Uloga QA testera u razvoju softvera: tehnologije, alati i izazovi

Jelena Mitrović
1019/2024
Matematički fakultet, Univerzitet u Beogradu
Seminarski rad u okviru predmeta Verifikacija softvera

14. novembar 2024.

Sažetak

Ovaj rad analizira evoluciju QA testiranja, počevši od svojih korena u industrijskoj proizvodnji do današnjih savremenih metoda koje uključuju automatizaciju, veštačku inteligenciju i testiranje u kontinuitetu. Fokus je na manuelnom testiranju, istraživačkom testiranju i razvoju vođenom ponašanjem (BDD), kao i ključnim tehnologijama kao što su Selenium, Appium i Jenkins koje omogućavaju efikasnu automatizaciju testiranja. Takođe, razmatrani su tehnološki trendovi u QA testiranju, poput primene veštačke inteligencije i mašinskog učenja, kao i izazovi sa kojima se QA testeri suočavaju u brzorastućem svetu softverskog razvoja.

Sadržaj

1 Uvod

Kvalitetno osiguranje (QA) predstavlja ključnu disciplinu u modernom razvoju softverskih sistema, a njegova uloga je prošla kroz značajnu evoluciju od svojih početaka u industriji 20. veka. Kako su softverski sistemi postajali sve složeniji, proces testiranja je neprestano napredovao, prilagođavajući se novim izazovima, od prvih formalnih metoda do savremenih tehnika automatizacije i veštačke inteligencije. Danas QA ne samo da osigurava ispravnost i funkcionalnost softverskih sistema, već igra i ključnu ulogu u unapređenju korisničkog iskustva, smanjenju grešaka i povećanju efikasnosti u razvoju softvera. Ovaj rad pruža pregled istorijskog razvoja QA metodologija, ključnih tehnika i tehnologija, kao i izazova i trendova koji oblikuju ovu oblast.

2 Evolucija QA testiranja

QA testiranje prošlo je kroz značajnu evoluciju od svojih začetaka u prvim decenijama 20. veka. U početku je fokus bio na kontroli kvaliteta u industrijskoj proizvodnji, gde su se statističke metode koristile za analizu i unapređenje kvaliteta proizvoda. Kako je softver postajao sve složeniji, tokom 1950-ih i 1960-ih, inženjeri su počeli da primenjuju slične principe u razvoju softvera. Tokom ovog perioda razvijene su prve formalne metode testiranja i verifikacije, čime su postavljeni temelji za QA testiranje kao disciplinu.

S porastom složenosti softverskih sistema tokom 1980-ih, postalo je očigledno da su potrebne nove strategije za osiguranje kvaliteta. Pojava standarda kao što su ISO 9000 i Capability Maturity Model (CMM) omogućila je uspostavljanje formalnih procesa i procedura u QA testiranju. Tokom 1990-ih, koncepti kao što su Test Driven Development (TDD) i Agilna metodologija transformisali su pristup testiranju, omogućavajući integraciju testiranja u razvojne procese, a ne kao poslednji korak.

Tokom 2000-ih i 2010-ih, fokus QA testiranja se dodatno proširio na automatizaciju, CI/CD (kontinuiranu integraciju i kontinuirano isporučivanje) i kolaboraciju između razvojnih i operativnih timova, dok je DevOps postao ključna praksa. Danas se QA testiranje oslanja na inovacije poput veštačke inteligencije i mašinskog učenja, čime se povećava efikasnost i smanjuje vreme potrebno za testiranje. Ova transformacija odražava neprekidnu potragu za kvalitetom u softverskom razvoju, gde je osiguranje kvaliteta postalo sastavni deo celokupnog procesa.

3 Manuelno testiranje

Manuelno testiranje podrazumeva proces provere softvera gde QA testeri ručno izvršavaju test slučajeve bez upotrebe automatizovanih alata. Ovaj tip testiranja omogućava testerima da direktno interaguju sa aplikacijom, ispitujući njenu funkcionalnost, korisničko iskustvo i identifikujući greške koje možda ne bi bile primećene u automatizovanim scenarijima. Manuelno testiranje je ključno za fleksibilne, intuitivne metode testiranja, poput istraživačkog testiranja i razvoja vođenog ponašanjem (BDD).

3.1 Istraživačko testiranje

Istraživačko testiranje predstavlja dinamički pristup koji se oslanja na veštine i intuiciju QA testera. Umesto da se striktno pridržavaju unapred definisanih test slučajeva, testeri aktivno istražuju aplikaciju kako bi identifikovali potencijalne greške i probleme. Ovaj metod omogućava testerima da koriste svoje znanje o funkcionalnostima sistema, kao i prethodna iskustva sa sličnim projektima, kako bi otkrili nepredviđene greške koje mogu proći neprimećene kroz tradicionalne test strategije.

