Softverski inženjering – Teorijski dio

Autor: Adnan Jusić

1. **Procedura je kombinacija čega?**
   Proceduraje kombinacija alata i metoda koji u međusobnom skladu proizvode dati proizvod.

2. **Sistem je skup čega?**
   Sistem je skup entiteta, aktivnosti, njihovih odnosa i granica sistema.

3. **U fazi projektovanja obično učestvuju?** Analitičar i projektant.

4. **Koji su razlozi zbog kojih se softverski inženjering stalno mijenja?**
   - Model vodopada
   - Vremenski aspekt
   - Značajna proširenja s aspekta propusnosti računarskih mreža
   - Interakcija s korisnikom je postala drugačija
   - Ekonomski aspekti
   - Isporuka softvera mora podržati pojavu različitih uređaja

5. **Koji model je specifičanpo dokumentaciji?** Model vodopada.
6. **Koji je nedostatak transofrmacionog modela?** Potrebna precizna formalna specifikacija

7. **Koji model smanjuje i kontroliše rizike?** Spiralni model.

8. **Koji su elementi koje naručilac i razvojni tim koriste za praćenje projekta?** Aktivnosti i prekretnice predstavljaju elemente koje naručilac i razvojni tim koriste za praćenje projekta.

9. **Metodom kritične putanje mogu se odrediti?**Analizom putanja među prekretnicama projekta, **metodomkritične putanje** (Critical Path Method), mogu se odrediti aktivnosti koje su najkritičnije za završetak projekta u predviđenom vremenu.

10. **Jedno ili više pitanja kad su u pitanju stilovi komunikacije učesnika, slijede u nastavku:**
    - Intuitivan introvert – priznaje osjećaje, pita drugog.
    - Intuitivan ekstrovert – priznaje osjećaje, saopštava drugima.
    - Racionalan introvert – odlučuje logično, pita drugog.
    - Racionalan ekstrovert – odlučuje logično, saopštava drugima.

11. **Ukoliko više ljudi učestvuje na projektu koju strukturu organizacije odabrati?** Metodu „Glavni programer“.

12. **Ukoliko se radi o projektima sa velikim stepenom izvijesnosti, stabilnosti, uniformnosti koju od struktura organizacije odabrati?** Metodu „Glavni programer“.

13. **Ukoliko projekat uključuje značajan stepen neizvijesnosti?** Metodu „Bez egoizma“.

14. **Procjena rizika uključuje?** Identifikovanje rizika (način, uzrok i potencijalno mjesto pojave različitih događaja), analiza rizika i dodjeljivanje prioriteta svakom riziku.

15. **Kontrola rizika uključuje?** Umanjenje rizika(izbjegavanje rizika izmjenom zahtjeva, prenošenje rizika dodjeljivanjem rizika drugim sistemima, podrazumijevanje rizika - prihvatanje i kontrola u okviru resursa projekta), planiranje rizika i razrješavanje rizika.

16. **Tehnike definisanja zahtjeva su:**
    - Analiza službene dokumentacije,
    - intervju,
    - anketiranje,
    - analiza scenarija,
    - posmatranje,
    - izrada pretpostavki,
    - pravljenje i korištenje video zapisa,
    - FAST tehnika i
    - izrada prototipa.

17. **Opisuje obavezno ponašanje softvera u kontekstu neophodnih aktivnosti, a neke od njih su: reakcije na ulaze, stanje svakog entiteta nakon okončanja aktivnosti i sl. O kakvim se zahtjevima radi?** Funkcionalni zahtjevi.

18. **Opisuje neke karakteristike kvaliteta koje softversko rješenje mora da posjeduje, a neki od njih su: kratko vrijeme odziva, lakoća korištenja, visoka pouzdanost, niski troškovi održavanja i sl.O kakvim se zahtjevima radi?** Nefunkcionalni zahtjevi.

19. **Predstavlja unaprijed donijetu odluku koja ograničava skup mogućih rješenja razmatranog problema, na primjer: izbor platforme, interfejs komponenti i sl. Riječ je o?** Projektno ograničenje.

20. **Predstavlja ograničenje koje se odnosi na tehnike ili resurse koji se mogu koristiti u izgradnji sistema. Riječ je o?**Procesno ograničenje.

21. **Kada se koristi posmatranje kao metoda prikupljanja zahtjeva?**
    - Kada nema dovoljno podataka ni dokumenata ili su procesi stohastički
    - Vrši se po utvrđenom planu
    - Rezultati se statistički obrađuju
22. **Koja je faza obavezna u evidentiranju zahtjeva?** Analiza službene dokumentacije.

