SE311 - L1

1. Šta omogućava apstrakcija unutar projektovanja softvera?

Apstrakcija omogućava uklanjanje detalja iz opisa problema uz zadržavanje suštinskih osobina svoje strukture. Projektant softvera treba da razume i razmišlja o sistemu na apstraktan način kroz događaje, entitete, objekte ili neke druge predmete koji su bitni za sistem. Proces projektovanja retko može biti konvergentan u smislu da bude u stanju da projektanta usmeri ka jednom poželjnom rešenju.

2. Na koji način se vrši podela procesa projektovanja na projektne aktivnosti?

Faze u procesu projektovanja su:
- Zahtevanje rešenja
- Projektovanje modela rešenja
- Evaluacija modela rešenja
- Elaboracija modela rešenja

Često se dešava da postoji nekoliko iteracija između navedenih faza unutar procesa projektovanja kao što je ponovno vraćanje na reviziju izbora projektovanja. Navedeni proces elaboriranja prvobitno definisanog modela može otkriti nedostatke ili čak i potpunu neprilagođenost početnim zahtevima čime se dovodi u pitanje kompletan proces projektovanja.

3. Kako faza održavanja ima uticaj na proces projektovanja softvera?

Proces održavanja softvera podrazumeva da svaka odluka bude unešena u inicijalni originalni model softvera. Informacije o odlukama i načinu izvršenja određene izmene pomažu u sagledavanju kompletnog projektnog rešenja softvera. Glavna motivacija za beleženje razloga donošenja određenih odluka u procesu projektovanja je kontrola kvaliteta, kako u fazi projektovanja, tako i u fazi održavanja softvera.

4. Kako testiranje softvera može ispuniti ciljeve verifikacije i validacije?

Testiranjem prototipova omogućeno je ranije identifikovanje neusaglašenosti u fazama procesa razvoja softvera. Verifikacija projektovanja softvera vrši se na osnovu softverske specifikacije. Provera neusaglašenosti između stvarnih zahteva korisnika i projektovanja softvera naziva se validacija. Razlika između procesa verifikacije i validacije je definisana od strane Boehma, i glasi:
- Verifikacija: da li pravilno proizvodimo proizvod?
- Validacija: da li proizvodimo pravi proizvod?

5. Kako unaprediti proces testiranja softvera tako da odgovara ciljevima verifikacije i validacije procesa projektovanja softvera?

Veoma je bitno da se pronađu greške u ranijim fazama procesa projektovanja. Uočavanje grešaka u kasnijim fazama zahteva dodatnu analizu ispravljanja grešaka i kako ispravljanje neusaglašenosti utiče na druge delove softvera. Verovatnoća da se neusaglašenost uoči tokom testiranja softvera je veća nego u toku između faza specifikacije i projektovanja. Upotrebom i testiranjem prototipova omogućeno je ranije identifikovanje neusaglašenosti u fazama procesa razvoja softvera.

6. Šta se podrazumeva pod razmenom znanja projektanta sa drugim učesnicima u procesu projektovanja softvera?

Projektant softvera treba da poseduje tri značajne karakteristike:

I. Poznavanje domena aplikacije omogućiće jednostavno mapiranje strukture problema i strukture potencijalnog rešenja
II. Veštine u prenošenju tehničke vizije projektnog rešenja drugim učesnicima procesa projektovanja
III. Identifikovanje tehničkog napretka projekta

Proces projektovanja softvera obuhvata dve komponente:

- **Deo predstavljanja**: Sadrži set opisnih oblika modela problema i opis strukturnih osobina rešenja za moguće implementatore
- **Deo procesa**: Sadrži opis izvršavanja neophodnih transformacija modela u delu predstavljanja

Unutar dela predstavljanja vrši se identifikovanje osobina objekta u proces projektovanja. Deo procesa bavi se:

- Identifikovanjem projektnih aktivnosti koje je potrebno obaviti
- Procedurama za vršenje transformacija
- Merenjem kvaliteta
- Operacijama verifikacije i validacije

7. Šta su kanali komunikacije projektanta softvera?

U procesu projektovanja softvera, potrebno je da projektant softvera ima određeni stepen znanja o domenu kao početni vid informacija koji je potreban za obavljanje bilo kog konkretnog zadatka u procesu projektovanja. Određene informacije može dobiti iz specifikacije, ali to često nije dovoljno.

8. Koja je uloga ograničenja u procesu projektovanja softvera?

Ograničenja služe da ograniče ukupni prostor mogućeg rešenja projektantu softvera. Ograničenja su specifična za svaki novi identifikovani problem i potrebno ih je razmotriti u svakoj fazi projektovanja softvera. Praćenje ograničenja je moguće kroz redovne preglede i revizije faza procesa projektovanja.

9. Šta je metoda prepoznavanja?

Metoda prepoznavanja je metoda u okviru koje projektant softvera prepoznaje rešenje problema koje je bilo u samom identifikovanom problemu.

10. Na koji način proces projektovanja softvera unapređuje proces razvoj softvera?

//

11. Šta je dugoročni plan izmena? U kojim slučajevima se koristi?

Često se u toku korišćenja softvera javi potreba za određenim modifikacijama i unapređenjima softvera. Planiranje dugoročnih izmena u procesu projektovanja zavisi od nekoliko faktora.

Jedan od faktora je budžet. Ukoliko održavanje nije unapred planirano, često se dešava da programeri nemaju podsticaj za izvršavanje izmena u toku procesa održavanja.
Drugi faktor je nemogućnost projektanata softvera da stvore plan razvoja i održavanja softvera u budućnosti.

12. Objasniti da li proces projektovanja usložnjava razvoj softvera ili ga čini jednostavnijim?

//

13. Šta je model rešenja?

Model rešenja je prikaz softverskog sistema na osnovu prethodno definisanih funkcionalnih i nefunkcionalnih zahteva. Model ne mora biti konačan ali treba da sadrži sve zahteve koji su dobijeni od strane naručioca softvera. Model rešenja može da se predstavi preko dijagrama arhitekture i drugih UML dijagrama, ukoliko je to potrebno za detaljni prikaz modela rešenja.

14. Kako može doći do neusaglašenosti u procesu projektovanja softvera?

Neusaglašenost u okviru procesa projektovanja može biti otkrivena u različitim fazama:

- Kada se vrši preraspodela unutar procesa projektovanja i upoređivanje informacije koje su dobijene iz različitih gledišta na softver
- U toku procesa razvoja softvera
- U toku sprovođenja implementacije softvera

Verovatnoća da se neusaglašenost identifikuje tokom testiranja softvera je veća nego u toku između faza specifikacije i projektovanja. Upotrebom i testiranjem prototipova omogućeno je ranije identifikovanje neusaglašenosti u fazama procesa razvoja softvera.

SE311 - L2

1. Kada je potrebno ostvariti visoke performanse softvera, na koji način je to moguće izvršiti kroz primenu softverske arhitekture?

Za skoro sve sisteme, atributi kvaliteta kao što su performanse pouzdanost, bezbednost i izmenljivost su jako bitni u smislu omogućavanja pravog odgovora na zadatke korisnika. Sposobnost softverskih sistema da proizvede tačan rezultat nije od pomoći korisniku ukoliko sistemu treba dugo vremena da proizvede rezultat ili sistem ima problem sa isporukom rezultata.

Ukoliko je potrebno ostvariti visoke performanse softvera, potrebno je:
- Ispitati potencijalni paralelizam kroz dekompoziciju sistema i sinhronizaciju sistema
- Identifikovati potencijalna uska grla performansa

2. Šta predstavlja cilj pisanja softverske dokumentacije?

Dokumentacija arhitekture je obavezna i opisna. Zavisno od čitaoca, ovaj vid dokumentacije propisuje šta bi trebalo da bude urađeno uz postavljena ograničenja na odluke koje treba donositi. Proces planiranja dokumentacije i kasnije analize zahteva da se omogući podrška mogućim izmenama i unapređenjima kako bi se kompletan softver ispratio na pravi način.

3. Na koji način se određuju korisnici dokumentacije softverske arhitekture?

Glavni korisnik softverske dokumentacije arhitekture je projektant. U budućnosti, ukoliko projektant softvera bude ista osoba koja je pisala dokumentaciju ili na to mesto dođe novi projektant, uvek će imati dokumentaciju koje može koristiti. Novi projektanti softvera koji su zainteresovani za razumevanje softvera, analizu eventualnih nedostataka i donešenih odluka u određenom tenutku procesa projektovanja mogu to jednostavno izvršiti.

4. Šta omogućavaju različiti pogledi na softversku arhitekturu?

Različiti pogledi omogućavaju uvid u različite atribute kvaliteta. Atributi kvaliteta utiču na izbor pogleda iz kog je potrebno izvršiti dokumentovanje softverske arhitekture. Pregled softverske arhitekture po slojevima može prikazati prenosivost i omogućiti razumevanje performansi i pouzdanosti sistema.

5. Šta se postiže dokumentovanjem različitih pogleda na arhitekturu softvera?

Različiti pogledi ističu različite sistemske elemente ili veze između elemenata. **Bitno je naglasiti da nijedan pogled ne može u potpunosti predstaviti arhitekturu.** Često je potrebno prikazati više različitih pogleda na softversku arhitekturu kako bi dokumentacija ispunila očekivanja od članova tima sa različitim ulogama. Suština pogleda softverske arhitekture je da prikaže samo informacije koje su potrebne određenom članu tima ili izvršavanju određenog zadatka u toku razvojnog procesa.

6. Za koje aktivnosti u toku procesa razvoja softvera koristimo dokumentaciju softverske arhitekture?

Analizom aktivnosti i zadataka u procesu razvoja softvera, projektant softvera može identifikovati kojim aktivnostima dokumentacija omogućava moguće smanjenje troškova ispravljanja greške. Tu najčesće spadaju aktivnosti kao što su kodiranje, reinženjering ili testiranje u kojima će ušteda troškova biti značajna.

7. Koji koraci su potrebni da bi se izvršilo kombinovanje različitih stilova softverske arhitekture?

Recimo da provajder usluga u servisno-orijentisanom sistemu može biti nepoznat drugim provajderima, usluga ili korisnicima usluga primenom višeslojnog stila. Dokumentovanje stila arhitekture koja opisuje celokupan sistem uz višeslojni prikaz servera na kome se sistem nalazi, videće rešenje softverske arhitekture.

8. Šta je element softverske arhitekture?