Testeri često koriste različite tehnike, kao što su razmišljanje u hodu, upotreba heuristika i fokusiranje na korisničko iskustvo, kako bi obavili istraživačko testiranje. Na primer, mogu interagovati sa aplikacijom na različite načine, menjajući unos podataka, istražujući različite funkcionalnosti i testirajući granice sistema. Ovaj pristup omogućava brže otkrivanje grešaka i može rezultirati otkrivanjem problema koji se ne javljaju u standardnim test slučajevima.

Istraživačko testiranje se često koristi u kombinaciji s automatizovanim testiranjem kako bi se osiguralo pokrivanje i rutinskih i kritičnih oblasti funkcionalnosti. Ovo je posebno korisno u agilnim timovima, gde su zahtevi često podložni promenama, jer testeri mogu brzo prilagoditi svoj pristup i reagovati na promene.

3.2 Razvoj vođen ponašanjem

Razvoj vođen ponašanjem (engl. *Behavior Driven Development* - BDD) predstavlja pristup razvoju softvera koji se fokusira na definisanje očekivanog ponašanja aplikacije kroz saradnju između članova tima. BDD se zasniva na ideji da svi učesnici, uključujući poslovne analitičare, QA testere i programere, treba da učestvuju u definisanju zahteva kroz jasne i razumljive scenarije.

U BDD-u, QA testeri rade na formulisanju test scenarija koji su napisani u jednostavnom jeziku, obično koristeći Gherkin sintaksu. Ovi scenariji osmišljeni su tako da predstavljaju konkretne situacije koje ilustruju kako bi se korisnici trebali ponašati prilikom interakcije sa aplikacijom. Na primer, scenarijum može opisivati

kako korisnik unosi podatke u obrazac i očekuje potvrdu nakon uspešnog podnošenja. Sintaksa scenarija je najčešće strukturirana kroz formu `Given-When-Then`, što pomaže timu da jasno definiše uslove, radnje i očekivane rezultate.

```
Primer scenarija u Gherkin sintaksi:
Feature: Prijava korisnika
Scenario: Uspešna prijava
Given korisnik je na stranici za prijavu
When korisnik unese ispravno korisničko ime i lozinku
Then korisnik je uspešno prijavljen
```

Jedna od ključnih prednosti BDD-a je to što pomaže u osiguravanju da su svi članovi tima na istoj strani kada je u pitanju razumevanje funkcionalnosti aplikacije. Kroz jasne definicije očekivanog ponašanja, QA testeri mogu razviti automatizovane testove koji direktno odražavaju zahteve, čime se smanjuje rizik od nesporazuma i grešaka.

Osim toga, BDD olakšava ranije otkrivanje grešaka, jer se testovi mogu pisati paralelno s razvojem funkcionalnosti. Ova praksa omogućava bržu povratnu informaciju i brže ispravke grešaka, čime se poboljšava ukupna efikasnost procesa razvoja softvera.

4 Ključne tehnologije za automatizaciju

U svetu automatizacije testiranja, *Selenium* [4], *Appium* [5] i *Jenkins* [6] imaju svoje specifične uloge. *Selenium* je idealan za web aplikacije zbog podrške za različite pregledače i programske jezike, dok *Appium* omogućava testiranje mobilnih aplikacija na iOS i Android platformama korišćenjem istog koda.

4.1 Selenium

U modernom razvoju softvera, automatizacija testiranja postaje sve važnija. Jedan od najpopularnijih alata za testiranje web aplikacija je *Selenium*. Ovaj alat omogućava pisanje testova korišćenjem različitih programskih jezika, poput Java, C# i Python, pružajući time veliku fleksibilnost. Podržava više pregledača, uključujući Chrome, Firefox i Safari, što je ključno za testiranje aplikacija na različitim platformama i osiguranje konzistentnosti rada na svim uređajima.

Slika 1: Selenium logo

Selenium se lako integriše sa alatima kao što su *TestNG* i *JUnit*, omogućavajući efikasno upravljanje testovima i generisanje izveštaja. Korišćenjem *Selenium WebDriver*-a, moguće je simulirati korisničke radnje direktnom interakcijom sa pregledačem, što poboljšava kvalitet testiranja.

4.2 Appium

Za automatizaciju testiranja mobilnih aplikacija, *Appium* je jedan od vodećih alata. Omogućava testiranje nativnih, hibridnih i mobilnih aplikacija na iOS i Android platformama korišćenjem istog koda, što značajno štedi vreme i resurse. Kao alat otvorenog koda, *Appium* pruža fleksibilnost i lako se prilagođava specifičnim potrebama projekta.

