Sadržaj

Uvod **1**

1. Planiranje testa **3**
Razumevanje zahteva 4
Test slučajevi 7
Test vektori 10
Test scenariji 11
Bagovi 12

2. Tehnike testiranja crne kutije **13**
Crna kutija nije crna rupa 15
Testiranje unosa podataka 17
Tekstualne vrednosti 17
Brojevi 26
Datumi i vreme 29
Testiranje obrade podataka 53
Zaokruživanje brojeva 53
Izrazi i formule 56
Obrada tekstualnih podataka 61
Obrada datuma 63
Provera poslovnih pravila 64
Testiranje izveštaja 65
Testiranje rezultata pretraga 66
Testiranje grafika i proračuna 70
Klase ekvivalencije 73
Granične vrednosti 77
Granične vrednosti klasa ekvivalencije 78

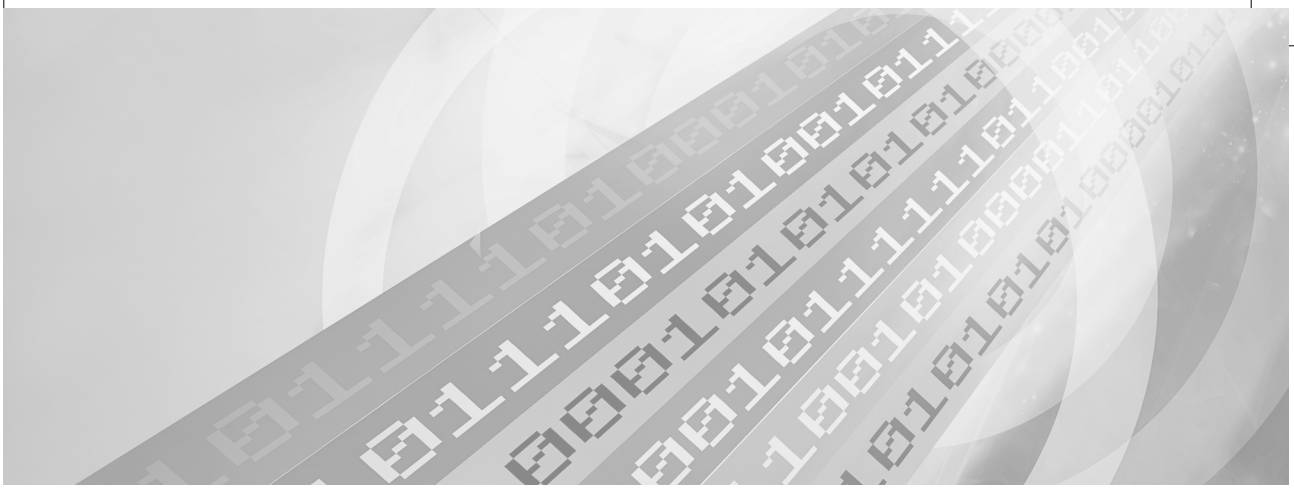
3. Kombinatorno testiranje **81**
Kombinovanje dimenzija 82
Testiranje svake vrednosti 84
Testiranje svih mogućih vrednosti 85
Testiranje svih parova 88
Testiranje zavisnosti među dimenzijama 91
Testiranje logičkih izraza 92
Složeni logički izrazi 94

4. Tehnike bele kutije 99
Testiranje algoritama i procesa 100
Testiranje svih akcija 103
Testiranje svih ivica 104
Testiranje svih putanja 105
Testiranje ponavljanja 106
Testiranje stanja i prelaza 107
Testiranje stanja prikaza korisničkog interfejsa 110
Testiranje toka korisničkog interfejsa 116
Testiranje relacija 119

5. Testiranje sigurnosti 123
Može li napadač da se prijavi u sistem? 123
Unos malicioznog teksta 125
Uloge i prava pristupa 128
Provera prava pristupa 130
Pristup veb-servisima 133

6. Alati 135
Organizujte svoje zadatke uz Trelo 135
GitHub 137
Atlassian JIRA 141
Azure DevOps 142
Alati za pristup bazama podataka 146

Zaključak 149



Uvod

Ovo je knjiga u kojoj ćete naučiti kako je potrebno testirati softverske aplikacije. Kada se nađete pred nekom aplikacijom ili njenom dokumentacijom, postoje dva načina da je testirate:

1. Možete koristiti logiku ili intuiciju kako biste zaključili šta je potrebno uneti u aplikaciju. Često se ovo svodi i na unošenje slučajno odabranih vrednosti. Ovo nije dobar način u velikom broju slučajeva.
2. Ispravan način – možete da uočite neke karakteristike aplikacije i primenite standardne „recepte", koji vam govore kako se te funkcionalnosti testiraju.

