UNIVERZITET U BEOGRADU

FAKULTET ORGANIZACIONIH NAUKA

KATEDRA ZA SOFTVERSKO INŽENJERSTVO

Predmet: Testiranje i kvalitet softvera

Proces testiranja metoda, funkcija, procedura, pogleda, i trigera korišćenjem unit testova i alata za testiranje TSQL-a

Mentor:
dr Saša D. Lazarević

Studenti:
Neda Stošić 2019/3706
Slobodanka Tešmanović 2019/3711

Beograd, 2020. Godine

Popis slika

Popis tabela

Sadržaj

1. Uvod

U kontekstu softverskog inženjerstva, kvalitet softvera odnosi se na dva povezana, ali različita pojma pojma:

Funkcionalni kvalitet softvera koji odražava koliko je softver u skladu sa zahtevanim dizajnom, odnosno u kojoj meri zadovoljava funkcionalne zahteve definisane softverskom specifikacijom

Strukturni kvalitet softvera koji se odnosi na to u kojoj meri softver ispunjava nefunkcionalne zahteve, odnosno one zahteve koji podržavaju ispunjenje funkcionalnih zahteva

Testiranje softvera igra važnu ulogu u proceni, postizanju i održavanju kvaliteta softverskog sistema. S jedne strane, kroz ponavljanje ciklusa *testiraj – pronađi nedostatke – ispravi nedostatke* poboljšavamo kvalitet softvera, dok sa druge strane testiranjem na nivou sistema procenjujemo koliko je softver dobar pre puštanja istog u rad.

U okviru ovog rada izvšeno je testiranje softverskog sistema za vođenje evidencije o iznajmljivanju ski pass-eva. Softverski sistem je razvijen kao Windows Forms aplikacija u *C#* programskom jeziku, dok je kao sistem za upravljanje bazom podataka korišćen *SQL Server Menagement Studio*, odnosno *SQL Server* relaciona baza podataka i *Transact SQL* za pisanje uskladištenih procedura, funkcija itd.

U narednom poglavlju detaljno je opisan studijski primer, odnosno softverski sistem koji je razvijen i koji je predmet testiranja. U nastavku je dat prikaz i poređenje tri alata koji su korišćeni za testiranje kôda pisanog u TSQL-u, a to su tSQLt, T.S.T. i tSQLUnit. Dalje je opisan postupak unit testiranja kôda pisanog u C#-u, kao i kratak osvrt na testiranjem vođen razvoj. Na kraju je dat zaključak kao i spisak literature korišćene tokom pisanja ovog rada.

2. Studijski primer

2.1 Verbalni model

Sistem treba da vodi evidenciju o korisnicima i ski pass-evima koje oni iznajmljuju. Svaki ski pass vezan je za jedan paket koji može sadržati više regiona. Region predstavlja skup staza koje se u njemu nalaze. Svaka staza može pripadati samo jednom regionu. Korisnik može više puta da iznajmi različite ski pass-eve.

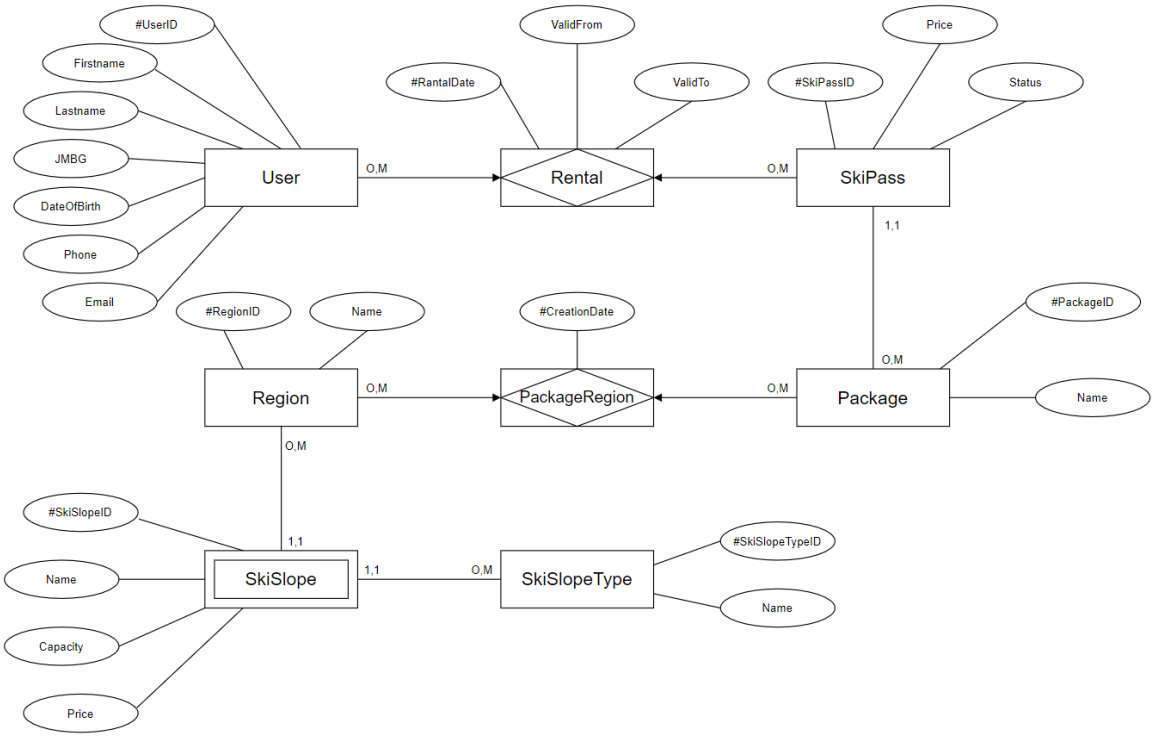
Sistem treba da omogući unos korisnika, ski pass-a, paketa i iznajmljivanja ski pass-a od strane korisnika. Pored toga, potrebno je omogućiti izmenu korisnika i paketa.

2.2 Prošireni model entiteta i relacija, relacioni model i rečnik podataka

2.2.1 Prošireni model entiteta i relacija

U proširenom modelu entiteta i relacija podaci se modeluju u skupovima entiteta, gde je entitet podatak koji postoji nezavisno od drugih entiteta u bazi podataka. Entiteti su nalik na slogove, imaju atribute i među njima mogu da postoje odnosi. Odnosi su označeni kardinalnošću i takođe mogu da imaju atribute.

Na sledećoj slici prikazan je prošireni model entiteta i relacija koji je korišćen:



Slika 1 : Prošireni model entiteta i relacija

2.2.2 Relacioni model

Relacioni model ispisuje logičke aspekte podataka, odnosno relacije koje postoje među različitim podacima. Relacioni model koji je izrađen na osnovu prethodno definisanog modela entiteta i relacija dat je u nastavku:

User (UserID, Firstname, Lastname, JMBG, DateOfBirth, Phone, Email)

SkiSlopeType (SkiSlopeTypeID, Name)

Region (RegionID, Name)

SkiSlope (SkiSlopeID, Name, Capacity, Price, *SkiSlopeTypeID*, *RegionID*)

Package (PackageID, Name)

PackageRegion (CreationDate, *PackageID*, *RegionID*)

SkiPass (SkiPassID, Price, Status, *PackageID*)

Rental (RentalDate, *UserID*, *SkiPassID*, ValidFrom, ValidTo)

2.2.3 Rečnik podataka

U sledećim tabelama prikazan je rečnik podataka:

Tabela User		**Prosto vrednosno ograničenje**		**Složeno vrednosno ograničenje**		**Strukturno ograničenje**
Atributi	**Ime**	**Tip atributa**	**Vrednost atributa**	**Međuzav. atributa jedne tabele**	**Međuzav. atributa više tabela**	UPDATE CASCADE Rental DELETE RESTRICT Rental
	UserID	bigint	not null and >0			
	Firstname	varchar(50)	not null			
	Lastname	varchar(50)	not null			
	JMBG	varchar(13)	not null			
	DateOfBirth	date	not null			
	Phone	varchar(50)	not null			
	Email	varchar(50)				

