TESTIRANJE INFORMACIJSKIH SUSTAVA I OSIGURANJE KVALITETE

Mrvelj, Marijana

Sveučilište Josipa Jurja Strossmayera u Osijeku
Ekonomski fakultet u Osijeku
Sveučilišni prijediplomski studij Poslovna informatika

Marijana Mrvelj

TESTIRANJE INFORMACIJSKIH SUSTAVA I OSIGURANJE KVALITETE

Završni rad

Osijek, 2023.

Sveučilište Josipa Jurja Strossmayera u Osijeku
Ekonomski fakultet u Osijeku
Sveučilišni prijediplomski studij Poslovna informatika

Marijana Mrvelj

TESTIRANJE INFORMACIJSKIH SUSTAVA I OSIGURANJE KVALITETE

Završni rad

Kolegij: Oblikovanje i implementacija IS-a

JMBAG: 0010234035
e-mail: mmrvelj@efos.hr

Mentor: doc. dr. sc. Dario Šebalj

Osijek, 2023.

Josip Juraj Strossmayer University of Osijek
Faculty of Economics and Business in Osijek
Undergraduate Study Business Informatics

Marijana Mrvelj

INFORMATION SYSTEMS TESTING AND QUALITY ASSURANCE

Final paper

Osijek, 2023.

IZJAVA

O AKADEMSKOJ ČESTITOSTI, PRAVU PRIJENOSA INTELEKTUALNOG VLASNIŠTVA, SUGLASNOSTI ZA OBJAVU U INSTITUCIJSKIM REPOZITORIJIMA I ISTOVJETNOSTI DIGITALNE I TISKANE VERZIJE RADA

1. Kojom izjavljujem i svojim potpisom potvrđujem da je_____završni_____ (navesti vrstu rada: završni / diplomski / specijalistički / doktorski) rad isključivo rezultat osobnoga rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu. Potvrđujem poštivanje nepovredivosti autorstva te točno citiranje radova drugih autora i referiranje na njih.
2. Kojom izjavljujem da je Ekonomski fakultet u Osijeku, bez naknade u vremenski i teritorijalno neograničenom opsegu, nositelj svih prava intelektualnoga vlasništva u odnosu na navedeni rad pod licencom *Creative Commons Imenovanje – Nekomercijalno – Dijeli pod istim uvjetima 3.0 Hrvatska.* 
3. Kojom izjavljujem da sam suglasan/suglasna da se trajno pohrani i objavi moj rad u institucijskom digitalnom repozitoriju Ekonomskoga fakulteta u Osijeku, repozitoriju Sveučilišta Josipa Jurja Strossmayera u Osijeku te javno dostupnom repozitoriju Nacionalne i sveučilišne knjižnice u Zagrebu (u skladu s odredbama Zakona o visokom obrazovanju i znanstvenoj djelatnosti, NN 119/2022).
4. izjavljujem da sam autor/autorica predanog rada i da je sadržaj predane elektroničke datoteke u potpunosti istovjetan sa dovršenom tiskanom verzijom rada predanom u svrhu obrane istog.

Ime i prezime studenta/studentice: Marijana Mrvelj

JMBAG: 0010234035

OIB: 21258837549

e-mail za kontakt: mrvelj04@gmail.com

Naziv studija: Sveučilišni prijediplomski studij, Poslovna informatika

Naslov rada: Testiranje informacijskih sustava i osiguranje kvalitete

Mentor/mentorica rada: doc. dr. sc. Dario Šebalj

U Osijeku, _____23. lipnja 2023._____ godine

Potpis Marijana Mrvelj

Testiranje informacijskih sustava i osiguranje kvalitete

SAŽETAK

Svrha ovoga rada je pobliže objasniti pojam testiranja te stvoriti dublje razumijevanje važnosti testiranja informacijskih sustava i osiguranja kvalitete. Zbog sve veće konkurencije i napretka tehnologije, potrebe klijenata postaju sve složenije te su i njihovi zahtjevi sve kompleksniji. Glavni cilj testiranja informacijskih sustava i osiguranja kvalitete je upravo razvoj ispravnog i visoko kvalitetnog softvera koji će odgovarati korisničkim potrebama prije nego što on bude pušten u uporabu. Testiranje informacijskih sustava i osiguranje kvalitete su ključne aktivnosti u razvoju softvera. Kroz njihovu primjenu poslovni subjekti i organizacije mogu ostvariti uspješan razvoj softvera koji zadovoljava zahtjeve korisnika te ostvaruje konkurentske prednosti. Poslovni subjekti se moraju prilagođavati novim trendovima, metodama i alatima kako bi osigurali da njihovi informacijski sustavi budu visokokvalitetni, pouzdani i sigurni. U ovome radu prikazane su prednosti osiguranja kvalitete kao što je povećanje povjerenja klijenta te izbjegavanje pogrešaka prilikom razvoja softvera što na kraju dovodi i do financijskih ušteda. Također, objašnjene su razine testiranja, kao što su jedinično testiranje, integracijsko, sistemsko testiranje i test prihvaćanja te metode i tehnike testiranja, od kojih su najpoznatije metode *black box* i *white box* testiranje. U radu je prikazano i kako izgleda sam proces testiranja te su ukratko predstavljeni alati koji se mogu koristiti prilikom testiranja softvera u različitim fazama i vrstama testiranja.

Ključne riječi: testiranje informacijskih sustava, osiguranje kvalitete, korisnički zahtjevi, metode i tehnike testiranja

Information systems testing and quality assurance

ABSTRACT

The purpose of this paper is to explain the concept of testing in more detail and to create a deeper understanding of the importance of information systems testing and quality assurance. Due to increasing competition and technological progress, the needs of clients are becoming much more complex, and their requirements as well. The main goal of information systems testing and quality assurance is precisely the development of correct and high-quality software that will meet user needs before it is put into use. Information systems testing and quality assurance are key activities in software development. Through their application, different businesses and organizations can achieve successful software development that meets user requirements and achieves competitive advantages. Businesses must adapt to new trends, methods, and tools to ensure that their informations systems are of high quality, and that they are reliable and secure. This paper presents the advantages of quality assurance, such as increasing client trust and avoiding errors during software development, leading to financial savings in the end. Additionaly, levels of testing are explained, including unit testing, integration testing, system testing, and acceptance testing, as well as the testing methods and techniques, of which the most known are black box and white box testing. The paper also illustrates the testing process itself and briefly introduces the tools that can be used during software testing in the different stages and types of testing.

Keywords: information systems testing, quality assurance, user requirements, testing methods and techniques

SADRŽAJ

1. Uvod

Testiranje informacijskih sustava i osiguranje kvalitete predstavljaju aktivnosti od iznimne važnosti u razvoju softvera i informacijskih sustava. Budući da danas informacijski sustavi postaju sve složeniji, i od iznimnog značaja za korisnike, funkcionalnost, pouzdanost i sigurnost softvera presudni su za uspjeh poslovnih subjekata, ali i za zadovoljstvo korisnika.

Testiranje informacijskih sustava se odnosi na proces provjere softvera radi otkrivanja nedostataka, pogrešaka ili različitih rizika koji bi mogli ugroziti njegovu kvalitetu i funkcionalnost. Ovaj proces uključuje izvođenje različitih testova te korištenje raznih metoda testiranja kako bi se provjerilo ponašanje informacijskog sustava u različitim scenarijima, uvjetima i u različitom okruženju. Cilj testiranja je identificirati pogreške i nedostatke te ih ispraviti prije nego što sustav bude pušten u uporabu.

Nasuprot tome, osiguranje kvalitete predstavlja sveobuhvatan pristup koji se primjenjuje tijekom cijelog ciklusa razvoja softvera, a sve s ciljem postizanja visokih standarda kvalitete proizvoda. Osiguranje kvalitete uključuje planiranje, implementaciju i praćenje različitih metoda i testova kako bi se osigurala ispravnost, pouzdanost i performanse softvera. Osim testiranja, osiguranje kvalitete obuhvaća i aktivnosti kao što su revizija koda, analiza korisničkih zahtjeva, upravljanje procesima te praćenje zadovoljstva klijenata.