Postoje neka pravila u procesu testiranja aplikacija pomoću kojih, na osnovu nekih tipičnih elemenata i funkcionalnosti, možete da identifikujete šta je potrebno da uradite tokom testa.

Postoje dve uloge u procesu testiranja za koje će biti veoma korisno da pročitaju ovu knjigu:

- **Testeri** koji ručno testiraju aplikacije. Testeri moraju da utvrde šta je potrebno probati kako bi se testirala aplikacija, analizirajući dokumentaciju koja opisuje aplikaciju ili korisnički interfejs aplikacije.
- **Test inženjeri** koji pišu automatske testove. Test inženjeri pišu programski kod kojim se testira aplikacija. U tom programskom kodu moraju da budu test podaci koji efikasno testiraju aplikaciju.

Ako radite kao tester ili želite da to postanete pošto se ranije niste bavili softverom, a smatrate da bi vam ovakva pozicija omogućila da uđete u softversku industriju,

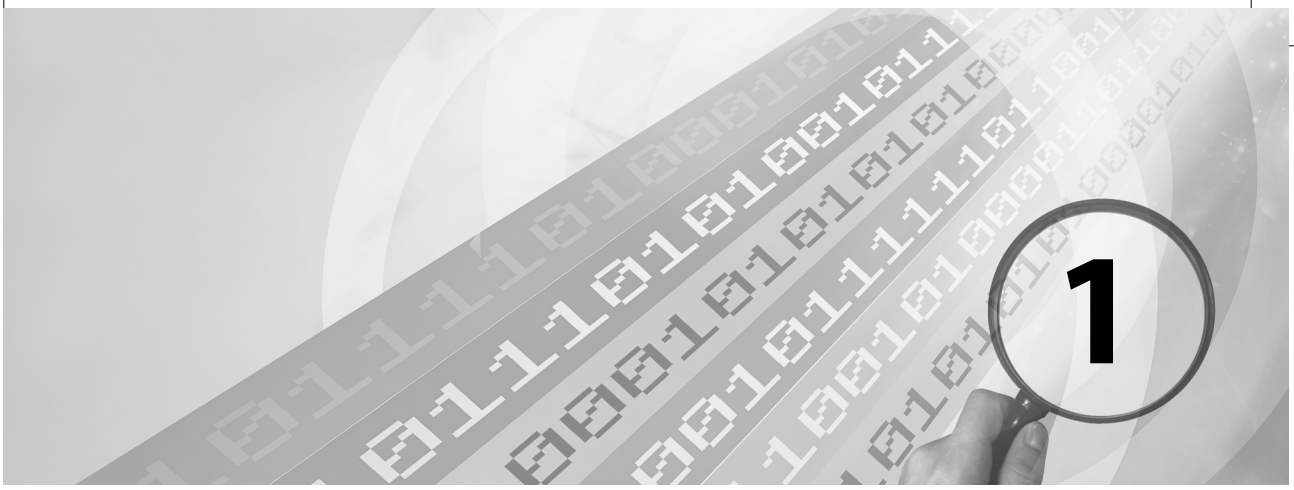
jedan od najčešćih zadataka će vam biti je testiranje aplikacija preko korisničkih interfejsa. Ova knjiga će vam pomoći da shvatite šta je potrebno testirati i koje podatke treba unositi dok testirate aplikacije.

Ako radite kao test inženjer i pišete testove u nekim alatima i programskim jezicima, najveći problem na koji ćete naići je koje podatke treba koristiti u testovima koje ste napisali. Problem je sličan kao i u slučaju testera – jedina razlika je u tome što vi ove test podatke unosite u programskom kodu, a oni kroz korisnički interfejs.

Postoji jednostavan odgovor na ovo pitanje – unećemo neke slučajne podatke. Često je funkcija `rand()` najbolji prijatelj velikog broja test inženjera pošto njome često mogu da „definišu" test podatke. Nažalost, ovo nije najbolja odluka. Unošenjem slučajno odabranih vrednosti ćete slučajno nalaziti probleme.

Ova knjiga će vam omogućiti da uradite nešto sasvim drugačije i, u velikom broju slučajeva, bolje. Pomoći će vam da identifikujete koje vrednosti treba uneti u aplikaciju prilikom testiranja kako biste bili sigurni da testirate na pravi način. Ovu knjigu zamislite kao skup recepata, načina na koje treba razmišljati i trikova koje treba koristiti kada testirate aplikaciju.

