Testiranje JavaScript aplikacija

Lucas da Costa

kompjuter biblioteka

MANNING

Testiranje JavaScript aplikacija

LUCAS DA COSTA

MANNING

kompjuter biblioteka

Izdavač:
kompjuter biblioteka

Obalskih radnika 4a, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: Lucas DA COSTA

Prevod: Slavica Prudkov

Lektura: Nemanja Lukić

Slog : Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2021.

Broj knjige: 542

Izdanje: Prvo

ISBN: 978-86-7310-565-9

Testing JavaScript Applications
LUCAS DA COSTA

2021
9781617297915

PREDGOVOR

Testiranje JavaScript aplikacija je knjiga koju bih voleo da sam pročitao pre šest godina. U to vreme bio sam pripravnik za garanciju kvaliteta (QA). Bilo je to moje prvo radno iskustvo u softverskoj industriji. Nažalost, nisam radio ono što mi se najviše dopada: bacanje čini sa tastature. Umesto toga, morao sam ručno da pregledam interfejse, klikćem dugmiće, popunjavam obrasce i proveravam da li softver koji smo napravili radi kako bi trebalo.

„Mora da postoji bolji način“, mislio sam. Tako sam počeo da izrađujem sopstvene čarolije da bi mašine obavljale posao, omogućavajući meni da budem kreativni čarobnjak koji sam želeo da budem.

Posle 18 meseci mislio sam da sam većinu toga shvatio. Do tada sam automatizovao poslove i umesto analitičara kvaliteta postao sam softverski inženjer.

Jednom kad sam počeo da pišem aplikacije, javilo mi se još više pitanja. Budući da sam se dugo bavio QA-om, nisam želeo da se oslanjam na druge programere da izgrade funkcionalan softver. Takođe, nisam želeo da trošim svoje dragoceno vreme klikćući dugmiće i popunjavajući obrasce, kao nekada.

I dalje sam razmišljao: „mora da postoji bolji način“. Tada sam počeo da čitam sve o testiranju softvera. Kad sam dobio pristup izvornom kodu, otkrio sam da mogu da izgradim pouzdaniji softver za manje vremena. Štaviše, mogao sam da oslobodim svoje QA prijatelje da rade kreativnije i proaktivnije, umesto da im prosledjujem softver koji će oni ručno testirati.

Najteže je bilo pronaći materijal iz kog ću naučiti kako se šta radi. Iako sam ponekad mogao da pronađem korisne članke online, većina njih je bila zastarela ili fokusirana na mali deo testne slagalice.

Sastavljanje tih delova bio je najizazovniji deo učenja o testiranju softvera. Da li bi softverski inženjeri trebalo uvek da pišu testove? Ako da, koje vrste testova, za šta i koliko? Kako se razvoj softvera i QA uklapaju?

Pre mnogo godina nije bilo sadržaja koji bi mogao da odgovori na sva ova pitanja. Knjiga koju sam želeo da pročitam nije postojala; stoga sam odlučio da je napišem sam.

Iako je dobar sadržaj raštrkan po celom internetu, veći deo ostaje nenapisan. Veliki deo slagalice za testiranje ostaje nestrukturiran u umovima onih koji održavaju biblioteke za testiranje koje drugi koriste.

U knjizi *Testiranje JavaScript aplikacija* sam te delove uklopio na razumljiv način. Kombinovao sam ono što sam naučio čitanjem, tokom godina i iz praktičnog radnog iskustva, sa onim što sam otkrio održavajući biblioteke za testiranje koje milioni ljudi koriste, kao što su Chai i Sinon.

Čvrsto verujem da je izvrsna praksa testiranja srž svakog uspešnog softvera. Ta praksa pomaže ljudima da napišu bolji softver, za manje vremena, za manje novca. Najvažnije je to što nas ljude spašava muke i daje nam vremena da radimo ono što radimo najbolje: da kreiramo softver koji je, po meni, još uvek prilično nalik magiji.

ZAHVALNICE

Budilnik moje mame uvek je zvonio pre 6:00, baš kao i moj. Da nije toga, ne znam kako bih mogao da napišem 500 stranica koje ćete pročitati, od kojih sam većinu napisao dok je svet još spavao. Mama me je naučila vrednostima discipline i napornog rada i na tome sam joj veoma zahvalan.

Poput nje i mnogi drugi ljudi zaslužuju moju zahvalnost za lekcije koje su me naučili i za pomoć koju su mi pružili.

Među tim ljudima, prvi kojima želim da zahvalim su moja porodica, koja navija za mene sa one strane Atlantika. Mom ocu Herciliju, koji je rekao da će me uvek podržavati u svemu što želim da radim; mojoj sestri, Luizi, najljubaznijoj osobi koju znam; i mojoj mami Patriciji, čiji trud sam pohvalio u prvom pasusu.

Pored njih, moram da zahvalim i baki i deki, koji su brinuli o meni dok su moji roditelji radili, a posebno mojoj baki Marli Teikeira da Costa, kojoj posvećujem ovu knjigu.

Bez obzira na to koliko je bio težak život u Brazilu, ona se uvek trudila da imam sve što mi je potrebno za posao, od knjiga do računara. Tokom nedelje pripremala mi je ručkove, a ponudila mi je sobu u svojoj kući pored univerziteta u kojoj sam mogao da se odmaram da bih mogao da pazim na časovima.

Pored moje porodice, postoji još nekoliko ljudi bez kojih ne bih mogao da dovršim ovo delo: Douglas Melo i Lorenzo Garibotti, koji su me naučili šta prijateljstvo zapravo znači; Ana Zardo, koja mi je pokazala da je svet veći nego što sam mislio; Alek Monk, moj terapeut, koji mi je pomogao da se snađem u promenama i da se izborim sa svojim čestim egzistencijalnim krizama; i Gideon Farrell, koji me je doveo u London i nastavlja da mi veruje i pomaže mi da radim najbolje što mogu.

Takođe ne mogu da zaboravim da zahvalim svima u zajednici JavaScript otvorenog koda za sve što su me naučili tokom godina: Lucas Vieira, kog sam upoznao na fakultetu, jedan je od najtalentovanijih inženjera koje poznajem; Carl-Erik Kopseng, koji me je uveo u Sinon.js i sa kojim sam imao zadovoljstvo da radim 2017. godine; i Keith Cirkel, koji me je pozvao da budem glavni za održavanje Chai.js-a i uvek mi je bio podrška. Pomogao mi je da se snađem kada sam se pre tri godine preselio u Englesku. Drago mi je što me je internet povezao sa tako neverovatnim ljudima!

Svim recenzentima: Sergio Arbeo, Jeremi Chen, Giuseppe De Marco, Lucian Enache, Foster Haines, Giampiero Granatella, Lee Harding, Joanna Kupis, Charles Lam, Bonnie Malec, Brian Miller, Barnabi Norman, Prabhuti Prakash, Dennis Reil, Satej Sahu, Liza Sessler, Raul Siles, Ezra Simeloff, Deniz Vehbi, Richard B. Vard i Rodnei Veis, vaši predlozi su pomogli da ova knjiga postane bolja.

Na kraju, želeo bih da zahvalim svojim urednicima i timu iz Manning-a, a to su Helen Stergius, Dennis Sellinger i Srihar Sridharan, što su pregledali svaku od ovih stranica i strpljivo odgovarali na brojna pitanja koja sam imao tokom procesa.

Obrigado

O OVOJ KNJIZI

Testiranje JavaScript aplikacija vas pomoću praktičnih primera uči da testirate JavaScript kod i objašnjava koje faktore bi trebalo uzeti u obzir prilikom odlučivanja koje testove da pišete i kada.

Pored toga što obuhvata najpopularnije alate za testiranje JavaScript-a i najbolje prakse, u knjizi je objašnjeno kako se različite vrste testova dopunjuju i kako se mogu uklopiti u razvojni proces, tako da za manje vremena kreirate bolji softver sa manje programskih grešaka i sa više samopouzdanja.

Ko bi trebalo da pročita ovu knjigu

Napisao sam knjigu *Testiranje JavaScript aplikacija* uglavnom za mlađe programere i za softverske inženjere koji misle da za izgradnju softvera koji funkcioniše „mora postojati bolji način“, ali još uvek nisu shvatili *koji*.

U ovoj knjizi pretpostavljamo da čitaoci znaju da pišu kod, ali nije potrebno nikakvo prethodno znanje o testiranju softvera.

Osim opisa praktičnih aspekata pisanja testova, objašnjeno je zašto su oni važni i kako utiču na projekte i osnažuju vas da donesete najbolje moguće odluke za *svoj* kontekst.

Kako je organizovana ova knjiga: Vodič

Ova knjiga sadrži 12 poglavlja podeljenih na tri dela.

