Metodologije testiranja softvera u svrhu osiguravanja kvalitete

Krajinović, Antonija

SVEUČILIŠTE U RIJECI
Fakultet za menadžment u turizmu i ugostiteljstvu
Preddiplomski sveučilišni studij

ANTONIJA KRAJINOVIĆ

Metodologije testiranja softvera u svrhu osiguravanja kvalitete

Software testing methodologies

Završni rad

Opatija, 2023.

SVEUČILIŠTE U RIJECI
Fakultet za menadžment u turizmu i ugostiteljstvu

Preddiplomski sveučilišni studij

Poslovna ekonomija u turizmu i ugostiteljstvu

Studijski smjer: Menadžment u turizmu

Metodologije testiranja softvera u svrhu osiguravanja kvalitete

Software testing methodologies

Završni rad

Kolegij: **Poslovna informatika**

Student: **Antonija KRAJINOVIĆ**

Mentor: **Prof. dr. sc. Mislav ŠIMUNIĆ**

Matični broj: **0116166500/19**

Opatija, rujan 2023.

SVEUČILIŠTE U RIJECI UNIVERSITY OF RIJEKA
FAKULTET ZA MENADŽMENT U TURIZMU I UGOSTITELJSTVU
FACULTY OF TOURISM AND HOSPITALITY MANAGEMENT
OPATIJA, HRVATSKA CROATIA

IZJAVA O AUTORSTVU RADA I O JAVNOJ OBJAVI OBRANJENOG ZAVRŠNOG RADA

Antonija Krajinović	**0116166500/19**
(ime i prezime studenta)	(matični broj studenta)

Metodologije testiranja softvera u svrhu osiguravanja kvalitete

(naslov rada)

Izjavljujem da sam ovaj rad samostalno izradila/o, te da su svi dijelovi rada, nalazi ili ideje koje su u radu citirane ili se temelje na drugim izvorima, bilo da su u pitanju knjige, znanstveni ili stručni članci, Internet stranice, zakoni i sl. u radu jasno označeni kao takvi, te navedeni u popisu literature.

Izjavljujem da kao student–autor završnog rada, dozvoljavam Fakultetu za menadžment u turizmu i ugostiteljstvu Sveučilišta u Rijeci da ga trajno javno objavi i besplatno učini dostupnim javnosti u cjelovitom tekstu u mrežnom digitalnom repozitoriju Fakulteta za menadžment u turizmu i ugostiteljstvu Sveučilišta u Rijeci.

U svrhu podržavanja otvorenog pristupa završnim radovima trajno objavljenim u javno dostupnom digitalnom repozitoriju Fakulteta za menadžment u turizmu i ugostiteljstvu Sveučilišta u Rijeci, ovom izjavom dajem neisključivo imovinsko pravo iskorištavanja bez sadržajnog, vremenskog i prostornog mog završnog rada kao autorskog djela pod uvjetima *Creative Commons* licencije CC BY Imenovanje, prema opisu dostupnom na http://creativecommons.org/liceses/.

U Opatiji, 05.09.2023.

Potpis studenta

Sažetak

Testiranje softvera praksa je koja se razvijala simultano s razvojem računalne tehnologije. Kroz povijest, razvijale su se metodologije i tehnike testiranja softvera.
U radu je opisan životni ciklus testiranja softvera koji se sastoji od šest važnih faza. U radu su objašnjene softverske greške, kako one nastaju te kakav je njihov utjecaj na softver. Nadalje, u radu su detaljno opisane metodologije testiranja softvera te je grafički prikazana hijerarhijska podjela metodologija. Istaknute su prednosti i nedostaci različitih metodologija testiranja te njihov utjecaj na softver. Zajednički cilj svih metodologija testiranja softvera je osiguranje visoke kvalitete softvera. U radu je objašnjena razlika između osiguravanja kvalitete softvera te kontrole kvalitete, istaknuti su ciljevi osiguravanja kvalitete te plan provođenja aktivnosti. Pravilnim odabirom metodologija testiranja softvera i provođenjem aktivnosti osiguravanja kvalitete softvera smanjuje se mogućnost pogreške a dugoročno zadovoljstvo korisnika je neupitno.
Cilj ovoga rada je približiti metodologije testiranja softvera i istaknuti važnost pravilnog odabira u svrhu osiguravanja kvalitete.

Ključne riječi: testiranje softvera; softverske greške; metodologije; osiguranje kvalitete

Sadržaj

Uvod

U današnjem dobu gdje je razvoj tehnologije brz, neprestan i kompleksan potrebe za testiranjem proizvoda sve su veće. Svaki softver je proizvod koji zahtjeva detaljnu analizu prije nego dođe do krajnjeg korisnika. Kroz povijesni razvoj tehnologije, stvorene su različite metodologije, tehnike, strategije i pristupi testiranju softvera. Testiranje softvera ima svoj životni ciklus te je svaka od faza jednako važna za uspješno provođenje testiranja. Metodologije testiranja navedene u radu provode se u svrhu osiguravanje kvalitete te zadovoljstva korisnika.
Iako je manualno testiranje neizostavan tip testiranja bilo kojeg softvera, automatizirani testovi i alati za provođenje istih postaju sve popularniji. Ovaj rad proučava prednosti i nedostatke svakog tipa testiranja. Na kraju rada, objašnjen je pojam osiguravanja kvalitete te važnost plana i aktivnosti koje se provode kako bi se kvaliteta osigurala.

1. Povijest testiranja softvera

Testiranje softvera predstavlja niz aktivnosti kao što su evaluacija, analiza, identifikacija i prijava pogreški u softveru. Zajednički cilj navedenih aktivnosti je stvaranje što kvalitetnijeg proizvoda s što manje pogrešaka. Kroz povijest razvoja tehnologije, testiranje softvera nametnulo se kao odgovor na brze i neprestane promjene u digitalnom svijetu.

1.1 Prvi zabilježeni slučaj

Smatra se da je prvi zabilježeni slučaj pronalaska greške u softveru slučaj koji se odvio na Sveučilištu Harvard u Massachusettsu, 9. rujna 1947. godine. Tim znanstvenika i inženjera koji je sudjelovao u razvoju elektromehaničkog računala "Mark II" primijetio je neispravnost u radu računala. Uzrok neispravnosti bio je moljac koji se zaglavio u komponente računala. Znanstvenici su odstranili moljca iz računala te ga zalijepili u zapisnik uz napomenu: "Prvi stvarni slučaj pronalaska buga". Ovaj slučaj je izrodio izraz "debugiranje" kao način za opisivanje procesa pronalaska i rješavanja pogrešaka i problema u softveru ili hardveru. Iako je izraz "bug" [1] korišten i ranije u kontekstu greške u sustavu, ovaj slučaj je najpoznatiji primjer u području računalne tehnologije.

[1] Bug u razvoju softvera odnosi se na nenamjernu pogrešku, nedostatak ili grešku koja uzrokuje neočekivano ponašanje ili proizvodi netočne rezultate unutar računalnog programa. Pressman, R.S. (2014). Software Engineering: A Practitioner's Approach (8th Edition). McGraw-Hill Education.

1.2 Faze razvoja kroz povijest

Razvoj testiranja softvera kroz povijest može se podijeliti u šest ključnih faza.

- Prva faza (1940.-1950.)

U prvoj fazi razvoja testiranja softvera, testiranje je bilo zasnovano na provjeri osnovnih funkcionalnosti računala kao što su ispravno računanje jednostavnih matematičkih operacija. Testiranje su provodili programeri te je sav proces bio provođen ručno. S obzirom na masivnost nekadašnjih računala, programeri su morali provjeravati i ispravnost hardvera te međusobnu povezanost svih komponenti računala. Nekadašnji softveri bili su jednostavni te je samim tim i testiranje istih bilo trivijalno u usporedbi s današnjim standardima razvoja softvera. Unatoč tome, prva faza razvoja testiranja softvera doprinijela je razvoju naprednih metodologija i alata koji se koriste do današnjeg dana te pridonose osiguranju kvalitete softvera.