Element softverske arhitekture predstavlja rezultat pažljivog i preciznog projektovanja u cilju ispunjavanja zahteve za kvalitet softvera, a i zahteve nametnute od strane logike poslovanja organizacije za koju se softver projektuje

9. Zašto je potrebno pisati softversku dokumentaciju iz ugla čitaoca?

Ukoliko svakom čitaocu bude trebalo u proseku dva minuta kako bi razumeo rečenicu, to u velikoj meri može uticati na vreme potrebno da prosečan čitalac razume celokupno dokumentaciju. Potrebno je pisati što konciznije i preciznije. Ukoliko je čitaocu dokument razumljiv, čitaće dokument svaki put kada mu bude potrebna pomoć u razumevanju softvera, a ako je nerazumljiv, onda ga neće koristiti naredni put.

10. Kako primena pravila za pisanje softverske dokumentacije utiče na proces pisanja softverske dokumentacije?

Sedam pravila dokumentacije služe kako bi prikazali smernice za pisanje i čitanje tehničke dokumentacije. Ova pravila daju objektivne kriterijume za ocenjivanje kvaliteta dokumentacije.

Sedam pravila za pisanje su:

I. Pisati dokumentaciju iz ugla čitaoca
II. Izbegavajte nepotrebno ponavljanje
III. Izbegavajte dvosmislenost
IV. Koristite standardnu organizaciju (šablon)
V. Snimiti obrazloženje
VI. Pregledati dokumentaciju

11. Na koji način se vrši odabir elemenata koji se predstavljaju u softverskoj dokumentaciji?

Postoje svojstva koja se koriste za analizu arhitekture. Ukoliko je potrebno analizirati performanse dokumentovane arhitekture, onda svojstva treba da imaju vreme odgovora najboljeg i najlošijeg slučaja, maksimalan broj događaja koje određeni element sistema može da obradi u jedinici vremena. Analiza bezbednosti arhitekture podrazumeva svojstva nivoa šifrovanja, pravila autorizacije za različite elemente i odnose.

12. Koje su prednosti pisanja dokumentacije u toku procesa razvoja softvera?

U procesu razvoja softvera, često je prisutna i rečenica: “Uradiću kvalitetnu dokumentaciju arhitekture softvera ukoliko budem imao slobodnog vremena za to”.

U tom slučaju, menadžeri projekta moraju da insistiraju na dokumentovanju softverske arhitekture koju je potrebno proizvesti tokom procesa razvoja softvera. Menadžeri su uglavnom voljni da ulažu u resurse u aktivnostima koje donose korist.

Cena izrade i održavanja dokumentacije arhitekture, na osnovu pretpostavki, trebao bi da bude pokriven izbegavanjem mogućih grešaka u procesu implementacije softvera.

13. U kojim slučajevima se koristi formalna notacija?

Za formalne notacije, pogledi su opisani u notaciji koja ima preciznu semantiku. Formalna analiza je u tom slučaju moguća. U nekim slučajevima, dešava se da su formalne notacije specijalizovane za određene stilove. Određivanje vrste notacije za određenu upotrebu zahteva i nekoliko ustupaka u procesu projektovanja. Formalne notacije zahtevaju dosta više vremena i napora projektanta softvera, ali doprinose smanjenju dvosmislenosti i većoj mogućnosti kasnije analize.

14. Da li možete identifikovati neke nedostatke pisanja softverske dokumentacije?

//

SE311 L03 ispitna pitanja

3.1. Kada se koristi izraz modul softvera?
3.2. Šta omogućavaju različiti pogledi na softversku arhitekturu?
3.3. Koja je osnovna relacija stila razlaganja?
3.4. Na koji način je stil razlaganja pogodan za nove članove razvojnog tima?
3.5. Po čemu se stil upotrebe razlikuje od stila razlaganja?
3.6. Šta podrazumevaju inkrementalni podskupovi modula?
3.7. Koje su prednosti stila generalizacije na raspolaganju projektantu softvera?
3.8. Na koji način je omogućeno "produžavanje" modula primenom stila generalizacije?
3.9. Kako se prikazuje softverska arhitektura kroz stil slojeva?
3.10. Koji stilovi softverske arhitekture su pogodni za korišćenje uz stil slojeva?
3.11. Šta su aspekti?
3.12. Koje informacije sadrži prikaz aspekta u softverskoj dokumentaciji?
3.13. Kada se koristi model podataka?
3.14. Koji su tipovi modela podataka?
3.15. Da li primena stilova arhitekture usložnjava ili olakšava proces projektovanja softvera projektantu softvera?

3.1

Projektanti softvera koriste izraz modul za objašnjavanje softverske strukture uključujući jedinice programskog jezika kao što su: C programi, Java ili C# klase, PL/SQL procedure ili opšte grupacije programskog koda kao što su Javapaketi. Modulima, kao što je već navedeno, moguće je dodeliti svojstva, definisati odgovornost u softveru i analizirati karakteristike. Moduli se mogu dodatno razložiti na nove module ili spojiti u jedan modul. Slojeviti stil identifikuje module i povezuje ih na osnovu relacije koja dozvoljava takav način korišćenja (allowed-to-use-relation) dok stil generalizacije identifikuje module na osnovu zajedničkih osobina

3.2

Svojstva modula pomažu prilikom procesa implementacije ili analize za prikazivanje određenog modula.

Lista svojstava određenog modula može da varira ali uglavnom sadrži sledeće:

• Ime: Ime modula predstavlja osnovnu informaciju za svaki modul softvera.Ime može biti povezano sa ulogom modula u softveru, ili pozicijom na kojoj se nalazi unutar

strukture (primer: "A.B.C" se odnosi na modul C koji je podmodul modula B a koji je sam podmodul modula A)

• Odgovornost: Svojstvo odgovornost modula predstavlja način identifikovanja uloge modula u softveru i uspostavljanje identiteta modula pored njegovog imena. Odgovornost mora biti opisana sa dovoljno detalja kako bi čitaocu bilo jasno šta koji modul unutar softvera radi.

• Vidljivost interfejsa: U slučaju da modul sadrži određene podmodule, interfejsi podmodula mogu imati svoje namene, tako da interfejs podmodula može biti nezavisan od interfejsa modula kome pripada.

• Informacije o implementaciji: Moduli predstavljaju jedinice implementacije i potrebno je navesti informacije u vezi njihove implementacije sa stanovišta upravljanja razvojem navedenih jedinica i razvojem softvera koji ih sadrži.

3.3

Uzimanjem elemenata i svojstava kroz pogled modula uz fokus na relaciju "je deo od" dobijamo stil razlaganja (dekompoziciju). Stil razlaganja opisuje strukturu programskog koda u vidu modula i podmodula i podelu sistemskih odgovornosti na njih.

3.4

Novi članovi se mogu fokusirati samo na određeni deo sistema od koga žele da krenu i zatim pratiti veze sa ostalim identifikovanim modulima sistema. Ne moraju znati kompletan sistem ili sve funkcionalnosti sistema da bi mogli da rade na implementaciji ili modifikaciji određenog modula.

3.5

Stil razlaganja se može predstaviti kroz pogled modula uz fokus na relaciju „je deo od“ dok se kod stila upotrebe koristi relacija „zavisi od“. Ispravnost stila upotrebe zavisi od ispravnosti drugih modula.

3.6

Definisanje inkrementalnih podskupova podrazumeva da moduli moraju biti pravog nivoa granularnosti.

3.7

Element generalizacije je modul. Relacija koja se koristi je generalizacija, konkretno "je" relacija. Stil generalizacije može biti primenjen na različite tipove softverske

arhitekture. Generalizacija može biti korišćena za: Objektno-orijentisano projektovanje, Produžavanje, Lokalna promena ili varijacija, Ponovna upotreba.

3.8

Produžavanje. Često je lakše razumeti modul koji je drugačiji od drugog poznatog modula nego razumeti nov modul od nule. Generalizacija predstavlja mehanizam za proizvodnju inkrementalnih opisa u cilju formiranja potpunog opisa modula.

3.9

Slojeviti stil kao i svi stilovi modula vrši podelu softvera na jedinice. Jedinice u ovom stilu predstavljaju slojeve. Svaki sloj predstavlja grupu modula koji su povezani i omogućavaju povezani skup usluga. Odnosi između slojeva moraju biti jednosmerni. Slojeviti prikaz arhitekture softvera jedan je od često korišćenih prikazana u dokumentaciji. Korišćena relacija ovog sloja je dozvoljeno da koristi (" allowed to use").

3.10

Veza sa drugim stilovima: Pored dijagrama slojevitog stila često se koriste drugi dijagrami: 1. Modul razlaganja. Slojevi u slojevitom pogledu su uvek povezani a sloj može da prikaže više od jednog modula. Dva podmodula mogu biti deo različitih slojeva. U slučaju da se modul prikazuje na više slojeva potrebno je to naglasiti korišćenjem različite boje na grafičkom prikazu sloja. 2. Položaj. Slojevi se često prikazuju sa nivoima što može zbuniti čitaoca dokumentacije. Višeslojni stil je stil komponente i konektora jer se skupljaju i prikazuju komponente izvršavanja.

3.11

Stil aspekta omogućava izolaciju određenog modula u softverskoj arhitekturi. Kada se vrši implementacija softverskih modula, logika je da se moduli izdvoje tako da svaki modul bude odgovoran za određenu funkcionalnost. Tako, na primer, ukoliko je potrebno napraviti bankarski sistem, moduli koji će činiti taj sistem su: "Račun", "Korisnik", "ATM". Modul "Račun" u ovom slučaju sadržao bi podatke koji se odnose na poslovnu logiku (otvaranje ili zatvaranje naloga, depozit, transfer).

3.12

Elementi unutar stila aspekta su aspektni moduli. Odnos koji se koristi između aspekata naziva se unakrsni rez. Aspekt vrši unakrsni rez modulu ukoliko aspekt sadrži funkcionalnost unakrsnog reza ka samom modulu. Aspekt može sadržati ista svojstva kao i regularan modul. Takođe, može sadržati i svojstva koja opisuju modul koji je cilj aspekta.

3.13

Izlaz iz procesa modeliranja predstavlja model podataka koji opisuje statičku strukturu informacija u vidu entiteta podataka i njihovih međusobnih odnosa (relacija). U bankarskom sistemu obično se koriste entitet Račun, Korisnik i Zajam. Modul Račun može imati nekoliko atributa kao što su broj računa, tip, status i trenutno stanje. Veza može prikazati da kupac može imati jedan ili više računa i da jedan račun može biti povezan sa jednim ili dva korisnika.

3.14

Model podataka se često crta upotrebom ERD dijagrama ili UML klasnog dijagrama. Tipovi modela podataka mogu biti:

• Konceptualni. Konceptualni model podataka se fokusira na entitete i njihove veze.

• Logički. Logički model podataka predstavlja evoluciju konceptualnog modela podataka kroz tehnologiju upravljanja podacima (kao što su relacione baze podataka).

