije
- testiranja su nepristrana jer se na neki način izbjegnu sukobi između testera i programera.

Nedostaci testiranja metodom sive kutije su:

- ograničen pristup unutarnjoj strukturi dovodi do ograničenog testiranja, a time i do propuštanja potencijalnih kritičnih grešaka
- može biti suvišno ako je dio testova već odradio programer.

---

[60] Software Testing Fundamentals. (2020). *Gray Box Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/gray-box-testing/ [04. ožujka 2021.]

[61] Acharya, S., Pandya, V. (2012). Bridge between Black Box and White Box–Gray Box Testing Technique. *International Journal of Electronics and Computer Science Engineering* [online], *2*(1). Dostupno na: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.303.4479&rep=rep1&type=pdf [05. ožujka 2021.]

4.2.6. Ručno testiranje

Ručno testiranje je metoda testiranja kod koje tester priprema testne scenarije, a zatim ih provodi ručno kako bi prepoznao grešku bez upotrebe bilo kakvih automatiziranih alata. Predstavlja najrigorozniju i najstariju metodu testiranja softvera.[62]

Ova metoda podrazumijeva određene vještine koje su potrebne za provođenje testa. Prije svega to je strpljivost budući da ručno testiranje može biti vrlo kompleksno, zamorno i ponavljajuće. Osim strpljivosti tester mora posjedovati vještine kreativnosti, inovativnosti, upornosti i određenu otvorenost i intuiciju za prepoznavanje problema.

Ručno testiranje je također vrlo ključno i to na samom početku testiranja, jer svaki novi sustav ili aplikacija koji se implementiraju moraju biti najprije ručno testirani pa tek onda automatizirani. Ručno testiranje zahtjeva puno više napora ali na samom početku testiranja je neophodno kako bi se provjerila mogućnost provedbe automatizacije. Potpuna automatizacija testiranja softvera nije moguća, a ne može se reći niti da je uvijek potpuno precizna. Prema tome ručno testiranje iako još uvijek kao najprimitivnija metoda ostaje imperativ u testiranju softvera.

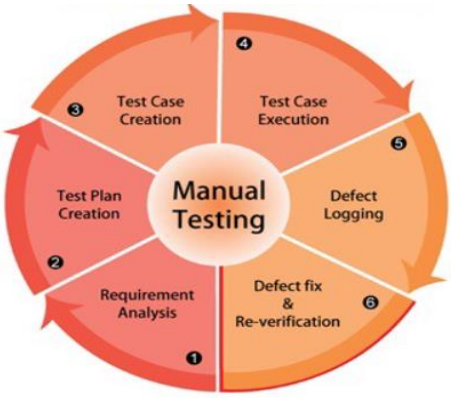
Kod provođenja ručnih testova testeri najčešće slijede određene procedure. Prije početka testiranja testeri detaljno proučavaju zahtjeve, specifikacije i razne analize. Na temelju takve dokumentacije testeri dobivaju dovoljno informacija kako bi izradili plan testiranja. Plan testiranja uključuje izradu dokumenta koji opisuje detalje kao što su uvjeti potrebni za testiranje, koji su ciljevi testiranja, koje komponente će se i na koji način testirati, tko su sudionici testa, vremenski raspored i plan provođenja testova te koji su testovi prioritetni. Ovi kriteriji moraju biti jasno definirani kako bi se moglo jasno procijeniti u kojoj je fazi testiranje kao i kada će ono biti završeno.

Nakon izrade plana testiranja testeri započinju raspisivanje testnih skripti. Testne skripte se sastoje od jasno definiranih testnih koraka koje testeru opisuju postupak testiranja. U fazi izvršavanja testnih skripti tester slijedi korake s ciljem pronalaženja grešaka uspoređujući dobiveni rezultat s očekivanim. Pronalaskom greške tester prijavljuje grešku razvojnom timu.

[62] Sharma, R. M. (2014). Quantitative analysis of automation and manual testing. *International journal of engineering and innovative technology [online]*, *4*(1). Dostupno na: https://www.imvcportal.com.au/uploads/study_material/1504593504IJEIT1412201407$%23@_46.pdf [06. ožujka 2021.]

Nakon izvršenih promjena u kodu razvojni tim potvrđuje promjene isporukom nove verzije aplikacije, a zatim novu verziju prosljeđuju testerima na novi krug testiranja.

Slika 14. Faze ručnog testiranja softvera



Izvor: Naznin, T. (2018). *Manual testing process* [online], Medium. Dostupno na: https://medium.com/oceanize-geeks/manual-testing-process-340173d40141 [06. ožujka 2021.]

Neke prednosti ručnog testiranja su:

- nije uvijek moguće sve testove automatizirati
- može se primjenjivati na manjem i većem softveru
- jednostavnije i lakše ažuriranje testnih scenarija
- daje širu perspektivu i sliku o cijelom sustavu
- ponekad su troškovi manji kada je potrebno provesti ad-hoc testiranja.

Nedostaci ručnog testiranja su:[63]

- za njihovo je provođenje potrebno više vremena i testera koji će provoditi test
- manje su pouzdani, ljudski je griješiti stoga je za očekivati da će biti propusta
- veća ulaganja u ljudske resurse zbog znanja i vještina koje testeri moraju usvojiti
- nije prikladno za testove koji se ponavljaju.

Razne studije slučaja su dokazale kako je ručno testiranje vrlo važno jer se neke ranjivosti sustava, kao na primjer sigurnosni propusti, mogu pronaći isključivo ručnim testovima. Testiranjem platnog sustava može se zaključiti kako je isto tako važno kombinirati ručne

[63] Sharma, R. M. (2014). Quantitative analysis of automation and manual testing. *International journal of engineering and innovative technology [online]*, *4*(1). Dostupno na: https://www.imvcportal.com.au/uploads/study_material/1504593504IJEIT1412201407$%23@_46.pdf [06. ožujka 2021.]

testove s određenim alatima kojima je uz ručne testove puno lakše pronaći veći broj grešaka nego kada se isključivo koristi jedna vrsta testa.

4.2.7. Automatizirano testiranje

Automatizirano testiranje predstavlja metodu koja se izvodi pomoću posebnih softverskih alata za izvršavanje testnih scenarija. Također ova metoda uključuje razvijanje alata na različitim programskim jezicima, kao što su Phyton, JavaScript, a koji će poslužiti za provođenje testnih scenarija s minimalnom intervencijom čovjeka.[64]

Ipak bitno je razlikovati, prema McKinley-u automatizacija se može podijeliti na dva različita termina: automatizirano testiranje i automatizacija testiranja. Iako imaju zajednički cilj, razviti pouzdan softver, i naizgled vrlo slični postoji ključna razlika među njima.[65]

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.[66]

Prednosti automatiziranog testiranja su različite:

- štede vrijeme budući da se automatizirani testovi brže izvode i bez velikog nadzora testera
- iako je potrebno uložiti određena sredstva za automatizaciju testova u konačnici su troškovi testiranja manji, a dugoročno mogu biti isplativiji
- zbog bržeg izvođenja testova osiguravaju veću pokrivenost
- ako su dobro isprogramirani manje su skloni pogreškama
- idealni su kod ponavljajućih testova koji za testera mogu biti vrlo zamorni
- ne zahtijevaju ljudsku intervenciju pa njihovo izvođenje nije ovisno o radnom vremenu.

---

[64] Sharma, R. M. (2014). Quantitative analysis of automation and manual testing. *International journal of engineering and innovative technology [online]*, *4*(1). Dostupno na: https://www.imvcportal.com.au/uploads/study_material/1504593504IJEIT1412201407$%23@_46.pdf [06. ožujka 2021.]

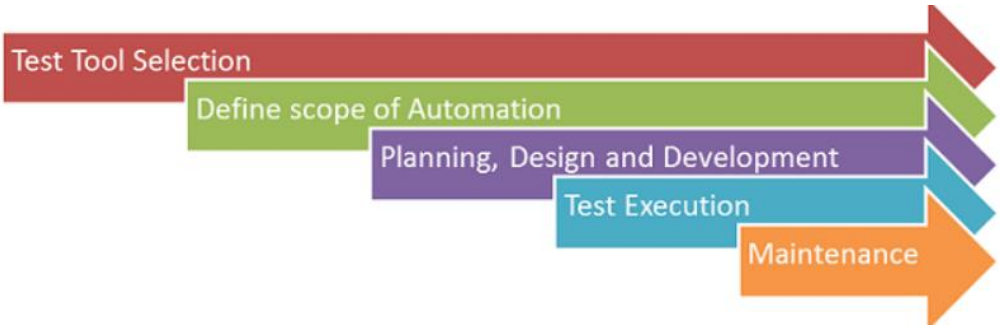
[65] McKinley, N. (2020). *Test Automation vs Automated Testing: Key Differences & Pros and Cons* [online]. Software Test Professionals. Dostupno na: https://www.softwaretestpro.com/test-automation-vs-automated-testing-key-differences-pros-and-cons/ [06. ožujka 2021.]

[66] Testim (2021). *Automated Testiing or Test Automation? You Need Both* [online]. Testim. Dostupno na: https://www.testim.io/blog/automated-testing-vs-test-automation/ [06. ožujka 2021.]

Nedostaci automatiziranih testova su:

- u početku ulaganje u automatizirano testiranje može biti skupo
- za razliku od ručnog testiranja, automatizirano nije lako prilagodljivo i svaka promjena u scenariju iziskuje dodatne izmjene u kodu
- greška u koracima i programiranju alata za automatizaciju može dovesti do fatalnih pogrešaka u radu sustava.

Slika 15. Faze automatiziranog testiranja softvera



Izvor: Guru99 (2021). *Automation Testin Tutorial: What is Automated Testing* [online]. Guru99. Dostupno na: https://www.guru99.com/automation-testing.html [06. ožujka 2021.]

Postupak automatizacije provodi se u nekoliko koraka. Prije samog početka potrebno je na temelju detaljne analize odabrati kvalitetan alat za testiranje koji će biti prilagođen potrebama softvera koji se testira. Zatim je potrebno odrediti opseg automatizacije. Definiranjem opsega postavljaju se prioriteti testiranja, kompleksnost testnih scenarija, količina podataka neophodna za provedbu testa, itd. Nakon definiranog opsega potrebno je razraditi strategiju, plan automatizacije te vremenski okvir provedbe testova. Nakon svih prethodno provedenih koraka slijedi provođenje samog testa. Nakon završenog testa alat najčešće daje detaljno izvješće o provedenom testu nakon čega slijedi određivanje zadovoljavaju li dobiveni rezultati definirane kriterije. Faza održavanja predstavlja posljednji korak. Bilo kakva izmjena u specifikaciji zahtjeva promjenu u provođenju testova stoga je promjene potrebno implementirati u same scenarije.

4.3. Tipovi testiranja

Svaki tester je na svom putu naišao ne nekoliko različitih vrsta testiranja. Postoji veliki raspon različitih vrsta testiranja. Svaka vrsta ima svoje specifične karakteristike, prednosti i nedostatke. Odabir same vrste ovisi o značajkama softvera koji se testira kao i o preferencijama testera. Generalno, postoje dvije osnovne vrste testiranja: funkcionalno i nefunkcionalno testiranje. Funkcionalna testiranja se provode prema funkcionalnim zahtjevima odnosno specifikacijama i provjeravaju ponaša li se softver prema očekivanom. [67] Zahtjevi ili specifikacije predstavljaju dokument u kojem je definirano koje je ponašanje softvera prihvatljivo.

U funkcionalna testiranja ubrajaju se:

- jedinična testiranja (engl. unit testing)
- integracijski testovi
- regresijska testiranja
- testovi prihvaćanja
- sistemski testovi.

Nefunkcionalna testiranja rade na principu nefunkcionalnih zahtjeva kao što su, performanse sustava, sigurnost, usklađenost, pouzdanost. Cilj nefunkcionalnih testova je povećati iskoristivost, učinkovitost, održivost, optimizaciju, dostupnost, skalabilnost, fleksibilnost, poboljšati znanje o ponašanju softvera.[68]

U nefunkcionalna testiranja ubrajaju se:

- testovi performansi
- sigurnosni testovi
- stress testovi
- testovi upotrebljivosti (engl. usability testing)
- testovi sukladnosti (engl. compliance testing).

[67] Software Testing Fundamentals. (2020). *Software Testing Types* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/software-testing-types/ [06. ožujka 2021.]

[68] Guru99 (2021). *What is Non Functional Testing? Types with Example* [online]. Guru99. Dostupno na: https://www.guru99.com/non-functional-testing.html [06. ožujka 2021.]

Tablica 3. Usporedba karakteristika funkcionalnih i nefunkcionalnih testova

	Funkcionalno testiranje	Nefunkcionalno testiranje
definicija	sustav se testira prema funkcionalnim zahtjevima	sustav se testira prema nefunkcionalnim zahtjevima (performanse, sigurnost, usklađenost)
fokus	temelji se na zahtjevima naručitelja	temelji se na očekivanjima naručitelja
funkcija	testira što softver radi	testira kako softver radi
provođenje	najčešće ručno	obično su automatizirana
razine	provode se na svim razinama testiranja	provodi se tijekom sistemskih testova i testova prihvaćanja
metoda	koristi se kod metoda testiranja crne kutije	koristi se kod metoda testiranja bijele kutije

Izvor: Software Testing Fundamentals (2020). Functional vs Non functional Testing [online]. Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/functional-testing-vs-non-functional-testing/ [07. ožujka 2021.]