U današnjem iznimno konkurentnom poslovnom okruženju, poslovni subjekti i organizacije se suočavaju sa sve kompleksnijim zahtjevima korisnika i brzim promjenama tehnologija i poslovanja. Stoga je testiranje informacijskih sustava i njihovo osiguranje kvalitete ključno kako bi se osiguralo da softver ispunjava korisničke potrebe, radi ispravno i sveukupno pruža pozitivno korisničko iskustvo. Učinkovito testiranje i osiguranje kvalitete može smanjiti rizike od financijske štete, gubitka povjerenja klijenata te na kraju i oštećenja ugleda poslovnog subjekta.

U ovom radu istraženi su različiti aspekti testiranja informacijskih sustava i osiguranja kvalitete. Objašnjene su različite razine testiranja, prikazane su najvažnije metode i tehnike testiranja, kao i alati koji se koriste prilikom testiranja softvera.

2. Osiguranje kvalitete

U današnjem konkurentnom poslovnom okruženju, poslovni subjekti vode iznimnu brigu o kvaliteti svojih proizvoda i usluga. Uspješna organizacija mora težiti unaprjeđivanju kvalitete u svakom području pa tako i kvalitete svojih informacijskih sustava. Najviša razina menadžmenta mora osigurati vodstvo, ohrabrenje i podršku za potrebne visokokvalitetne IT resurse (Tilley, 2019:353).

Bez obzira na to koliko je pozorno i precizno sustav dizajniran i implementiran, problemi se mogu pojaviti. Rigorozna testiranja mogu otkriti pogreške tijekom implementacije, ali je daleko jeftinije ispraviti te pogreške ranije, odnosno u samom procesu razvoja. Glavni cilj sustava osiguranja kvalitete (engl. *Quality Assurance*) informacijskih sustava je izbjeći probleme ili ih identificirati što je prije moguće. Loša kvaliteta može prouzrokovati netočne zahtjeve, probleme u dizajnu, pogreške kodiranja, neispravnu dokumentaciju te na kraju i neučinkovito testiranje informacijskog sustava. Kako bi poboljšali gotov proizvod, programeri softverskih sustava trebali bi uzeti u obzir najbolju praksu u softverskom inženjerstvu, sistemskom inženjerstvu i međunarodno priznatim standardima kvalitete (Tilley, 2019:353).

Dakle, osiguranje kvalitete informacijskog sustava ili softvera, kao što i sam naziv kaže, je proces ili uloga softverskog inženjera koji osigurava da nema "ustupaka ili iskliznuća" u informacijskom sustavu u odnosu na one zahtjeve koje je postavio korisnik (Goyal, n.d.).
Kompletan proces osiguranja kvalitete oslanja se na PDCA ciklus (engl. *Plan, Do, Check, Act*) ili Demingov ciklus koji se sastoji od četiri faze za kontinuirano poboljšanje prilikom upravljanja poslovnim procesima, a te faze su planiranje, izvođenje, provjera i djelovanje (Wigmore, 2015).

Planiranje je zapravo planiranje mjera potrebnih za održavanje standarda softvera na visokoj kvaliteti, izvođenje je razvojni proces koji uključuje procese izgradnje i testiranja, provjera uključuje promatranje i ispitivanje provedbenih puteva, a djelovanje podrazumijeva poduzimanje aktivnosti koje su potrebne za održavanje kvalitete (Goyal, n.d.).

Osiguranje kvalitete uključuje različite aktivnosti. Prva aktivnost je svakako postavljanje kontrolnih točki. Tim uspostavlja kontrolne točke u određenim intervalima kako bi provjerio napredak, kvalitetu, performanse softvera i je li rad na kvaliteti softvera obavljen na vrijeme

prema rasporedu i dokumentima. Zatim, druga aktivnost je mjerenje utjecaja promjene. U slučaju greške koju je prijavio tim za osiguranje kvalitete i popravio programer, vrlo je važno ponovno testirati popravljenu grešku i osigurati da ispravljena greška ne uvodi nove greške u softver. U tu svrhu održavaju se testne metrike koje promatraju menadžeri i programeri kako bi provjerili pojavu grešaka uvođenjem novih značajki ili ispravljanjem grešaka. Nadalje, sljedeća aktivnost podrazumijeva imati strategiju višestrukog testiranja. Ne treba se oslanjati na jedan pristup i strategiju testiranja softvera. U softver bi se trebale implementirati različite strategije testiranja kako bi se testiralo iz različitih kutova i kako bi se pokrila sva područja. Uz sve navedene aktivnosti, uz njih se ističe i aktivnost vođenja zapisa i izvješća. Važno je da se svi zapisi i dokumenti o osiguranju kvalitete čuvaju i pravodobno budu dostupni ostalim dionicima. Izvršeni testni slučajevi, testni ciklusi, zabilježene i ispravljene pogreške, kreirani testni slučajevi, promjene zahtjeva korisnika za određeni testni slučaj moraju biti ispravno dokumentirani za buduću upotrebu (Goyal, n.d.).

Osim aktivnosti, osiguranje kvalitete obuhvaća i određene tehnike. Jedna od tehnika osiguranja kvalitete je revizija. Tijekom revizije, održava se sastanak unutarnjih i vanjskih dionika kako bi se pregledao cijeli projekt, analizirao cijeli softver i, ako se naiđe na problem, razlučuje se je li došlo do problema tijekom testiranja, razvoja ili dizajna. Glavni cilj je izmjeriti kvalitetu softvera, uvjeriti se ispunjava li očekivanja korisnika i slijedi li standardne procese. Sljedeća tehnika osiguranja kvalitete je funkcionalno testiranje. U funkcionalnom testiranju, testira se funkcionalnost cjelokupnog softvera kako bi se vidjelo radi li prema očekivanjima. Provjerava se "što sustav radi", pritom bez znanja "kako sustav radi". Nadalje, bitna tehnika osiguranja kvalitete je svakako i provjera koda. Provjera koda je jedna od formalnijih vrsta pregleda, s osnovnim ciljem pronalaženja grešaka u kodu. Provjeru vrši kvalificirani moderator, a ne autor koda. Sastanak ima odgovarajuće ulazne i izlazne kriterije. Korisnicima je potrebna temeljita priprema prije sastanka kako bi se upoznali sa svim dokumentima prije nego što iznesu svoje mišljenje. Osim funkcionalnog testiranja, provodi se i testiranje otpornosti na stres. Ono se provodi kako bi se provjerilo kako sustav radi pod velikim opterećenjem. Ovi testovi igraju važnu ulogu u kvaliteti softvera jer se provode kako bi se testirao kapacitet softvera, odnosno koliko korisnika može pristupiti aplikaciji u isto vrijeme (Goyal, n.d.).

Proces osiguranja kvalitete informacijskih sustava svakako ima svoje prednosti. Jedna od njih je da se povećava povjerenje klijenata. Ispravna kontrola kvalitete softvera te uključivanje unutarnjih i vanjskih dionika osigurava povjerenje klijenata. Također, podnošenje tjednih

izvješća uvelike pomaže u uvjeravanju i osiguravanju klijenta da se posao obavlja na vrijeme. Druga prednost se očituje kroz uštedu novca. Greške pronađene u ranoj fazi, bilo u prikupljanju zahtjeva, kodu, testiranju, jednostavne su i isplative. Odgovarajuća kontrola izvedena na više razina pomaže u smanjenju tih rizika, budući da se većina grešaka otkrije i ispravi u ranim fazama, čime se štedi novac (Goyal, n.d.).

3. Testiranje informacijskih sustava

Popović (2012:1) navodi kako je testiranje vrlo važna aktivnost u razvoju informacijskih sustava ili softvera te je izuzetno dobila na značaju kako je vrijednost softvera počela rasti. Pogreške u softveru, koji može vrijediti i milijune dolara, mogu prouzrokovati ogromnu financijsku štetu i stoga ih treba popraviti što je prije moguće. Postupno, testiranje postaje aktivnost koja je jednako važna kao i sam razvoj informacijskog sustava.

U razvoju informacijskog sustava, testiranje predstavlja pokušaj pronalaženja pogrešaka u softveru koji se stvara. Informacijski sustav se implementira prema potrebama korisnika kako bi se riješio problem ili kako bi se stvorila korisna funkcija koja je potrebna krajnjem korisniku. Jednom implementiran, informacijski sustav može više ili manje zadovoljiti izvorne potrebe za koje je dizajniran. Svako ponašanje informacijskog sustava koje ne zadovoljava izvorne zahtjeve je pogreška koju treba identificirati i popraviti (Popović, 2012:1-2).