• Fizički. Fizički model podataka vrši implementaciju entiteta podataka. Omogućava optimizaciju, kreiranje identifikacionih ključeva i indeksa i vrši optimizaciju performansi.

3.15

Primena stilova arhitekture jeste složen proces i potrebno je dosta vremena za uspešno projektovanje sistema, međutim primenom stilova se olakšava ostatak posla tako da je vreme koje je utrošeno na primenu malo u odnosu na olakšanje koje donosi.

Lekcija 4

1. Na koji način se u neformalnoj notaciji predstavljaju komponente i konektori?

Kao i za druge poglede softverske arhitekture, i za pogled komponente i konektora neformalna notacija podrazumeva upotrebu kutija i linija, gde kutije predstavljaju komponente a konektori su predstavljeni linijama. Takođe, moguće je dodatno obeležiti komponente i konektore unutar neformalne notacije i tako unaprediti prikaz softverske arhitekture. Pored nazivanja komponenti potrebno je precizirati i njihovo značenje. To je moguće izvršiti u dodatnom opisu na dijagramu

2. Koja svojstva ima elemenat komponenta u pogledu komponente i konektora softverske arhitekture?

Bitno je naglasiti da svaka komponenta ima ime na osnovu koje je moguće odrediti funkciju komponente. Takođe, ime može omogućiti jednostavnije dokumentovanje grafičkog prikaza. Komponente imaju interfejse koji se nazivaju portovi. Port predstavlja specifičnu tačku potencijalne interakcije komponente sa okruženjem. Najčešće ima eksplicitan tip koji definiše različite načine interakcije. Komponenta može imati više portova istog tipa. Takođe, moguće je da komponenta ima nekoliko portova istog tipa koji će upravljati ulaznim podacima ili komponenta server može imati više portova koji se koriste za omogućavanje interakcije sa klijentima Komponenta unutar komponente i konektor pogleda može predstavljati i kompleksan podsistem koji se i sam može predstaviti kroz navedeni pogled. Podarhitektura može biti predstavljena grafički u slučaju da nije previše kompleksna

3. Kada se koristi stil toka podataka?

Primena ovih stilova je u okviru domena gde se obavlja obrada podataka kroz nekoliko transformacionih koraka.
Stilovi toka podataka omogućavaju model u kome se komponente prikazuju kao transformatori podataka a gde konektori prenose podatke iz izlaza jedne komponente na ulaz druge komponente. Svaka vrsta komponente u stilu toka podataka ima određeni broj ulaznih i izlaznih portova. Posao komponente je da obrađuje podatke dobijene na ulazne portove i prosleđuje ih na izlazne portove. Najpoznatiji stil toka podataka je stil cevi i filtera.

4. Koja je osnovna razlika između klijent-server stila i stila peer-to-peer?

Stil peer-to-peer omogućava da komponente direktno komuniciraju kao komponente na istom nivou uz razmenu usluga.
Stil klijent-server predstavlja sistemski prikaz koji razdvaja klijentske aplikacije od usluga koje koriste.

5. Šta podrazumeva grafičko predstavljanje pogleda?

Predstavljanje dijagrama kroz softversku dokumentaciju podrazumeva tekstualni opis svih identifikovanih elemenata. Pored glavnog dokumenta, dodatna dokumentacija treba da objasni dostupnost sistema ili eventualno izvrši razlaganje sistema na podsisteme.
Grafičko predstavljanje i dokumentovanje pogleda komponenta i konektor:
• deluje kao ključ za prateću dokumentaciju gde se mogu naći detalji o elementima, relacijama i njihovim svojstvima
• unutar dijagrama koristi se samo ključ predviđen za pogled komponenta i konektor
• prikazuje broj i vrstu interfejsa na svojim komponentama i konektorima
• koristi apstrakcije komponenti i konektora koji mogu imati složenu implementaciju

6. Koja je svrha elementa konektor u pogledu komponente i konektora?

Konektori su druga vrsta elemenata u pogledu komponenta i konektor softverske arhitekture. Primeri konektora su: poziv za servisiranje, asinhrone poruke, tokovi podataka. Konektori često predstavljaju složene oblike interakcije kao što je recimo kanal komunikacije orijentisan ka transakcijama između servera baze podataka i klijenta ili magistrala servisnih usluga koja posreduje u interakcijama između usluga korisnika i provajdera.
Konektori imaju uloge koji su njihovi interfejsi koji definišu načine na koje konektor može koristiti komponente za obavljanje interakcije.

7. Koja su svojstva koja se koriste u pogledu komponente i konektora?

Ukoliko je potrebno izvršiti analizu performansi sistema na osnovu pogleda komponente i konektora, projektant može dodati svojstva koja se tiču kapaciteta, latence ili prioriteta.Najčešća svojstva koje se mogu koristiti su:
• **Pouzdanost**. Koristi se za utvrđivanje ukupne pouzdanosti sistema. Primer pitanja u okviru analize pouzdanosti može biti: "Koja je verovatnoća prestanka rada komponente ili konektora?"
• **Performanse**. Koristi se za analizu sistemskih mogućnosti, testiranje potrebnog vremena za odgovor na određeni zadatak ili za utvrđivanje propusnog opsega određenog elementa softverske arhitekture.
• **Potrebni resursi**. Koristi se za utvrđivanje potrebnog hardvera za određeni sistem. Primer analize potrebnih resursa može se bazirati na pitanju: " Koje su potrebe obrade i skladištenja podataka komponente ili konektora?".
• **Funkcionalnost**. Koristi se za razumevanje kompletnih funkcija i mogućnosti unutar sistema. Primer analize funkcionalnosti može biti: "Koje funkcije obavlja element (bilo da se radi o

komponenti ili konektoru)?"
• **Bezbednost**. Koristi se za određivanje potencijalnih ranjivih tačaka sistema u okviru pogleda komponenta i konektor.
• **Konzistentnost**. Koristi se u procesu analize ili simulacije performansi komponenti i za identifikovanje mogućih blokada u sistemu.
• **Nivo**. Koristi se za definisanje procedure izgradnje sistema i raspoređivanje komponenti ili konektora u softverskoj arhitekturi.

8. Kada se koristi stil komponente i konektora za predstavljanje softverske arhitekture?

Elementi stila cevi i filtera su cevi i filteri na osnovu kojih se vrši predstavljanje softverske arhitekture. Upotreba stila cevi i filtera najčešće je u sistemima za transformaciju podataka gde se ukupna obrada može razvrstati u niz nezavisnih koraka gde je svaki korak odgovaran za inkrementalnu transformaciju svojih ulaznih podataka. Na taj način, filteri mogu primati podatke od senzora filtrirajući ih u podatke iz kojih mogu biti generisani određeni izveštaji za korisnike. Analize koje je moguće izvršiti upotrebom stila cevi i filtera mogu pokazati rezultate koji se tiču performansi, propusnosti sistema, protoku informacija na ulazu i izlazu sistema

9. Koji pogledi na softversku arhteikturu mogu biti kombinovani sa pogledom komponente i konektora?

- STIL POZIV - POVRATAK SOFTVERSKE ARHITEKTURE Primeri stilova poziv-povratak su klijent-server, peer to peer i REST stilovi.
- STIL SOFTVERSKE ARHITEKTURE ZASNOVAN NA DOGAĐAJU Stilovi zasnovani na događajima omogućavaju komunikaciju između komponenata korišćenjem asinhronih poruka
- STIL PEER-TO-PEER SOFTVERSKE ARHITEKTURE Stil peer-to-peer omogućava da komponente direktno komuniciraju kao komponente na istom nivou uz razmenu usluga.
- Stil klijent-server
- Stil cevi i filtera

10. Koji stilovi softverske arhitekture spadaju u pogled komponente i konektora?

• konektori klijenta i servera - omogućava set klijenata koji dobijaju podatke kroz servisne zahteve od servera. Ovakav pristup omogućava prebacivanje podataka sa glavnog servera na rezervni u slučaju otkazivanja
• konektor za pristup bazi podataka - podržava transakcijski, autentifikovan pristup za čitanje, pisanje i praćenje baze podataka
• konektor za "objavljivanje-pretplatu" (publish-subscribe) - omogućava obaveštavanje (na osnovu pretplaćivanja) i objavljivanje asinhronih događaja

11. Šta je pružalac usluga? Koja su njegova svojstva?

Komponente koje su podnosioci zahteva se nazivaju klijenti a komponente pružaoci usluga su serveri. Serveri mogu pružati skup usloga preko jednog ili više portova. Neke komponente mogu delovati kao klijenti i serveri u softverskoj arhitekturi. Pružaoci usluga i potrošači usluga mogu da rade na različitim platformama i da se integrišu u različite sisteme i stare sisteme. Komponente kao što su registar ili ESB takođe omogućavaju dinamičku rekonfiguraciju koja može biti korisna kada postoji potreba da se zamene ili dodaju nove komponente bez prekida rada sistema.

12. Koji su elementi sistema koji koristi klijent-server softversku arhitekturu?

- Klijent, predstavlja komponentu koja poziva servise serverske komponente
- Server, predstavlja komponentu koja omogucava servise klijentskoj komponenti. Svojstva se razlikuju u zavisnosti od planova projektanta softvera ali najcesce ukljucuju informacije o prirodi portova servera i karakteristike koje uticu na performanse.
- Konektor za upit/odgovor, koristi ga klijent za pozivanje servisa na serveru. Konektori za upit/odgovor imaju dve uloge: ulogu upita i ulogu odgovora. Svojstva konektora ukljucuju tip poziva i tip podataka, da li su podaci sifrovani ili ne.

Lekcija 5

1. Šta se dobija specijalizacijom stilova alokacije?

Specijalizacija stila alokacije se vrši u slučaju potrebe za ponovnim korišćenjem stila u različitim delovima određenog sistema. Drugi stilovi alokacije su takođe mogući, moguće je definisati zahteve za stilove alokacije koji povezuju sistemske zahteve sa softverskim elementima arhitekture. Primer za tako nešto može biti i specijalizacija određenog stila alokacije:
• **Specijalizacija stila raspoređivanja**: Microsoft je razvio šablon nazvan "Tiered Distribution" koji propisuje raspodelu softverskih komponenti u višeslojnoj arhitekturi na hardverske elemente koji se koriste za pokretanje. Opisani šablon omogućava generički 18 stil primene. Takođe, kompanija IBM ima svoju verzije šablona kao što je: topologija jedne mašine (stand-alone server), topologija vertikalnog skaliranja (vertical scaling topology), topologija horizontalnog skaliranja (horizontal scaling topology) itd. • **Specijalizacija stila dodeljivanja radnih zadataka**: Moguće je dokumentovati korišćene šeme dodele radnih zadataka u timu kroz specijalizaciju stila dodele radnih zadataka. Specijalizacija stila može biti izvršena kroz:
-Stil platforme. Postavljanje procesa razvoja softvera u obliku proizvodne linije gde određeni deo razvojnog tima proizvodi objekte koji se mogu ponovo koristiti a drugi deo razvojnog tima ponovo koristi tako razvijene objekte u procesu razvoja softvera.
-Stil kompentencije. Radni zadatak se dodeljuje određenim delovima tima u zavisnosti od

tehničkog znanja. Na primer, projektovanje korisničkog interfejsa je izvršeno u onom delu tima gde se nalaze stručnjaci za projektovanje korisničkog interfejsa.

