INTERNACIONALNI UNIVERZITET TRAVNIK U TRAVNIKU
FAKULTET POLITEHNIČKIH NAUKA TRAVNIK U TRAVNIKU

ZAVRŠNI RAD

TESTIRANJE MOBILNE APLIKACIJE

Prof. dr Mladen Radivojević

Emina Mujić
PT-122/16-II

Travnik, 2019. godine

SADRŽAJ

UVOD

Mobilni uređaji su kroz posljednjih nekoliko godina postali jedan od najvažnijih vidova komunikacije u svijetu. Prvobitni cilj mobilnih uređaja jeste bila komunikacija, međutim danas postoje uređaji sa širokim spektrom upotrebe. Upotrebljavaju se za komunikaciju, posao, zabavu, obrazovanje, vijesti, itd. Nekad su mobilni uređaji bili stvar luksuza, ali danas rijetko ko ne posjeduje jedan takav uređaj. Mobilni uređaji su od primarnih funkcija evoluirali u mali pokretni računar sa vlastitim operativnim sistemom, mnogim različitim funkcijama i aplikacijama, a među svim tim aplikacijama postoji velika konkurencija. Brojne mogućnosti i praktičnost mobilnih uređaja su dovele do nezaustavljivog rasta u razvoju aplikacija za pokretne uređaje, a tržište aplikacija je postalo jedno od najbrže rastućih grana industrije. Ponuda aplikacija je svakim danom sve veća i danas možemo naći aplikaciju za gotovo sve što poželimo.

Sve to je dovelo do povećanja važnosti testiranja mobilnih aplikacija. Bitno je imati aplikaciju koja radi bez greške i koja je bolja od konkurentskih aplikacija, a to se može osigurati jedino konstantnim testiranjem aplikacije. Konstantno testiranje se može izvoditi ručno, ali se sve više ide prema automatizaciji testiranja gdje se sastavljaju testovi koji se konstantno izvršavaju tokom svih faza razvojnog procesa aplikacije i osiguravaju njen rad bez grešaka.

1. 1. Zadatak diplomskog rada

U ovom diplomskom radu ću nastojati opisati proces i načela testiranja mobilnih aplikacija. Nakon toga će se prikazati, na primjeru napravljene mobilne aplikacije za Android i iOS uređaje, automatizacija testiranja.

U drugom poglavlju ovog rada opisati ću mobilne uređaje, različite operativne sisteme kao i vrste aplikacija koje se pojavljuju na njima. Treće poglavlje se bavi testiranjem. Odgovara na pitanja zašto je testiranje bitno, kako se to izvodi testiranje i upoznajemo se sa različitim vrstama testiranja. Četvrto poglavlje se bavi automatizacijom testiranja. Navode se različite tehnologije i načini automatiziranja testiranja. Peto poglavlje sadrži proces pisanja aplikacija i testova, opis napisanih aplikacija i testova te demonstraciju automatiziranog testiranja.

2. MOBILNI UREĐAJI

Prije nego što krenem na obradu teme testiranja aplikacija, potrebno je objasniti današnje mobilne uređaje, tj. pametne mobitele i koliko su oni zapravo važni, različite dostupne operativne sisteme kao i aplikacije s kojima se susrećemo na ovim uređajima. Prilikom samog testiranja aplikacije potrebno je znati kako funkcioniraju različiti operativni sistemi i aplikacije na njima kako bi se što učinkovitije provelo testiranje.

2.1. Pametni mobiteli

Prvi pametni telefoni izašli su na tržište davne 1994. Godine. Kreirali su ih inženjeri kompanija IBM i BellSouth, koji su prvi došli na ideju da naprave kombinaciju mobilnog telefona i ličnog digitalnog asistenta (PDA uređaja)[1].Bio je opremljen ekranom na dodir baš kao i današnji uređaji. Ovaj mobitel je također imao nekoliko aplikacija. Imao je mogućnost slanja i primanja email-ova, kalkulator, zabilješke, navigaciju i vijesti. Pametni mobiteli kakvi su danas prisutni počeli su se razvijati 2007. godine kada je Apple predstavio prvi iPhone.

Niti jedan uređaj nije doživio tako široku i brzu primjenu i upotrebu kao mobilni telefon. Adaptacija mobilnih telefona kod mladih globalni je fenomen posljednjih nekoliko godina. Danas je taj uređaj integralni dio svakodnevnog života posebno populacije mladih, te je glavni oblik elektronske komunikacije. Stoga možemo reći kako je mobilni telefon napravio pomak od tehnološkog prema društvenom alatu.

Slika 1. Razvoj mobilnih telefona
Photo source: www.mtel.ba

Prema istraživanju analitičke kompanije Canalys, smartfon tržište u Evropi su tokom 2018. godine karakterisala dva glavna trenda – kineski proizvođači su preuzeli trećinu tržišta, a premijum modeli su doživeli veliki pad popularnosti Zapadna Evropa je najveća podregija, gde je isporučeno **128 miliona** telefona, od ukupno 197 miliona pametnih telefona. Evropsko tržište u cjelini je doživjelo pad od 4% na godišnjem nivou, dok je Zapad generalno imao najmanju isporuku još od 2013. Godine[2].

Samsung i **Apple** i dalje zadržavaju svoje vodeće pozicije na tržištu, ali su obje kompanije doživjele pad prodaje u Evropi. Na trećem mjestu nalazi se **Huawei**, a potom slijede **Xiaomi** i **Nokia**. Trenutni vodeći proizvođač pametnih telefona je Samsung sa ostvarenim tržišnim udelom od 22,9 odsto i prodatih oko 98 miliona uređaja. Nokia zauzima drugu poziciju sa 19,2 odsto tržišnog učešća i prodatih 82 milona uređaja. Apple ima prodatih 23,5 miliona pametnih telefona što čini 5,5 odsto tržišta[3].

European smartphone shipments and annual growth
Canalys Smartphone Market: Q4 2018

Vendor	Q4 2018 shipments (million)	Q4 2018 Market share	Q4 2017 shipments (million)	Q4 2017 Market share	Annual growth
Samsung	16.2	28.7%	16.4	28.3%	-1.0%
Apple	14.7	26.0%	15.5	26.8%	-5.1%
Huawei	13.3	23.6%	8.6	14.8%	+55.7%
Xiaomi	3.4	6.0%	2.1	3.6%	+62.0%
HMD Global	1.3	2.4%	1.6	2.8%	-18.3%
Others	7.6	13.4%	13.8	23.7%	-44.8%
Total	56.6	100.0%	58.0	100.0%	-2.3%

canalys

Source: Canalys Smartphone Analysis (sell-in shipments), February 2019

Slika 2. Zastupljenost proizvodjača mobilnih uređaja
Photo source: www.balkandroid.com

Iz slike 2. vidimo da su najrasprostranjeniji Samsung mobilni uređaji koje slijedi Apple, pa Huawei i Xiaomi. Samsung, Xiaomi i Huawei kao temelj operativnog sistema upotrebljavaju Android. Iz te se podjele može doći do zaključka da su Android i iOS najrasprostranjeniji operativni sistemi. Postoji jako puno mobilnih uređaja s različitim operativnim sistemima i njihovim verzijama pa zbog toga testiranje postaje komplicirano. Aplikacija se može ponašati drugačije na različitim uređajima, ali se to sve ne može testirati. Zato je bitno uzeti u obzir statistike o tome koje uređaje koristi najviše korisnika aplikacije i na njima provesti testiranja. Najrasprostranjeniji operativni sistemi su Android i iOS pa će se u ovome radu koncentrirati na testiranje aplikacija na ta dva operativna sistema. U trećoj slici prikazane su sličnosti i razlike između ta dva operativna sistema i njihove karakteristike.

3. OPERATIVNI SISTEMI

Najrasprostranjeniji operativni sistemi su Android i iOS pa će se u ovome radu koncentrirati na testiranje aplikacija na ta dva operativna sistema. U trećoj slici prikazane su sličnosti i razlike između ta dva operativna sistema i njihove karakteristike.

	iOS	Android
Računar	Apple Mac OSX	Bilo koji računar
Razvojno okruženje	Xcode	Eclipse, netbeans, Idea intellij
SDK	iOS SDK	Java i Android SDK
Jezik	Objective C	Java

Slika 3. Poređenje IOS i Android operativnog sistema
Photo source: www.etf.bg.ac.rs

Ove dvije platforme su slične ali ne i jednake u nekim funkcijama kao što su prepoznavanje glasa, kalendar, email, video pozivi, mape i slično, ali su u svojoj jezgri drugačije.

3. 1. Android OS

Google Android je prvi otvoreni operacijski sistem za pokretne uređaje (mobilni telefoni, tableti, netbook računara, Google TV) pokrenut od strane Google Inc. i vođen od strane Open Handset Alliance - grupe koja danas broji preko 80 tehnoloških kompanija, čiji je cilj ubrzati inovacije na području operativnih sistema za pokretne uređaje, a samim time ponuditi krajnjim kupcima bogatije, jeftinije i bolje iskustvo korištenja[4].