U užem smislu, testiranje jednostavno znači provjeru je li navedeni informacijski sustav u potpunosti implementiran u skladu s izvornim zahtjevima korisnika. U širem smislu, testiranje označava sustav kontrole kvaliteta (*QA – Quality Assurance*) kojim se ne provjerava samo informacijski sustav, nego i sve njegove komponente i funkcije (Popović, 2012:1-2).

3.1. Razine testiranja

Planiranje testiranja skup je aktivnosti koje određuju što se testira, kada i kako. Ishod testiranja je plan koji definira cjelokupan proces testiranja. Za veći projekt koji zahtijeva opsežnije testiranje, plan testiranja može se rastaviti na nekoliko manjih test planova tako da jedan glavni plan definira osnovni proces, dok drugi planovi određuju pojedinosti o jediničnom testiranju, integracijskom, sistemskom testiranju te testu prihvaćanja (Popović, 2012:75).

3.1.1. Jedinično testiranje

Damodar (2023:33) navodi da se najmanji dio koji se može izdvojiti iz izvornog koda aplikacije, naziva testiranje jedinica (funkcija, procedura) te se testovi provode na tim jedinicama kako bi se provjerio njihov kod i rade li one prema očekivanjima ili ne.

Popović (2012:77) navodi kako jedinično testiranje predstavlja testiranje izoliranih cjelina u sustavu. Preduvjet za jedinično testiranje je da se značajka koja se testira može percipirati kao

neovisna cjelina koja se može izdvojiti iz konteksta sustava te testirati odvojeno od ostalih značajki. Te značajke mogu biti dio korisničkog sučelja (engl. *interface*) ili značajke koje izvode određene radnje i obrađuju rezultate.

Značajka koja se testira izvlači se iz konteksta sustava, gdje komunicira s drugim značajkama i postavlja se u test kontekst koji predstavlja simulaciju stvarnog sustava sa značajkama koje simuliraju rad stvarnih značajki (Popović, 2012:77).

Prema TechTargetu (n.d.), jedinično testiranje je proces razvoja softvera u kojem se najmanji dijelovi aplikacije koji se mogu testirati, koji se nazivaju jedinicama, pojedinačno testiraju radi ispravnosti koda. Programeri softvera i ponekad osoblje osiguranja kvalitete izvršavaju jedinične testove tijekom procesa razvoja. Glavni cilj jediničnog testiranja je izolirati pisani kod i utvrditi radi li on onako kako je predviđeno. Jedinično testiranje je sastavni dio razvoja vođenog testiranjem pragmatične metodologije (engl. TDD – *test-driven development*) koja ima pristup izradi proizvoda putem kontinuiranog testiranja i revizije.

Jedinični test se sastoji od tri faze: planiranje, pisanje testnih slučajeva i izvođenje samog testa. U prvom koraku programeri ili stručnjaci za osiguranje kvalitete pripremaju i pregledavaju jedinični test. U sljedećem koraku pišu se testni slučajevi i skripte, a u posljednjem koraku se testira kod. Razvoj aplikacije vođen testiranjem zahtijeva od programera da napišu neuspjele jedinične testove. Zatim, pišu kod i restrukturiraju aplikaciju dok test ne bude uspješan (TechTarget, n.d.).

Jedinično testiranje uključuje samo one karakteristike koje su vitalne za rad jedinice koja se testira. Ovo potiče programere da modificiraju izvorni kod bez brige o tome kako bi takve promjene mogle utjecati na funkcioniranje drugih jedinica u cjelini. Nakon što sve jedinice u programu rade na najučinkovitiji mogući način i bez grešaka, timovi mogu evaluirati veće komponente programa pomoću integracijskog testiranja (TechTarget, n.d.).

3.1.2. Integracijsko testiranje

Popović (2012:79) iznosi da se integracijsko testiranje provodi kao prirodni nastavak jediničnog testiranja. Kada su značajke stvorene te testirane svaka pojedinačno, moraju se testirati u radu, ne sa simuliranim značajkama, već s drugim stvarnim značajkama. Nakon testiranja pojedinih

značajki one se mogu spojiti u cjelinu koja bi trebala dobro funkcionirati. Međutim, testovi sa simuliranim značajkama još nisu dovoljno pouzdani kako bi potvrdili da značajke rade sa stvarnim značajkama, jednako dobro kao i u simulacijama. Zbog toga je dobra ideja da se značajke povezuju u manjim dijelovima te se provjerava rade li u manjim cjelinama koje se postupno nadograđuju nadovezivanjem novih značajki.

Dakle, integracijsko testiranje je razina testiranja softvera gdje se pojedinačne jedinice kombiniraju i testiraju kao grupa. Svrha ove razine testiranja je otkrivanje grešaka u interakciji između integriranih jedinica. Testni upravljački programi (engl. *Test Driver*) i testni dodaci (engl. *Test Stubs*) koriste se kao pomoć u integracijskom testiranju. Integracijsko testiranje obično provode programeri (Software Testing Fundamentals, 2023).

Razlikuju se četiri pristupa integracijskog testiranja. Prvi od njih je *Big Bang* pristup. Kod ovog pristupa integracijskom testiranju sve ili većina jedinica se kombinira zajedno i testira odjednom. *Big Bang* pristup se koristi kada tim za testiranje dobije cijeli softver te testiraju interakcije između jedinica. Drugi pristup integracijskom testiranju je *Top Down* pristup. Ovo je pristup gdje se prvo testiraju jedinice najviše razine, a zatim jedinice niže razine korak po korak. Ovaj pristup se koristi kada se slijedi razvojni pristup odozgo prema dolje. Testni dodaci su potrebni za simulaciju jedinica niže razine. Sljedeći pristup je *Bottom Up* pristup. To je pristup integracijskom testiranju gdje se prvo testiraju jedinice najniže razine, a potom jedinice više razine korak po korak te se ovaj pristup koristi kada se slijedi razvojni pristup odozdo prema gore i ovdje su potrebni testni upravljački programi za simulaciju jedinica više razine. Posljednji pristup integracijskog testiranja je hibridni pristup koji je kombinacija prethodna dva pristupa, *Top Down* i *Bottom Up* pristupa, gdje se testiranje izvodi u oba smjera istovremeno (Software Testing Fundamentals, 2023).

3.1.3. Sistemsko testiranje

Sistemsko testiranje označava testiranje sustava koji je u potpunosti integriran. Iako se sistemsko testiranje može činiti kao posljednja faza integracijskog testiranja, u kojoj se posljednja značajka integrira i testira, ipak postoji bitna razlika. Integracijsko testiranje uvijek testira izravno povezane značajke i traži pogreške u komunikaciji sa sučeljima, dok sistemsko testiranje ispituje zajednički rad značajki koje nisu izravno povezane jedna s drugom i ne komuniciraju izravno, ali imaju neku ovisnost po podacima (Popović, 2012:80).

Još jedna od definicija sistemskog testiranja je da sistemsko testiranje predstavlja proces razvoja softvera kojim se procjenjuje može li dovršeni program ispravno funkcionirati i ispuniti specifikacije klijenta. Obično uključuje niz procjena kako bi se vidjelo kako programski kod radi na hardverskom sustavu računala i kako bi se provjerilo jesu li različite značajke softvera operativne. Kako bi se smanjila mogućnost utjecaja programera na proces, testeri softvera često te procjene provode u zasebnom okruženju. Sistemsko testiranje provjerava može li se programski jezik softvera prevesti u upotrebljiv program. To je vrsta testiranje crne kutije (engl. *Black box testing*), što znači da tester može vidjeti program samo iz perspektive korisnika u sličnom okruženju. Testeri mogu provjeriti podudaraju li se namjeravani rezultati kodnog jezika s onim što opažaju dok rade s programom. Također, mogu locirati pogreške u izvedbi programa koje se mogu pojaviti nakon što kodni jezik stupi u interakciju s hardverom računala (Indeed, 2023).