-Stil otvorenog koda. Većina nezavisnih saradnika doprinosi razvijanju softverskog proizvoda u skladu sa strategijom za tehničku integraciju. Centralizovana kontrola je minimalna osim u slučaju kada nezavisni saradnik integriše svoj programski kod u softverski proizvod.

2. U kojim stilovima alokacije je potrebno prikazivati hardverske elemente okruženja softvera?

- **Stil rasporedjivanja**
  opisuje povezanost softverskih komponenti i konektora sa hardverom kompjuterske platforme na kojoj se softver izvršava
- **Stil instalacije**
  opisuje povezanost između softverskih komponenti i struktura fajl sistema u produkcionom okruženju softvera

3. Koja je osnovna relacija stila alokacije?

Primena stilova alokacije zahteva korišćenje definisanih elemenata, relacija i svojstava. Relacija koja se koristi u stilu alokacije je dodeljen prema ("allocated to"). Stilove alokacije moguće je primeniti i obrnutim tokom, tako što se elementi okruženja softvera povezuju sa komponentama softverske arhitekture. Element softverske arhitekture može biti dodeljen ka više elemenata okruženja softvera a više elemenata softverske arhitekture može biti dodeljeno jednom elementu okruženja softvera

4. Na koji način se vrši specijalizacija stila raspoređivanja?

Specijalizacija stila raspoređivanja: Microsoft je razvio šablon nazvan "Tiered Distribution" koji propisuje raspodelu softverskih komponenti u višeslojnoj arhitekturi na hardverske elemente koji se koriste za pokretanje. Opisani šablon omogućava generički 18 stil primene. Takođe, kompanija IBM ima svoju verzije šablona kao što je: topologija jedne mašine (stand- alone server), topologija vertikalnog skaliranja (vertical scaling topology), topologija horizontalnog skaliranja (horizontal scaling topology) itd.

5. Zašto je neophodno kombinovati različite poglede na softversku arhitekturu?

Osnovni principi dokumentovanja softverske arhitekture predstavljaju grupu različitih pogleda koji omogućavaju detaljan prikaz softverske arhitekture. Često se dešava da tako odabrani pogledi nemaju zajedničke elemente ili relacije sa drugim pogledima pa tako čitaoci projektne dokumentacije nemaju uvid u ono što je projektant kroz poglede hteo da prikaže. Kako su

pogledi na softversku arhitekturu delovi iste softverske arhitekture i postoje kako bi čitaocu detaljno prikazali softver većina pogleda ima međusobnu povezanost sa drugim pogledima

6. Da li je moguće kombinovati bilo koji pogled softverske arhitekture sa nekim drugim pogledom?

Prilikom razmatranja kombinovanog pogleda, potrebno je proveriti da li je asocijacija između elemenata jasna. U suprotnom, pogledi verovatno nisu dobri za kombinovanje jer će krajnji rezultat biti kompleksan i zbunjujući pogled. U tom slučaju bi bilo bolje upravljati asocijacom pojedinačno kroz poglede

Povezivanje različitih pogleda na softversku arhitekturu moguće je korišćenjem asocijacija. Pogledi su povezani jedni sa drugima na različite načine i korišćenjem različitih veza:

• više na jedan asocijacija: Prikaz više elemenata u jednom pogledu je povezan sa jednim elementom u drugom pogledu. Implementacione jedinice su najčešće povezane sa izvršnim komponentama koje postaju. Asocijacija treba da detaljno prikaže koji moduli su mapirani sa kojom komponentom.

• jedan na više asocijacija: Prikaz jednog elementa koji je povezan sa jedinom pogledom na više elemenata drugog pogleda. Primer može biti: "Modul korisnička korpa za kupovinu ("shopping cart") je povezana sa više komponenti aplikacije za veb kupovinu."

• više na više asocijacija: Prikazuje asocijaciju između grupe elemenata jednog pogleda sa grupom elemenata drugog pogleda. Ovaj tip asocijacije reflektuje kompleksnost na dva pogleda, u cilju prikaza glavnih aspekata odabranih pogleda.

7. Kada se primenjuje stil dodeljivanja radnih zadataka? Na koji način se vrši primena stila dodeljivanja radnih zadataka?

Stil dodeljivanja radnih zadataka (work assignement style): opisuje povezanost softverskih modula sa ljudima, timovima ili organizacijom posla u toku razvojnog procesa

Stil dodeljivanja radnog zadatka se koristi za podelu sistema na module i dodeljivanje određenih modula timovima ili članovima tima koji je odgovoran za realizaciju sistema.

Radni zadaci predstavljaju mapiranje softverske arhitekture na grupe ljudi kroz stil dodeljivanja radnog zadatka

Primena stila dodeljivanja radnog zadatka omogućava projektantu da razmisli na koji način je moguće podeliti posao u delove kojima može upravljati.

8. Koja je razlika između predstavljanja softverske arhitekture kroz stilove alokacije i kroz 4+1 Kručtenove poglede?

Primenom Kručtenovih pogleda omogućeno je sagledavanje arhitekture softvera iz logičkog, procesnog, fizičkog i pogleda raspoređivanja softverskih komponenti. Elementi softverske arhitekture su definisani kako bi rešili osnovne funkcionalnosti i korisničke zahteve
Za predstavljanje softverske arhitekture koriste se tri stila alokacije, stil raspoređivanja, stil instalacije i stil dodeljivanja radnih zadataka. Elementi softvera i elementi okruženja softvera imaju jasno definisana svojstva po kojim se vrši raspodela

9. Šta omogućava primena 4+1 Kručtenovih pogleda na softversku arhitekturu?

Primenom Kručtenovih pogleda omogućeno je sagledavanje arhitekture softvera iz logičkog, procesnog, fizičkog i pogleda raspoređivanja softverskih komponenti. Opisani pogledi na softversku arhitekturu zahtevaju modelovanje sledećih dijagrama:
• Logički pogled: modelovanje klasnog dijagrama i dijagrama stanja
• Procesni pogled: dijagram aktivnosti
• Fizički pogled: dijagram komponenti
• Pogled raspoređivanja: dijagram komponenti

10. Zašto je potrebno kombinovati stil raspoređivanja i stil instalacije?

Prikaz stila raspoređivanja kroz UML jezik omogućava prikaz čvorova koji su povezani određenim komunikacionim relacijama. Da bi čitalac softverske dokumentacije znao koja komponenta je raspoređena na koji specifični čvor potrebno je da koristi stil instalacije softverske arhitekture. Stil raspoređivanja je povezan sa drugim komponenta i konektor stilovima koji omogućavaju softverske elemente dodeljene hardveru određene računarske platforme. Takođe, stil raspoređivanja softverske arhitekture korišćenjem UML jezika raspoređivanja je povezan sa stilom instalacije koji prikazuje sadržaje datoteka koje su raspoređene na hardverske čvorove.

11. Šta je specijalizacija stilova alokacije?

Specijalizacija stila alokacije se vrši u slučaju potrebe za ponovnim korišćenjem stila u različitim delovima određenog sistema. Drugi stilovi alokacije su takođe mogući, moguće je definisati zahteve za stilove alokacije koji povezuju sistemske zahteve sa softverskim elementima arhitekture. Primer za tako nešto može biti i specijalizacija određenog stila alokacije:
• **Specijalizacija stila raspoređivanja**: Microsoft je razvio šablon nazvan "Tiered Distribution" koji propisuje raspodelu softverskih komponenti u višeslojnoj arhitekturi na hardverske elemente koji se koriste za pokretanje. Opisani šablon omogućava generički 18 stil primene. Takođe, kompanija IBM ima svoju verzije šablona kao što je: topologija jedne mašine (stand-alone server), topologija vertikalnog skaliranja (vertical scaling topology), topologija

horizontalnog skaliranja (horizontal scaling topology) itd. • **Specijalizacija stila dodeljivanja radnih zadataka**: Moguće je dokumentovati korišćene šeme dodele radnih zadataka u timu kroz specijalizaciju stila dodele radnih zadataka. Specijalizacija stila može biti izvršena kroz:

-Stil platforme. Postavljanje procesa razvoja softvera u obliku proizvodne linije gde određeni deo razvojnog tima proizvodi objekte koji se mogu ponovo koristiti a drugi deo razvojnog tima ponovo koristi tako razvijene objekte u procesu razvoja softvera.

-Stil kompentencije. Radni zadatak se dodeljuje određenim delovima tima u zavisnosti od tehničkog znanja. Na primer, projektovanje korisničkog interfejsa je izvršeno u onom delu tima gde se nalaze stručnjaci za projektovanje korisničkog interfejsa.

-Stil otvorenog koda. Većina nezavisnih saradnika doprinosi razvijanju softverskog proizvoda u skladu sa strategijom za tehničku integraciju. Centralizovana kontrola je minimalna osim u slučaju kada nezavisni saradnik integriše svoj programski kod u softverski proizvod.

12. Šta su kombinovani pogledi na softversku arhitekturu? Koji su načini za kreiranje kombinovanje pogleda?

Osnovni principi dokumentovanja softverske arhitekture predstavljaju grupu različitih pogleda koji omogućavaju detaljan prikaz softverske arhitekture.
Povezivanje različitih pogleda na softversku arhitekturu moguće je korišćenjem asocijacija. Pogledi su povezani jedni sa drugima na različite načine i korišćenjem različitih veza:

• više na jedan asocijacija: Prikaz više elemenata u jednom pogledu je povezan sa jednim elementom u drugom pogledu. Implementacione jedinice su najčešće povezane sa izvršnim komponentama koje postaju. Asocijacija treba da detaljno prikaže koji moduli su mapirani sa kojom komponentom.

• jedan na više asocijacija: Prikaz jednog elementa koji je povezan sa jedinom pogledom na više elemenata drugog pogleda. Primer može biti: "Modul korisnička korpa za kupovinu ("shopping cart") je povezana sa više komponenti aplikacije za veb kupovinu."

• više na više asocijacija: Prikazuje asocijaciju između grupe elemenata jednog pogleda sa grupom elemenata drugog pogleda. Ovaj tip asocijacije reflektuje kompleksnost na dva pogleda, u cilju prikaza glavnih aspekata odabranih pogleda.

