1. Navesti i kratko opisati atribute dobrog softvera.

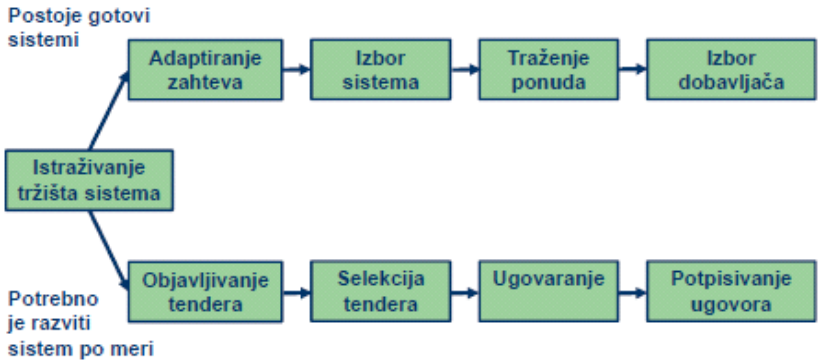
Pogodnost za odrzavanje - treba da je u stanju da moze lako da se menja.

Stabilnost - mora da uliva povrenje sto podrazumeva: pouzdanost, bezbednost i sigurnost.

Efikasnost - mora da ekonomicno koristi resurse sistema.

Upotrebljiost - mora da bude pogodan za koriscenje

2. Graficki ilustrovati i objasniti proces nabavke sistema.



Trazenje sistema koji zadovoljava potrebe organizacije.

Pre nabavke je neophodno izvesti u nekom obliku specijalno i arhitekturno projektovanje sistema:

Potrebna vam je secifikacija ba ste ugovorili razvoj sistema.

Specifikacija vam moze omoguciti kupovinu komercijalnog COTS sistema.
Gotovo je uvek to jeftinije nego razvoj novog sistema.

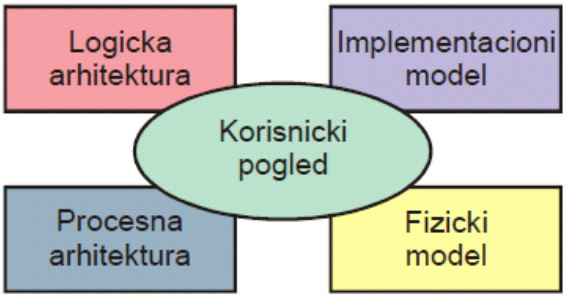
3. Sta je softverski proces i koje su aktivnosti zajednicke za sve softverske procese ?

Softverski proces je skup aktivnosti i pridruzenih rezultata ciji je cilj proizvodnja softvera.

Aktivnost zajednicke za sve softverske procese:

- Specifikacija softvera,
- Razvoj softvera,
- Validacija softvera,
- Evolucija softvera

4. Graficki predstaviti i kratko opisati 4+1 model sistema.



Logicka arhitektura sistema opisuje najvaznije klase u sistemu, njihovu organizaciju u pakete i podisteme kao i organizaciju paketa i podistema u nivoe. Za predstavljaje logicke arhitekture se koristi dijagram klasa.

Procesna arhitektura opisuje najvaznije procese i niti u sistemu i njihovu organizaciju.

Za prikaz impementacionog modela koristi se dijagram komponenti.

Fizicki model opisuje fizicke cvorove u sistemu i njigov razmestajuprostoru.

5. Navesti koji sve sastanci po Scrum-u postoje i kratko ih opisati.

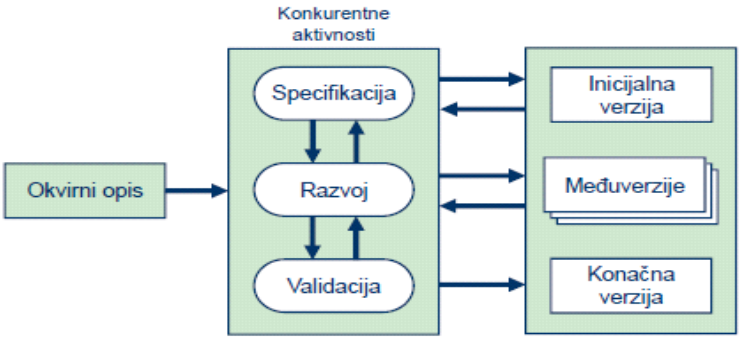
Planiranje sprinta - tim bira stavke iz product backlog-a za koje moze da obaveze da ih moze zavrsiti.