Svaki od navedenih testova ima svoje karakteristike po čemu su prepoznatljivi. Usporedba tih karakteristika prikazana je u tablici 3. U sklopu testiranja platnog sustava provodili su se regresijski testovi, testovi performansi i sigurnosni testovi. Kratki opis i osvrt na ove testove slijedi u nastavku poglavlja.

4.3.1. Regresijsko testiranje

Svrha regresijskog testiranja je osigurati da promjene napravljene na softveru, poput dodavanja novih funkcionalnosti ili izmjena postojećih, nisu negativno utjecale na funkcionalnosti softvera koje se ne bi trebale mijenjati.[69] To zapravo znači da se regresijskim testiranjem ponavljaju testovi koji su već izvršeni u prethodnim iteracijama, ali se provode ponovo kako bi tester bio siguran da novoimplementirane funkcionalnosti nisu utjecale na rad postojećih i da sustav radi sukladno očekivanjima.

Regresijska testiranja provode se u slučaju:[70]

- novih zahtjeva u specifikaciji softvera

[69] Wong, W. E., Horgan, J. R., London, S., Agrawal, H. (1997) A study of effective regression testing in practice. *Proceedings The Eighth International Symposium On Software Reliability Engineering* [online]. Dostupno na: https://ieeexplore.ieee.org/abstract/document/630875 [07. ožujka 2021.]

[70] Živković, M. (2018) *Testiranje softvera*. Prvo izdanje. Beograd: Univerzitet Singidunum.

- dodavanja novih funkcionalnosti
- uklanjanja postojećih funkcionalnosti
- ispravljanje kritičnih grešaka
- optimizacije u svrhu poboljšanje performansi sustava.

Ovisno o situaciji regresijsko testiranje može imati različit opseg. Promjene u softveru često se klasificiraju kao korektivne i preventivne, kakve god bile zahtijevaju određeni opseg regresijskog testiranja.[71]

Prema tome se ovaj tip testiranja može provoditi kao potpuni test regresije gdje se podrazumijeva da se uz nove testove kojima se testira nova funkcionalnost provedu i postojeći testovi iz prethodnih iteracija. Iako je ovakav način idealan, u praksi je skup, oduzima puno vremena, resursa, a vrlo često nije u potpunosti niti izvediv.

Suprotno od potpunih regresijskih testova su testovi djelomične regresije. Test djelomične regresije svodi se na odabir i izvršavanje dijela testova, dok se dio testova izostavlja. Izbor testa se temelji na određivanju prioriteta i određivanju verzije. Kod određivanja prioriteta prednost se daje testnim scenarijima koji imaju veći poslovni utjecaj, složenu implementaciju, kritične funkcionalnosti ili pak često korištene funkcionalnosti. S druge strane kod određivanja verzije prednost se daje testnim scenarijima kojima će se pokriti testiranje promjena u isporučenoj verziji.

Regresijska testiranja mogu oduzimati puno vremena ako je njihovo ponavljanje učestalo, pogotovo ako se regresijska testiranja provode ručno. Ručna testiranja osim što oduzimaju vrijeme mogu značajno povećati i troškove. Kako živimo u vremenu s velikim tehnološkim otkrićima i promjenama, vrijeme isporuke verzije se skraćuje, pa je isporuku nove verzije softvera potrebno napraviti gotovo odmah. Uzimajući u obzir samo tu činjenicu može se zaključiti kako ručno regresijsko testiranje i nije najsretniji odabir. Prema tome regresijska testiranja su jako dobar kandidat za automatizaciju pa se u praksi ako je to moguće regresijski testovi ili u potpunosti automatiziraju ili se koristi neki alat za automatizaciju dijela testa.[72]

Prednost regresijskog testiranja svakako je to što igra presudnu ulogu u agilnom okruženju gdje se sa svakim sprintom mora osigurati stabilna verzija softvera, pa prema tome regresija

[71] Ammann, P., Offutt J. (2008) *Intraduction to Software Testing*. Cambridge: Cambridge University Press.
[72] Živković, M. (2018) *Testiranje softvera*. Prvo izdanje. Beograd: Univerzitet Singidunum.

na neki način jamči kontinuitet poslovanja.[73] Osim što povećava kontinuitet poslovanja kao prednost se također ističe lakše identificiranje pogrešaka budući da se svakom iteracijom povećava šansa za pronalazak grešaka.

Glavni nedostatak regresijskog testiranja su najčešće kratki vremenski rokovi, višestruko izvođenje testova i za najmanju promjenu koda, težnja maksimalnoj pokrivenosti, ograničeni resursi.

4.3.2. Performansni testovi

Iako vrlo bitna, funkcionalnost nije jedina karakteristika koja se testira. Jednako važne su i performanse sustava, kao što su karakteristike izvedbe, vremena odaziva, skalabilnost, pouzdanost.[74] Funkcionalnim testovima se navedene karakteristike ne mogu testirati, a ako se ne testiraju softver se često u 'živom' radu može susresti s raznim problemima poput sporog rada.

Svrha performansnih testova je utvrditi kako sustav funkcionira i koja je njegova stabilnost ali kada je pod većim opterećenjem od očekivanog.[75] Cilj ovih testova nije pronalaženje grešaka već eliminacija uskih grla koji utječu na performanse sustava, ubrzati rad softvera, osigurati softver stabilnim i pouzdanim za korištenje.

Testiranjem performansi sustava utvrđuje se zadovoljava li softver:[76]

- brzinu, provjerava se brzina kojom se aplikacija odaziva
- skalabilnost kojom se provjerava maksimalno opterećenje koje softver može podnijeti
- stabilnost kojom se provjerava koliko je sustav stabilan pod određenim opterećenjem
- pouzdanost kojom se utvrđuje je li softver siguran ili nije.

Testovi performansi mogu se provoditi u nekoliko različitih oblika: [77]

---

[73] Testsigma (2021). *A Detailed analysis on Advatages, Disadvantages, Challenges and Risks of Regression Testing* [online]. Testsigma. Dostupno na https://testsigma.com/regression-testing/advantages-of-regression-testing [07. ožujka 2021.]

[74] Denaro, G., Polini, A. Emmerich, W. (2004) Early Performance testing of distributed software applications. *Association for Computing Machinery* [online], 29(1). Dostpuno na: https://dl.acm.org/doi/abs/10.1145/974044.974059 [27. veljače 2021.]

[75] Software Testing Fundamentals. (2020). *Performance Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/performance-testing/ [08.ožujka 2021.]

[76] GeeksforGeeks (2019). *Performance Testing: Software Testing* [online]. Noida: GeeksforGeeks. Dostupno na: https://www.geeksforgeeks.org/performance-testing-software-testing/ [08.03.2021.]

[77] Software Testing Fundamentals. (2020). *Performance Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/performance-testing/ [08.ožujka 2021.]

- testiranje opterećenja (engl. load testing) je test koji se provodi radi procjene ponašanja sustava pri povećanom radnom opterećenju.
- stress test (engl. stress testing) je test koji se provodi kako bi se procijenilo ponašanje sustava na granicama očekivanog opterećenja ili izvan njih.
- testiranje izdržljivosti (engl. endurance testing) koje se provodi radi procjene ponašanja sustava kada se kontinuirano daje značajno radno opterećenje.
- 'spike' testiranje (engl. spike testing) koje se provodi kako bi se utvrdilo ponašanje softvera kada je opterećenje naglo i znatno povećano.

Postoji više različitih parametara koji se mogu pratiti tijekom provođenja testa performansi. Najčešće su to opterećenost rada procesora, upotreba memorije, opterećenje mreže, vremena odaziva, propusnost sustava, brzina pristupanja bazi, broj istovremenih konekcija prema bazi, maksimalan broj sesija aktivnih u istom trenutku, itd.

Tijekom testiranja instant platnog sustava najčešći problemi s kojima se testni tim susretao su brzina rada sustava, vremena odaziva i vremena obrade poruka. Budući da u platnom sustavu sudjeluje više sudionika sve navedene komponente su ovisile o vremenima odaziva svih sudionika. Stoga je za postizanje optimalnog vremena obrade poruka od svega nekoliko sekundi bila potrebna suradnja svih sudionika kako bi se zadovoljili očekivani rezultati. Kao uska grla u provedenim testiranjima najčešće su se pokazale greške u programiranju, ponekad i loše vrijeme odaziva u komunikaciji s bazom ili u krajnjem slučaju loše postavljena mrežna komunikacija između sudionika.

4.3.3. Sigurnosni testovi

Koliko god smo kao čovječanstvo napredovali u razvoju tehnologije, digitalnom poslovanju toliko je s druge strane ta ista tehnologija i poslovanje nezaštićeno i ranjivo. Sigurnosne prijetnje postale su svakodnevica, a zbog njihovog porasta važnost sigurnosti u cyber svijetu postala je presudna za gotovo sve oblike poslovanja.

Sigurnosno testiranje može se opisati kao vrsta testiranja softvera čiji je cilj identificiranje ranjivosti koje potencijalno mogu dopustiti zlonamjerni napad.[78]

[78] Zola, A. (2020). What's the role of security testing in software development?. Information Age. Dostupno na: https://www.information-age.com/whats-role-security-testing-software-development-123487978/ [09.ožujka 2021.]

Nedavno istraživanje pokazalo je kako samo 36% ispitanika uključuje timove za kibernetsku sigurnost u početne faze digitalnih inicijativa. Istodobno njih 60% je reklo da je zabilježen porast sigurnosnih napada tijekom prošle godine. U prilog tome govori i činjenica kako su troškovi napada na podatke u posljednjih 5 godina porasli na oko 3,92 milijuna dolara. [79]

U sigurnosnom testiranju testeri koristite pristupe temeljene na riziku, odnosno ti pristupi temelje se na arhitektonskoj stvarnosti sustava ali i na načinu razmišljanja napadača i sve to da bi na odgovarajući način procijenili sigurnost softvera. Utvrđivanjem rizika u sustavu i stvaranjem testova vođenih tim rizicima, tester se tada može usredotočiti na područja koda u kojima će napad vjerojatno i uspjeti.[80]

Prema QA TestLab-u postoji 7 vrsta sigurnosnih testiranja: [81]

- skeniranje ranjivost (engl. Vulnerability scanning) provodi se provjerom sustava skeniranjem poznatih ranjivosti.
- sigurnosno skeniranje (engl. Security scanning) podrazumijeva provjeru mrežnih i sistemskih slabosti.
- penetracijski testovi (engl. Penetration testing) simuliraju napad zlonamjernog koda te provjeravaju potencijalne ranjivosti na pokušaje vanjskih neovlaštenih upada u sustav.
- procjena rizika (engl. Risk assessment) analizira sigurnosne rizike uočene u organizaciji, kao rezultat daju se preporuke kontrole i mjere za smanjenje rizika.
- revizija ranjivosti (engl. Security Auditing) je interna revizija sustava s ciljem pronalaska sigurnosnih nedostataka.
- procjena sigurnosnog držanja (engl. Posture Assessment) odnosi se na sigurnosni status i osviještenost o važnosti sigurnosti unutar sustava ili organizacije.
- etičko hakiranje (engl. Ethical hacking) je metoda legalnog provaljivanja u računala i uređaje unutar organizacije s ciljem testiranja obrane organizacije.

Mnogo je različitih čimbenika koji će utjecati na odabir određene metode ili alata za provođenje sigurnosnih testova. Iako ne postoji jedinstveno rješenje ili metoda provedba ove

[79] Ibid
[80] Potter, B., McGraw, G. (2004). Software security testing [online]. *IEEE Security & Privacy*, *2*(5). Dostupno na: https://ieeexplore.ieee.org/abstract/document/1341418 [09. ožujka 2021.]
[81] Vasylyna, N. (2020). *7 Types of Security Testing* [online]. QA TestLab Blog. Dostupno na: https://blog.qatestlab.com/2020/09/07/security-testing-types/ [08.ožujka 2021.]

vrste testiranja od iznimne je važnosti u zaštiti podataka, aplikacije ili cijele organizacije. Poduzimanje pravovremenih mjera u zaštiti sustava i organizacije svakako će u konačnici utjecati na reputaciju organizacije.

4.4. Životni ciklus greške

Greška predstavlja određeni nedostatak u softveru koji utječe bilo na rad samo pojedine komponente bilo na rad cijelog softvera. Nemoguće je napraviti softver bez grešaka, a greške su prisutne u čitavom životnom ciklusu softvera. Cilj testiranja softvera je pronalaženje i reduciranje broja grešaka te isporuka softvera sa što manjim brojem grešaka koje mogu utjecati na kvalitetu samog softvera.

Proces upravljanja greškama (eng. Defect Management Process) je postupak upravljanja greškama jednostavnim prepoznavanjem i popravljanjem greške.[82]

Nakon što tester pronađe greške one se evidentiraju kako bi se lakše mogao pratiti njihov status, a na kraju krajeva kako bi i razvojni tim imao bolji uvid u dio softvera koji je potrebno ispraviti. Postoje razni alati koji omogućuju praćenje životnog ciklusa greške.

Svaka greška ima svoj utjecaj na rad sustava stoga se svakoj treba pristupiti prema stupnju ozbiljnosti greške. Kategorije ozbiljnosti greške mogu biti:[83]

- kritična (engl. critical) koja najčešće rezultira prekidom rada cijelog sustava ili pojedinih komponenti, ugrožen je rad cijelog sustava koji postaje neupotrebljiv.
- visoka (engl. serious) koja također rezultira prekidom rada cijelog sustava ili pojedinih ključnih komponenti ali ipak postoji alternativna metoda kao zaobilazno rješenje.
- umjerena (engl. moderate) koja ne izaziva prekid rada sustava, utječe na manje funkcionalnosti ali i dalje djeluje na ponašanje softvera koje nije u skladu sa zahtjevima.

