

**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

Slika 1 : Prošireni model entiteta i relacija .....	6
Slika 2 : Tradicionalan pristup testiranju .....	21
Slika 3 : Test driven developement .....	21

## Popis tabela

Tabela 1 : Tabela User.....	7
Tabela 2 : Tabela SkiSlopeType .....	7
Tabela 3 : Tabela Region.....	8
Tabela 4 : Tabela SkiSlope .....	8
Tabela 5 : Tabela Package .....	9
Tabela 6 : Tabela SkiPass .....	9
Tabela 7 : Tabela Rental.....	10

## Sadržaj

Popis slika.....	2
Popis tabela .....	2
1. Uvod .....	4
2. Studijski primer .....	5
2.1 Verbalni model.....	5
2.2 Prošireni model entiteta i relacija, relacioni model i rečnik podataka.....	5
2.2.1 Prošireni model entiteta i relacija.....	5
2.2.2 Relacioni model .....	6
2.2.3 Rečnik podataka .....	7
2.3 Specifikacija vrednosnih ograničenja .....	11
2.3.1 Ograničenja nad domenom .....	11
2.3.2 Ograničenja na vrednosti atributa.....	11
3. Alati za testiranje TSQL kôda.....	14
3.1 tSQLt .....	14
3.1.1 Primena alata tSQLt.....	16
3.2 T.S.T.....	18
3.2.1 Primena alata T.S.T.....	18
3.3. TSQLUnit.....	19
3.2.2 Primena alata tSQLUnit.....	19
4. Unit testiranje u C#-u .....	20
4.1 Test-driven development.....	20
4.2 Primena unit testiranja u C#-u.....	22
5. Zaključak.....	24
6. Literatura.....	25

## 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 – pronadi 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 izvršeno 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.

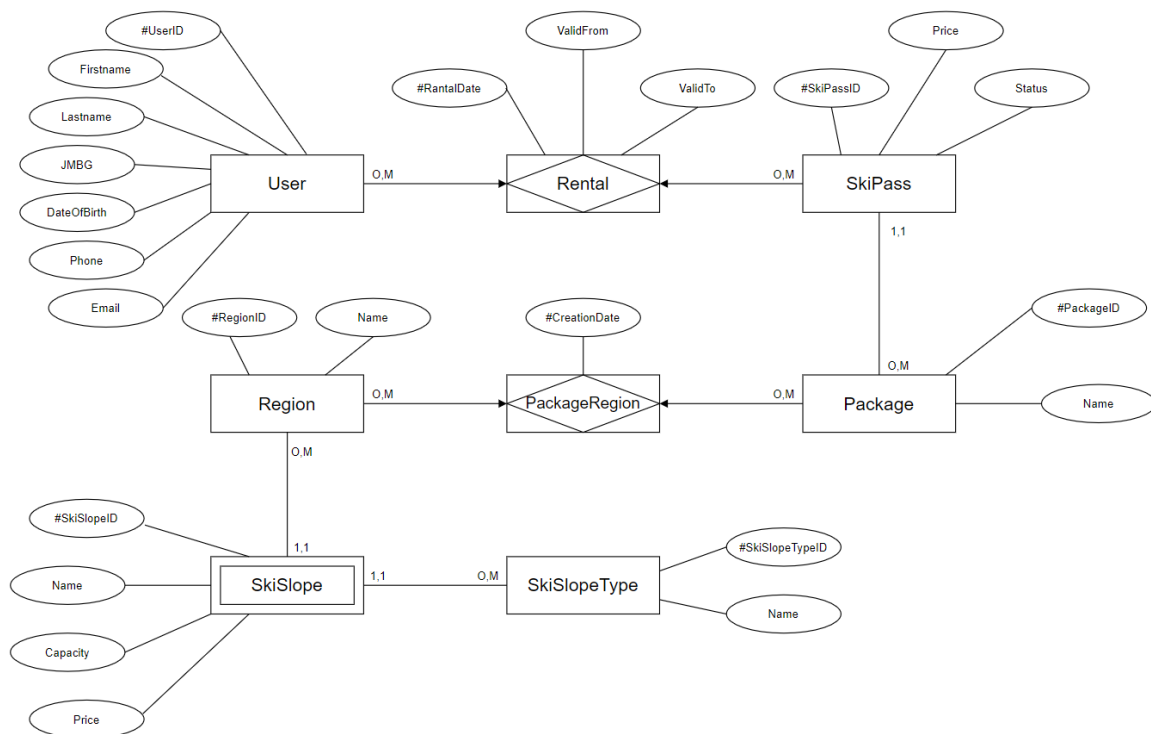
Sistem treba da omogući unos korisnika, ski pass-a, paketa i iznajmljivanja ski pass-a od strane korisnika. Pored toga, potrebno je omogućiti izmenu korisnika i paketa.