Pregled sprinta - tim prepoznaje ono sto je uradjeno u vreme sprinta.

Retrospektiva sprinta - Radi se posle svakog sprinta i podrazumeva periodicno razmestanje sta je dobro a sta ne, obisno traje 15 do 30 mini ceo tim ucestvuje.

Dnevni Scrum sastanak - dnevno 15 min, stand-up, svi su pozvani ali samo clanovi tima, scrum master i vlasnik proizvoda smeju da pricaju, pomaze da se izbegnu ostali nepotrebni sastanci, svi odgovaraju na tri pitanja: sta si radio juce, sta ces danas raditi, da li ti nestostoji na putu.

6. Graficki ilustrovati i opisati inkrementni razvoj. Navesti osnovne prednosti i nedostatke.



Inkrementni razvoj softvera podrazumeva razvoj koji kao ulazne zahteve ima okvirni opis kranjeg proizvoda. Na osnovu njega se onda kreira inicjalna verzija softvera nakon cega se kruzno vrsi specifikacija, razvoj i validacija medjuversija softvera u

inkrementima tako da se svaka sedeca medjuverzija bude kompletnija, priblizniji finalnom proizvodu i u vecoj meri zadovoljava ralne potrebe zahtevaoca.

Kod Inkrementnog razvoja specifikacija, razvoj i validacija se preklapaju. Moze biti planom vodjen ili agilni.

Prednosti:

Smanjenje cene izmene korisnickig zahteva u toku razvoja - kolicina analiza i dokumentacija koju treba ponovo uraditi mnogo je manja u odnosu na model vodopada.

Olaksano dobijanje povratne informacije od korisnika u toku razvoja.

Moguca brza isporuka korisnog softvera naruciocu.

Nedostaci:

Proces nije vidjiv - menadzerima su potrebne redovne isporuke kako bi merili napredovanje - ako bi se sistem brzo razvijao ne bi bilo efikasno dokumentovati svaku verziju sistema.

Struktura sistema ima tendenciju degradacije dodavanjem novih inkremenata.

7. Navesti faze RUP metodologije i objasniti sta se dobija kao rezultat svake od faza.

RUP faze:

Pocetna faza - analiza problema, razumevanje potreba korisnika, generalno definisanje sitema.

Rezultat ove faze je vizija sistema.

Faza razrade - izrada plana projekta, organizacija i ekipni rad, detaljna definicija zahteva.

Rezultat ove faze su: Plan projekta, Use-case specifikacija i arhitekturni projekat sistema .