Android je modularan i prilagodljiv pa tako postoje slučajevi njegovog prenošenja na razne uređaje. Sadrži operativni sistem, međuopremu i razne aplikacije za pokretne uređaje. Također, sadrži i velik broj API-ja , koji nezavisnim proizvođačima omogućuju izradu aplikacija[5].

Android upotrebljava većina proizvođača mobilnih uređaja kao bazni OS. Posljednja verzija Android operativnog sistema izašla je sredinom 2018 godine i naziva se Android 9 Pie. Deveta po redu verzija, Android Pie, donijela je mnogo novih funkcija koje se temelje na vještačkoj inteligenciji da bi se u potpunosti prilagodio korisničkim potrebama[6].

Android ima problem jer podržava jako puno različitih uređaja, a ti uređaji ne mogu ići u korak s razvojem operativnog sistema. Aplikacije za Android pišu se u Javi, kompajliraju u Java bajt kod, a nakon toga u Dalvik ili ART bajt kod. Dalvik upotrebljava tačno na vrijeme (engl. just in time) kompajliranje, što znači da se kod kompajlira netom prije izvršavanja zbog čega upotrebljava manje resursa. ART upotrebljava prijevremeno kompajliranje (engl. ahead of time), što znači da se aplikacija kompajlira samo jednom prilikom instalacije. ART-om se smanjuje sporost pokretanja aplikacije. Sav pristup hardveru i nekim sistemski funkcionalnostima ide preko ART/Dalvik-a. Time se osigurava da se programeri aplikacija ne moraju brinuti o hardveru[7].

3. 2. iOS OS

iOS (prije *iPhone OS*) je operativni sistem kojeg je razvila američka tvrtka Apple Inc. za iPhone. Prvi put predstavljen je na Macworld Conference & Expou 9. januara 2007. godine istovremeno s mobilnim uređajem iPhoneom, a pušten je u prodaju u maju iste godine. Prema podatcima iz 2015., iOS se po zastupljenosti mobilni operativnih sistema nalazi na drugom mjestu (ispred iOS-a je Android s 52%).

Trenutno najnovija verzija iOS-a je iOS 12. Izvorno je dostupan samo na seriji pametnih telefona iPhone, ali danas se koristi i u seriji multimedijalnih uređaja iPod Touch i seriji tableta iPad. iOS 13 dobit će samo uređaji iz serija iPhone i iPod Touch, dok će uređaje iz serije iPad primiti zaseban operacijski sustav temeljen na iOS-u, iPadOS[8].

U junu 2010. godine, Eplov iPhone OS preimenovan je u "iOS". Zaštitni znak "iOS" je koristio Cisco više od jedne decenije za svoj operativni sistem, korišćen na njihovim ruterima. Da bi se izbegla bilo kakva potencijala tužba, Apple je licencirao "IOS" od strane kompanije Cisco[9].

Sistem je u potpunosti zatvoren i mogu ga upotrebljavati samo Apple proizvodi. Kako bi programer mogao pisati aplikacije za iOS uređaje on mora posjedovati računalo s macOS-om jer je razvojna okolina (Xcode) dostupna samo za korisnike Apple proizvoda. Aplikacije za iOS pišu se u Objectiv-C-u ili u Swift-u. Swift se kompajlira u mašinski kod i najbolje funkcionira kad se upotrebljava prijevremeno kompajliranje. Operativni sistem iOS se konstantno unapređuje i izdaju se nove verzije. Trenutno je posljednja verzija iOS 12. Za razliku od Androida, posljednja verzija iOS-a je i najrasprostranjenija. Razlog tome je što Apple podržava samo svoje proizvode i rade kompatibilnost prema starijim verzijama za nove verzije operativnog sistema tako da ih i stariji uređaji mogu koristiti[10].

U nastavku ce se prikazati trgovine (online platforme za distribuciju aplikacija) iOS i Android OS-a

3. 3. Google Play

GooglePlay (ranije Android Market) Googleova je mrežna trgovina aplikacija, muzike, filmova i ostalih digitalnih sadržaja. Predstavljen je u augustu 2008. godine, a njegovo korištenje počinje od oktobar 2008. godine. Pojava prvih komercijalnih aplikacija započinje od strane britanskih i američkih programera od februara 2009. Dakle, GooglePlay nastao je 6. marta 2012. spajanjem Android Marketa, Google muzike i Google trgovine eKnjigama[11].

Nisu sve aplikacije besplatne, ali veći broj jeste. Za razliku od prodavnice aplikacija za iPhone gdje se za većinu aplikacija mora izdvojiti određena suma novca, PlayStore je totalna suprotnost. Velika većina aplikacija je besplatna, a mali broj se plaća, to su obično kvalitetne 3D igrice i još dosta toga bez čega se može.Google Play je jedna od najbitnijih aplikacija jer omogućava preuzimanje svih ostalih aplikacija i čini Android uređaj funkcionalnim.

3. 4. App Store

App Store predstavljen je 10. jula 2008. godine, godinu dana nakon što je prvi iPhone ugledao svjetlo dana. Isprva nije nudio druge aplikacije osim Appleovih, a i one su bile tek smanjena verzija do tada korištenih na računalima. Tih početnih nešto više od 500 aplikacija naraslo je u međuvremenu na više od dva miliona. U deset godina zabilježeno je više od 130 milijardi preuzimanja aplikacija[12].

Trgovina korisnicima omogućuje pregledavanje i preuzimanje aplikacija razvijenih pomoću Appleovog razvojnog kompleta za iOS softver. Aplikacije se mogu preuzimati na pametnom telefonu iPhone , ručnom računalu iPod Touch ili tablet računaru iPad , a neke se mogu prenijeti na pametni sat Apple Watch ili 4. generacije ili novije Apple TV-ove kao ekstenzije iPhone aplikacija[13].

3. 4. Google Play vs App Store

Uspoređujući ove dvije trgovine potrebno je napomenuti da su to dvije trgovine koje su se razvile u isto doba te se u mnogo toga razlikuju u samoj ponudi aplikacija. U ne tako davnoj 2008. godini nastale su ove najviše popularne trgovine aplikacijama u svijetu. Google Play trgovina pojavila se pod imenom Android Market i kasnije je dobila ime kakvo ima i danas. Uključujući kupovinu časopisa, virtualnih knjiga, aplikacija i igara dostigla je svoj vrhunac, iako u prvobitnome stanju nije bila toliko popularna od strane korisnika mobilnih terminalnih uređaja sa android OS-om. Svoj vrhunac nikada nije doživjela jer stalno raste i nikada neće prestati rast jer Android zastupa 73,38% OS-a cijelog svijeta, prema StatsCounteru. Ima više milijardu korisnika u oko 190 zemalja, sa oko 40 miliona pjesama, 5 miliona knjiga i stalno rastući broj filmova. Slična je priča i sa aplikacijama na App Store-u, njihov rast je nezamjetan i nudi korisnicima izbor besplatne i plaćene aplikacije: Mnoga istraživanja pokazala su da iOS korisnicima ne smeta plaćati aplikacije i potrošiti 10 dolara svaki mjesec. Slučaj s Google Play je potpuno drugačija. Zato programeri često nude besplatne aplikaciju, kako bi privukli više korisnika. S druge strane, aplikacije sa Google Play su aplikacije pune oglasa.

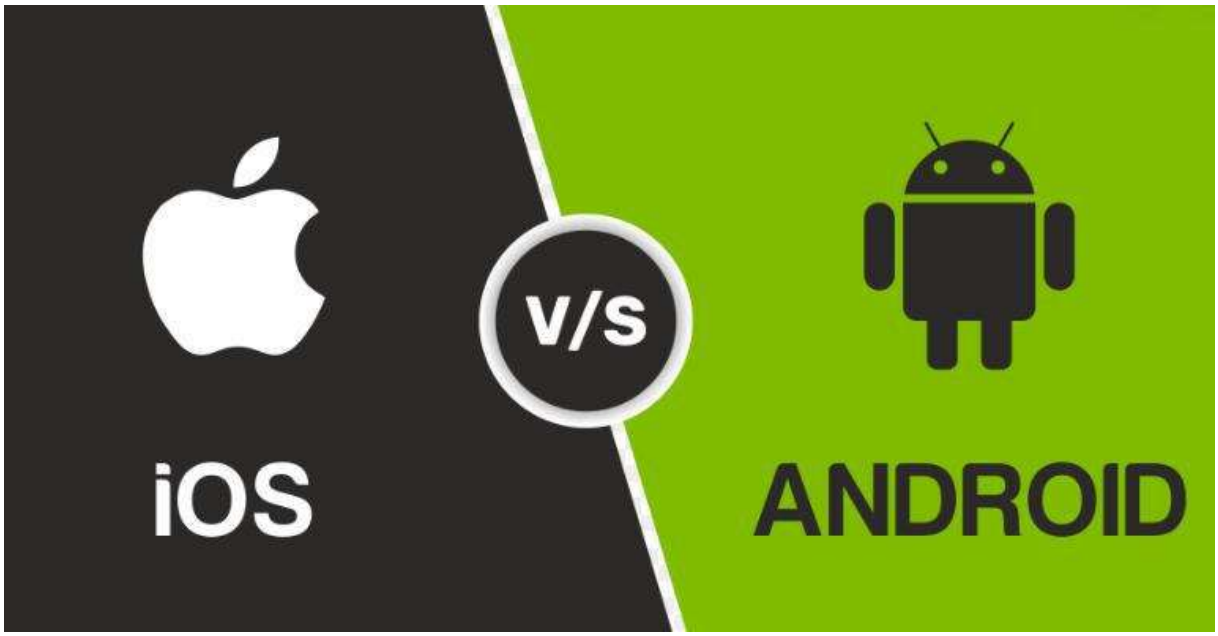
Osim toga, Google Play omogućuje programerima ponuditi test verziju aplikacije kako bi ih korisnici mogli testirati i dati povratne informacije. Te aplikacije su u većini slučajeva stabilne. Kod iOS korisnika, može se dobit samo potvrda u obliku ugradnje i ispitivanja od strane dobavljača.