U prvom delu knjige *Testiranje JavaScript aplikacija* opisani su automatizovani testovi, njihova važnost, različiti tipovi automatizovanih testova i kako koji tip testa utiče na projekat.

- Poglavlje 1 – objašnjeno je šta su automatizovani testovi i prednosti pisanja automatizovanih testova.
- Poglavlje 2 – opisani su različiti tipovi automatizovanih testova i prednosti i nedostaci svakog od njih, tako da znate šta je potrebno uzeti u obzir prilikom odlučivanja koje testove da pišete. Osim toga, opisani su osnovni obrasci koje možete primeniti na sve vrste testova.

Drugi deo vas kroz praktične primere uči da pišete različite tipove testova o kojima ste učili u prvom delu.

- Poglavlje 3 – opisane su tehnike testiranja koje vam pomažu da svoje testove koristite na najbolji način. Naučićete da organizujete više testova u okviru paketa testova tako da dobijate precizne povratne informacije, da pišete temeljne tvrdnje tako da hvatate više grešaka i koje delove koda je potrebno da izolujete tokom testova. Osim toga, objašnjeno je šta je pokrivenost kodom i kako je možete meriti i pokazano je koliko ona ponekad može biti obmanjujuća.

- Poglavlje 4 - naučićete da pišete testove za backend aplikacije. Obuhvaćeni su osnovni aspekti koje bi trebalo uzeti u obzir da bi aplikacija mogla da bude testirana, demonstrirano je kako testirati rute servera i njegove posredničke programe i kako se baviti spoljnim zavisnostima, kao što su baze podataka ili nezavisni API-ji.
- Poglavlje 5 – predstavljene su tehnike koje će vam pomoći da smanjite troškove backend testova i učinite ih bržim i pouzdanijim. Opisano je kako da eliminišete nepredvidive testove, kako istovremeno da pokrenete testove i kako da smanjite preklapanja među njima.
- Poglavlje 6 – opisano je kako da testirate osnovnu JavaScript frontend aplikaciju. U ovom poglavlju je objašnjeno kako možete da simulirate okruženje pretraživača u testnom radnom okviru i prikazano je kako da pišete testove koji komuniciraju sa interfejsom aplikacije, suočavaju se sa API-jem pretraživača i obrađuju HTTP zahteve i WebSocket-e.
- Poglavlje 7 – opisan je React ekosistem za testiranje. Nadovezuje se na ono što ste naučili u prethodnom poglavlju da biste mogli da objasnite kako funkcionišu testovi za React aplikaciju. Osim toga, dali smo vam pregled različitih alata koje možete da koristite za testiranje React aplikacija i opisano je kako da napišete svoje prve React testove. Takođe, dati su saveti kako da primenite slične tehnike na druge JavaScript biblioteke i radne okvire.
- Poglavlje 8 – detaljnije je opisana praktičnost testiranja React aplikacije. U ovom poglavlju ću vam objasniti kako da testirate komponente koje međusobno komuniciraju, kako da testirate stilove komponente i šta je to testiranje snimka stanja i šta treba uzeti u obzir prilikom odlučivanja da li da ih koristite. Štaviše, učićete o važnosti testiranja prihvatanja na nivou komponenti i kako vam ova praksa može pomoći da brže izgradite bolje React aplikacije.
- Poglavlje 9 - govorimo o razvoju vođenom testiranjem (TDD). Objašnjeno je kako da primenjujete ovu tehniku razvoja softvera, zašto je korisno da je usvojite i kada da to učinite. Osim opisa praktičnih aspekata TDD-a, objašnjeno je kako ova tehnika utiče na timove i kako da kreirate okruženje u kom TDD može da bude uspešan. Takođe je opisan odnos TDD-a i prakse nazvane razvoj vođen ponašanjem, što može poboljšati komunikaciju različitih zainteresovanih strana i kvalitet softvera.
- Poglavlje 10 – opisano je šta su end-to-end testovi zasnovani na korisničkom interfejsu i kako oni utiču na poslovanje. Takođe je objašnjeno kako se ovi testovi razlikuju od ostalih tipova testova i kako da odlučite kada da ih pišete.
- Poglavlje 11 – obuhćen je praktičan aspekt end-to-end testova zasnovanih na korisničkom interfejsu. U ovom poglavlju ćete naučiti da pišete svoje prve end-to-end testove zasnovane na korisničkom interfejsu, da ih učinite robusnim i pouzdanim i da ih pokrećete u više pretraživača. Osim toga, opisano je kako da u testove uvrstite testiranje vizuelne regresije i objašnjeno je kako bi ovaj novi tip testova mogao da bude koristan.

Trećim delom obuhvaćene su komplementarne tehnike za pojačavanje pozitivnog uticaja koji testovi mogu imati na vaše poslovanje.

- Poglavlje 12 – opisano je šta su kontinualna integracija i kontinualna isporuka, zašto su to korisne tehnike i objašnjene su osnove koje bi trebalo da znate da biste ih primenili u svojim projektima.

- Poglavlje 13 – opisane su tehnologije, alati i tehnike komplementarne testovima. Govorićemo o tome kako tipovi mogu da vam pomognu u hvatanju programske greške i učine testove efikasnijim, objašnjeno je kako pregledi koda poboljšavaju kvalitet vašeg koda i opisan je uticaj koji dokumentacija i nadzor imaju na izgradnju softvera koji funkcioniše. Pored toga, opisano je kako brže i sigurnije da otklanjate greške u kodu.

Čitaocima preporučujem da prva tri poglavlja pročitaju uzastopno pre nego što počnu da čitaju bilo koje drugo poglavlje. U prvih nekoliko poglavlja opisani su osnovni koncepti testiranja i njihovi međusobni odnosi. Neophodno je da prvo pročitate ova poglavlja, jer će vam informacije u njima biti neophodne da maksimalno iskoristite ostatak knjige.

Zatim čitaoci mogu da pređu direktno na poglavlje koje ih najviše zanima, u zavisnosti od tipa softvera koji žele da testiraju.

U idealnom slučaju, čitaoci bi trebalo da prođu poglavlja 12 i 13 tek kada su već postavili testove i žele da razumeju kako da dopune svoje tehnike testiranja i infrastrukturu.

O kodu

Testiranje JavaScript aplikacija sadrži brojne praktične primere. Svi oni su dostupni online u GitHub skladištu ove knjige, koje čitaoci mogu naći na adresi https://github.com/lucasfcosta/testing-javascript-applications. U tom skladištu sam odvojio primere u posebne direktorijume za svako poglavlje. Unutar svakog direktorijuma grupisao sam primere po odeljcima.

Umetnut kod i zasebne liste koda formatirani su pomoću `fonta fiksne širine sličnog ovom`, tako da se razlikuje od običnog teksta. Ponekad je kod takođe `podebljan` da istakne kod koji je promenjen u odnosu na prethodne korake u poglavlju, na primer kada nova funkcija dodaje kod u postojeću liniju koda.

U mnogim slučajevima originalan izvorni kod je preformatiran; dodali smo prelome redova i preradili uvlake da bismo se prilagodili raspoloživom prostoru na stranici knjige. U retkim slučajevima ni ovo nije bilo dovoljno, pa smo listama dodali markere za nastavak linije (➥). Pored toga, komentari u izvornom kodu često su uklonjeni iz liste kada je kod opisan u tekstu. Oznake koda prate mnoge liste, ističući važne koncepte.

U ovoj knjizi sam označio svaki značajan primer da bih istakao važne koncepte i objasnio čitaocima čemu svaki deo koda služi.

Kod za primere u ovoj knjizi takođe je dostupan za preuzimanje sa veb sajta Manning, na adresi www.manning.com/books/testing-javascript-applications.

Sistemski zahtevi

Svi uzorci koda ove knjige napisani su i testirani na sistemu macOS Catalina. Međutim, trebalo bi da rade na svim platformama, uključujući Linux i Windows.

Jedine promene koje je možda potrebno izvršiti da biste pokrenuli primere prikazane u ovoj knjizi je prilagođavanje načina postavljanja promenljivih okruženja, u zavisnosti od shell-a i operativnog sistema koji koristite. Na primer, ako koristite PowerShell na Windows mašini, ne možete samo da dodate `VAR_NAME =value` komandama da biste postavili vrednost promenljive okruženja.

Da biste pokrenuli primere iz ove knjige, na računar morate da instalirate Node.js i NPM, koji obično dolaze u paketu. Kada instalirate Node.js, *obično* dobijate i NPM. Da biste preuzeli i instalirali ova dva softvera, sledite uputstva na adresi https://nodejs.org/en/download/. Verzije Node.js i NPM-a koje sam koristio prilikom izrade primera za ovu knjigu bile su verzije 12.18 i 6.14.

