VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
KOMPIUTERINIO IR DUOMENŲ MODELIAVIMO KATEDRA

Kursinis darbas

Automatinio testavimo metodologijų tyrimas

Atliko:
Adomas Bazinys parašas

Vadovas:
Dr. Joana Katina

Vilnius
2019

Turinys

Santrauka

Per pastaruosius 30 metų programinės įrangos pramonė padarė didelę pažangą ir pakeitė daugelio žmonių gyvenimą. Šiais laikais skambinant, užvedant automobilį, įjungiant elektrinę viryklę, atsiskaitant debeto kortele, daugelis net nenumanome, kad tokių procesų metu vykdome tam tikrą programinės įrangos kodą. Paprastai programinė įranga susiduria su dvejais iššūkiais: kuo mažesnės programavimo darbų išlaidos ir produkto kokybės užtikrinimas. Kokybiška programinė įranga gali būti apibūdinama kaip naudinga, patikima ir saugi programinė įranga. Programinės įrangos kokybė turi įtakos pasaulio ekonomikai, nacionaliniam saugumui, sveikatos priežiūrai, todėl programinės įrangos kokybė yra didžiulė atsakomybė. Šis darbas yra orientuotas į internetinių aplikacijų kokybę bei turi tikslą atsakyti į klausimą, kaip automatinis testavimas gali prisidėti prie internetinių aplikacijų kokybės.

Summary

Automated Testing Frameworks Reliability Impact on Web Application Development Life Cycle

Over the last 30 years, the software industry has made great progress and has changed the lives of many people. Nowadays, when calling a phone, starting a car, switching on an electric stove, making a debit card payment we do not even think that during such a processes we run a certain program code. Typically, software has to overcome two challenges: reduce the cost of development as possible and ensure the high quality. High quality software can be described as usable, dependable, and safe software. Software quality a ects the global economy, national security, health, safety, so software quality is a huge responsibility. This work is focused on the quality of web applications and will try to answer the question of how automated testing can contribute to software quality.

Trumpinių sąrašas

ODBC - Open Database Connectivity.

CSV - Comma Separated Values.

SQL - Structured Query Language.

SUT - System Under Test.

Įvadas

Didėjant WEB aplikacijos apimčiai ir naudotojų kiekiui, tampa vis sunkiau užtikrinti sistemos kokybę rankiniu būdu ir nepriekaištingas kokybės užtikrinimo procesas tampa neatskiriama aplikacijos vystymo dalimi. Tokiu atveju rankinis testavimas atsiranda poreikis naudoti automatinį testavimą.

Atliekant kodo pertvarkymą (**angl.** refactoring) arba naujų funkcijų įgyvenimą dažnai klaidos aptinkamos jau egzistuojančiuose kodo vienetuose. Tai gali padaryti didžiulę įtaką būsimo produkto kokybei, todėl testavimas turi būti pradedamas jau ankstyvoje programinės įrangos vystymosi ciklo stadijoje. Automatinis testavimas yra efektyvus būdas patikrinti aplikacijos kokybę prieš išleidžiant ją į produkciją.

Šio darbo tikslas yra išsiaiškinti ir pagrįsti kodėl ir kuris automatinio testavimo tipas yra efektyviausias, patikimiausias, bei turintis aukščiausią pakartotinio panaudojimo (**angl.** reusability) lygį. Bus rašomi testai, naudojant ir išbandant kiekvieną egzistuojantį automatinio testavimo tipą ir samprotaujama, kuris iš jų labiausiai atitinka paminėtus kriterijus. Pavyzdžiam bus naudojama JAVA programavimo kalba ir Selenium bei Cucumber testavimo karkasai.

1. Įvadas į testavimą

Daugelis programuotojų tikriausiai sutiktų, kad defektai aplikacijoje dažnai atsiranda ne tik ją kuriant, bet ir išleidus ją į produkciją. Tikimybę sumažinti defektų skaičių padeda automatinis testavimas [4].

Automatinis aplikacijų testavimas paprastai yra atliekamas rašant ir vykdant tam tikrus kodo vienetus, vadinamus testais. Kiekvienas testas naudoja jau egzistuojančią funkciją, kurią norima ištestuoti. Pagrindinis šio testavimo tikslas yra užtikrinti, kad gaunamas rezultatas yra toks, kokio tikimasi.

Kai kurios programavimo kalbos testavimui turi tam tikrus **assert** metodus, kurie tikrina, ar sąlyga yra teisinga - kitu atveju grąžina pranešimą apie įvykusią klaidą. Tačiau vien to testavimui neužtenka. Pavyzdžiui, mes norime paleisti tam tikrą kodo dalį prieš vykdant kiekvieną testą, taip pat, kad būtų grąžinamas tikslus pranešimas apie tai, kur yra klaida ir kodėl ji įvyko. Tai yra priežastys, kodėl testavime yra reikalingas testavimo karkasas (**liet.** testing framework) [2].