Pretraživanje aplikacija: Pretraživanje aplikacija na Google Play je učinkovitije. Google ima bolji algoritam za pretraživanje i opciju koja omogućuje vidjeti pregled programa, komentare na društvenoj mreži, preporuke i slično. Apple App Store gura aplikacije s 4 zvjezdice ili više ocjena prema vrhu kod pretraživanja što opet omogućava korisnicima brže pronalaženje boljih aplikacija prema rejtingu. Također je jednostavnije naći primjenu po ključnoj riječi neko u Google Play trgovini.

Povrat kupljenih aplikacija: Što se tiče kupljenih aplikacija ako vam se ne sviđa što ste kupili, Google Play omogućava poništiti kupnju u roku od dva sata. iOS korisnici to mogu učiniti najviše 90 dana od kupnje aplikacije uz odgovarajući razlog upućen developeru.

Sigurnost: App Store je provjerava relativno detaljno svaki zahtjev te sve aplikacije moraju slijediti iste smjernice. Međutim što se tiče sigurnosti i pri pregledu koda aplikacija mogu se provući propusti. Na drugoj strani Google ne pregledava aplikacije prije objave, ali ih inertno skida ako se ukaže da imaju neki maliciozni kod. Najviše malicioznog koda krije se u igrama[14].

Koja je trgovina bolja dokazuju samo brojevi prema kojima je Google Play najzastupljeniji u količini preuzetih aplikacija, količini aplikacija koje se tamo nalaze i po posjećenosti zbog svojjih besplatnih aplikacija od strane razvojnih timova, najboljem pretraživanju aplikacija prema gotovo svim kriterijima, dostupnošću i jednostavnosti, ali opet i vrlo lijepom izgledu.



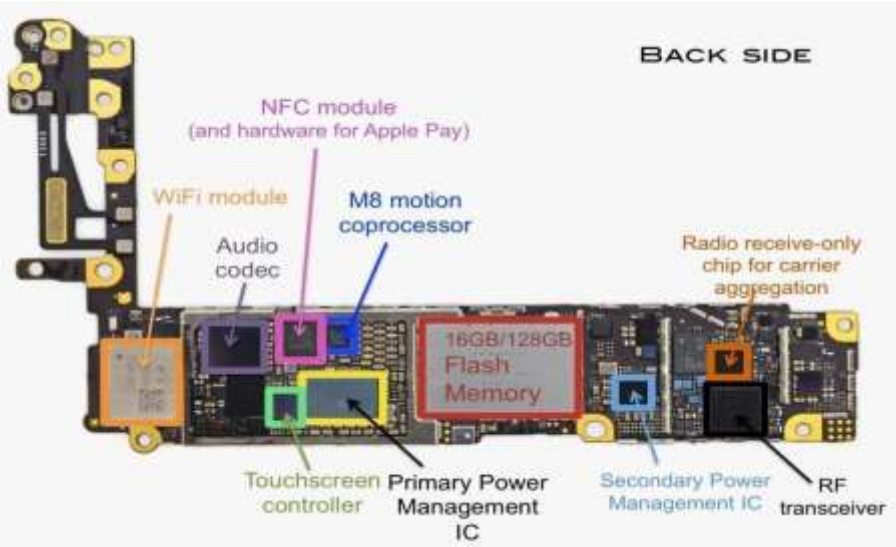
Slika 4. IOS vs Android
Photo source: www.forbes.com

4. HARDVER

Pametni su mobiteli u svojoj srži minijaturni računari. Sadrže jednake osnovne dijelove kao oni, na primjer memoriju i procesor. Razlika je u tome što se u mobitelima nalazi puno senzora [15] (npr. za osvjetljenje, udaljenost, ubrzanje, nagib…), uglavnom imaju dvije kamere, različite tehnologije za povezivanje (NFC, bluetooth, GPS, WiFi, podatkovni promet…) i, ovisno o proizvođaču, različite opcije za povezivanje (audio jack, mini USB, USB type-C, lightning…).

Najveći dio pametnog mobitela je ekran na dodir. Postoje dvije vrste ekrana na dodir. Prva je rezistivni ekran koji se sastoji od dva sloja konduktivnog materijala između kojih je praznina koja se ponaša kao otpornik. Kada se ekran dodirne, ta se dva sloja dodirnu i na tom se mjestu provodi struja. Druga vrsta je kapacitivni ekran koji se sastoji od sloja stakla presvučenim tankim 6 slojem konduktora. S obzirom na to da je i ljudsko tijelo konduktor, dodirivanje ekrana iskrivljuje elektrostatičkog polja koji se mjeri kao kapacitet[16].

Tipični se pametni mobitel sastoji od : matične ploče, CPU, GPU, memorije, različite antene za različite tehnologije, senzori, baterija itd., a to je prikazano na slici 5:



Slika 5. Arhitektura mobilnog uređaja
Photo source: www.evelta.com

Prilikom testiranja bitno je znati na kojem se hardveru testira jer o njemu ovisi i ponašanje aplikacije. Uglavnom su aplikacije za iOS i Android drugačije napravljene iako izgledaju kao ista aplikacija.

4. 1. Vrste aplikacija

Nakon hardvera, razlog zbog kojega su aplikacije na uređajima drugačije je to što obje platforme imaju smjernice za dizajn koje se razlikuju. Svaka platforma ima drugačiji preporučeni izgled aplikacije, tok aplikacije i način rada. Te se smjernice moraju poštovati kako bi aplikaciju bilo moguće prodavati u trgovini sistema. Bitno je pratiti smjernice jer su korisnici naviknuti na određeni „osjećaj“, pokrete i naredbe u aplikaciji, a kao što je rečeno korisnik će vrlo brzo obrisati aplikaciju ako se u njoj ne snalazi.

S naglim razvojem uređaja i sve veće njihove prisutnosti korisnici su postali izbirljivi. Postoji veliki broj različitih aplikacija koje imaju istu funkciju. Ako se u App store tražilicu na iPhone-u upiše „kalkulator“ dobit će se bezbroj aplikacija koje obavljaju tu funkciju; isto vrijedi i za Play Store na Androidu. Prema [18] 52% korisnika obriše mobilnu aplikaciju ili napusti web stranicu ako prilikom prvog korištenja naiđe na problem ili se ne mogu snaći. 44% korisnika upotrebljava manje od 5 različitih aplikacija dnevno. Korisnicima je najbitnije kako aplikacija radi (bitno je brzo učitavanje), je li dostupna i koliko je intuitivna. Zbog toga je bitno testirati aplikaciju i pobrinuti se da radi kako je očekivano i da je kvalitetna jer je izbor aplikacija beskonačan. Kako postoje različiti uređaji i različiti operativni sistemi za njih, postoje i različiti tipovi aplikacija. Aplikacija može biti nativna, hibridna ili web. Po samom izgledu aplikacije i njenom ponašanju ne može se znati je li ona nativna ili hibridna, a web aplikaciju se može prepoznati po tome što se prilikom njenog pokretanja otvara web pretraživač.

Nativne aplikacije su aplikacije programirane u jeziku koji je specifičan za mobilni operativni sistem. To znači Java za Android i Swift ili Objectiv-C za iOS. One mogu upotrebljavati sve biblioteke i API-je specifične za sistem čime im se omogućuje pristup svim dostupnim resursima sistema i uređajima. Zbog toga su aplikacije brže jer mogu upotrebljavati i donekle upravljati resursima CPU-a i GPU-a direktno. S obzirom da je nativna aplikacija pravljena za tačno određeni sistem, korisnici mogu koristiti sve standardne pokrete specifične za taj sistem, a i aplikacija je napravljena po smjernicama za dizajn tog sistema. Takva aplikacija može raditi i kad uređaj nije spojen na Internet[15]**.** Negativna strana nativnih aplikacija je to što se takva aplikacija može upotrebljavati samo na tom specifičnom sistemu kojemu je aplikacija nativna. Potrebno je praviti potpuno novu aplikaciju za drugi sistem.