Forum za diskusiju liveBook

Kupovinom knjige *Testiranje JavaScript aplikacija* dobijate besplatan pristup privatnom veb forumu koji vodi Manning Publications, gde možete da date komentar o knjizi, da postavite tehnička pitanja i da dobijete pomoć od autora i drugih korisnika. Da biste pristupili forumu, otvorite stranicu https://livebook.manning.com/#!/book/testing-javascript-applications/discussion. Takođe možete da saznate više o Manning-ovim forumima i pravilima ponašanja na stranici https://livebook.manning.com/#!/discussion.

Manning je posvećen svojim čitaocima i obezbedio im je mesto na kom se može odvijati smislen dijalog između pojedinih čitalaca i između čitalaca i autora. Autor nije obavezan da učestvuje u diskusiji, već je njegov doprinos forumu dobrovoljan (i neplaćen). Predlažemo da autoru postavite nekoliko izazovnih pitanja da njegovo interesovanje ne bi nestalo! Forum i arhiva prethodnih rasprava biće dostupni na veb stranici izdavača sve dok se knjiga štampa.

O AUTORU

LUCAS DA COSTA je softverski inženjer, objavljivan autor, međunarodni govornik i profesionalni rešavač problema. Kao aktivan član zajednice otvorenog koda održava neke od najpopularnijih JavaScript biblioteka za testiranje, Chai i Sinon. Takođe, doprineo je brojnim drugim projektima, uključujući Jest i NodeSchool.

U poslednjih nekoliko godina, Lucas je govorio na brojnim konferencijama softverskog inženjerstva u više od 10 zemalja.

Njegovi tekstovi su prevedeni na mnoge jezike, uključujući ruski, mandarinski, francuski, portugalski i španski i služe kao referentni materijal na više kurseva softverskog inženjerstva širom sveta.

Lucas voli knjige, lep kod, dobro projektovanu prozu, interfejse komandne linije i Vim. U stvari, toliko voli Vim da mu je na zglobu istetovirano `:w`

O ILUSTRACIJI NASLOVNE STRANICE

Slika na naslovnoj stranici knjige *Testiranje JavaScript aplikacija* nosi naslov „Buržoaski Pariz“. Ilustracija je preuzeta iz knjige kolekcija kostima iz različitih zemalja, autora Jackues Grasseta de Saint-Sauveura (1757–1810), pod naslovom *Costumes civils de actuals de toue les peuples connus,* koja je objavljena u Francuskoj 1788. Svaka ilustracija je fino nacrtana i obojena ručno. Bogata raznolikost kolekcije Grasseta de Saint-Sauveura nas živo podseća na to koliko su svetski gradovi i regioni bili kulturno odvojeni pre samo 200 godina. Izolovani jedni od drugih, ljudi su govorili različitim dijalektima i jezicima. Na ulici ili na selu bilo je lako prepoznati ljude samo po njihovoj odeći, znalo se gde žive i kakva je roba kojom trguju ili status u životu.

Način oblačenja promenio se od tada a raznolikost odeće po regionima, toliko bogata u to vreme, je nestala. Sada je teško razlikovati stanovnike različitih kontinenata, a kamoli različitih gradova, regiona ili zemalja. Možda smo kulturnu raznolikost zamenili za raznovrsniji lični život - svakako za raznovrsniji i brži tempo tehnološkog života.

U vreme kada je teško razlikovati jednu računarsku knjigu od druge, Manning slavi inventivnost i inicijativu računarskog poslovanja koricama knjiga zasnovanim na bogatoj raznolikosti regionalnog života od pre dva veka, koji su oživeli pomoću slika Grasseta de Saint-Sauveura.

I DEO

TESTIRANJE JAVASCRIPT APLIKACIJA

Bilo da dizajnirate veb sajt za pekaru svog ujaka ili platformu za trgovinu akcijama, najvažnija karakteristika softvera je da funkcioniše. Mušterije vašeg ujaka sigurno će naručivati više čizkejkova ako ima intuitivan i lepo dizajniran veb sajt. Takođe, ako je platforma brza i prilagodljiva, brokeri na Wall Street-u zarađivaće više novca. Ipak, korisnicima nije važan trud uložen u performanse i dizajn ukoliko je softver nepouzdan.

Ako program ne funkcioniše, nije važno koliko je lep ili brz.

U prvom delu knjige Testiranje JavaScript aplikacija objašnjeno je kako vam automatizovani testovi pomažu da ljudima pružite ono što žele: softver koji funkcioniše.

Osim toga, naučićete kako taj softver da isporučite brže i sa više samopouzdanja.

U poglavlju 1 ću predstaviti automatizovane testove i opisaću kako oni mogu da pomognu vama i vašem timu.

U poglavlju 2 predstaviću više tipova automatizovanih testova. Objasniću kada da pišete koji tip testa, prednosti i nedostatke svakog tipa i osnovne obrasce koje ćete primenjivati tokom čitanja knjige.

1

UVOD U AUTOMATIZOVANO TESTIRANJE

Ovim poglavljem obuhvaćene su sledeće teme:

- Šta je automatizovan test
- Cilj pisanja automatizovanih testova
- Kako automatizovani testovi mogu da vam pomognu pri pisanju boljeg koda, za manje vremena i sa više samopouzdanja

Kada softver funkcioniše, od pekare vašeg ujaka do ekonomije zemlje, potražnja za novim mogućnostima raste eksponencijalno, a sve je važnija isporuka softvera koji funkcioniše i koji često isporučujemo - nadamo se, više puta dnevno. Za to služe automatizovani testovi. Odavno je prošlo vreme kada su programeri mogli sebi da priušte luksuz povremenog testiranja svog softvera. U ovom trenutku, pisanje testova nije samo dobra praksa, već i industrijski standard. Ako sad pretražujete oglase za posao, skoro u svim se zahteva određeni nivo znanja o automatizovanom testiranju softvera.

Nije važan broj kupaca ili obim podataka sa kojima imate posla. Pisanje efikasnih testova je vredno za kompanije svih veličina, od divova Silikonske Doline podržavanih rizičnim kapitalom do nedavno pokrenutih, malih preduzeća. Testovi su preporučljivi za projekte svih veličina jer olakšavaju komunikaciju među programerima i pomažu u izbegavanju grešaka. Iz ovih razloga, važnost testiranja raste proporcionalno broju programera uključenih u projekat i troškovima potencijalnog neuspeha.

Ova knjiga je namenjena profesionalcima koji već pišu softver, ali još ne mogu da napišu testove ili ne znaju zašto je to neophodno raditi. Dok sam pisao ove stranice, imao sam na umu ljude koji su tek izašli iz centara za obuku ili su nedavno dobili prvi posao programera i žele da se zadrže u branši. Očekujem da čitaoci znaju osnove JavaScript-a i da razumeju koncepte, kao što su promise i povratni pozivi. Ne morate biti stručnjak za JavaScript. Dovoljno je da možete

da napišete funkcionalne programe. U slučaju da se sve uklapa, a vi ste zainteresovani za kreiranje najvrednije vrste softvera - softvera koji funkcioniše - ova knjiga je za vas.

Ova knjiga **nije** namenjena profesionalcima za kontrolu kvaliteta ili ne-tehničkim menadžerima. U knjizi su teme opisane sa stanovišta programera, sa fokusom na to kako možete da koristite povratne informacije testova za brže kreiranje kvalitetnijeg koda. Ja neću govoriti o tome kako da obavljate ručno ili istraživačko testiranje, niti kako da pišete izveštaje o greškama ili kako da upravljate testiranjem radnih procesa. Te zadatke još uvek ne možemo da automatizujemo. Ako želite da saznate više o njima, preporučujem da pročitate knjigu pisanu za analitičare kvaliteta.

Primaran alat koji ćete koristiti u ovoj knjizi je Jest. Pisanjem praktičnih automatizovanih testova za nekoliko malih aplikacija jednostavno ćete naučiti proces. Za aplikacije ćete koristiti običan JavaScript i popularne biblioteke, kao što su Express i React. Korisno je poznavanje Express-a, a posebno React-a, ali čak i ako ih ne poznajete, trebalo bi da bude dovoljno kratko upoznavanje. Sve primere ću kreirati od nule i pretpostavljaću da nemate dovoljno znanja, pa zato preporučujem da istražujete dok radite, umesto unapred.

U ovom poglavlju opisani su koncepti koji prožimaju sve vežbe. Nerazumevanje testova, šta se njima može i šta je potrebno postići, najistaknutiji je uzrok loše napisanih testova, pa time počinjemo ovo poglavlje.