Tabela 1 : Tabela User

Tabela SkiSlopeType		**Prosto vrednosno ograničenje**		**Složeno vrednosno ograničenje**		**Strukturno ograničenje**
Atributi	**Ime**	**Tip atributa**	**Vrednost atributa**	**Međuzav. atributa jedne tabele**	**Međuzav. atributa više tabela**	UPDATE CASCADE SkiSlope DELETE RESTRICT SkiSlope
	SkiSlopeTypeID	bigint	not null and >0			
	Name	varchar(50)	not null			

Tabela 2 : Tabela SkiSlopeType

Tabela Region		Prosto vrednosno ograničenje		Složeno vrednosno ograničenje		Strukturno ograničenje
Atributi	**Ime**	**Tip atributa**	**Vrednost atributa**	**Međuzav. atributa jedne tabele**	**Međuzav. atributa više tabela**	UPDATE CASCADE PackageRegion DELETE RESTRICT PackageRegion
	RegionID	bigint	not null and >0			
	Name	varchar(50)	not null			

Tabela 3 : Tabela Region

Tabela SkiSlope		Prosto vrednosno ograničenje		Složeno vrednosno ograničenje		Strukturno ograničenje
Atributi	**Ime**	**Tip atributa**	**Vrednost atributa**	**Međuzav. atributa jedne tabele**	**Međuzav. atributa više tabela**	UPDATE RESTRICT Region, SkiSlopeType DELETE /
	SkiSlopeID	bigint	not null and >0			
	Name	varchar(50)	not null			
	Capacity	int	not null, >0			
	Price	float	not null, >0			
	SkiSlopeTypeID	bigint	not null			
	RegionID	bigint	not null			

Tabela 4 : Tabela SkiSlope

Tabela Package		Prosto vrednosno ograničenje		Složeno vrednosno ograničenje		Strukturno ograničenje
Atributi	**Ime**	**Tip atributa**	**Vrednost atributa**	**Međuzav. atributa jedne tabele**	**Međuzav. atributa više tabela**	UPDATE CASCADE PackageRegion DELETE RESTRICT PackageRegion
	PackageID	bigint	not null and >0			
	Name	varchar(50)	not null			

Tabela 5 : Tabela Package

Tabela SkiPass		Prosto vrednosno ograničenje		Složeno vrednosno ograničenje		Strukturno ograničenje
Atributi	**Ime**	**Tip atributa**	**Vrednost atributa**	**Međuzav. atributa jedne tabele**	**Međuzav. atributa više tabela**	UPDATE CASCADE Rental RESTRICT Package DELETE RESTRICT Rental
	SkiPassID	bigint	not null and >0			
	Price	varchar(50)	not null			
	Status	int	not null, >0			
	Price	float	not null, >0			
	PackageID	bigint	not null			

Tabela 6 : Tabela SkiPass

Tabela Rental		Prosto vrednosno ograničenje		Složeno vrednosno ograničenje		Strukturno ograničenje
Atributi	**Ime**	**Tip atributa**	**Vrednost atributa**	**Međuzav. atributa jedne tabele**	**Međuzav. atributa više tabela**	UPDATE RESTRICT User, SkiPass DELETE /
	RentalDate	datetime	not null and >0			
	UserID	bigint	not null			
	SkiPassID	bigint	not null, >0			
	ValidFrom	datetime	not null, >0			
	ValidTo	datetime	not null			

Tabela 7 : Tabela Rental

2.3 Specifikacija vrednosnih ograničenja

Prilikom definisanja proširenog modela entiteta i relacija (PMER) uvodi se i koncept kardinalnosti preslikavanja u relaciji između dva entiteta, koji se sastoji iz donje i gornje granice i on predstavlja najvažniji deo strukturnih ograničenja. Pored strukturnih, postoje i složenija semantička ograničenja koja nije moguće prikazati strukturom PMER. Ta ograničenja najčešće su definisana nad vrednostima atributa, te se nazivaju vrednosnim ograničenjima. Za njihovo specifikovanje koristi se jezik koji se naziva račun entiteta (RE).

2.3.1 Ograničenja nad domenom

- Domen **DName** je tipa *string* od najviše 50 znakova, a prvi znak mora da bude veliko slovo abecede.

  DOMAIN **DName** :=

  String(50),

  LIKE '[A-Ž]%';

- Domen **DPrice** je tipa integer, i ne sme ima vrednost manju od nule

  DOMAIN **DPrice**:=

  int,

  >0;

2.3.2 Ograničenja na vrednosti atributa

2.3.2.1 Ograničenja na vrednost jednog atributa

- Atribut DateOfBirth entiteta User je datum koji ne sme da bude u budućnosti

  CONSTRAINT DateOfBirth:=

  FOR EACH (Us IN USER)

  Us.DateOfBirth <= Datetime.Now;

- Atribut Phone entiteta User je broj mobilnog telefona i mora da zadovolji odredjeni format

  CONSTRAINT Phone:=

  FOR EACH (Us IN USER)

  Us.Phone LIKE ‘^(\+381)?(\s|)?6(([0-6]|[8-9])\d{8}|(77|78)\d{7}){1}$’;

- Atribut Capacity entiteta SkiSlope ne sme da ima vrednost manju od 0

  CONSTRAINT Capacity:=

  FOR EACH (Ss IN SKISLOPE)

  Ss.Cpacity > 0;

- Atribut DateFrom entiteta Rental je datum koji ne sme biti posle atributa datuma DateValid istog entiteta

  CONSTRAINT DateFrom:=

  FOR EACH (Re IN RENTAL)

  Re.DateFrom < Re.DateTo;

- Atribut RentalDate entiteta Rental je datum koji ne sme da bude u budućnosti

  CONSTRAINT RentalDate:=

  FOR EACH (Re IN RENTAL)

  Re.RentalDate <= Datetime.Now;

2.3.2.2 Međuzavisnost vrednosti dva ili više atributa istog entiteta

- Vrednost atributa Price entiteta SkiPass jednaka je sumi cena svih ski staza koje se nalaze u regionima koje sadrži paket koji je iznajmljen uz određeni ski pass

```
CONSTRAINT SkiPassPrice :=
    FOR EACH (Sp IN SKIPASS)
        St.Price =
        SUM(Sp.PACKAGE.PACKAGEREGION.REGION.SKISLOPE.Price);
```

2.3.3 Ograničenja prelaza iz stanja u stanje

- Ukoliko korisnik iznajmi isti ski pass više puta u toku jedne godine, svaki put (sem prvog) dobija popust od 5% na regularnu cenu.

Napomena:

(1) Operacija UPDATE je makro operacija:

UPDATE = DELETE Old + INSERT New;

Zato uvodimo dva stanja svakog entiteta:

Old = vrednosti atributa razmatranog entiteta PRE prelaza u novo stanje, tj. pre DELETE Old

New = vrednosti atributa razmatranog entiteta POSLE prelaza u novo stanje, tj. posle INSERT New

```
CONSTRAINT KorisnikPopust :=
    FOR EACH (Us IN USER)
        Us.New.Price <= 0.95 * Us.Old.Price;
```

3. Alati za testiranje TSQL kôda

SQL unit testiranje je proces testiranja koji nam omogućava da testiramo atomski programabilni deo baze podataka.Ovo testiranje ima ključnu ulogu u savremenom ciklusu razvoja baze podataka jer omogućava testiranje pojedinih delova baze podataka kako bi se uvidelo da li oni rade onako kako bi trebalo. SQL unit testiranje dodaje veliku vrednost projektu baze podataka jer su ovi testovi daleko pouzdaniji od manuelnih metoda testiranja.

SQL unit testiranje povećava pouzdanost kôda, jer programer otklanja nedostatke i uočene greške u početnim fazama razvoja. Ključna stvar je u tome što je otklanjanje grešaka u fazi razvoja daleko jeftinije nego što je to u kasnijim fazama.