Web aplikacije su posebna vrsta aplikacija. Kada govorimo o mobilnoj web aplikaciji zapravo mislimo na "web stranicu" posebno prilagođenu dizajnom i funkcionalnostima za mobilne uređaje kojoj se pristupa putem mobilnog Internet preglednika. Razvijene su upotrebom JavaScript-a, HTML-a (najčešće HTML5) i CSS-a. Takve aplikacije nisu pohranjene na mobitelu i može im se pristupiti samo preko interneta, što znači da nema instalacije i aplikacija se može jednostavno ažurirati. One ne mogu pristupiti većini hardvera pa ne mogu upotrebljavati kameru ni senzore, ali mogu dobiti GPS lokaciju ili davati obavijesti. Takve aplikacije su responzivne, mogu prikazivati videozapise i slike, ali nemaju sve moguće funkcionalnosti koje nativne aplikacije imaju. One se brzo i jednostavno razvijaju i mogu se pokretati na svim sistemima[19].

Problem se može pojaviti kod web pretraživača jer različiti pretraživači mogu prikazivati aplikaciju drugačije pa je zbog toga potrebno testirati aplikaciju na različitim pretraživačima koje korisnici koriste za pristup aplikaciji. Ako korisnik ima sporu internetsku vezu, aplikaciji će trebati više vremena kako bi se učitala jer se sav sadržaj prvo treba skinuti s interneta na mobitel kako bi se mogao prikazati.

Hibridne aplikacije su nastale kao kombinacija nativne i web aplikacije. Razvija se tako da se web aplikacija omota nativnom aplikacijom kako bi korisnici dobili osjećaj da upotrebljavaju nativnu aplikaciju, a ne web stranicu. Takve se aplikacije mogu upotrebljavati na Androidu, iOS-u i Windows-u, a omogućuju korištenje hardvera uređaja koji inače nije dostupan web aplikacijama[20]. Nedostatci takvih aplikaciju su to što su spore jer zavise o serveru i nisu pisane za tačno određenu arhitekturu. Problem je i što se ne mogu jedinstvenim dizajnom pokriti smjernice svih sistema.

5. TESTIRANJE

Ljudi su stalno u interakciji sa softverom. Koriste ga u svakodnevnom životu, od bankarstva i automobila do pametnih friždera. Neki od problema s kojima se ljudi redovno susreću su greške prilikom plaćanja karticom, sporo učitavanje web stranice, neke opcije aplikacije koji ne radi kako bi trebale raditi itd. Većina tih problema su softverski problemi.

Nemaju svi sistemi jednaku važnost i greške nemaju jednak utjecaj na ljude. Neki problemi su trivijalni, dok neki mogu koštati novaca i reputacije, a mogu rezultirati ozljedom ili smrću. Na primjer, može doći do pogreške prilikom prikaza podataka iz baze. Ako neka web stranica krivo prikazuje opis tvrtke, ne može se ništa loše dogoditi, jedino mogu izgledati neprofesionalno. Greška prikaza podatka kod medicinske aplikacije, koja nadzire razinu šećera u krvi i predlaže dozu inzulina pacijentima, je jako rizična i bitno je da do pogreške nikada ne dođe.

Testiranje je potrebno jer svi rade pogreške, a testira se kako bi aplikacija radila bez pogrešaka za što više korisnika. Neke pogreške mogu biti neprimjetne, ali neke mogu biti skupe i opasne. Svaki se proizvod treba provjeravati i testirati tokom razvoja jer uvijek nešto može poći po zlu. Testiranje aplikacije je potrebno izvoditi tokom razvoja aplikacije kako bi se greške što prije otkrile i kako bi se smanjila mogućnost pogrešaka kod završene aplikacije. Testirati se može sve u aplikaciji, od korisničkog interfeacea do toka aplikacije[21].

KVALITETA

Testiranje je proces analize programskog koda s ciljem otkrivanja informacija o ispravnosti i kvaliteti. Osigurava pouzdanost, ispravnost i otkrivanje pogrešaka. Aplikacija je kvalitetna ako je napravljena po dobro definisanim specifikacijama koje zadovoljava. Iz toga proizlaze verifikacija i validacija aplikacije. Verifikacija se odnosi na postupak provjere ponaša li se program prema specifikacijama i regulacijama te odgovara na pitanje „radimo li proizvod dobro?“. Validacija je postupak koji provjerava zadovoljava li program zahtjeve i odgovara na pitanje „radimo li dobar proizvod?“Međunarodni standard ISO/IEC 25010:2011 [14] za kvalitetu softvera definiše kvalitetu kao „stepen u kojem skup bitnih značajki zadovoljava zahtjeve“. Definišu šest karakteristika kvalitete softvera:

1. Funkcionalnost

Zadovoljavanje specifikacija i standarda, siguran i prikladan za upotrebu.

2. Pouzdanost

Mogućnost rada u određenim uvjetima kroz neki vremenski period. Podrazumijeva i mogućnost oporavka sistema ako dođe do greške.

3. Prikladnost za upotrebu
4. Učinkovitost
5. Održivost
6. Prenosivost

Softver se mora moći upotrebljavati u različitim okruženjima[30].

GREŠKE

Greške (engl. *bugs*), defekti ili mane su neki od naziva za probleme koji se pojavljuju u aplikaciji. Zatajenje aplikacije se događa kad aplikacija nije u skladu sazadanimspecifikacijama ili kad specifikacije ne opisuju dobro njenu funkciju. Svaka aplikacija ima zadane specifikacije koje treba zadovoljiti. Specifikacije definišu kako se aplikacija treba ponašati, šta će raditi i šta neće raditi. Greške nastaje kad [30]:

1. aplikacija ne radi što specifikacije kažu da bi trebala raditi
2. aplikacija radi ono što specifikacije kažu da ne bi trebala raditi
3. aplikacija radi nešto što nije navedeno u specifikacijama
4. aplikacija ne radi nešto što nije navedeno u specifikacijama, a trebalo bi biti
5. aplikacija je spora, nepregledna ili ju je teško koristiti.

Greške se mogu rangirati prema ozbiljnosti i prioritetu. Prema ozbiljnosti mogu biti: kritične, značajne, niže ili kozmetičke. Kritične greške onemogućuju rad aplikacije u potpunosti i bitno ih je što prije ispraviti. Značajne greške označavaju nerad neke funkcionalnosti aplikacije, ali se može zaobići greška koja se dobije. Niže greške su one koje ne utječu na rad aplikacije, a kozmetičke greške su odstupanje od zadanog dizajna[21].

PRINCIPI TESTIRANJA

Postoje principi testiranja koji su sastavljeni tokom proteklih pola stoljeća testiranja i smatraju se generalnim smjernicama prilikom svih vrsta testiranja.

1. Testiranje dokazuje prisutnost grešaka

Testiranje dokazuje prisutnost grešaka, ali ne može dokazati da greške ne postoje. Testiranjesmanjuje vjerovatnost postojanja neotkrivenih grešaka, ali ako greške nisu pronađene, to ne znači da one ne postoje

2. Nemoguće je testirati cijelu aplikaciju

Nemoguće je testirati sve moguće unose, prethodna stanja i rezultate. Umjesto sveobuhvatnog testiranja, radi se testiranje bitnih dijelova aplikacije i čestih akcija koji utječu na doživljaj korisnika. Redundantni i nepotrebni testovi se zanemaruju kako bi se uštedilo vrijeme.

3. Rano testiranje

Potrebno je što ranije tokom razvojnog procesa početi s testiranjem aplikacije kako bi se odmah pronašle i ispravile greške.

4. Imunizacija aplikacije

Ako se konstantno izvršava isti set testova, taj set testova neće pronaći nove greške jer će one biti ispravljene. Kako bi se izbjeglo nepronalaženje novih grešaka, testove je potrebno evaluirati i sukladno evaluaciji promijeniti. Potrebno ih je usmjeriti na neke druge dijelove aplikacije.

5. Testiranje zavisi o kontekstu

Testiranju se pristupa ovisno o kontekstu. Različite vrste aplikacije se testiraju na drugačije načine.

6. Zabluda o nepostojanju grešaka

Pronalazak i popravljanje grešaka ne znači da će korisnici prihvatiti aplikaciju. Broj grešaka nije najbitnije obilježje aplikacije. Ako je ona nestabilna, spora i ne zadovoljava korisnikove zahtjeve, neće biti uspješna.

