TESTIRANJE DJANGO APLIKACIJA

TEHNIKE CRNE KUTIJE (BLACK BOX)

- Kod tehnika crne kutije analiziramo samo funkcionalnosti na osnovu funkcionalne specifikacije definisanih zahteva
- Koristimo iz projekta:
  - *Dokument projektni zadatak (iz faze 1)*
  - *Specifikaciju svih slučajeva upotrebe (iz faze 2)*
- Primenjujemo metodu klasa ekvivalencije (eng. Equivalence class partitioning)
- Realizujemo test plana sa test primerima
- Automatizovano testiranje korišćenjem nekog alata i provera ponašanja u različitim pregledačima

METOD DELJENJA NA KLASE EKVIVALENCIJE

- Ideja: podela ulaznih podataka na podskupove (klase)
- Svi ulazni podaci koji daju iste (slične) rezultate pripadaju istoj reprezentativnoj klasi
- Idealni slučaj: podskupovi međusobno disjunktni i pokrivaju ceo skup ulaznih podataka
- Za svaki uslov se posmatraju dve grupe klasa:
  - *Legalne klase - obuhvataju ispravne situacije (ulazne podatke)*
  - *Nelegalne klase - obuhvataju sve ostale situacije (ulazne podatke)*
- Prednost ovog metoda: vreme potrebno za testiranje

IDENTIFIKACIJA KLASA

- Ulazni podatak na osnovu opsega:
  - *jedna legalna klasa (unutar opsega)*
  - *dve nelegalne klase (levo i desno od opsega)*
- Primer: mesec uzima broj iz opsega 1-12
- Ulazni podatak na osnovu dužine ili formata
  - *jedna legalna klasa (tačan broj karaktera/cifara)*
  - *dve nelegalne klase (za manji/veći broj karaktera)*
  - *Primer: JMBG ima tačno 13 cifara*
- Tačno definisan skup ulaznih podataka
  - *po jedna legalna klasa (za svaku pojedinačnu vrednost) i jedna zajednička nelegalna klasa*

IDENTIFIKACIJA KLASA

- Uslov obaveznosti za ulazni podatak
  - *jedna legalna klasa za ispravnu vrednost*
  - *i bar jedna nelegalna klasa za neispravnu vrednost*
- U slučaju da program ne tretira jednako sve elemente neke klase ekvivalencije, nju treba podeliti na više manjih klasa

TEST PLAN

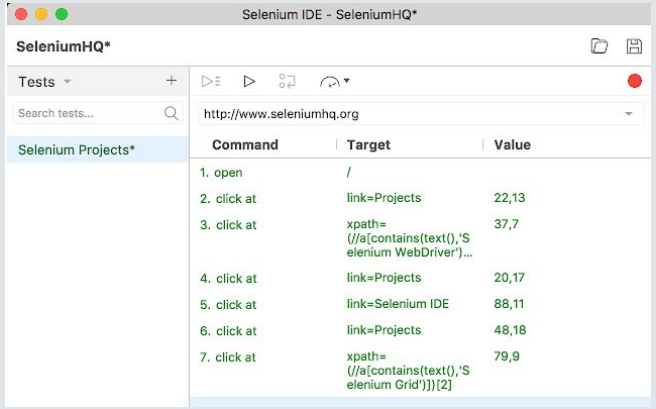
- Test plan (eng. Test suite) sastoji se iz test primera (eng. Test case)
- Svaki test primer treba da bude dizajniran pre implementacije i izvršavanja testa
- Nakon što se odrede klase ekvivalencije, formiraju se test primeri za:
  - *sve legalne klase ekvivalencije (cilj je da se što manje test primera primeni na što više klasa ekvivalencije)*
  - *sve nelegalne klase ekvivalencije (za svaku nelegalnu klasu po jedan test primer, da se ne bi desilo da jedan test primer koji nije regularan maskira neki drugi test primer)*

GRANIČNI SLUČAJEVI

- Opciono možemo koristiti i metod graničnih slučajeva (eng. Boundary value testing)
- Metod klasa ekvivalencije proširujemo, tako da se fokusiramo na granice svake klase, jer se tu krije veliki broj grešaka
  - *Primer: popust na avio karte u kompaniji Air Serbia, za decu do 7 godina, da li je popust ako dete nije napunilo 7, ili u celoj kalendarskoj godini, dok ne napuni 8 god.*

