1. U nekoliko rečenica navedite i argumentirajte tri razloga zašto je iterativni razvoj bolji od klasičnog waterfall procesa razvoja. Navedite i jednu „kontradikciju“ iterativnog razvoja (što potencijalno može biti problem)

Waterfall model procesa razvoja softwarea se bazira na jasno odvojenim fazama razvoja – izrada zahtjeva i specifikacije, dizajn sustava, implmentacija i testiranje. Svaka faza se mora u potpunosti dovršiti kako bi sljedeća faza mogla započeti.

Iterativni model faze razvoja vrti u ciklusima (iteracijama) prilikom čega se u svakoj iteraciji rješava dio problema, a ne cijeli problem kao kod Waterfalla.

Iterativni model omogućuje brže i lakše reagiranje na promjene zahtjeva/potrebe korisnika – važnost ranog feedbacka. Izbjegavamo Rush to Code i Analysis Paralysis. Od velikog je značaja učinkoviti Refactoring.

Ono u čemu agilni pristup nije najbolji odabir su veliki projekti koji zaista zahtijevaju jasno određene i dugotrajne faze razvoja. Također, organizacija tima u iterativnom razvoju može biti teža nego u waterfall pristupu.

Opasnost agilnog razvoja je pojava takozvanih anti-patterna: manjak usmjerenosti (fokusa) u dizajnu, opasnost pojave dugih iteracija, potencijalni problemi s poslovnim partnerima, nedovoljno trenirano osoblje, manjak testinga te česta potreba za mijenjanjem već napisanih testova,…

2. Objasnite koncept agregata u kontekstu DDD-a i kakvu ulogu ima u osiguravanju konzistentnosti promjena nad objektima u modelu.

Agregat je skupina povezanih objekata koje tretiramo kao jedinku prilikom promjene podataka. Definiraju ga korijen (jedan točno određeni entity objekt koji je dio agregata) i granica (definira što se nalazi unutar agregata). Vanjski objekti smiju pristupati samo korijenskim elementima drugih agregata.

Važno je spomenuti invarijante – pravila konzistentnosti koja se moraju održati kadgod se promjene podaci. Invarijante koje vrijede unutar agregata su sigurno provedene nakon završetka transakcije iz sljedećih raloga:

- Korijenski entitet ima globalni identitet i odgovoran je za provjeravanje invarijanti
- Entiteti unutar granice imaju samo lokalni identitet, jedinstven unutar agregata
- Ništa izvan agregata ne može držati referencu na objekt unutar granice
- Iz upita nad bazom možemo dobiti samo korijenske entitete
- Objekti unutar agregata mogu sadržati referencu na korijene drugih agregata
- Operacija brisanja mora iz agregata sve ukloniti odjedanput

3. U nekoliko rečenica opišite osnovne principe TDD-a. Koje karakteristike moraju imati unit testovi?

Cilj test driven developmenta je provjera dinamike izgrađenog sustava i ispravnosti izgrađenog poslovnog modela.

Fokus prilikom razvoja mora biti na pisanju testova. Pišu se takozvani test cases prilikom svake iteracije u razvoju kako bi se ispitale nove funkcionalnosti. Od velike važnosti je automacija testiranja jer nam omogućuje brzo i jednostavno testiranje već postojećeg codebasea kako bi se provjerilo je li dodavanje nove funkcionalnosti neželjeno utjecalo na rad već postojeće funkcionalnosti.

Možda i najvažnija stvar u TDD-u je to da TDD potiče razvoj sustava na način da sustav bude lagano testiran. Objektni model koji se može lagano testirati je vrlo često dobrih svojstava (mali coupling, visoka kohezija). Samim time, refactoring i daljnji razvoj postaju lakši.

Unit testovi trebaju zadovoljavati FIRST principe:

- **F**ast – testovi se brzo izvode
- **I**solated – testovi jasno ukazuju na komponentu koja je pala na testu
- **R**epeatable – prilikom svakoga pokretanja testa rezultat mora biti isti (naravno ako nema promjena u sustavu)
- **S**elf-verifying – test jednoznačno prolazi ili ne prolazi
- **T**imely – test napisan otprilike u istom trenutku kada i kod kojeg provjerava, pa čak i ranije

4. Što je to problem „fino granulirane perzistencije“ (fine grained persistance)? Kako se rješava u NHibernate frameworku? Ilustrirajte primjerom klase Transakcija u kojoj je iznos transakcije modeliran preko razreda MonetaryAmount koji sadrži Iznos (float) i Valutu (int koji predstavlja šifru valute).

Zbog složenosti objekata dolazimo do pitanja treba li baš svaki razred imati svoju tablicu. Fine-grained objektni model je poželjan jer potiče veći code reuse, strožu tipizaciju i bolju enkapsulaciju. Međutim, postoje problemi s fine-grained objektima.

Pretpostavimo da svaki razred ima svoju tablicu te da se koriste spajanja u bazi podataka kako bi se kompozirali složeniji objekti. Ako objekt Transakcija sadrži objekt MonetaryAmount, prilikom dohvata objekta Transakcija moramo raditi join između tablica TRANSAKCIJA i MONETARY_AMOUNT. Join je generalno vremenski veoma zahtjevna operacija u bazi podataka, pogotovo za veliki broj unosa.

Zbog poboljšanja performansi denormaliziramo tablice te MonetaryAmount shvaćamo kao value-type object, a ne entitet. To znači da ga možemo „ugurati“ u istu tablicu kao i Transaction (koje je entity-type). Sada imamo problem da iz jednog retka tablice moramo u našem programu stvoriti više objekata – neki stupci pripadaju primjerice vremenu transkacije, a ostali iznosu i oznaci valute.