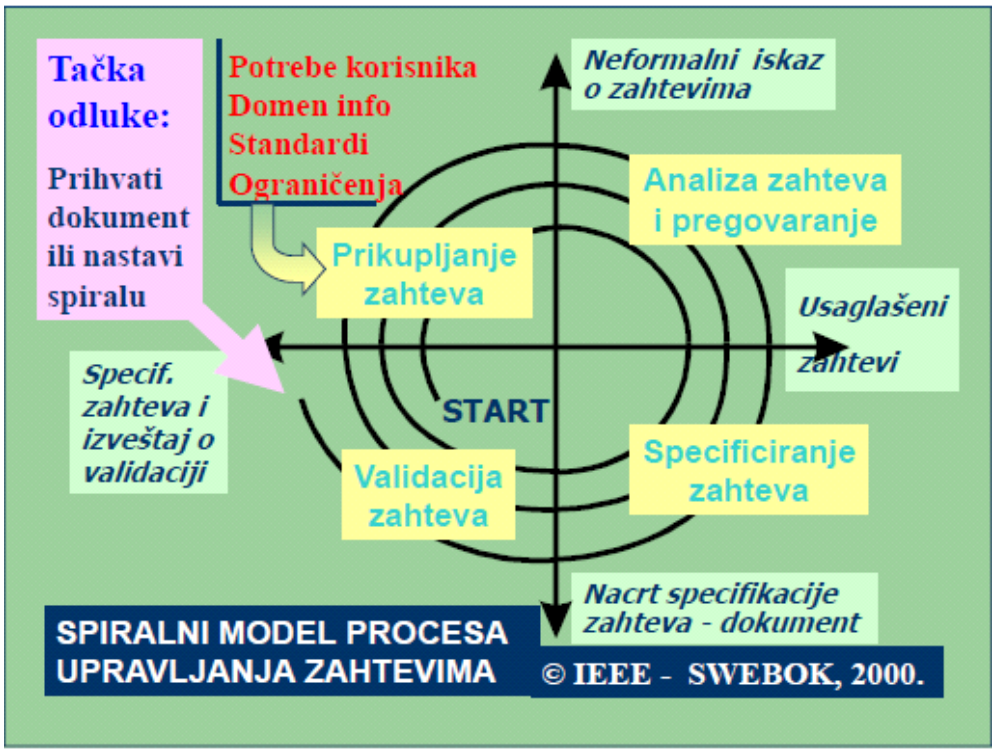
Faza izrade - realizacije sistema, testiranje.

Rezultat ove faze su: Plan testiranja, test specifikacija,detaljni projekat sistema, softverski proizvod.

Faza isoruke - finalizacija softverskog sistema, alfa (beta) testiranje, izrada korisnicke dokumentacije (uputsva), obuka korisnika, uvodjenje sistema kod korisnika.

Rezultat ove faze su: Test izvestj, korisnicko uputsvo, instalacija sitem.

8. Graficki ilustrovati i opisati spiralni model procesa specifikacije zahteva.



Proces upravljanja zahtevima sastoji se od medjusobno nerazdvojivih podprocesa: prikupljanje, analiza, specifikacija, validacija zahteva.

Kvalitetan softver je proizvod vrlo dobro realizovanog dizajna zasnovanog natacnim zahtevima koji su rezultat komunikacije i saradnje izmedju proektanata i kupaca.

Prikupljanje zahteva: glavne kategorije ucesnika - kupac, korisnik, developer. Niko od njih nema kompltnu sliku o softverskom proizvodu. Glavni problem: nedostatak komunikacije.

9. Navesti osnovne princpe agilnog razvoja softvera izrazene kroz Agile Manifesto.

Zadovoljstvo korisnika brzom isporukom korisnog softvera.

Mogucnost promena zahteva pa cak i u podmakloj fazi razvoja.

Cesta isporuka softvera, u razmaku od par nedelja.

Ispravan softver je osnovna mera napredka.

Razvoj koje je u stanju da odrzi konstantan tempo.

Bliska saradnja izmedju projektanata i poslovnih sradnika.

Najbolji tip komunikacije je komunikacija licem u lice,

Projeti se izvode u okruzenjima u kojhem su motivisani pojedinci, u kojem se moze imati poverenja.

Kontinualno usmeravanje paznje ka tehnickoj vestini i dobrom dizajnu.

Jednostavnost.

Samoorganizovani timovi.

Prilagodjavanje promenljivim okolnostima.

10. Use-case metoda i scenariji dogadjaja. Ilustrovat na primeru.

Use-case metode:

Sistem se posmatra sa stanovista korisnika sistema.

Opisuje se slucajevima koriscenja sistema i scenarija ponasanja.