U okviru ovog rada korišćena su tri alata za testiranje kôda pisanog u TSQL-u:

- tSQLt
- T.S.T.
- tSQLUnit

3.1 tSQLt

TSQLt je besplatni open source framework za testiranje kôda pisanog u TSQL-u. Ovaj alat je veoma jednostavan za instalaciju i upotrebu. Kompatibilan je sa svim verzijama SQL Server-a od verzije *2005 Service Pack 2* pa na dalje.

Karakteristike ovog alata su:

- Testovi se automatski pokreću u okviru transakcija - ovo obezbeđuje da testovi budu nezavisni i smanjuje svaki posao „čišćenja” podataka nakon završenog testa
- Testovi se mogu grupisati u šeme - što omogućava organizaciju testova prema sopstvenim potrebama
- Izlaz (output) se može generisati u običnom tekstu ili XML-u
- Lažne (fake) tabele i pogledi – omogućavaju izolaciju kôda koji se testira

Benefiti pri korišćenju ovog alata su sledeći:

- Omogućava upotrebu TSQL kôda prilikom pisanja unit testova što znači da nije potrebno učenje novog programskog jezika ili platforme da bi se kreirao i pokrenuo TSQL unit test

- Potpuno je besplatan za instalaciju i korišćenje

- Može se lako integrisati u SSDT projekte ili softver nezavisnih proizvođača

Kako bi se počelo sa pisanjem testova, neophodno je da se napravi tzv. test klasa. Unutar te klase prave se uskladištene procedure čije ime treba da započne rečju *test* i one predstavljaju testove. Izvršavanje testova može se pokrenuti na dva načina:

- tSQLt.RunTestClass – tSQLt tada traži sve uskladištene procedure čiji naziv počinje sa *test* unutar navedene test klase i pokreće njihovo izvršavanje

- tSQLt.RunAll – tSQLt traži sve šeme u bazi podataka koje su označene kao test klase i zatim za svaku od njih izvršava naredbu tSQLt.RunTestClass

Ovaj alat sadrži više predefinisanih uskladištenih procedura koje služe za proveravanje tačnosti (assert procedure), to su:

- AssertEmptyTable
- AssertEquals
- AssertEqualsString
- AssertEqualsTable
- AssertEqualsTableSchema
- AssertNotEquals
- AssertObjectDoesNotExist
- AssertObjectExists
- AssertResultSetsHaveSameMetaData
- Fail
- AssertLike

Za potrebe testiranje određenog kôda za koji se (ne) očekuje da baci izuzetak obezbeđene su dve uskladištene procedure:

- ExpectException
- ExpectNoException

Takođe, postoje mehanizmi za izolaciju zavisnosti. Ova grupa procedura omogućava izolaciju kôda koji se testira:

- FakeTable – omogućava pisanje testova nezavisno od ograničenja koji postoje nad tabelom. FakeTable kreira praznu verziju tabele bez ograničenja. Korisno za situacije kada želimo da uradimo insert u tabelu ali ne želimo da ubacujemo podatke koji su irelevantni za određeni test.
- ApplyConstraint – u slučaju kada se koristi FakeTable omogućava aktiviranje određenih ograničenja u skladu sa potrebama testa
- ApplyTrigger – funkcioniše na isti način kao prethodna, samo u slučaju trigera
- FakeFunction – kôd koji poziva funkciju koja u sebi sadrži složenu logiku može biti komplikovan za testiranje. Da bi se kreirali nezavisni testovi, pozvana funkcija može da se zameni sa lažnom (fake) funkcijom koja će izvoditi mnogo jednostavniju logiku koja služi svrsi testa
- RemoveObject – služi za zamenu postojećeg objekta mock-om. Prvi korak jeste uklanjanje postojećeg objekta preimenovanjem. Novo ime objekta se automatski generiše kako ne bi došlo do kolizije sa već postojećim objektima.
- RemoveObjectIfExists – predstavlja uprošćenu verziju RemoveObject uz uslov @IfExist = 1
- SpyProcedure – omogućava pisanje testova za procedure izolojući druge procedure koje se unutar nje pozivaju

Uz alat se dobija baza podataka sa nekoliko testova koji mogu da posluže kao primer pri početku korišćenja alata.

3.1.1 Primena alata tSQLt

U okviru ovog rada kreirani su sledeći testovi korišćenjem alata tSQLt:

- `[SkiPassTests].[test insert rental - date valid to must be after date valid from]` - testira proceduru za unos iznajmljivanja koja ne bi trebalo da dozvoli unos iznajmljivanja kom je datum od kada traje (DateFrom) posle datuma do kada traje (DateTo). U okviru test korišćenja je FakeTable procedura kako ne bismo morali da vodimo računa o stranim ključevima. U ovom testu korišćena je procedura `ExpectException` kojom se proverava da li je procedura koja se testira bacila očekivanu grešku.

Procedura InsertRental data je u nastavku:

```
ALTER PROCEDURE [dbo].[SkiPass.InsertRental]
       @RentalDate                  DATETIME,
       @UserID                                    BIGINT,
       @SkiPassID                  BIGINT,
       @ValidFrom                                 DATETIME,
          @ValidTo                                DATETIME
AS
BEGIN
     SET NOCOUNT ON


        IF @ValidTo < @ValidFrom
        BEGIN
        RAISERROR ('Date "Valid to" must be after date "Valid from"!',16,1)
        RETURN
        END


     INSERT INTO dbo.Rental
          (
            RentalDate,
            UserID,
            SkiPassID,
            ValidFrom,
                      ValidTo
          )
     VALUES
          (
           @RentalDate,
            @UserID,
            @SkiPassID,
            @ValidFrom,
                      @ValidTo
          )


       END
```

U nastavku je test koji je napisan za nju pomo'u alata tSQLt:

```
ALTER PROCEDURE [SkiPassTests].[test insert rental - date valid to must be after date valid
from]
AS
BEGIN
     SET NOCOUNT ON

        EXEC tSQLt.FakeTable 'dbo.Rental';

        EXEC tSQLt.ExpectException @Message = 'Date "Valid to" must be after date "Valid
from"!';

        EXEC dbo.[SkiPass.InsertRental] @RentalDate = '2020-01-15', @UserID = null, @SkiPassID
= null, @ValidFrom = '2020-01-16', @ValidTo = '2020-01-14'


       END
```

- `[SkiPassTests].[test insert ski pass - price must be greater than 0]` – testira proceduru za unos ski pass-a koji ne bi trebalo da dozvoli upis cene koja ima negativnu vrednost.

- `[SkiPassTests].[test InsertPackageRegion - foreign key constraints are violated]` - testira proceduru za upis u agregaciju PackageRegion koja ne bi trebalo da dozvoli upis PackageID-a koji ne postoji u tabeli Package. Drugim rečima, ovde se testira slučaj gde je narušeno ograničenje za spoljni ključ tabele PackageRegion. U ovom testu korišćena je FakeTable procedura ali i procedura ApplyConstraint, s obzirom da je svrha testa provera ograničenja. Provera se vrši kroz hvatanje greške, odnosno očekivano ponašanje procedure jeste da baci grešku sa određenom porukom ukoliko se pokuša upis vrednosti koja ne zadovoljava ograničenje.