Bez obzira na koji način testirate aplikaciju, ideje koje ćete naći u ovoj knjizi će vam omogućiti da bolje, efikasnije i profesionalnije uradite svoj posao.



Planiranje testa

Prva stvar koju treba uraditi pre testiranja (a i svake druge radnje) je isplanirati šta ćete da uradite. Bez plana, samo ćete raditi slučajne radnje vođene intuicijom ili iskustvom, uz veliku verovatnoću da radite pogrešne stvari, dok vam neopaženo promiču prave stvari.

Koja je razlika između šahiste ili fudbalera koji igraju sa planom igre i bez njega? Igra bez plana se svodi na povlačenje poteza ili dodavanje lopte, trčanje i šutiranje, uz nadu da će protivnik da pogreši. Iskusni šahisti i dobri fudbaleri će čak i bez nekog plana verovatno praviti bolje poteze od neiskusnih. Međutim, koliko god da su iskusni, to se ne može meriti sa šahistima ili fudbalerima koji počinju da igraju sa planom i pripremljenom taktikom. Plan pravi razliku između onih koji su pripremljeni za uspeh i ostalih, koji uspevaju zahvaljujući sreći ili iskustvu koje su stekli tako što su iz dovoljnog broja neuspeha shvatili šta verovatno ne treba da rade.

Nažalost, često vam se može desiti da budete u situaciji da vam neko iznenada zatraži da testirate nešto bez plana. U ovom slučaju će vas zamoliti da „malo testirate aplikaciju" bez neke pripreme. Ovo ne mora da bude ništa zlonamerno. Rokovi su uvek kratki, uvek se kasni, preostalo je još malo vremena do roka za isporuku, i često je potrebno da neko uskoči na brzinu i „malo isproba aplikaciju" pre nego što se isporuči.

Ako ikada dođete u situaciju da vam neko traži da ovako testirate aplikaciju, morate da shvatite šta su očekivanja. Ovo je testiranje gde se neko nada da ćete slučajnim akcijama i unosima naći neke probleme. Međutim, ne možete da garantujete da aplikacija radi pošto ne znate šta sve niste probali; često ne možete da

reprodukujete problem pošto niste sigurni šta ste tačno uradili, ne možete da znate šta tačno radi i da li ste našli sve što ne radi.

Bez plana i pripreme ne može biti ni očekivanja.

Imajte na umu da ovo nije neuobičajen način testiranja. Veoma često se organizuju bag-baš (engl. *Bug bash*) događaji, u kojima softverski tim pripremi aplikaciju za koju veruje da je spremna i ceo tim isprobava razne stvari. Ideja bag-baš testiranja je da članovi tima bez nekih ograničenja iz nekog novog ugla pristupe aplikaciji, primene neke neuobičajene alate ili situacije da dovedu aplikaciju u problem. Iako su bag-baš događaji korisni, ovo nije standardno testiranje koje se vrši svaki dan.

Ako ćete stvarno testirati neku aplikaciju, morate da uradite sledeće:

- Shvatite šta su zahtevi na osnovu kojih je implementirana aplikacija. Zahtevi će vam definisati šta treba da uradite.
- Odredite koje testove treba izvršiti kako biste proverili da li je aplikacija implementirana u skladu sa očekivanjima.
- Smislite na koji način najefikasnije možete da izvršite testove.
- Organizujete svoje aktivnosti tako da uvek znate šta treba testirati. Imajte na umu da testiranje nije samo rad sa aplikacijom. Veliki deo vremena ćete provesti na čitanju dokumenata, verifikovanju da li je problem ispravljen i slično.

U ovoj knjizi ćete naučiti kako da najefikasnije izvršite ove akcije.

Razumevanje zahteva

Prva stvar koju bi trebalo da uradite pre nego što počnete da testirate aplikaciju je da shvatite šta bi trebalo da radi, a šta ne. Ako ne shvatite zahteve po kojima su implementirane funkcionalnosti aplikacije, onda ne možete da je testirate. Bez razumevanja zahteva ćete unositi slučajne vrednosti u aplikaciju, dobijati slučajne rezultate i često nećete moći da shvatite da li je nešto bag ili očekivano ponašanje.