Koristi se uml notacija

Koristi se kod RUP modela razvoja softvera

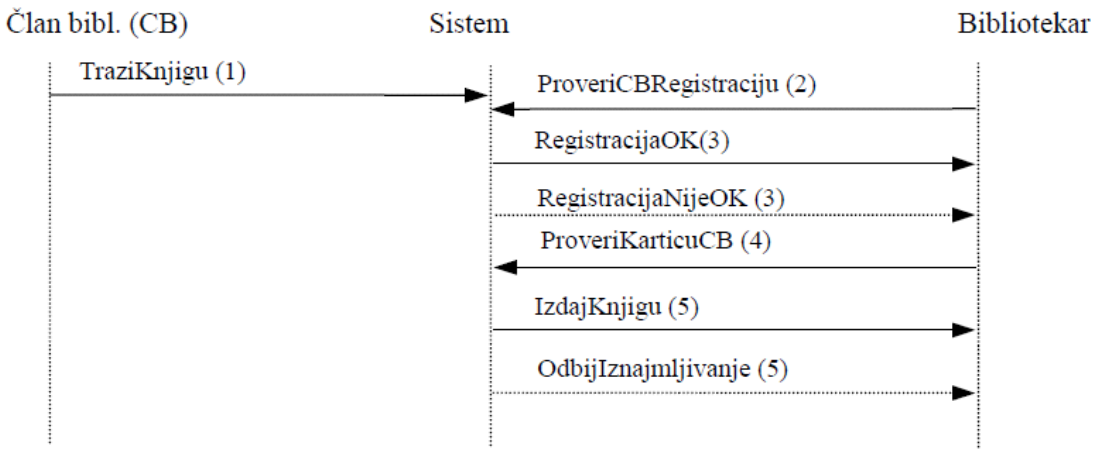
Use-case scenarii dogadjaja:

Servisi objekata mogu biti otkiveni i modeliranjem scenarija dogadjaja za razlicite funkcije sistema.

Dogadjaji se prate do objekata koji reaguju na njih.

TIpican model scenarija dogadjaja je interakcija izmedju korisnika i sstema.

Scenarija su primeri kako ce sistem biti koriscen u realnom zivotu.



11. Navesti i kratko opisati osobine dobro projektovanog softvera.

Osobine dobrog projekta su:

Hijerarhija - dobar projekat bi trebalo da bude organiovan u drobro projektovanu hijerarhiju komponenata

Modularnost - treba izvrsiti dekompoziciju sitema u posebne celine - module sa jasno definisanim interfejsom.

Nezavisnost - treba grupisati slicne stvari u nezavisne module. AKo se kasnije menjaju neke bitne odlike projekta, posledice ce biti lokalizovane za modul u kome su odradjene.

Jednostavan interfejs - treba izbeci komplikovan korisnici interfejs, interfejse sa velikim brojjem mogucnosti upotrebe kao i nterfejse cija bi izmena mogla izazvati nezeljene efekte.

12. Ukratko opisati i navesti prednosti i nedostatke slojevitog arhitekturnog modela.

Slojeviti (layered) model se koristi kod modeliranja intefejsa medju podistemima. Sistem se orgaizuje u skup slojeva od kojih svaki obezvedjuje jedan skup funkcionalnosti sloju iznad i sluzi kao klijent sloju ispod.

Prednosti model:

Promena inerfejsa jednog sloja moze da utice na maksimalno dva sloja.

Laga zamena jedngo sloja drugim ukoliko su im interfejsi identicni

Baziran je na visokom nivou apstrakcije

Nedostaci modela:

Ne mogu svi sistemi lako da se organizuju po ovom modelu

13. Sta predstavlja i koji su ciljevi inspekcije softvera.

Inspekcija softvra predstavlja analizu statickih reprezanticija sistema da bi se otkrili problemi. Cilj inspekcije softvera je ispitivanje izvorne reprezentacije softvera radi otkrivanja anomalija i defekata. Inspecija ne zahteva izvrsenje softvera tako da se moze izvoditi pre njegove implementacija. Moze biti primenjena na bilo koju reprezentaciju softvera a prema istrazivanjima, najefikasnija primena je na slucajeve koriscenja.

14. Navesti, krato opisati i uporediti razlicite metode testiranja softvera.

Metode testiranja se mogu svrstatri u 3 grupe:

Metode crne kutije - Test inzenjer pristupa softveru koji testira preko interfejsa koji je manjenjen kranim korisnicima.