- Druga faza (1960.-1970.)

Druga faza obilježena je značajnim rastom i širenjem prakse testiranja softvera. Računalna industrija doživjela je ekspanziju, a softveri su postajali sve kompleksniji. Kompleksnost softvera doprinijela je razvoju i formalizaciji metodologija testiranja. U drugoj fazi prvi put se spominju metode poput testiranja s crnom i bijelom kutijom, metode koje danas predstavljaju standardnu praksu. Testiranje se također podijelilo na nekoliko razina. Osim novih metodologija, u drugoj fazi posebna pozornost se stavlja na potrebu za kvalitetom te pouzdanošću proizvoda. Računalne tvrtke promijenile su percepciju o testiranju te je ono uvedeno kao neizostavni korak u razvoju softvera.

- Treća faza (1980.-1990.)

Razdoblje između 1980. i 1990. obilježeno je formalizacijom testiranja softvera. Naglasak je stavljen na korištenje matematičkih modela za dokazivanje kvalitete softvera. Razvijene su metode kao što su strukturno testiranje te testiranje iscrpljivanjem koje su imale za cilj pokriti što veći broj slučajeva koji potencijalno mogu izazvati grešku u sustavu. Tradicionalno testiranje koje se provodilo do ranih 1980-ih odnosilo se na testiranje

proizvoda u trenutku kada je on dovršen, no promjenom procesa, softver testeri postaju uključeni u gotovo sve faze razvoja softvera.[2] U ovoj fazi testiranje postaje precizno, pouzdano i dokazivo.

- Četvrta faza (1990.-2000.)

U četvrtoj fazi razvoja testiranja softvera najznačajniji događaj je uspostavljanje standarda testiranja. Organizacije poput IEEE (eng. Institute of Electrical and Electronics Engineers) i ISO (eng. International Organization for Standardization) razvile su standarde koji uključuju detaljan plan testiranja, definiciju slučaja, način izvršenja testova te sustav praćenja rezultata. Osim standarda i jasno definiranih metodologija razvijena su i tehnička rješenja za praćenje i podršku procesa testiranja. Automatizirani testovi koji se izvršavaju pomoću računalnih alata doprinijeli su bržem i učinkovitijem testiranju te su omogućili dodatnu provjeru kvalitete softvera. Najpoznatiji alati za izvršavanje automatiziranih skripti za testiranje su Selenium i JUnit.

- Peta faza (2000.-2010.)

Peta faza razvoja testiranja obilježena je integracijom Agilnih metodologija kao što su Scrum i Kanban u proces testiranja softvera. Agilne metodologije predstavljaju skup vrijednosti, načela i praksi koje uključuju iterativni razvoj, testiranje i povratnu informaciju u novi stil razvoja aplikacija.[3] U ovoj fazi naglasak je na ranom prepoznavanju grešaka i sustavnom testiranju za vrijeme razvoja proizvoda. Testiranje svih sigurnosnih aspekata softvera značajno je dobilo na važnosti, kao i samo zadovoljstvo krajnjeg korisnika. Unaprjeđenja u ovoj fazi stvorila su temelj za inovacije u narednim godinama.

---

[2] William E. Lewis, *Software Testing and Continuous Quality Improvement*, 2nd ed. (Boca Raton: Auerbach Publications, 2005).

[3] [3] William E. Lewis, *Software Testing and Continuous Quality Improvement*, 2nd ed. (Boca Raton: Auerbach Publications, 2005).

- Šesta faza (2010. – danas)

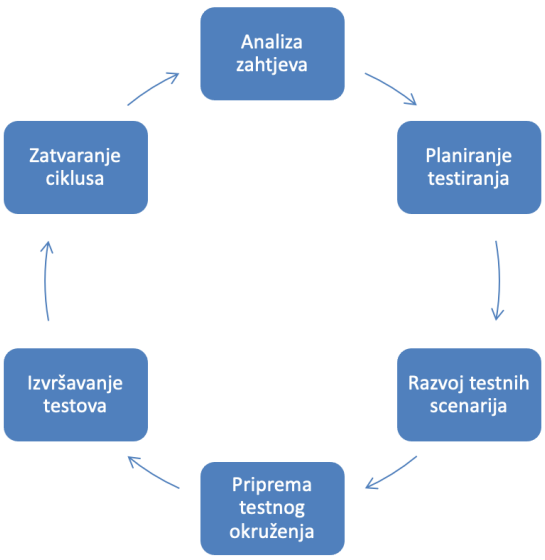
U fazi razvoja testiranja softvera koja započinje 2010. godine te traje do danas, naglašena je okrenutost prema DevOps pristupu razvoja softvera. DevOps je definiran kao način rada u kojemu se programeri i sistem operateri povezuju i blisko surađuju te rade prema zajedničkom cilju s malo ili nimalo organizacijskih prepreka ili granica između njih.[4] Kontinuirano testiranje jedna je od glavnih značajki DevOpsa koja pruža važne povratne informacije o kvaliteti u svim fazama razvoja softvera. Integracijom testova u svaku iteraciju razvojnog procesa omogućeno je nadzirati svaku funkcionalnost, performansu te sigurnosni aspekt proizvoda prije isporuke korisniku.

[4] Swartout, Paul. *Continuous Delivery and DevOps: A Quickstart Guide*. Packt Publishing, 2014.

2. Životni ciklus testiranja softvera

Životni ciklus testiranja softvera označava sistematski pristup testiranju softvera kako bi se zadovoljili svi zahtjevi i kriteriji te osigurala najviša razina kvalitete. Ciklus se sastoji od faza koje su unaprijed definirane, a to su:

- Analiza zahtjeva
- Planiranje testiranja
- Razvoj testnih scenarija
- Priprema testnog okruženja
- Izvršavanje testova
- Zatvaranje ciklusa



Slika 1. Životni ciklus testiranja softvera

Izvor: Autorica

2.1 Analiza zahtjeva

Analiza zahtjeva prvi je korak u životnom ciklusu softvera. U ovoj fazi tim za osiguravanje kvalitete (eng. Quality Assurance team) prikuplja zahtjeve vezane za proizvod koji će biti testiran. Aktivnosti koje se provode u ovoj fazi odnose se na detaljnu analizu dokumentacije koja uključuju identifikaciju potencijalnih sigurnosnih propusta, anomalija, nepotpunih ili neizvedivih zahtjeva. Na kraju ove faze, zaduženi tim bi trebao imati listu svih potrebnih testova koji će biti izvedeni kao i listu svih potrebnih alata za izvođenje testova.

2.2 Planiranje testiranja

U fazi planiranja testiranja određuje se potrebno vrijeme, broj ljudi te procjena svih troškova testiranja. Definira se strategija koja uključuje metode i tehnike testiranja, testno okruženje i potrebne resurse. Članovi tima raspoređuju zadatke te odgovornost. Na kraju ove faze, zaduženi tim bi trebao imati jasno razumijevanje svih testnih aktivnosti kao i detaljan plan istih.

2.3 Razvoj testnih scenarija

U fazi razvoja testnih scenarija provodi se raspisivanje testnih scenarija. Testni scenariji uključuju testne podatke, korake u testiranju, zadani i očekivani output. Dokumentiranjem testnih scenarija postiže se adekvatna pokrivenost softvera testovima te se osigurava da proces testiranja bude precizan i detaljan. Na kraju ove faze, tim treba imati što veći postotak pokrivenosti softvera testnim scenarijima.