Test je dat u nastavku:

```
ALTER PROCEDURE [SkiPassTests].[test InsertPackageRegion - foreign key constraints are
violated]
AS
BEGIN

  EXEC tSQLt.FakeTable 'dbo.Package';
  EXEC tSQLt.FakeTable 'dbo.Region';
  EXEC tSQLt.FakeTable 'dbo.PackageRegion';
  EXEC tSQLt.ApplyConstraint 'dbo.PackageRegion', 'FK_PackageRegion_Package';
  EXEC tSQLt.ApplyConstraint 'dbo.PackageRegion', 'FK_PackageRegion_Region';

  DECLARE @err NVARCHAR(MAX); SET @err = '<No Exception Thrown!>';
  BEGIN TRY
    INSERT INTO dbo.PackageRegion(PackageID) VALUES (5);
  END TRY
  BEGIN CATCH
    SET @err = ERROR_MESSAGE();
  END CATCH

  IF (@err NOT LIKE '%FK_PackageRegion_Package%')
  BEGIN
    EXEC tSQLt.Fail 'Expected exception (FK_PackageRegion_Package) not thrown. Instead:',@err;
  END;

  BEGIN TRY
    INSERT INTO dbo.PackageRegion(RegionID) VALUES (5);
  END TRY
  BEGIN CATCH
    SET @err = ERROR_MESSAGE();
  END CATCH

  IF (@err NOT LIKE '%FK_PackageRegion_Region%')
  BEGIN
    EXEC tSQLt.Fail 'Expected exception (FK_PackageRegion_Region) not thrown. Instead:',@err;
  END;


      END;
```

- `[SkiPassTests].[test InsertSkiPass - foreign key constraint FK_SkiPass_Region is violated]` – radi isto što i u prethodnom testu, samo se provera vrši za ograničenje koje se odnosi na tabelu Region.
- `[SkiPassTests].[test InsertTrail - foreign key constraints are violated]` – slično kao u prethodna dva testa, vrši se provera ograničenja nad tabelom Trail
- `[SkiPassTests].[test is JMBG valid]` – testira funkciju za proveru ispravnosti formata JMBG-a. Ovde se koristi procedura `AssertEqualsString` koja vrši poređenje poruke koju vraća funkcija sa onim što je očekivani rezultat.

U nastavku je data funkcija za proveru JMBG-a:

```
ALTER FUNCTION [dbo].[fn_JMBGValidation]
(
	@JMBG NVarchar(MAX)
)
RETURNS NVARCHAR(MAX)
AS
BEGIN

	DECLARE @ErrorMessage NVARCHAR(MAX) = '';

	IF @JMBG IS NULL OR @JMBG = ''
	SET @ErrorMessage = 'JMBG cant be null or empty. ';

	IF LEN(@ErrorMessage) > 0
	RETURN @ErrorMessage;

	IF ISNUMERIC(@JMBG)<>1
	SET @ErrorMessage = @ErrorMessage + 'JMBG can only contain numbers. ';

	IF LEN(@ErrorMessage) > 0
	RETURN @ErrorMessage;

	IF LEN(@JMBG) > 13 or LEN(@JMBG) < 13
	SET @ErrorMessage  = @ErrorMessage + 'JMBG must contain 13 digits. ';

	IF LEN(@ErrorMessage) > 0
	RETURN @ErrorMessage;


		DECLARE @Day NVARCHAR(2);
		DECLARE @Month NVARCHAR(2);
		DECLARE @Year NVARCHAR(4);
		DECLARE @Region NVARCHAR(2);
		DECLARE @OrderNum NVARCHAR(3);

		SELECT @Day = SUBSTRING(@JMBG,1,2);
		SELECT @Month = SUBSTRING(@JMBG,3,2);
		SELECT @Year = SUBSTRING(@JMBG,5,3);
		SELECT @Region = SUBSTRING(@JMBG,8,2);
		SELECT @OrderNum = SUBSTRING(@JMBG,10,3);

		IF @Year<800 SET @Year = @Year + 2000;
		ELSE SET @Year = @Year + 1000;
```

```
		IF ISDATE(@Year + @Month + @Day) <> 1
		SET @ErrorMessage = @ErrorMessage + 'Date of birth is not valid. '
		ELSE
		BEGIN
			DECLARE @DatumRodj DATE;
			SELECT @DatumRodj = CONVERT(DATE,@Year + @Month + @Day,112);
			IF @DatumRodj > GETDATE()
			BEGIN
				SET @ErrorMessage = @ErrorMessage + 'Date of birth is in the
future. ';
			END;
		END;

		DECLARE @A INT;
		DECLARE @B INT;
		DECLARE @C INT;
		DECLARE @D INT;
		DECLARE @E INT;
		DECLARE @F INT;
		DECLARE @G INT;
		DECLARE @H INT;
		DECLARE @I INT;
		DECLARE @J INT;
		DECLARE @K INT;
		DECLARE @L INT;
		DECLARE @M INT;

		SELECT @A = SUBSTRING(@JMBG,1,1);
		SELECT @B = SUBSTRING(@JMBG,2,1);
		SELECT @C = SUBSTRING(@JMBG,3,1);
		SELECT @D = SUBSTRING(@JMBG,4,1);
		SELECT @E = SUBSTRING(@JMBG,5,1);
		SELECT @F = SUBSTRING(@JMBG,6,1);
		SELECT @G = SUBSTRING(@JMBG,7,1);
		SELECT @H = SUBSTRING(@JMBG,8,1);
		SELECT @I = SUBSTRING(@JMBG,9,1);
		SELECT @J = SUBSTRING(@JMBG,10,1);
		SELECT @K = SUBSTRING(@JMBG,11,1);
		SELECT @L = SUBSTRING(@JMBG,12,1);
		SELECT @M = SUBSTRING(@JMBG,13,1);

		DECLARE @Check INT;

		SELECT @Check =  11 - (( 7*(@A+@G) + 6*(@B+@H) + 5*(@C+@I) + 4*(@D+@J) +
3*(@E+@K) + 2*(@F+@L) ) % 11);
		IF (@Check > 9) SET @Check = 0;

		IF @Check <> @M
		BEGIN
			SET @ErrorMessage = @ErrorMessage + 'Control number is not valid.';
		END



	RETURN @ErrorMessage;
END
```

U nastavku je i test napisan za nju pomoću tSQLt-a:

```
ALTER PROCEDURE [SkiPassTests].[test is JMBG valid]
AS
BEGIN

  DECLARE @ErrorMessage NVARCHAR(MAX);
  DECLARE @ValidJMBG NVARCHAR(13) = '0401001365287';
  DECLARE @InvalidControlNumberJMBG NVARCHAR(13) = '1510987360289';
  DECLARE @ContainsLettersJMBG NVARCHAR(13) = '15109873Af289';
  DECLARE @WrongNumberOfDigitsJMBG NVARCHAR(15) = '151098732528955';
  DECLARE @InvalidDateOfBirthJMBG NVARCHAR(13) = '3401001365287';
  DECLARE @DateOfBirthInFutureJMBG NVARCHAR(13) = '0401025365287';

  SELECT @ErrorMessage = dbo.fn_JMBGValidation(@ValidJMBG);

  EXEC tSQLt.AssertEqualsString '', @ErrorMessage;

  SELECT @ErrorMessage = dbo.fn_JMBGValidation(@InvalidControlNumberJMBG);
  EXEC tSQLt.AssertEqualsString 'Control number is not valid.', @ErrorMessage;

  SELECT @ErrorMessage = dbo.fn_JMBGValidation('');
  EXEC tSQLt.AssertEqualsString 'JMBG cant be null or empty.', @ErrorMessage;

  SELECT @ErrorMessage = dbo.fn_JMBGValidation(@ContainsLettersJMBG);
  EXEC tSQLt.AssertEqualsString 'JMBG can only contain numbers.', @ErrorMessage;

  SELECT @ErrorMessage = dbo.fn_JMBGValidation(@WrongNumberOfDigitsJMBG);
  EXEC tSQLt.AssertEqualsString 'JMBG must contain 13 digits.', @ErrorMessage;

  SELECT @ErrorMessage = dbo.fn_JMBGValidation(@InvalidDateOfBirthJMBG);
  EXEC tSQLt.AssertEqualsString 'Date of birth is not valid. Control number is not valid.',
@ErrorMessage;

  SELECT @ErrorMessage = dbo.fn_JMBGValidation(@DateOfBirthInFutureJMBG);
  EXEC tSQLt.AssertEqualsString 'Date of birth is in the future. Control number is not
valid.', @ErrorMessage;

END;
```

- `[SkiPassTests].[test PackageStatistics returns number of ski passes by packages]` – testira funkciju `PackageStatistics` koja vraća tabelu sa statistikom za pakete, odnosno prikazuje broj pojavljivanja paketa u ski pass-evima, po paketima.
- `[SkiPassTests].[test trg_calculate_price]` – testira triger koji se poziva nakon unosa rekorda u tabelu Rental koji ima za cilj da izračuna i upiše cenu u tabelu SkiPass.