Lekcija 6

Navedite primer kada je moguće koristiti šablon Abstraction-Occurrence.

Primeri scenarija u kojima je moguće koristiti ovaj šablon projektovanja su:
• Sve epizode televizijske serije. Epizode imaju istog producenta, naziv serije a različite priče po epizodi.
• Letovi koji svakog dana poleću za istu destinaciju. Letovi imaju isti broj leta a različite datume, putnike i posadu.
• Kopije iste knjige u biblioteci. Kopije knjiga imaju isti naziv i autora a različit barkod i pozajmljene su od različitih korisnika.

Koja su ograničenja u primeni šablona Abstraction-Occurrence?

Potrebno je prikazati članove svakog skupa pojava bez dupliranja zajedničkih informacija. Dupliranje informacija bi iskoristilo prostor i zahtevalo bi mejanje svih pojava u slučaju menjanja zajedničkih informacija. Takođe, potrebno je izbeći rizik od nekonzistentnosti koja bi rezultovana promenu zajedničkih informacija samo u određenim objektima. Potrebno je modelovati rešenje koje omogućava maksimalnu fleksibilnost sistema.

Da li šablon General Hierarchy koristi nasleđivanje?

Greška koja se može javiti prilikom upotrebe ovog šablona je upotreba nasleđivanja. Bitno je naglasiti da se ne podrazumeva da svaka hijerarhija mora biti hijerarhija nasleđivanja.

Koja asocijacija se koristi u šablonu General Hierarchy?

Primer može biti odnos između menadžera i članova tima koji se nalaze ispod njega, zatim direktorijuma (foldera), poddirektorijuma i datoteka u njima. Svaki objekat u tako kreiranoj hijerarhiji može imati nula ili više objekata iznad u hijerarhiji i nula ili više objekata ispod njih

U kojim situacijama je potrebno koristiti šablon Player-Role?

Ovaj šablon koristi se kada objekat može "igrati" ("play") određene uloge u određenom kontekstu. Primer može biti, student koji je na Univerzitetu i može biti diplomirani ili ne diplomirani student u određenom trenutku i on može promeniti ulogu iz jedne u drugu.

Šta predstavlja "uloga" (role) u primeni šablona Player-Role?

Uloga ("role") predstavlja skup mogućnosti povezanih sa objektom u određenom kontekstu. Objekat može "igrati" ("play") određene uloge u određenom kontekstu. Primer može biti, student koji je na Univerzitetu i može biti diplomirani ili ne diplomirani student u određenom trenutku i on može promeniti ulogu iz jedne u drugu. Takođe, student može biti registrovan na kurs u punom vremenu ili u određenom vremenskom intervalu. Student po potrebi može menjati tip vremena koji provodi na kursu.

Šta predstavlja nepromenljiv objekat?

Šablon Immutable omogućava nepromenljiv objekat koji ima stanje koje se nikad ne menja nakon njegovog kreiranja.

Koji su srodni šabloni šablonu Immutable?

Šablon "Read-only Interface" omogućava iste mogućnosti kao i šablon Immutable samim tim, on je srodi šablon šablonu Immutable

Kada se koristi šablon Read-Only Interface?

Koristi se kada je potrebno definisati klase sistema koje mogu da modifikuju atribute objekata koji su inače nepromenljivi. Ovaj šablon je usko povezan sa šablonom Immutable

Šta je klasa "teške kategorije"? (heavyweight class)?

Često je potrebno dosta vremena kako bi se pristupilo instanci klase. Takve klase nazivaju se klase teške kategorije ("heavyweight classes"). Instance od klasa teške kategorije mogu uvek biti u bazi podataka. Da bi se instance koristile u programu potrebno je da ih konstruktor učita sa podacima iz baze podataka. Slično tome objekti teške kategorije mogu postojati samo na severu, pre korišćenja objekta klijent treba da pošalje zahtev a nakon toga sačeka da objekat stigne. U obe situacije postoji vremensko kašnjenje i kompleksan mehanizam koji uključuje stvaranje objekata u memoriji.

Koji su srodni šabloni šablonu Proxy?

Šablon Delegation je sličan šablonu Proksi i može se koristiti zajedno sa ovim šablonom.

Šta predstavlja primetan sloj?

Korišćenjem Observable sloja moguće je izvršiti povezivanje sa drugim klasama sistema.

Koji su glavni nedostaci primene šablona projektovanja?

Potrebno je proceniti da li upotreba šablona može unaprediti sistem u okviru koga se primenjuje. Razvijanje šablona je teško. Pisanje dobrih šablona zahteva značajan i dugotrajan rad. Često se primenom šablona mogu doneti nepravilne odluke koje mogu proces projektovanja odvesti u pogrešnom smeru. Nije potrebno pisati šablone za druge projektante softvera, potrebno je pisati šablon tako da odgovara upotrebi u konkretnom sistemu koji se razvija. Takođe, svaki šablon je potrebno detaljno dokumentovati i specificirati kako bi se održao u kasnijim fazama razvoja softvera.

Da li primena šablona Delegation omogućava uštedu resursa potrebnih za razvoj softvera?

Da li se primenom šablona Delegation duplira programski kod?

Koje su prednosti korišćenja šablona Read-Only Interface?

Na koji način primena šablona omogućava povećanje fleksibilnosti projektovanog sistema?

Lekacija 7

Na koji način se dobija deklarativno znanje o korišćenju metoda?

Znanje o korišćenju metoda projektovanja softvera deli se na deklarativno i proceduralno znanje. Deklarativno znanje je znanje o načinu primene određene metode u određenoj situaciji. Deklarativno znanje se stiče iskustvom projektanta koji koristi metodu. Iskustvo se može steći direktno (kroz primenu metode) ili kroz studije slučaja koje omogućavaju konkretne primere primene određenih metoda.

Kako skup heuristika utiče na aktivnosti u procesu projektovanja softvera?

Skup heuristika: omogućava smernice o načinu definisanja određenih aktivnosti unutar procesnog dela i organizacije određenih klasa problema. Bazirani su na iskustvu i prethodnoj upotrebi metode sa specifičnim domenom problema. Skup heuristika se sastoji od nekoliko različitih tehnika koje se preporučuju za upotrebu u različitim situacijama i za rešavanje različitih problema. Heuristike se generišu u određenom vremenskom periodu a iskustvo primene heuristika se stiče korišćenjem metode u širem problemskom domenu.

Navedite primer tehničkih problema koji mogu nastati primenom metoda projektovanja softvera.

***Mehanizam prenosa znanja**: Omogućava da iskusni projektanti softvera često mogu predviđati buduće događaje u procesu projektovanja softvera na osnovu svog iskustva. Takav pristup može biti pouzdan ukoliko projektant softvera radi u sebi poznatoj oblasti ali ukoliko se radi o projektantu koji nije imao iskustva u radu sa takvim načinom projektovanja, u tom slučaju, potrebno je koristiti različite ponuđene metode

* **Korišćenje metode projektovanja**: Treba da pomogne projektantu softvera da proizvede sistem strukturiran na dosledan način i razumljiv u slučaju da postoji više projektanata softvera

*** Korišćenje metode**: Omogućava proizvodnju zapisa i evidenciju korišćenih standarda u procesu projektovanja kako bi tim koji je zadužen za fazu održavanja imao uvid u način razmišljanja i postavljanja arhitekture sistema od strane projektanta softvera

*** Korišćenje metode**: Upotreba metoda treba da pomogne u smanjivanju verovatnoće potencijalnih grešaka u logičkoj strukturi sistema i da obezbedi da svi faktori uključeni u problem budu pravilno evidentirani i dostupni projektantu softvera za analizu.

/ovde pise dva puta “korišćenje metoda” u lekciji ali mislim da u predzadnjem treba da bude nešto drugo /

Šta je dizajn virtuelna mašina i u kojim slučajevima se koristi?

Svaka metoda projektovanja pruža poseban oblik nazvan "Dizajn virtuelne mašine" (DVM) koji predstavlja okvir koji projektant softvera treba da razvije i pridržava se u toku projektovanja.

Korisnik takve virtuelne mašine strukturira ideju oko modela ponašanja koji je omogućen od strane te virtuelne mašine.

Metoda projektovanja softvera omogućava korisniku virtuelnu mašinu koja se može koristiti za izražavanje ideja o programskim formama na veoma visokom nivou apstrakcije. Delovi procesa metode projektovanja prikazanih na slici 2 često zahtevaju dva ni

Šta su dekompozicione metode? Kako se primenjuju?

Na metode projektovanja softvera ne utiču samo tehnička pitanja, tu su i kulturni uticaji, organizacioni uticaji i nacionalni uticaji kao i konstantna očekivanja i potreba korisnika. Navedene faktore teško je klasifikovati na bilo koji sistematski način ali su definisane široke grupe, jedna od njih su i dekompozicione metode: primenom pogleda "odozgo na dole" u procesu projektovanja softvera u cilju razvijanja modela projektovanja kroz proces podele sistema.

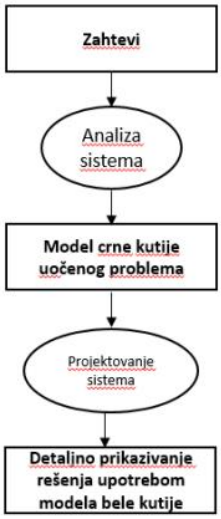
Strategija dekompozicije je podela glavnog zadatka programa na manje zadatke koje je moguće realizovati kao podprograme.

Odluke u toku dekompozicije "odozgo na dole" moraju biti donete na početku procesa projektovanja ili će

efekti loše odluke biti vidljivi kroz kompletan proces projektovanja. Loša odluka može dovesti do ponovnog projektovanja sistema.

Na koji način je moguće primeniti proceduralni model procesa projektovanja softvera?

Procesni deo metode je usko povezan sa delom predstavljanja jer pruža "proceduralne" smernice o tome kako treba razvijati modele.
Na slici prikazan je proceduralni model procesa projektovanja softvera. Osnovni simboli koji se koriste za opisivanje metode projektovanja softvera su:
• pravougaonik za označavanje oblika reprezentacije
• ovalni oblik za označavanje proceduralnog koraka
• luk za označavanje koraka u sekvenci



Slika 3.1 Opšti proceduralni model procesa projektovanja softvera

Na koji način se vrši projektovanje po kompoziciji?

Pristup suprotan od dekompozicije predstavlja kompoziciona strategija za izgradnju modela projektovanja. Model kompozicije je zasnovan na razvoju opisa za određeni skup entiteta objekata koji mogu biti prepoznati kao mogući problemi. Pored opisa entiteta, opisuju se i veze između tih entiteta. Takođe, entiteti u okviru modela variraju shodno problemu koji je potrebno predstaviti

Koji su nedostaci u primeni metoda projektovanja softvera?