> *Zamislite da testirate aplikaciju koja ima samo jedno polje u koje se unosi broj i u kojoj kao rezultat dobijate novi broj. Kako ćete testirati takvu aplikaciju? Ako ne znate šta su bili zahtevi, ne možete znati koja funkcija se primenjuje na unetom broju. Testiranje koje predstavlja unošenje različitih brojeva i posmatranje izlaza će vam možda pomoći da shvatite koja je to funkcija koja transformiše broj, ali to je nešto što ste morali da pročitate u zahtevima.*

Dobra vest je da veliki broj softverskih aplikacija ima dokumentovane zahteve. S obzirom na to da je neko pre vas morao da implementira funkcionalnosti koje ćete testirati, on je morao da radi po nekoj listi zahteva, dokumentima ili specifikacijama koje opisuju šta je potrebno uraditi.

Loša vest je da tih dokumenata često ima previše, neki su zastareli, neki su protivrečni, tako da se ne možete uvek pouzdati u dokumente koje ćete dobiti. Često ćete naići na nepotpune ili zastarele informacije zato što je u poslednjem trenutku neko tražio da se nešto promeni u aplikaciji ili da se reši neki problem koji je promenio ponašanje aplikacije koje nije ispraćeno ažuriranjem dokumentacije. U nekim slučajevima ćete naići na ogromne količine (često protivrečnih) dokumenata i zahteva koje ćete teško upotrebiti zato što su se zahtevi menjali, a neko nije ažurno promenio sve delove dokumenata koji više nisu tačni. Informacije su često i u glavama ljudi koji su implementirali aplikaciju ili koji su se dogovarali sa korisnicima (a često ti ljudi možda više nisu u timovima koji rade sa aplikacijom). U nekim slučajevima su zahtevi sama aplikacija – ono što je zatečeno stanje u aplikaciji je i „očekivana" funkcionalnost, sem ako ne radi potpuno pogrešno. Nažalost, ovo je realnost u nekim slučajevima i sa tim se morate izboriti.

Bez obzira na kakve prepreke možete naići, morate iz dokumentacija ili razgovora sa ljudima da zaključite ŠTA aplikacija treba da radi pre nego što počnete da je testirate.

U idealnom slučaju, imaćete jasnu specifikaciju ili zahteve kojima se opisuju funkcionalnosti aplikacije. Ako je nemate, ili ako imate previše informacije, bilo bi dobro da napravite svoju sažetu listu zahteva. Jedan od najboljih načina da predstavite zahteve je da napravite hijerarhijsku dekompoziciju zahteva kao u sledećem primeru:

1. Prijavljivanje u sistem
    1.1. Učenik se prijavljuje pomoću korisničkog imena i lozinke
    1.2. Roditelj učenika se prijavljuje pomoću korisničkog imena i lozinke
    1.3. Nastavnik se prijavljuje u sistem pomoću:
        1.3.1. Korisničkog imena i lozinke
        1.3.2. Imejl adrese i lozinke
    1.4. Operator se prijavljuje pomoću korisničkog imena i lozinke

2. Pretraga učenika
   2.1. Nastavnik ili operator mogu da pretražuju učenike po imenu, razredu ili polu
   2.2. Rezultati se mogu sortirati i dodatno filtrirati
3. Unošenje učenika
   3.1. Operator može da unese informacije o novom učeniku.
4. Promena podataka o učeniku
   4.1. Razredni starešina može da unese zaključne ocene učenika za sve predmete
   4.2. Nastavnik može da unese ocenu učenika za predmet koji mu predaje
   4.3. Operator može da promeni podatke o adresi i roditeljima učenika
   4.4. Operator može da promeni razred učeniku

Ovakva funkcionalna dekompozicija zahteva će vam dati sliku o funkcionalnostima u sistemu. Čak i ako postoji definicija zahteva koja je opisana u nekim specifikacijama ili alatima za održavanje zahteva, može vam biti od koristi da ipak napravite ovakvu strukturu kako biste izlistali šta je sve potrebno da se testira.

U nekim projektima će zahtevi biti opisani narativno, tako da ćete naići na naziv neke akcije i opis kako neko može da je izvrši. Ovakvi zahtevi se nazivaju korisničke priče (engl. *user stories*):

Ja kao profesor unosim ocenu učeniku

Profesor određenog predmeta je prijavljen. Pronalazi predmet za koji treba da unese ocenu učenika. Pošto pređe na sekciju svojih predmeta, bira naziv predmeta. Tada mu se prikazuju učenici kojima predaje taj predmet. U listi bira učenika kom treba da unese ocenu; vidi sve ocene koje je učenik dobio iz tog predmeta. Profesor unosi novu ocenu i ona se dodaje u listu ocena