Triger koji se poziva nakon unosa rekorda u tabelu Rental izgleda ovako:

```
ALTER TRIGGER [dbo].[trg_calculate_price]
ON [dbo].[Rental]
AFTER INSERT
AS
DECLARE @ski_pass_id bigint;
DECLARE @rental_date datetime;
DECLARE @user_id bigint;
DECLARE @slope_prices_sum float;
DECLARE @days int;
BEGIN;
	select @ski_pass_id = SkiPassID, @rental_date = RentalDate, @user_id = UserID
	from inserted;

	select @days = DATEDIFF(day, ValidFrom, ValidTo)
	from Rental
	where SkiPassID = @ski_pass_id and RentalDate = @rental_date and UserID = @user_id;

	select @slope_prices_sum = sum(ski_slope.price)
	from dbo.SkiSlope as ski_slope
	join dbo.Region as region on ski_slope.RegionID = region.RegionID
	join dbo.PackageRegion as package_region on region.RegionID = package_region.PackageID
	join dbo.Package as package on package_region.PackageID = package.PackageID
	join dbo.SkiPass as ski_pass on package.PackageID = ski_pass.PackageID
	where ski_pass.SkiPassID = @ski_pass_id;

    UPDATE dbo.SkiPass set price = @slope_prices_sum * @days where SkiPassID = @ski_pass_id;



END
```

U nastavku je dat test koji je napisan za ovaj triger u tSQLt-u:

```
ALTER PROCEDURE [SkiPassTests].[test trg_calculate_price]
AS
BEGIN

	DECLARE @ExpectedPrice float;
	DECLARE @ActualPrice float;

	EXEC tSQLt.FakeTable 'dbo.Region';
	EXEC tSQLt.FakeTable 'dbo.PackageRegion';
	EXEC tSQLt.FakeTable 'dbo.SkiSlope';
	EXEC tSQLt.FakeTable 'dbo.Package';
	EXEC tSQLt.FakeTable 'dbo.SkiPass';
	EXEC tSQLt.FakeTable 'dbo.User';

	INSERT INTO Region (RegionID, Name) values (1, 'A');

	 INSERT INTO SkiSlope (SlopeID, RegionID, Name, Capacity, Price, SlopeTypeID)
	 VALUES (1,1,'ss1',200,1,1);
	 INSERT INTO SkiSlope (SlopeID, RegionID, Name, Capacity, Price, SlopeTypeID)
	 VALUES (2,1,'ss1',500,2,1);
	 INSERT INTO SkiSlope (SlopeID, RegionID, Name, Capacity, Price, SlopeTypeID)
	 VALUES (3,1,'ss1',1000,5,1);

	 INSERT INTO Package (PackageID, Name) VALUES (1, 'P1');
```

```
        INSERT INTO PackageRegion (PackageID, RegionID, CreationDate)
    VALUES (1,1,GETDATE());

        INSERT INTO SkiPass (SkiPassID, Price, Status, PackageID)
        VALUES (1,0,1,1);

        INSERT INTO [dbo].[User](UserID, Firstname, Lastname, JMBG, DateOfBirth, Phone, Email
)
        VALUES (1, 'FirstName', 'LastName', 'JMBG', GETDATE(), 'Phone', 'Email');

        INSERT INTO Rental (RentalDate, SkiPassID, UserID, ValidFrom, ValidTo)
        VALUES (GETDATE(), 1, 1, '2020-01-15', '2020-01-18');

        set @ExpectedPrice = 3*(1+2+5);

        set @ActualPrice = ( SELECT Price FROM SkiPass WHERE SkiPassID = 1);

        EXEC tSQLt.assertEquals @ExpectedPrice, @ActualPrice;


END
```

3.2 T.S.T.

TST je alat koji pojednostavljuje pisanje i pokretanje automatskih testova za kôd napisan u TSQL-u. U osnovi ovog alata je TST baza podataka koja sadrži niz uskladištenih procedzra koje predstavljaju API.

Karakteristike ovog alata su:

- Pruža podršku za validaciju pogleda, uskladištenih procedura i funkcija koje kao povratnu vrednosti imaju tabelu
- Pouzdana primena procedura Assert.Equals/Assert.NotEquals – ove procedure otkrivaju kada poređenje ne bi trebalo da se izvrši zbog nekompatibilnosti tipova podataka
- Ima mogućnost pokretanja različitih sesija testiranja nad istom ili različitim bazama podataka
- Ima mogućnost generisanja rezultata u XML formatu
- Automatski rollback promena nad podacima nakon svakog testa
- TST će otkriti ukoliko sopstveni mehanizam rollback-a postane neefikasan – onemogućavanje rollback-a može se vršiti na testnom, nivou paketa ili globalnom nivou
- Može se pokrenuti iz komandne linije ili direktno pozivanjem neke od uskladištenih procedura
- TST infrastruktura je izolovana u zasebnoj bazi podataka

3.2.1 Primena alata T.S.T.

Korišćenjem ovog alata kreirani su sledeći testovi:

- `[dbo].[SQLTest_InsertUser]` – testira izvršenje procedure [dbo].[SkiPass.InsertUser] poredi testne podatke pre i posle unosa.

Procedura InsertUser izgleda ovako:

```
ALTER PROCEDURE [dbo].[SkiPass.InsertUser]
	@UserID BIGINT,
	@Firstname NVARCHAR(50),
	@Lastname NVARCHAR(50),
	@JMBG NVARCHAR(13),
	@DateOfBirth DATE,
	@Phone NVARCHAR(50),
	@Email NVARCHAR(50)
AS
BEGIN
SET NOCOUNT ON;
DECLARE @RetStat INT, @ErrorMessage NVARCHAR(500)
SET @RetStat = 0
    IF @JMBG IS NULL
	BEGIN
		SET @RetStat = 1
		SET @ErrorMessage = 'Polje JMBG je obavezno.'
	END

	IF @EMAIL NOT LIKE '_%@__%.__%'
	BEGIN
		SET @RetStat = 2
		SET @ErrorMessage = 'Email nije u validom formatu.'
	END

	IF @RetStat <> 0
	BEGIN
		RAISERROR (@ErrorMessage,16,@RetStat)
		RETURN
	END

IF @UserID = -2
	BEGIN
		INSERT INTO [dbo].[User]( Firstname, Lastname, JMBG, DateOfBirth, Phone, Email
)
		VALUES ( @Firstname, @Lastname, @JMBG, @DateOfBirth, @Phone, @Email)
		SET @UserID = SCOPE_IDENTITY()
	END
ELSE
	BEGIN
	UPDATE [dbo].[User]
	SET Firstname = @Firstname, Lastname = @Lastname, JMBG = @JMBG, DateOfBirth =
@DateOfBirth, Phone = @Phone, Email = @Email
	WHERE UserID = @UserID
	END

	SET @RetStat = @@ERROR

	IF @RetStat <> 0
	BEGIN
		RAISERROR ('Greška prilikom inserta User-a.',16,@RetStat)
		RETURN
	END

	SELECT 0 AS Greska, 'Uspešno obavljen insert u tabelu dbo.User.' AS Poruka, @UserID AS
ID
END
```