23. **Šta je validacija zahtjeva i čemu ona služi?** Validacija zahtjeva daje garancije da se razvija pravi sistem. Služi kao mehanizam utvrđivanja stepena odstupanja definisanih zahtjeva i stvarnih zahtjeva naručioca. Ponekad podrazumijeva jednostavno čitanje kreiranih dokumenata i izvještavanje o greškama.

24. **Šta je verifikacija zahtjeva i čemu ona služi?** Verifikacija provjerava da li sistem koji zadovoljava specifikaciju ujedno zadovoljava i zahtjeve naručioca. Analiziraju se mogućnosti praćenja. Ukoliko formirane pretpostavke ne ispunjavaju zahtjeve naručioca, potrebno je izmijeniti specifikaciju ili smanjiti zahtjeve koji se pokušavaju zadovoljiti.

25. **Koji se stilovi najčešće koriste?**
    - Cijevi i filteri
    - Objektno-orijentisan dizajn (OO dizajn)
    - Implicitno povezivanje
    - Slojevi
    - Skladišta
    - Interpreteri
    - Kontrola procesa

26. **Koje karakteristike treba da posjeduje dobar konceptualni dizajn?**
    - Da bude pisan jezikom klijenta (klijentu razumljivim rječnikom)
    - Da ne sadrži previše tehničkih termina
    - Da opisuje funkcije sistema
    - Da ne zavisi od implementacije
    - Da bude povezan s dokumentima specifikacije zahtjeva

27. **Na koje načine se dizajn može realizovati?**
    - Modularno raščlanjivanje
    - Raščlanjivanje na osnovu podataka
    - Raščlanjivanje na osnovu događaja
    - Dizajn spolja ka unutra
    - Objektno-orijentisani dizajn

28. **Značenje svakog od slova u SOLID?**

- **S**ingle Responsibility Principle (SRP) - svakoj komponenti (klasi, modulu i sl.) se dodjeljuje samo jedno zaduženje.
- **O**pen-Closed Principle (OCP) - radna verzija komponenti(postojeći – funkcionalni sadržaj) se samo može proširivati, a nikako i mijenjati
- **L**iskov Substitution Principle (LSP) - implementirana nasljeđivanja mora biti ispravna (kako bi se moglaosigurati adekvatna supstitucija).
- **I**nterface Segregation Principle (ISP) - kada interfejs postane previše složen, potrebno ga je raščlaniti na specifičnije interfejse.
- **D**ependency Inversion Principle (DIP) - zavisnosti i veze bazirati na apstrakciji, a ne na konkretnim primjercima.

29. **U okviru faze implementacije softvera je potrebno voditi računa o kojim aspektima?**
    - minimiziranju kompleksnosti
    - budućim promjenama/nadogradnji
    - verifikacijskim mehanizmima
    - primjeni standarda
    - ponovnoj iskoristivosti koda

30. **Kome je namijenjena interna dokumentacija?** Interna (unutrašnja) dokumentacija sadrži informacije namijenjene onima koji će imati uvida u programski kod projekta (razvojni tim).

31. **Kome je namijenjena eksterna dokumentacija?** Eksterna (spoljnja) dokumentacija je namijenjena onima koji možda nikada neće imati uvid u programski kod projekta (klijent, korisnici itd.).

32. **Najčešća podjela korisničkih predložaka?**
    - **kreacijski predlošci** (engl. Creational) – primjenjuju se u okviru procesa kreiranja objekata
    - **strukturalni predlošci** (engl. Structural) – primjenjuju se u okviru procesa kreiranja odnosa/kompozicije između klasa/objekata
    - **predlošci ponašanja** (engl. Behavioral) – primjenjuju se prilikom definisanja načina interakcije i raspodjele obaveza između klasa/objekata

33. **Kreacijski predlošci su:**
    - Builder
    - Factory method
    - Prototype
    - Singleton

34. **Strukturalni predlošci su:**
    - Adapter
    - Bridge
    - Composite
    - Decorator
    - Facade
    - Flyweight
    - Proxy

35. **Predlošci ponašanja (behavioral) su:**
    - Interpreter
    - Mediator
    - Momento
    - Observer
    - Strategy
    - Template method