2.4 Priprema testnog okruženja

Priprema testnog okruženja je važan korak u životnom ciklusu testiranja softvera jer ona definira preduvjete za nastavak testiranja. U ovoj fazi važno je sudjelovanje drugih timova koji će osigurati testnom timu sve potrebne uvjete za izvršavanje testova.

2.5 Izvršavanje testova

Kada su dovršene sve prethodne faze, kreće faza testiranja softvera. Testiranje se izvršava prema isplaniranim i pripremljenim testnim scenarijama. U ovoj fazi izvršavaju se testne skripte, nadgleda se rezultat izvršavanja skripti te se prijavljuju uočene nepravilnosti i greške. Ukoliko se pronađe greška, ona se prijavljuje programeru te nakon ispravka se ponovno izvršavaju testovi.

2.6 Zatvaranje ciklusa

U posljednjoj fazi testiranja softvera, završavaju se sve testne aktivnosti. Testni tim potom analizira sve dobivene rezultate te razvija strategiju za daljnje životne cikluse testiranja softvera. Cilj posljednje faze je usvojiti dobre prakse iz prethodno završenog životnog ciklusa softvera.

3. Softverske pogreške

Softverske pogreške odnose se na greške, neispravnosti i nedostatke u softveru. Rezultat postojanja takvih grešaka je neželjeno ponašanje softvera. Pogreške mogu nastati zbog nejasnih specifikacija, nekompatibilnosti, nerazumijevanja korisnika te zbog još mnogobrojnih čimbenika. Testiranjem i korištenjem pravih metodologija testiranja softvera pogreške se mogu smanjiti ili spriječiti. Nije moguće potpuno isključiti nastajanje pogreški no kontinuiranom analizom i evaluacijom softvera moguće je stvoriti stabilan i pouzdan softver.
U kontekstu razvoja softvera razlikujemo nekoliko različitih vrsta grešaka.

3.1.1 Bug

Bug je greška ili nedostatak u softveru koja rezultira neočekivanim ponašanjem softvera. Razlikujemo logičke bugove, bugove u algoritmu i bugove resursa. Logički bug se još naziva i programski bug nastaje u slučaju nepotpunog ili neprikladnog programskog koda i pozadinske logike. Bugovi u algoritmu nastaju zbog pogreške pri prisanju algoritma koji služi za određenju kalkulaciju u programskog kodu. Ovakvi bugovi mogu utjecati samo na brzinu softvera ali i mogu rezultirati neispravnim rezultatima. Bugovi resursa odnose se na pogreške koje nastaju zbog neispravnog upravljanja resursima softvera kao što su memorija, baze podataka, procesorsko vrijeme ili mrežne veze.

3.1.2 Nedostatak (eng. defect)

Nedostatak ili defekt u softveru odnosi se na nedostatak i neispravnost u dizajnu, dokumentaciji ili programerskom kodu. Defekt se može i definirati kao razlika između željenih i dobivenih rezultata ponašanja softvera. Razlika između defekta i buga je u tome što se defekt u softveru identificira za vrijeme testiranja, dok se bug odnosi na defekt koji još uvijek nije identificiran. Pronađeni defekti u softveru moraju se prioritizirati kako bi njihovo rješavanje bilo što uspješnije.

3.1.3 Error

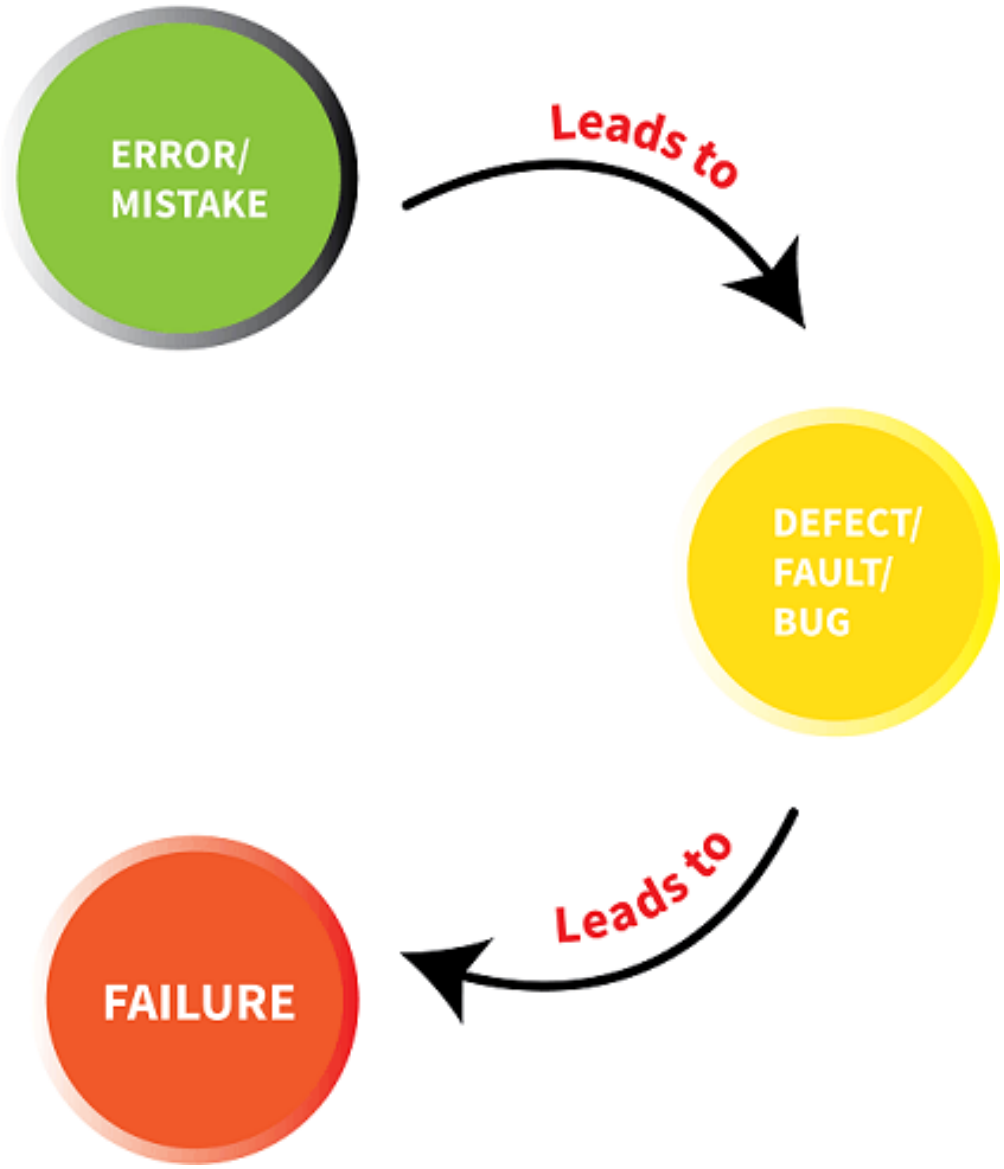
Pogreške u programerskom kodu dovode do errora koji rezultira nemogućnošću pokretanja softvera. Ovakve pogreške obično nastaju zbog nerazumijevanje zahtjeva od strane programera. Razlikujemo errore sintakse, errore korisničkog sučelja, errore u kalkulaciji te hardverske errore.

3.1.4 Neuspjeh (eng. failure)

Neuspjeh se definira kao skup različitih softverskih pogreški koje rezultiraju neuspjehom softvera. Posljedice neuspjeha softvera mogu biti gubitak važnih podataka, gubitak povjerenja korisnika i sl.