Metod projektovana softvera ne omogućava automatsko uklanjanje svih problema. Metoda omogućava projektantu softvera okvir u kome može organizovati proces projektovanja. Predstavlja skup preporučenih šablona koji mogu dati uvid u probleme koji se mogu identifikovati u nekoj fazi projektovanja. Takođe, omogućava uputstva o koracima koje je potrebno izvršiti u procesu projektovanja, savete koje je potrebno uzeti u obzir u određenim koracima kao i o kriterijumima koje je potrebno zadovoljiti prilikom projektovanja. Bitno je naglasiti da nijedna od navedenih smernica ne može biti specifična za problem, već je potrebno akcenat postaviti na dobijanje i primenu ideje koja može uz metode projektovanja softvera rešiti određeni problem

Kako se izvršava transformacija u procesu projektovanja softvera?

Odgovor je ili ovo:

*Faza elaboracije: Projektant proširuje dijagram toka podataka

*Restrktuiranje nakon faze elaboracije: Projektant softvera, nakon faze elaboracije, uzima kreirani dijagram i vrši restrukturiranje u strukturni grafikon pri čemu svaki balon postaje podprogram.

Ili ovo:

Korak transformacije predstavlja korak u kome projektant softvera modifikuje strukturu modela sistema. Sastoji se od ponovne interpretacije kroz drugačije poglede. Takav korak zahteva da projektant softvera izvrši određene jasne odluke u projektovanju.

Kako faktori organizacije utiču na proces projektovanja softvera?

Faktori organizacije najčešće ne utiču direktno na deo reprezentacije i njihov uticaj može imati efekat proširenja. Primer mogu biti međunarodne agencije kao i organi centralne i lokalne samouprave koji su glavni kupci softverskih sistema. Takvi sistemi su uglavnom specijalizovani sistemi za obradu podataka u toku radnog vremena organizacije , sistemi za kontrolu zaliha ili oporezivanje. Većina ovih sistema je veoma velika i jako teško je precizirati zahteve.

Kako faza elaboracije i transformacije utiče na proces projektovanja softvera?

/**mislim da ovo nije baš odgovor na pitanje ali je najbliže što sam našla**/

• Korak transformacije predstavlja korak u kome projektant softvera modifikuje strukturu modela sistema. Sastoji se od ponovne interpretacije kroz drugačije poglede. Takav korak zahteva da projektant softvera izvrši određene jasne odluke u projektovanju.
• Korak elaboracije ne podrazumeva interpretaciju kroz različite poglede i uglavnom se bavi restrukturiranjem ili reorganizacijom modela projektovanja u okviru trenutnog pogleda. Svrha koraka elaboracije je da omogući dodatne informacije u modelu i prikaz trenutnog stanja planiranja projektovanja kroz restrukturiranje u cilju moguće transformacije.

Koje informacije projektant dobija iz faza procesa projektovanja?

SE311 – Projektovanje i arhitektura softvera

Lekcija 10: Projektovana softvera primenom komponenata

10.1. Šta je softverska komponenta? Nevedite pet karakteristika softverskih komponenata i objasnite ih.

Komponenta je softverska jedinica koja se može spojiti sa drugim komponentama radi kreiranja jednog softverskog sistema.
Karakteristike:

- **Standardizovane** – Upotreba komponente koja zadovoljava standard za modele komponente.
- **Nezavisne** – Može se sastaviti i primeniti bez upotrebe drugih komponenti.
- Kompozitne – Sve spoljne interakcije se sprovode preko definisanih interfejsa.
- **Rasporedljive** – Mora da bude sposobna da radi kao samostalan entitet na komponentnoj platformi koja obezbeđuje primenu modela komponente.
- **Dokumentovane** – Moraju da budu u potpunosti dokumentovane, tako da korisnik može da odluči da li komponenta zadovoljava njegove potrebe.

10.2. Objasnite dve kritične karakteristike softverske komponente: nezavisnost i nuđene servisa posredstvom interfejsa komponente? Koja je uloga interfejsa u slučaju primene softverskih komponenata.

Komponenta je nezavisni izvršni entitet definisan svojim interfejsima. Servisi koje nudi komponenta su raspoložive preko interfejsa i sve interakcije se odvijaju samo preko njega.
Interfejsi definišu i specificiraju servise koje obezbeđuje komponenta.

10.3. Postoje dve vrste interfejsa komponenata. Koje su to vrste i objasnite razliku između njih.

Interfejs ''obezbeđuje'' definiše servise koje obezbeđuje komponenta, dok interfejs ''potražuje'' specifira servise koje treba da joj obezbede druge komponente da bi komponenta ispravno radila.

10.4. Šta je model komponente? Koji su osnovni elementi modela komponenata? Šta oni sadrže?

Model komponente je standard za implementaciju, dokumentaciju i raspoređivanje komponente.
Elementi modela komponente: interfejsi(sadrže: definicija interfejsa, kompozicija, specifični interfejsi), informacija o korišćenju(sadrži: konvencija imena, pristup meta podacima, priladođavanje), raspoređivanje i upotreba(sadrži: pakovanje, dokumentacija, podrška evoluciji).

10.5. Koja je razlika između servisa platforme i servisa podrške?

Servis platforme omogućavaju komponentama da komuniciraju i međusobno usaglašeno rade u distribuirano okruženju, dok servisi podrške se nude kao zajednički servisi koji eventualno mogu biti korišćeni od strane različitih komponenata.

10.6. Procesi softverskog inženjerstva zasnovanim na komponenatama omogućavaju projektovanje softvera primenom komponenata (PSPK) ili, na engleskom, CBSE procesi (Component-Based Software Engineering). Postije dva tipa PSPK: Razvoj za ponovnu upotrebu (development for reuse) i Razvoj sa ponovnom upotrebom (development with reuse). Koja je razlika između ova dva tipa projektovanja softvera primenom komponenata?

Razvoj za ponovnu upotrebu je proces koji se bavi razvojem komponenata ili servisa koji će biti ponovo korišćeni u drugim aplikacijama, dok razvoj sa ponovnom upotrebom je proces razvoja novih aplikacija koji upotrebljava postojeće komponente i servise.

10.7. Pri korišćenju softverskih komponenata, postavlja se pitanja rada sa izuzecima.

Svaka komponenta treba da definiše izuzetke koji se mogu javiti i trebalo bi da ih javno objavi, kao deo njenog interfejsa. Međutim, to izaziva javljanje dva problema:

1. Javno objavljivanje izuzetaka može da zatrpa interfejs, te on postaje teško razumljiv. Zbog toga, potencijalni korisnici kogu da odustanu od primene ove komponente.

2. Rad komponente postaje zavistan od lokalne obrade izuzetaka i promene tog rada mogu imati ozbiljne implikacije na funkcionalnost komponente.

10.8. Šta se podrazumeva pod upravljanjem komponentama?

Primena zahteva izvesne promene thi komponenata. To su obično delovi koji se odnose na specifična svojstva aplikacija i interfejse koji nisu zahtevani od

drugih komponenata. Zato se one komponente prilagođavaju i proširuju tako da postaju uopštenije, te i upotrebljivije.

10.9. Navedite aktivnosti procesa projektovanja softvera primenom komponenti (PSPK). U čemu je specifičnost ovog procesa u odnosu na projektovanje softvrea bey primene komponenti?

Aktivnosti primenom PSPK:

1. Utvrđivanje sistemskih zahteva
2. Utvrdi komponente kandidate
3. Modifikuj zahteve u skladu sa nađenim komponentama
4. Projektno rešenje arhitekture
5. Projektuj i implementiraj komponentu
6. Sastavi komponente i kreiraj sistem

Razlike između PSPK i običnog:

1. Korisnički zahtevi se uopštenije navode
2. Zahtevi se detaljišu i menjaju rano u procesu
3. Posle postavljanja arhitekture, nastavlja se pretraga komponenata i poboljšanje projektnog rešenja
4. Projektovanje je jedan proces spajanja i integrisanja pronađenih komponenti.

10.10. Postoje tri vrste spajanja (sastavljanja) komponenata pri razvoju softvera. Nevedite te tri vrste i ukratko opišite.

- **Sekvencijalno spajanje** kreira novu komponentu spajanjem 2 postojeće komponente, njihovim sekvencijalnim pozivanjem.
- **Hijerarhijsko spajanje** se realizuje kada jedna komponenta poziva servise drugog.
- Aditivno spajanje – je slučaj kada se dve komponente sastavljaju radi kreiranja nove komponente koja kombinuje njihovu funkcionalnost.

10.11. Pri povezivanju komponenata, često se suočavamo sa nekompatibilnim interfejsima komponenata. Koje su to nekompatibilnosti interfejsa?

- **Nekompatabilnost parametara**: Operacije na svakoj strani interfejsa imaju isto ime, ali njihovi tipovi parametara ili njihov broj je različit.
- **Nekompatibilnost operacija**: Imena operacija i kod interfejsa koji obezbeđuje i koji zahteva su različite.
- **Nekompletnost operacija**: Interfejsa komponente koji obezbeđuje servis je podskup interfejsa koji zahteva servise druge komponente ili suprotno.

10.12. Šta su adaptori i čemu oni služe pri spajanju komponenata?

Adapter preuzima i prevodi poruku od kontrolera podataka i šalje je senzoru. Poruke senzora prevodi i šalje kolektoru podataka poruke.

10.13. Navedite najčešće odluke koje projektanti treba da donesu kada koriste komponente pri rayvoju softvera.

1. Koji sastav kompononenata daje najveće efekte na zadovoljenje funkcionalnih i nefunkcionalnih zahteva sistema?
2. Koji sastav komponenata čini sistem lakši za prilagođavanje komponenata kada dođe do promena u zahtevima?
3. Koje će svojstva nastati u sistemu?

Lekcija 11: Projektovane distribuiranih softverskih sistema

11.1. Šta su distribuisani softverski sistemi? Zašto se oni koriste? Koje su prednosti koje njihova primena dovodi?

Distribuirani sistem je onaj koji se realizuje na nekoliko kompijutera. Omogućava različitim mašinama da komuniciraju i koordiniraju u ostvarivanju zajedničkih ciljeva. Trpi neuspeh pojedinačnih računara tako da preostali računari nastave da rade i pružaju usluge korisnicima.

Prednosti:

1. Deljenje resursa
2. Otvorenost
3. Konkurentnost
4. Proširivost
5. Otpornost u slučaju grešaka

11.2. Šta utiče na performanse distribuiranih softverskih sistema?