7. Što se više grešaka pronađe, to više grešaka postoji

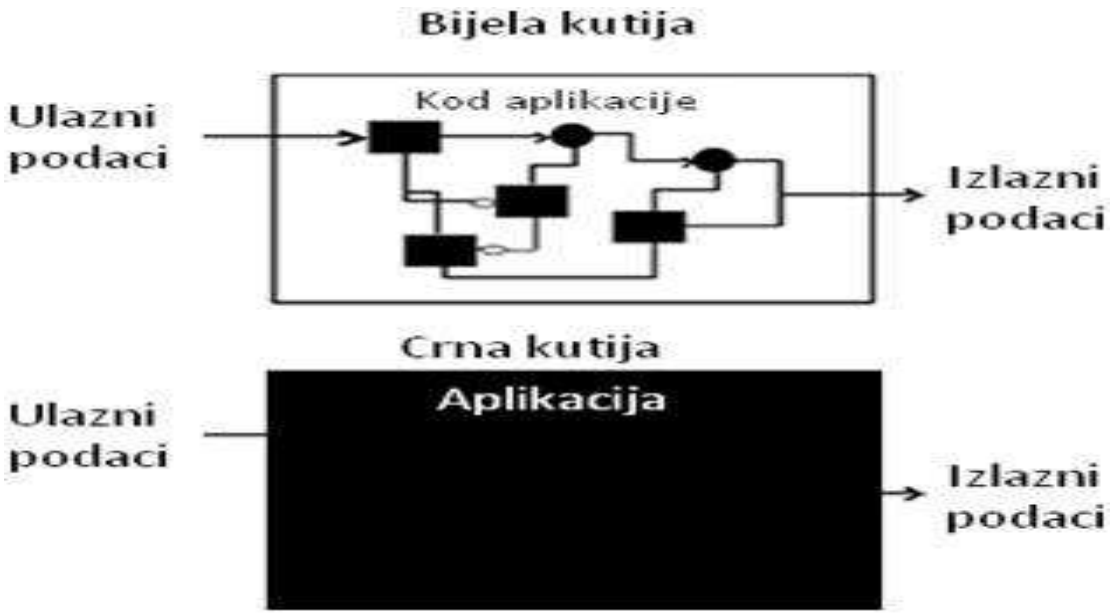
Ako se prilikom testiranja pronađe greška, postoji velika vjerovatnost da postoji još nekoliko takvih pogrešaka ili pogrešaka oko te pronađene. Razlog tome je što programeri ponavljaju pogreške ili što pronađena pogreška može biti samo nagovještaj većih problema u pozadini aplikacije.

8. Neće sve greške biti ispravljene

Zbog nedostatka vremena, neisplativosti ispravka ili iz razloga što pronađena greška zapravo nije greška neće sve greške biti ispravljene[23].

DINAMIČKO I STATIČKO TESTIRANJE

Testiranje aplikacije može se podijeliti u dvije skupine, dinamičko i statičko testiranje. Kod statičkog testiranja se ne izvršava aplikacija nego se gleda sam kod. Testeri i programeri bi trebali tokom razvoja aplikacije raditi evaluaciju koda (engl. *code review*), odnosno neko ko nije napisao kod bi trebao pogledati taj kod i odlučiti prati li on smjernice kodiranja i je li on prihvatljiv. Druga vrsta evaluacije se može raditi prije razvoja aplikacije kontroliranjem zadanih specifikacija kako se tokom razvoja ne bi morale mijenjati specifikacije. Tokom statičke faze, kod se može provjeravati pomoću alata kako bi se osiguralo da prati zadane smjernice formata kodiranja[23]. Kod dinamičkog se testiranja kod izvršava i nadzire se rad aplikacije. Dinamičko se testiranje može podijeliti u dvije skupine testiranja: bijela kutija (engl. *white box*) i crna kutija (engl. *black box*) testiranje prikazanih na slici.



Slika 6 :Bijela i crna kutija načini testiranja
Photo source: www.unios.hr

Bijela kutija testiranje opisuje testiranje prilikom kojeg se poznaje unutrašnja struktura metoda ili klasa. Cilj takvog testiranja je provjeriti moguće putanje, uvjete i petlje kako bi se aplikacija osigurala od rušenja, beskonačnih petlji, nedostatka memorije, kako bi se otkrili neki sigurnosni nedostaci itd. Takvo testiranje uglavnom vrše sami programeri prilikom pisanja koda i ako pronađu greške, odmah ih ispravljaju. Naziva se i testiranje vođeno logikom (engl. *logic-driven*) jer se testni scenariji izvode iz logike aplikacije i njenog ponašanja[22].

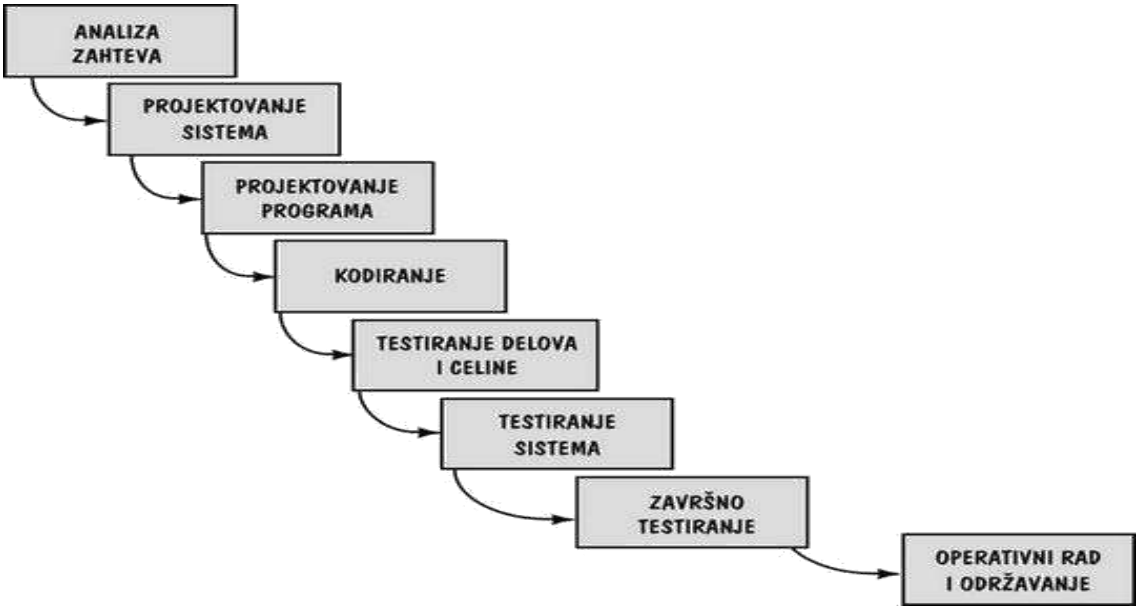
Crna kutija testiranje opisuje testiranje prilikom kojega se ne poznaje unutarnja struktura metoda i klasa. Takvo testiranje uglavnom rade testeri koji znaju što aplikacija treba raditi, ali ne i kako. U ovome slučaju testni scenariji se pišu iz specifikacija. Takva vrsta testiranja još se naziva i testiranje vođeno podatcima (engl. *data-driven*) jer se testiraju uneseni i dobiveni podaci. Razlike između crne i bijele kutije načina testiranja prikazane su u tablici ispod[22].

Crna kutija	Bijela kutija
Testeru nije poznata unutarnja struktura ni kod aplikacije	Tester zna unutarnju strukturu i kod aplikacije
Sistemsko i testiranje prihvatljivosti	Testiranje modula
Testove izvode testeri	Testove izvode programeri
Nije potrebno poznavanje programskog jezika u kojemu je aplikacija pisana	Potrebno je dobro poznavanje programskog jezika u kojemu je aplikacija pisana

Slika 7. Razlike u načinu testiranja bijele i crne kutije
Photo source: www.unizg.hr.

RAZINE TESTIRANJA

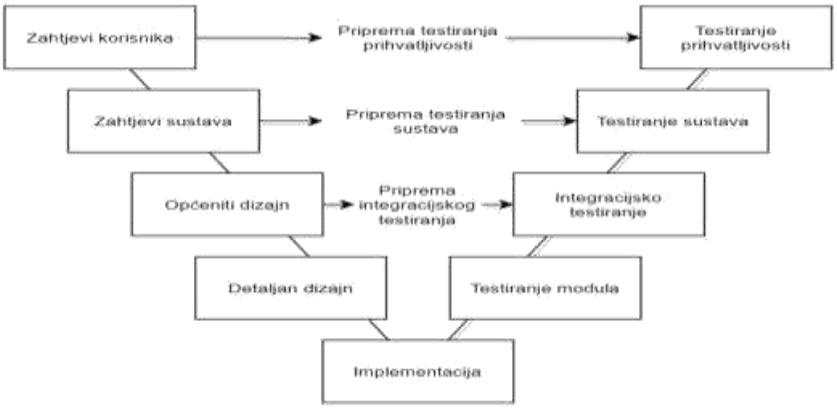
Testiranje softvera je široki pojam, te postoje razna područja koja se mogu testirati. Neke od mogućnosti su testiranje zahtjeva i specifikacije softvera, testiranje dizajna, testiranje ponašanja izvornog koda itd… Danas je razvijeno puno modela po kojima se razvijaju aplikacije, međtim odabir modela ovisi o aplikaciji i njenim ciljevima. Neke od tih metoda se zasnivaju na brzini i zaradi, a neke na kvaliteti i dokumentiranju. Testiranje je uvijek uključeno u razvojni ciklus, ali svaka metoda osigurava drugačije mjesto testiranju. V-model razvoja aplikacije je razvijen jer su tradicionalni načini razvoja, kao na primjer model vodopad, imali problema s pravovremenim otkrivanjem grešaka[23].