3.1.5 Greška (eng. fault)

Greška u softveru generalni je izraz za stanje u kojemu softver ne izvršava zadanu funkciju. Greške su uglavnom izazvane od strane čovjeka odnosno programera, a možemo ih podijeliti na greške poslovne logike, funkcionalne greške, grafičke greške i sigurnosne greške.
Na slici je vidljivo kako pojava erora ili pogreške dovodi do pojave defekta koji potom rezultira neuspjehom softvera.



Slika 2. Odnos pogrešaka u softveru

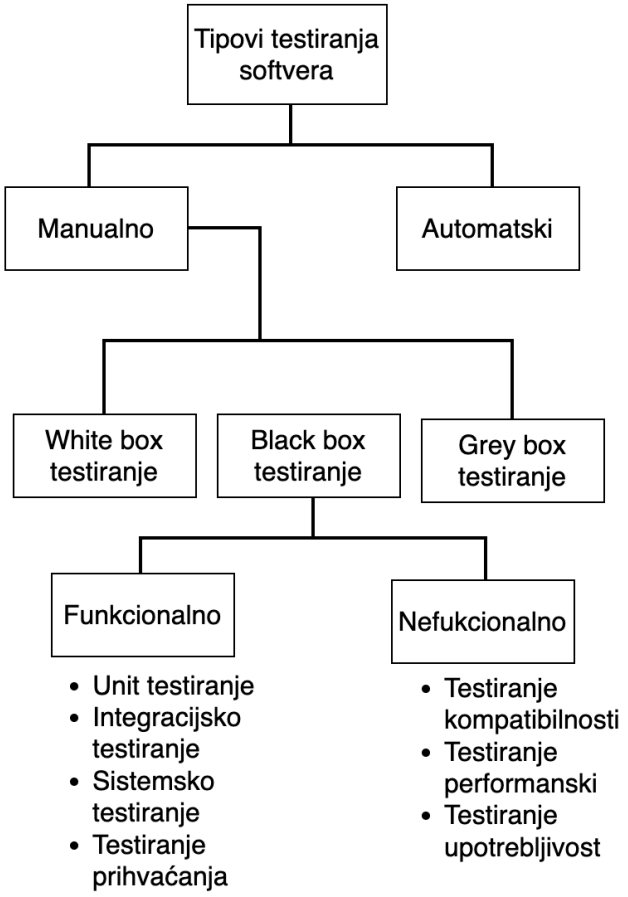
Izvor: https://www.tutorialandexample.com/error-vs-defect-vs-failure-in-software-testing

4. Metodologije testiranja softvera

Metodologije testiranja softvera su sistematični pristupi, okviri i tehnike koje se koriste za planiranje, dizajniranje, izvođenje i upravljanje aktivnostima testiranja softvera. Ove metodologije pružaju strukturiran i organiziran pristup kako bi se osigurala kvaliteta, pouzdanost i efektivnost softverskih sistema.[5]

S obzirom na raznolikost i robusnost današnjih softvera, ne postoji univerzalni set metodologija koji je primjenjiv u testiranju svakog softvera. Testni timovi moraju znati odabrati najbolju strategiju u testiranju softvera unutar okvira projekta. Na slici 3 vidljiva je hijerarhijska podjela testiranje softvera prema tipu, a u nastavku rada će biti detaljno opisan svaki tip i njegove karakteristike.

[5] Mathur, Aditya P. (2008). Foundations of Software Testing. Pearson Education.



Slika 3. Hijerarhija tipova testiranja softvera

Izvor: Autorica

4.1 Manualno testiranje softvera

Manualno testiranje softvera je tip testiranje u kojemu se test scenariji izvršavaju ručno od strane testera bez korištenja alata za automatizaciju. Svrha manualnog testiranja je pronalazak bugova te osiguravanje da softver udovoljava zadanih zahtjevima.

Proces manualnog testiranja započinje analizom zahtjeva. Softver testeri moraju analizirati sve zahtjeve kako bi mogli pripremiti testne scenarije te osigurati što veću pokrivenost testovima. Nakon analize zahtjeva, testeri moraju razviti plan i strategiju testiranja. Prema razvijenoj testnoj strategiji, izrađuju se testni scenariji koji uključuju korake u testiranju i očekivane rezultate testiranje. Softveri testeri trebaju izvršavati testne scenarije prema planu te

pratiti ponašanje softvera. Svi rezultati se trebaju prikupiti i analizirati te je potrebno pripremiti izvješće o rezultatima testiranja. Izvješće o rezultatima testiranja (eng. bug report) sadrži detaljan opis svih pronađenih pogreški u toku manualnog testiranja. U izvješću mora postojati jasan opis problema, koraci za reproduciranje, očekivano ponašanje te trenutno ponašanje softvera. Izvješće o rezultatima testiranje potrebno je nadopuniti dodatnim materijalima kao što su slike zaslona na kojemu je pronađena greška, video materijalima ili error porukama.

Kako bi manualno testiranje softvera bilo uspješno, važno je da softver testeri prvenstveno imaju duboko razumijevanje samog proizvoda kojeg testiraju. Razumijevanjem softvera tj. proizvoda testeri osiguravaju veću pokrivenost test scenarijima te generalnu kvalitetu softvera. Osim toga, važno je da softver testeri obrate pozornost na detalje kako bi preciznost bila što veća. Dobre komunikacijske vještine, fleksibilnost i želja za suradnjom s drugim članovima koji sudjeluju u razvoju softvera su odlike dobrog softver testera.

4.1.1 Prednosti manualnog testiranja softvera su:

- Prilagodljivost - Testni scenariji koji se izvršavaju ručno mogu se lako mijenjati te prilagođavati brzim promjenama softvera.
- Isplativost – Manualno testiranje je isplativo posebice za manje projekte jer ne zahtjeva ulaganje u različite alate za automatiziranje testova, certifikate, itd.
- Ljudski utjecaj – Manualno testiranje izvršavaju ljudi, koji mogu značajno doprinijeti svojom perspektivom i razumijevanjem krajnjeg korisnika.
- Rano otkrivanje – Ozbiljni nedostaci u softveru mogu se otkriti u ranim fazama razvoja ukoliko se primjenjuje manualno testiranje.
- Jednostavnost – Izvršavanje manualnih testova je jednostavno i ne iziskuje dodatne alate.

4.1.2 Mane manualnog testiranja softvera su:

- Vremenski zahtjevno – Manualno testiranje zahtjeva puno vremena jer se svi testni scenariji izvršavaju ručno. Složeni softveri zahtijevaju velik broj ljudskih resursa za rad na testiranju, što odnosi vrijeme, a i resurse.

- Ljudske greške – Mogućnost ljudske greške u manualnom testiranju je velika. Manualno testiranje može biti zamorno i iscrpljujuće, a pad koncentracije testera može izazvati značajan propust.
- Nemogućnost ponovne upotrebe – Većina manualnih testnih scenarija nisu ponovno upotrebljivi, odnosno svaki novi softver zahtjeva nove testne scenarije. Automatizirani testovi se sastoje od skripti koje je moguće ponovno upotrebljavati.
- Cijena – Manualno testiranje može biti skupo, u slučaju velikih projekata koji zahtijevaju česte verzije novog softvera. To može povećati troškove testiranja jer je potreban značajan broj resura (zaposlenici, vrijeme, oprema za rad).
- Mjerenje rezultata – Gotovo je nemoguće kvantitativno izraziti proces manualnog testiranja, a samim tim teško je procijeniti učinkovitost manualnog testiranja i postignutu pokrivenost test scenarijima.