### **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 attribute i među njima mogu da postoje odnosi. Odnosi su označeni kardinalnošću i takođe mogu da imaju attribute.

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
	RegionID	bigint	not null and >0			DELETE RESTRICT PackageRegion
	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
	SkiSlopeID	bigint	not null and >0			DELETE /
	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
	PackageID	bigint	not null and >0			DELETE RESTRICT PackageRegion
	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
	SkiPassID	bigint	not null and >0			DELETE RESTRICT Rental
	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
	RentalDate	datetime	not null and >0			DELETE /
	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 imati 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.Capacity > 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 testiranja 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 @IfExists = 1
- SpyProcedure – omogućava pisanje testova za procedure izolujući 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.
- [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.

- [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.
- [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 trigger koji se poziva nakon unosa rekorda u tabelu Rental koji ima za cilj da izračuna i upiše cenu u tabelu SkiPass.

## 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 procedura 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.
- [dbo].[SQLTest\_SelectPackageRegion] – testira za uneti id paketa da li sadrži regije, jer paket mora sadržati najmanje jednu regiju.

### 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`
- `[dbo].[ut_TestViewDailyRentals]` – kojim se testira view koji treba da prikaže broj iznajmljivanja prema danima
- `[dbo].[ut_TestTriggerCalculatePrice]` – testira trigger za izračunavanje i upis cene ski pass-a prilikom inserta u tabelu `Rental`

## 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.

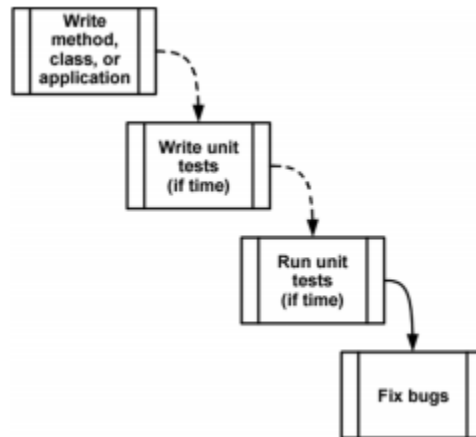
Karakteristike dobro napisanog jediničnog testa su sledeće:

- 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

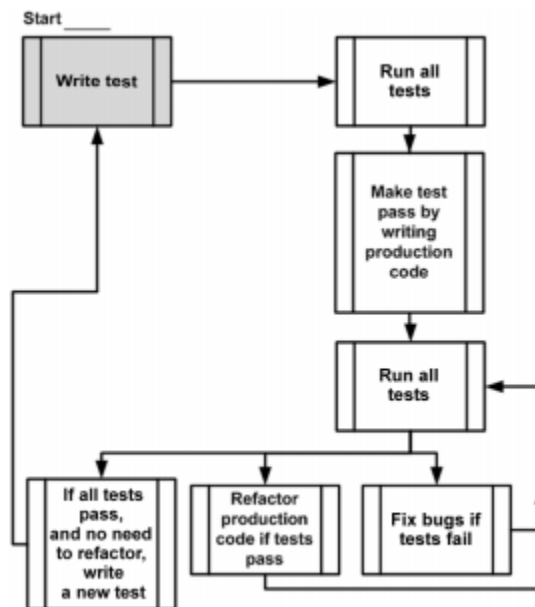
Kada se usvoji znanje o pisanju struktuiranih i održivih testova, postavlja se pitanje kada treba pisati testove. Mnogo je mišljenja da je pravo vreme za to nakon što je softver gotov, ali je sve više onih koji misle da bi unit testove trebalo pisati pre pisanja produkcijskog kôda. Ovakav pristup se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

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()`

Dok su u okviru Service sloja, kreirane testne metode za proveru konekcije 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 SaveUpdateUserTest()`

- `public void InsertRentalTestFail()`
- `public void SelectRegionsByPackageIdTest()`
- `public void UpdatePackageTest()`

## 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.