[82] GeeksforGeeks (2020). *Defect Management Process* [online]. Noida: GeeksforGeeks. Dostupno na: https://www.geeksforgeeks.org/defect-management-process/ [10. ožujka 2021.]
[83] Noor, R., Khan, M. F. (2014) Defect management in agile software development. *International Journal of Modern Education and Computer Science* [online], *6*(3). Dostupno na: http://www.mecs-press.org.ua/ijmecs/ijmecs-v6-n3/IJMECS-V6-N3-7.pdf [10. ožujka 2021.]

- manja (engl. low priority) koja ne utječe na prekid rada sustava i njen ispravak nije vremenski određen.

Osim prema ozbiljnosti greške se mogu klasificirati i prema prioritetu:[84]

- hitan prioritet imaju greške koje zahtijevaju trenutno ispravljanje i bez čijeg ispravka nema smisla nastavljati testiranje, a sam softver je neupotrebljiv
- visok prioritet imaju greške koje ugrožavaju osnovnu funkcionalnost i čija ispravka mora biti u vrlo kratkom roku
- srednji prioritet imaju greške koje ne ugrožavaju rad sustava, a njihov ispravak može se pričekati
- niski prioritet imaju greške koje nemaju nikakav utjecaj na rad sustava i čija ispravka može biti odgođena dok se ne poprave greške s višim prioritetom.

Životni ciklus greške predstavlja specifičan skup faza kroz koje greška prolazi tijekom svog života. Ciklus započinje u onom trenutku kada se greška detektira, a završava kada se ispravi tj. osigura da se ona više ne reproducira.[85]

Životni ciklus greške može se razlikovati od organizacije do organizacije pa čak i od projekta do projekta. Ponajviše ovisi o čimbenicima poput modela kojim se softver razvija, vremenskog ograničenja, politike organizacije, strukture tima.

Prema slici 16 životni ciklus greške uključuje sljedeće statuse:

- kada se greška prvi puta evidentira ona dobiva status 'novo'
- kada je greška upućena razvojnom timu ili određenom programeru u timu tada ona dobiva status 'dodijeljena'
- nakon što programer preuzme grešku te počne raditi na njenom ispravljanju ona dobiva status 'otvorena'
- nakon što programer napravi određene promjene u kodu i potvrdi napravljene promjene tada ona dobiva status 'ispravljena' te se iz tog statusa prosljeđuje testnom timu na testiranje

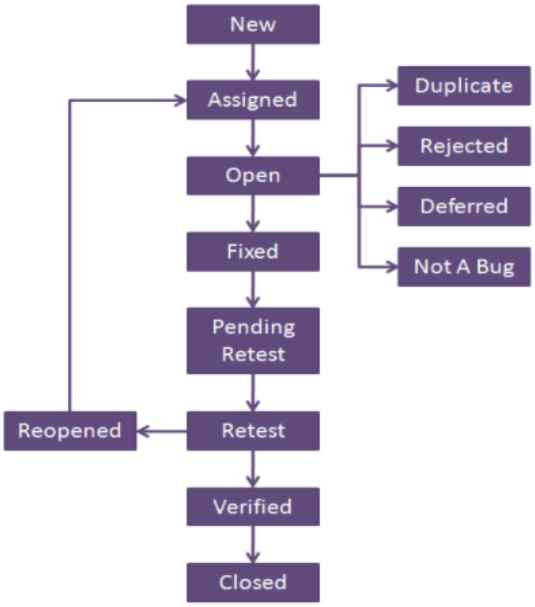
[84] Software Testing Fundamentals. (2020). *Defect Severity* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/defect-severity/ [08.ožujka 2021.]
[85] Try QA (2021). What is a Defect Life Cycle or Bug lifecycle in software testing? [online], Try QA. Dostupno na: http://tryqa.com/what-is-a-defect-life-cycle/ [12. ožujka 2021.]

- iz statusa ispravljena programer prebacuje grešku na testni tim te tada ona dobiva status 'čeka na ponovo testiranje'
- kada tester preuzme grešku za testiranje ona dobiva status 'ponovno testiranje'
- ako tester prilikom retestiranja nije pronašao grešku, greška se smatra 'ispravljenom'
- ako je greška i dalje prisutna tj. nije ispravljena tada ju tester vraća u obradu odnosno postavlja status 'ponovo otvaranje'
- ako je tester utvrdio da je greška ispravljena i odobrena tada joj postavlja status na 'zatvorena'

Osim ovih statusa greška iz statusa otvorena može biti postavljena u sljedeće statuse:

- 'duplikat' kada je već napravljena prijava iste greške
- 'odbijena' kada programer smatra da prijava nema osnovu za prijavu te uz obrazloženje objašnjava zašto se greška odbija
- 'odgođena' kada se ispravljanje greške postavlja na privremeno čekanje, razlog tome može biti promjena prioriteta iz visok u nizak ili nisu jasno definirana poslovna pravila pa ispravak greške traži dodatnu poslovnu analizu
- status ‘nije greška’ može se postaviti u situacijama kada je došlo do promjene funkcionalnosti.

Slika 16. Životni ciklus greške



Izvor: Try QA (2021). What is a Defect Life Cycle or Bug lifecycle in software testing? [online], Try QA. Dostupno na: http://tryqa.com/what-is-a-defect-life-cycle/ [12. ožujka 2021.]

Kada su svi testovi izvršeni i proces testiranja se smatra završenim potrebno je evidentirati rezultate testiranja kako bi se lakše mogao pratiti tijek provođenja testa, koji su testovi izvršeni u kojem vremenskom roku te tko su bili sudionici testa. Dokument s rezultatima testiranja naziva se izvještaj o testiranju. Ovisno od organizacije do organizacije izvještaji najčešće imaju definiranu formu s jasno definiranim podacima koje izvještaj mora sadržavati. Najčešće su to uvodni dio s općenitim podacima o tome kada se provodio test, tko je sudionik testiranja, koja vrsta testova je provođena, podaci o okolinama na kojima je test provođen, verzija softvera, itd. Nakon općenitih informacija izvještaj najčešće sadrži prikaz rezultata testiranja s detaljnim popisom i objašnjenjem pojedine greške, stupanj ozbiljnosti, prioriteta, mogućnost njenog ponavljanja. Na kraju izvještaja se nalazi zaključak o kvaliteti softvera te njegovom stanju za produkcijski rad. Izvještaj se podnosi voditeljima projekata, menadžerima i ostalim zainteresiranima.

4.5. Metrike testiranja softvera

Tijekom razvoja softvera izuzetno je važno mjeriti kvalitetu, trošak i učinkovitost. Mjerenje je oduvijek bilo temelj za napredak mnogih disciplina stoga ima jednaku važnost i u testiranju softvera. Softverske metrike se koriste u procjeni procesa testiranja pružajući objektivne kriterije mjerenja za donošenje upravljačkih odluka.[86]

Metrika zapravo predstavlja mjeru koja pomaže u procjeni napretka i kvalitete proizvoda u procesu testiranja softvera. Važnost metrike u testiranju softvera može se navesti kroz nekoliko prednosti: [87]

- pomažu u donošenju odluka za sljedeću fazu aktivnosti
- dokaz su tvrdnje ili predviđanja
- pomažu u razumijevanju potrebe za poboljšanjem softvera

[86] Farooq, S. U., Quadri, S. M. K., Ahmad, N. (2011) Software measurements and metrics: Role in effective software testing. *International Journal of Engineering Science and Technology* [online], *3*(1). Dostupno na: https://www.researchgate.net/profile/Sheikh-Umar-Farooq/publication/50392202_SOFTWARE_MEASUREMENTS_AND_METRICS_ROLE_IN_EFFECTIVE_SOFTWARE_TESTING/links/53ee42d00cf26b9b7dc65bb7/SOFTWARE-MEASUREMENTS-AND-METRICS-ROLE-IN-EFFECTIVE-SOFTWARE-TESTING.pdf [13. ožujka 2021.]

[87] Kalwan, A. (2020) *What is Software Testing Metrics and What are the Types?* [online]. Edureka!. Dostupno na: https://www.edureka.co/blog/software-testing-metrics/ [13.. ožujka 2021.]

- olakšava postupak odlučivanja ili promjene tehnologije.

Softverske metrike mogu se klasificirati u sljedeće kategorije:[88]

- metrike procesa (engl. process metrics) procjenjuju različite karakteristike procesa te daju rezultate koji se mogu koristiti za poboljšanje procesne učinkovitosti u životnom ciklusu razvoja softvera
- metrike proizvoda (engl. product metrics) se mogu koristiti kod određivanja kvalitete softvera
- metrike projekta (engl. project metrics) se mogu prikupiti u bilo kojoj fazi projekta, a koriste se kako bi se stekla bolja predodžba o statusu projekta ali i šira slika o samom projektu.

Nakon definiranja što su softverske metrike potrebno je navesti i kategorije mjernih podataka. Mjerni podaci omogućuju procjenu uspjeha neke aktivnosti, a uspjeh se svakako može definirati kao postizanje određenog cilja. Cilj metrika koje se provode u testiranju svakako su postizanje određene razine kvalitete softvera, zadovoljstvo kupaca, isporuka pouzdanog softvera, niski troškovi razvoja. Jedan od mjernih podataka je pokrivenost testom. Pokrivenost testom može razjasniti koji su to dijelovi softvera testirani. Primjer ovakvih mjernih podataka su: kolika je pokrivenost zahtjevima, testnim scenarijima, kolika je pokrivenost provedenih jediničnih testova, ručnih ili automatiziranih testova.[89]

Učinkovitost testiranja je mjerni podatak koji pokazuje koliko su bili korisni testovi prilikom otkrivanja grešaka. Učinkovitost uključuje podatke kao što su postotak uspješno provedenih testova, postotak kritičnih grešaka u odnosu na postotak svih grešaka.

Testni napor uključuje podatke kao broj provedenih testova, broj prijavljenih grešaka po danu, prosječno vrijeme potrebno za ispravak greške.

[88] Tutorialspoint (2021). Software Mesurement Metrics [online]. Tutorialspoint. Dostupno na: https://www.tutorialspoint.com/software_quality_management/software_quality_measurement_metrics.htm [12. ožujka 2021.]

[89] SeaLights (2021). *What Are Test Metrics?* [online]. SeaLights. Dostupno na: https://www.sealights.io/agile-testing/test-metrics/ [02.03.2021.]

Jedan od vodećih mjernih podataka je postotak izvršenosti testa kojim se utvrđuje koliko je testova stvarno provedeno, koliko ih je klasificirano kao neuspješno provedeni, nedovršeni, prekinuti. Glavna prednost ovog mjernog podatka je njegova jednostavna procjena.

Regresija je mjerni podatak kojim se utvrđuje stabilnost softvera nakon provedenih izmjena u kodu. Regresija omogućuje procjenu koliko je ta promjena koda bila učinkovita.

Trošak testiranja svakako je jedan od mjernih podataka koji može doprinijeti planiranju proračuna testnih aktivnosti. Primjer mjernih podataka su ukupni trošak testiranja, stvarni trošak testiranja, proračunska odstupanja, cijena ispravka po grešci, cijena implementacije novih zahtjeva, itd.

Osim metrika testiranja kroz različitu literaturu se ističu i metrike kvalitete softvera. One se opisuju kao mjere kojima se procjenjuje kvaliteta konačnog projekta. Glavne karakteristike kvalitetnog softvera su:[90]

- pouzdanost, kvalitetan softver bi trebao raditi bez zastoja stoga je na vrijeme potrebno spriječiti ozbiljne kvarove i prekide rada koji mogu utjecati na konačnu kvalitetu. Pouzdanost se može procijeniti provjerom broja incidenata, testiranjem pouzdanosti, opterećenja, provjerom prosječnog broja prijavljenih kvarova.
- održavanje, kvalitetan softver jednostavan je za održavanje što znači da je njegov kod razumljiv i jednostavan za provjeru. Lako održivom kodu potrebno je manje vremena i troškova za prilagodbu promjenama. Održivost koda može se izmjeriti provjerom linija koda i provođenjem statičke analize koda.
- sigurnost, manje ranjiv softver sigurniji je za korištenje. Veći broj ranjivosti s visokom ozbiljnosti važan je pokazatelj njegove sigurnosti. Sigurnost se može izmjeriti utvrđivanjem broja ranjivosti, utvrđivanjem vremena potrebnog za rješavanje incidenata, uvođenjem sigurnosnih procedura.
- prijenosnost, kvalitetan softver je moguće koristiti u raznim okruženjima.

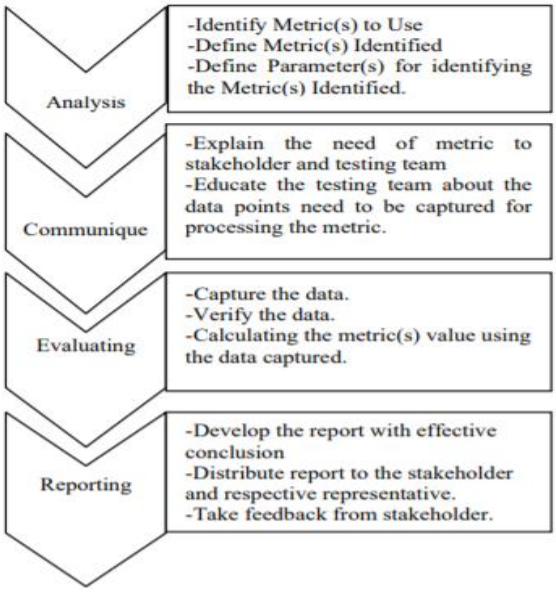
U zrelijim organizacijama postoje jasno definirane procedure i načini provođenja bilo automatiziranih ili ručnih provjera metrika softvera. Tijekom testiranja platnog sustava može se zaključiti kako je platni sustav prošao razna mjerenja. Neke od provjera koje su provođene

[90] Diceus (2020) *Top 10 Software Quality Metrics That Matter* [online]. Diceus. Dostupno na: https://diceus.com/top-7-software-quality-metrics-matter/ [14. ožujka 2021.]

nad platnim sustavom su statička analiza koda, provjera opterećenja sustava, provjera ranjivosti sustava, jasno su definirane procedure postupanja u slučaju incidenata te su jasno definirane sigurnosne procedure i pravila.