36. **Šta zahtijeva implementacija singleton predloška?**
    - Deklaraciju konstruktora kao privatnog ili zaštićenog
    - Deklaraciju privatnog statičkog atributa koji je istog tipa kao i sama klasa
    - Definiciju javne statičke funkcije koja će predstavljati jedini način za pristup prethodno pomenutom atributu

37. **Koje su ključne komponente Builder predloška?**
    - **Builder** interfejsa – specificira interfejs za kreiranje objekata
    - **ConcreteBuilder** klasa – kreira dijelove neophodne za konstruisanje određenog (kompleksnog) objekta
    - **Director** klasa – konstruiše kompleksne objekte zahvaljujući Builder interfejsu
    - **Product** klasa – predstavlja kompleksni objekat koji je potrebno kreirati iz više dijelova

38. **Koje su ključne komponente Adapter predloška?**
    - **ITarget** – definiše specifični interfejs koji koristi klijent
    - **Adapter** – prilagođava interfejs **Adaptee** konkretnom **Target** interfejsu
    - **Adaptee** – postojeći interfejs kojeg treba prilagoditi
    - **Client** – komunicira sa objektima koji implementiraju **Target** interfejs

39. **Koje su ključne komponente Observer predloška?**
    - **Observable** – interfejs koji definiše operacije za pretplatu posmatrača (Observer-a) na određeni događaj. Ovaj interfejs se često naziva **Subject**
    - **ConcreteObservable** – klasa koja čuva i obavještava pretplaćene posmatrače

- **Observer** – interfejs koji definiše operacije za obavještavanje posmatrača (Observer-a) o nastalim promjenama.
- **ConcreteObserver** – konkretna implementacija posmatrača

40. **Definicije vrsta testiranja.**
    - Jedinično testiranje– provjera da li pojedinačne komponente ispravno funkcionišu sa svim očekivanim tipovima ulaza.
    - Integraciono testiranje– provjera da li sistemske komponente sarađuju kao što je opisano u specifikacijama dizajna sistema.
    - Funkcionalno testiranje– provjera da li integrisani sistem zaista izvršava funkcije opisane u specifikaciji zahtjeva.
    - Testiranje performansi- sistem se poredi sa ostatkom hardverskih i softverskih zahtjeva.
    - Test prihvatanja– test u kome se provjerava usklađenost sistema sa opisom zahtjeva naručioca.
    - Instalacioni test– potvrđuje da li sistem ispravno funkcioniše i nakon isporuke.

41. **Šta je otkaz?** Otkaz je situacija u kojoj softver ne radi ono što je opisano u zahtjevima.

42. **Najčešći uzroci otkaza su?**
    - pogrešna specifikacija zahtjeva
    - nemogućnost implementacije definisanog zahtjeva
    - postojanje greške u dizajnu sistema
    - postojanje greške u programskom kodu

43. **Šta je slučaj, a šta kolekcija?** Slučaj za testiranje je konkretan izbor ulaznih podataka koji će se upotrijebiti za testiranje programa, a test je konačna kolekcija slučajeva.

44. **Programer i tester potvrđuju validnost korisničkih priča?** Tačno.

45. **Šta procesi i alati predstavljaju u XP-u?** Interakciju.

46. **Razlike između SCRUM-a i XP-a?**
    - Trajanje pojedinih iteracija
    - Način prihvatanja promjena u pojedinim fazama razvoja
    - XP posjeduje više inžinjerski, praktični pristup

47. **Praktični principi XP-a su:**
    - Stalno prisustvo klijenta - naručioca softvera

- Članovi tima na istoj lokaciji (prostorija/sprat)
- Programiranje u paru
- Kolektivna svojina
- Standardi kodiranja
- Stalne modifikacije bez značajnog utjecaja na sistem (refaktorisanje)
- Stalno testiranje - **TDD** (Test-Driven Development) *pisanje testova prije kodiranja (Unit test)* i **AT** (Acceptance Testing) *otkriva nejasnoće u zahtjevima u ranijim fazama.*
- Kontinuirana integracija
- Igra planiranja (otkriva funkcionalnosti i cijenu softvera)
- Višestruke verzije softvera (manje, ali funkcionalne verzije softvera, povrat uloženih sredstava i brza povratna informacija)
- Metafore (jednostavno pojašnjavanje koncepata i funkcionalnosti)
- Jednostavan dizajn
- Održiv tempo rada
- Individualne zabilješke

48. **Koliko članova se preporučuje za XP tim?** 2-12.