Slika 8. Model vodopada

Photo source: http://www.link-university.com/lekcija/Model-vo

Razvoj softvera ovom metodom se sastoji od šest ključnih faza. Razvoj započinje analizom zahtjeva tokom koje se skupljaju sve potrebne informacije o proizvodu, željama klijenta i moguće funkcionalnosti. Nakon toga se smišlja dizajn sistema. Nakon toga dolazi razdoblje implementacije gdje se pravi cijela aplikacija. Nakon završetka aplikacije, ona se testira i objavljuje kao gotov proizvod. Tada se aplikacija održava. Takav način razvoja aplikacije je linearan sekvencijalni i ne dolazi do preklapanja faza razvoja. Probleme kod takvog načina rada je što testiranje dolazi tek pred kraj razvojnog procesa i greške se kasno otkrivaju[24]. Jedna od glavnih smjernica V-modela je da se s testiranjem počne što ranije tokom razvojnog procesa što je i jedan od ranije spomenutih principa testiranja. Testiranje bi se trebalo izvoditi paralelno s razvojem aplikacije u svim fazama razvoja.



Slika 9. V-model razvoja i testiranja aplikacije
Photo source: https://repozitorij.etfos.hr

Navedene su razine testiranja: testiranje komponenti, integracijsko testiranje, sistemsko testiranje, testiranje prihvaćenosti koje će detaljnije biti objašnjene. Te se razine mogu kombinirati s nekim drugim modelima razvoja ili se ne moraju sve izvršavati. Kao što jedan od principa testiranja govori, testiranje zavisi od konteksta i vrste aplikacije.

Aplikacija se može prikazati kao puno malih komponenti koji se spajaju ili imaju interakciju. Moduli su najmanje jedinice u aplikaciji, kao na primjer funkcije, metode ili objekti. Sve komponente zajedno tvore konačnu aplikaciju i sve njene funkcionalnosti.

TESTIRANJE MODULA – UNIT TESTIRANJE

Unit testing su testovi koji služe za testiranje konkretne implementacije, odnosno testovi koji testiraju izvorni kod. Ova razina testiranja je najniža razina testova, te je najbitnija ako želimo testirati linije koda, te metode u određenim klasama. Ovakvi testovi se u pravilo pišu na razini specifikacije gdje svaki mali zadatak tijekom implementacije treba imati svoj unit test. Ovi testovi su pisani od strane programera, te u idealnom slučaju za svaku metodu u izvornom kodu postoji referentni test koji potvrđuje njezinu funkcionalnost. Uglavnom su izvode u razvojnom okruženju uz dostupne debugging opcije[25].

Greške pronađene prilikom testiranja komponenti se odmah ispravljaju i nema nikakvih posljedica. Takva se vrsta testiranja ubraja pod testiranje bijele kutije.

INTEGRACIJSKO TESTIRANJE

Integracijski testovi su testovi koji testiraju interfejs između dvije različite komponente, odnosno ovi testovi nam garantiraju da dvije komponente komuniciraju kako bi trebale, te da je interakcija između njih onakva kakva treba biti. Ovo je viša razina testova, te dok unit testovima testiramo da pojedini mali dio softvera radi ono za što je namijenjen, s integracijskim testovima testiramo da dva mala (ili velika) dijela softvera međusobno funkcioniraju kako bi trebala[26].

Postoji i drugi način, inkrementalno testiranje, koje podrazumijeva integraciju jedne po jedne komponente. Prednost takvog načina rada je što se jednostavno može otkriti uzrok greške jer nema puno novih komponenata, a nedostatak je što nisu sve promjene u aplikaciji napravljene pa se moraju pisati zamjenski kodovi[23]. Inkrementalnom se testiranju može pristupiti na dva načina. Od gore prema dolje i od dolje prema gore. Od dolje prema gore metoda znači da se testiraju niže komponente u odnosu na više komponente, odnosno module. Od gore prema dolje pristup testira od cijele aplikacije prema nižim komponentama.

SISTEMSKI TESTOVI

Sistemski testovi su testovi koji testiraju arhitekturu sistema, odnosno testiraju da neka cjelina radi na način kako je zamišljena. Takva neka cjelina može sadržavati više komponenata koje komuniciraju na razne načine, no sa sistemskim testovima možemo biti sigurni da te cjeline rade kako su zamišljene.

TESTIRANJE PRIHVATLJIVOSTI

Testovi prihvatljivosti su testovi koji nužno ne trebaju biti provođeni od strane programera, jer za razliku od svih ostalih testova, testovi prihvatljivosti ne trebaju imati napisane testove već ručno testiranje od strane korisnika ili bilo koje druge osobe može biti dovoljno da bi se ovi testovi izvršili.

VRSTE TESTOVA

Vrste testova postoje kako bi se definirali ciljevi testova na određenoj razini. Potrebno je definirati vrstu testa jer testiranje jedne komponente i njene ispravne funkcionalnosti ne znači da je ta komponenta bez greške kod ostalih vrsta testova. Jednostavnije je testove podijeliti po ciljevima koje trebaju ostvariti. Cilj nekog testa može biti testiranje funkcionalnosti, ali isti tako može biti i neka nefunkcijska karakteristika, na primjer provjera sigurnosti ili provjera korištenja resursa. Po svome cilju testovi se mogu podijeliti na funkcijske i nefunkcijske[23].

FUNKCIJSKI TESTOVI

Funkcijsko testiranje služi za provjeru ispravnosti aplikacije, odnosno radi li aplikacija ono što je zadano u specifikacijama. Može se reći da provjerava „šta aplikacija radi“. Funkcijsko se testiranje izvodi tako da se napravi neka akcija ili unos u aplikaciji i dobiveni se rezultat uspoređuje s očekivanim. Provjerava se je li rezultat jednak očekivanom, ako nije došlo je do greške. Takvo se testiranje može izvoditi kao crna ili bijela kutija, testiranje modula, integracijsko testiranje, testiranje sistema ili testiranje prihvatljivosti[27].

Funkcijsko se testiranje izvodi s krajnjim korisnicima na umu pokušavajući upotrebljavati aplikaciju kao što bi je korisnik upotrebljavao. Testni scenariji se pišu uzimajući u obzir ponašanje korisnika i njihova moguća očekivanja od aplikacije i ne uključuje testiranje performansi. Zato postoji druga vrsta testova, nefunkcijski testovi[27].

NEFUNKCIJSKI TESTOVI

Jedan od nefunkcijskih testiranja je testiranje sigurnosti. Tu se provjerava štiti li aplikacija korisnikove podatke. Pod testiranje sigurnosti se ubrajaju provjera autentifikacije i autorizacije. Autentifikacija je proces u kojemu aplikacija traži od korisnika na primjer korisničko ime i lozinku kako bi korisnik mogao pristupiti svojim podacima. Bitno je da niko osim osobe s pravom kombinacijom korisničkog imena i lozinke ne može pristupiti podacima. Autorizacija govori o privilegijama i pravima pristupa koje određeni korisnik može imati. Bitno je osigurati da korisnici nemaju ovlasti nad aplikacijom ili podacima za koje nisu autorizirani[28].

Još neki od nefunkcijskih testova su testiranje izdržljivosti gdje se testira kako sistem podnosi velika opterećenja, testiranje internacionalizacije i lokalizacije gdje se provjerava reagira li aplikacija u skladu s postavljenom lokacijom ili jezikom, itd.

RUČNO TESTIRANJE

Ručno (engl. manual) testiranje je najstarija verzija testiranja. Ručno testiranje zahtjeva testera koji je upućen u aplikaciju i zna koje sve funkcionalnosti ona pruža. Tester sastavlja testne scenarije koji trebaju biti napisani tako da provjeravaju aplikaciju i njene rubne slučajeve. Nakon toga, tester izvršava testove i označava jesu li prošli (dogodilo se što je očekivano) ili su pali (dogodilo se nešto neočekivano što se nije trebalo dogoditi). Ako je neki test pao, bitno je imati zapis o tome što se točno dogodilo i koje su akcije dovele do tog pada kako bi programeri znali ponoviti korake i vidjeti što je pošlo po krivu te popraviti grešku. Problem kod ručnog testiranja je što zahtijeva puno vremena i ljudskih resursa, a što aplikacija postaje veća i kompliciranija, problem raste. Ručno testiranje se ne smatra konzistentnim ni ponovljivim jer svaki ručni tester može pristupiti testnom scenariju ili testu drugačije. Razlike u brzini izvođenja nekih operacija ili mjestu klika mogu rezultirati drugačijim rezultatima[29].

Ako se uzme u obzir pet razina testiranja, kojima se ostvaruje osiguranje kvalitete, jasno je kako ručno testiranje na svakoj razini oduzima previše vremena, zbog čega dolazi do kašnjenja u isporuci dijela i na taj način prekrše rokovi isporuke. Kako bi se riješili problemi ručnog testiranja, preporučuje se automatsko testiranje gdje se pišu skripte koje izvršavaju zadane testove.

6. AUTOMATIZACIJA TESTIRANJA

Automatizacija testiranja ima puno prednosti za testiranje softvera. Poboljšava rezultate i kvalitetu, povećava pouzdanost i smanjuje odstupanja u rezultatima, ubrzava proces, povećava pokrivenost testovima, a u konačnici može povećati sveukupnu kvalitetu softvera. Dobra automatizacija može dovesti do ubrzanja objavljivanja aplikacije, poboljšati kvalitetu objave, povećati pokrivenost testovima, smanjiti troškove testiranja i omogućiti ranije otkrivanje grešaka ako se pravilno testira[31].