1.1. Defekto apibrėžimas

Svarbu suprasti, kad ne kiekviena problema aplikacijoje yra defektas. Defektas yra problema, kuri apriboja aplikacijos funkcionalumą arba neatitinka kliento reikalavimų. Jei aplikacija veikia tinkamai ir atitinka kliento poreikius, galima teigti, kad ji neturi defektų [10].

1.2. Rankinis testavimas

Rankinis testavimas yra primytiviausias testavimo būdas, kai testuotojas testavimo scenarijus vykdo rankiniu būdu - nenaudodamas jokių testų automatizacijai skirtų įrankių. Rankinio testavimo specialistas paprastai atlieka galutinio vartotojo vaidmenį, siekiant įsitikinti, ar visos tinklalapio funkcijos veikia taip, kaip tikimasi. Rankinis testuotojas dažnai vadovaujasi testavimo planu, į kurį yra įtraukti svarbiausi testavimo scenarijai [5].

1.2.1. Privalumai

- Nereikia programavimo kalbos žinių.
- Puikiai tinkamas būdas testuoti nedidelės apimties aplikacijas.

1.2.2. Trūkumai

- Neefektyvu, reikalauja daug laiko.
- Sunkiai vykdomas regresinis testavimas.
- Neįmanoma pratestuoti aplikacijos greitaveikos (**angl.** performance).

1.3. Automatinis testavimas

Automatinis testavimas yra testavimo būdas, kai yra rašomi testavimo skriptai, naudojantis kokia nors programavimo kalba ar įrankiu, atliekantys tam tikrus paspaudimus grafinėje sąsajoje ir fiksuojantys testų rezultatus.

1.3.1. Privalumai

- Lengva atlikti regresinį testavimą.
- Patogus, automatiškai atliekamas rezultatų apibendrinimas.
- Parašius testavimo skriptą, testavimo procesas tampa daug efektyvesnis.

1.3.2. Trūkumai

- Būtinos bent vienos programavimo kalbos žinios.

2. WEB aplikacijos plėtojimo ciklas

Kuriant WEB aplikaciją, būtina remtis ir sekti tam tikrą vystymo metodologiją, norint užtikrinti nuoseklumą ir užbaigtumą. WEB aplikacijos vystymosi ciklas susideda iš kelių fazių [3]:

1. Planavimas.
2. Analizė.
3. Plėtojimas.
4. Testavimas.
5. Paleidimas ir palaikymas.

2.1. Planavimas

Pati pirmoji WEB aplikacijos plėtojimo ciklo fazė yra planavimas. Jei planavimo fazė yra praleidžiama arba į ją žiūrima atmestinai, tada visos sekančios fazės užtikrintai bus taip pat nenusisekusios, o tai reikštų projekto žlugimą [3]. Į planavimą įeina:

- Apsisprendimas, kokias technologijas naudoti.
- Projekto tikslo apibrėžimas.
- Tikslinės grupės apibrėžimas.
- Nustatymas, kokį turinį aplikacija talpins.

2.2. Analizė

Tai yra fazė, kuomet analitikas renka informaciją apie reikalvimus, sistemingai juos analizuoja ir aprašo aplikacijos funkcionalumą pagal surinktą informaciją. Taip pat šioje stadijoje yra renkama informacija apie kliento lūkesčius greitaveikai (**angl.** performance), serverio atsako laikui (**angl.** response time) ir panašiai [3].

2.3. Plėtojimas

Šioje fazėje yra sukuriama pati WEB aplikacija. Programuotojai ir dizaineriai paverčia ankstensėse fazėse sukurtą sistemos modelį į pilnai funkcionuojančią aplikaciją [3].

2.4. Testavimas

Testavimas yra pati svarbiausia WEB aplikacijos vystymo ciklo dalis. Šioje fazėje lyginame, ar esamas funkcionalumas yra toks pats kaip ir funkcionalumas, kurio yra tikimasi. Jei skirtumo tarp šių funkcionalumų nėra, galime laikyti, kad sukurtas produktas yra kokybiškas ir atitinkantis kliento reikalavimus [3]. Į šią fazę taip pat įeina tokie patikrinimai kaip:

- Nuorodų WEB aplikacijoje veikimas.
- Serverio atsako laikas.
- WEB aplikacijos veikimas skirtingose naršyklėse.
- Žmogaus Kompiuterio sąveika.
- Validacijos.
- WEB aplikacijos paleidimas įvairiuose įrenginiuose: telefonai, planšetės ir t.t.

2.5. Paleidimas ir palaikymas

Į šią fazę įeina WEB aplikacijos, duomenų bazių valdymo sistemos paleidimas į produkciją, kad produktu galėtų naudotis tikslinis vartotojas [3]. Šioje fazėje atliekamos tokios užduotys, kaip:

- Paleidimas į produkciją.
- Turinio atnaujinimas.
- Sistemos stebėsena.
- Informacijos apie lankytojus registravimas (**angl.** logging).