KORACI METODE GRANIČNIH SLUJAJEVA

- Identifikujemo klase ekvivalencije, a zatim identifikujemo granice svake klase ekvivalencije.
- Pravimo test primer za svaku graničnu vrednost tako što biramo:
  - *Jednu tačku A na samoj granici*
  - *Jednu tačku B ispod granice*
  - *Jednu tačku C iznad granice*
  - *Tačke B i C treba vrlo pažljivo odabrati i te tačke zavise od jedinice vrednosti podatka koji analiziramo. Takve tačke mogu biti i u drugim klasama ekvivalencije, pa voditi računa da testovi NE BUDU DUPLIRANI.*



TESTIRANJE VEB APLIKACIJA

- Manuelno testiranje
- Automatsko GUI testiranje (web automation testing)
- Primer:
  - *Selenium IDE framework*
  - *iMacros*
- Postoji i Selenium WebDriver:
  - *kompaktni OO API*
  - *za različite veb pregledače (cross browser testing)*

TEHNIKE BELE KUTIJE (WHITE BOX)

- U literaturi još i kao:
  - *strukturno testiranje (eng. structural testing),*
  - *testiranje čistom kutijom (eng. clear box),*
  - *otvorenom kutijom (eng. open box),*
  - *staklenom/transparentnom kutijom (eng. glass box / transparent box),*
  - *zasnovano na kodu (eng. code-based testing)*
- Primarni izvor za projektovanje testova je izvorni kod sa fokusom na tok kontrole i tok podataka
- Cilj strukturnog testiranja nije da se izvrše sve moguće funkcije programa, već da se izvrše/aktiviraju različite programske i strukture podataka u programu

TEHNIKE KONTROLE TOKA

- Grafovi kontrole toka predstavljaju vizuelnu reprezentaciju strukture koda nekog programa. Izvršavanjem programa (na primer, za neke test podatke) vrši se izbor neke putanje u grafu.
- Vrste:
  - *Pokrivenost iskaza (eng. Statement Coverage) = pokrivenost instrukcija ili pokrivenost koda*
  - *Pokrivenost odluka ili pokrivenost grana (eng. Decision or Branch Coverage)*
  - *Pokrivenost uslova (eng. Condition Coverage)*
  - *Pokrivenost višestrukih uslova (eng. Multiple Condition Coverage)*
  - *Minimalna pokrivenost višestrukih uslova (eng. Minimal Multiple Condition Coverage)*
  - *Pokrivenost odluka i uslova (eng. Decision/Condition Coverage)*
  - *Modifikovana pokrivenost odluka i uslova (MC/DC)*

NIVOI TESTIRANJA

- Jedinično testiranje (eng. unit testing)
- Integraciono testirane (eng. integration testing)
- Sistemsko testiranje (eng. system testing)

CILJ JEDINIČNOG TESTIRANJA

- Šta dobijamo jediničnim testovima?
  - *identifikovanje grešaka i ispravke u kodu*
  - *refaktorizacija koda*
  - *dokumentacija za najmanje elemente programskog koda (koji se testiraju)*
- Da bi sve ovo postigli, moramo da pokrijemo sve putanje u kodu
- Jedan test obično pokrije jednu putanju u funkciji

DJANGO UNIT TESTIRANJE

- Django ima podršku za testiranje korišćenjem unittest modula (https://docs.djangoproject.com/en/5.0/topics/testing/)
- Svi testovi za jednu aplikaciju se pišu u okviru tests.py fajla
- Svaki test se piše u okviru test klasa
- Test klase se izvode iz:
  - *SimpleTestCase*
  - *TestCase, radi isto kao SimpleTestCase, samo sadrži dodatne funkcionalnosti*
- Metode setUp i tearDown test klasa – aktiviraju se pre i posle svakog testa
- Testovi su ostale metode test klasa, naziv im počinje sa test_

PROVERAVANJE ISPRAVNOSTI

- Samu ispravnost ponašanja aplikacije je moguće testirati put metoda za proveru – assert
- assertEquals koja proverava da li je prva vrednost jednaka drugoj
- Da bi se emuliralo ponašanje klijenta kako pristupa aplikaciji, koristi se klasa Client
- Primer testa koji proverava status odgovora:

```
def test_movies_GET(self):
    response = self.client.get(reverse('movies'))
    self.assertEquals(response.status_code, 302)
```