49. **Osnovne uloge XP trenera su:**
    - Praćenje poštivanja XP principa i praksi
    - Predvodi razvojni tim tokom XP procesa
    - Uparuje članove tima u fazama u kojima je to neophodno

50. **Osnovne uloge XP programera su:**
    - Procjena *korisničkih priča* i podjela na manje zadatke
    - Procjena vremena potrebnog za razvoj
    - Kreiranje testova (*Unit test*)
    - Refaktorisanje
    - Interakcija s korisnikom

<u>NAPOMENA</u>: Posljednja dva pitanja vezana su za određene uloge u XP timu i na ispitu su se pojavila studentima. Međutim, u bazi vjerovatno postoji po jedno pitanje za svaku ulogu u XP timu!

51. **Vrijednosti XP-a?**
    - Komunikacija
    - Jednostavnost
    - Brze povratne informacije
    - Hrabrost

Praktično:

1. **Šta je ispravno od sljedećeg:**
   A: ctx.Studenti.Where(x=>x.ImeIPrezime == "Ime studenta").ToList();
   B: ctx.Studenti.ToList().Where(x=>x.ImeIPrezime == "Ime studenta").ToList();

   Oboje je ispravno. Ukoliko se „Where“ nađe prije „ToList()“ funkcije, filtriranje se vrši na SQL serveru, a ako se „Where“ nađe poslije „ToList()“ funkcije, filtriranje se vrši u C# kodu.

2. **Da li je moguće uraditi sljedeće?**

```
public class Student
{
        public int Id { get; set; }
        public string Ime { get; set; }
        public string Prezime { get; set; }
        public string Ime_Prezime {set; get { return Ime + " " + Prezime }};
}
```

   Da, moguće je kreirati složeni atribut koji kao povratnu vrijednost ima kombinovanu vrijednost drugih atributa, ali ti atributi moraju biti istog tipa kao i složeni atribut. Ukoliko nisu istog tipa, moguća je konverzija u taj tip i to će biti prihvaćeno. Primjer u nastavku:

```
public string Ime_Prezime_DatumRodjenja
{set; get { return Ime + " " + Prezime + " " + DatumRodjenja.ToString() }};
```

   Iako *DatumRodjenja* nije tipa string, sasvim je legalan način konvertovati datum u string i to će odgovarati uvjetu da su svi atributi koji čine složeni - istog tipa.

3. **Primarni ključevi u EntityFrameworku.**
   EntityFramework prepoznaje primarni ključ, po defaultu, na osnovu imena. Validni nazivi za primarni ključ su **Id** ili **<NazivModela>Id**, npr. ako imamo model „Vozila“, primarni ključ za model nije validan ako je ~~**VoziloId**~~, validni su jedino **VozilaId** ili **Id.** Naravno, moguće je i to postići, ali za to je potrebno da u ModelBuilderu navedemo eksplicitno šta je ključ za taj model (jer ga EntityFramework neće moći prepoznati automatski), i to na sljedeći način:

```
modelBuilder.Entity<Vozila>.HasKey(x => x.VoziloId);
```

Primarni ključ, bez dodatnih modifikacija u kodu, ne može biti string, ispisuje se sljedeća poruka o grešci:

„Identity column 'AutoId' must be of data type int, bigint, smallint, tinyint, or decimal or numeric with a scale of 0, and constrained to be nonnullable.“
To se dešava jer EntityFramework automatski za odabrani primarni ključ stavlja auto inkrement, dakle, automatsko povećanje primarnog ključa. S obzirom da EF jedino numeričke tipove zna (može) inkrementirati, ispisuje poruku o grešci. To je moguće riješiti dodavanjem sljedeće linije koda iznad primarnog ključa (u modelu):
`[DatabaseGenerated(DatabaseGeneratedOption.None)]`

Ovo će isključiti auto inkrement i bit će moguće koristiti string kao Id.

4. **Ukoliko imamo dvije akcije koje preusmjeravaju jedna na drugu, šta će se desiti? Koji zahtjev je u pitanju (POST ili GET)?**