4.2 Automatizirano testiranje

Automatizirano testiranje odnosi se na automatiziranje procesa manualnog testiranja. Koristi se za zamjenu ili nadopunu manualnog testiranja s nizom alata za testiranje. Alati za automatizirano testiranje pomažu testerima softvera da procijene kvalitetu softvera automatizacijom mehaničkih aspekata softvera. Automatizirano testiranje je značajno promijenilo svijet testiranja softvera te upotrebom pravih alata moguće je postići brže, točnije i mjerljivije rezultate testiranja. Kao i u procesu manualnog testiranja, automatizirano testiranje mora pratiti korake kao što su planiranje testova, priprema testnog okruženja i priprema testnog materijala. Testni materijal uključuje skripte koje su napisane u odabranom programskom jeziku. Testne skripte zamišljene su kao način za oponašanje radnji krajnjeg korisnika. Izvršavanjem testne skripte, analiziraju se i validiraju dobiveni rezultati. Automatizirano testiranje zahtjeva povremeno održavanje, nadopunjavanje i ažuriranje testnih skripti. Kako se softver razvija i mijenja, tako se testne skripte moraju prilagođavati.
Najpoznatiji alati za automatizirane testove su:

1. Selenium – popularni alat koji se koristi za automatizaciju mobilnih aplikacija na

različitim platformama. Razina integracija s različitim uređajima i emulatorima je vrlo visoka te zbog toga je korištenje ovog alata vrlo lako

2. Appium – alat koji je besplatan za sve korisnike te ga je moguće prilagoditi za testiranje svih vrsta softvera. Appium podržava različite programske jezike, lako se integrira i okuplja velik broj korisnika.

3. JUnit – najpopularniji alat za testiranje Java aplikacija. Omogućava pisanje testova na jednostavan i fleksibilan način, te pomaže programerima da provjere svaku pojedinačnu jedinicu koda.

4.2.1 Prednosti automatiziranog testiranja softvera

- Brzina - Automatizirani testovi mogu se izvršavati puno brže u usporedbi s manualnim testovima. Za vrijeme izvršavanja testne skripte, softver testeri mogu odrađivati druge zadatke, a po završetku izvršenja skripte, analizirati dobivene rezultate.
- Pouzdanost - Automatizirani testovi izvršavaju se prema unaprijed napisanim skriptama te prate zadane uvjete smanjujući mogućnost ljudske pogreške. Rezultati automatiziranog testiranja mogu se ocijeniti kao pouzdani i mjerljivi.
- Kontinuirano testiranje - Kontinuirano testiranje odnosi se na neprekidan niz izvršenih automatiziranih testnih skripti koje mogu u svakom trenutku uočiti pogrešku u softveru. Kontinuirano testiranje omogućava lakši i brži način validiranja softvera.
- Izvještavanje - Moderni alati za automatizirane testove nude bezbroj mogućnosti generiranja izvještaja o stanju softvera. U usporedbi s izvještavanjem kod manualnog testiranja, izvještavanje automatiziranih testova je brže, preciznije i detaljnije.
- Pokrivenost softvera testovima – Kompleksni softveri zahtijevaju veliku pokrivenost testnim scenarijima, a automatizirani testovi omogućavaju skoro potpuno pokrivenost gotovo bilo kojeg softvera. Prema World Quality Reportu iz 2020-21 zahtjev za

pokrivenošću softvera testovima je primarni parametar za ocjenjivanje kvalitete aplikacija i efikasnosti alata za automatizirano testiranje. [6]

4.2.2 Mane automatiziranog testiranja softvera

- Cijena – Automatizirani testovi te alati za korištenje istih mogu zahtijevati velike početne financijske izdatke. Također, cijena radnika koji su specijalizirani za automatizirano testiranje dostiže sve veće iznose. Automatizirani testovi nisu isplativi za male projekte.
- Testiranje korisničkog iskustva – Automatizirani testovi ne mogu testirati softver iz perspektive korisnika, ne mogu ocijeniti razinu ergonomije te realne upotrebljivosti aplikacije koja se testira.

  Otklanjanje pogrešaka (eng. debugging) – Nakon izvršavanja automatiziranih testnih skripti, ukoliko softver tester nema visoko razvijeno znanje o programskom kodu te dobru suradnju s timom programera, izazovno je samostalno otkriti i ukloniti pronađene pogreške.
- Sigurnost – Automatizirani testovi pružaju visoku razinu točnosti, no ne mogu u potpunosti garantirati odsutnost pogreški. Moguće je da automatizirane testne skripte pruže lažno pozitivne rezultate. Preporučljivo je da se uz automatizirano testiranje uvijek uključi i manualno testiranje.
- Održavanje – Automatizirani testovi i testne skripte zahtijevaju konstantno održavanje i ažuriranje kako bi se prilagodili promjenama softvera. Izmjene testnih skriptu mogu iziskivati puno vremena te resursa. Ukoliko se testne skripte ne ažuriraju, automatizirani testovi mogu izgubiti na pouzdanosti što može dovesti do velikih propusta u procesu testiranja.

[6] https://www.sogeti.com/explore/reports/world-quality-report-2020/

5. Metode testiranja

5.1 Testiranje bijele kutije (eng. white box testing)

Testiranje bijele kutije (poznato i kao clear box testiranje, glass box testiranje i strukturno testiranja) je metoda testiranja u kojoj se ispituje unutarnja struktura softvera, dizajn i programski kod. Testiranje bijele kutije uključuje analizu strukture programskog koda te analizu implementacije softvera. Ova metoda testiranja provodi se kako bi se validirala logika programskog koda. Primarna zadaća testiranja bijele kutije je provjera odgovaraju li predefinirani inputi dobivenim outputima. Za ovu metodu testiranja potrebno je znanje programiranja, stoga je praksa da testiranje bijele kutije provode programeri te nakon toga daljnje testiranje preuzima testni tim. Neke od tehnika koje se koriste su: testiranje putanja, testiranje petlji, testiranje uvjeta, testiranje performansi. Prednosti testiranja bijele kutije su optimizacija programskog koda, lak proces izrade automatiziranih testova te detaljni uvid u kod i sve njegove funkcionalnosti. Testiranje bijele kutije također omogućava jasan uvid u rezultate testiranja te može pokazati softverske greške u ranoj fazi razvoja softvera. Primjer testiranja bijele kutije je testiranje kalkulatora. U slučaju testiranja crne kutije, softver testeri bi fokus stavili na to je li rezultat zadane kalkulacije ispravan, dok se testiranjem bijele kutije provjerava način na koji je dobiven rezultat izračunat, odnosno koja je struktura i logika u radu kalkulatora. Ova metoda testiranja ima i nedostatke kao što su uloženo vrijeme i visoki troškovi, no unatoč tome metoda testiranje bijele kutije jedna je od prihvaénijih metoda u testiranju softvera.

5.2 Testiranje crne kutije (eng. black box testing)

Slika 4. Prikaz testiranja crne kutije

Izvor: https://www.javatpoint.com/black-box-testing

Testiranje crne kutije je pristup testiranju softvera u kojemu osoba koja testira softver nije upoznata s detaljima i specifikacijama programskog koda koji testira. Fokus je na testiranju funkcionalnosti i ponašanju softvera na temelju ulaznih varijabli te očekivanih izlaznih varijabli. Softver tester tretira softver kao crnu kutiju te sa softverom komunicira samo putem korisničkog sučelja. Ukoliko zadani input nije rezultirao očekivanim outputom, test se smatra neuspješnim te se pogreška prijavljuje programerskom timu na uvid. Koraci u testiranju crne kutije su jednostavni. Prvi korak je analiza zahtjeva, zatim se kreiraju testni scenariji s različitim inputima (pozitivnim i negativnim), nakon toga se izvršavaju svi kreirani testni scenariji, a naposlijetku se analiziraju dobiveni rezultati. Testiranje crne kutije može se provoditi na svim razinama testiranja.