Uz podršku za više platformi, Appium omogućava testiranje aplikacija na različitim uređajima bez prepravke koda za svaku platformu. Koristi *WebDriver* protokol, što olakšava prelazak sa web testiranja na mobilno, pružajući poznat pristup onima koji su radili sa Selenium-om.

Slika 2: Appium logo

4.2.1 Primer testiranja kalkulatora u Appium-u

Sledeći primer prikazuje kako koristiti *Appium* za testiranje osnovne funkcionalnosti Android kalkulator aplikacije – konkretno, sabiranja brojeva 5 i 3.

```
from appium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as
    EC

# Podesavanje parametara za povezivanje sa Appium serverom
caps = {
    "platformName": "Android",
    "deviceName": "emulator-5554",
    "appPackage": "com.android.calculator2",
    "appActivity": "com.android.calculator2.Calculator"
}
```

```
# Povezivanje sa Appium serverom
driver = webdriver.Remote("http://localhost:4723/wd/hub", caps)

try:
    # Cekanje da se aplikacija ucita
    wait = WebDriverWait(driver, 10)

    # Klik na dugme "5"
    five_button = wait.until(EC.presence_of_element_located((By
        .ID, "com.android.calculator2:id/digit_5")))
    five_button.click()

    # Klik na dugme "+"
    add_button = wait.until(EC.presence_of_element_located((By.
        ID, "com.android.calculator2:id/op_add")))
    add_button.click()

    # Klik na dugme "3"
    three_button = wait.until(EC.presence_of_element_located((
        By.ID, "com.android.calculator2:id/digit_3")))
    three_button.click()

    # Klik na dugme "="
    equals_button = wait.until(EC.presence_of_element_located((
        By.ID, "com.android.calculator2:id/eq")))
    equals_button.click()

    # Preuzimanje i provera rezultata
    result = wait.until(EC.presence_of_element_located((By.ID,
        "com.android.calculator2:id/result"))).text

    # Provera da li je rezultat 8
   expected_result = "8"
   assert result == expected_result, f"Test neuspesan.
      Ocekivani rezultat: {expected_result}, ali dobijeni
      rezultat je: {result}"

   print(f"Test uspesan! Rezultat je tacan: {result}!")


finally:
    driver.quit()
```

Listing 1: Kod za testiranje kalkulatora u Appium-u

4.3 Jenkins

Jenkins igra ključnu ulogu u procesu kontinuirane integracije i kontinuirane isporuke (*CI/CD*). Ovaj alat automatizuje procese izgradnje, testiranja i implementacije softverskih aplikacija. Jedna od najvažnijih karakteristika Jenkins-a je mogućnost automatskog pokretanja testova svaki put kada se izvrši promena u kodu, što omogućava brzo otkrivanje grešaka i smanjuje vreme potrebno za ispravke.

Jenkins ima bogat ekosistem dodataka koji omogućavaju integraciju sa alatima za testiranje poput Selenium-a i Appium-a, čime se poboljšava efikasnost celokupnog procesa testiranja i omogućava timovima brzu reakciju na promene u kodu.

Slika 3: Jenkins logo

Jenkins se fokusira na kontinuiranu integraciju i isporuku, omogućavajući automatizaciju celokupnog procesa izgradnje i testiranja. Kombinacijom ovih tehnologija, timovi mogu postići visoke standarde kvaliteta i efikasnosti u razvoju softvera.

5 Tehnološki trendovi

Savremeni svet softverskog razvoja suočava se s brzim promenama koje donose nove tehnologije. Ove promene neizbežno utiču na pristupe testiranju kvaliteta (QA), a među najznačajnijim trendovima su veštačka inteligencija (AI) i mašinsko učenje (ML), kao i testiranje u kontinuitetu.

5.1 Veštačka inteligencija i mašinsko učenje

Veštačka inteligencija i mašinsko učenje postaju ključni alati u QA testiranju, omogućavajući efikasnije i preciznije analize. Korišćenjem AI i ML tehnologija, QA testeri mogu automatski analizirati velike količine podataka i prepoznavati obrasce grešaka, što doprinosi bržem identifikovanju problema i unapređenju procesa testiranja.

Jedan od načina na koji AI može poboljšati QA testiranje jeste automatska generacija test slučajeva. Na osnovu istorijskih podataka o greškama, algoritmi mogu

predložiti nove test scenarije koji pokrivaju potencijalno problematična područja. Ovo ne samo da smanjuje vreme potrebno za pripremu testova, već i povećava šanse za otkrivanje skrivenih grešaka.