Performanse najviše zavise od propusnog kapaciteta računarske mreže, od njenog trenutnog opterećenja, kao i od brzina svih računara distribuiranog sistema.

11.3. Projektan softvera treba d abude svestan problema upravljivosti distribuiranim sistemom. Na koja pitanja projektan treba da obrati pažnju pri projektovanju sistema, kako bi oni bolje upravljiv? Objasnite zašto su ta pitanja relevantna.

1. **Transparentnost :** Do koje mere bi trebalo da se distribuirani sistem predstavi korisniku kao jedan sistem?
2. **Otvorenost:** Da li treba projektovati sistem koji koristi standardne protokole interoperabilnosti, ili bi trebalo da koristi specijalizovane protokole, i time da ograniče slobodu projektanta?
3. **Proširivost:** Kako treba postaviti arhitekturu sistema tako da on bude proširiv? Kako projektovati ceo sistem tako da se njegov kapacitet može povećati u skladu sa povećanjem zahteva za korišćenjem sistema?
4. **Bezbednost:** Kako definisati i primeniti bezbedonosna pravila korišćenja sistema na svim podsistemima koje čine jedan distribuirani sistem?
5. **Kvalitet servisa:** Kako specificirati kvalitet servisa sistema koji se isporučuje kupcima i kako ga realizovati da bi bio isporučen sa prihvatljivim kvalitetom servisa svim korisnicima?
6. **Upravljanje otkazima:** Kako se mogu otkriti otkazi u sistemu, šta učiniti da se minimizita njhov negativni efekat na druge komponente sistema i kako se mogu izvršiti popravke sistema?

11.4. Šta je posrednički softver, tj. middlver (middleware)?

Midlver se nalazi u stolu između komponenata sistema i operativnog sistema. Obezbeđuje podršku interakcijama komponenti, kao i više zajedničkih servisa svojim komponentama.

11.5. Šta su otvoreni distribuirani sistemI? Šta podrazumeva njihova «Otvorenost»?

Otvoreni distribuirani sistemi su sistemi koji su sagrađeni u skaldu s generalno prihvaćenim standardima. To znači da se komponente različitih dobavljača integrišu sistematski da one mogu da rade sa ostalim sistemskim komponentama. Otvorenost podrazumeva da su sistemske komponente nezavisno razvijene u bilo kom programskom jeziku i ako su razvijene primenom standarda, one mogu da međusobno komuniciraju i rade.

11.6. Šta je proširivost (scalability) distribuiranih sistema? Koje su tr i dimenzije proširivosti sistem? Objasnite ih.

Proširenje sistema podrazumeva zamenu komponenata sa moćnijim komponentama, a povećanje sistema podrazumeva povećanje broja komponenti, što je isplatljivije.

Dimenzije proširivosti:

1. **Veličina**: Trebalo bi da bude mogućno da se dodaju resursi sistemu da bi on mogao da odgovori na povećan broj svojih korisnika.

2. **Distribucija**: Trebalo bi da bude mogućno da se komponente prostorno raštrkaju a da to ne umanji njegove performanse.
3. **Upravljivost**: Neohodna je uspešno upravljanje sistemom i kada se povećava njegova veličina, čak i u slučajevima kada su delovi sistema locirani u nezavisnim organizacijama.

11.7. Šta je razlika između prišireњa sistema (scaling-up) i povećanja sistema (scaling-out)?

Proširenje sistema(scalling up) podrazumeva zamenu komponenata sa moćnijim komponentama, a povećanje sistema(scalling out) podrazumeva povećanje broja istih komponenata, što je najčešće isplatifije.

11.8. Distribuirani sistem su ranjive na napade, te su manje bezbedni od centralizovanih sistema. Zašto? Nevdite i objasnite vreste napada kojima su distribuirani sistemi izloženi.

Vrste napada:

1. **Presretanje** kada se komunikacije između delova sistema presreću od strane napadača, tako da dolazi do pada poverenja i bezbednost sistema.
2. **Presecanje** kada se napadaju servisi sistema te oni ne mogu da se ostvare, onako kako se očekuje. To se postiže bombradovanjem nekog čvora sistema sa nelegitimnih zahtevima za servisima, da bi se sprečili njegove odgovore na legitimne zahteve.
3. **Promena** kada se menjaju podaci ili servisi u sistemu od strane napadača.
4. **Fabrikacija** kada napadač generiše informaciju koja ne bi trebalo da postoji i koju onda upotrebljava da bi stekao neke privilegije, tj. prava pristupa. Na primer, napadač može da generiše lažnu lozinku i da na osnovu nje uđe u sistem.

11.9. Šta je kvalitet servisa? Koji problemi prate zahteve za većim kvalitetom servisa? Navedite neki primer.

Kvalitet usluga koje nudi distribuirani sistem održava sposobnost sistema da isporuči traženi servis pouzdano, sa prihvatljivim vremenom odziva, i prihvatljivim protokom informacija.

Problemi:

1. Nije prihvatljiva cena takvog sistema prilikom maksimalnog opterećenja. Zbog toga, najveći deo resursa stoji neiskorišćen. Najveći razlog za primenu klaud računarstva je upravo u ovome, jer se njegovim primenom optimalno koriišćenja raspoloživih resursa.

Upotreba Računarstva oblaka (cloud computing), postiže se dodavanjem resursa kada poraste broj zahteva.

2. Parametri QoS mogu međusobno da budu kontradiktorni. Na primer, porast pouzdanosti može da smanji protok informacija, zbog procedura proveravanja radi obezbeđenja ispravnosti svih ulaza u sistem. QoS je posebno kritičan kada sistem radi sa kritičnim vremenskim podacima, kao što su audio i video strimovi.

11.10. Kako se upravlja otkazima u slučaju distriburanih softverskih sistema?

Sistem treba da pronađe komponentu u kojoj je došlo do otkaza i da sistem nastavi da pruža što više servisa, bez obzira na otkaz u nekoj komponenti. Sistemi bi trebalo da primeni mere za automatski oporavak do otkaza u nekoj komponenti.

11.11. Postoje dva osnovna modela interakcija između čvorova distribuiranog sistema. Koja su to dva modela i objasnite ih.

1. **Proceduralna interakcija** uključuje računar koji poziva poznati servis koji nudi neki drugi računar i obično čeka isporuku tog servisa.
2. **Interakcija poruka** uključuje računar koji definiše informaciju o onome što bi trebalo da sadrži poruka, koja se onda šalje drugom računaru. Poruke obično prenose više informacija u jednoj interakciji nego proceduralna interakcija.

11.12. Šta je proceduralni poziv (RPC)? Šta je tačka povezivanja (stub)? Gde se nalazi udaljena procedura i koja je njena funkcija?

Proceduralni poziv (RPC) koristi komponenta koja poziva drugu komponentu kao u slučaju poziva lokalne operacije, tj. metoda. Interakcija sa porukama toleriše nedostupnost primaoca.

RPC zahteva "tačku povezivanja" (**stub**) za proceduru koju poziva da bi ona bila pristupačna računaru koji inicilizira poziv.

Udaljena procedura upotrebljava biblioteku funkcija za konverziju parametara u zahtevani format, realizuje obradu podataka, i komunicira rezultat preko tačke pozivanja (stub) koja predstavlja komponentu koja inicira poziv.

11.13. Šta je interakcija sa porukeama? Oja je učaga midlvera u tome?

Interakcija porukama normalno uključuje komponentu koja kreira poruku koja sadrži detalje zahtevanog servisa koji nudi neka druga komponenta.

Preko midlver sistema (posredničkog softvera), poruka se šalje komponenti koja treba da je primi.

11.14. Koja je razlika proceduralnog poziva (RPC) i razmene poruka? Objasnite..

Proceduralni poziv (RPC) koristi komponenta koja poziva drugu komponentu kao u slučaju poziva lokalne operacije, dok interakcija porukama normalno uključuje komponentu koja kreira poruku koja sadrži detalje zahtevanog servisa koji nudi neka druga komponenta.

11.15. Koje su najčešće funkcije midlvera? Kako midlver podržava interakciju čvorova?

Funkcije midlvera:

- Upravljanje komunikacijama
- Rad sa bazama podataka
- Upravljanje transakcijama
- Konverzija podataka
- Kontrola komunikacija

Podržavanje interakcije čvorova:

1. **Podrška interakciji** kada midlver koordiniše interakcije između različitih komponenti sistema. Midlver obezbeđu lokacijsku transparentnost.

2. **Obezbeđenje zajedničkih servisa** komponentama sistema. To omogućava nihovu interoperabilnost i pružanje korisničkih servisa na konsistetan način.

11.16. Navedite i objasnite nivoe slojevite arhitekture distribuiranih klijent-server sistema. Koji su problemi dvoslojne rahitekture?

1. **Sloj prezentacije**, koji prikazuje informaciju korisniku i upravlja interakcijom korisnika,
2. **Sloj upravljanja podacima** koji proverava podatke, generiše veb strane i dr.
3. **Sloj obrade aplikacije**, koji implementira logiku aplikacije i obezbeđuje zahtevanu funkcionalnost krajnjim korisnicima.
4. **Sloj baze podataka**, koji skladišti podatke i obezbeđuje upravljanje transakcijama servisa i dr.

11.17. Koja je razlika klijent-server arhitekture sa «debelim» i «tankim» klijentom? Navedite prednosti i nedostatke primene «tankih» a posebno, «debelih» klijenata?

Tanki klijent	**Debeli klijent**
Jednostavan za implementaciju jer ne zahteva nikakvu dodatnu ili specijalizovanu instalaciju softvera	Skuplje je rasporediti i više rada za IT implementirati
Potrebno je da se potvrdi sa serverom	Podaci verifikovani od strane klijenta a

nakon zauzimanja podataka	ne servera
Ako server padne, prikupljanje podataka se zaustavlja jer je klijentu potrebna stalna komunikacija sa serverom	Robusna tehnologija omogućava bolji radni vek
Ne može se povezati sa drugom opremom (na primer u postrojenjima ili fabričkim podešavanjima)	Potrebna je samo isprekidana komunikacija sa serverom
Klijenti rade samo i tačno onako kako je specificirano od strane servera	Skuplje za instaliranje i više posla za IT raspoređivanje

11.18. Koje su predmosti višeslojne arhitekture? A koji su njeni problemi?

Prednosti su bolja održivost(poslovna logika se može menjati bez ažuriranja svake klijentske mašine), a ako se koristi u kombinaciji sa sistemom za obradu transakcija ili sistemom za objedinjavanje veza, n-slojeva mogu pružiti bolje performanse od dvoslojne.