Postoje razne vrste sistemskog testiranja koje se mogu izvršiti. Primjerice, testiranje instalacije uključuje procjenu može li korisnik uspješno instalirati softverski program na računalo. Često uključuje promatranje postupka instalacije programa, uključujući njegovu sposobnost da identificira raspoloživi prostor na tvrdom disku. Zatim, još jedna vrsta sistemskog testiranja je testiranje funkcionalnosti koje uključuje provjeru značajki programa kako bi se vidjelo rade li one ispravno. Može uključivati sveobuhvatnu procjenu jednog aspekta, kao što je gumb za prijavu, ili općenitiji pregled programa (Indeed, 2023).

3.1.4. Test prihvaćanja

Popović (2012:81) navodi kako test prihvaćanja predstavlja testiranje cijelog sustava, ne za sve moguće slučajeve, već za scenarije koje koriste krajnji korisnici. S jedne strane, to je blaži test od sistemskog testiranja jer se ne primjenjuju svi detaljno dizajnirani testni slučajevi. S druge strane, može biti prilično teško testiranje jer je potrebno pronaći scenarije korištenja koje bi krajnji korisnici inače koristili.

Nadalje, test prihvatljivosti podrazumijeva formalno testiranje koje se temelji na zahtjevima korisnika i funkcijskoj obradi. Određuje je li softver u skladu s određenim zahtjevima. Provodi se kao neka vrsta testiranja crne kutije gdje određeni broj potrebnih korisnika uključenih u testiranje, testiraju razinu prihvatljivosti sustava. Test prihvatljivosti obično provodi kupac, stručnjak za domenu, na svoje zadovoljstvo i provjerava radi li aplikacija prema zadanim

poslovnim scenarijima. Pri tome se koncentriraju samo na one značajke i scenarije koje korisnici redovito koriste. Test prihvatljivosti je nužan iz više razloga. Na primjer, razvojni programeri razvijaju određene funkcije programa na temelju dokumenta koji sadržava popis zahtjeva. Budući da ga razvijaju prema vlastitom razumijevanju, postoji mogućnost da programeri možda neće razumjeti stvarne zahtjeve klijenta. Također, možda postoje neke manje pogreške koje se mogu identificirati samo kada sustav koristi krajnji korisnik u stvarnom scenariju, tako da je test prihvaćanja ključan kako bi se otkrile manje pogreške (Javatpoint, n.d.a).

3.2. Metode i tehnike testiranja

Metode testiranja informacijskih sustava ili softvera izuzetno su bitne prilikom njihovog razvoja. Metode pomažu programerima kako bi se znali nositi s različitim vrstama pogrešaka. Te pogreške mogu varirati od nedostajuće točke ili zareza u kodu pa do kritičnih poslovnih zahtjeva. U razvoju softvera postoje različite metode testiranja. Odabir pravih alata prilikom testiranja se ponekad može pokazati kao nespretan proces. Stoga je posao tima za osiguranje kvalitete, uspostaviti prikladnu metodu testiranja (Stackify, 2020).

3.2.1. Statičko i dinamičko testiranje

Statičko testiranje ili statička analiza ne uključuje stvarno izvršavanje koda. Umjesto toga, ispituje sva moguća ponašanja koja bi se mogla pojaviti tijekom izvođenja. Testeri provode statičko testiranje pregledom dokumentacije. Ta dokumentacija uključuje specifikaciju sistemskih zahtjeva, modele, arhitekturu dizajna i stare kodove. Tradicionalna statička analiza uključuje pregled koda i njegovu inspekciju, analizu algoritama i dokaz o ispravnosti (Stackify, 2020).

Glavni ciljevi statičkog testiranja su identificirati i ispraviti nedostatke u zahtjevima kako bi se spriječili nedostaci u dizajnu ili kodu te identificirati i ispraviti nedostatke u testnim slučajevima. Nadalje, cilj je i smanjenje troškova tijekom razvoja softvera te poboljšanje timske komunikacije i suradnje (Software Testing Fundamentals, 2022).

Prilikom statičkog testiranja dobro je slijediti određene smjernice, kao što su ove u nastavku (Software Testing Fundamentals, 2022):

- provjeriti jesu li ciljevi svakog statičkog testa jasni,

- pojasniti uloge i odgovornosti svake uključene osobe,
- započeti sa statičkim testiranjem na vrijeme,
- izraditi i koristiti popise za provjeru,
- sudionicima dati dovoljno vremena za pripremu,
- uključiti statističko testiranje u politike/procedure poslovnog subjekta tako da postoji organizacijska podrška.

Dinamičko testiranje uključuje stvarno izvršavanje programa kako bi se otkrile moguće pogreške i neispravne funkcije. Također, dio dinamičkog testiranja su svojstva ponašanja i performanse softvera. Cilj ispitivača softvera je identificirati što više pogrešaka. Statička analiza je najbolja za prepoznavanje pogrešaka u ranoj fazi razvoja softvera. S druge strane, dinamička analiza jako dobro funkcionira paralelno s razvojem. Stoga, statička i dinamička analiza su komplementarne te nude bolju učinkovitost ako se koriste zajedno (Stackify, 2020).

3.2.2. Black box, White box i Grey box testiranje

Black box testiranje je tehnika testiranja informacijskog sustava bez ikakvog znanja o unutarnjem radu aplikacije. Tester nije svjestan arhitekture sustava i nema pristup izvornom kodu. Inače, tijekom izvođenja *black box* testiranja, tester će komunicirati s korisničkim sučeljem sustava pružajući određene ulazne elemente i ispitujući izlazne elemente, a da pri tome ne zna kako i gdje se radi s ulaznim elementima. *Black box* testiranje pruža određene prednosti, a neke od njih su sljedeće (Tutorials Point, n.d.):

- *black box* testiranje je dobro prilagođeno i učinkovito za velike segmente koda,
- nije potreban pristup kodu,
- jasno odvaja perspektivu korisnika od perspektive programera kroz jasno definirane uloge,
- velik broj "umjereno kvalificiranih" testera može testirati aplikaciju bez znanja o implementaciji, programskom jeziku ili operativnim sustavima.

White box testiranje predstavlja istraživanje unutarnje logike i strukture koda. Također, naziva se i testiranje staklene kutije (engl. *glass box testing*) ili testiranje otvorene kutije (engl. *open box testing*). Kako bi se izvršilo *white box* testiranje, tester treba poznavati pozadinski kod. Tester mora pogledati unutar izvornog koda i otkriti koja se jedinica ili dio koda ponaša neprikladno (Tutorials Point, n.d.).

Kao i sve ostale metode, *white box* testiranje ima svoje prednosti (Tutorials Point, n.d.):

- kako tester ima znanja o izvornom kodu, vrlo lako se može saznati koja vrsta podataka može pomoći u učinkovitom testiranju aplikacije,
- *white box* testiranje pomaže u optimizaciji koda,
- mogu se ukloniti redovi koda koji mogu dovesti do skrivenih nedostataka.

Grey box testiranje je tehnika testiranja aplikacije s ograničenim znanjem o unutarnjem radu aplikacije. Ovladavanje domenom sustava, testeru uvijek daje prednost u odnosu na nekoga s ograničenim znanjem o domeni. Za razliku od *black box* testiranja, gdje ispitivač testira samo korisničko sučelje aplikacije; u *grey box* testiranju, tester ima pristup projektnim dokumentima i bazi podataka. Imajući to znanje, tester može pripremiti bolje testne podatke i testne scenarije dok izrađuje testni plan. Prednosti *grey box* testiranja su (Tutorials Point, n.d.):

- *grey box* testeri ne oslanjaju se na izvorni kod, već na definiciju sučelja i na funkcionalne specifikacije,
- na temelju ograničenih dostupnih informacija, tester može dizajnirati izvrsne scenarije testiranja,
- test se radi sa stajališta korisnika, a ne dizajnera.

3.2.3. Ručno i automatizirano testiranje

Pri ručnom testiranju, ljudska prosudba i intuicija uvijek idu u korist projekta. Testeri provode ručno testiranje pomoću određenih istraživačkih metoda. Prednost ručnog testiranja je dobivanje brze i točne povratne informacije. Također, ovo je jeftinija metoda jer ne uključuje alate i procese za automatizaciju (Stackify, 2020).