3. Automatinio testavimo karkasų tipai

Automatinio testavimo karkasas yra rinkinys įrankių ir taisyklių, skirtų testavimo scenarijų kūrimui. Tai yra automatizuoto testavimo dalis, kuri padeda testuotojams efektyviau naudoti išteklius ir palengvina jų darbą.

3.1. „Linear Scripting“ testavimas

Ši testavimo metodika dar kitaip vadinamas "Įrašyk ir Atkurk" (**angl.** "Record and Playback") karkasu. Ji naudojamas testuoti nedidelės apimties WEB aplikacijoms. Naudojant šį testavimo būdą yra galimybė sugeneruoti testavimo skriptą neskiriant tam daug laiko, tačiau tai turi labai didelių trūkumų kaip pakartotinio panaudojamumo (**angl.** reusability) trūkumas, į testavimo skriptą įkoduotos reikšmės, kas neleidžia vykdyti scenarijaus naudojant kitus duomenis ar jų rinkinius. Šiam testavimo būdui praktiškai nereikalingos jokios programavimo žinios, norint testuoti šiuo būdu tereikia savo kompiuterio naršyklėje susinstaliuoti papildinius, tokius kaip Selenium ID, Katalon Recorder arba Robot Framework ir įrašinėti tam tikras veiksmų sekas galutiniame rezultate sugeneruojant testavimo scenarijaus skriptą norima kalba [7].

3.2. Ištrauka iš sugeneruoto "Record and Playback" skripto

1 išeities kodas. "Record and Playback" tipo testas.

```
click on linkText=Register OK
click on id=gender-male OK
click on id=FirstName OK
type on id=FirstName with value Tomas OK
click on id=LastName OK
type on id=LastName with value Tomaitis OK
click on name=DateOfBirthDay OK
select on name=DateOfBirthDay with value label=9 OK
click on name=DateOfBirthDay OK
click on name=DateOfBirthMonth OK
select on name=DateOfBirthMonth with value label=March OK
click on name=DateOfBirthMonth OK
click on css=.fieldset:nth-child(1) > .form-fields OK
click on name=DateOfBirthYear OK
select on name=DateOfBirthYear with value label=1997 OK
click on name=DateOfBirthYear OK
click on id=Email OK
type on id=Email with value adomas.adomaitis@gmail.com OK
click on id=Company OK
type on id=Company with value Company OK
click on css=.fieldset:nth-child(4) > .form-fields OK
click on id=Password OK
type on id=Password with value password OK
click on id=ConfirmPassword OK
type on id=ConfirmPassword with value password OK
click on css=.page-body > form OK
click on id=register-button OK
```

Šį Selenium IDE sugeneruotą skriptą konvertuoti į JAVA programavimo kalbą ir sėkmingai pasileisti.

3.3. „Modularity Driven“ testavimas

Daugelyje WEB aplikacijų egzistuoja funkcijos, kurios yra dažnai kartojamos ir priklausomos nuo kitų funkcijos. Ši testavimo metodika yra paremtas modulinėmis klasėmis, kuriose saugomi tam tikros funkcijos iškvietimo metodai. Tam, kad kiekvieną kartą neapsirašinėti kažkokios WEB puslapio funkcijos iškvietimo, modulinėje klasėje yra naudinga apsirašyti metodą ir esant poreikiui jį išsikviesti. Tai vadinama moduliškumu (**angl.** modularity). Norint atlikti pakeitimą tam tikros funkcijos iškvietime, užtenkta tik pakeisti metodą, iškviečiantį WEB puslapio funkciją. Metodas gali būti iškviečiamas keliose vietose, todėl šio testavimo tipo nauda ta, kad nereikia keisti to pačio kodo keliose vietose. Tai sumažina kodo duplikaciją ir palaiko "švarią" testinio kodo struktūrą [6].

3.3.1. Modelinės klasės pavyzdys

2 išeities kodas. Testavimo naudojant modulius pavyzdys - modelinė klasė.

```
public class PriceFilterModule {
...
@FindBy(linkText = "Padangos")
private WebElement tiresCat;

@FindBy(linkText = "Vasarin[U+FFFD]s padangos")
private WebElement summerTiresCat;

public void chooseTiresCategory() {
   tiresCat.click();
}

public void chooseSummerTiresCategory() {
   summerTiresCat.click();
}

public boolean doesItemsCorrespondToPriceRange() throws
   IOException {
   for (int i = 0; i < fetchItemsPrices().size(); i++) {
      Double checkingValue = fetchItemsPrices().get(i);
      if (!(FilteringConstants.MIN_PRICE < checkingValue &&
         checkingValue < FilteringConstants.MAX_PRICE)) {
         return false;
      }
   }
   return true;
}
...
}
```

Šis metodas tiesioginės įtakos produkto kokybei nedaro, tačiau turi įtakos teisingai ir "švariai" testavimo scenarijų kodo struktūrai. Ji daro įtaką atliekamų testų kokybei, kurie parodydami teigiamą rezultatą suklaidina testuotoją ir produktas prieš išleidžiamas į produkciją nėra kokybiškai ištestuojamas.