Najpoznatije tehnike su:
Tablica odlučivanja – Ova tehnika koristi se za provjeru ponašanja softvera u slučaju različitih kombinacija zadanih inputa. Zadani inputi te dobiveni outputi upisuju se u tabelarnu formu te takva tablica može doprinijeti provjeri svih mogućih kombinacija u testiranju jedne komponente. Prednost ove tehnike je da je vrlo jednostavna, lako se tumači, ima mogućnost velike pokrivenost testnim scenarijima.

Condition	Input 1	Input 2	Input 3	Input 4
Username	False	False	True	True
Password	False	True	False	True
Output	Error	Error	Error	Valid

Slika 5. Prikaz tablice odlučivanja

Izvor: https://www.turing.com/blog/black-box-testing-vs-white-box-testing-understanding-key-differences/

Analiza graničnih vrijednosti – Ovom tehnikom testiraju se granične vrijednosti, pri čemu su te vrijednosti one koje sadrže gornju i donju granicu varijabli. Tehnikom analize graničnih vrijednosti provjerava se je li output točan ili ne. Testiranjem na ovaj način smanjuje se broj testnih scenarija.

Testiranje klasa ekvivalencije – Ova tehnika koristi se kako bi se smanjio broj mogućih testnih scenarija uz uvjet da se zadrži razumna pokrivenost testova. Ideja iza ove tehnike testiranja je da se inputi koji imaju isti output podijele u različite klase ili grupe, potom se testni scenariji dizajniraju prema svakoj ekvivalentnoj klasi, kako bi se osigurala pokrivenost testova za sve reprezentativne vrijednosti.

Glavna prednost testiranja crne kutije je da su testovi orijentirani prema onomu kako se softver treba ponašati te prema zadovoljavanju krajnjih korisnika. Ova metoda je vremenski ne zahtjevna te se može provoditi na različitim platformama i operativnim sustavima. Softver testeri ne moraju imati programersko znanje, dovoljno je razumijevanje zahtjeva za softver koji se testira.

Nedostaci ove metode su očituju u tome što se testovi ne mogu unaprijed kreirati, ponajviše zbog nedostatka specifikacija softvera. Osim toga, kada se softver testeri susretnu s pogreškom u sustavu, teško je doći do korijena samog problema s obzirom da se testiranjem crne kutije testira samo korisničko sučelje.

5.3 Testiranje sive kutije (eng. gray box testing)

Testiranje bijele kutije odnosi se na testiranje strukture softvera, testiranje crne kutije odnosi se na testiranje funkcionalnosti softvera, a testiranje sive kutije objedinjuje te dvije metode u jednu. Ova metoda može se nazvati i hibridnom metodom jer omogućava softver testerima dublje znanje o samom sustavu, no istovremeno zadržavajući razinu neovisnosti o njegovom funkcioniranju. Softver testeri nemaju potpuno razumijevanje logike programskoga koda, no imaju uvid u pojedine dijelove kao što su baze podataka. Ovakav pristup omogućava softver testerima da kreiraju specifične testne scenarije.

Prednost testiranje sive kutije je da može otkriti pogreške koje ne bi bile pronađene korištenjem neke druge metode poput testiranja crne kutije. Ova metoda može se provoditi na svim razinama životnog ciklusa testiranja softvera. Korištenjem ove metode postiže se ravnoteža između vanjskog dijela softvera (korisničkog sučelja) te unutarnje strukture. Mogući problemi ove metode su ponavljajući testni scenariji te propuštene pogreške zbog nemogućnosti uvida u programski kod.

5.3.1 Razlike između testiranja bijele i crne kutije

Metode testiranja bijele i crne kutije imaju isti cilj, a to je pronalaženje pogrešaka u softveru, no razlikuju se u određenim principima, tehnikama i postupcima testiranja.

Testiranje crne kutije zasniva se na testiranju funkcionalnosti softvera bez poznavanja pozadinske logike, dok se u testiranju bijele kutije poznaje pozadinska logika i struktura. Razina programerskog znanja za provođenje metode testiranja crne kutije je minimalna ili nikakva, dok je za testiranje bijele kutije obavezno poznavanje nekog programskog jezika. Automatizirani testovi se mogu lako kreirati koristeći metodu testiranja bijele kutije, dok je to teže izvodivo ukoliko se koristi metode crne kutije zbog međusobne ovisnosti programerskog i testnog tima. Za testiranje softvera metodom crne kutije potrebno je uložiti manje vremena jer ona ovisi o dostupnosti funkcionalnosti softvera, a metoda testiranja bijele kutije može oduzeti puno više vremena jer se testira sam kod. Samim tim, metoda testiranja bijele kutije karakterizira se kao

metoda koja je iscrpljujuća i naporna za provođenje, za razliku od metode testiranja crne kutije. Prednost metode testiranja bijele kutije naspram metode testiranje crne kutije je ta da ona omogućava pronalaženje grešaka duboko u kodu te doprinosi optimizaciji koda, dok metoda testiranja crne kutije otkriva pogreške samo na razini korisničkog sučelja.

Obje metode predstavljaju dva značajna pristupa testiranju softvera te svaki pristup nudi različite jedinstvene mogućnosti za osiguranje kvalitete softvera.

6. Funkcionalno i nefunkcionalno testiranje

6.1 Funkcionalno testiranje

Funkcionalno testiranje se uglavnom odnosi na testiranje funkcionalnost softvera te služi za provjeru baznih funkcija, upotrebljivosti i pristupačnosti softvera za krajnjeg korisnika.

6.1.1 Unit testiranje

Unit testiranje je proces testiranja individualnih dijelova softvera. Umjesto testiranja softvera u cijelini, fokus se stavlja na manje dijelove. [7] Unit testiranje odvija se za vrijeme razvoja softvera te se provodi od strane programera. Unit testovi se mogu provoditi manualnom ili automatizirano. Najpoznatiji alati za automatiziranje unit testova su: Junit, NUnit, JMockit, PHPUnit i EMMA. Mnogobrojne su prednosti pisanja unit testova, a najznačajnija je ta da tehnike unit testiranja omogućavaju testiranje pojedinačnih dijelova koda neovisno o tome jesu li drugi dijelovi gotovi. Na taj način postiže se ranija detekcija pogreški te veća mogućnost za popravak i optimizaciju koda. Unit testovi također mogu poslužiti kao svojevrsna dokumentacija koja se sastoji od realnih primjera kako bi se softver trebao ponašati te koji su

[7] Myers, Glenford J. *The Art of Software Testing*, 2nd edition.

očekivani rezultati. Unit testiranje nije dovoljno za potpunu pokrivenost, već je potrebno provesti i druge tehnike testiranja.

6.1.2 Integracijsko testiranje

Integracijsko testiranje je proces testiranja koji je fokusiran na provjeru interakcija i razmjene podataka između različitih komponenti softvera. Ono se provedi nakon što je dovršeno unit testiranje. Cilj integracijskog testiranja je prepoznati potencijalne pogreške koje nastaju kombiniranjem dvije ili više komponente softvera. Ovaj način testiranja ima nekoliko različitih strategija, a to su:

- Big Bang integracija
- Integracija od vrha prema dnu (eng. Top-down integration)
- Integracija od dna prema vrhu (eng. Bottom-up integration)
- Sendvič integracija
- Integracija po grafu poziva

Za uspješno provođenje integracijskog testiranja važno je odrediti strategiju koja će se primjenjivati, pripremiti testne podatke te kreirati testne scenarije. Ovisno o specifičnostima softvera koji se testira, razvijaju se testni scenariji te se uključuju različite komponente u proces testiranja.

6.1.3 Testiranje prihvaćanja

Testiranje prihvaćanja ili testiranje korisničkog prihvaćanja (eng. UAT – User Acceptance Testing) definira se kao tehnika testiranja koja odlučuje zadovoljava li softver uvjete krajnjih korisnika. Ova tehnika omogućuje prikupljanje mišljenja korisnika o softveru i njegovim funkcionalnostima te doprinosi programerskom i testnom timu.