Automatizirano testiranje koristi određene skripte i alate te to ovu metodu čini pouzdanijom jer se test pokreće automatski. Kao rezultat toga, pronalazi više pogrešaka u usporedbi s ljudskim testerom. Također, testerima je dopušteno snimanje procesa automatizacije, što im omogućuje ponovnu upotrebu i izvršavanje istih operacija testiranja. Neki alati za testiranje softvera, kao što je Selenium, mogu povećati produktivnost i pružiti brze i točne rezultate testiranje. Također, ima široku pokrivenost jer automatizirano testiranje nikada ne preskače čak ni najmanju jedinicu koja se testira. Međutim, alati koji se koriste za automatizirano testiranje su prilično skupi, što može povećati troškove projekta testiranja. To je rizik koji treba preuzeti jer postoje alati koji još nisu sigurni (Stackify, 2020).

4. Proces testiranja

Proces testiranja informacijskog sustava podrazumijeva skup aktivnosti koje tim za testiranje provodi prilikom testiranja projekta. Može se definirati na razini pojedinačnog projekta, ali isto tako se u većim organizacijama proces testiranja može utvrditi na razini cjelokupne organizacije (Popović, 2012:41).

Sa stajališta testiranja na pojedinačnim projektima, proces testiranja uključuje aktivnosti planiranja testiranja, koje predstavljaju identifikaciju aktivnosti i resursa, dizajniranje testova, što podrazumijeva analizu projekta koji se testira, identifikaciju testnih slučajeva te njihovu analizu kako će se ti testni slučajevi testirati te na kraju i sam postupak testiranja i prijavljivanja pogrešaka, kao i njihovo praćenje i rješavanje (Popović, 2012:41).

Šire gledano, proces testiranja uključuje aktivnosti koje se izvode na organizacijskoj razini koje definiraju koje standarde treba postići, kako testirati i s kojim alatima (Popović, 2012:41).

4.1. Planiranje testiranja

Planiranje označava kontinuirani proces analize poslova koje je potrebno učiniti na projektu i definiranja tko i kada će ih odraditi. Osim identifikacije poslova i dodjele zadataka koji trebaju biti odrađeni, planiranje obuhvaća i osiguranje svih potrebnih uvjeta za rad, kao što su resursi i edukacija (Popović, 2012:42).

Nadalje, planiranje testiranja je važna aktivnost koja se usredotočuje na testiranje različitih značajki softvera prije njegovog izdavanja. Kako bi proveli temeljito testiranje, voditelji testiranja započinju pisanjem plana testiranja. Plan testiranja predstavlja sveobuhvatan dokument koji opisuje strategiju, ciljeve, rokove i prirodu testova. Služi kao nacrt za provođenje različitih testova za određivanje performansi softvera (Indeed, 2022).

Plan testiranja je vrlo dinamičan, a njegovi sadržaji variraju kako projekt napreduje. Ovisno o rezultatima i različitim rasporedima, voditelji testiranja redovito ažuriraju plan testiranja. Većina timova za testiranje i programera smatra plan testiranja referentnom točkom za određivanje vrste testova koje treba provesti. Voditelji dijele plan s onima koji su uključeni, uključujući razvojne timove, poslovne menadžere i voditelje projekta, što nudi veću transparentnost svim dionicima. Pisanje sveobuhvatnog plana prvi je dio planiranja niza

testova. Bez plana, razvojni timovi nemaju reference ili koncept o opsegu testova koje treba provesti. Plan testiranja važan je iz više razloga. Kao što je već ranije navedeno, dobro napisan plan testiranja omogućuje članovima i dionicima koji nisu dio tima za osiguranje kvalitete, razumijevanje parametara testiranja te nudi veću transparentnost svim stranama o tome kako upravitelji namjeravaju testirati softver. Ovo je važno za održavanje učinkovite komunikacije jer informira poslovne voditelje i menadžere točno kada i kako razvojni timovi planiraju provesti testove. Plan je prilično temeljit, usredotočen je na prirodu testova i njihove očekivane ishode. Također, naglašava opseg testiranja, budući da većina softverskih rješenja sadrži previše značajki da bi se testirale istovremeno. Inženjeri se mogu pozvati na plan testiranja pri pregledu ciljeva testa. To timovima za osiguranje kvalitete olakšava planiranje različitih testova i identificiranje što je moguće više grešaka prije početnog izdanja. Plan pomaže inženjerskim timovima poboljšati kvalitetu testiranja i omogućuje im planiranje testova koji učinkovito koriste resurse tvrtke za postizanje očekivanih rezultata (Indeed, 2022).

Između ostalog, plan testiranja pojednostavljuje proces pregleda. Programeri i inženjeri koriste ga kao vodič za planiranje različitih aktivnosti i određivanje raspodjele resursa. Objedinjavanjem svih informacija u jednom dokumentu omogućeno je bolje razumijevanje učinkovitosti testiranja. Kada se kasnije provodi opsežniji pregled, može se pozvati na plan kako bi se usporedili njegovi očekivani rezultati sa stvarnim rezultatima. To daje bolje razumijevanje kako planirati aktivnosti testiranja i omogućuje točnije procjene u budućnosti. Također, može se procijeniti potrošnja resursa tijekom aktivnosti testiranja i planirati učinkovito budžetiranje za buduće testove (Indeed, 2022).

Koraci koje bi trebalo slijediti prilikom kreiranja plana testiranja su sljedeći (Indeed, 2022):

- analiza proizvoda,
- procjena rizika i isticanje ciljeva,
- definiranje opsega testiranja,
- alokacija resursa,
- testiranje rezultata.

Dakle, prvi korak je precizna analiza proizvoda kako bi se utvrdila njegova glavna svrha i krajnji korisnici. Voditelji testiranja zahtijevaju sveobuhvatno razumijevanje proizvoda i

njegovih specifikacija. Preciznom analizom softvera, voditelji testiranja mogu steći bolje razumijevanje različitih elemenata koji mogu zahtijevati dodatnu pozornost (Indeed, 2022).

Sljedeći korak je isticanje uočenih rizika testiranja i ocrtavanje ciljeva. Ovaj korak je važan jer daje detaljan opis ciljeva koje tvrtka planira postići provođenjem testova. Primjerice, tvrtka može pokrenuti aktivnosti testiranja novog softvera kako bi identificirala i popravila određeni broj pogrešaka ili može pokrenuti testove kako bi utvrdila radi li novododana značajka ispravno (Indeed, 2022).

Popović (2012:43) navodi da se analizom svojstava, značajki i okoline u kojoj se razvijeni sustav koristi, mogu identificirati različite prijetnje sustavu. Ovi rizici govore testnom timu na što treba obratiti pozornost prilikom testiranja. Rizici su izravno povezani s aktivnostima koje se provode u projektu i s njegovim karakteristikama. Stoga se identifikacija rizika vrši pregledom aktivnosti u planu.

Primjer nekih aktivnosti i uvjeta koji sa sobom nose određene rizike su (Popović, 2012:43-44):

1. Sa svakom implementiranom ili promijenjenom značajkom, postoji rizik da neće biti implementirana prema potrebi.
2. Svaki dokument, model koji dokumentira potrebe korisnika i njihove zahtjeve, sa sobom nosi prijetnju da neće biti prilagođen stvarnim potrebama korisnika.
3. Svaka funkcija koja izvodi složenu obradu podataka nosi rizik ugrožavanja performansi sustava.
4. Kada sustav sadrži podatke koji bi mogli biti vrijedni ili korisni nekome tko nije izravni korisnik sustava, postoji rizik da bi netko mogao provaliti u sustav i pokušati ga preuzeti.
5. U slučaju da sustav predstavlja konkurenciju drugom sustavu, postoji prijetnja da će ga netko pokušati napasti.

Nakon analize zadataka i zahtjeva projekta te izrade liste rizika, za svaki rizik se utvrđuje vjerojatnost njegova nastanka i procjena štete koju on može prouzročiti. Vjerojatnost rizika predstavlja vjerojatnost da će se rizik pojaviti prilikom korištenja sustava. Šteta ili kritičnost procjenjuje se analizom broja korisnika aplikacije koji će biti pogođeni rizikom. Primjerice, rizik od usporavanja sustava zbog izrade izvješća daleko je manje štetan od rizika da netko uništi podatke u sustavu (Popović, 2012:44).