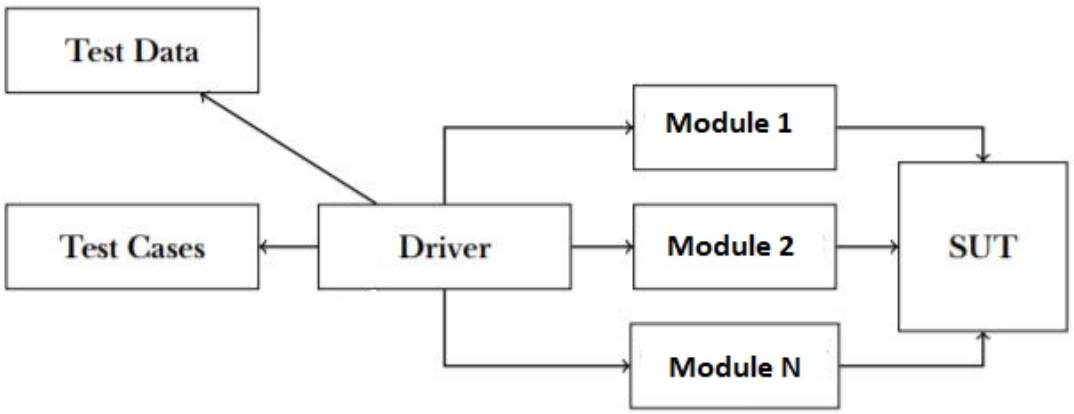
3.3.2. Testinės klasės pavyzdys

3 išeities kodas. Testinė klasė, kviečianti modulines klases.

```
public class PriceFilterModuleTest extends BaseTest {

    private HomePageModule homePageModule;
    private PriceFilterModule priceFilterModule;

    @Test
    public void filterItemsByPrice() throws Exception {
        homePageModule = new HomePageModule(driver);
        priceFilterModule = new PriceFilterModule(driver);
        commonFunc = new CommonFunctions(driver);
        homePageModule.openHomePage();
        commonFunc.maximazeWindow();
        homePageModule.acceptCookies();
        homePageModule.chooseCarsCategory();
        priceFilterModule.chooseTiresCategory();
        priceFilterModule.chooseSummerTiresCategory();
        priceFilterModule.setPriceScale();
        priceFilterModule.checkPriceFilterCorrectness();
    }
}
```



1 pav. "Modularity Driven" testavimo vykdymo schema.

3.4. „Data Driven“ testavimas

„Data Driven“ testavimas yra metodika, kai testavimo duomenys yra saugomi lenteliniu formatu (ODBC, CSV, Excel, SQL) - kiekvienos eilutės duomenys yra atskiras testavimo scenarijus. Šios metodikos esmė tokia, kad testavimo skriptas ima testavimo duomenis iš vieno iš paminėtų šaltinių ir įvykdo skriptą su kiekvienu duomenų rinkiniu. Paprastai testavimo skriptai naudoja duomenis, kurie yra tiesiogiai įkoduoti į skriptą, todėl norint pakartoti testą su kitais duomenimis, reikia pakeisti ir patį skriptą, todėl metodika palengvina darbą tada, kai testavimo duomenis reikia nuolat keisti [8].

Tinkamas atvėjis, kai gali būti taikomas "Data Driven" testavimas, yra kai dviems ar daugiau testavimo scenarijų galioja tos pačios instrukcijos, bet reikalingi skirtingi įvesties parametrai ir skirtingi tikėtini rezultatai [8].

3.4.1. Parametrizavimas

Parametrizavimas yra šios testavimo technikos dalis, skirta gauti duomenis iš išorinio šaltinio ir naudoti juos kaip įvesties parametrus testo programiniame kode [8].