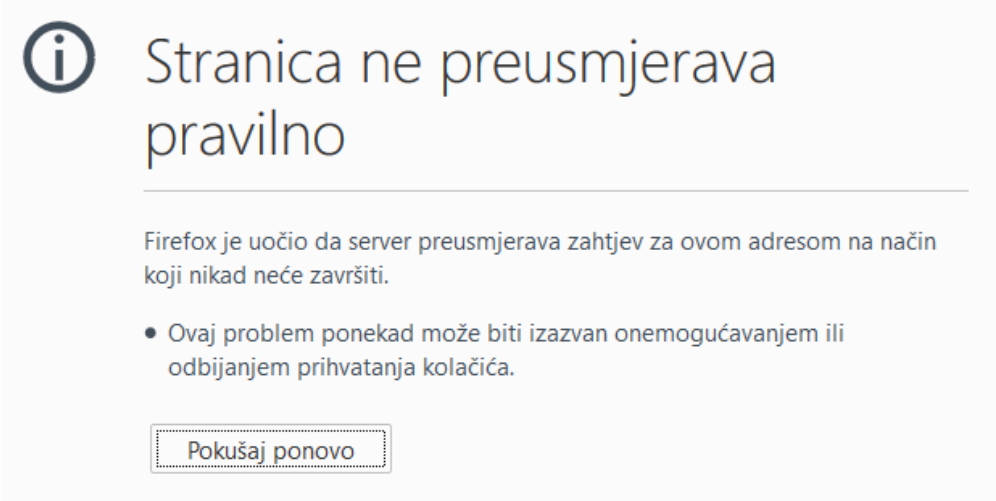
```
public ActionResult A()
{

    return RedirectToAction("B");
}

public ActionResult B()
{

    return RedirectToAction("A");
}
```

Browser prepoznaje da se radi o nepravilnom preusmjeravanju koje neće nikad završiti i ispisuje sljedeću grešku:



Kad se koristi `RedirectToAction` u kontroleru, akcija automatski preusmjerava koristeći **HTTP GET**. Kad preusmjerite zahtjev negdje, HTTP „Location“ zaglavlje govori internet pregledniku gdje da ide, a zatim internet preglednik pošalje GET zahtjev za tu stranicu.

5. **Veze u EntityFrameworku koristeći fluent API.**

   Naredne dvije linije koda predstavljaju iste relacije (video lekcija):

   modelBuilder.Entity<Korisnik>().HasOptional(x=> x.Student).WithRequired(x=>x.Korisnik);

   Objašnjenje: Entitet Korisnik ne mora, a može sadržavati (opcinalno) entitet Student (njegove atribute), a Student mora sadržavati entitet Korisnik (njegove atribute).

   modelBuilder.Entity<Student>().HasRequired(x=> x.Korisnik).WithOptional(x=>Student);

   Objašnjenje: Entitet Student mora sadržavati entitet Korisnik (njegove atribute), a Korisnik ne mora, a može sadržavati (opcionalno) entitet Student (njegove atribute).

6. **Ukoliko je uključena migracija na EF, da li je moguća izmjena strukture baze podataka na SQL Serveru?** Nije, jer promjena nad tabelama u bazi neće biti učitana u modelima u kodu. Zbog toga se ovaj pristup i zove *Code-first.*

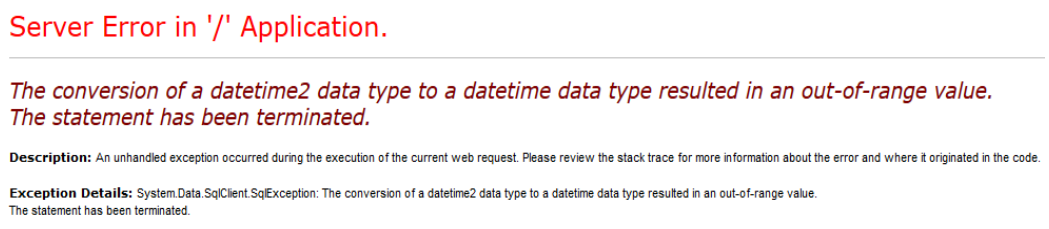
7. **Da li se DbSetovi instanciraju u migracijskom fajlu?** Ne.

8. **Ukoliko model izgleda kao u nastavku:**

```
public class Opcina
{
        public int Id {get; set;}
        public string Naziv {get; set;}
        public DateTime DanOpcine { get; set; }
}
```

   dan općine nije stavljen kao nullable (public DateTime? DanOpcine { get; set; }**), i ako u kontroleru ne setujemo (ne dodijelimo vrijednost) atributu DanOpcine, hoće li se pojaviti greška?**

   Da, pojavit će se greška koja je prikazana u nastavku:


Server Error in '/' Application.

The conversion of a datetime2 data type to a datetime data type resulted in an out-of-range value.
The statement has been terminated.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: The conversion of a datetime2 data type to a datetime data type resulted in an out-of-range value.
The statement has been terminated.