Nakon što voditelji testiranja razumiju softver, mogu definirati opseg aktivnosti testiranja. To pripada komponenti strategije testiranja i razrađuje softverske ili hardverske komponente koje tvrtka namjerava testirati. Opseg testiranja može se promijeniti, ali početni plan ističe izvornu strategiju koju programeri planiraju slijediti. Također, naglašava komponente koje tvrtka ne namjerava testirati (Indeed, 2022).

Popović (2012:46-47) navodi kako se opseg posla sastoji od skupa zadataka koje treba izvršiti tijekom procesa testiranja. To je obavezan korak u planiranju procesa testiranja jer se treba definirati i dokumentirati što će se testirati i koje metode će biti korištene. Polazna točka za identifikaciju aktivnosti i vrsta testiranja su rizici identificirani kao dio aktivnosti procjene rizika. Svaki identificirani rizik za kojeg se smatra da zahtijeva testiranje, služi kao osnova za određivanje aktivnosti koje će se provesti tijekom testiranja. Osim osnovnih kategorija aktivnosti i zadataka, treba identificirati sve proizvode i dokumente koji podliježu provjeri. Vrijedno je zapamtiti da testiranje ne obuhvaća samo aplikaciju, već i zahtjeve, programski kod, dokumente pa i proces razvoja. Sve što se događa ili nastaje u projektu, a nema jamstva da će biti izrađeno najkvalitetnije, podliježe testiranju. Kategorije aktivnosti i proizvodi te procesi koji se testiraju predstavljaju opseg rada.

Sljedeći korak koji je potrebno učiniti je alokacija resursa. Ova faza stvara detaljnu analizu svih resursa potrebnih za završetak projekta. Resursi uključuju ljudske napore, opremu i svu infrastrukturu potrebnu za točno i sveobuhvatno testiranje. Ovaj dio planiranja testiranja odlučuje o potrebnim mjerama resursa za projekt, što uključuju broj testera i opreme. Također, pomaže voditeljima testiranja da formuliraju ispravan raspored i procjenu za projekt (Bose, 2022).

Popović (2012:48) iznosi da je, nakon što se odredi opseg poslova i aktivnosti koje će se provoditi tijekom testiranja, potrebno alocirati sve resurse potrebne za provedbu identificiranih aktivnosti i zadataka. Resursi su ljudi koji poduzimaju određene radnje, oprema u obliku servera, mobilni telefoni potrebni za testiranje, dokumentacija i tečajevi potrebni za stjecanje dodatnog znanja potrebnog u timu i tome slično.

Projekti koji ne upravljaju ispravno raspodjelom resursa često imaju poteškoća s drugim projektima, budući da ponekad isti ljudi moraju raditi na dva projekta u isto vrijeme. Planiranje

i alokacija resursa obuhvaća i planiranje znanja, treninga i obuke koje zaposlenici trebaju proći kako bi mogli izvršiti planirane zadatke i aktivnosti (Popović, 2012:48).

Naposljetku, dolazi se do posljednjeg koraka, a to je isticanje i testiranje rezultata. To uključuje dokumente i alate koje timovi održavaju za podršku aktivnostima testiranja. Voditelji testiranja mogu redovito ažurirati rezultate, ovisno o opremi koja je timovima potrebna. Mogu koristiti početne testne podatke za optimizaciju i promjenu zahtjeva prema potrebi. Razvojni timovi također održavaju izvješća o nedostacima, "*bugovima*", zapisnike pogrešaka i cjelokupnu izvedbu. Nakon što je testiranje završeno voditelji testiranja uspoređuju isporučene rezultate sa svojim izvornim očekivanjima te koriste te informacije kako bi utvrdili jesu li aktivnosti testiranja bile uspješne. To poboljšava planiranje resursa i učinkovitost aktivnosti testiranja (Indeed, 2022).

4.2. Dizajniranje testa

Dizajniranje testa predstavlja otkrivanje načina kako odrediti najbolji pristup testiranju sustava, tako da se uz što manje vremena i truda testira što veći dio sustava. Za svaku pojedinu vrstu testa koja se koristi u planu testiranja, moraju se navesti pojedinosti o tome kako se ti testovi izvršavaju. Postoje različite strategije za definiranje načina na koji se sustav testira. Strategije testiranja informacijskog sustava opisuju kako se sustav analizira da bi se odredilo kako testirati informacijski sustav. Strategije dizajna testa su metode koje se koriste za identifikaciju testnih slučajeva, analizu varijanti testnih slučajeva, određivanje koje podatke koristiti u testovima i koje očekivane rezultate pregledati nakon izvršenja testa. Strategije izvršenja testa predstavljaju metode koje se koriste za provjeru definiranih testnih slučajeva i testiranje softvera (Popović, 2012:103).

Cilj dizajna testa je sustavno identificirati i opisati sve moguće test slučajeve kako bi se ustanovilo što više problema dok je sustav još u razvojnom okruženju gdje ih je lakše riješiti. Postoje metode za identificiranje i opisivanje testnih slučajeva koje testiraju sve moguće varijante korištenja sustava. Testni slučajevi koje treba provjeriti, dokumentirani su tako da ih svi članovi tima mogu pregledati, promijeniti ili dopuniti u slučaju propusta (Popović, 2012:104).

Proces dizajna testova obuhvaća sve aktivnosti koje se trebaju odraditi kako bi se odredilo na koji će se način testirati sustav. Te aktivnosti obuhvaćaju (Popović, 2012:104):

1. analizu korisničkih zahtjeva,
2. validaciju korisničkih zahtjeva i specifikacije i
3. specifikaciju test dizajna.

Priya (2023) navodi da je dizajn testa proces koji definira kako se testiranje mora provesti. Uključuje proces identificiranja tehnika testiranja, testnih scenarija, testnih slučajeva, podataka i očekivanih rezultata. Također, testeri trebaju biti specifični u pogledu testnih slučajeva koje stvaraju, kao što je davanje specifičnih inputa, koraka i testnih podataka za svaki testni slučaj. Dizajn testa treba izraditi nakon što su definirani uvjeti testiranja i kada su dostupne odgovarajuće informacije za izradu testnih slučajeva visoke i niske razine.

4.3. Test slučajevi

Testni slučaj se definira kao skupina uvjeta pod kojima ispitivač utvrđuje radi li softverska aplikacija prema zahtjevima korisnika ili ne. Projektiranje testnog slučaja uključuje preduvjete, naziv slučaja, ulazne uvjete i očekivani rezultat. Testni slučaj je akcija prve razine i izvedena je iz testnih scenarija. Testni slučaj predstavlja dokument s pojedinostima koji sadrži sve moguće inpute (pozitivne kao i negativne) i navigacijske korake koji se koriste za proces izvođenja testa. Pisanje testnih slučajeva je jednokratni pokušaj koji se može koristiti u budućnosti u vrijeme regresijskog testiranja. Testni slučaj dalje detaljne informacije o strategiji testiranja, procesu testiranja, preduvjetima i očekivanom rezultatu. Izvršava se tijekom procesa testiranja kako bi se provjerilo izvršava li informacijski sustav zadatak za koji je razvijen ili ne (Javatpoint, n.d.b).

Druga definicija je da je testni slučaj skup preduvjeta koje treba slijediti s ulaznim podacima i očekivano ponašanje za provjeru funkcionalnosti sustava. Jednostavnim riječima, testni slučaj je kratak opis onoga što testirati i kako testirati. Postoje tri vrste funkcionalnih test slučajeva, a to su (Damodar, 2012):

- pozitivni test slučajevi,
- negativni test slučajevi i
- test validacije poslovanja.

Pozitivni test slučajevi predstavljaju pripremljene testove koji provjeravaju što sustav treba raditi, dok negativni test slučajevi predstavljaju testove koji provjeravaju što sustav ne bi trebao raditi. Naposljetku, test validacije poslovanja je u suštini testni slučaj pripremljen za provjeru uvjeta poslovanja (Damodar, 2012).

Osnovni test slučajevi se identificiraju pregledom funkcionalnosti koje su u sustavu, na osnovu čega se za svaku funkcionalnost koja se testira, opisuje što treba provjeriti i testirati kako bi se dokazalo da sustav radi prema očekivanom (Popović, 2012:113).

Kao primjer testnog slučaja može se uzeti testiranje prijave u sustav. Korisnici sustava u formu za prijavu unose svoje korisničko ime i lozinku. Ako su uneseni podaci ispravni, korisnik je identificiran i odobrava mu se pristup, u protivnom se ponovno pojavljuje forma za prijavu. Potrebno je identificirati test slučajeve za navedenu funkcionalnost (Popović, 2012:113).