Korisničke priče su opisane narativno, pomoću nekoliko pasusa teksta. Alternativno, pasus se može zameniti listom akcija koje opisuju šta korisnik treba da uradi i šta će biti odgovor sistema. Ovakav format se naziva slučaj korišćenja (engl. *use case*). Primer jednog slučaja korišćenja je prikazan u sledećem primeru:

Ja kao profesor unosim ocenu učeniku

1. Profesor se prijavljuje. Aplikacija mu prikazuje listu predmeta.
2. Profesor pronalazi predmet za koji treba da unese ocenu učenika. Na strani o detaljima predmeta mu se prikazuju učenici kojima predaje taj predmet.
3. U listi bira učenika kom treba da unese ocenu. Sistem prikazuje sve ocene koje je učenik dobio iz tog predmeta.
4. Profesor unosi novu ocenu i ona se dodaje u listu ocena.

Slučajevi korišćenja su struktuirani kao lista koraka u obliku akcija korisnika – reakcija aplikacije. Bez obzira na to kako su opisani zahtevi, vaš zadatak je da se upoznate sa aplikacijom i proverite da li svaki od izlistanih zahteva radi kako je očekivano.

U nekim slučajevima ćete naići na loše formulisane ili neorganizovane zahteve. Ništa vas ne sprečava da sami napravite ovakve liste ili opise na osnovu postojećih zahteva na način koji vam odgovara. Ovo se može posmatrati kao pravljenje test plana i procedura.

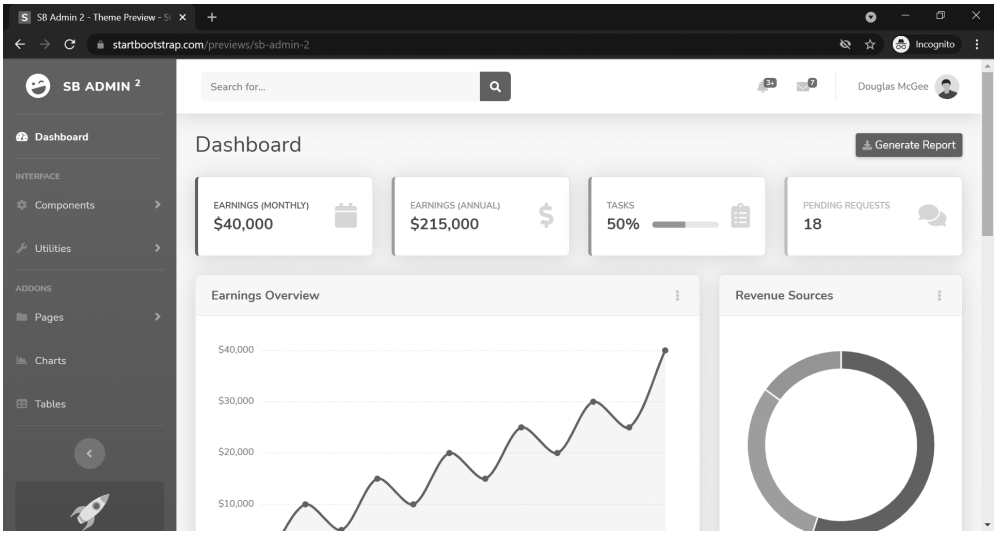
U nastavku ove knjige ćete se upoznati sa nekim od najbitnijih tehnika koje ćete koristiti dok testirate aplikaciju.

Test slučajevi

Test slučajevi opisuju šta je potrebno testirati. Kao što softverski zahtev opisuje šta softverski tim treba da implementira, tako i test slučaj opisuje test timu šta je potrebno da testira. Da biste testirali aplikaciju, potrebno je da identifikujete što je moguće više test slučajeva kako biste verifikovali da aplikacija radi ispravno. S druge strane, ne smete identifikovati previše test slučajeva pošto biste mogli da potrošite sve vreme koje vam je na raspolaganju na testiranje jedne funkcionalnosti.

Test slučajevi se identifikuju pregledom dokumentacije, zahteva, funkcionalnosti i dijagrama pomoću kojih možete da shvatite šta aplikacija radi.

Posmatrajmo izgled jedne strane napravljene pomoću SB Admin 2 teme:



Slika 1.1. Korisnički interfejs aplikacije, implementiran pomoću SB Admin teme

Pregledom ovakvog korisničkog interfejsa možemo da identifikujemo šta je potrebno testirati u aplikaciji.

Prvo je potrebno da identifikujete šta vidite u aplikaciji ili dokumentu. Po potrebi, pitajte druge članove tima da li postoje skrivene funkcionalnosti koje se ne vide na prvi pogled ili isprobajte aplikaciju. Kada identifikujete šta sve postoji na strani, razmislite šta sve treba da probate kako biste potvrdili da sve što ste identifikovali stvarno i radi. Na ovaj način identifikujete test slučajeve.