Tijekom provođenja mjerenja ono prolazi određene faze. Na slici uz objašnjenje ispod slike je prikazan životni ciklus softverske metrike. [91]

Slika 17. Životni ciklus softverske metrike



Izvor: Nirpal, P. B., Kale, K. V. (2011) A brief overview of software testing metrics. *International Journal on Computer Science and Engineering* [online], *3*(1). Dostupno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.5632&rep=rep1&type=pdf [12. ožujka 2021.]

Životni ciklus započinje analizom koja je odgovorna za identificiranje metrike koje će se koristiti kao i definiciju mjernih podataka i parametara. Sljedeća faza u životnom ciklusu je komunikacija. Komunikacija je bitna kako bi se testnom timu ukazala potreba za provođenjem softverskih metrika. Osim toga ova faza podrazumijeva i educiranje članova testnog tima o mjernim podacima koje je potrebno prikupiti za provođenje mjerenja. Faza procjene snima i verificira podatke zatim se na temelju snimljenih podataka izračunavaju rezultati. Na kraju životnog ciklusa nalazi se faza izvješćivanja. U ovoj fazi izrađuju se izvješća s konkretnim zaključcima. Izvješća s rezultatima provjere se dijele sa svim sudionicima te se na temelju rezultata donose konačne odluke koje u konačnici mogu pomoći u poboljšanju kvalitete softvera.

[91] Nirpal, P. B., Kale, K. V. (2011) A brief overview of software testing metrics. *International Journal on Computer Science and Engineering* [online], *3*(1). Dostupno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.5632&rep=rep1&type=pdf [12. ožujka 2021.]

5. ANALIZA PREDNOSTI I NEDOSTATAKA RUČNIH I AUTOMATIZIRANIH TESTOVA U FAZI RAZVOJA PLATNOG SUSTAVA

U nastavku ovog poglavlja biti će opisani platni sustavi u Hrvatskoj, što su i koja je njihova funkcija. Poseban naglasak stavit će se na instant platni sustav te njegov razvoj s fokusom na proces testiranja. Osim opisa instant platnog sustava u poglavlju će biti opisane temeljne funkcionalnosti koje je platni sustav morao zadovoljiti kako bi zaživio i kao takav krenuo u produkcijski rad. Instant platni sustav predstavlja digitalizaciju platnog prometa i primjer je nacionalnog sustava koji je izgrađen, obzirom na kompleksnost sustava, u vrlo kratkom roku temeljem znanja i iskustva internih specijalista i stručnjaka.

Da bi sustav zaživio morao je proći temeljita testiranja i provjere kvalitete kako bi na kraju postao siguran, pouzdan sustav za izvršenje platnih transakcija.

Tijekom procesa testiranja provodile su se različite vrste, metode i tehnike testiranja. Metode i tehnike testiranja koje su korištene tijekom razvoja teoretski su opisane u prethodnim poglavljima dok će njihova praktična primjena biti detaljnije opisana u nastavku poglavlja. U ovom poglavlju dati ću osvrt na prednosti i nedostatke testova koji su provođeni, naročito ručnih i automatiziranih. Temeljem iskustva stečenog na razvoju sustava biti će definirano nekoliko prednosti koje bi mogle doprinijeti procesu testiranja u budućnosti ili u primjeni na nekim budućim projektima.

S obzirom na specifičnosti drugih platnih sustava razvoj instant platnog sustava značio je veliku promjenu u načinu razmišljanja kao i u dosadašnjem načinu poslovanja. To je svakako od svih članova projektnog tima iziskivalo razvoj i korištenje novih vještina koje su omogućile savladavanje svih izazova koje je sustav stavio pred timove.

5.1. Opće informacije i funkcionalnosti platnog sustava

Sve vrste plaćanja koja se odvijaju među subjektima putem banaka odvijaju se kroz platni promet. Platni promet predstavlja neizostavan dio gospodarstva svake zemlje i zadužen je za

osiguranje sigurne uporabe novca. Plaćanje u platnom prometu odnosi se na obračun putem računa, uplata na račun i isplata s računa. Uspješno i sigurno funkcioniranje platnog prometa iznimno je važno i za središnju banku, instituciju odgovornu za funkcioniranje platnog prometa u zemlji kao i za cjelokupni financijski sustav i sve njegove sudionike.[92]

Platni sustav u Republici Hrvatskoj funkcionira na temelju nekoliko zakona: Zakon o platnom prometu, Zakon o elektroničkom novcu, Zakon o provedbi uredbe Europske unije iz područja platnog prometa, Zakon o konačnosti namire u platnim sustavima i sustavima za namiru financijskih instrumenata, Zakon o deviznom poslovanju.

Hrvatska narodna banka (HNB) ima vrlo važnu ulogu u platnom prometu u Republici Hrvatskoj. Ona je zadužena za osiguranje transparentnog i sigurnog funkcioniranja platnog prometa, zadužena je za nadgledanje i unapređivanje sustava platnog prometa. Dakle, Hrvatska narodna banka predstavlja regulatora platnog prometa u Republici Hrvatskoj.

Platni promet u Republici Hrvatskoj izvršava se preko pet platnih sustava:

- Hrvatski sustav velikih plaćanja (HSVP)
- Nacionalni klirinški sustav (NKS)
- NKSInst
- TARGET2
- EuroNKS.

Hrvatski sustav velikih plaćanja (HSVP) platni je sustav u kojemu se platne transakcije u kunama namiruju u realnom vremenu na bruto načelu, a riječ je uglavnom o relativno velikim iznosima. HSVP je platni sustav u kojemu se u kunama namiruju platne transakcije u svrhu provođenja mjera monetarne politike HNB-a, platne transakcije u svrhu opskrbe banaka gotovim novcem, platne transakcije u svrhu provođenja konačne namire drugih platnih sustava i ostale platne transakcije.[93]

Sustav TARGET2 je platni sustav za namiru platnih transakcija u eurima u realnom vremenu na bruto načelu. Europska središnja banka glavno je nadzorno tijelo sustava TARGET2.

[92] Hrvatska narodna banka. (2021) *O platnom prometu* [online]. Zagreb: HNB. Dostupno na: https://www.hnb.hr/temeljne-funkcije/platni-promet/o-platnom-prometu [17. ožujka 2021.]
[93] Hrvatska narodna banka. (2021) *Platni sustavi* [online]. Zagreb: HNB. Dostupno na: https://www.hnb.hr/temeljne-funkcije/platni-promet/platni-sustavi/hsvp [17. ožujka 2021.]

Razvijen je od strane središnje banke EU-a s ciljem omogućavanja sigurne i efikasne namire platnih transakcija (nacionalnih i prekograničnih) u eurima na RTGS načelu (engl. Real Time Gross Settlement) te olakšavanja provođenja monetarne politike EU-a.[94]

Nacionalni klirinški sustav (NKS) je međubankovni platni sustav za obračun bezgotovinskih platnih transakcija u kunama svih sudionika NKS-a koje glase na relativno male iznose. [95] Rad NKS-a usklađen je sa SEPA shemom za kreditne transfere, a osim kreditnih transfera obrađuje i izravna terećenja u skladu s nacionalnim SDD Core i B2B shemama.

EuroNKS platni je sustav koji obrađuje međubankovne platne transakcije SEPA kreditnih transfera u eurima. Sustav je interoperabilan s drugim klirinškim sustavima za mala plaćanja drugih država članica Europske unije s kojima je povezan posredno preko Hrvatske narodne banke čime ostvaruje punu SEPA dostupnost za sve sudionike EuroNKS-a.[96]

Hrvatska narodna banka u NKS-u i EuroNKS-u ima ulogu regulatora, a Financijska agencija je ta koja definira operativne postupke.

NKSInst je platni sustav za izvršenje platnih transakcija u kunama, između platitelja i primatelja plaćanja, u gotovo realnom vremenu.[97] Sustav je u potpunosti usklađen s pravilima SCTInst platne sheme Europskog platnog vijeća i pravilima HRK SCTInst platne sheme u poslovnom i tehničkom smislu.

Upravitelj instant platnog sustava je Financijska agencija dok HNB sudjeluje u funkciji nadzora platnog sustava te kao posrednik u bankovnom osiguranju likvidnosti u novcu središnje banke.

To je platni sustav čiji sudionici razmjenjuju instant kreditni transfer koji se izvršava u vrlo kratkom roku. Pod terminom vrlo kratki rok podrazumijeva se da se transakcije izvršavaju u gotovo realnom vremenu, u svega nekoliko sekundi, odnosno instantno.

Do sada je prijenos sredstava s računa jedne banke na račun druge banke ovisio o obračunskom danu i obračunskim ciklusima i takav prijenos sredstava mogao je potrajati nekoliko sati pa čak i nekoliko dana ako je nalog zadan vikendom, neradnim danom ili

---

[94] Hrvatska narodna banka. (2021) *Platni sustavi: TARGET2* [online]. Zagreb: HNB. Dostupno na: https://www.hnb.hr/temeljne-funkcije/platni-promet/platni-sustavi/target2 [17. ožujka 2021.]
[95] Fina (2021) *Nacionalni klirinški sustav* [online]. Zagreb: Fina. Dostupno na: https://www.fina.hr/nks [17. ožujka 2021.]
[96] Fina (2021) *EuroNKS* [online]. Zagreb: Fina. Dostupno na: https://www.fina.hr/euronks [17. ožujka 2021.]
[97] Hrvatska narodna banka. (2021) *Platni sustavi: NKSInst* [online]. Zagreb: HNB. Dostupno na: https://www.hnb.hr/temeljne-funkcije/platni-promet/platni-sustavi/nksinst [17. ožujka 2021.]

praznicima. Uvođenjem instant platnog sustava, novac je primatelju dostupan u svega nekoliko sekundi.

Osnovne funkcionalnosti koje instant platni sustav omogućuje su:

- izvršenje plaćanja u gotovo realnom vremenu što podrazumijeva obračun i namirenje transakcija do maksimalno 10 sekundi, s time da je dosadašnje prosječno vrijeme izvršenja transakcija trajalo oko 4 sekunde
- plaćanje je moguće provoditi 24 sata dnevno, 7 dana u tjednu, 365 dana u godini (24/7/365) za razliku od drugih platnih sustava kada se plaćanje može provoditi samo radnim danom
- obračunski dan jednak je kalendarskom što znači da traje od 00:00 do 24:00 sati
- poruke koje sudionici razmjenjuju se elektronički potpisuju kako bi se osigurala autentičnost podataka koji se razmjenjuju
- svaka poruka koje sudionici međusobno razmjenjuju prolazi formalne kontrole kako bi se osiguralo da poruka udovoljava propisanim pravilima
- upravljanje likvidnošću omogućuje sudionicima povećanje odnosno smanjenje raspoloživih novčanih sredstava
- kreiranje i dostavljanje dnevnih i statističkih izvještaja o stanju i prometu obračunskih računa
- sudionicima je omogućen nadzor i praćenje statusa obrade poruka kroz web aplikaciju
- sudionicima se dostavljaju automatska upozorenja o važnim događajima u sustavu
- omogućeno je pohranjivanje podataka u određenim vremenskim intervalima.

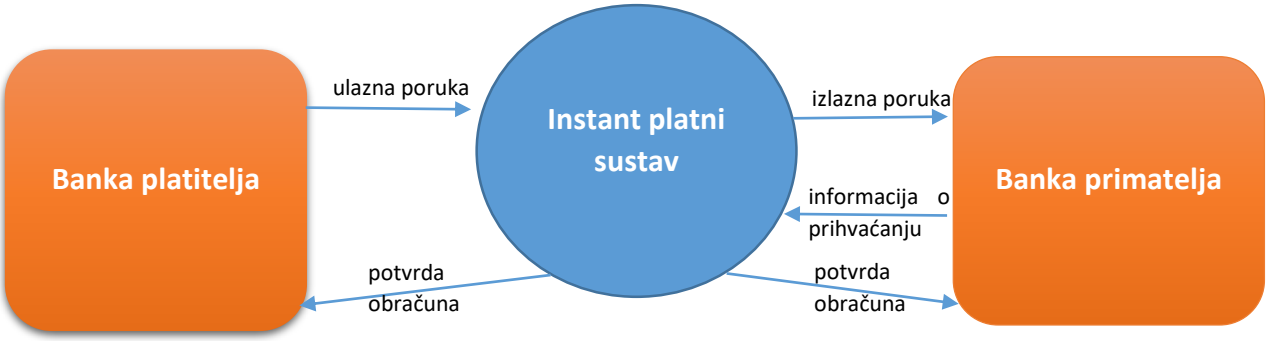
Sudionici instant platnog sustava mogu biti banke i štedne banke, podružnice kreditnih institucija i Hrvatska narodna banka kada djeluje kao pružatelj platnih usluga.