U ovom primjeru se može identificirati jedna funkcionalnost za koju se može definirati nekoliko testnih slučajeva. Potreban je minimalno jedan pozitivan slučaj kojim se provjerava radi li funkcionalnost. Također, postoji i jedan negativan testni slučaj kojim se provjerava odobrava li forma za prijavu pristup i u slučaju neispravnih podataka. U slučaju identifikacije dodatnih rizika, kao što je upad u sustav, ovoj funkcionalnosti se pridružuje još jedan testni slučaj kojim se provjerava sigurnost. Primjer testnih slučajeva za funkcionalnost prijave u sustav prikazan je u sljedećoj tablici (Popović, 2012:113).

Tablica 1. Testni slučajevi za funkcionalnost prijave u sustav

Testni slučaj		Akcija
ID	Naziv	Opis
1	Uspješna prijava	Provjeriti je li unosom ispravnih korisničkih imena i lozinki omogućeno puštanje korisnika u sustav.
2	Neuspješna prijava	Provjeriti je li unosom neispravnih korisničkih imena i lozinki, korisnik vraćen na početnu formu za prijavu.
3	Otpornost na upad u sustav	Provjeriti može li korisnik ući u sustav bez validnih podataka

Izvor: Vlastita izrada autora prema Popović (2012)

Iz priloženog se može vidjeti kako su osnovni testni slučajevi samo lista stvari koje treba provjeriti bez konkretnih uputa kako to treba učiniti. Osnovnim testnim slučajevima se može verificirati da osnovna funkcionalnost sustava postoji i radi, ali oni ne mogu garantirati da su obuhvaćene sve alternative te da su otkriveni svi potencijalni problemi (Popović, 2012:113-114).

5. Alati za testiranje softvera

Danas na tržištu postoji velik broj dostupnih alata za testiranje softvera od kojih su neki alati otvorenog koda (engl. *open-source*) i oni koji se plaćaju. Značajna razlika između *open-source* i plaćenog alata je u tome što *open-source* imaju ograničene značajke, dok plaćeni alati ili komercijalni alati nemaju ograničenja za značajke. Odabir alata ovisi o zahtjevima korisnika. Alati za testiranje softvera mogu se kategorizirati ovisno o licenciranju, korištenju tehnologije, vrsti testiranja i slično. Uz pomoć alata za testiranje omogućeno je poboljšanje performansi softvera, isporuka proizvoda visoke kvalitete i smanjenje trajanja testiranja. Alati za testiranje softvera mogu se podijeliti na (Javatpoint, n.d.c):

- alati za upravljanje testiranjem,
- alati za praćenje *bugova*,
- alati za automatsko testiranje,
- alati za testiranje performansi,
- alati za testiranje na različitim preglednicima,
- alati za integracijsko testiranje,
- alati za jedinično testiranje,
- mobilni/android alati za testiranje,
- GUI alati za testiranje
- alati za testiranje sigurnosti.

5.1. Alati za upravljanje testiranjem

Alati za upravljanje testiranjem koriste se za praćenje svih aktivnosti testiranja, brzu analizu podataka, upravljanje ručnim i automatiziranim testnim slučajevima, različitim okruženjima te planiranje i održavanje ručnog testiranja. Ovi su alati najprikladniji za planiranje, bilježenje pogrešaka, praćenje i analizu. Neki od najčešće korištenih alata za upravljanje testiranjem su sljedeći (Javatpoint, n.d.d):

- Quality center
- RTH
- Testpad
- Test Monitor
- PractiTest

5.2. Alati za praćenje *bugova*

Alati za praćenje pogrešaka (engl. *bugova*) koriste se za praćenje ispravaka pogrešaka i osiguranje kvalitetnog proizvoda. Ovi alati pomažu pri pronalaženju pogrešaka u fazi testiranja kako bi se mogli dobiti podaci bez pogrešaka u proizvodnom poslužitelju. Najčešće korišteni alati za praćenje *bugova* su (Javatpoint, n.d.e):

- Jira
- Bugzilla
- BugNet
- Redmine
- Mantis
- Trac
- Backlog

5.3. Alati za automatsko testiranje

Automatizirano testiranje koristi se za promjenu ručnih testnih slučajeva uz testnu skriptu uz pomoć nekih alata za automatizaciju. Na tržištu postoje različite vrste alata za automatsko testiranje. Neki od najčešće korištenih alata za automatsko testiranje su sljedeći (Javatpoint, n.d.f):

- Selenium
- Watir
- QTP
- Telerik Studio
- Testim
- Applitools
- Applitools

5.4. Alati za testiranje performansi

Kada je potrebno izmjeriti opterećenje, stabilnost, vrijeme odaziva aplikacije, potrebni su alati za testiranje performansi (opterećenja). Alati za testiranje performansi mogu biti *open-source* i komercijalni. Najčešće korišteni alati za testiranje performansi su upravo sljedeći (Javatpoint, n.d.g):

- Apache JMeter
- LoadRunner[HP]

- LoadNinja
- WebLOAD
- LoadComplete
- NeoLoad
- LoadView

5.5. Alati za testiranje na različitim preglednicima

Alati za testiranje na različitim preglednicima pomažu pri osiguranju da web aplikacija ispravno radi u više vrsta preglednika. Ovi alati će se pokrenuti kada i poslužitelj i klijent pristupaju web aplikaciji u više web preglednika. Uz pomoć ovih alata, može se izvršiti testiranje kompatibilnosti putem različitih preglednika za aplikaciju. Ponekad testiranje softvera u jednom web pregledniku nije dovoljno; zato su potrebni alati za testiranje na različitim preglednicima. Na tržištu su prisutni različiti alati, neki od njih su (Javatpoint, n.d.h):

- LambdaTest
- SauceLabs
- CrossBrowser Testing
- BrowserStack
- GhostLab
- Browsera

5.6. Alati za integracijsko testiranje

Alati za integracijsko testiranje koriste se za testiranje sučelja između modula i pronalaženje pogrešaka; te se pogreške mogu dogoditi zbog integracije više modula. Glavni cilj ovih alata je osigurati da određeni moduli rade prema potrebama i zahtjevima klijenta. Za integracijsko testiranje najčešće se koriste sljedeći alati (Javatpoint, n.d.i):

- Citrus
- FitNesse
- TESSY
- Protractor
- Rational Integration tester

5.7. Alati za jedinično testiranje

Kada je potrebno pronaći i provjeriti autentičnost određenog modula ili jedinice koda, tada se koriste alati za jedinično testiranje. Uz pomoć ovih alata omogućena je izgradnja sigurnog dizajna i dokumentacije te smanjenje broja pogrešaka. Općenito, testiranje jedinica je ručni proces, ali neke organizacije su automatizirale testiranje jedinica uz pomoć ovih alata. Neki od najpoznatijih alata za jedinično testiranje su (Javatpoint, n.d.j):

- NUnit
- JUnit
- TestNG
- Mockito
- PHPUnit

5.8. Mobilni/android alati za testiranje

Za testiranje mobilne aplikacije potrebni su alati koji pomažu provjeriti upotrebljivost, funkcionalnost, sigurnost i dosljednost aplikacije. Mobilne aplikacije se široko koriste na platformama Android i iOS, što povećava pouzdanost klijenta prema aplikacijama. Alati za android/mobilno testiranje su (Javatpoint, n.d.k):

- Appium
- Calabash
- Testdroid
- Kobiton
- TestComplete
- TestingBot

5.9. GUI alati za testiranje

GUI (engl. *Graphical User Interface*) alati za testiranje koriste se za pronalaženje nedostataka koji su se dogodili u fazi projektiranja, a koji poboljšavaju kvalitetu softvera. Testira se aplikacija na temelju performansi, što je povezano s radnjama miša i tipkovnice, te nekim od GUI stavki kao što su gumbi, alatne trake, dijaloški okviri, trake izbornika i polja za uređivanje. Nekoliko bitnih strategija koje se mogu izvesti prilikom GUI testiranja su provjera valjanosti navigacije, provjera ispravnosti integriteta podataka, provjera situacija upotrebljivosti te provjera numeričkih formata polja datuma. Najčešće korišteni GUI alati za testiranje su (Javatpoint, n.d.l):