Cilj ove tehnike testiranja je pronalaženje grešaka u softveru koje mogu zadati probleme korisnicima koji će koristiti taj softver.

Uobičajeni koraci u izvođenju testiranja prihvaćanje su analiza zahtjeva, kreiranje testnih scenarija, priprema testnih podataka, izvršavanje testova te potvrda izvršenih testova.

Razlikujemo 2 tipa testiranja prihvaćanja.
Alpha testiranje - Proces testiranja koji se provodi prije nego što se softver proslijedi korisnicima na testiranje. U ovom procesu cilj je simulirati ponašanje krajnjeg korisnika te biti što objektivniji.
Beta testiranje – Proces testiranja koji izvode krajnji korisnici u realnim uvjetima. U ovom procesu cilj je otkriti nedostatke softvera u što ranijoj fazi kako bi se sve potencijalne pogreške stigle analizirati prije isporuke softvera svim korisnicima.

Primjer testiranja prihvaćanja može biti sljedeći:
Pretpostavimo da je razvijen novi softver za fiskalnu blagajnu u restoranima. U alpha fazi testiranja, softver testeri zaduženi su za simuliranje rada na blagajni te kreiranja testnih scenarija koji su što sličniji stvarnima. U toj fazi moguće je testirati dodavanje hrane u narudžbu, dodavanje napojnice, brisanja hrane iz narudžbe, printanje računa i slično. Primjer testnog scenarija napisan u koracima:
- Ulogiraj se u blagajnu s odgovarajućim PIN-om
- Dodaj jelo iz menija u narudžbu
- Dodaj piće iz menija u narudžbu
- Zaključi narudžbu
- Izdaj račun

6.2 Nefunkcionalno testiranje

Nefunkcionalno testiranje definira se kao vrsta testiranja softvera koja provjerava aspekte softvera kao što su performanse, upotrebljivost i pouzdanost softvera. Ovakvim načinom testiranja postiže se optimizacija softverskih značajki, održavanja i izvještavanja o radu softvera.

6.2.1 Testiranje kompatibilnosti

Testiranje kompatibilnosti je proces testiranja provjere kako softver funkcionira u različitim

okruženjima, operativnim sustavima, na različitim uređajima i preglednicima. Cilj ovakvog testiranje je da se softver ispravno prikazuje, funkcionira i surađuje na svim platformama.
Razlikujemo dvije vrste testiranja kompatibilnosti ovisno o verziji softvera:

1. Testiranje napredne kompatibilnosti – Kompatibilnost i ponašanje softvera provjerava se koristeći najnoviju verziju softvera
2. Testiranje povratne kompatibilnosti – Kompatibilnost i ponašanje softvera provjerava se koristeći prijašnju verziju softvera

6.2.2 Testiranje performansi

Testiranje performansi je proces testiranje softvera koji se primarno koristi za testiranje brzine, pouzdanosti, stabilnosti, skalabilnosti i korištenja resursa softvera pod određenim opterećenjem. Testiranjem brzine utvrđuje se brzina odgovora na zahtjev korisnika te se samim tim ocjenjuje korisničko iskustvo. Skalabilnost mjeri koje je maksimalno opterećenje koje softver može podnijeti. Stabilnost se ispituje stvaranjem različitih opterećenja na softver te praćenjem ponašanja pod opterećenjem.

Najčešći problemi na koje se nailazi za vrijeme testiranje performansi su:

1. Dugo vrijeme učitavanja – vrijeme učitavanja označava potrebno vrijeme za pokretanje aplikacije.
2. Stvaranje uskog grla (eng. network bottleneck) – Usko grlo predstavlja prepreke u sustavu koje nastaju zbog programerskih pogrešaka te pogoršavaju performanse softvera
3. Loša skalabilnost – Skalabilnost predstavlja razinu opterećenja koju softver može podnijeti

6.2.3 Testiranje upotrebljivosti

Testiranje upotrebljivosti poznato i kao testiranje korisničkog iskustva (UX) definira se kao proces testiranja softvera koji mjeri i iskazuje koliko je softver jednostavan za korištenje. Testiranje upotrebljivosti fokusira se na lakoću uporabe, fleksibilnost u korištenju te sposobnost

softvera da ispuni zahtjeve korisnika. Glavni cilj ove tehnike je da se softver napravi intuitivnim kako bi korisnici mogli s lakoćom koristiti softver bez nepotrebnog napora. U proces testiranja upotrebljivosti važno je uključiti i stvarne korisnike koji mogu pružiti vrijedne informacije o svom iskustvu korištenja. Važno je pratiti standarde industrije koji se odnose na dizajn i upotrebljivost kako bi se osigurala kvaliteta i korisničko zadovoljstvo.

7. Osiguravanje kvalitete softvera (eng. Software Quality Assurance – SQA)

Osiguravanje kvalitete softvera (SQA) je proces koji osigurava da su svi procesi, metode, aktivnosti i elementi razvoja softvera popraćeni i usklađeni s definiranim standardima. Osiguranje kvalitete softvera je proces koji se odvija paralelno s razvojem softvera. Fokus je na generalnom poboljšanju procesa razvoja softvera kako bi se potencijalni problemi mogli spriječiti prije nego što postanu ozbiljan problem.

7.1 Razlika između osiguravanja kvaliteta softvera (SQA) i kontrole kvalitete (eng. Quality Control – QC)

Osiguravanje kvalitete softvera brine o svim bitnim stavkama koje mogu utjecati na krajnji proizvod. Ova metode brine o pristupima, tehnologijama i metodama korištenim u razvoju softvera te analizira implementaciju istih. Osiguravanje kvalitete softvera provodi se prije kontrole kvalitete, odnosno u cijelom procesu razvoja. Kontrola kvalitete fokusirana je na prepoznavanju pogrešaka i propusta u proizvodu nakon što razvoj završi. Kontrola kvalitete može se definirati kao korektivni alat, dok je osiguravanje kvalitete softvera menadžerski alat.

OSIGURAVANJE KVALITETE SOFTVERA	KONTROLA KVALITETE
Prevencija pogrešaka	Detekcija pogrešaka
Planiranje	Djelovanje
Verifikacija	Validacija

Slika 6. Razlike između osiguravanja kvalitete softvera i kontrole kvalitete

Izvor: Autorica

7.2 Ciljevi osiguravanja kvalitete softvera

Ciljevi osiguravanja kvalitete uključuju:

1. Osiguravanje zadovoljstva korisnika: Jedan od najvažnijih ciljeva osiguravanja kvalitete je isporuka proizvoda ili usluga koji zadovoljavaju ili premašuju očekivanja korisnika, rezultirajući visokim stupnjem zadovoljstva korisnika.
2. Usklađivanje sa standardima i propisima: Osiguranje kvalitete osigurava da se svi procesi odvijaju prema standardima i propisima industrije.
3. Prevencija pogrešaka: Cilj osiguranja kvalitete je identificirati i riješiti potencijalne pogreške i probleme u ranim fazama razvoja softvera, smanjujući vjerojatnost da pogreške u softveru dođe do krajnjeg korisnika.