KOMUNIKACIJA SA BAZOM

- Django podrazumevano pravi praznu kopiju baze podataka nad kojom se radi testiranje, da bi nezavisno radili uvek
- Ovo podešavanje je moguće pregaziti
- Samim tim svaka interakcija sa bazom podataka je u samo okviru testova
- setUp i tearDown se koriste za inicijalizaciju/deinicijalizaciju baze za testiranje
- Primer kreiranja korisnika:

```
def setUp(self):
    self.client = Client()
    user = User.objects.create(username='luka')
    user.set_password('Luka12345')
    user.save()
```

TESTIRANJE SADRŽAJA STRANICE

- Česta situacija (kao što je i rađeno primarno na predmetu) je da poslovna logika klijentu ne vraća čiste podatke, već HTML sadržaj
- Kako se testira ovo?
- Korišćenjem assertContains, koja proverava postojanje sadržaja u HTTP odgovoru koji je prvi argument
- Da bi se testirao sam HTML sadržaj stranice, potrebno je postaviti html parametar metode na True
- Primer:

```
self.assertContains(response, 'Dune', html=True)
```

ŠTA AKO JE FUNKCIONALNOST ULOGOVANOG KORISNIKA?

- U ovim situacijama, potrebno je emulirati efekat login-a korisnika da bi se testirala takva funkcionalnost
- Postoje dva načina za ovo:

1. Slanje POST zahteva na putanju za login sa ispravnim kredencijalima
2. Korišćenjem integrisane metode login klase Client

```
self.client.post(reverse('index'), data = {
    'username' : 'luka',
    'password' : 'Luka12345'
})
```

```
self.client.login(
    username='luka',
    password='Luka12345'
)
```

INTEGRACIJA SA SELENIUMOM

- Same mogućnosti testiranja HTML sadržaja u HTTP odgovoru je veoma šturo
- Zato se često integriše Selenium WebDriver sa Django testiranjem radi lakšeg testiranja na osnovu HTML sadržaja
- Da bi radilo testiranje sa Selenium-om, testovi koji koriste ovu biblioteku se izvode iz klase StaticLiveServerTestCase
- Neophodno je podesiti i pretraživač kojim se vrši testiranje, u ovom slučaju Chrome
- Potrebno je preuzeti Chrome Driver (https://developer.chrome.com/docs/chromedriver/downloads)

PODEŠAVANJE PRETRAŽIVAČA UZ SELENIUM

- Potrebno je podesiti sam pretraživač nad kojim se radi testiranje – prosleđivanjem putanje do Chrome Driver-a
- Takođe se podešava korena putanja nad kojom se izvršavaju svi testovi
- Ovo može da se radi u setUp metodi:

```
service = webdriver.ChromeService(executable_path='<putanja>\\djangoProject\\cinema\\chromedriver.exe')
self.browser = webdriver.Chrome(service = service)
self.appUrl = self.live_server_url + "/cinema"
```

- Gašenje pretraživača u tearDown metodi

```
self.browser.close()
```

PREDNOSTI WEBDRIVER-A

- Sada je moguće naredbama zahtevati svaki klik na pretraživaču u okviru aplikacije, odnosno simuliranje ponašanja klijenta kao u Selenium IDE
- Primer unosa putanje u pretraživaču: `self.browser.get(self.appUrl)`
- Primeri dohvatanja elemenata: `username = self.browser.find_element(By.NAME, 'username')`
- Unos elementa u formi (username je dohvaćen element): `username.send_keys("luka")`
- Primer dohvatanja elementa koje je dugme u formi i klika za predaju forme:

```
submit = self.browser.find_element(By.XPATH, '//input[@type="Submit"]')
submit.click()
```

- Ako se prelazi na drugu stranicu, idealno je staviti pauzu pre izvršavanja naredne komande (npr. sa time.sleep)

TESTIRANJE POKRIVENOSTI KODA

- Moguća integracija sa alatima za pokrivanje koda
- Ovi alati daju informacije o pokrivenosti koda aplikacije prilikom izvršavanja testa
- Najčešći alat koji se koristi za ove potrebe je coverage.py (https://docs.djangoproject.com/en/5.0/topics/testing/advanced/#integration-with-coverage-py)
- Nakon instalacije coverage paketa, pokrenuti komandu:
  `coverage run --source='.' manage.py test myapp`
- A izveštaj se generiše sa: `coverage report`