- Eggplant
- AutoIT
- Ranorex Studio
- Squish
- RIATest

5.10. Alati za testiranje sigurnosti

Alati za testiranje sigurnosti koriste se kako bi se osiguralo da su podaci spremljeni i da im neovlašteni korisnici ne mogu pristupiti. Također, pomažu pri pronalasku nedostataka i sigurnosnih propuštanja sustava u ranijoj fazi. Ovi alati mogu raditi na aspektima autorizacije, povjerljivosti, provjere autentičnosti i dostupnosti. Zahvaljujući ovim alatima može se izbjeći gubitak relevantnih informacija, iznenadni kvarovi, dodatni troškovi potrebni za popravak aplikacije nakon napada te nepredvidiv rad aplikacije. Na tržištu su dostupni sljedeći alati za testiranje sigurnosti (Javatpoint, n.d.m):

- SonarQube
- ZAP
- Netsparker
- Arachni
- IronWASP

6. Zaključak

Neprekidno unapređivanje kvalitete informacijskih sustava postaje ključan čimbenik za sve poslovne subjekte i organizacije koje se žele istaknuti kao konkurenti na tržištu. Kroz sustavno testiranje, identificiraju se pogreške i nedostaci, čime se smanjuje rizik od neispravnosti i neispunjenja korisničkih zahtjeva. Osim testiranja, osiguranje kvalitete pruža cjelovit pristup koji obuhvaća aktivnosti kao što su planiranje, izvođenje, provjera i djelovanje sustava kako bi se postigla visoka kvaliteta softvera.

Uspješno testiranje informacijskih sustava i sam proces osiguranja kvalitete donose brojne prednosti. Najprije, to se odnosi na smanjenje rizika od toga da sustav ne bude funkcionalan i pouzdan, zatim na smanjenje rizika od financijskih gubitaka. Nadalje, prednosti su i povećanje povjerenja klijenata i korisnika te poboljšanje ugleda samog poslovnog subjekta.

Ipak, postoje i određeni izazovi koji se tiču testiranja informacijskih sustava i procesa osiguranja kvalitete. Sama kompleksnost korisničkih zahtjeva, brze promjene u poslovanju i tehnologiji te ograničeni resursi su neke od prepreka s kojima se poslovni subjekti i organizacije suočavaju. Zbog toga je iznimno važno uložiti u pravilno osposobljavanje i edukaciju timova za testiranje te kontinuirano pratiti i provoditi najnovije trendove u poslovanju.

Također, važno je napomenuti kako se testiranje informacijskih sustava i osiguranje kvalitete kontinuirano primjenjuju i provode tijekom cijelog životnog ciklusa softvera. Redovito ažuriranje i praćenje testnih planova i prilagodba metoda i alata za testiranje su od iznimne važnosti za uspješno održavanje visokih standarda kvalitete softvera.

Literatura

1. Bose, S (2022). *Test Planning: A Detailed Guide*. Dostupno na: https://www.browserstack.com/guide/test-planning [Pristupljeno: 7. lipnja 2023.]
2. Damodar, C (2023). *Practical Software Testing - eBook.* [Online] Software Testing Help. Dostupno na: https://www.softwaretestinghelp.com/practical-software-testing-new-free-ebook-download/ [Pristupljeno: 15. svibnja 2023.]
3. Damodar, C (2012). Manual Testing Help eBook. [Online] Software Testing Help. Dostupno na: https://www.softwaretestinghelp.com/practical-software-testing-new-free-ebook-download/ [Pristupljeno: 7. lipnja 2023.]
4. Goyal, Y. (n.d.). *Software Quality Assurance*. Dostupno na: https://www.educba.com/software-quality-assurance/ [Pristupljeno: 4. travnja 2023.]
5. Indeed, (2023). *System Testing Guide: What It Is and What It Verifies*. Dostupno na: https://www.indeed.com/career-advice/career-development/system-testing [Pristupljeno: 17. svibnja 2023.]
6. Indeed, (2022). *Why is test planning important? (How to create a test plan)*. Dostupno na: https://uk.indeed.com/career-advice/career-development/test-planning [Pristupljeno: 7. lipnja 2023.]
7. Javatpoint, (n.d.a). *Acceptance testing*. Dostupno na: https://www.javatpoint.com/acceptance-testing [Pristupljeno: 17. svibnja 2023.]
8. Javatpoint, (n.d.b). *Test Case*. Dostupno na: https://www.javatpoint.com/test-case [Pristupljeno: 7. lipnja 2023.]
9. Javatpoint, (n.d.c). *Software Testing Tools*. Dostupno na: https://www.javatpoint.com/software-testing-tools [Pristupljeno: 8. lipnja 2023.]
10. Javatpoint, (n.d.d). *Test Management Tool*. Dostupno na: https://www.javatpoint.com/test-management-tool [Pristupljeno: 8. lipnja 2023.]
11. Javatpoint, (n.d.e). *Defect/Bug testing tool*. Dostupno na: https://www.javatpoint.com/defect-or-bug-tracking-tool [Pristupljeno: 8. lipnja 2023.]
12. Javatpoint, (n.d.f). *Automation testing tool*. Dostupno na: https://www.javatpoint.com/automation-testing-tool [Pristupljeno: 8. lipnja 2023.]
13. Javatpoint, (n.d.g). *Performance testing tools (Load testing tools).* Dostupno na: https://www.javatpoint.com/performance-testing-tools [Pristupljeno: 8. lipnja 2023.]
14. Javatpoint, (n.d.h). *Cross-browser testing tools.* Dostupno na: https://www.javatpoint.com/cross-browser-testing-tools [Pristupljeno: 8. lipnja 2023.]

15. Javatpoint, (n.d.i). *Integration testing tools*. Dostupno na: https://www.javatpoint.com/integration-testing-tools [Pristupljeno: 8. lipnja 2023.]
16. Javatpoint, (n.d.j). *Unit testing tools.* Dostupno na: https://www.javatpoint.com/unit-testing-tools [Pristupljeno: 8. lipnja 2023.]
17. Javatpoint, (n.d.k). *Mobile Testing Tools*. Dostupno na: https://www.javatpoint.com/mobile-testing-tools [Pristupljeno: 8. lipnja 2023.]
18. Javatpoint, (n.d.l) *GUI testing tools*. Dostupno na: https://www.javatpoint.com/gui-testing-tools [Pristupljeno: 8. lipnja 2023.]
19. Javatpoint, (n.d.m). *Security testing tools*. Dostupno na: https://www.javatpoint.com/security-testing-tools [Pristupljeno: 8. lipnja 2023.]
20. Popović, J. (2012). *Testiranje softvera u praksi*. Beograd: Računarski fakultet
21. Priya, Y. (2023). *Test Design in Software Testing – A Comprehensive Guide*. Dostupno na: https://testsigma.com/blog/test-design/ [Pristupljeno: 7. lipnja 2023.]
22. Software Testing Fundamentals (2023). *Integration Testing*. Dostupno na: https://softwaretestingfundamentals.com/integration-testing/ [Pristupljeno: 17. svibnja 2023.]
23. Software Testing Fundamentals (2022). *Static Testing*. Dostupno na: https://softwaretestingfundamentals.com/static-testing/ [Pristupljeno: 19. svibnja 2023.]
24. Stackify (2020). Best Software Testing Methods. Dostupno na: https://stackify.com/best-software-testing-methods/ [Pristupljeno: 19. svibnja 2023.]
25. TechTarget (n.d.). What is unit testing?. Dostupno na: https://www.techtarget.com/searchsoftwarequality/definition/unit-testing [Pristupljeno: 15. svibnja 2023.]
26. Tilley, S. (2019). *Systems Analysis and Design, Twelfth Edition*. Boston: Cengage Learning
27. Tutorials Point (n.d.). *Software Testing – Methods*. Dostupno na: https://www.tutorialspoint.com/software_testing/software_testing_methods.htm# [Pristupljeno: 19. svibnja 2023.]
28. Wigmore, I. (2015). PDCA (plan-do-check-act). Dostupno na: https://www.techtarget.com/whatis/definition/PDCA-plan-do-check-act [Pristupljeno: 16. lipnja 2023.]

Popis tablica