4. Unaprjeđivanje procesa: Osiguranje kvalitete uključuje kontinuirano i unaprjeđivanje procesa razvoja i isporuke radi poboljšanja učinkovitosti, produktivnosti i kvalitete rezultata.
5. Upravljanje rizicima: Osiguranje kvalitete ima za cilj identificirati i smanjiti rizike te unaprijediti sigurnost, pouzdanost i usklađenost proizvoda.
6. Kontinuirano praćenje i evaluacija: Osiguranje kvalitete uključuje kontinuirano praćenje, mjerenje i evaluaciju kvalitativnih metrika radi identifikacije područja za poboljšanje i osiguranja učinkovitosti mjera kontrole kvalitete.
7. Razvoj vještina: Osiguranje kvalitete ima za cilj poticanje razvoja vještina i znanja među članovima tima radi poboljšanja njihove sposobnosti za isporuku što boljeg proizvoda.
8. Kontinuirano poboljšanje: Osiguranje kvalitete fokusira se na uspostavljanje kulture kontinuiranog poboljšanja, u kojoj se naučene lekcije iz prethodnih projekata i iskustava ugrađuju u buduće procese radi poboljšanja ukupne kvalitete i performansi.

7.3 Plan osiguravanja kvalitete softvera

Plan za osiguravanje kvalitete softvera ključni je dokument koji definira pristup, metodologije, aktivnosti i resurse potrebne za osiguravanje kvalitete softvera. Prvi korak u planu osiguravanje kvalitete odnosi se na definiranje ciljeva i opsega projekta na kojemu se radi. Važno je imati definirane sve zahtjeve i očekivanja menadžmenta vezano za razvoj softvera.

Sljedeći korak se odnosi na definiranje aktivnosti i procesa koji će biti provođeni. U ovom koraku odlučuje se koje aktivnosti će provoditi testni tim, to mogu biti analiza zahtjeva, kreiranje testnih scenarija, kreiranje automatiziranih testova itd. Sve aktivnosti moraju biti dokumentirane te detaljno opisane, uz navedene alate koji će biti korišteni. Treći korak u planiranju je definiranje uloga i podjela odgovornosti unutar tima. Kako bi se proces osiguravanja kvalitete softvera proveo što lakše, važno je podijeliti uloge te odgovornosti. Jasno definirane uloge pridonose učinkovitijoj suradnji unutar tima. Za učinkovitu suradnju i komunikaciju važno je uspostaviti mehanizme komunikacije i izvještavanja. Plan treba sadržavati kanale i način komunikacije između članova tima, menadžmenta i ostalih važnih dionika. Izvještavanje je vrlo bitan element osiguravanja kvalitete softvera, a plan treba opisati koji su zahtjevi za izvještavanjem, kolika je učestalost izvještavanja te tko je zadužen za

praćenje izvještaja. Posljednji korak u planu osiguravanja kvalitete je definiranje kriterija kvalitete. Kriteriji kvalitete moraju imati mjerne pokazatelje kao što su pokrivenost softvera testnim scenarijima, usklađenost sa standardima industrije, zadovoljstvo krajnjeg korisnika. Mjerni pokazatelji moraju imati referentne vrijednosti kako bi se osigurala objektivna analiza dobivenih rezultata.

7.4 Najbolje prakse u osiguravanju kvalitete softvera

Nakon izrađenog plana za provođenje osiguravanje kvalitete, potrebno je započeti aktivnosti osiguravanja kvalitete. Aktivnosti se mogu razlikovati ovisno o kompleksnosti i specifičnosti softvera, no neki procesi su se pokazali kao dobra praksa u mnogobrojnim industrijama.

7.4.1 Uključivanje testnog tima

Testni tim sastoji se od softver testera te inženjera za osiguravanje kvalitete. Za kvalitetno provođenje svih planiranih aktivnosti u osiguravanju kvalitete softvera važno je da testni tim bude uključen u proces razvoja softvera od samoga početka. Osobito je važno da testni tim razumije postavljene zahtjeve kako bi kasnije efikasnije provodio testiranje softvera. Ovakva praksa smanjuje mogućnost grešaka i otklanja potencijalne probleme u početnim fazama razvoja softvera.

7.4.2 Postavljanje kontrolnih točaka

Kontrolne točke odnose se na jasno određeni trenutak u toku razvoja, u kojemu se analiziraju dosadašnji rezultati i napredak. Kontrolne točke omogućuju mjerenje uspjeha te otkrivaju potencijalne probleme ili zastoje u razvoju. Sustavnim praćenjem razvoja softvera olakšava se usklađivanje istog sa standardima industrije, zahtjevima menadžmenta te korisnika.

7.4.3 Odabir višestrukih strategija testiranja

Odabirom samo jedne strategije testiranja razina kvalitete softvera ne može biti jednaka kao kada se softver promatra i testira iz različitih strategijskih perspektiva. Kombinacijom više strategija postiže se učinkovitije, preciznije i sigurnije testiranje. Potrebno je obratiti posebnu pozornost na sigurnosni aspekt softvera koji se mora testirati u svakoj fazi razvoja.

8. Zaključak

Testiranje softvera postavlja se kao standardni proces u IT industriji, neovisno o specifičnosti softvera koji se razvija. Testiranje omogućava proizvod koji je napravljen prema standardima industrije i očekivanjima korisnika. Životni ciklus testiranja softvera određuje faze koje moraju biti poštovane da bi se ostvario krajnji cilj. Razvijene su različite metodologije i pristupi testiranju softvera, a najvažnija podjela je podjela na manualno i automatizirano testiranje te na funkcionalno i nefunkcionalno testiranje. Metodologije se biraju ovisno o zahtjevima softvera, a uz pravilnu kombinaciju moguće je osigurati visoku razinu kvalitete. Ulaganje u osiguravanje kvalitete softvera je odluka za koju se može reći da je dugoročno isplativa. Svaka tvrtka koja želi osigurati stabilan i siguran softver svojim korisnicima treba razumjeti da je osiguranje kvalitete softvera proces koji ubrzava i olakšava postizanje uspjeha na digitalnom tržištu.

Bibliografija

Black box Testing. Javatpoint. Dostupno na: https://www.javatpoint.com/black-box-testing

Chopra, Rajiv. *Software Testing: Principles and Practices*. Mercury Learning & Information, 2018. Homes, Bernard. *Fundamentals of Software Testing*. Wiley-ISTE, 2011.

Integration Testing. Javatpoint. Dostupno na: https://www.javatpoint.com/integration-testing

Isha, Sunita Sangwan. *Software Testing Techniques and Strategies*. Isha International Journal of Engineering Research and Applications, vol. 4, issue 4 (Version 9), April 2014, pp. 99-102. ISSN 2248-9622.Dostupno na: www.ijera.com.

Lewis, William E. *Software Testing and Continuous Quality Improvement*. Chicago: ABC Publications, 2008.

Manual Testing. Art of Testing. Dostupno na:. https://artoftesting.com/manual-testing.

Myers, Glenford J., Tom Badgett, Todd M. Thomas, and Corey Sandler. *The Art of Software Testing*. Business Data Processing S. John Wiley & Sons Inc, 2004.

Quality Assurance, Quality Control and Testing – The Basics of Software Quality Management. Altexsoft. Dostupno na: https://www.altexsoft.com/whitepapers/quality-assurance-quality-control-and-testing-the-basics-of-software-quality-management/

Sfetsos, Panagiotis. *Agile Software Development Quality Assurance*. Information Science Reference, 2007.

Software Engineering, Software Quality Assurance. Geeks for Geeks. Dostupno na: https://www.geeksforgeeks.org/software-engineering-software-quality-assurance/

Software Quality Assurance. Javatpoint. Dostupno na: https://www.javatpoint.com/software-quality-assurance

Software Testing Methodologies. Interviewbit. Dostupno na: https://www.interviewbit.com/blog/software-testing-methodologies/

Tian, Jeff. *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. Wiley-IEEE Computer Society Press, 2005.

What is Software Quality Assurance (SQA)?. Nioyatech. Dostupno na: https://nioyatech.com/software-quality-assurance/

White Box Testing Tutorial: What is, Example, Tools & Techniques. Guru 99. Dostupno na: https://www.guru99.com/white-box-testing.html

Popis ilustracija