3.4.2. Testavimo duomenys

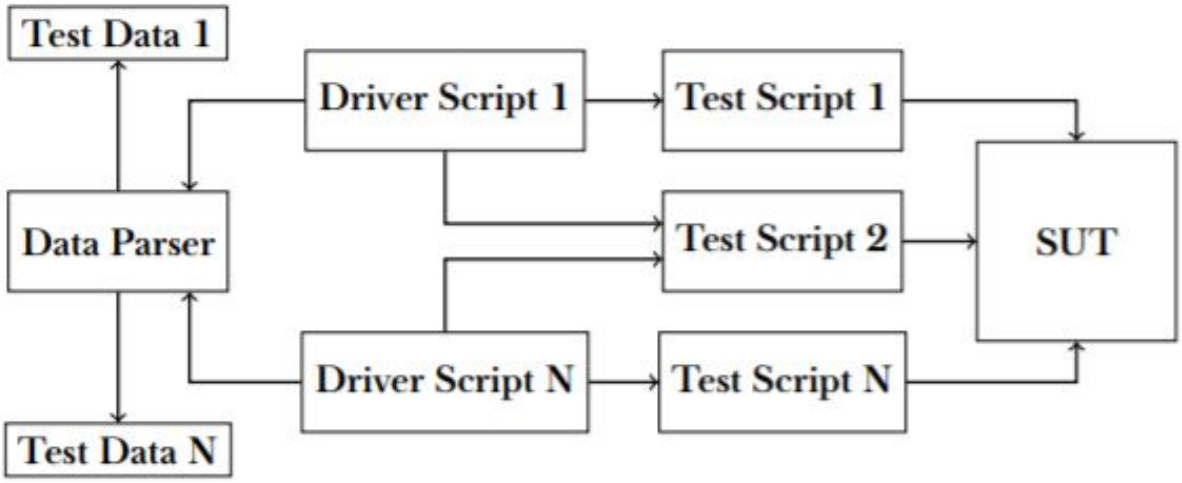
Reikia pratestuoti tinklalapio registracijos ir prisijungimo funkcionalumą, pakartoti testą naudojant skirtingus testavimo duomenis, kurie pavaizduoti žemiau.

Vardas	Pavardė	Diena	Mėnesis	Gimimo metai	El. paštas	Slaptažodis
Jonas	Jonaitis	1	8	1990	jonas@gmail.com	password
Lukas	Lukaitis	2	9	1991	lukas@gmail.com	password
Adomas	Adomaitis	3	10	1992	adomas@gmail.com	password

1 lentelė. Testavimo duomenų formato pavyzdys.

Žemiau pavaizduotoje schemoje egzistuoja du elementai (Žiūrėti 3 Fig.), kurie yra būtini "Data Driven" testavimui:

- **"Data Parser"**. Jis yra atsakingas už duomenų nuskaitymą iš CSV failo, duomenų bazės lentelės ar kitų šaltinių. Jo pagrindinė užduotis yra nuskaityti testams skirtus duomenis ir perduoti juos testavimo skriptui, suteikiant prieigą prie visų šaltinio duomenų.
- **"Driver Script"**. Ji yra atsakinga už bandymų vykdymo proceso naudojimą, naudojant bandymų bibliotekų teikiamas testavimo funkcijas, kurios sąveikauja su bandoma sistema ir bandymų duomenimis, kuriuos skaito duomenų analizatorius.



2 pav. "Data Driven" testavimo vykdymo schema.

3.4.3. Duomenų nuskaitymas iš failo

Tam, kad atlikti „Data Driven“ testus, reikia nuskaityti duomenis iš duomenų šaltinio, tai mūsų atvėju yra Excel failas.

4 išeities kodas. Duomenų nuskaitymas iš Excel failo.

```
public Object[][] fetchExcelData() throws IOException {
   inputStream = getFileInputStram();
   XSSFWorkbook excelFile = new XSSFWorkbook(inputStream);
   XSSFSheet sheet = excelFile.getSheetAt(0);
   int totalNumberOfRows = sheet.getLastRowNum() + 1;
   int totalNumberOfColumns = 7;
   String[][] arrayOfExcelData = new
      String[totalNumberOfRows][totalNumberOfColumns];
   for (int i = 0; i < totalNumberOfRows; i++) {
      for (int j = 0; j < totalNumberOfColumns; j++) {
         XSSFRow row = sheet.getRow(i);
         arrayOfExcelData[i][j] = row.getCell(j).toString();
      }
   }
   excelFile.close();
   return arrayOfExcelData;
}
```

3.4.4. Testavimo scenarijaus įgyvenimas

Vėliau iš šaltinio paimti duomenys yra naudojami atliekant testą.

5 išeities kodas. Testo vykdymas ir parametrizacijos procesas naudojant duomenis iš Excel failo.

```
@Test(dataProvider = "ExcelData")
public void registrationAndLogin(String fName, String lName,
   String day, String month, String year,
String email, String password) throws InterruptedException {
   homePage = new HomePage(driver);
   userRegister = new RegistrationPage(driver);
   homePage.openRegisterationPage();
   userRegister.userRegisteration(fName, lName, day, month,
      year, email, password);
   Assert.assertTrue(userRegister.succesfulNotification
   .getText().contains("Your registration completed"));
   userRegister.userLogOut();
   userLogInObject = new LoginPage(driver);
   homePage.openLogInPage();
   userLogInObject.userLogIn(email, password);
   Assert.assertTrue(userRegister.logoutLink.getText()
   .contains("Log out"));
   userRegister.userLogOut();
}
```

Ši testavimo metodika yra geriausias ir patikimiausias būdas patikrinti, kaip aplikacija elgiasi su skirtingais duomenimis, tai tikrinti kiekviename, net ir ne WEB projekte, yra būtina. Todėl testavimo procesas niekaip negali egzistuoti be šios metodikos.