Kad opišemo šta su testovi i cilj pisanja testova, govorićemo o višestrukim slučajevima kada nam pisanje testova može pomoći da kreiramo bolji softver za manje vremena i da olakšamo saradnju između programera. Razumevanje ovih osnovnih koncepata presudno je za početak pisanja prvih testova u poglavlju 2.

1.1 Šta je automatizovan test?

Ujka Luj nije poznat u Njujorku, ali u Londonu je poznat po svojim čizkejkovima od vanile. Zbog svoje izuzetne popularnosti nije mu bilo potrebno mnogo vremena da shvati da vođenje pekare olovkom, na papiru, ne funkcioniše. Da bi održao korak sa sve većim porudžbinama, odlučio je da za izradu svoje internet prodavnice angažuje najboljeg programera kog poznaje: vas.

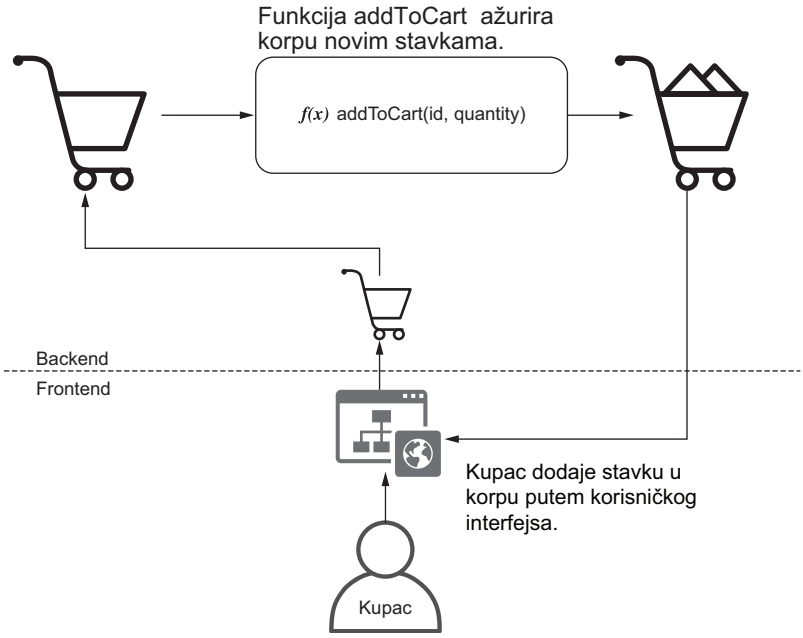
Njegovi zahtevi su jednostavni: kupci moraju da imaju mogućnost da naruče artikle iz pekare, da unesu adresu za isporuku i da izvrše online proveru. Kada implementirate te funkcije, želećete da osigurate da prodavnica radi na odgovarajući način. Kreirajte baze podataka, podesite početne vrednosti, pokrenite server i pristupite veb sajtu na svojoj mašini, da biste pokušali da naručite nekoliko kolača. Tokom ovog procesa, pretpostavimo da ste pronašli grešku. Na primer, otkrili ste da istovremeno možete da imate samo jednu jedinicu stavke u korpi za kupovinu.

Za Luja bi bila katastrofa ako bi veb sajt bio pokrenut sa takvom greškom. Svi znaju da je nemoguće pojesti samo jedan makaron, pa on stoga ne bi prodao nijedan makaron - jedan od Lujevih specijaliteta. Da biste izbegli da se ponovi ta greška, odlučujete da je dodavanje više jedinica stavke slučaj upotrebe koji je uvek potrebno da testirate.

Možete da odlučite da ručno pregledate svako izdanje, kao što su nekada pregledane linije programskog sklopa. Ali to je nedostižan pristup. Predugo traje, a kao u svakom ručnom procesu, lako je napraviti greške. Da biste rešili ovaj problem, morate da zamenite sebe, odnosno kupca, kodom.

Razmislimo o tome kako korisnik kaže vašem programu da doda nešto u korpu. Ova vežba je korisna da bi se utvrdilo koje delove toka akcije je potrebno zameniti automatizovanim testovima.

Korisnici komuniciraju sa aplikacijom putem veb sajta koji šalje HTTP zahtev backend-u. Taj zahtev obaveštava funkciju `addToCart` koju stavku i koliko jedinica žele da dodaju u svoju korpu. Korpa kupca se identifikuje pregledom sesije pošiljaoca. Kada su stavke dodate u korpu, veb sajt se ažurira u skladu sa odgovorom servera. Ovaj proces je prikazan na slici 1.1.



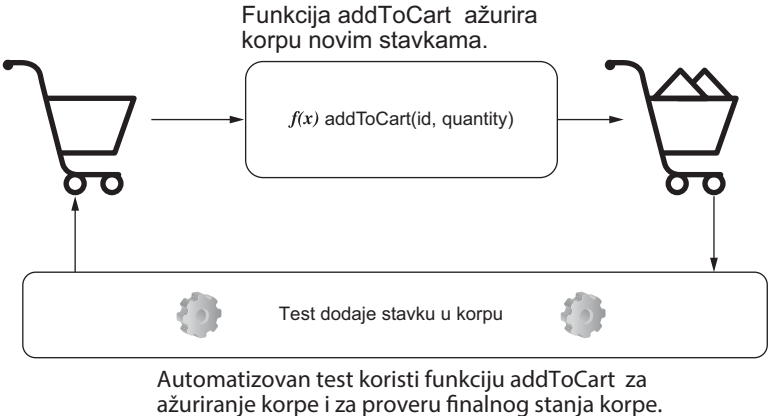
Slika 1.1 Tok akcije porudžbine

> **NAPOMENA** Oznaka `f(x)` je, jednostavno, ikonica koju sam izabrao da predstavlja funkcije na dijagramima u ovoj knjizi. Ne mora obavezno označavati koji su to parametri funkcije.

Zamenimo kupca delom softvera koji može da pozove `addToCartFunction` funkciju. Sada ne zavisite od toga da li će neko ručno dodati stavke u korpu i pogledati odgovor. Umesto toga, imate blok koda koji vrši verifikaciju umesto vas. To je automatizovan test.

AUTOMATIZOVAN TEST Automatizovani testovi su programi koji automatizuju zadatak testiranja softvera. Oni se povezuju sa aplikacijom da bi izvršavali akcije i upoređivali stvarni rezultat sa očekivanim izlazom, koji ste prethodno definisali.

Kod za testiranje kreira korpu i govori funkciji `addToCart` da doda stavke u nju. Kada dobije odgovor, proverava da li su tražene stavke u korpi, kao što je prikazano na slici 1.2.



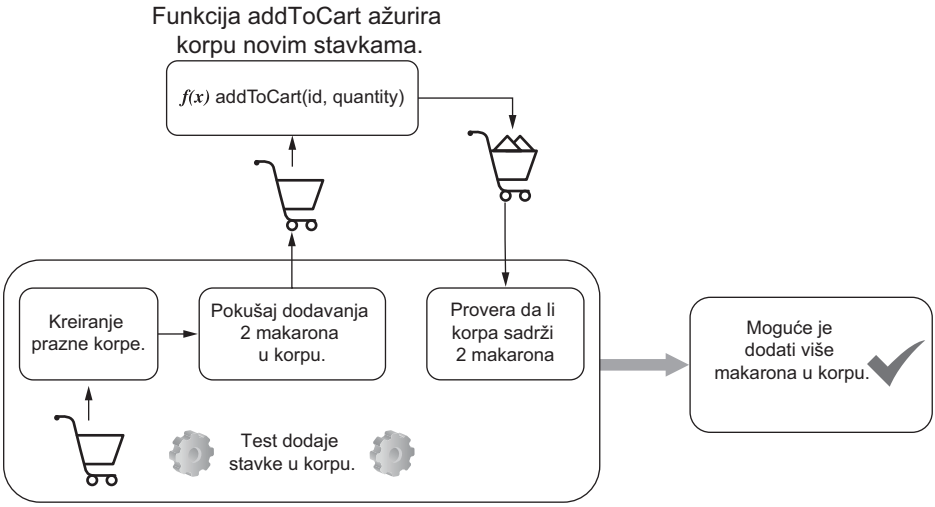
Slika 1.2 Tok akcije za testiranje funkcije addToCart

U okviru testa možete da simulirate tačan scenario po kom korisnici mogu da dodaju samo jedan makaron u svoju korpu:

1. Kreirajte instancu korpe.
2. Pozovite funkciju `addToCart` i recite joj da u tu korpu doda makaron.
3. Proverite da li korpa sadrži dva makarona.

Navođenjem testa da reprodukuje korake koji bi mogli da dovedu do greške možete da dokažete da se ta specifična greška više neće pojavljivati.