Metode bele kutije - Test inzenjer pristupa izvornom kodu i moze pisati kod koji linkuje sa bibliotekama koje su linkovane sa softverom koji se testira. Obisno se koristi kod komponentnog testiranja.

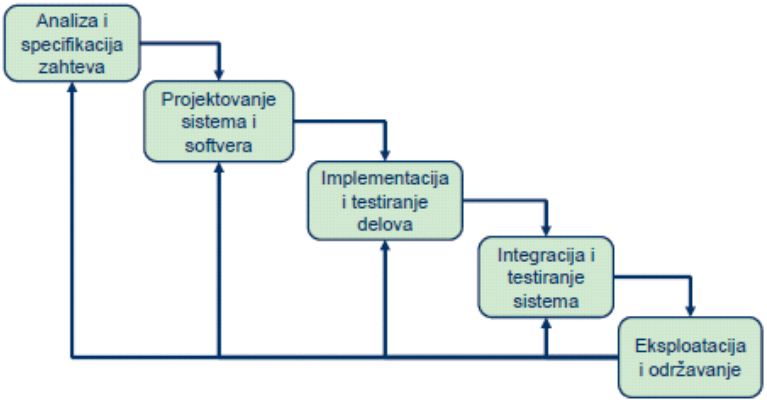
Metode sive kutije - Test inzenjer poze postaviti ili manipulisati nekom o kolinom za testiranje i moze videti stanje softvera posle svake akcije. Koriste ih gotov iskljucivo klijent-server test inzenjeri ili inzenjeri oji koriste bazu podataka kao repozitorijum informacija.

Kriterijumi za ovu kategorizaciju su: da li pri razvju test slucajeva pristup izvornom kodu softvera koji se testira i dali se testiranje vrsi preko korisnickog interfejsa ili preko programskog interfejsa (API).

15. Po cemu se razlikuju softversko inzenjerstvo od informatike.

Nauka o racunarstvu ili informatika se bavi teorijom i osnovama racunarstva dok se softversko inzenjerstvo bavi prakticnom stranom razvoja i isporukom korisnog softvera.

16. Razvoj softvera po modelu vodopata (dijagram, prednosti i problemi).



Model vodopada je planom vodjen model. Potpuno odvojene faze specifikacije i razvoja. Osnovni nedostatak modela je nemogucnost prihvatanja izmene u zahtevima korisnika kad je proces u toku. Jedna faza mora da se zavrsi da bi pocela sledeca, pa izmena zahteva resetuje ceo poces. Model vodopada se najcesce koristi kod velikih sistema gde se razvoj deli na nekoliko lokacija.

17. Navesti osnovne karakteristike kao i prednosti i nedostatke event-driven arhitekturnog modela.

Ovaj model se koristi kod sistema koji su upravljani eksterno generisaim dogadjajima. Postoje dve osnovne grupe ovih modela:

Broadcast modeli i interrupt-driven modeli

Prednosti event-driven modela:

Podrska visestrukom koriscenju softvera.

Laka evolucija sistema.

Lako uvodjenje nove komponente u sistem.

Nedostaci event-driven modela:

Kada komponenta generise dogadjaj ona ne moze da zna da li ce naka komponenta da odgovori na njega i kada ce obrada dogadjaja biti zavrsena.

18. Navesti i kratko opisati kategorije strategija za upravljanje rizikom.

Kategorije strategija za upravljanje rizikom:

Strategija izbegavanja - Smanjuje se verovatnoca pojave rizika npr. ako je rizik da ce neko bitan da nam napusti tim mi se onda zuzetno trudimo da ga zadrizmo, dajemo mu povlastice, povisice i razne benefite sto vecu zelju da ostane tj. kako bi rizik njegovog odlaska bio manji.

Strategija minimizacije - Smanjiti utacaj datog rizika na projekat ili proizvod. Opet ako je rizik nekog odlaska iz tima mi se onda trudimo da u nimu ne postoji niko ko neki posao radi skroz sam i da niko drugi nije upoznat sa tim delom posla, vec se raspolozivo ljudstv tako rasporedjuje tako da ko god da odluci da ode moglo bi da dodje do preraspodele tako da neko drugi pokupi posao koje je tad radio.