Pored toga, AI se može koristiti za analizu korisničkog ponašanja i performansi aplikacije, pružajući QA timovima dragocene uvide o tome gde bi mogli postojati problemi. Ova sposobnost prepoznavanja obrazaca omogućava proaktivno rešavanje potencijalnih problema pre nego što dođu do krajnjih korisnika.

5.2 Testiranje u kontinuitetu

U svetu brzog razvoja softvera, testiranje u kontinuitetu (Continuous Testing) postaje sve važnije. Praksa integracije i kontinuirane isporuke (CI/CD) omogućava timovima da automatizuju test procese tokom celog životnog ciklusa razvoja softvera, čime se poboljšava efikasnost i smanjuje vreme potrebno za isporuku.

Testiranje u kontinuitetu uključuje kontinuirano pokretanje testova kako se kod razvija, omogućavajući ranu detekciju grešaka. Ovaj pristup pruža bržu povratnu informaciju i pomaže programerima da odmah reše probleme, čime se smanjuje rizik od kasnijih grešaka u produkciji. Osim toga, integracija sa CI/CD alatima omogućava automatsko pokretanje testova nakon svake promene u kodu, čime se osigurava da svaka nova funkcionalnost ne ometa postojeće.

Kako se testiranje u kontinuitetu nastavlja razvijati, očekuje se da će doprineti sve većoj automatizaciji, čime se QA timovima oslobađa vreme za fokusiranje na složenije aspekte testiranja i kvalitet softverskih rešenja..

6 Izazovi QA Testera

Kao ključni članovi timova za razvoj softverskih rešenja, QA testeri suočavaju se s brojnim izazovima u savremenom okruženju razvoja softvera. Sa stalnim napretkom tehnologije, rastućim zahtevima tržišta i sve složenijim aplikacijama, izazovi u QA testiranju postaju sve kompleksniji. U nastavku ćemo razmotriti neke od ključnih izazova s kojima se QA testeri suočavaju u svom radu.

6.1 Brzina razvoja softvera i potrebe za bržim testiranjem

Jedan od glavnih izazova za QA testere je ubrzanje procesa razvoja softvera. U eri agilnih metodologija i kontinuirane isporuke (CI/CD), programi se stalno razvijaju, a nove funkcionalnosti se implementiraju vrlo brzo. Ovo postavlja visok pritisak na QA timove da testiranje sprovedu brzo i efikasno, a istovremeno očuvaju visok kvalitet softverskih rešenja. Brzina razvoja često se kosi s potrebom za temeljitim testiranjem, što može dovesti do kompromisa u kvalitetu.

6.2 Kompleksnost aplikacija i sistemskih integracija

Sofisticirane aplikacije danas često zahtevaju integraciju s drugim sistemima, bazama podataka, cloud rešenjima i raznim servisima. Testiranje ovih složenih sistema je veliki izazov jer zahteva razumevanje celokupnog ekosistema, a testiranje integracija između različitih komponenti može biti izuzetno teško. QA testeri moraju osigurati da svi delovi sistema funkcionišu zajedno besprekorno, što može biti izazovno, posebno kada su u pitanju aplikacije sa velikim brojem korisnika ili s velikim brojem međusobnih zavisnosti.

6.3 Automatizacija testiranja

Automatizacija testiranja predstavlja izazov u mnogim timovima, posebno u projektima koji se brzo razvijaju. Automatizovani testovi mogu značajno ubrzati proces testiranja i smanjiti ljudsku grešku, ali njihova implementacija zahteva vreme i resurse. Osim toga, automatizovani testovi moraju biti pravilno dizajnirani kako bi bili efikasni, a održavanje tih testova tokom vremena može postati problematično, naročito kada se aplikacija stalno menja i evoluira.

6.4 Testiranje mobilnih aplikacija i različitih platformi

Testiranje mobilnih aplikacija predstavlja dodatni izazov jer se mora obaviti na različitim uređajima, operativnim sistemima, i ekranima sa različitim rezolucijama. QA testeri moraju obezbediti da aplikacija radi besprekorno na svim podržanim platformama, što uključuje Android i iOS uređaje, kao i različite verzije tih operativnih sistema. Ovaj izazov postaje još veći ako aplikacija mora da funkcioniše sa različitim mrežnim uslovima i s različitim konfiguracijama uređaja.