Sledeći test koji pišemo garantuje da je moguće dodati više makarona u korpu. Test kreira sopstvenu instancu korpe i koristi funkciju `addToCart` da pokuša da u nju doda dva makarona. Nakon pozivanja funkcije `addToCart`, test proverava sadržaj korpe. Ako se sadržaj korpe podudara sa očekivanjima to vam govori da sve funkcioniše kako bi trebalo. Sada smo sigurni da je u korpu moguće dodati dva makrona, kao što je prikazano na slici 1.3.



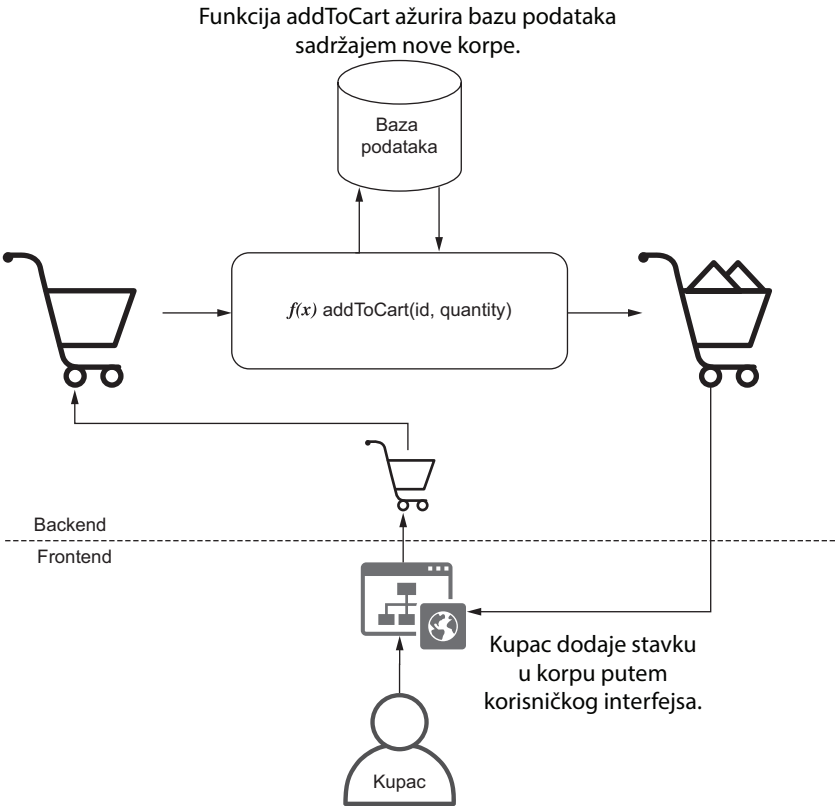
Slika 1.3 Tok akcije za test koji proverava da li se može dodati više makarona u korpu

Sad kada kupci mogu da imaju koliko god žele makarona u korpi – kao što bi i trebalo da bude - recimo da pokušate da simulirate kupovinu koju obavlja kupac: 10.000 makarona. Iznenađujuće je da je porudžbina uspešna, ali ujka Luj nema toliko makarona na lageru. S obzirom na to da je njegova pekara još uvek mala, takođe ne može da izvrši tako velike isporuke u kratkom roku. Da bi bio siguran da može na vreme da dostavi besprekorne poslastice svima, Luj traži od vas da se uverite da kupci mogu da naruče samo onoliko makarona koliko ih ima na lageru.

Da bismo identifikovali koje bi delove toka akcije trebalo zameniti automatizovanim testovima, potrebno je da definišemo šta bi trebalo da se desi kada kupci dodaju stavke u svoje korpe, pa u skladu sa tim da prilagodimo aplikaciju.

Kada kupci kliknu dugme „Add to Cart“ na veb sajtu, kao što je prikazano na slici 1.4, klijent šalje HTTP zahtev serveru za dodavanje 10.000 makarona u korpu. Pre nego što ih doda u korpu, server mora da pregleda bazu podataka da bi proverio da li ima dovoljno robe na zalihama. Ako je količina zaliha manja ili jednaka traženoj količini, makaroni će biti dodati u korpu, a server će poslati odgovor klijentu koji se u skladu s tim ažurira.

NAPOMENA Za testove koristite posebnu bazu podataka za testiranje. Ne zagađujte svoju proizvodnu bazu podataka podacima za testiranje.
Testovi će manipulisati svim vrstama podataka, što može dovesti do gubitka podataka ili do neskladnog stanja baze podataka.
Korišćenje odvojene baze podataka takođe olakšava utvrđivanje osnovnog uzroka greške. Budući da u potpunosti kontrolišete stanje baze podataka za testove, akcije kupaca ne ometaju rezultate testova.



Slika 1.4. Željeni tok akcije za dodavanje samo dostupnih stavki u korpu

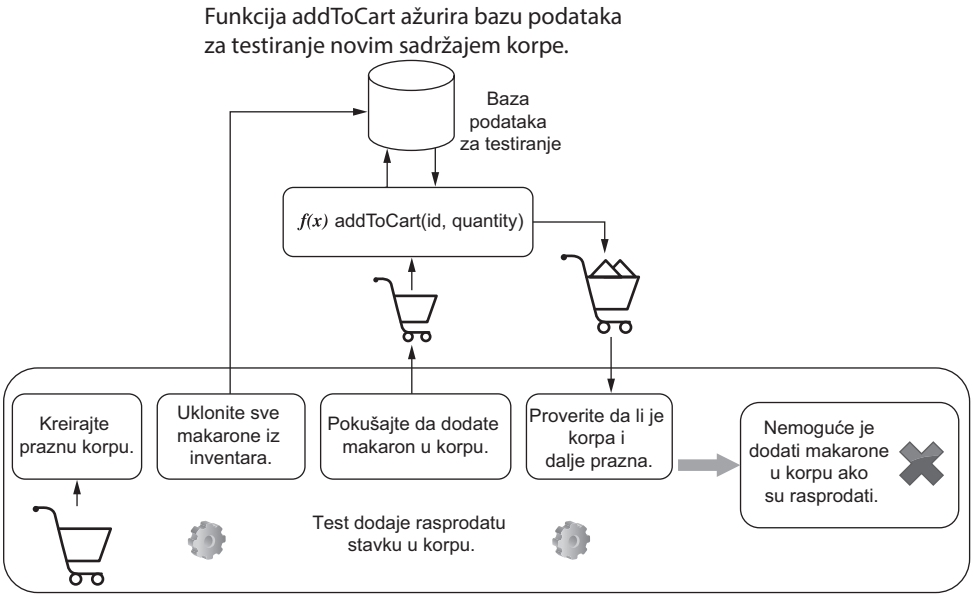
Ova greška je još kritičnija, pa morate biti dvostruko oprezniji. Da biste bili sigurniji u test, možete da ga napišete pre nego što stvarno ispravite grešku, tako da možete da vidite da li je neuspešan kao što bi trebalo.

Jedini koristan tip testa je test koji će biti neuspešan kada aplikacija ne funkcioniše.

Ovaj test je sličan prethodnom: zamenjuje korisnika delom softvera i simulira njegove akcije. Razlika je u tome što je u ovom slučaju potrebno da dodate još jedan korak da biste uklonili sve makarone iz inventara. Test mora da postavi scenario i da simulira akcije koje bi mogle da dovedu do greške; vidite sliku 1.5.

Kada je test postavljen mnogo je lakše ispraviti grešku. Svaki put kad izvršite promenu, test će vam pokazati da li je greška nestala. Ne morate se ručno prijavljivati u bazu podataka, uklanjati sve makarone, otvarati veb sajt i pokušavati da ih dodate u korpu. Test to može uraditi umesto vas, mnogo brže.

S obzirom na to da ste takođe napisali test za proveru da li kupci mogu da dodaju više stavki u korpu, ako zbog vaše ispravke ponovo nastane neka druga greška, test će vas na to upozoriti. Testovi pružaju brze povratne informacije, pa ste sigurni da vaš softver funkcioniše.



Slika 1.5 Neophodni koraci za test da bismo proverili da li možemo dodati rasprodate stavke u korpu

Međutim, moram da vas upozorim da automatizovani testovi nisu rešenje za kreiranje softvera koji funkcioniše. **Testovi ne mogu dokazati da softver funkcioniše; oni samo mogu dokazati da ne funkcioniše**. Čak i ako dodavanje 10.001 makarona u korpu i dalje zanemaruje njihovu dostupnost, to ne znate ukoliko ne testirate taj određeni unos.

Testovi su kao eksperimenti. Očekivanja o načinu na koji softver funkcioniše kodirate u svoje testove, a budući da su u prošlosti bili uspešni, odlučujete da verujete da će se aplikacija ponašati na isti način u budućnosti, iako to nije uvek tačno. Što više testova imate i što više podsećaju na ono što pravi korisnici rade, to vam daju više garancija.