Jedan od glavnih problema kod automatizacije testiranja je skalabilnost i održavanje testova. Softveri i aplikacije se vremenom mijenjaju. Mijenja im se dizajn i funkcije pa se može dogoditi da stari testovi više nisu prikladni za novu verziju aplikacije. Drugi problem je niska pokrivenost automatiziranih testova. Kao što je spomenuto, aplikacija se mijenja i stari testovi prestaju biti aktuelni. Potrebno ih je nadopuniti ili promijeniti, što zahtijeva ljudske resurse[31].

Automatizacija testiranja je proces u kojem se upotrebljavaju alati za testiranje softvera. Kreiraju se skripte koji se izvršavaju i na kraju se rezultat skripte uspoređuje s očekivanim rezultatom. Automatizacija testova se vrši pomoću alata za automatizaciju testiranja. Alat za automatizaciju testiranja je softverska aplikacija koja omogućuje analizu aplikacije i izvršavanje testova. Prvi bi trebali biti automatizirani testovi koje je potrebno konstantno ponavljati i testovi koji zahtijevaju puno podataka za izvršavanje. To su na primjer testovi koje je teško izvoditi ručno, koji provjeravaju rad aplikacije na različitim platformama ili povjeravaju GUI. Ne bi trebalo automatizirati testove koji će se izvršiti samo jednom i nikada više, kao ni testove koji testiraju granice aplikacije i izmišljaju se na mjestu (ad hoc) ili testovi čiju prolaznost procjenjuje čovjek[32].

Selenium – Web Driver

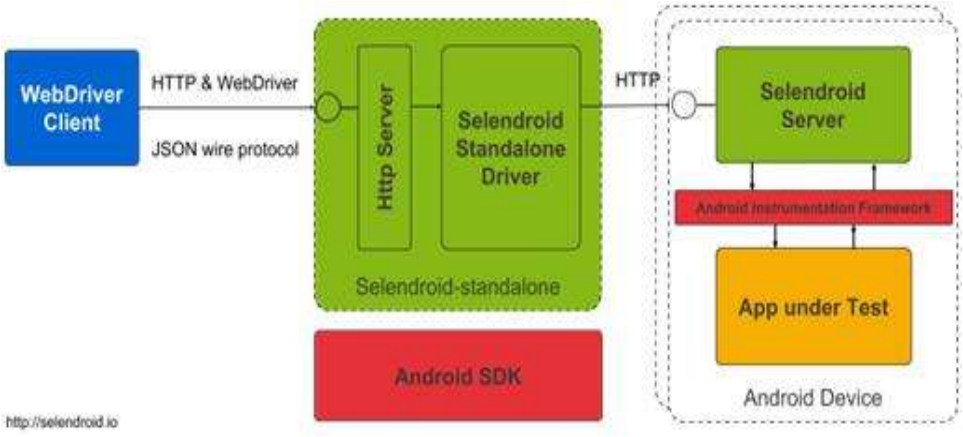
Selenium je nastao 2004. godine kada je Jason Huggins, radeći ručno testiranje web stranice, došao na ideju kako bi se repetitivni testovi mogli automatizirati i sami izvoditi. Napisao je JavaScript biblioteku koja je omogućavala interakciju koda s internetskom stranicom i ponovljivo izvršavanje testova na različitim internetskim preglednicima. Ta je biblioteka postala osnova za današnje Selenium alate. S vremenom su internetski preglednici počeli ograničavati JavaScript i zbog toga je Selenium postajalo sve teže upotrebljavati. 2006. godine Google-ov inženjer Simon Stewart je počeo raditi na projektu koji je nazvao WebDriver i trebao je služiti kao zamjena za tadašnji Selenium. WebDriver je bio zamišljen kao alat za testiranje koji direktno komunicira s preglednikom izbjegavajući ograničenja JavaScript-a. 2008. godine Selenium i WebDriver su spojeni u jedan projekt pod nazivom Selenium 2.0 [22].

Jedan od alata koje Selenium nudi je Selenium IDE koji omogućuje pravljenje jednostavnih i brzih testova pritiskanjem po stranici i spremanjem tih akcija kao jedan test, ali se on ne preporučuje za automatizaciju testiranja svih testova. Preporučuje se pisanje skripti upotrebom Selenium-WebDriver-a. Selenium-WebDriver podržava sve popularne Internet preglednike (Chrome, Firefox, Safari, Opera, IE,…), ali i mobilne platforme iOS i Android uz pomoć nekih dodatnih alata (Appium, Selendroid ili ios-driver). Podržava pisanje testova u Javi, Pyhtonu, Rubyju i C# [22].

Aplikacijski programski interface (API, engl. *Application Programming Interface*) se sastoji od tehničkih specifikacija koje određuju kako se informacije razmjenjuju između programa i interfacea napisanog po tim specifikacijama i objavljenog za upotrebu. Aplikacije pomoću API-ja međusobno komuniciraju i razmjenjuju podatke [23]. WebDriver Selenium-u služi kao API za interakciju s preglednikom i upravljanje njime. Kako bi se Selenium 2.0 koristio u projektu, potrebno ga je uvesti. Najjednostavniji način za to učiniti je pomoću Maven-a. Maven je alat koji služi za automatsko skidanje ovisnosti i njihovo uvoženje u projekt uzimajući u obzir verzije. Svaki malo bolji program za uređivanje koda ima opciju upotrebe Maven-a

SELENDROID

Selendroid je okvir za automatizaciju testova za nativne, hibridne i web Android aplikacije. Oslanja se na Selenium WebDriver. Testove je moguće izvršavati na emulatorima i fizičkim uređajima. Nije potrebno modificirati aplikaciju koja se testira i moguće je izvršavati više od jednog testa u isto vrijeme. Sastoji se od četiri glavne komponente: WebDriver klijenta, Selendroid servera, AndroidDriver applikacije i samostojećeg Selendroida [24]. Na slici je prikazana povezanost tih komponenti.



Slika 10. Arhitektura Selendroid-a
Photo source: www.ibm.com

WebDriver klijent je je Java klijentska biblioteka zasnovana na Selenium Java klijentu. Sasmostojeći Selendroid se upotrebljava za instalaciju servera i aplikacije pod testom na uređaj, služi za komunikaciju između klijenta i servera. Selendroid server je server koji se izvršava uz aplikaciju na mobilnom uređaju. AndroidDriver je aplikacija na mobilnom uređaju koja omogućuje pristup elementima mobilne aplikacije pod testom [24].

iOS DRIVER

Ios-driver služi za automatizaciju testiranja aplikacija na iOS mobilnim uređajima. Omogućuje testiranje nativnih, hibridnih i web aplikacija na emulatorima i fizičkim uređajima. Upotrebljava pristup skoro identičan Selendroid-u samo za iOS arhitekturu. Implementira JSON Wire protokol kako bi komunicirao s iOS aplikacijom [25].

APPIUM

Appium je alat otvorenog koda za automatizaciju nativnih, web i hibridnih aplikacija za iOS, Android i Windows platforme, što znači da omogućuje više platformsko (engl. *crossplatform*) testiranje. Višeplatformsko znači da se isti kod (u ovom slučaju test) može upotrijebiti za različite platforme [26]. U pozadini za iOS upotrebljava XCUITest i UIAutomation okvire. Ti su okviri za testiranje odobreni i napravljeni od strane Apple-a za pisanje testova upotrebom Xcode razvojnog okruženja tokom samog razvoja aplikacije, dakle za testiranje modula. Za Android upotrebljava UiAutomator. Upotrebljavaju se WebDriver i JSON Wire protokol koji je proširen u Mobile JSON Wire protokol. Proširenje je napravljeno dodavanjem novih, za mobilne uređaje specifičnih, ponašanja i načina lociranja kao na primjer informacije vezane za rotaciju uređaja. Appium je zapravo server napisan u Node.js-u koji prima zahtjev od klijenta, izvršava taj zahtjev na mobilnom uređaju i šalje rezultat i odgovor klijentu. Automatizacija se izvršava kao sjednica (engl. *session*).

Slika 11 : Ulaz na Appium
Photo source: www.sap.com

PISANJE TESTOVA

Java je programski jezik koji se upotrebljava za različite projekte i u različite svrhe. Kako bi se Java upotrebljavala za testiranje, potrebno je upotrijebiti neki od okvira (engl. *framework*) za Javu za testiranje. Ti okviri donose nove funkcionalnosti Javi i WebDriver-u te olakšavaju upravljanje testovima, njihovo pokretanje i izvršavanje. Omogućavaju selektivno ili sekvencijalno izvršavanje testova, generiranje rezultata, ispis koraka testa, spremanje slike s ekrana uspecificiranim trenucima i mnoge druge mogućnosti. Spomenut će se i usporediti dva dostupna okvira: JUnit i TestNG jer su to jedni od najpopularnijih i najkorištenijih okvira

J-UNIT

JUnit je okvir za pisanje testova, a kao testovi koji se mogu najbolje izvesti pomoću njega navode se testovi modula. Besplatan je i otvorenog je koda.

Bitan je kod testiranja vođenog razvojem jer omogućuje pisanje testova i prije pisanja samog koda aplikacije.