Test slučajevi koji definišu šta se testira na ovoj strani mogu da budu:

- Potrebno je proveriti da li se može uraditi pretraga pomoću tekstualnog polja na vrhu.
- Potrebno je proveriti da li se prikazuje ispravna vrednost za mesečnu zaradu.
- Potrebno je proveriti da li se prikazuje ispravna vrednost za godišnju zaradu.
- Potrebno je proveriti da li se prikazuje ispravan broj zadataka.

Svaka komponenta, broj, izveštaj, dugme, link ili bilo koji element koji prikazuje podatke ili omogućava korisniku da izvrši neku akciju je test slučaj koji treba proveriti. Koliko god da trivijalno izgleda neki link koji samo vodi na neku stranu, ako ga ne probate, može se desiti da referencira na pogrešnu ili nepostojeću stranu. Zbog toga je svaki element koji vraća informacije korisniku ili mu omogućava da nešto uradi kandidat za test slučaj koji treba proveriti.

Test slučajevi u prethodnom primeru su prikazani kao lista, ali vas ništa ne sprečava da ih predstavite u obliku hijerarhijske strukture, gde su grupisani slični slučajevi korišćenja. Čak i svaki zahtev u funkcionalnoj dekompoziciji zahteva, koja je prikazana u prethodnoj sekciji, može da se posmatra kao jedan test slučaj.

Postoje dve vrste test slučajeva:

- **Činjenice** su statički test slučajevi koji se izvršavaju sa fiksnim i predefinisanim podacima i uvek daju isti rezultat. Primer može biti prijavljivanje validnim korisničkim imenom i lozinkom.
- **Teorije** su dinamički i varijabilni test slučajevi u kojima se unose različite vrednosti test podataka kako bi se potvrdilo da se bez obzira na ulazne podatke uvek dobija očekivani rezultat. Na primer, pretrage se vrše sa različitim kriterijumima pretrage, ali bez obzira na to šta je uneto, uvek moraju da se prikažu rezultati pretrage koji odgovaraju kriterijumu.

Osnovna razlika između ovih test slučajeva su test podaci koji se koriste. U statičkim slučajevima se uobičajeno koriste precizno definisani podaci i očekuje se da aplikacija vrati tačno definisan odgovor. S druge strane, podaci koje koristite u teorijama su varijabilni i mogu se koristiti vrednosti i podaci iz određenih skupova. Odgovor je isto varijabilan i zavisi od ulaza i nekih pravila koja opisuju izlaz (npr. porez koji aplikacija prikazuje je 18% od unete cene), pa ćete nekada morati da izračunate šta treba da se prikaže u zavisnosti od ulaznih podataka.



Test slučajevi se mogu zamisliti kao lista stavki koje treba proveriti u aplikaciji i za svaki test slučaj potvrditi da radi kao što je očekivano. U situaciji da neki test slučaj ne radi kako je predviđeno, ovo se prijavljuje kao problem koji mora da se reši.

Kao što bi tim koji razvija softver prolazio kroz listu zahteva i implementirao svaki od zahteva, tako i tim za testiranje prolazi kroz listu test slučajeva i za svaki od njih verifikuje šta je potrebno uraditi.

Test slučajevi mogu da budu pozitivni ili negativni. Pozitivni test slučajevi predstavljaju testove kojima se proverava da li aplikacija radi ono što je očekivano. Ovakvi test slučajevi se identifikuju direktno na osnovu zahteva. Za svaki zahtev mora da postoji jedan ili više pozitivnih test slučajeva kojima se proverava da li aplikacija radi ono što je predviđeno. Negativnim test slučajevima se proveravaju sledeći scenariji:

- Da li aplikacija radi nešto što ne bi trebala da radi.
- Da li izvršava očekivane akcije u slučajevima u kojima korisnik radi nešto nepredviđeno.

Negativni test slučajevi često zahtevaju više kreativnosti kako bi bili identifikovani. Za razliku od pozitivnih test slučajeva, koji se definišu direktno na osnovu zahteva, u negativnim je potrebno izvršiti „duplu negaciju" zahteva ili identifikovati neočekivane ulazne vrednosti ili sekvence akcija koje nisu eksplicitno opisane zahtevima. Neki primeri negativnih test slučajeva su:

- Aplikacija kojom se pretražuju proizvodi mora da vrati sve proizvode koji zadovoljavaju kriterijum pretrage. Pored toga, aplikacija **ne sme** da vrati nijedan proizvod koji **ne zadovoljava** kriterijum pretrage.
- Aplikacija koja traži od korisnika da se prijavi korisničkim imenom i lozinkom treba da omogući korisniku koji je uneo ispravne podatke da pristupi sistemu. Pored toga, potrebno je potvrditi da korisnik koji unese **neispravne** podatke **neće** moći da pristupi sistemu.