Planovi na nepredvidjene dogadjaje - Ukoliko prve dve strategije nisu opcija onda treba napraviti ceo plan sta radit u slucaju da se rizik desi, tj. kako nastaviti sa radom sa prisutnim rizikom. Npr. za slucaj da organizacija naidje na finansijske probleme pri razvoju projekta onda treba ici na ubedjivanje rukovodstva u to da je bas vas projekat bitan i da njega treba ordzati u zivotu.

19. Sta predstavlja verifikaciju a sta validaciju.

Verifikacija je odgovor na pitanje "Da li na pravi nacin gradimo proizvod ?" i ispituje dali je softver usaglasen sa svojom specifikacijom.

Validacija odgovara na pitanje "Da li gradimo pravi proizvod ?" i ispituje da li sotver radi ono sto korisnik stvarno trazi.

Dva glavna cilja validacije i verifikacije su otkrivanje defekata u sistemu i procena da li sitem upotrebljiv i koristan u realnim uslovima.

20. Sta je refactoring a sta reinzenjering, navesti slicnosti i razlike.

Refactoring je proces unapredjenja program kako bi se smanjila njegova degradacija kroz izmene tj. predstavlja "preventivno odrzavanje" koja smanjuje problemi pri buducim izmenam. Refactoring ukljucuje izmenu program kako bi se poboljsala njegova struktura smanjila slozenost ili povecala razumljivost. Kada se raftorise program treba izbegavati dodavanje novig funkcionalnosti, vec se skoncetrisati na njegovo unapredjenje.

Reinzenjering sistema predstavlja restruktuiranje ili ponovno pisanje dela ili celo nasledjenog sistema bez promene njegove funkcionalnosti. Primenjiv je onda kad neki ali ne svi podsistemi veceg sistema zahtevaju cesto odrzavanje.

Reinzenjering utice na sanjenje rizika i smanjenje cene. Kod rzvija novog softvera postoji veliki rizk, a cena razvoja novog softvera je najcesce znatno veca nego njegov reinzenjering.

21. Sta je softver i cime se bavi softversko inzenjerstvo.

Softver je racunarski program, pridruzena dokumentacija i kofiguracioni podaci neophodni da bi softver radio korektno.

Postoje dva osnova tipa softvera:

genericki - samostalni sistem namenjen za prodaju na slobodnom trzistu

ugovorni - razvijni za jednog korisnika prema njegovim zahtevima

Softversko inzenjerstvo je izenjerska disciplina koja obuhvata sve spekte proizvodnje softvera.

22. Navesti i kratko opisati faze u RUP-u. Objasniti odnos faza i iteracija u RUP-u.

RUP faze:

Pocetna faza - analiza problema, razumevanje potreba korisnika, generalno definisanje sitema.

Rezultat ove faze je vizija sistema.

Faza razrade - izrada pana projekta, organizacija i ekipni rad, detaljna definicija zahteva.

Rezultat ove faze su: Plan projekta, Use-case specifikacija i arhitekturni projekat sistema .

Faza izrade - realizacije sistema, testiranje.

Rezultat ove faze su: Plan testiranja, test specifikacija,detaljni projekat sistema, softverski proizvod.

Faza isoruke - finalizacija softverskog sistema, alfa (beta) testiranje, izrada korisnicke dokumentacije (uputsva), obuka korisnika, uvodjenje sistema kod korisnika.

Rezultat ove faze su: Test izvestj, korisnicko uputsvo, instalacija sitem.

Svaka faza moze imati proizvljan broj iteracija isvaka iteracija treba da rezultira izvrsnom verzijom koja se moze testirati.

23. Objasniti osnovne tipove veza kod Use-case dijagrama i ilustrovati jednim dijagramom.

24. Sta predstavlja upravljanje zahtevima.

Upravljanje zahtevima znaci prevodjenje zahteva korisnika u skup njihovih potreba i funkcionalnosti sistema. Ovaj skup se kasnije pretvara u detaljnu specifikaciju funkcionalnih i nefunkcionalnih zahteva. Detaljna specifikacija se prevodi u test prcedure, projekat i korsnicku dokumentciju. Potrebno je definisati proceduru u slucaju promene zahteva korisnika.