Omogućuje definiranje testnih metoda, izvođenje testova, uspoređivanje dobivenih i očekivanih rezultata. Testne klase je moguće povezati i izvoditi ih jedne za drugom [28].

TEST-NG

Test- NG je okvir za pisanje testova nastao kao poboljšanje JUnite-a tako što su dodane nove funkcionalnosti. Postaje sve popularniji i prikladan je za pisanje svih vrsta testova. Neke od dodanih funkcionalnosti su mogućnost izvršavanja koda prije i poslije izvršavanja grupe testova. Omogućuje označavanje testa pripadanjem u neku grupu pa se tako testovi mogu označiti nekom oznakom i onda se kasnije mogu izvršiti samo testovi s tom oznakom. TestNG pruža mogućnost izvršavanja istog testa paralelno na više niti. Još jedna od mogućnosti koje JUnit nema je prihvaćanje unosnih podataka iz XML, CSV ili običnog tekstualnog dokumenta[33].

FIZIČKI UREĐAJ, EMULATOR I SIMULATOR

Testiranje na fizičkom uređaju je najlogičnije jer se aplikacije prave za fizičke uređaje i korisnik će ih tako upotrebljavati. Potrebno je znati ograničenja fizičkog uređaja i napraviti aplikaciju koja radi u skladu s njima. Problem kod takvog testiranja je što postoji puno različitih uređaja (pogotovo za Android) s različitim hardverom, a i softverom.

Emulatori su programi koji kompajliraju kod aplikacije tako da se aplikacija može izvršavati na računalu, a ne na mobilnom uređaju. Emulatori se ponašaju kao fizički uređaji i oponašaju njihov hardver i operativni sustav. Nedostatak emulatora je što se aplikacija zapravo izvršava na računalu pa taj emulirani virtualni uređaj nema senzore ni hardver kao što su kamera ili mikrofon [5]. Neki emulatori nude mogućnost i upravljanja senzorima. Na primjer Genymotion [30], Android emulator, ima mogućnost davanja GPS lokacije uređaju, upotrebljavanje kamere i mikrofona (ako ih računalo ima) i još neke mogućnosti.

Simulatori su jednostavniji softveri koji simuliraju samo dio rada fizičkog uređaja i hardvera. Jednostavniji su za uporabu i „lakši“ za računalo na kojemu se izvršavaju. Nema nikakve mogućnosti interakcije s hardverom [5].

Najbolja je praksa upotrebljavati emulator ili simulator na početku razvoja kako bi programer bio siguran da se aplikacija izvršava kao što je zamišljeno. Oni uglavnom dolaze besplatno u sklopu razvojnog okruženja. Za testiranje je bitno testirati aplikaciju na pravom uređaju (čak i kada se ne mogu testirati svi mogući uređaji) jer simulatori i emulatori ne daju pravu sliku o izvršavanju aplikacije. Emulator i simulator pružaju kontrolirano okruženje u kakvom aplikacija nikad neće biti izvršavana kod korisnika.

ZAKLJUČAK

Testiranje je neizbježan proces ukoliko želimo izgraditi dobar i uspješan softver. Softver možemo testirati ručno ili automatski pomoću specijalnih alata za testiranje. Ipak oba pristupa imaju svoje prednosti i nedostatke. Rekla bih da je automatsko testiranje još uvijek nekompletan proces i da još uvijek ne može u potpunosti zamijeniti ručno testiranje. Upravo zbog toga treba naći način kako kombinirati ove pristupe testiranja kako bi izgradili što uspješniji softver bez grešaka koje nas mogu skupo koštati. Također, valja napomenuti kako je izgradnja svakog softvera unikatan proces. Prema tome, jedan način kombinacije automatskog i ručnog testiranja može biti uspješan i isplativ za neki softver, dok za neki drugi ne mora biti tako.

Ključne riječi: Android, iOs, testiranje, automatizacija testiranja, pametni telefoni, mobilne aplikacije, testovi.

LITERATURA

[1] MobilniBlog, https://www.mobilnishop.com/blog/zanimljivosti/prvi-pametni-telefoni/

(Pristupljeno: August, 2019)

[2] Balkan android, https://balkanandroid.com/ovo-je-trenutno-5-najpopularnijih-proizvodaca-telefona-u-evropi/

(Pristupljeno: August, 2019)

[3] Wikipedia – Android, iOS and Smartphones

[4] What is Android, http://developer.android.com/about/index.html

(Pristupljeno: August, 2019)

[5] Android Developers, http://source.android.com/

(Pristupljeno: August, 2019)

[6] Ttelegraf.rs, https://www.telegraf.rs/hi-tech/mobilni/2981482-android-90-dobio-je-i-svoje-zvanicno-ime-android-pie-i-dostupan-je-vec-danas

(Pristupljeno: August, 2019)

[7] M., Haris, B., Jadoon, M., Yousaf, F. H., Khan, Evolution of Android operating system: a review, 2nd International Conference on Advanced Research, Australia, 2017

[8] Wikipedia- iOS

[9] CBR-Computer Business Review, https://www.cbronline.com/what-is/what-is-ios-4899783/

(Pristupljeno: August, 2019)

[10] Apple Developer**,** https://developer.apple.com/develop/

(Pristupljeno, August, 2019)

[11] https://hr.wikipedia.org/wiki/Google_Play

(Pristupljeno, August, 2019)

[12] https://www.tportal.hr/tehno/clanak/app-store-trgovina-koja-je-promijenila-svijet-foto-20180717

(Pristupljeno, August, 2019)

[13]

https://translate.google.com/translate?hl=hr&sl=en&u=https://en.wikipedia.org/wiki/App_Sto re_(iOS)&prev=search

(Pristupljeno, August, 2019)

[14] https://preporucamo.com/koje-razlike-izmedu-google-play-store-app-store/2017/03/12/

(Pristupljeno, August, 2019)

[15] Knott, Hands-on mobile app testing : a guide for mobile testers and anyone involved in the mobile app, Pearson Education,Inc, United States of America, 2015.

[16] The Interaction Design Foundation, https://www.interaction-design.org/literature/article/theanatomy-of-a-smartphone-things-for-designers-to-consider-for-mobile-development].

(Pristupljeno, Septembar, 2019)

[17] https://qnovo.com/anatomy-of-an-

iphone/ (Pristupljeno, Septembar, 2019)

[18] Appdynamics, The App Attention Indeks, A 2017 study examining the impact of app

performance on consumer behavior and business outcomes, http://docplayer.net/67929934-Theapp-attention-index-a-2017-study-examining-the-impact-of-app-performance-on-consumerbehavior-and-business-outcomes.html

(Pristupljeno, Septembar, 2019)

19] HDonWEB Nativna Aplikacija ili Mobilna Web Aplikacija https://www.hdonweb.com/mobiteli/nativna-aplikacija-mobilna-web-stranica

(Pristupljeno, Septembar, 2019)

[20] Web programiranje, Uvod u hibridne mobilne aplikacije, https://www.webprogramiranje.org/uvod-u-hibridne-mobilne-aplikacije/

[21] www.unios.hr Fundak Dora " Automatizacija testiranja mobilnih aplikacija, https://repozitorij.etfos.hr/islandora/object/etfos%3A1874/datastream/PDF/view

[22] www.sum.ba (Sveučilište u Mostaru) Antonio Stanić '' VoIP uređaji i način rada '' Mostar, 2016. https://sum.ba.AntonioStanic.pdf

[23] D., Graham, E., van Veenendaal, I., Evans, R., Black, Foundations of software testing, ISTQB certification, International Thomson Business Press, London, 2006.

[24] Robert Manger, skripta SOFTVERSKO INZENJERSTVO, Zagreb, 2015

[25] Software Testing-Unit Testing

http://softwaretestingfundamentals.com/unit-testing/

(Pristupljeno, Septembar, 2019)

[26] Automatsko testiranje, Milan Milinčević

http://mrkve.etfos.hr/pred/ozm/si/sem19.pdf

(Pristupljeno, Septembar, 2019)

[27] What is Functional Testing? Types, Tips, Limitations & More, https://stackify.com/functional-testing-types-tips-limitations/

(Pristupljeno, Septembar, 2019)

28] Software testing revealed, Drugo izdanje, https://www.testinstitute.org/Software_Testing_Books_International_Software_Test_Institute.php

(Pristupljeno, Septembar, 2019)

[29] H., Q., Nguyen, M., Hackett, B., K., Whitelock, Happy about global software test automation, Happy about, Kalifornija, 2006.

[30] ISO/IEC 25010:2011, https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

(Pristupljeno, Septembar, 2019)

[31] H., Q., Nguyen, M., Hackett, B., K., Whitelock, Happy about global software test automation, Happy about, Kalifornija, 2006.

[32] Testiranje softvera, Matematički Fakultet Univerziteta u Beogradu, Senad Ibraimoski, 2012

[33] What is TestNG? https://www.toolsqa.com/testng/what-is-testng/

(Pristupljeno, Septembar, 2019)

[34] M. Pranic, Alati za testiranje softvera, završni rad, Split, septembar 2010.

POPIS SLIKA