3.5. „Keyword Driven“ testavimas

Ši automatinio testavimo atšaka yra loginis praplėtimas "Data driven" testavimo metodiką. Neskaitant testavimo duomenų, ji naudoja raktažodžius, susijusius su testuojama aplikacija. Šie raktažodžiai apibūdina veiksmų rinkinį, reikalingų atlikti tam tikrą žingsnį testavimo scenarijuje - skripte. Pirmiausia yra sukuriamas raktinių žodžių rinkinys, siejamas su tam tikru veiksmu teste. Kiekvienas bandomasis veiksmas kaip pvz. naršyklės atidarymas, naršyklės uždarymas, pelės paspaudimas yra aprašytas pasirinktu raktiniu žodžiu [1].

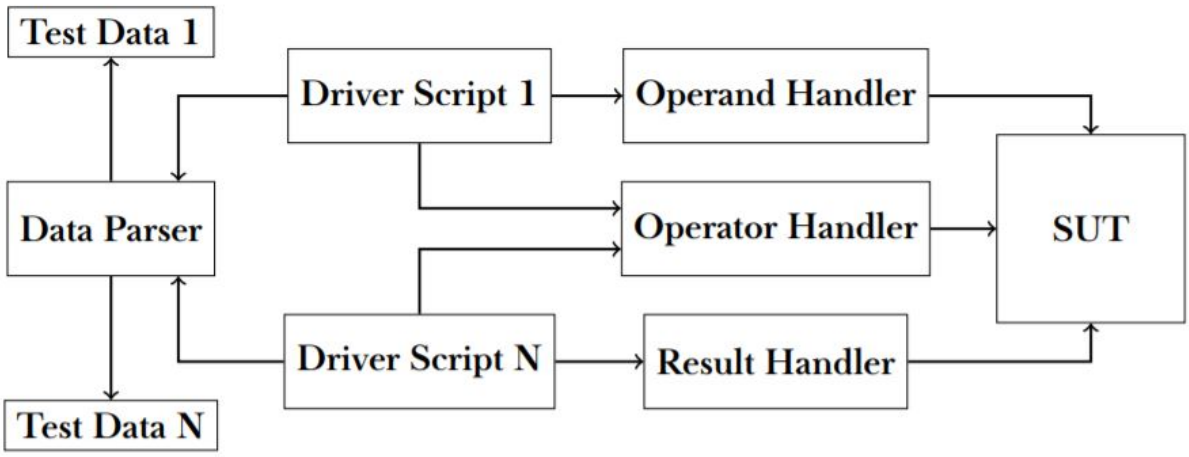
3.5.1. "Keyword driven" testo duomenys

"Keyword driven" testavime yra keli elementai (Žiūrėti 4 Fig.), kurie privalo būti įtraukti:

1. **"Data Parser"**. Duomenų nuskaitymas iš šaltinių.
2. **"Driver Script"**. Apdoroja raktažodžius ir duomenis, kurie reikalingi testo vykdymui.
3. **"Handlers"**. Kiekvienas raktažodis atitinka vieną atskirą test. skripto fragmentą, jų seka (raktažodžių ir reikšmių) paverčia juos į vykdomus testus.

TSID	Description	Keyword	WebElement	ProceedOnFail	TestDataField
TS01	Navigate to website	Navigate	http://website.com/		
TS02	Click on SIGN IN	ClickOnLink	login.signIn	Y	
TS03	Enter Email Address	InputText	login.createAccEmailAddr	Y	email
TS04	Click on Create An Account	ClickOnLink	login.createAnAccount	Y	
TS05	Waiting For Registration page	expliciteWait	registration.mrRadionButton	Y	
TS06	Select mr radio button	selectRadioButton	registration.mrRadionButton	Y	
TS07	Enter First Name	InputText	registration.firstName	Y	firstname
TS08	Enter Last Name	InputText	registration.lastname	Y	lastname
TS09	Enter Password	InputText	registration.password	Y	password
TS10	Select Days In Drop Down	selectDaysInDropDown		Y	day

3 pav. Ištrauka iš "Keyword Driven" testavimo scenarijaus.



4 pav. "Keyword Driven" testavimo vykdymo schema.

3.6. „Behavior Driven“ testavimas

Tai yra programinės įrangos plėtojimo procesas, kurio metu yra sudaromi scenarijai, kaip aplikacija turi elgtis žiūrint iš galutinio vartotojo perspektyvos. Šio testavimo tikslas - pagerinti suinteresuotųjų šalių, tokių kaip kūrėjai, testuotojai, produktų vadovai ir verslo analitikai, bendradarbiavimą, rašant testavimo scenarijus lengvai suprantama Gherkin kalba. Naudojant šią metodiką yra parašomas testavimo skriptas, fiksuojant kiekvieną testavimo žingsnį ir jį išvedant į ekraną. Šią testavimo metodiką galima įgyvendinti naudojant Cucumber, SpecFlow karkasus arba jų alternatyvas [9].