U nastavku je dat odgovarajući test napisan pomoću alata T.S.T.:

```
CREATE PROCEDURE [dbo].[SQLTest_InsertUser]

AS
BEGIN
DECLARE @UserID BIGINT
CREATE TABLE #ExpectedResult(ID BIGINT  NOT NULL PRIMARY KEY IDENTITY,
                                                Firstname NVARCHAR(50),
                                                Lastname NVARCHAR(50),
                                                JMBG NVARCHAR(13),
                                                DateOfBirth DATETIME,
                                                Phone NVARCHAR(50),
                                                Email NVARCHAR(50))

CREATE TABLE #ActualResult(  ID BIGINT  NOT NULL PRIMARY KEY IDENTITY,
                                                Firstname NVARCHAR(50),
                                                Lastname NVARCHAR(50),
                                                JMBG NVARCHAR(13),
                                                DateOfBirth DATETIME,
                                                Phone NVARCHAR(50),
                                                Email NVARCHAR(50))

INSERT INTO #ExpectedResult(Firstname,Lastname,JMBG,DateOfBirth,Phone,Email)
              VALUES('testIme', 'testPrezime', '1111111111111', '2020-02-02', '+381656011065',
'test@gmail.com')

EXEC dbo.[SkiPass.InsertUserReturnID]      @UserID OUTPUT,
                                                                        @Firstname =
'testIme',
                                                                        @Lastname =
'testPrezime',
                                                                        @JMBG =
'1111111111111',
                                                                        @DateOfBirth = '2020-
02-02',
                                                                        @Phone =
'+381656011065',
                                                                        @Email =
'test@gmail.com'

INSERT INTO #ActualResult
       SELECT Firstname,Lastname,JMBG,DateOfBirth,Phone,Email
       FROM dbo.[User]
       WHERE UserID = @UserID

EXEC TST.Assert.TableEquals 'Provera inserta user-a.'

DROP TABLE #ActualResult
DROP TABLE #ExpectedResult

END
GO
```

- [dbo].[SQLTest_SelectPackageRegion] – testira za uneti id paketa da li sadrži regije, jer paket mora sadržati najmanje jednu regiju.

```
ALTER PROCEDURE [dbo].[SQLTest_SelectPackageRegion]
AS
BEGIN
DECLARE @PackageID BIGINT
SET @PackageID = 1

CREATE TABLE #ActualResult(PackageName NVARCHAR(50), RegionName NVARCHAR(50))

INSERT INTO #ActualResult
     SELECT Package.Name, Region.Name
     FROM   Package
                    INNER JOIN PackageRegion
                           ON Package.PackageID = PackageRegion.PackageID
                    INNER JOIN Region
                           ON PackageRegion.RegionID = Region.RegionID
     WHERE Package.PackageID = @PackageID
     EXEC TST.Assert.IsTableNotEmpty Package must contain region.'
DROP TABLE #ActualResult
END
GO
```

3.3. TSQLUnit

TSQLUnit je takođe besplatan open source alat za testiranje kôda pisanog u TSQL-u. Jednostavan je za instalaciju i za korišćenje, s obzirom da se testiranje kôda vrši takođe u TSQL-u.

Mana ovog alata je nedostatak dokumentacije, s obzirom da se uz alat dobija jedino uputstvo za instalaciju istog. Takođe, ovaj alat nije u širokoj upotrebi, te ne postoji dobro razrađena korisnička podrška.

U dobre karakteristike ovog alata može se ubrojati to što ima mogućnost pripreme pre, kao i čišćenja podataka nakon izvršavanja testa, te onaj ko piše test ne mora to da radi manuelno.

Testovi se pokreću izvršavanjem procedure *tsu_runTests,* a da bi test bio prepoznat kao takav neophodno je da njegov naziv počinje sa *ut*.

3.2.2 Primena alata tSQLUnit

U okviru ovog rada kreirani su sledeći testovi korišćenjem ovog alata:

- `[dbo].[ut_TestInsertSkiPass]` – koji testira insert u tabelu SkiPass

```
ALTER PROCEDURE [dbo].[ut_TestInsertSkiPass] AS

 EXEC [SkiPass.InsertSkiPass] 0,1,1

 IF @@ROWCOUNT = 0

  EXEC tsu_failure 'ut_TestInsertSkiPass failed'
```

- `[dbo].[ut_TestViewDailyRentals]` – kojim se testira view koji treba da prikaže broj iznajmljivanja prema danima

U nastavku je dat pomenuti view:

```
ALTER VIEW [dbo].[Daily_rentals]
AS
    select  year(RentalDate)     AS year,
                  month(RentalDate)     AS month,
                  day(RentalDate)              AS day,
                  count(*)                     AS total
    from Rental
    group by RentalDate;
GO
```

A odgovarajući test napisan u tSQLUnit-u:

```
ALTER PROCEDURE [dbo].[ut_TestViewDailyRentals]
AS
 DECLARE @count_before int;
 DECLARE @sum_before int;
 DECLARE @count int;
 DECLARE @sum int;

 set @count_before = (select count(*) from (SELECT count(*) as count FROM dbo.Daily_rentals)
as count)
 set @sum_before = (SELECT SUM(total) FROM dbo.Daily_rentals)

 INSERT INTO Rental (RentalDate) VALUES ('2020-01-16')
 INSERT INTO Rental (RentalDate) VALUES ('2020-01-16')
 INSERT INTO Rental (RentalDate) VALUES ('2020-01-17')
 INSERT INTO Rental (RentalDate) VALUES ('2020-01-18')
 INSERT INTO Rental (RentalDate) VALUES ('2020-01-20')
 INSERT INTO Rental (RentalDate) VALUES ('2020-01-21')
 INSERT INTO Rental (RentalDate) VALUES ('2020-01-21')

 SET @count = @count_before + 5
 SET @sum = @sum_before + 7

 print @count_before
 print @count
 print @sum_before
 print @sum


 IF @count - @count_before != 5

  EXEC tsu_failure 'ut_TestInsertSkiPass failed'

  IF @sum - @sum_before != 7

  EXEC tsu_failure 'ut_TestInsertSkiPass failed'
```

- `[dbo].[ut_TestTriggerCalculatePrice]` – testira trigger za izračunavanje i upis cene ski pass-a prilikom inserta u tabelu Rental

```
ALTER PROCEDURE [dbo].[ut_TestTriggerCalculatePrice]
    AS
     DECLARE @PackageID BIGINT;
     DECLARE @RegionID BIGINT;
     DECLARE @SkiPassID BIGINT;
     DECLARE @UserID BIGINT;
     DECLARE @ExpectedPrice FLOAT;
     DECLARE @ActualPrice FLOAT;
     DECLARE @MaxSlopeID BIGINT;
     SET NOCOUNT ON

     INSERT INTO Region (Name) values ('A');

     SET @RegionID = @@IDENTITY;

     SET @MaxSlopeID = (SELECT MAX(SlopeID) FROM SkiSlope);
```

```
INSERT INTO SkiSlope (SlopeID, RegionID, Name, Capacity, Price, SlopeTypeID)
VALUES (@MaxSlopeID+1,@RegionID,'ss1',200,0.88,1);

INSERT INTO SkiSlope (SlopeID, RegionID, Name, Capacity, Price, SlopeTypeID)
VALUES (@MaxSlopeID+2,@RegionID,'ss2',200,0.4,1);

INSERT INTO SkiSlope (SlopeID, RegionID, Name, Capacity, Price, SlopeTypeID)
VALUES (@MaxSlopeID+3,@RegionID,'ss3',300,0.31,1);

SET @PackageID = (SELECT MAX(PackageID) FROM Package) + 1;

INSERT INTO Package (PackageID, Name) VALUES (@PackageID, 'P1');

INSERT INTO PackageRegion (PackageID, RegionID, CreationDate)
VALUES (@PackageID,@RegionID,GETDATE());

INSERT INTO SkiPass (Price, Status, PackageID)
VALUES (0,1,@PackageID);

SET @SkiPassID = @@IDENTITY;

INSERT INTO [dbo].[User](Firstname, Lastname, JMBG, DateOfBirth, Phone, Email  )
VALUES ('FirstName', 'LastName', 'JMBG', GETDATE(), 'Phone', 'Email');

SET @UserID = @@IDENTITY;

INSERT INTO Rental (RentalDate, SkiPassID, UserID, ValidFrom, ValidTo)
VALUES (GETDATE(), @SkiPassID, @UserID, '2020-01-15', '2020-01-18');

set @ExpectedPrice = 3*(0.88+0.4+0.31);

set @ActualPrice = ( SELECT Price FROM SkiPass WHERE SkiPassID = @SkiPassID);

IF (@ExpectedPrice != @ActualPrice)
   EXEC dbo.tsu_Failure 'TestTriggerCalculatePrice failed.'
```

4. Unit testiranje u C#-u

Unit test je deo kôda (obično metoda) koji poziva drugi deo kôda i nakon toga proverava ispravnost određenih pretpostavki. Ako se ispostavi da su pretpostavke pogrešne, jedinični (unit) test nije uspeo.