Proces obrade poruke (prikazan na slici 18.) započinje primitkom poruke u sustav koju šalje banka platitelja. Ako je poruka ispravna po poslovnim pravilima sustava i sudionik ima dovoljno raspoloživih sredstava na svom obračunskom računu sustav provodi obračun/namiru platnih transakcija te prosljeđuje transakciju do banke primatelja. Ako sudionik ne raspolaže s dovoljno novčanih sredstava transakcija se odbija. Kada banka primatelja zaprimi transakciju ona šalje povratan odgovor u sustav. Taj odgovor može biti informacija o prihvaćanju ili informacija o odbijanju transakcije. Ako banka odgovara s

informacijom o prihvaćanju, sustav zaprima tu informaciju te izvršava obračun/namiru tako da tereti banku platitelja, a odobrava banku primatelja.

Ako banka primatelja odgovara s informacijom o odbijanju tada sustav zaprima takvu informaciju te uklanja rezervaciju sredstava koju potom vraća banci platitelju koja je izdvojila novčana sredstva za obračun transakcije. U krajnjem slučaju može se dogoditi da banka primatelja ne odgovori na poruku primljenu od banke platitelja te takvu transakciju nakon isteka krajnjeg roka (20 sekundi) za odgovor sustav odbija. Svaka platna transakcija koja je obračunata u instant platnom sustavu smatra se i konačno namirenom.

Slika 18. Tijek obrade poruka u instant platnom sustavu (autorski rad)



5.2. Prednosti i nedostaci ručnih testova u procesu testiranja platnog sustava

Kao što je to već definirano u prethodnim poglavljima ručno testiranje predstavlja postupak otkrivanja grešaka bez pomoći alata, odnosno tester sva testiranja provodi ručno prema unaprijed pripremljenim testnim scenarijima.

Tijekom procesa testiranja platnog sustava ručno su se provodili sljedeći testovi:

- testovi funkcionalnosti
- integracijski testovi
- regresijska testiranja.

Sam proces ručnih testova provodi se tako da tester priprema testne scenarije i testne uzorke za svaku definiranu specifikaciju. Nakon provedenog testiranja upisuju se dobiveni rezultati, a

ako rezultati odstupaju od očekivanih prijavljuju se greške razvojnom timu. Nakon isporuke nove verzije aplikacije testiranje se ponavlja prema koracima testne skripte. Postupak se ponavlja dok rezultati izvođenja nisu u potpunosti u skladu sa specifikacijom ili dok se ne usvoji zahtjev za izmjenom specifikacije. Nakon testiranja svake pojedine specifikacije slijedi integracijsko testiranje čime se povezuje nekoliko specifikacija u cjelinu.

Funkcionalni testovi provođeni su prema specifikacijama i detaljnom dizajnu čiji je osnovni cilj provjera rada svih navedenih funkcionalnosti. Funkcionalne testove su provodili testeri bez pomoćnih alata. Tijekom testa ručno su izrađivani uzorci koji su ustvari xml poruke koje sudionici međusobno izmjenjuju. Zatim se simulira slanje tih poruka u sustav prema testnim koracima pojedinog scenarija. Jedna od prednosti provođenja ručnih funkcionalnih testova su svakako niži početni troškovi kada sustav u prvim fazama i nije bio toliko razvijen, međutim s vremenom kako je sustav postao kompleksniji i kako su zahtjevi postali teži i veći taj trošak je počeo rasti.

Pod tim troškovima podrazumijeva se broj ljudi koji je sudjelovao u testu a koji se s razvojem sustava povećavao, to je dovelo i do povećanja opreme koja je testerima neophodna za rad, povećanja vremena potrebnog za testiranje. Ručno testiranje pokazalo se vrlo pogodno za ah hoc testiranja i vrlo jednostavno za testiranje bilo kakvih izmjena u specifikaciji sustava. U tom slučaju testeru je dovoljan dokument s izmjenama, a zatim vrlo brzo može prilagoditi testne korake i testne uzorke. Međutim takav način rada ostavlja dovoljno prostora za napraviti grešku.

U kasnijim fazama projekta kada sustav postaje kompleksniji veće su šanse da tester previdi situaciji, da preskoči testni korak, ne primijeti grešku. Ljudski je griješiti, testiranje je vrlo zamorno i ponavljajuće te zahtjeva dobro poznavanje rada sustava. Stoga je nerealno očekivati da će sve greške biti otkrivene. Usporedba prednosti i nedostataka ručnih funkcionalnih testova tijekom testirnaja platnog sustava nalazi se u tablici ispod.

Tablica 4. Prikaz prednosti i nedostataka kod ručnog provođenja funkcionalnih testova (autorski rad)

	Prednosti funkcionalnih ručnih testova	Nedostaci funkcionalnih ručnih testova
1.	Manji početni trošak testiranja	S vremenom i ponavljanjem testova trošak testiranja raste
2.	Pogodan za ad hoc testiranja	Zbog mogućih ljudskih grešaka manje je pouzdano te su se često znale potkrasti greške koje nisu otkrivene
3.	Jednostavna promjena testnih koraka u slučaju izmjene specifikacije	Nije pogodno za kompleksne scenarije poput simuliranja slanja velikog broja uzoraka

Integracijski test obuhvaćao je simulaciju rada nekoliko funkcionalnosti tijekom jednog i više obračunskih dana (obrada instant kreditnih transfera, razmjena poruka za upravljanje likvidnošću, slanje izvještaja, prelazak u novi obračunski dan). Iako je pri testiranju korištena aplikacija Load generator koja generira veći broj instant kreditnih transfera kako bi se simulirao stvarni rad sustava, glavni nedostatak ovako provođenog testa je ručna provjera rezultata obrade. Kako se radi o provođenju kompleksnijeg testa, ovakva provjera zahtijevala je više vremena, više uloženog truda, više koncentracije i više testera kako bi bila uspješna. Dodatno kako je riječ o testiranju većeg broja povezanih funkcionalnosti ponekad kada bi se i pronašla greška bilo ju je teško uočiti i lokalizirati. To je dodatno oduzelo vremena za prijavu greške budući da forma za prijavu grešaka zahtjeva čim više preciznih informacija o grešci. Budući da je integracijsko testiranje moglo započeti tek nakon što su implementirani gotovo svi dijelovi sustava, odnosno u kasnijim fazama projekta, to je značilo da je testerima ostalo manje vremena za provođenje testa.

Kao prednosti ovakvog testa svakako treba istaknuti pronalazak grešaka na mjestima gdje se funkcionalnosti povezuju. To se pogotovo pokazalo korisnim kod povezivanja instant platnog sustava sa sustavom za razmjenu poruka za upravljanje likvidnošću (Swift), zatim kod povezivanja sa sustavima banaka koji razmjenjuju poruke s instant platnim sustavom, povezivanje s PKI infrastrukturom. Probleme na koje smo naišli tijekom povezivanja na spomenute sustave bilo bi teško uočiti bez integracijskih testova. Usporedba prednosti i

nedostataka ručnih integracijskih testova tijekom testiranja platnog sustava nalazi se u tablici ispod.

Tablica 5. Prikaz prednosti i nedostataka kod ručnog provođenja integracijskih testova (autorski rad)

	Prednosti integracijskih ručnih testova	Nedostaci integracijskih ručnih testova
1.	Pronalazak grešaka kod povezivanja s drugim sustavima i podsustavima	Ručna provjera rezultata obrade
2.	Osigurali su da različite funkcionalnosti rade povezano i jedinstveno	Poteškoće kod uočavanja i lokalizacije greške zbog kompleksnosti sustava
3.	Veća pokrivenost testa	Bitno skraćeno vrijeme testiranja zbog provođenja testa u kasnim fazama razvoja projekta

Regresijska testiranja provođena su ručno i tijekom svih faza razvoja platnog sustava, počevši od onih najranijih kada je postojalo tek nekoliko funkcionalnosti pa da završnih faza kada je sustav bio gotovo završen. Najvažnija uloga ovog testa tijekom samog testiranja bila je što omogućava kompletan test funkcionalnosti koje su bile mijenjane da li zbog greške ili zbog poslovne potrebe. Ovaj test je omogućio je utvrđivanje radi li izmijenjena funkcionalnost jednako kao i prije uvođenja izmjena. Provođenjem regresijskih testova omogućeno je kontinuirano testiranje što je značajno doprinijelo ukupnoj kvaliteti i stabilnosti platnog sustava. S obzirom na to da su testovi provođeni ručno zahtijevali puno ljudskog truda, vremena i resursa. Svaka izmjena u kodu, pa i ona najmanja, zahtijevala je posebnu pažnju kojoj se morao posvetiti najmanje jedan tester. Samim time taj tester više nije bio raspoloživ za druga testiranja dok nije prošao test cijele funkcionalnosti koja se mijenjala. Za kompleksne funkcionalnosti to je značilo odvajanje puno više vremena i testiranje po složenim testnim skriptama što je u konačnici rezultiralo i probijanjem vremenskih rokova.

Nakon svake nove isporuke testni tim se suočavao s ograničenjima u smislu proračuna, rasporeda i raspoloživih testera. Kako bi prevladao ovakve izazove testni tim je unaprijed pripremao plan testiranja, ažurirao testne uzorke i testne scenarije kako bi odstupanja od dogovorenih vremenskih rokova bila što manja. Usporedba prednosti i nedostataka ručnih regresijskih testova tijekom testiranja platnog sustava nalazi se u tablici ispod.

Tablica 6. Prikaz prednosti i nedostataka kod ručnog provođenja regresijskih testova (autorski rad)

	Prednosti regresijskih ručnih testova	Nedostaci regresijskih ručnih testova
1.	Omogućuje kompletan test funkcionalnosti, veća pokrivenost testa	Svaka izmjena koda zahtjeva novo ponavljanje ručnog testa
2.	Kontinuirano testiranje osiguralo je veću kvalitetu i stabilnost platnog sustava	Veliki utrošak vremena u odnosu na druga testiranja
3.	Testovi su provođeni u svim fazama razvoja sustava	Probijanje vremenskih rokova
4.	Jednostavno ažuriranje testnih scenarija (najčešće u Microsoft Wordu)	Ograničen u smislu raspoloživosti testera, proračuna namijenjenog za testiranje
5.	-	Ponavljajuće i zamorno s većom mogućnosti neprepoznavanja greške

Iako često vrlo zamorno ručno testiranje je činilo presudnu kariku u testiranju platnog sustava. Na sebi svojstven način osiguralo je da ključne komponente sustava rade savršeno povezano te je postavilo dobre temelje za eventualno uvođenje automatiziranih testova. Iako analizom procesa testiranja platnog sustava dolazim do zaključka kako je ipak jedan mail dio scenarija previše kompleksan kako bi se automatizirao, a time ručno testiranje i dalje ostaje bitna komponenta testiranja.

5.3. Prednosti i nedostaci automatiziranih testova u procesu testiranja platnog sustava

Kako je već u nekoliko navrata i zaključeno i u razvoju instant platnog sustava testiranje se pokazalo kao neophodno u osiguranju kvalitete. Provođenjem automatiziranih testova postupak testiranja je pojednostavljen, a testeri su oslobođeni suvišnih zadataka. Kod automatiziranih testova korišteno je nekoliko alata (JUnit, Mockito framework, Checkmarx), a s obzirom na specifičnost sustava dio testova je automatiziran izradom vlastitih programskih alata.

Tijekom procesa testiranja platnog sustava automatizirani su sljedeći testovi:

- jedinični testovi
- testovi performansi
- sigurnosni testovi.

U sigurnosne testove organizacija je uložila puno truda, vremena i sredstava. Sigurnost se pokazala kao jedna od ključnih komponenata. Reputacija organizacije i sustava ovisi o načinu upravljanja sigurnošću i podacima koje sustav sadrži. Sigurnosni testovi obuhvaćali su statičku i dinamičku analizu programskog koda. Statička analiza koda provodila se putem alata Checkmarx. Nakon provedenih testova napravljena je optimizacija programskog koda sukladno sugestijama alata. Tako su otklonjeni sigurnosni problemi unutar programskog koda. Testovi su se provodili nekoliko puta tijekom razvoja sustava, a osim toga test je proveden i prije isporuke platnog sustava u završnu fazu.

Dinamička analiza koda provodila se kroz penetracijske testove u suradnji s ovlaštenim tvrtkama stručnim i osposobljenim za to područje rada. Nakon završene analize testova revidirane su sigurnosne smjernice te je učinjena implementacija smjernica koje nisu bile predviđene. Osim toga napravljen je ispravak pogrešno implementiranih sigurnosnih smjernica. Svakako se kao prednost provođenja penetracijskih automatiziranih testova nad instant platnim sustavom posebno ističe vrlo jednostavan način pronalaska sigurnosnih prijetnji i ranjivosti sustava. Pronalaskom takvih grešaka spriječene su velike štete koje su mogle utjecati osim financijski i na reputaciju organizacije. Samim provođenjem testova organizacija je ispoštovala i uskladila rad sustava s Općom uredom o zaštiti podataka (GDPR) te normom ISO 27001 čime je poseban naglasak stavljen na kontinuirano poboljšanje upravljanja sigurnošću instant platnog sustava.

Upotrebom automatiziranog alata smanjeni su troškovi testiranja jer je test mogao izvoditi jedan tester. Osim utjecaja na financijsku komponentu upotreba alata smanjila je vrijeme potrebno za pronalazak ranjivosti budući da alat za razliku od čovjeka može puno brže proći po kodu te uočiti nepravilnosti. Iako je pronalazak ranjivosti ubrzan kao nedostatak se pokazalo kako je alat potrebno održavati, ispravno ga konfigurirati i na kraju iščitavati rezultate. Nedovoljno samostalan na kraju ipak ovisi o čovjeku i njegovom znanju. Nakon provedenih testova alat je vrlo brzo dao preporuke za poboljšanje koje su sa svakom sljedećom isporukom implementirane u sustav. I u davanju rezultata alat može prikazati lažno pozitivne rezultate stoga se ključnim pokazalo testerovo znanje i iskustvo kako bi bili protumačeni na ispravan način. Usporedba prednosti i nedostataka automatiziranih penetracijskih testova tijekom testiranja platnog sustava nalazi se u tablici 7.