Bitno je da za svaku funkcionalnost identifikujete i pozitivne i negativne test slučajeve. Negativne test slučajeve je teže identifikovati, ali zato se uz njih nalazi više bagova. Kada se promaši neki negativni test slučaj, česta je reakcija „Kako se niko nije setio da proba ovo?".

U naredim poglavljima ćemo videti neke tehnike kojima se identifikuju test slučajevi na osnovu funkcionalnosti. Potom ćemo se upoznati sa nekim alatima koje možete koristiti kako biste unosili test slučajeve i prijavljivali probleme koje nađete.

Test vektori

U test slučajevima morate da koristite neke podatke koje ćete unositi u aplikaciju i koje ćete koristiti kako biste poredili rezultate. Na primer, ako testirate prijavljivanje korisnika u sistem, morate da unesete dve tekstualne vrednosti – korisničko ime i lozinku.

Test vektori su grupe vrednosti koje ćete koristiti u testovima. Na primer, zamislimo da se testira formular u koji se unose informacije o osobi, kao na primer ime, godina rođenja i informacija da li je zaposlena ili ne. U ovom slučaju se mogu koristiti sledeći test vektori:

TV1	(Danica, 2020-09-23, true)
TV2	(, 2020-02-29, false)
TV3	(Lana, 2020-01-01,)

Cilj planiranja testa je pronaći najmanji skup test vektora kojim će se testirati najviše funkcionalnosti za najmanje vremena.

Vrednosti u test vektorima se unose u polja za unos podataka, formulare, fajlove i slično. Ove vrednosti se nazivaju **dimenzije**. U prethodnom primeru, dimenzije su ime, godina rođenja i status zaposlenja. Sve vrednosti vektora imaju iste dimenzije, mada u nekim vektorima neke od dimenzija mogu biti bez podataka.

Neki test slučajevi će se odnositi na jedan test vektor, i to su uglavnom statički test slučajevi, odnosno činjenice u kojima test vektor tačno definiše koje podatke treba uneti. Pritom test vektor sadrži i informacije koje treba uneti i informacije koje treba očekivati kao rezultat. Informacije u test vektoru se koriste kako bi se proverile činjenice.

U drugim test slučajevima možete imati skup različitih test vektora kojima ćete proveriti da li se sistem ponaša očekivano. Test vektori se dinamički grade tako što se vrednosti dimenzija slučajno ili po nekom pravilu uzimaju iz predefinisanog skupa vrednosti ili tako što se vrednosti iz različitih skupova kombinuju u jedan vektor. Ovo su dinamički vektori kojima se proveravaju određene teorije o aplikaciji.

Test scenariji

Test scenario je sekvenca akcija koje treba da uradite kako biste testirali aplikaciju. Test scenario se najlakše može zamisliti kao skript koji treba da pratite, izvršavate akcije i provere. Primer test scenarija može da bude:

1. Prijavi se u aplikaciju kao profesor. Verifikuj da se prikazala početna strana za profesore sa imenom prijavljenog profesora u naslovu.
2. Pređi na raspored časova. Trebalo bi da se prikaže strana sa kalendarom časova, podeljenim po danima.
3. Odaberi neki čas u kalendaru. Trebalo bi da se prikažu detalji o času, koji sadrže naziv predmeta, kao i plan za dati čas.

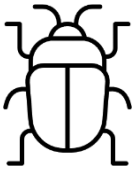
Test scenariji su često opisani kao liste akcije kroz koje treba proći, sa pravilima koja treba proveriti kako bi se testiralo ponašanje.

Svaka akcija u test scenariju može da bude jedan test slučaj koji treba proveriti. Kada napravite skup test slučajeva, test scenario može da opiše šta je potrebno uraditi sa aplikacijom kako bi se proverili svi identifikovani test slučajevi. U drugim slučajevima možete imati test slučajeve koji nisu jednostavna provera oblika radi/

ne radi. Neki test slučajevi mogu da sadrže svoje test scenarije koji opisuju šta je sve potrebno uraditi kako bi se izvršila provera.