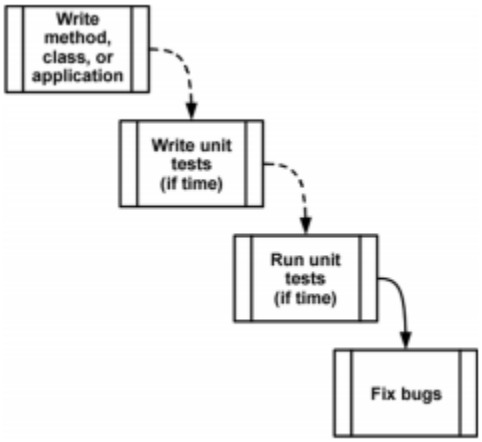
Karakteristike dobro napisanog jediničnog testa su sledeće:

- Test bi trebalo da bude automatizovan i ponovljiv
- Trebalo bi da bude lak za sprovođenje
- Svako bi trebalo da bude u mogućnosti da pokrene test pritiskom na dugme
- Treba da radi brzo
- Treba da bude dosledan u rezultatima – uvek vraća isti rezultat ako se ništa nije promenilo između pokretanja
- Treba da bude potpuno izolovan – da radi nezavisno od ostalih testova
- Kada test ne uspe, trebalo bi da može lako da se otkrije šta se očekivalo i u čemu je problem

4.1 Test-driven development

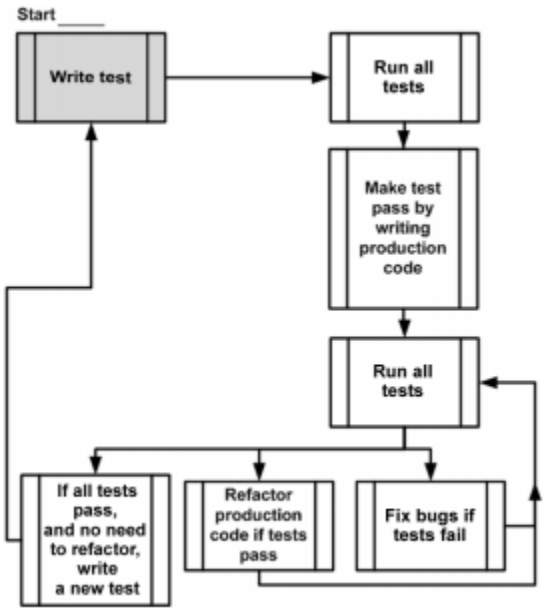
Kada se usvoji znanje o pisanju stuktuiranih i održivih testova, postavlja se pitanje kada treba pisati testove. Mnogo je mišljenja da je pravo vreme za to nakon što je softver gotov, ali je sve više onih koji misle da bi unit testove trebalo pisali pre pisanja produkcijskog kôda. Ovakav pristup se naziva *test-first* ili *test-driven developement (TDD)*.

Na sledećoj slici prikazan je tradicionalan pristup u pisanju jediničnih testova, gde vidimo da se prvo pišu metode, zatim testovi koji se izvršavaju i nakon toga se uklanjaju uočeni nedostaci:



Slika 2 : Tradicionalan pristup testiranju

Test driven developement se razlikuje od tradicionalnog razvoja, kao što je prikazano na narednoj slici. Razvoj se započinje pisanjem testova koji ne prolaze, zatim se piše kôd tako da testovi prođu i nastavlja sa refaktorisanjem kôda ili pisanjem novih testova koji ne prolaze:



Slika 3 : Test driven developement

Tehnika ovog pristupa je veoma jednostavna i sastoji se iz tri koraka:

- Pisanje testova koji se ne završavaju uspešno kako bi se dokazalo da određena funkcionalnost nedostaje finalnom proizvodu – test je napisan kao da softver već radi kako treba, a nije uspeo jer postoji određeni nedostatak

- Napraviti test tako što će se dodati kôd koji ispunjava očekivanja testa. Kod koji se dodaje treba da bude što jednostavniji

- Rekaftorisati (prepraviti) postojeći kôd – prelazak na testiranje naredne celine ili prepravljanje već postojećeg kôda kako bi on bio čitljiviji

Prethodni koraci deluju kao čisto tehnički i laki za sprovođenje, međutim iza njih mora postojati dobro razrađen plan. TDD može poboljšati kvalitet kôda, smanjiti broj grešaka i skratiti vreme potrebno za pronalaženje grešaka ali samo ukoliko je urađen na dobar način, u suprotnom može rezultirati gubljenjem vremena i smanjivanjem kvaliteta kôda.

4.2 Primena unit testiranja u C#-u

Struktura aplikacija SkiPass kreirana je korišćenjem MVP (*Model-View-Presenter*) paterna. Testiranje korišćenjem Unit testova sprovedeno je na slojevima View i Service.

U okviru View sloja, kreirani su sledeće test metode za validacije polja za unos novog korisnika:

- `public void ValidationEmailTest()`
- `public void ValidationEmailTestFail()`
- `public void ValidationPhoneTest()`
- `public void ValidationPhoneTestFail()`
- `public void RequiredFieldsTestFail()`

Klasa ViewUserTests data je u nastavku:

```
namespace SkiPass.View.Tests
{
    [TestClass()]
    public class ViewUserTests
    {
        [TestMethod()]
        public void ValidationEmailTest()
        {
            var view = new ViewUser();
            Assert.IsTrue(view.ValidationEmail("boba@gmail.com"));
        }

        [TestMethod()]
        public void ValidationEmailTestFail()
        {
            var view = new ViewUser();
            Assert.IsFalse(view.ValidationEmail("bobaaa.com"));
        }

        [TestMethod()]
        public void ValidationPhoneTest()
        {
            var view = new ViewUser();
            Assert.IsTrue(view.ValidationPhone("0656011065") &&
view.ValidationPhone("+3810656011065"));
        }

        [TestMethod()]
        public void ValidationPhoneTestFail()
        {
            var view = new ViewUser();
            Assert.IsFalse(view.ValidationPhone("0656asdsa011065"));
```

```
        }


        [TestMethod()]
        public void RequiredFieldsTestFail()
        {
            var view = new ViewUser();
            var date = new DateTime();
            Assert.IsFalse(view.RequiredFields(string.Empty, string.Empty,
string.Empty, string.Empty, string.Empty,date));
        }
    }
}
```

Dok su u okviru Service sloja, kreirane testne metode za proveru konecije na bazu, kao i izvršenje različitih procedura. Testne metode za Service sloj su sledeće:

- `public void ConnectionTest()`
- `public void SelectUsersTest()`
- `public void SelectPackagesTest()`
- `public void SelectRegionsTest()`
- `public void SaveUpadateUserTest()`
- `public void InsertRentalTestFail()`
- `public void SelectRegionsByPackageIdTest()`
- `public void UpdatePackageTest()`