Automatizovani testovi takođe ne podrazumevaju zanemarivanje ručnog testiranja. I dalje su neophodne provere vašeg rada kao krajnjeg korisnika i ulaganje vremena u istraživačka ispitivanja. Budući da je ova knjiga namenjena programerima softvera a ne QA analitičarima, u kontekstu ovog poglavlja pozvaću se na nepotreban postupak ručnog testiranja koji se često radi tokom razvoja, kao i na ručno testiranje.

1.2 Zašto su automatizovani testovi važni

Testovi su važni jer vam daju brze povratne informacije o greškama. U ovom poglavlju detaljno ćemo razmotriti kako brze i precizne povratne informacije poboljšavaju proces razvoja softvera, čineći razvojni tok ujednačenijim i predvidljivijim, olakšavajući reprodukciju problema i dokumentovanje testova, olakšavajući saradnju između programera ili timova i skraćivanjem vremena potrebnog za isporuku visokokvalitetnog softvera.

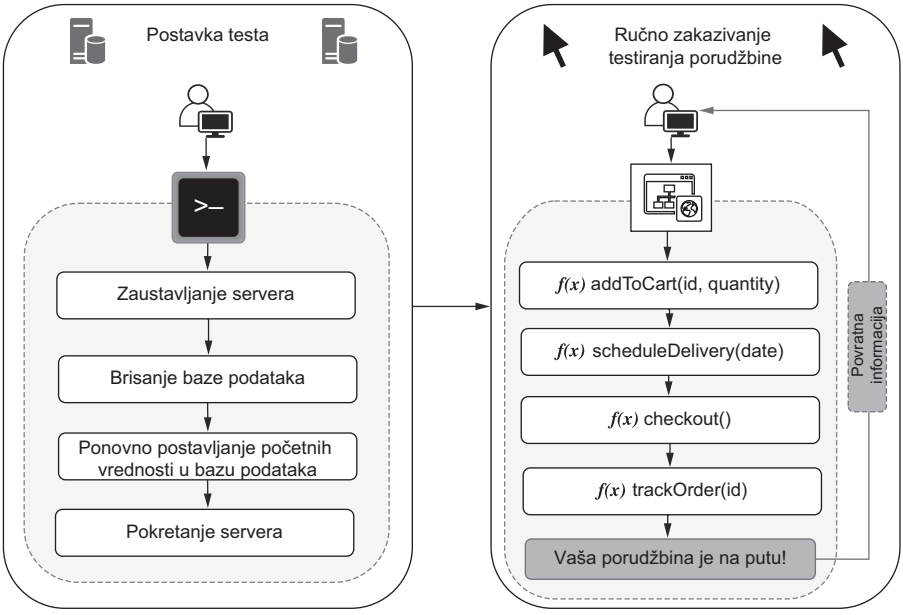
1.2.1 Predvidljivost

Predvidljiv proces razvoja znači sprečavanje neočekivanog ponašanja tokom implementacije funkcije ili ispravljanja greške. Smanjivanje broja iznenađenja tokom razvoja takođe olakšava procenu zadataka i uzrokuje da programeri ređe pregledaju svoj rad.

Ručna provera da li celokupan softver radi onako kako očekujete je dugotrajan proces sklon greškama. Testovi poboljšavaju ovaj proces jer smanjuju vreme potrebno za dobijanje povratne informacije o kodu koji pišete i zbog toga ubrzavaju ispravljanje grešaka. Što je manja udaljenost između čina pisanja koda i primanja povratnih informacija, razvoj je predvidljiviji.

Da bismo ilustrovali kako testovi mogu razvoj učiniti predvidljivijim, zamislimo da je Luj tražio od vas novu funkciju. On želi da kupci mogu da prate status svojih porudžbina. Ova funkcija bi mu pomogla da provodi više vremena praveći kolače a manje vremena javljajući se na telefon da bi uverio kupce da će njihova porudžbina biti isporučena na vreme. Luj je strastven kad su u pitanju čizkejkovi, a ne telefonski pozivi.

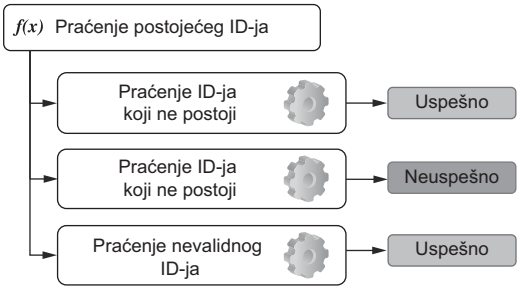
Ako funkciju praćenja implementirate bez automatizovanih testova, morate ručno da prolazite kroz ceo postupak kupovine da biste videli da li funkcioniše, kao što je prikazano na slici 1.6. Svaki put kad je potrebno da ga ponovo testirate, pored ponovnog pokretanja servera, potrebno je i da obrišete baze podataka da biste se uverili da su u doslednom stanju, otvorite pretraživač, dodate stavke u korpu, zakažete isporuku, prođete kroz naplatu i tek tada biste konačno mogli da testirate praćenje porudžbine.



Slika 1.6 Koraci za testiranje praćenja porudžbine

Pre nego što ručno testirate ovu funkciju, ona mora biti dostupna na veb sajtu. Potrebno je da napišete njen interfejs i dobar deo backend-a sa kojim klijent komunicira.

Ako nemate automatizovane testove, potrebno je da napišete previše koda pre nego što proverite da li funkcija radi. Ako morate da prolazite kroz dug i dosadan proces svaki put kada unosite promene, odjednom ćete pisati veće delove koda. S obzirom na to da je potrebno toliko vremena da se dobije povratna informacija kada napišete veće delove koda, do trenutka kada je primite moglo bi da bude prekasno. Napisali ste previše koda pre testiranja, pa sada ima više mesta za skrivanje grešaka. Gde je, među hiljadu novih linija koda, greška koju ste upravo videli?



Slika 1.7 Testovi za funkciju trackOrder mogu pozvati tu funkciju direktno, pa nema potrebe da dodirujete ostale delove aplikacije.

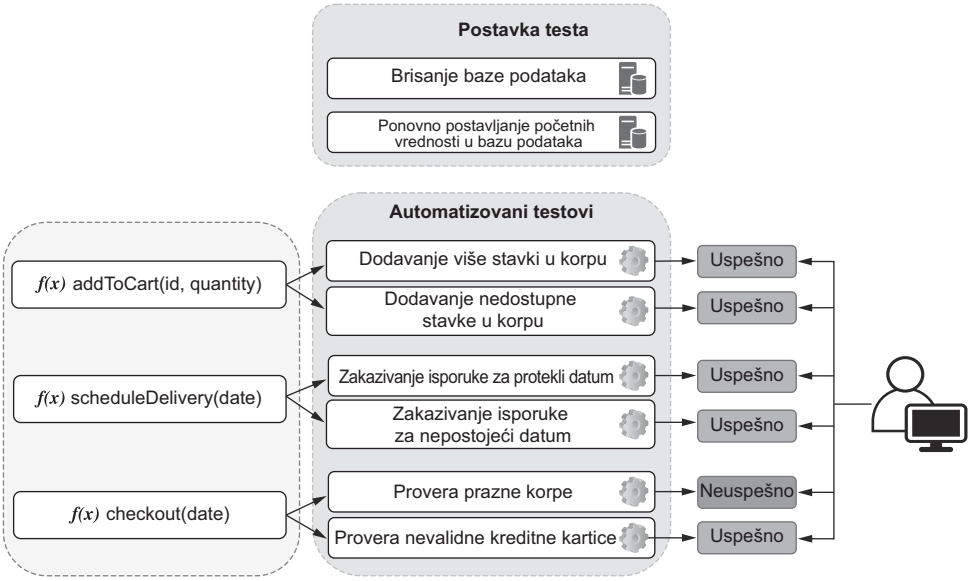
Pomoću automatizovanog testa, poput ovog prikazanog na slici 1.7, možete da napišete manje koda pre nego što dobijete povratne informacije. Kada automatizovani testovi mogu direktno da pozovu funkciju `trackOrder`, možete da izbegnete dodirivanje nepotrebnih delova aplikacije pre nego što budete sigurni da funkcija `trackOrder` radi.

Kada je test neuspešan nakon što napišete samo 10 linija koda, imate samo 10 linija koda o kojima treba da brinete. Čak i ako greška nije unutar tih 10 linija, postaje mnogo lakše otkriti koja je od njih negde drugde izazvala loše ponašanje.

Situacija se može pogoršati ako prekinete druge delove aplikacije. Ako izazovete greške u proceduri naplate, morate da proverite kako su promene uticale na to. Što ste više promena izvršili, to je teže pronaći problem.