NHibernate omogućava developeru stvaranje distinkcije između entity i value typeovam, što se opisuje u metapodacima. U bazi ćemo imati jednu tablicu TRANSAKCIJA koja će sadržati sve podatke o transkaciji (npr. stupac VRIJEME_TRANSAKCIJE) zajedno sa „umetnutim“ potrebnim vrijednostima za MonetaryAmount (stupci IZNOS i VALUTA). Prilikom pisanja mapiranja treba se koristiti ključna riječ Component.

5. Koji problemi se javljaju prilikom mapiranja objektnog modela u relacijske baze („Impedance Mismatch“)? Navedite sve koje znate i svaki opišite u jednoj ili dvije rečenice.

Pojam object-relational impedance mismatch označava konceptualne i tehničke poteškoće prilikom korištenja relacijskih baza podataka za pohranu objekata objektno-orijentiranog sustava.

- Neslaganje između objektne i relacijske tehnologije – različiti teorijski temelji
    - Relacije predstavljaju činjenice o povezanosti podataka
    - Objekti su model realnosti
- Neslaganje pravila o jedinstvenosti
    - Dva retka koji sadrže iste podatke (isključujući eventualni ID) su isti
    - Objekti koji drže iste podatke su različiti (različite adrese u memoriji)
- Neslaganje topologije podataka
    - Relacijske tablice su „ravne“

    - Objekti su hijerarhijski ustrojeni

Problemi koji nastaju prilikom spremanja objekata u relacijske baze:

- Problemi s granularnosti
    - Postoji mnoštvo objekata različitih razreda te je iz stajališta baze neefikasno stvarati nove tablice za „sitne“ razrede kao što je npr. MonetaryAmount iz 4. pitanja
    - Rješava se denormalizacijom i mapiranjem komponenti
- Problemi s podtipovima
    - Problem je s nasljeđivanjem i spremanjem hijerarhije u tablice
    - Različita rješenja za različite situacije
        - Po mogućnoti izbjegavati nasljeđivanje i koristiti kompoziciju
- Problemi s identiteta
    - Kako provjeriti da su dva objekta jednaka?
        - Kod objekata provjeravamo jednakost provjerom memorijskih adresa
        - Kod redaka tablice gledamo vrijednost primarnog ključa
- Problemi asocijacija
    - Odnosi između entiteta
        - OO – preko referenci na objekte – usmejrana
        - SQL – preko stranih ključeva – bidirekcionalna
- Problemi s navigacijom po grafovima objekata
    - OO – „šetanje“ po referencama
    - Mnoštvo SELECT i JOIN naredbi
- Troškovi
    - Značajan trud je potreban za nadvladavanje Impedance Mismatcha
    - Značajan dio koda potreban za povezivanje objektnog modela i baze podataka

6. Kako glasi Demeterov zakon? S kojim je principom objektno-orijentirane paradigme blisko vezan? Ilustrirajte na jednostavnom primjeru te navedite dobre i one „ne tako dobre“ posljedice tog zakona.

Demeterov zakon govori o tome kome sve objekt smije slati poruke. Tehnički, objekt bi trebao pozivati servise samo sljedećih objekata:

- this (odnosno samog sebe)
- Objekti koji su mu predani kao parametri
- Objekti koje je sam kreirao
- Bilo koji objekt na koji ima referencu, a koji je njegov podobjekt

Primjer dostavljača i novčanika. Osoba koja naručuje plaća pizzu dostavljaču. Kako bi osoba platila, uzima svoj novčanik, vadi iz njega novac i uručuje ga dostavljaču. Ono što ne želimo je da osoba uručuje svoj novčanik dostavljaču te da dostavljač uzima novac i vraća novčanik. U pseudokodu, dostavljač izvršava sljedeći kod: Decimal myNumber = osoba.GetPare(35), a osoba u metodi GetPare pristupa objektu novčanik. Ono što ne želimo je da dostavljač radi sljedeće: osoba.GetNovčanik().GetPare(35). Poštivanje Demeterovog zakona je poželjno jer jasnije postavlja odgovornosti objekata.

Poštivanje Demeterovog zakon zahtijeva mnogo delegacije između objekata te smanjuje koheziju razreda pošto oni moraju pružati mnoge metode prosljeđivanja svojim podobjektima ako ne žele davati referencu na te podobjekte.

7. Navedite koje uloge i odgovornosti imaju model, view i controller u MVC arhitekturi i koje su prednosti takve arhitekture.

MVC je design pattern namijenjen odvajanju koda koji reprezentira problem od koda koji prezentira problem korisniku.

Imamo tri uloge:

- Model
    - Sadrži podatke koji predstavljaju problem (objektni model iz DDD)
    - Uz podatke sadrži i ponašanje
- Controller
    - Odgovara na korisnikove akcije govoreći modelu kako da se promijeni
- View
    - Prikazuje trenutno stanje modela korisniku

Prednosti:

- View se može lako zamijeniti ili promijeniti
- Promjene u podacima se reflektiraju u svim sučeljima
- Bolja skalabilnost rješenja zahvaljujući separaciji UI-a i aplikacijske logike
- Pojednostavljeno distribuiranje preko mreže
- Lakše održavanje
- Reusability (poglavito modela)

Problemi:

- Poslovna logika se „ušulja“ u Controller
- Vezivanje između modela i viewa te između modela i controllera
- Veća složenost ako se koristi observer mehanizam
- Nije prilagođen modernim GUI frameworcima (ali je prilagođen za Web)

8. Što podrazumijeva svako od 5 pravila pisanja unit testova (unit test FIRST principles)?

Vidi 3. pitanje.