Tablica 7. Prikaz prednosti i nedostataka automatiziranog penetracijskog testiranja (autorski rad)

	Prednosti automatiziranih penetracijskih testova	Nedostaci automatiziranih penetracijskih testova
1.	Manji troškovi, ušteda zbog brže sanacije, skraćuje se vrijeme potrebno za prepoznavanje napada	Održavanje alata za provođenje testa ovisi o čovjeku
2.	Brže izvještavanje uz preporuku poboljšanja	Izvješća daju automatski generirane savjete
3.	Pronalazak vrlo osjetljivih i visokorizičnih propusta	Lažno pozitivni rezultati
4.	Prikupljene su velike količine podataka o mreži	Nije moguće pronaći sve ranjivosti
5.	Usklađenost sa sigurnosnim standardom ISO 27001 i Općom uredbom o zaštiti podataka	-

Kod testova statičke analize koda alat je omogućio vrlo jednostavno i brzo pronalaženje slabosti u kodu. Alatom je skenirana cijela baza koda i kao rezultat dala vrlo preciznu poziciju greške. Statička analiza koda provođena je u svim fazama razvoja sustava što je omogućilo kontinuirano praćenje razvoja sustava ali u konačnici i smanjenje troškova popravljanja. Slično kao i kod alata za dinamičku analizu koda moguća je pojava lažno pozitivnih rezultata te je u tom slučaju potrebna intervencija razvojnog tima koji dobro poznaju kod i mogu procijeniti je li stvarno u pitanju greška. Osim toga alat može pružiti lažan osjećaj sigurnosti. Tablica 8. prikazuje prednosti i nedostatke automatizirane statičke analize koda.

Tablica 8. Prikaz prednosti i nedostataka automatizirane statičke analize koda (autorski rad)

	Prednosti statičke analize koda	Nedostaci statičke analize koda
1.	Brz i jednostavan pronalazak slabosti koda	Ručna provjera rezultata obrade
2.	Skenirana je cijela baza koda	Pojava lažno pozitivnih rezultata
3.	Provodi se u svim fazama razvoja sustava	Pruža osjećaj lažne sigurnosti

Kako su osnovne karakteristike instant platnog sustava obrada transakcija u svega par sekundi, kontinuirana raspoloživost i visoka pouzdanost posebna pažnja posvetila se provođenju performansnih testova. Performansnim testovima testirane su karakteristike

ponašanja sustava te su tako otkrivana uska grla. Testiranjem opterećenja provjerena je izdržljivost sustava pod različitim uvjetima opterećenja. Tako je tijekom testova u sustav poslano po nekoliko stotina transakcija u minuti. U tu svrhu korištena je aplikacija Load generator koja generira veći broj instant transakcija. Transakcije bi potom pristizale u testnu aplikaciju koja je automatski generirala potrebne odgovore banaka. Glavna prednost provođenja ovakvih testova je pronalazak grešaka u funkcionalnosti platnog sustava zatim na mrežnim komponentama, na postavkama servera. Takve greške uočene su tek kada je sustav bio pod velikim opterećenjem i bilo ih je nemoguće otkriti tijekom drugih testova. Ovakvim testovima dobivene su povratne informacije o brzini, točnosti i stabilnosti platnog sustava. Pravovremenim ispravcima grešaka rad je optimiziran, a kapacitet sustava je prilagođen definiranim zahtjevima.

Osim internih performansnih testova provođeni su u nekoliko navrata i eksterni testovi sa sudionicima platnog sustava (banke). Ovakvim testovima dobivene su povratne informacije o brzini prijenosa podataka između banaka i platnog sustava, vremenu odaziva, povratne informacije o komunikacijskim vezama i konfiguracijama, mrežnim postavkama, procesorskim opterećenjima, konekcijama prema bazi i potrošnji memorije. Tablica 9. prikazuje prednosti i nedostatke automatiziranih performansih testova tijekom testiranja platnog sustava.

Tablica 9. Prikaz prednosti i nedostataka automatiziranih performansnih testova (autorski rad)

	Prednosti performansnih testova	Nedostaci performansnih testova
1.	Pronalazak grešaka u funkcionalnostima, mrežnim komponentama, postavkama servera	Testiranje na neprodukcijskoj okolini
2.	Dobivene povratne informacije o brzini, točnosti i stabilnosti sustava	Izmjenom zahtjeva potrebna izmjena alata kojim je testiran sustav
3.	Optimiziran rad sustava	Svaka izmjena znači skuplje održavanje
4.	Uključenje drugih sudionika u test	Testiranje izvan radnog vremena
5.	-	Djelomična ručna provjera rezultata i ručno pisanje izvještaja testiranja

S obzirom na to da je test performansi automatiziran svaka izmjena u zahtjevima značila je i izmjenu u alatu što je dovelo do povećanja troškova razvoja sustava. Ako na sustav gledamo

dugoročno to bi značilo i kasnije skuplje održavanje. Dodatno se pokazalo kao nepraktično, iz više razloga, nemogućnost provođenja testova na produkcijskoj okolini. Testna okolina je manjeg kapaciteta u svim segmentima stoga se povremeno nailazilo na neplanirane probleme.

Zahvaljujući dobro osmišljenim i provedenim performansnim testovima cijeli sustav je izgrađen sa skalabilnim i redundantnim komponentama te je u potpunosti implementiran prema svim očekivanjima, definiranim zahtjevima i specifikacijama.

Kao sigurnosni i performansni i jedinični testovi su automatizirani te su ih pisali i izvodili članovi razvojnog tima. Testovi su organizirani po programskim paketima. Svako izvršavanje testova uzrokuje slanje e-maila u kojem se nalazi popis i broj uspješno provedenih, preskočenih i palih testova te njihovo trajanje. Jedinično testiranje je pružilo niz prednosti, a značajnija je svakako da su greške u kodu otkrivene u vrlo ranim fazama razvoja sustava. Osim pronađenih grešaka u kodu ovim testovima pronađene su i nelogičnosti u samim specifikacijama sustava. Na ovakav način smanjeni su troškovi testiranja jer su se greške identificirane i ispravljene u ranim fazama razvoja. Međutim, jediničnim testovima se mogu testirati samo funkcionalnosti odnosno nisu pogodni za testiranje više povezanih komponenti. Osim toga svaka promjena u kodu zahtjeva ponovo testiranje. Kao i kod drugih vrsta testova niti jediničnim testovima ne može se u potpunosti zagarantirati odsutstvo grešaka. Vrlo često se potkradaju greške bilo zbog promjena u kodu kada se preskoči test jedinice ili se jednostavno pojedina jedinica ne može testirati jediničnim testovima. Tablica 10. prikazuje prednosti i nedostatke automatiziranih jediničnih testova tijekom testiranja platnog sustava.

Tablica 10. Prikaz prednosti i nedostataka automatiziranih jediničnih testova (autorski rad)

	Prednosti jediničnih testova	**Nedostaci jediničnih testova**
1.	Otkrivanje greška u ranim fazama razvoja sustava	Nisu pogodni za test više povezanih komponenti sustava
2.	Pronalazak nelogičnosti u specifikacijama	Svaka promjena koda zahtjeva novo testiranje
3.	Smanjenje troškova testiranja zbog otkrivanja grešaka u ranim fazama razvoja sustava	Ne jamči odsutstvo grešaka

5.4. Preporuke za poboljšanje testiranja platnog sustava

Prethodno provedeno istraživanje nad prednostima i nedostacima ručnih i automatiziranih testova u platnom sustavu pokazalo je kako niti jedna vrsta, niti ručna niti automatizirana, nije isključiva.

I kod ručnog i automatiziranog testiranja se pokazalo kako su pojedine komponente kritične i provlače se kroz obje vrste testiranja u vidu prednosti ili nedostataka:

- vrijeme
- uloženi napor/trud
- resursi
- troškovi.

Kod ručno provođenih testova vrijeme predstavlja gotovo ključni nedostatak u procesu testiranja. Sve tri vrste testiranja koje su provođene ručno (funkcionalno, regresijsko i integracijsko) su ponavljane kroz cijeli proces razvoja sustava. Pokazalo se kako je ponavljanje neophodno ako je bilo promjena u kodu ili je došlo do promjena u specifikaciji, međutim svaka ta promjena iziskivala je duplu potrošnju vremena. Kada se uz vrijeme uključi i preostale tri komponente tada je jasno vidljivo kako je za provedbu gore navedenih ručnih testova prije svega potrebna vrlo dobra organizacija testera, dovoljno novčanih sredstava i puno truda kako bi se test uspio provesti. Ovdje se kao rješenje sama od sebe nameće automatizacija ručno provođenih testova.

Zbog specifičnosti sustava smatram automatizaciju regresijskog testiranja kao najbolju opciju. U tom slučaju automatiziran test bi bio idealan za izvođenje iterativnih i paralelnih testova te bi se tako skratilo vrijeme testiranja. Osim vremena, broj raspoloživih testera bi se povećao te bi više njih bilo na raspolaganju za druge zadatke. Na ovaj način testeri bi svoje znanje i trud mogli usmjeriti na nedovršene zadatke ili druge testove koje čekaju na provođenje.

S obzirom na karakteristike i osjetljivost sustava kao programsko rješenje idealno bi bilo razvijanje internog alata koji bi testerima omogućio samostalno oblikovanje automatiziranog testa te vrlo jednostavnu prilagodbu koraka testnog scenarija u slučaju izmjena u kodu ili specifikacijama. Očekivano je kako bi trošak razvoja takvog alata u početku bio veći, međutim nakon što se razvije alat ušteda i vremena i ljudskih resursa je neupitna. Osim toga, uz manje

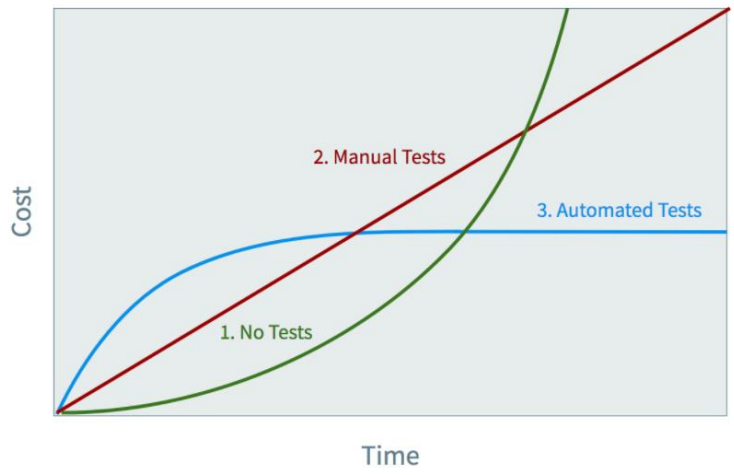
prilagodbe alat bi se mogao prilagoditi za upotrebu i drugim odjelima koji također provode ručne testove.

Početni troškovi manifestirali bi se u vidu:

- troškova razvoja softvera
- troškova nabave hardvera
- troškova obuke ljudi za korištenje softvera
- isplata plaća
- troškova održavanja.

U usporedbi s ručnim testiranjem vidljivo je da su početni troškovi razvoja automatiziranog testa veći, ali isto tako ti troškovi se s vremenom smanjuju. Automatizacija predstavlja dugoročni proces čije se blagodati ne vide odmah. Međutim implementacija alata dugoročno gledajući vraća uloženo vrijeme i novac ponajprije iz razloga jer se greške mogu otkriti puno ranije, a samim time mogu biti i ranije ispravljene što je korak bliže isporuci kvalitetnog softvera. Osim toga za razliku od testera koji dnevno može utrošiti do 8 sati na testiranje, implementacijom alata to vrijeme bi se moglo povećati i na 16 sati dnevno, smanjujući pritom prosječne troškove testera. Dodatno, na vremenu bi se moglo uštedjeti i paralelnim pokretanjem nekoliko testova čime bi se test dodatno ubrzao. Slika 19. prikazuje ponašanje troška kroz vrijeme kod manualnih i automatziranih testova te njihov rast kada se testovi uopće ne provode.

Slika 19. Troškovi testiranja kroz vrijeme



Izvor: Huges, K. (2021). *The Importance of Software Testing* [online]. Karl Huges. Dostupno na: https://www.karllhughes.com/posts/testing-matters [08. travnja 2021.]

Prema tome, lako je zaključiti kako su temeljni ciljevi implementacije alata kojim bi se provodila automatizirana regresijska testiranja:

- skraćeno vrijeme testiranja
- smanjenje broja kritičnih grešaka koje uzrokuju prekid rada sustava
- osiguranje kvalitetnog i sigurnog sustava
- smanjenje troškova testiranja
- veća produktivnost.

Prijelaz s ručnog na automatizirano testiranje velik je zadatak. Kao i kod svakog projekta tijekom implementacije se mogu javiti zamke koje uz dobru organizaciju mogu biti savladane. Ovdje se nikako ne smije zanemariti uloga ručnog testiranja. Iako je zahtjevno, skupo i troši mnoge resurse, fokusirano ručno testiranje može doprinijeti i uvelike olakšati razvoj alata. No, da bi ručno testiranje bila dobra zamjena automatiziranom testiranju, svakako je nužno pripremiti primjerene testne scenarije, testne uzorke i usredotočiti se na konkretne probleme kako bi razvoj alata bio što jednostavniji ali u konačnici i isplativiji.