25. Sta predstavlja upravljanje projektima i po cemu su softverski projekti osobeni ?

Pod upravljanje projektima se podrazumeva organizovanje, planiranje i raporedjivanje softverskih projekata.

Upravljanje softverskim projektima odnosi se na aktivnosi koje treba osiguraju da se softver isporuci na vreme i po planu, a saglasno zahtevima organizacija koja ga razvija i nabavlja.

Softverski projektni su specificni zbog toga sto:

Proizvod je nevidljiv

Proizvod je fleksibilan

Softversko inzenjerstvo nema status zdrave inzenerske discpline kao sto su masinstvo, energetika, itd.

Proces razvoja softvera nije standardizovan

Mnogi softverski proizvodi su unikatni

26. Opisati i upotrebiti top-down i bottom-up pristupe projektovanja softvera.

Top-down projektovanje softvera:

Polazi se od vrha sistema, odnosno od najvisih slojeva i onda se polako dolazi do podsistema koji su na nizim nivoima. Losa strana ove tehnike ta sto se forsira razvoj pojedinih grna sistema dok neke druge nisu ni zapocete. Takodje, ova tehnika ne sagledava na pravi nacin postojece komponente koje se mogu koristiti.

Bottom-up projektovanje softvera:

Polazi se od dna sistema, odnosno od najnizih slojeva i onda se polako dolazi do podsistema koji su na visim nivoima. Obico se krece od gotovih komponenata koje se povezuju kako bi se realizovali neki delovi sistema. Losa strana ove tehnike je ta sto se najvisi slojevi podsistema dbijaju u kasim fazam implementacije.

27. Opisati strukturu test slucajeva i ilustrovati na jednom primeru.

U softverskom inzinjerstvu test slucaj je skup uslova na osnovu kojih tester moze utvrditi da li softver delimicno ili potpuno sipunjava neki zahtev.

Zatestirnje softvera potreban je veliki broj test sucajeva. RUP preporucuje kreiranje bar po 2 test slucaja za svaki zahtev. TRest slucaj treba da sadrzi opis funkcionalnosti koja se testira i kako treba pripremiti okruzenje da bi bili sigurni sta testiramo.

Struktura test slucaja:

Uvod **-** sadrzi opis informacije o test slucaju

Aktivnost test slucaja - Okruzanje/konfiguracija test slucaja, inicijalizacija, finalizacija, akcija, ulazni podaci

Ocekivani rezultati - opisuje sta ce tester videti posle izvodjenja svih koraka.

28. Navesti i kratko opisati kategorije zahteva.

Funkcija: "Sta" sistem mora da bude sposaban da uradi

Osobine: "Koliko dobro" ce funkcije biti izvrsavane.

Cena: koliko ce kostati kreiranje i drzavanje funkcije njihovih osovina.

Ogranicenja: bilo koja restrikcija u slobodi definisanja zahteva ili dizajnu.

29. Ukratko opisati i navesti prednosti i nedostatke klijent/server arhitekturnog modela.

Klijent/server model se koristi kod distribuiranih sistm.

Sastoji se od skupa stand-aloneservera koji obezbedjuju specificne servise, skup klijenata koji pozivaju te servise i mreze koje omogucavaju udaljeni pristup. Komponente ovog modela su server i klijenti a konektori mreza i servisi servera.

Prednosti ovog modela:

Efikasno koriscenje mreznih sistema.

Omogucava koriscenje slabijeg hardvera na klijentskoj strani, obzirom da server odradjuje vecinu posla.

Lako dodavanje novih servera i nadogradnja postojecih.

Nedostaci ovog modela:

Neefikasna razmena podataka izmedju klijenata (sve medjusobne interakcije klijenata moraju da prodju kroz server).

Redudantnost podataka.

Ne postoji centralni registar imena servera i servisa. Nije lako otkriti koji serveri i servisi su na rasolaganju.