3.6.1. Testavimo scenarijus parašytas su Gherkin

6 išeities kodas. Gherkin kalba parašytas testas.

```
Scenario: Checkout a tea type
  Given I open the tea Categories Page
  When I click on the Red Tea checkout link
  And I enter name "Adomas" in the Name field
  And I enter address "Kovo g." in the Address field
  And I enter the email "adomas@gmail.com" in the email field.
  And I click submit Button
  Then I should successfully navigate to Menu page
```

Žemiau yra aprašomas testavimo scenarijus Gherkin kalba, kuri yra lengvai suprantama visiems, net ir tiem kurie neturi programavimo žinių, tačiau kiekvienas šis scenarijaus žingnis yra suprogramuotas, naudojant Selenium karkasą, taip, kaip nurodyta žemiau.

7 išeities kodas. Gherkin testas įgyvendintas naudojant Selenium karkasą ir Java kalbą.

```
public class TeaOrder {
    ...
    @Given("^I open the tea Categories Page$")
    public void i_open_the_tea_Categories_Page(){
       hp = CheckoutUtil.openHomePage();
       tcp = hp.clickLink_Menu();
    }
    @When("^I click on the Red Tea checkout link$")
    public void i_click_on_the_Red_Tea_checkout_link(){
       cop = tcp.clickBtn_RedTea();
    }
    @When("^I enter name \"([^\"]*)\" in the Name field$")
    public void i_enter_name_in_the_Name_field(String name){
       cop.type_Txt_name(name);
    }
    @When("^I enter address \"([^\"]*)\" in the Address field$")
    public void i_enter_address_in_the_Address_field(String
       address){
       cop.type_Txt_address(address);
    }
    @When("^I enter the email \"([^\"]*)\" in the email field\\.$")
    public void i_enter_the_email_in_the_email_field(String email){
       cop.type_Txt_email(email);
    }
    @When("^I click submit Button$")
    public void i_click_submit_Button(){
       tcp = cop.click_Btn_Submit();
    }
    @Then("^I should successfully navigate to Menu page$")
    public void i_should_successfully_navigate_to_Menu_page(){
       Assert.assertEquals("Menu", tcp.getDriver().getTitle());
    }
    ...
    }
```

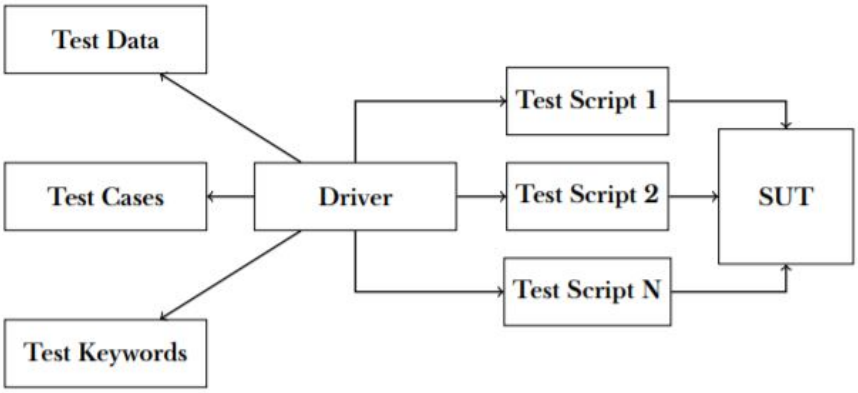
Kaip jau minėta, ši metodika palengvina komunikaciją tarp specialistų, dirbančių prie projektą, kitaip tariant - padeda "susikalbėti". "Susikalbėjimas" ir vienas kito supratimas WEB projektuose yra labai svarbus, kitaip gali būti patiriami nuostoliai, kurių

3.7. „Hybrid Driven“ testavimas

Ši testavimo metodika yra praktiškiausia ir dažniausiai sutinkama praktikoje. Ji naudoja kelias skirtingas automatinio testavimo savybes ir todėl yra vadinama "Hybrid Driven" metodika. Plačiausiai naudojama kombinacija yra testavimo projekte panaudojant "Data Driven" ir "Keyword Driven" testavimo metodikų savybes, taip kaip nurodyta schemoje apačioje (Žiūrėti 5 Fig.). Naudojant šią testavimo metodiką, kūrėjai ir testuotojai gali kompensuoti vieno automatinio testavimo tipo naudojimo trūkumus. Šis metodas yra labai tinkamas plačios apimties sistemų testavimui, kuriose yra daug duomenų rinkinių ir duomenų perkėlimo atvejų [2].