6.5 Bezbednost i privatnost podataka

Testiranje u kontekstu bezbednosti je jedan od najvažnijih, ali i najizazovnijih aspekata QA testiranja. S obzirom na stalne pretnje od cyber napada, QA testeri moraju biti sigurni da aplikacija ne samo da funkcioniše ispravno, već i da je bezbedna za korišćenje. To podrazumeva testiranje na ranjivosti, zaštitu privatnosti korisničkih podataka i implementaciju sigurnosnih protokola. U svetu sve većih zahteva za zaštitu podataka i usklađenost s regulativama (kao što su GDPR ili HIPAA), QA testeri moraju biti pažljivi na sve aspekte sigurnosti aplikacija.

7 Uticaj QA na korisničko iskustvo

Kvalitet softvera ima direktan uticaj na korisničko iskustvo i zadovoljstvo korisnika. Kako se očekivanja korisnika povećavaju, uloga QA testera postaje još značajnija u obezbeđivanju da proizvodi ne samo da funkcionišu ispravno, već i da pružaju izvanredno korisničko iskustvo.

U prvom redu, visok kvalitet softvera smanjuje broj grešaka koje korisnici mogu iskusiti prilikom korišćenja aplikacije. Kada su greške minimalne, korisnici mogu nesmetano koristiti proizvod, što povećava njihovo zadovoljstvo. S obzirom na to da korisnici često brzo napuštaju aplikacije koje nisu intuitivne ili imaju problema s performansama, QA testeri igraju ključnu ulogu u osiguravanju da su interfejsi korisnički prijateljski i da funkcionalnosti rade onako kako je očekivano.

Osim toga, QA testeri pomažu u razumevanju potreba i očekivanja korisnika kroz testiranje korisničkog iskustva. Učestvujući u procesu razvoja, oni mogu pružiti povratne informacije koje pomažu timu da optimizuje proizvode prema korisničkim zahtevima. Testiranje usmereno na korisnika, kao što je istraživačko testiranje ili testiranje upotrebljivosti, omogućava QA timovima da otkriju područja za poboljšanje koja direktno utiču na zadovoljstvo korisnika.

Na kraju, dobar kvalitet softvera ne samo da poboljšava korisničko iskustvo, već i gradi poverenje između korisnika i brenda. Kada korisnici znaju da mogu da računaju na pouzdane proizvode, verovatno će ostati lojalni i preporučiti ih drugima. U tom smislu, QA testeri ne samo da osiguravaju kvalitet, već i igraju ključnu ulogu u izgradnji reputacije proizvoda i brenda.

8 Zaključak

Uloga QA testera u razvoju softvera postala je ključna za obezbeđivanje kvaliteta i efikasnosti u savremenim procesima razvoja. Evolucija QA testiranja od manuelnih metoda do implementacije automatizacije i veštačke inteligencije značajno je unapredila preciznost, brzinu i pokrivenost testiranja, što je od suštinskog značaja za uspešan razvoj složenih softverskih sistema. Tehnologije kao što su Selenium, Appium i Jenkins omogućavaju efikasno automatizovano testiranje i integraciju u CI/CD procese, dok se trendovi poput veštačke inteligencije i mašinskog učenja sve više primenjuju u analizi podataka i generisanju testova.

Međutim, uprkos napretku, QA testeri se i dalje suočavaju sa izazovima, kao što su brzo promenjivi zahtevi, rastuća složenost aplikacija i pritisak za bržim razvojem. Stoga, uloga QA testera nije samo tehnička, već i strateška, jer testeri moraju da se prilagode novim tehnologijama, razumeju poslovne potrebe i unapređuju korisničko iskustvo.

S obzirom na ove izazove, u budućnosti će se očekivati još veća integracija

automatizacije, AI i ML tehnologija u QA testiranje, kao i povećana potreba za stručnjacima koji mogu kombinovati tehničke veštine sa poslovnim razumevanjem. Uloga QA testera neće biti samo detekcija grešaka, već aktivno učestvovanje u unapređenju celokupnog procesa razvoja softverskih proizvoda.

Literatura

[1] Milena Vujošević Janičić. *Verifikacija softvera.* Matematički fakultet, Univerzitet u Beogradu, 2023.

[2] Saša Malkov. *Agilni razvoj softvera- odlomak iz knjige u pripremi.* Matematički fakultet, Univerzitet u Beogradu, 2023.

[3] Saša Malkov. *Testiranje softvera- odlomak iz knjige u pripremi.* Matematički fakultet, Univerzitet u Beogradu, 2023.

[4] *Selenium* Dostupno na linku: `https://www.selenium.dev/`

[5] *Appium* Dostupno na linku: `https://appium.io/docs/en/2.12/commands/`

[6] *Jenkins* Dostupno na linku: `https://www.jenkins.io/`