6. ZAKLJUČAK

Čak i razvoj najmanjeg softvera zahtijeva pravilno upravljanje. Pravilno upravljanje zapravo podrazumijeva poznavanje i primjenu različitih metodologija i okvira za razvoj softvera. Poznavanje raznih metodologija pomaže u standardizaciji postupaka jasno otkrivajući što se u pojedinoj fazi projekta događa sa softverom. Na svom putu do finalnog proizvoda, softver prolazi kroz različite faze i aktivnosti koje će mu u konačnici omogućiti visoku kvalitetu.

Jedna od faza životnog ciklusa razvoja softvera jest testiranje softvera koje je predmet istraživanja ovog rada. Testiranje se ističe kao jedan od ključnih, ali i najkritičnijih procesa u životnom ciklusu iz razloga jer organizacijama omogućuje sveobuhvatnu procjenu softvera. Ako se ne provede testiranje softvera prije njegove isporuke krajnjem korisniku, programske greške mogu negativno utjecati na troškove razvoja ali i na reputaciju organizacije.

U radu su definirane različite metode, tehnike i tipovi testiranja. Niti jednom podjelom ne može se izdvojiti izričito jedna vrsta testiranja kao bitnija i značajnija. Svaka definirana metoda, tehnika ili vrsta ima svoje specifičnosti koje se mogu primijeniti u različitim vrstama projekata kao i u različitim fazama životnog ciklusa testiranja softvera. Kroz istraživanje testiranja u razvoju platnog sustava na kraju samog rada, poseban naglasak stavio se na definiranje karakterističnosti, prednosti i nedostataka ručnih i automatiziranih testova. Iako su ručni testovi spori, ponavljajući, zamorni te oduzimaju puno vremena zanemariti njihovu važnost bila bi pogreška. S druge strane automatizirani testovi imaju reputaciju brzine, učinkovitosti i ekonomičnosti stoga njihova popularnost raste velikom brzinom. Istraživanjem procesa testiranja u platnom sustavu ustanovljeno je kako je ručno testiranje bolje i prigodnije za područja u kojima automatizacija ne može ponoviti čovjekovu inovativnost, kreativnost i cjelovito razumijevanje. Osim toga predstavlja savršenu podlogu za pripremu automatiziranih testova. Automatizirani testovi pokazali su se kao najbolja opcija za rutinska i ponavljajuća testiranja koja ako se provode ručno mogu biti vrlo dugotrajna i zamorna.

Cilj ovog rada bio je ukazati na važnost procesa testiranja u razvoju softvera te naglasiti kako se njegova uloga nikako ne može umanjiti. Testiranje ublažava potrebu za stalnom nadogradnjom i popravcima, jer testeri prepoznaju pogreške i prije nego što se problemi pojave. Testiranje čini temeljni postupak stvaranja pouzdanih, kvalitetnih i upotrebljivih programskih rješenja.

POPIS LITERATURE

1. Acharya, S., Pandya, V. (2012) Bridge between Black Box and White Box–Gray Box Testing Technique. *International Journal of Electronics and Computer Science Engineering* [online], *2*(1). Dostupno na: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.303.4479&rep=rep1&type=pdf [05. ožujka 2021.]
2. Agility in Mind. (2020). *What is eXtreme Programming (XP)?: XP Values* [online]. Agility in Mind. Dostupno na: https://agility.im/frequent-agile-question/what-is-extreme-programming/ [26. siječnja 2021.]
3. Altexsoft. (2021). *Quality Assurance, Quality Control and Testing — the Basics of Software Quality Management* [online]. Altexsoft. Dostupno na: https://www.altexsoft.com/whitepapers/quality-assurance-quality-control-and-testing-the-basics-of-software-quality-management/ [22.veljače 2021.]
4. Ammann, P., Offutt, P. (2017) *Introduction to software testing*. Cambridgeshire: Cambridge University Press Solutions.
5. American Society for Quality (2021). *Quality Assurance & Quality Control* [online]. USA: ASQ. Dostupno na: https://asq.org/customer-services/contact-asq [18. veljače 2021.]
6. Apoorva, M., Deepty, D. (2013) A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios: Intraduction. *International Journal of Advance Research in Computer Science and Management Studies* [online], 1(5). Dostupno na: https://www.researchgate.net/publication/289526047_A_Comparative_Study_of_Different_Software_Development_Life_Cycle_Models_in_Different_Scenarios [08. veljače 2021.]
7. Balaji, S., Murugaiyan Sundararajan, M. (2012). Waterfall vs V-Model vs Agile: A Comparative Study on SLDC*, International Journal of Information Technology and Business Management* [online]*,* 2(1). Dostupno na: https://mediaweb.saintleo.edu/Courses/COM430/M2Readings/WATEERFALLVs%20

V-MODEL%20Vs%20AGILE%20A%20COMPARATIVE%20STUDY%20ON%20SDLC.pdf [16. siječnja 2021.]

8. Compliencehelp Consulting (2020). *What are Management System Standards?* [online]. Dostupno na: Dostupno na: https://quality-assurance.com.au/what-are-management-system-standards/ [28. veljače 2021.]
9. Čubranić, D., Kaluža, M., Novak, J. (2013) *Standardne metode u funkciji razvoja softvera u Republici Hrvatskoj*. Rijeka: Zbornik Veleučilišta u Rijeci, Rijeka.
10. DATAROB. (2019). *Software Development Life Cycle (SDLC) – Basics, Stages, Models: Stages* [online], Estonia, Datarob. Dostupno na: https://datarob.com/essentials-software-development-life-cycle/ [08.veljače 2021.]
11. Denaro, G., Polini, A. Emmerich, W. (2004) Early Performance testing of distributed software applications. *Association for Computing Machinery* [online], 29(1). Dostpuno na: https://dl.acm.org/doi/abs/10.1145/974044.974059 [27. veljače 2021.]
12. Diceus (2020) *Top 10 Software Quality Metrics That Matter* [online]. Diceus. Dostupno na: https://diceus.com/top-7-software-quality-metrics-matter/ [14. ožujka 2021.]
13. Eby, K. (2017). *The Ultimate Guide to Understanding and Using a System Development Life Cycle* [online]. Smartsheet. Dostupno na: https://www.smartsheet.com/system-development-life-cycle-guide [08. veljače 2021.]
14. Educba (2020). Software Quality Assurance [online]. Educba. Dostupno na: https://www.educba.com/software-quality-assurance/ [28. veljače 2021.]
15. Everett, G. D., McLeod, R. (2007) *Software Testing:Testing Acros The Entire Software Development Lyfe Cycle*. IEEE
16. Farooq, S. U., Quadri, S. M. K., Ahmad, N. (2011) Software measurements and metrics: Role in effective software testing. *International Journal of Engineering Science and Technology* [online], *3*(1). Dostupno na: https://www.researchgate.net/publication/50392202_SOFTWARE_MEASUREMENTS_AND_METRICS_ROLE_IN_EFFECTIVE_SOFTWARE_TESTING [13. ožujka 2021.]
17. Fina (2020). *Financijska agencija dobila odobrenje za rad novog platnog sustava NKSInst* [online]. Zagreb: Fina. Dostupno na: https://www.fina.hr/-/financijska-agencija-dobila-odobrenje-za-rad-novog-platnog-sustava-nksinst [13. prosinca 2020.]

18. Fina (2021) *EuroNKS* [online]. Zagreb: Fina. Dostupno na: https://www.fina.hr/euronks [17. ožujka 2021.]
19. Fina (2021) *Nacionalni klirinški sustav* [online]. Zagreb: Fina. Dostupno na: https://www.fina.hr/nks [17. ožujka 2021.]
20. Financier Worldwide (2019). *Self-healing enterprise: How is AI changing the value of the enterprise?* [online]. Financier Worldwide. Dostupno na: https://www.financierworldwide.com/self-healing-enterprise-how-is-ai-changing-the-value-profile-of-the-enterprise#.X-xOBGhKhPb [26. prosinca 2020.]
21. GeeksforGeeks (2020). *Defect Management Process* [online]. Noida: GeeksforGeeks. Dostupno na: https://www.geeksforgeeks.org/defect-management-process/ [10. ožujka 2021.]
22. GeeksforGeeks (2019). *Performance Testing: Software Testing* [online]. Noida: GeeksforGeeks. Dostupno na: https://www.geeksforgeeks.org/performance-testing-software-testing/ [08. ožujka 2021.]
23. Gelperin, D., Hetzel, B. (1988) The Growth of Software Testing. *Communications of ACM* [online], 31(6). Dostupno na: http://understandingrequirements.com/resources/2.2%20%20Growth%20of%20SW%20Testing.pdf [08. veljače 2021.]
24. Guru99 (2021). *Automation Testin Tutorial: What is Automated Testing* [online]. Guru99. Dostupno na: https://www.guru99.com/automation-testing.html [06. ožujka 2021.]
25. Guru99 (2021). *What is Non Functional Testing? Types with Example* [online]. Guru99. Dostupno na: https://www.guru99.com/non-functional-testing.html [06. ožujka 2021.]
26. Guru99 (2021). *What is software testing life cycle (STLC)?* [online]. Guru99. Dostupno na: https://www.guru99.com/software-testing-life-cycle.html [10. veljače 2021.]
27. Guru99 (2021). *What is Non Functional Testing? Types with Example* [online]. Guru99. Dostupno na: https://www.guru99.com/non-functional-testing.html [06. ožujka 2021.]

28. Hrvatska narodna banka. (2021). *Platni sustavi* [online]. Zagreb: HNB. Dostupno na: https://www.hnb.hr/temeljne-funkcije/platni-promet/platni-sustavi/hsvp [17. ožujka 2021.]
29. Hrvatska narodna banka. (2021). *Platni sustavi: TARGET2* [online]. Zagreb: HNB. Dostupno na: https://www.hnb.hr/temeljne-funkcije/platni-promet/platni-sustavi/target2 [17. ožujka 2021.]
30. Hrvatska narodna banka. (2021). *Platni sustavi: NKSInst* [online]. Zagreb: HNB. Dostupno na: https://www.hnb.hr/temeljne-funkcije/platni-promet/platni-sustavi/nksinst [17. ožujka 2021.]
31. Huges, K. (2021). *The Importance of Software Testing* [online]. Karl Huges. Dostupno na: https://www.karllhughes.com/posts/testing-matters [08. travnja 2021.]
32. Jaaksola M. (2018) *Software testing failures through the history and how to prepare for them.* Magistarski rad. Helsinki: Metropolia University of Applied Sciences.
33. Jyotsna, Varshney M., Garg S., Rajpoot A. K. (2017). Automated Testing: An Edge Over Manual Software Testing. *International Journal of Trend in Scientific Research and Development (ijtsrd)* [online], 1(4). Dostupno na: https://www.ijtsrd.com/papers/ijtsrd2232.pdf [12. prosinca 2020.]
34. Kalwan, A. (2020). *What is Software Testing Metrics and What are the Types?* [online]. Edureka!. Dostupno na: https://www.edureka.co/blog/software-testing-metrics/ [13. ožujka 2021.]
35. Khan, M. E., Khan, F. (2012) A comparative study of white box, black box and grey box testing techniques [online]. *International Journal of Advanced Computer Science and Applications (IJACSA), 3*(6). Dostupno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.685.1887&rep=rep1&type=pdf#page=22 [05. ožujka 2021.]
36. Kumar, D., Mishra, K. K. (2016). *The Impact of Test Automation on Software's Cost, Quality and Time to Market. Procedia Computer Science* [online] *79*, Dostupno na: https://core.ac.uk/download/pdf/82601631.pdf [01.ožujka 2021.]
37. Laporte, C.Y., April, A. (2018) Software Quality Assurance. USA: John Wiley & Sons.

38. Liviu, M. (2014) Comparative study on software development methodologies. *Database System Journal* [online], vol. V, no. 3/2014. Dostupno na: https://dbjournal.ro/archive/17/17_4.pdf [16. siječnja 2021.]
39. Lučić, M. (2019) *Prakse testiranja programskih proizvoda.* Završni rad. Varaždin: Fakultet organizacije i informatike.
40. Matković, P., Tumbas, P. (2010) A Comparative Overwiev of the Evolution of Software Development Models*, Journal of Industrial Engineering and Management* [online], 2(4). Dostupno na: https://www.researchgate.net/publication/267711880_A_Comparative_Overview_of_the_Evolution_of_Software_Development_Models [16.siječnja 2021.]
41. McKinley, N. (2020). *Test Automation vs Automated Testing: Key Differences & Pros and Cons* [online]. Software Test Professionals. Dostupno na: https://www.softwaretestpro.com/test-automation-vs-automated-testing-key-differences-pros-and-cons/ [06. ožujka 2021.]
42. Mekni, M. , Buddhavarapu, G. , Chinthapatla, S. and Gangula, M. (2018) Software Architectural Design in Agile Environments. *Journal of Computer and Communications* [online], 6(1). Dostupno na: https://www.scirp.org/(S(czeh2tfqyw2orz553k1w0r45))/journal/paperinformation.aspx?paperid=81436 [20. siječnja 2021.]
43. Miller, R., Collins, C. T. (2001). Acceptance testing. *Proc. XPUniverse* [online]. Dostupno na: http://www.dsc.ufcg.edu.br/~jacques/cursos/map/recursos/Testing05.pdf [03. ožujka 2021.]
44. Muslija A., Enoiu EP. (2018) On the correlation between testing effort and software complexity metric. *PeerJ Preprints* [online]. Dostupno na: https://peerj.com/preprints/27312v1/ [15. rujna 2020.]
45. Myers, G.J. (2004) The Art of Software Testing. 2nd. ed. Hoboken, New Jersey: John Wiley & Sons, Inc.
46. Naznin, T. (2018). *Manual testing process* [online], Medium. Dostupno na: https://medium.com/oceanize-geeks/manual-testing-process-340173d40141 [06. ožujka 2021.]