Klasa ServiceSkiPassTest data je u nastavku:

```
[TestClass()]
    public class ServiceSkiPassTest
    {
        public SerivceSkiPass ServiceTest { get; set; }


        public string ConnectionString { get; set; }
        public ServiceSkiPassTest()
        {
            ConnectionTest();
            ServiceTest = new SerivceSkiPass(ConnectionString);
```

```
        }
        [TestMethod()]
        public void ConnectionTest()
        {
            ConnectionString = "Persist security info=false;Integrated
security=SSPI;Data source=DESKTOP-HMK35GO;Initial catalog=SkiPass";
            try
            {
                using (SqlConnection conn = new SqlConnection(ConnectionString))
                {
                    conn.Open();
                }
            }
            catch (SqlException ex)
            {
                Assert.Fail();
            }

        }

        [TestMethod()]
        public void SelectUsersTest()
        {
            ServiceResult result =  ServiceTest.SelectUsers();

            if (result.Value.GetType() != typeof(List<User>))
                Assert.Fail();
        }

        [TestMethod()]
        public void SelectPackagesTest()
        {
            ServiceResult result = ServiceTest.SelectPackages();
```

```
    if (result.Value.GetType() != typeof(List<Package>))
        Assert.Fail();
}

[TestMethod()]
public void SelectRegionsTest()
{
    ServiceResult result = ServiceTest.SelectRegions();

    if (result.Value.GetType() != typeof(List<Region>))
        Assert.Fail();
}
[TestMethod()]
public void SaveUpadateUserTest()
{
    // arrange
    User user = new User()
    {
        Firstname = "tFirstName",
        Lastname = "tLastName",
        Email = "test@gmail.com",
        Phone = "+3815555",
        DateOfBirth = DateTime.Now,
        JMBG = "1206996715192"
    };
    // act
    user.UserID = (int)ServiceTest.SaveUpadateUser(user).Value;

    if (user.UserID <= 0)
        Assert.Fail("Can not save new user.");
```

```
        }

        [TestMethod()]
        public void InsertRentalTestFail()
        {
            Assert.ThrowsException<NullReferenceException>(() => _ =
ServiceTest.InsertRental(null, null, null, null));
        }

        private void AddRegionFromPackage(Package pack)
        {
            ServiceTest.UpdatePackage(pack);
            Assert.AreEqual(pack.Regions.Count,
ServiceTest.SelectRegionsByPackageId(pack.PackageID).Count);
        }

        private Region RemoveRegionFromPackage(Package pack)
        {
            Region testRegion = new Region();
            if (pack.Regions.Count > 1)
            {
                pack.Regions.FirstOrDefault().Status = Status.DELETE;
                testRegion = pack.Regions.FirstOrDefault();
            }

            ServiceTest.UpdatePackage(pack);
            pack.Regions.Remove(testRegion);

            Assert.AreEqual(pack.Regions.Count,
ServiceTest.SelectRegionsByPackageId(pack.PackageID).Count);
```

```
            return testRegion;
        }

        [TestMethod()]
        public void SelectRegionsByPackageIdTest()
        {
            List<Package> packs =
(List<Package>)ServiceTest.SelectPackages().Value;

            foreach (Package pack in packs)
            {
                pack.Regions =
ServiceTest.SelectRegionsByPackageId(pack.PackageID);
                if (pack.Regions == null)
                {
                    Assert.Fail("[Error]: Object Regions is null.");
                    return;
                }

                if (pack.Regions.Count == 0)
                {
                    Assert.Fail("[Error]: Package with id: <"+pack.PackageID+">
doesn't contain any regions.");
                    return;
                }
            }
        }

        [TestMethod()]
        public void UpdatePackageTest()
        {
            // arrange
            Package pack =
((List<Package>)ServiceTest.SelectPackages().Value).FirstOrDefault();
```

```
            pack.Regions = ServiceTest.SelectRegionsByPackageId(pack.PackageID);

            // act1 and assert
            Region testRegion = RemoveRegionFromPackage(pack);

            testRegion.Status = Status.ADD;
            pack.Regions.Add(testRegion);

            //act2 and assert
            AddRegionFromPackage(pack);
        }
    }
```

5. Zaključak

Unit testiranje ima za cilj održavanje kvaliteta napisanog kôda, kao i što ranije otkrivanje i uklanjanje grešaka, čime se štede resursi, uzimajući u obzir činjenicu da je otklanjanje greške jeftinije u ranijim fazama razvoja softvera. Ukoliko se primeni testovima vođen razvoj softvera (Test driven developement) ovaj benefit unit testiranja još više dolazi do izražaja s obzirom da se testovi pišu i pre samog kôda koji obezbeđuje funkcionalnost sistema koji se razvija.

Što se tiče testiranja kôda pisanog u TSQL-u, ne postoji širok izbor besplatnih alata koji se mogu koristiti za isto. Poređenjem tri alata koja su korišćena u okviru ovog rada, a što se tiče testiranja svih delova baze podataka (uskladištene procedure, funkcije svih vrsta, pogleda, trigera) alat tSQLt se pokazao kao najbolji za korišćenje. Njegova prednost leži u tome što uz njega dolazi dokumentacija koja je sasvim dovoljna za brzo usvajanje osnova pisanja testova upotrebom ovog alata, kao i u tome što je dostupna baza podataka koja služi kao primer, gde je napisano nekoliko testova koji pokrivaju dosta slučajeva koji su potrebni za pisanje testova u praksi. Takođe, ovaj alat ima dobru podršku za mock-ovanje (FakeTable, FakeFunction, SpyProcedure), kao i još dodatnih funkcionalnosti kao što su ApplyConstraint i ApplyTrigger što omogućava fokusiranje na svrhu testa bez potrebe da se bavimo stvarima koje su za sam test irelevantne.

Primer koji pokazuje prednost alata tSQLt u odnosu na tSQLUnit jeste test za insert u tabelu koja u sebi ima polje na kom postoji strani ključ. Prvenstveno, tSQLt obezbeđuje korišćenje FakeTable gde ne moramo da vodimo računa o primarnom ključu, već se tabela ponaša kao da u njoj nema podataka. Drugo, ne moramo voditi računa o stranim ključevima tabele ukoliko nam to nije trenutno u fokusu, a ukoliko jeste možemo ih aktivirati korišćenjem procedure ApplyConstraint.

Funkcionalnost koju poseduju alati tSQLt i T.S.T. jeste organizacija testova u više celina, te imamo mogućnost da pozovemo izvršavanje samo određenog skupa testova. Ovo posebno dobija na značaju ukoliko se radi o velikom sistemu koji se sastoji iz više logički odvojenih celina.

Ono što možemo navesti kao prednost svih korišćenih alata jeste to što ne mora da se vodi računa o čišćenju podataka nakon izvršavanja testova, već svi alati imaju mehanizam koji to obezbeđuje.

Generalno, tSQLt obezbeđuje pisanje konciznih, jednostavnih testova, gde je potrebno fokusirati se samo na ono što je predmet tj svrha konkretnog testa, što uz prisustvo obimne dokumentacije ide u prilog činjenici da je od svih korišćenih alata ov

6. Literatura

[1] Computer Science and Information Technology Guide for GATE/ PSUs, Disha Publication, 2016..

[2] J. Tian, Software Quality Engineering - Testing, Quality Assurance, and Quantifiable Improvement, John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.

[3] d. J. Graovac, Projektovanje baza podataka, 2016.

[4] S. D. Lazarević, Ograničenja i pravila integriteta, Beograd, 2018.

[5] E. Erkec, "SQL unit testing best practices," SQLShack, 8 April 2019. [Online]. Available: https://www.sqlshack.com/sql-unit-testing-best-practices/. [Accessed 1 September 2020].

[6] "WELCOME TO TSQLT, THE OPEN SOURCE DATABASE UNIT TESTING FRAMEWORK FOR SQL SERVER," [Online]. Available: https://tsqlt.org/. [Accessed 1 9 2020].

[7] D. Wentzel, "TSQL Unit Testing Tool Comparisons," 17 5 2013. [Online]. Available: https://davewentzel.com/content/tsql-unit-testing-tool-comparisons/. [Accessed 6 9 2020].

[8] R. Osherove, The art of unit testing with examples in C#, Manning Publications, 2014.