Bagovi

Dok testirate aplikaciju, pronalazićete različite probleme, nelogičnosti i nekonzistentnosti. Svako pogrešno ponašanje aplikacije se naziva **bag** (engl. *bug*).



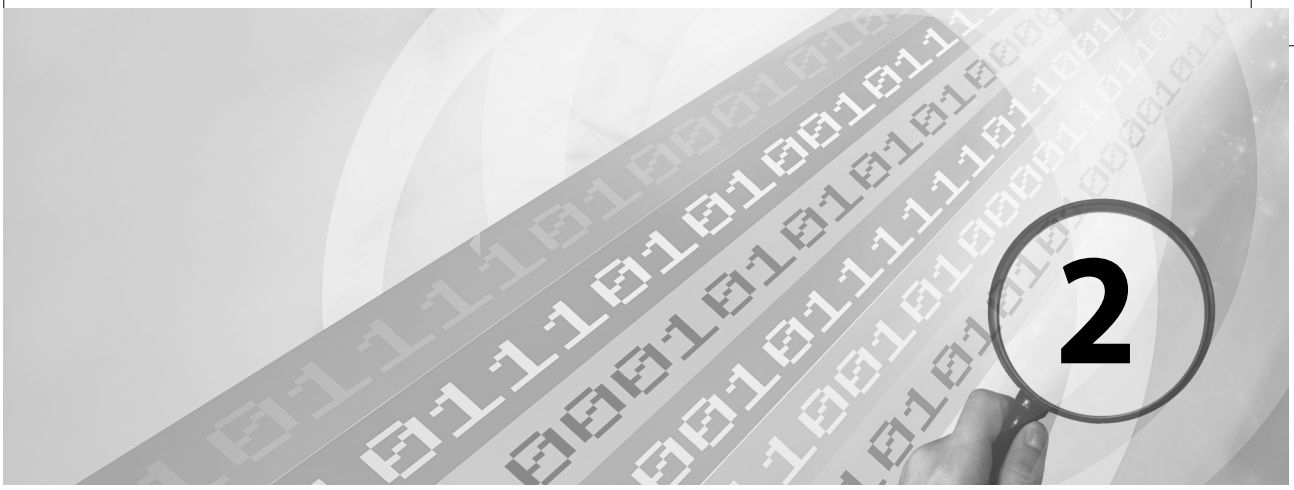
Bag je naziv nastao po bubama koje uzrokovale kvarove dok su ulazile u elektronske cevi starijih vrsta računara. Proces uklanjanja bagova se naziva **debagovanje**.

Postoje neke informacije da je još Tomas Edison govorio o bagovima koji su pravili probleme u njegovim uređajima.

Vaš zadatak je da napravite plan kako ćete koristiti aplikaciju, pratiti test slučajeve i proveravati test slučajeve kako biste identifikovali bagove, koji predstavljaju odstupanje od očekivanog ponašanja.

Svaki bag koji pronađete mora biti jasno opisan kako bi neko ko će ispraviti taj bag imao sve informacije koje će mu biti potrebne da ga reprodukuje. Ovo može da uključi korake koje treba preduzeti, podatke koje treba uneti, slike stanja aplikacije koju vidite (engl. *screenshots*), opis okruženja (npr. vrsta pregledača, rezolucija ekrana) i slično. Ako prijavite bag koji nema dovoljno opisa, možda niko neće moći da ga reprodukuje. U još gorem slučaju, vi ćete ga jednom pronaći, ali nećete moći da ponovite korake i ponovite istu grešku zato što nemate sve informacije o koracima koje ste preduzeli.

Zbog toga je potrebno da budete veoma disciplinovani kada prijavljujete bagove i unesete sve potrebne informacije koje će i vama i drugima pomoći da ga reprodukujete.



Tehnike testiranja crne kutije

Crna kutija (engl. *Black box*) je sistem u kojem znamo šta treba da radi, ali ne znamo kako. Kada razumete zahteve na osnovu kojih je implementirana aplikacija, ali ne znate koje tehnologije, arhitekture i komponente se koriste u pozadini, ovu aplikaciju možete posmatrati kao crnu kutiju.

Veoma često ćete dobijati zahteve da proverite da li rade strane koje izgledaju kao strana na sledećoj slici:



Slika 2.1. Strana aplikacije koju je potrebno testirati

Videćete izgled korisničkog interfejsa, dobiti neke zahteve i opise koji govore šta radi ovaj korisnički interfejs, i na osnovu toga bi trebalo da proverite da li ovaj deo aplikacije radi kako se očekuje. Ako ne radi, potrebno je da prijavite bagove koji predstavljaju greške koje ste našli.