"Hybrid driven" testavimas naudojamas tam, kad išpildyti tokias charakteristikas kaip:

1. Priežiūros paprastumas (**angl.** Maintainability)
2. Pakartotinio panaudojamumas (**angl.** Reusability)
3. Patikimumas



5 pav. "Hybrid Driven" testavimo vykdymo schema.

Ši testavimo metodika yra pranaši tuo, kad naudojant ją testuose yra panaudojami kitų testavimo metodikų, kaip data-driven, keyword-driven ir t.t, pranašumai. Šios metodikos naudojimas žymiai sumažina riziką palikti defektų WEB ar kitokio tipo projekte,

4. Testavimo metodikų palyginimas

Metodika	**Privalumai**	**Trūkumai**
Data driven	Lengva palaikyti, vykdyti testus naudojant skirtingus duomenis.	Norint įgyvendinti šią metodiką, reikia kvalifikuotų išteklių.
Keyword driven	Testavimo scenarijai yra suprantami ir lengvai modifikuojami, aukštas pakartotinio panaudojamumo lygis, apibrėžus raktažodžius, sudarinėti test. scenarijus gali beveik bet kas.	Raktinių žodžių ir susijusių funkcijų kūrimas yra daug laiko reikalaujantis procesas.
Hybrid driven	Apjungia visus automatinio testavimo metodų privalumus į vieną metodą.	Sunkiai įgyvendinamas testavimo metodas, nes reikia išmanyti visas aut. testavimo metodikas.
Modularity driven	Sumažina kodo duplikaciją, palaiko švarų kodą, testai yra lengvai plečiami.	Yra mažiau veiksmingas nei kitos metodikos kaip "Keyword driven" ir "Data driven".
Behaviour driven	Palengvina bendradarbiavimą tarp visų prie produkto dirbančių specialistų, padeda susikoncentruoti ties reikalavimais.	
Linear scripting	Greitas testų generavimas, visiškai nereikalinga testų automatiacijos patirtis.	Testavimo duomenys įkoduoti į skriptą, žemas pakartotinio panaudojamumo lygis.

2 lentelė. Testavimo metodikų palyginimas.

Išvados ir rekomendacijos

Tinkamo testavimo karkaso pasirinkimas priklauso nuo projektų, komandos patirties ir turimų išteklių ir turi didelę įtaką internetinių aplikacijų kūrimo ciklo sklandumui bei galutinio produkto kokybei, todėl ši internetinių puslapių kūrimo ciklo dalis yra gyvybiškai svarbi ir negali būti praleista. Pradedant projektą svarbu nusistatyti, į ką WEB aplikacija bus orientuota - ar tai bus aplikacija, kuri valdys didelius įvairių duomenų kiekius, ar aplikacija, kuri turės daug smulkių funkcijų, kurios yra priklausomos vienos nuo kitos ir pagal tai pasirinkti data-driven, behavior-driven ar kitokią automatinio testavimo metodiką, tačiau visgi rekomenduotina automatiniam testavimui naudoti hybrid-driven automatinio testavimo metodologiją, jungiančią daugelio metodikų privalumus į atskirą metodiką, padėsiančią užtikrinti aplikacijos kokybę vertinant ją skirtingais aspektais, kuriuos atstovauja kiekviena iš metodikų.

Tam, kad automatizacijos procesą palaikyti veiksmingą, tiek testuotojai, tiek produkto šeimininkai, vadybininkai, klientai turi norėti ir turėti galimybę atlikti pakeitimus produkte, jei to reikalauja susiklosčiusi situacija.

Literatūros šaltiniai

[1] Rashmi and Neha Bajpai. A keyword driven framework for testing web applications. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 3:pp. 8–14, 03 2012.

[2] Muhammad Abid Jamil, Muhammad Arif, Normi Sham Awang Abubakar, and Akhlaq Ahmad. Software testing techniques: A literature review. pages 177–182, 11 2016.

[3] Aaron M French. Web development life cycle: a new methodology for developing web applications. *The Journal of Internet Banking and Commerce*, 16(2):1–11, 1970.

[4] Dorota Huizinga and Adam Kolawa. *Automated defect prevention: best practices in software management*. John Wiley & Sons, 2007.

[5] Juha Itkonen, Mika V Mantyla, and Casper Lassenius. Defect detection efficiency: Test case based vs. exploratory testing. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 61–70. IEEE, 2007.

[6] Michael Kelly. Choosing a test automation framework. *IBM DeveloperWorks*, 18, 2003.

[7] Gerard Meszaros. Agile regression testing using record & playback. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 353–360. ACM, 2003.

[8] Oracle. Functional testing openscript user's guide. pages 9–10, 2019.

[9] John Ferguson Smart. *BDD in Action*. Manning Publications, 2014.

[10] Ghazia Zaineb, Dr Irfan, and Irfan Manarvi. Identification and analysis of causes for software bug rejection with their impact over testing efficiency. *International Journal of Software Engineering Applications*, 2, 10 2011.