Kada imate automatizovane testove, poput ovih na slici 1.8, oni vas mogu upozoriti čim se nešto prekine da biste lakše mogli ispraviti grešku. Ako često pokrećete testove, dobićete precizne povratne informacije o tome koji deo aplikacije je prekinut, čim je prekinete. Zapamtite da **što je manje vremena potrebno za dobijanje povratne informacije nakon što napišete kod, to će vaš razvojni proces biti predvidljiviji.**

Često vidim da programeri moraju da odbace posao jer su odjednom izvršili previše promena. Kad te promene uzrokuju prekid u više delova aplikacije, programeri ne znaju odakle da počnu rešavanje problema. Lakše je da počnu ispočetka, nego da poprave nered koji su već kreirali. Koliko puta ste to uradili?



Slika 1.8 Automatizovani testovi mogu da provere delove koda pojedinačno i pruže vam precizne povratne informacije o prekidu čim se on desi.

1.2.2 Ponovljivost

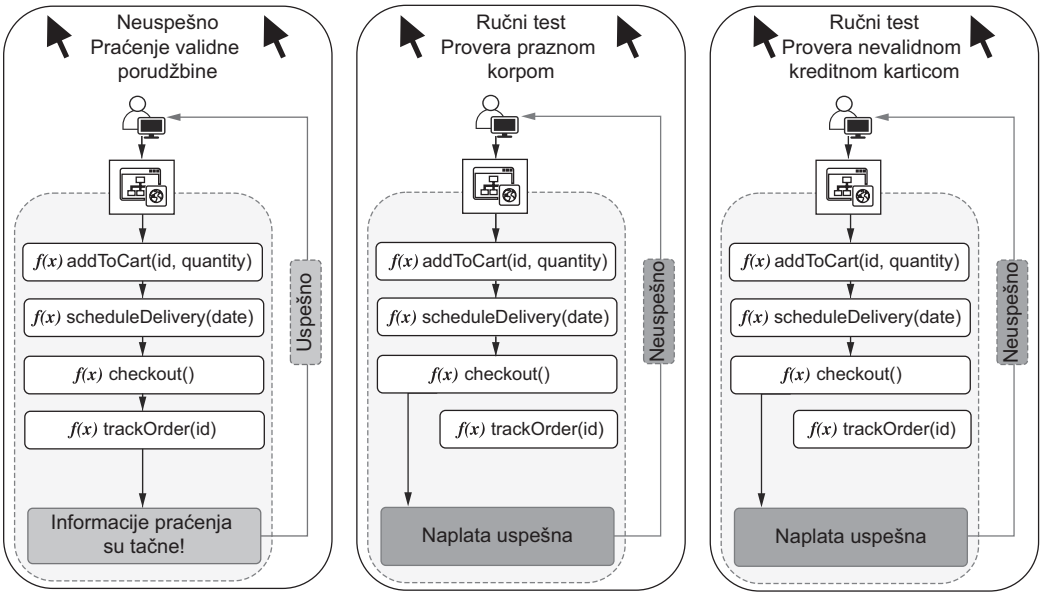
Što više koraka ima određeni zadatak, veća je verovatnoća da će čovek pogrešiti sledeći ih. Automatizovani testovi olakšavaju i ubrzavaju reprodukovanje programske greške i osiguravaju da greške više nisu prisutne.

Da bi kupac mogao da prati status porudžbine, mora da prođe više koraka. Mora da doda stavke u svoju korpu, izabere datum isporuke i da prođe postupak plaćanja. Da biste testirali aplikaciju i osigurali da će ona funkcionisati za kupce, morate slediti te korake. Ovaj proces je relativno dug i podložan greškama, a svakom koraku se može pristupiti na mnogo različitih načina. Pomoću automatizovanih testova možemo da osiguramo da se ovi koraci tačno slede.

Pretpostavimo da pronađete greške tokom testiranja aplikacije, poput mogućnosti odjavljivanja sa praznom korpom ili nevažećom kreditnom karticom. Da biste pronašli te greške, morate ručno da prolazite niz koraka.

Da biste izbegli da se te greške ponove, morate reprodukovati potpuno iste korake koji uzrokuju svaku od njih. Ako lista test slučajeva raste i preduga je, ili ako postoji previše koraka, prostor za ljudske greške postaje veći. Ako nemate kontrolnu listu koju svaki put strogo pratite, greške će se pojaviti (videti sliku 1.9).

Naručivanje torte je nešto što ćete sigurno da proverite, ali šta je sa naručivanjem -1 torte, ili čak `NaN` torti? Ljudi zaboravljaju i prave greške, pa samim tim dolazi i do prekida softvera. Ljudi bi trebalo da rade ono u čemu su dobri, a izvršavanje ponavljajućih zadataka nije na tom spisku.



Slika 1.9 Koraci koje morate pratiti prilikom testiranja svake funkcije

Čak i ako odlučite da održavate kontrolnu listu za te testne slučajeve, imaćete velike troškove za redovno ažuriranje te dokumentacije. Ako zaboravite da je ažurirate i dogodi se nešto što nije opisano u test slučaju, ko greši - aplikacija ili dokumentacija?

Automatizovani testovi izvršavaju potpuno iste akcije svaki put kada ih izvršite. Kada mašina izvršava testove, ona ne zaboravlja nijedan korak, niti pravi greške.

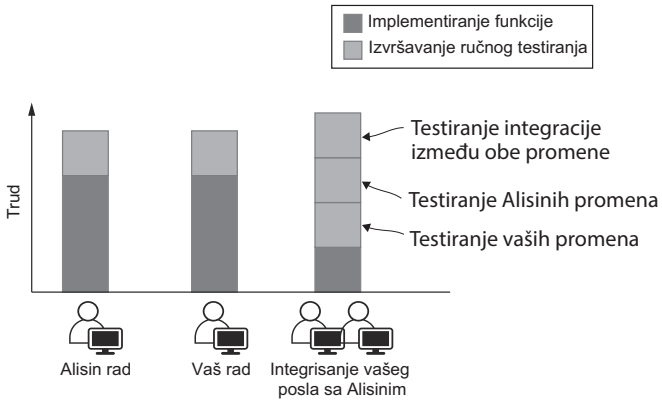
1.2.3 Saradnja

Svako ko proba Lujeve banofija pite zna da je on jedan korak do toga da postane zvezda Great British Bake Off-a (britanska tv emisija). Ako sve radite kako treba sa softverske strane, možda će Luj jednog dana otvoriti pekare svuda, od San Franciska do Sankt Petersburga. U tom scenariju, jedan programer jednostavno nije dovoljan.

Ako angažujete druge programere da rade sa vama, odjednom počinjete da imate nove i drugačije probleme. Ako implementirate novi sistem za popust, a Alis implementira način za generisanje kupona, šta ćete učiniti ako vaše izmene u proceduri plaćanja onemoguće klijentima da primene kupone na svoje porudžbine? Drugim rečima, kako možete osigurati da vaš rad neće ometati njen rad i obrnuto?

Ako Alis prvo doda svoju funkciju u osnovu koda, morate je pitati kako je potrebno da testirate njen rad da biste bili sigurni da ga vaš rad nije prekinuo. Dodavanje vašeg rada oduzeće vreme i vama i Alis.

Trud koji ste vi i Alis uložili u ručno testiranje vaših promena moraćete da ponovite kada integrišete svoj rad sa njenim. Osim toga, potreban je dodatni napor da se testira integracija između obe promene, kao što je prikazano na slici 1.10.



Slika 1.10 Napor potreban za verifikaciju promena u svakoj fazi razvojnog procesa prilikom ručnog testiranja

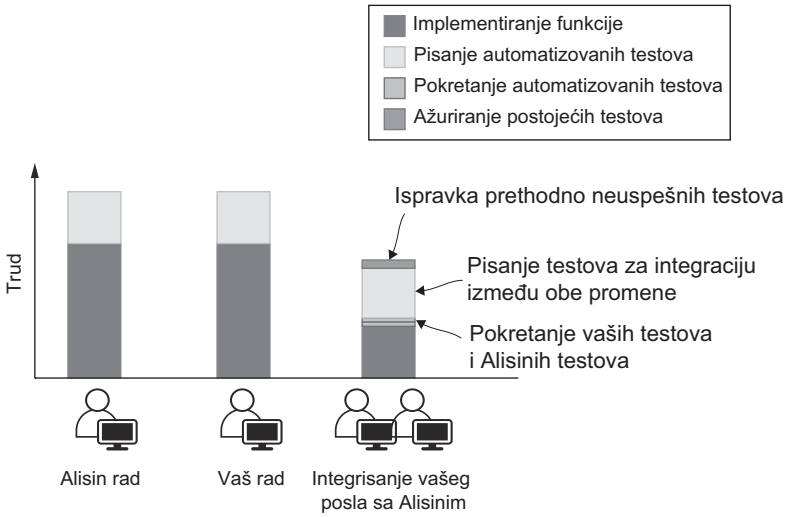
Osim što je dugotrajan, ovaj proces je sklon greškama. Morate da zapamtite sve korake i granične slučajeve da biste ih testirali i u svom i u Alisinom radu. I, čak i ako ih zapamtite, i dalje morate tačno da ih sledite.