47. Nidhra, S., Dondeti, J. (2012) Black box and white box testing techniques-a literature review [online]. *International Journal of Embedded Systems and Applications (IJESA)*, *2*(2). Dostupno na: https://www.researchgate.net/publication/276198111_Black_Box_and_White_Box_Testing_Techniques_-_A_Literature_Review [04. ožujka 2021.]
48. Nindel-Edwards, J., Steinke, G. (2006) A Full Life Cycle Defect Process Model That Supports Defect Tracking, Software Product Cycles, And Test Iterations [online]. *ResearchGate.* Dostupno na: https://www.researchgate.net/publication/228739283_A_Full_Life_Cycle_Defect_Process_Model_That_Supports_Defect_Tracking_Software_Product_Cycles_And_Test_Iterations [27. ožujka 2021.]
49. Nirpal, P. B., Kale, K. V. (2011) A brief overview of software testing metrics. *International Journal on Computer Science and Engineering* [online], *3*(1). Dostupno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.5632&rep=rep1&type=pdf [12. ožujka 2021.]
50. NIST. (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing* [online], Gaithersburg, National Institute of Standards and Technology. Dostupno na: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf [09.veljače 2021.]
51. Noor, R., Khan, M. F. (2014) Defect management in agile software development. *International Journal of Modern Education and Computer Science* [online], *6*(3). Dostupno na: http://www.mecs-press.org.ua/ijmecs/ijmecs-v6-n3/IJMECS-V6-N3-7.pdf [10. ožujka 2021.]
52. Organize Agile. (2020). Half of companies applying Agile metodologies & practicws [online], Europe. Consultancy.eu. Dostupno na https://www.consultancy.eu/news/4153/half-of-companies-applying-agile-methodologies-practices [08. veljače 2021.]
53. Perfecto (2019). *Manual Testing vs. Automated Testing* [online]. Perfecto. Dostupno na: https://www.perfecto.io/blog/automated-testing-vs-manual-testing-vs-continuous-testing

54. Poppendieck, M. i Poppendieck, T. (2003) *Lean Software Development: An Agile toolkit* : *Guided Tour*. Addison-Wesley Longman Publishing Co., Inc., USA.
55. Potter, B., McGraw, G. (2004) Software security testing [online]. *IEEE Security & Privacy*, *2*(5). Dostupno na: https://www.cs.purdue.edu/homes/xyzhang/fall07/Papers/sw-test.pdf [09. ožujka 2021.]
56. Poudel, R. (2019) Network Traffic Visualization with Scapy and ELK Stack*: Extreme Programming Methodology (Selected Methodology)* [online]. ResearchGate. Dostupno na: https://www.researchgate.net/publication/337304617_Network_Traffic_Visualization_with_Scapy_and_ELK_Stack [21. siječnja 2021.]
57. Rexhepi, B., Rexhepi, A. (2018*)* Software testing tehniques and principles. *Knowledge International Journal* [online], 28(4). Dostupno na: https://ikm.mk/ojs/index.php/KIJ/article/view/232/732 [30. studeni 2020.]
58. Rizwan Jameel Qureshi M., Ikram J.S. (2015) Proposal of Enhanced Extreme Programming Model: Introduction. *I.J. Information Engineering and Electronic Business* [online], 7(1). Dostupno na: http://www.mecs-press.org/ijieeb/ijieeb-v7-n1/IJIEEB-V7-N1-5.pdf [26. siječnja 2021.]
59. Rubey, J.R., Brewer, A. (1991). *Software quality assurance standards-a comparison and an integration* [online]. San Diego: IEEE Computer Society. Dostupno na: https://www.computer.org/csdl/proceedings-article/sesaw/1991/00151534/12OmNyv7mvm [23. veljače 2021.]
60. Sauce Labs (2018). *Testing trends for 2018* [online]. Sauce Labs. Dostupno na: https://saucelabs.com/resources/white-papers/testing-trends-for-2018 [15. siječnja 2021.]
61. SEPA (2020). *SEPA instant kreditni transfer* [online]. SEPA. Dostupno na: http://www.sepa.hr/upute-za-korisnike/sepa-instant-kreditni-transfer/ [15. siječnja 2021.]
62. Schwaber, K. (2004) *Agile project management with Scrum*. Washington: Microsoft Press.
63. Schwaber, K. (2021). *What is Scrum?* [online]. Scrum.org. Dostupno na: https://www.scrum.org/resources/what-is-scrum [20. siječnja 2021.]

64. Sharma, R. M. (2014). Quantitative analysis of automation and manual testing. *International journal of engineering and innovative technology [online]*, *4*(1). Dostupno na: https://www.imvcportal.com.au/uploads/study_material/1504593504IJEIT1412201407$%23@_46.pdf [06. ožujka 2021.]
65. Sheikh, U. F. (2011) Software Measurements and Metric: Role in effective software testing. *International Journal of Engineering Science and Technology* (IJEST) [online], 3(1). Dostupno na: http://www.ijest.info/docs/IJEST11-03-01-141.pdf [30. studeni 2020.]
66. Sherrill, C. (2017) *On The Sholders of Giants: A Brief History of SDLC Models* [online]. Business Analyst Coach. Dostupno na: https://businessanalystcoach.blog/2017/12/15/on-the-shoulders-of-giants-a-brief-history-of-sdlc-models/ [15. veljače 2021.]
67. Shetty V. (2020) Software Testing- An Important Phase of Software Development Cycle. *International Journal for Research in Applied Science and Engineering Technology* [online], 8(7). Dostupno na: https://www.ijraset.com/fileserve.php?FID=30091 [30. studeni 2020.]
68. Software Testing Fundamentals. (2020). *Dynamic Testing* [online]. Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/dynamic-testing/ [04. ožujka 2021.]
69. Software Testing Fundamentals. (2020). *Defect Severity* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/defect-severity/ [08.ožujka 2021.]
70. Software Testing Fundamentals (2020). *Test definition* [online]. Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/test-definition/ [20. siječnja 2021.]
71. Software Testing Fundamentals (2020). *Functional vs Non functional Testing* [online]. Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/functional-testing-vs-non-functional-testing/ [07. ožujka 2021.]

72. Software Testing Fundamentals. (2020). *Gray Box Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/gray-box-testing/ [04. ožujka 2021.]
73. Software Testing Fundamentals. (2020). *White Box Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/white-box-testing/ [04. ožujka 2021.]
74. Software Testing Fundamentals. (2020). *Black Box Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/black-box-testing/ [04. ožujka 2021.]
75. Software Testing Fundamentals. (2020). *Unit Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/unit-testing/ [02. ožujka 2021.]
76. Software Testing Fundamentals (2020). *Software Quality* [online]. Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/test-definition/ [20. siječnja 2021.]
77. Software Testing Fundamentals (2020). *Performance Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/performance-testing/ [08.ožujka 2021.]
78. Software Testing Fundamentals (2020). *Static Testing* [online], Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/static-testing/ [03. ožujka 2021.]
79. Software Testing Fundamentals (2020). *What is Defect/Bug LifeCycle in Software Testing?* [online]. Software Testing Fundamentals. Dostupno na: https://softwaretestingfundamentals.com/test-definition/ [20. siječnja 2021.]
80. Software Testing Help (2021). *Software Developmet And Testing Methodologies (With Pros And Cons)* [online]. Software Testing Help. Dostupno na: https://www.softwaretestinghelp.com/software-development-testing-methodologies/ [01. ožujka 2021.]
81. Tutorialspoint (2021). *Software Mesurement Metrics* [online]. Tutorialspoint. Dostupno na:

https://www.tutorialspoint.com/software_quality_management/software_quality_measurement_metrics.htm [12. ožujka 2021.]

82. Tutorialspoint (2021). *Standards and Certificates* [online]. Tutorialspoint. Dostupno na:https://www.tutorialspoint.com/software_quality_management/software_quality_management_standards_and_certificates.htm [28. veljače 2021.]
83. SeaLights (2021). *What Are Test Metrics?* [online]. SeaLights. Dostupno na: https://www.sealights.io/agile-testing/test-metrics/ [02. ožujka 2021.]
84. Spillner, A., Linz, T., Schafer H. (2014) *Software testing foundations*. Santa Barbara: Rocky Nook Inc.
85. Spremić, M. (2017) *Digitalna transformacija poslovanja*. Zagreb: Sveučilište u Zagrebu, Ekonomski Fakultet.
86. Spemić, M. (2017) *Sigurnost i revizija informacijskih sustava u okruženju digitalne ekonomije*. Zagreb: Sveučilište u Zagrebu, Ekonomski fakultet.
87. Svitis, C. (2013) *Lean Software Development Theory validation in terms of cost-reduction and quality-improvement.* Bachelor of Science Thesis. University of Gothenburg.
88. Test Automation Resources (2018). *5 Software Testing Methods* [online]. Test Automation Resources. Dostupno na: https://testautomationresources.com/software-testing-basics/software-testing-methods/ [21. siječnja 2021.]
89. Testim (2021). *Automated Testiing or Test Automation? You Need Both* [online]. Testim. Dostupno na: https://www.testim.io/blog/automated-testing-vs-test-automation/ [06. ožujka 2021.]
90. Testim (2019). *Test Automation vs Manual Testing: Picking the Right Balance* [online]. Testim. Dostupno na: https://www.testim.io/blog/test-automation-vs-manual-testing/ [06. ožujka 2021.]
91. Testsigma (2021). *A Detailed analysis on Advantages, Disadvantages, Challenges and Risks of Regression Testing* [online]. Testsigma. Dostupno na https://testsigma.com/regression-testing/advantages-of-regression-testing [07. ožujka 2021.]
92. Tian, J. (2005) *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. Hoboken, New Jersey: John Wiley & Sons, Inc..

93. Try QA (2021). *Dynamic Testing Technique* [online], Try QA. Dostupno na: http://tryqa.com/what-is-dynamic-testing-technique/ [04. ožujka 2021.]
94. Try QA (2021). *What is a Defect Life Cycle or Bug lifecycle in software testing?* [online], Try QA. Dostupno na: http://tryqa.com/what-is-a-defect-life-cycle/ [12. ožujka 2021.]
95. Tutorialspoint (2021). *Dynamic Testing* [online], Tutorialspoint. Dostupno na: https://www.tutorialspoint.com/software_testing_dictionary/dynamic_testing.htm [04. ožujka 2021.]
96. Umar, M. A. (2020) *Comprehensive study of software testing: Categories, levels, techniques, and types* [online]. Jilin, China: TechRxiv by IEEE. Dostupno na: https://www.techrxiv.org/articles/preprint/A_Study_of_Software_Testing_Categories_Levels_Techniques_and_Types/12578714 [03. prosinca 2020.]
97. Vasylyna, N. (2021). Difference Between QA and Testing [online]. QATestLab Blog. Dostupno na: https://blog.qatestlab.com/2011/04/07/what-is-the-difference-between-qa-and-testing/ [23. veljače 2021.]
98. Vasylyna, N. (2020). *7 Types of Security Testing* [online]. QA TestLab Blog. Dostupno na: https://blog.qatestlab.com/2020/09/07/security-testing-types/ [08.ožujka 2021.]
99. Wong, W. E., Horgan, J. R., London, S., Agrawal, H. (1997) A study of effective regression testing in practice. *Proceedings The Eighth International Symposium On Software Reliability Engineering* [online]. Dostupno na: https://www.semanticscholar.org/paper/A-study-of-effective-regression-testing-in-practice-Wong-Horgan/141d95585958b2294069f0eb424bc31471fcb4d3 [07. ožujka 2021.]
100. Young, D. (2014). Software Development Metodologies, *ResearchGate* [online], Dostupno na: https://www.researchgate.net/publication/255710396_Software_Development_Methodologies [16. siječnja 2021.]
101. Zola, A. (2020). What's the role of security testing in software development?. Information Age. Dostupno na: https://www.information-age.com/whats-role-security-testing-software-development-123487978/ [09.ožujka 2021.]
102. Živković, M. (2018) *Testiranje softvera*. Prvo izdanje. Beograd: Univerzitet Singidunum.

POPIS SLIKA

POPIS TABLICA

ŽIVOTOPIS KANDIDATA

Osobni podaci	
Prezime / Ime	**Mamić Andreja**
Adresa	Posedarska 33Z/19, 10 020, Zagreb, Hrvatska
Broj mobilnog telefona	+385 91 210 0307
E-mail	papic.andreja1@gmail.com
Državljanstvo	Hrvatsko
Datum rođenja	13.05.1986.
Radno iskustvo	2017. – Financijska agencija – viši specijalist sistem analitičar 2015. – 2017. Financijska agencija – viši specijalist za podršku IT sustavu 2010. - 2015. Osnovna škola Tužno, Medicinska škola Varaždin, Srednja Strukovna škola Varaždin – profesor računalstva i informatike
Obrazovanje i osposobljavanje	2020. Ekonomski fakultet u Zagrebu, poslijediplomski specijalistički studij – smjer "Informatički menadžment" 2010. Učiteljski fakultet Zagreb, Odsjek u Čakovcu pedagoško psihološka izobrazba 2004. Fakultet organizacije i informatike, Varaždin, - smjer "Informacijski sustavi" 2000. Medicinska škola, Varaždin