Kada programer doda automatizovane testove za funkcije, svi ostali u timu imaju koristi. Ako Alisin rad ima testove, ne morate da je pitate kako da testirate njene promene. Kada dođe vreme da spojite oba dela, možete jednostavno da pokrenete postojeće automatizovane testove, umesto da ponovo prolazite kroz ceo postupak ručnog testiranja.

Čak i ako se vaše promene nadovezuju na njene, testovi će poslužiti kao ažurna dokumentacija koja će voditi dalji rad. Dobro napisani testovi su najbolja dokumentacija koju programer može da ima. S obzirom na to da treba da se izvrše, testovi će uvek biti ažurni. Ako svakako pišete tehničku dokumentaciju, zašto ne biste umesto toga napisali test?

Ako se vaš kod integriše sa Alisinim, dodaćete više automatizovanih testova koji obuhvataju integraciju između vašeg posla i njenog. Ove nove testove će koristiti drugi programeri prilikom integracije povezanih funkcija i zbog toga ćete im uštedeti vreme. Pisanje testova kad god izvršite promene stvara odličan ciklus saradnje, gde jedan programer pomaže onima koji će u budućnosti koristiti taj deo osnove koda (pogledajte sliku 1.11).

Ovaj pristup smanjuje dodatni trošak komunikacije, ali ne eliminiše potrebu za komunikacijom, koja je kamen temeljac uspešnosti svakog projekta. Automatizovani testovi izuzetno poboljšavaju proces saradnje, ali postaju još efikasniji u kombinaciji sa drugim praksama, kao što su pregledi koda.



Slika 1.11 Napor potreban za verifikaciju promena u svakoj fazi razvojnog procesa kada postoje automatizovani testovi

Jedan od najizazovnijih zadataka u softverskom inženjerstvu je naterati više programera da efikasno sarađuju, a testovi su jedan od najkorisnijih alata za to.

1.2.4 Brzina

Luja nije briga koji jezik koristite, a još manje koliko testova pišete. Luj želi da prodaje peciva, kolače i druga slatka čudesa koja može da proizvede. Luju je stalo do prihoda. Ako više funkcija čini kupce sretnijim i donosi više prihoda, on želi da te funkcije isporučite što je brže moguće. Postoji samo jedan zahtev: one moraju da funkcionišu.

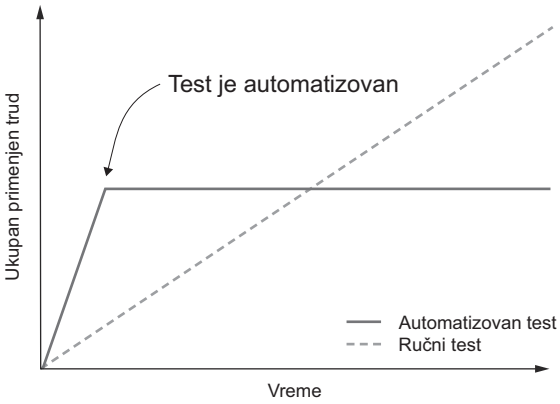
Za posao su bitni brzina i ispravnost, a ne testovi. U svim prethodnim odeljcima govorili smo o tome kako su testovi poboljšali razvojni proces čineći ga predvidljivijim, ponovljivijim i kolaborativnijim, ali na kraju, to su prednosti samo zato što nam pomažu u kreiranju boljeg softvera za manje vremena.

Kada vam je potrebno manje vremena za kreiranje koda, dokazivanje da nema određene greške i integrisanje tog koda sa radom svih ostalih programera je uspešan posao. Kada sprečite regresije, posao je uspeo. Kada raspoređivanje učinite sigurnijim, posao je uspeo.

S obzirom na to da je potrebno vreme da napišete testove, oni imaju trošak. Ali mi insistiramo na pisanju testova, jer su prednosti znatno veće od nedostataka.

U početku pisanje testa takođe može oduzeti vreme, čak i više od ručnog testiranja, ali što ga više pokrećete, iz njega izvlačite više vrednosti. Ako vam je potreban jedan minut da izvršite ručno testiranje, a vi potrošite pet minuta na pisanje automatizovanog testa, isplatiće se čim ga pokrenete peti put – a verujte mi, taj test ćete pokretati više od pet puta.

Za razliku od ručnog testiranja, čije će izvršenje trajati isto ili duže od prethodnog puta, automatizacija testa uzrokuje da vreme i napor potrebni za njegovo pokretanje padnu gotovo na nulu. Kako vreme prolazi, ukupan napor uključen u ručno testiranje raste mnogo brže. Ova razlika u naporu između pisanja automatizovanih testova i izvršavanja ručnog testiranja je ilustrovana na slici 1.12.



Slika 1.12 Trud primenjen vremenom kada se izvršava ručno testiranje nasuprot automatizovanog testiranja

Pisanje testova je poput kupovine akcija. Možete da platite veliku cenu unapred, ali ćete još dugo ubirati dividende. Kao i u finansijama, vrsta investicije koju ćete izvršiti - i da li ćete je izvršiti - zavisi od toga kada je potrebno da povratite novac. Dugoročni projekti su oni koji imaju najviše koristi od testova. Što se projekat duže pokreće, više napora se štedi i više možete uložiti u nove funkcije ili druge značajne aktivnosti. Kratkoročni projekti, kao kada se skupi grupa programera da izvrše neki posao na brzinu, nemaju mnogo koristi od testova. Oni ne "žive" dovoljno dugo da bi opravdali trud koji ćete uštedeti testiranjem.

Poslednji put kada vas je Luj pitao da li možete brže da isporučujete funkcije ako ne pišete toliko testova, ipak niste koristili finansijsku analogiju. Rekli ste mu da bi to bilo poput povećanja temperature rerne da bi kolač bio pre gotov. Ivice se prepeku, ali sredina je još uvek sirova.

Rezime

- Automatizovani testovi su programi koji automatizuju zadatak testiranja vašeg softvera. Ti testovi komuniciraju sa aplikacijom i upoređuju njen stvarni izlaz sa očekivanim izlazom. Testovi su uspešni kada je izlaz ispravan i pruža vam značajne povratne informacije kada je neispravan.
- Testovi koji nikad nisu neuspešni su beskorisni. Cilj testova je da budu neuspešni kada aplikacija više ne funkcioniše na određeni način.
- Ne možete dokazati da softver funkcioniše. Možete dokazati samo da ne funkcioniše. Testovi pokazuju da određene greške više nisu prisutne - a ne da nema grešaka. Skoro neograničen broj mogućih unosa može se dati aplikaciji i nije izvodljivo testirati sve unose. Testovi obično obuhvataju greške koje ste ranije videli ili određene situacije za koje želite da budete sigurni da će funkcionisati.

- Automatizovani testovi smanjuju vreme između čina pisanja koda i dobijanja povratnih informacija. Stoga čine vaš razvojni proces strukturiranijim i smanjuju broj iznenađenja. Predvidljiv razvojni proces olakšava procenu zadataka i omogućava programerima da ređe pregledaju svoj rad.
- Automatizovani testovi uvek slede potpuno istu seriju koraka. Ne zaboravljaju, niti greše. Oni osiguravaju da se test slučajevi detaljno prate i olakšavaju reprodukciju grešaka.
- Kada su testovi automatizovani, troškovi prerade i komunikacije se smanjuju. Programeri mogu sami odmah da verifikuju rad drugih ljudi i uvere se da nisu prekinuti drugi delovi aplikacije.
- Dobro napisani testovi su najbolja dokumentacija koju programer može da ima. S obzirom na to da testovi moraju da prođu uspešno, oni uvek moraju da budu ažurni. Testovi prikazuju upotrebu API-ja i pomažu drugim programerima da razumeju kako funkcioniše baza kodova.
- Preduzeća nije briga za vaše testove. Preduzeća brinu o ostvarivanju dobiti. Na kraju, automatizovani testovi su korisni jer donose profit pomažući programerima da brže isporuče kvalitetniji softver.
- Pri pisanju testova plaćate veliku cenu unapred, ulaganjem dodatnog vremena u njihovo kreiranje. Međutim, vraćate vrednost u vidu dividende. Što se test češće pokreće, to vam je uštedeo više vremena. Stoga, što je životni ciklus projekta duži, to su testovi važniji.