Problemi:

- Otežava programerima da menjaju aplikaciju sa fleksibilnošću koja im je potrebna da bi išli u korak sa očekivanjima mobilnih korisnika.
- Promena bilo kog modula zahteva ponovnu izgradnju i testiranje cele aplikacije.
- Nedostaje skalabilnost, dizajnirana u doba kada ideja o elastičnosti i brzom skaliranju nije široko postojala.

11.19. Koja je korist od primene sistema sa distribuiranim komponentama?

Korist:

1. **Dozvoljava projektantu sistema** da odloži odluku gde i kako da se pojedini servisi obezbede. Komponente koje obezbeđuju servise mogu da se izvršavaju na bilo kom čvoru mreže.

2. **Arhitektura sistema** je vrlo otvorena , tako da se mogu dodavati novi resursi, ako je to potrebno. Pri tome ne dolazi do značajnog poremećaja rada postojećeg sistema.

3. **Sistem je fleksibilan i proširljiv.** Nove ili replicirane komponente se mogu dodavati kada dolazi do porasta opterećenja sistema, bez ometanja rada ostalih delova sistema.

4. **Moguće je izvršiti dinamičku rekonfiguraciju sistema** sa komponentama koje se prebacuju po mreži, ako je to potrebno. Ovo može biti značajno u slučajevima kada dolazi do fluktacije zahteva za korišćenje servisa. Komponenta koja obezbeđuje servis može da se prabaci na isti procesor na kome je i komponeta koja zahteva taj servis, što poboljšava performanse sistema.

11.20. Koji su nedostaci sistema sa distribuiranim komponentama?

Nedostaci:

1. **Složenije su za projektovanje nego klijent-server sistemi**. Višeslojna klijent-server sistemi su dosta intuitivni. Oni odražavaju mnoge ljudske transakcije kako ljudi traže i pružaju servise drugim ljudima. Nasuprot ovome, arhitekture sa distribuiranim komponentama su teže za razumevanje ljudi i za vizualizaciju.

2. **Standardizovani midlver za sisteme sa distribuiranim komponenatama nije prihvaćen od korisnika**. Umesto toga, pojedini proizvođači (Microsoft, Sun) su razvili različit i nekompatibilni. midlver. Midlver je složen sistem i oslanjanje na njih povećava ukupnu složenost sistema sa distribuiranim komponentama

11.21. Uporedite servisno-orijentisanu arhitekturu sa arhitekturom sistema sa distribuiranim komponentama. Koja ima bolje preformase? Zašto?

--

11.22. Na kojim principa rade sistemi ravnopravnih računara (peer-to-peer, or p2p)? Kako su njeniu čvorovi povezani? Kada se koristi p2p arhitektura?

Sistemi ravnopravnih računara su decentralizovani sistemi u kojima obrada podataka može da se vrši u bilo kom čvoru mreže, koji koriste istu kopiju aplikacije sa protokolima komunikacije.

Standardi i protokoli koji omogućavaju komunikacije među čvorovima su ugrađene u samu aplikaciju i svaki čvor mora da izvršava kopiju te aplikacije.

p2p tehnologije se najviše koriste za personalne aplikacije.(Skype, BitTorrent)

11.23. Šta je polucentralizovana arhitektura sa ravnopravnim računarima? U čemu je njena prednost? Da li ona ima problem bezbednosti?

Polucentralizovana arhitektura ima , jedan ili više čvorova koji služe kao serveri za olakšavanje komunikacija među čvorovima. Ovo smanjuje količinu prenosa podataka između čvorova.

Obezbeđuje redudantnost što obezbeđuje otpornost sistema na otkaze a i na isključenje pojedinih čvorova sa mreže.

Problem bezbednosti: u p2p sistemima dovode vaš računar u direktnu interakciju sa drugim računarima u mreži, a to znači da ovi sistemi mogu, potencijalno, da pristupaju bilo kom vašem resursu.

11.24. Šta e «softver-kao-servis» (SaaS) ? Koji su ključni elementi SaaS koncepta? Koje su njegove prednosti?

Softver kao servis podrazumeva instalaciju softvera u nekom udaljenom serveru koji omogućava pristup klijentima preko Interneta i veb pretraživača.
Ključni elementi:
1. **Softver je raspoređen na serveru** (ili na više servera)
2. **Softver je u vlasništvu provajdera softvera** koji i njim upravlja
3. **Korisnici plaćaju korišćenje softvera** u skladu sa obimom njegovog korišćenja ili plaćanjem mesečne ili godišnje cene servisa. Ako je servis besplatan (npr. Youtube), onda korisnici moraju da prihvate reklame koje finansiraju servis.

Prednosti:
1. Oslobađanje troškova upravljanja softvera.
2. Odgovoran za otklanjanje grešaka u softveru I za instalaciju novih verzija.
3. Prilagođavanje novih verzija operativnog sistema.
4. Osiguranje potrebnog kapaciteta hardvera.

11.25. Koji su problemi primene SaaS koncepta?

Problemi:
1. Troškovi transporta podataka ka udaljenom servisu.
2. Nedostatak kontrole evolucije softvera
3. Zakoni i propisi

11.26. Koja je razlika SaaS i SOA?

SaaS je način obezbeđivanja funkcionalnosti na udaljenom serveru kome pristupaju klijenti preko Interneta uz korišćenje veb pretraživača. Server održava podatke korisnika i njihovo stanje za vreme interaktivne sesije. Transakcije su obično duge (npr. izmena nekog dokumenta).

SOA je pristup strukturisanom softverskom sistemu u vidu skupa posebnih servisa, koji nemaju svoje stanje (stateless). Ovo može da obezbeđuje veći broj provajdera i može da bude i distribuiran. Transakcije su obično kratke, i obuhvataju poziv servisa, negov rad i vraćanje rezultata korisniku.

11.27. Na koje faktore morate d aobratite pažnjupri primeni SaaS?

1. **Konfugurabilnost**: Da li ste konfigurisali softver u skladu sa specifičnim zahtevima svake od organizacije?
2. **Višestruki zakup**: Kako predstavite softversku uslugu korisnicima tako da oni oni dobiju utisak da rade sa svojom sopstvenom kopijom sistema, dok, u isto vreme, efikasno koriste resurse sistema.
3. **Proširljivost**: Kako projektujete sistem tako da on bude proširljiv da bi mogao da zadovolji nepredviđeno veliki broj korisnika?

11.28. Šta je dinamičko konfigurisanje softvera? Šta dozvoljava dinamičko konfigurisanje?

Preko interfejsa konfiguracije, korisnici specificiraju svoje potrebe (opcije) u skladu sa čim se sistem dinamički konfiguriše tako da daje utisak da je korisnik jedini korisnik sistema.

Dinamičko konfigurisanje dozvoljava:

1. **Brendiranje**: Korisnici iz svake organizacije koriste interfejs koji odražava njihovu organizaciju (kompanijski logo i dr.).
2. **Poslovna pravila i radni tokovi**: Svaka organizacija može da definiše svoja pravila korišćenja servisa i njenih podataka.
3. **Proširenja baze prodataka**: Svaka organizacija definiše kako se opšti model podataka servisa proširuje da bi zadovoljio njihove specifične potrebe.
4. **Kontrola pristupa**: Kupci servisa kreiraju posebne račune za svoje zaposlene i definišu resurse i funkcije koje mogu da koriste njihovi zaposleni.

11.29. Šta je proširivost SaaS sistema? Koje su preoruke za realizaciju prišireja SaaS sistema?

Proširljivost je sposobnost sistema da radi sa povećanim brojem korisnika bez smanivenja ukupnog kvaliteta servisa (QoS) koji se pruža svakom korisniku, dodavanjem novih server.

Preporuke za proširivost:

1. **Svaka komponenta treba da bude razvijena kao jednostavni servis** bez stanja koji se može izvršavati na bilo kom serveru. U okviru iste transakcije, korisnik može da radi sa delovima servisa koji su instalirani na različitim računarima.

2. **Pri projektovanju koristite asinhrone komunikacije** kako aplikacija ne bi čekala rezultat interakcije. Na taj način, izvršenje aplikacije se nastavlja i za vreme dok korisnik priprema svoju interakciju.

3. **Upravljajte resursima**, kao što su veze mreže i baza podataka, kao pul, tako da nijedan pojedinačni server ne dođe do nedostatka svojih resursa.

4. **Projektujte svoju bazu** tako da omogućava zaključavanje ne što detaljnije nivou (fine-grain locking). Ne zaključavajte ceo slog baze kada koristite samo jedan njegov deo.

SE311 PROJEKTOVANJE I ARHITEKTURA SOFTVERA

ISPITNA PITANJA

Pripremljeno 01.10.2019

Ovde se daje lista ispitnih pitanja koja treba da ukažu na razumevanje studenta izloženih programskih sadržaja na predavanjima, tj. stečena teorijska znanja koja su podloga za uspešno rešavanje zadataka koji se daju u drugom delu ispita. Pitanja pokrivaju ceo program predmeta, te ne ukazuju studentu koji deo predmeta je prioritetan za ispit. Ispit treba da pokaže u kojoj meri je student stekao znanja celog programa predmeta, te i pitanja, zbog toga, moraju da pokriju ceo program predmeta. Spisak sadrži pitanja po lekcijama, te za svaku od 15 lekcija, dat je spisak pitanja. Zbog ograničenog vremena (45 minuta) student dobija po jedno pitanje iz 10 slučajno odabranih lekcija, te se izbor menja od ispita do ispita.

Odgovori na 10 pitanja obezbeđuju do 10 poena. Odgovore na pitanja studenti kucaju u bilo kom tekstualnom editoru na računaru i odmah po završetku, a pre pristupa praktičnom delu ispita (rešavanje zadataka) šalju preko Zimbre svom asistentu. Po predaji odgovora na pitanja, student prelazi na rešavanje datih zadataka, koji obezbeđuju do 20 poena. Da bi student položio ispit mora da ima najmanje po 50% poena predviđenih i od odgovora na pitanja i od rešenja zadatih zadataka.

Lekcija 1: Osnove projektovanja softvera

1.1. Šta omogućava apstrakcija unutar projektovanja softvera?
1.2. Na koji način se vrši podela procesa projektovanja na projektne aktivnosti?
1.3. Kako faza održavanja ima uticaj na proces projektovanja softvera?
1.4. Kako testiranje softvera može ispuniti ciljeve verifikacije i validacije?
1.5. Kako unaprediti proces testiranja softvera tako da odgovara ciljevima verifikacije i validacije procesa projektovanja softvera?

1.6. Šta se podrazumeva pod razmenom znanja projektanta sa drugim učesnicima u procesu projektovanja softvera?
1.7. Šta su kanali komunikacije projektanta softvera?
1.8. Koja je uloga ograničenja u procesu projektovanja softvera?
1.9. Šta je metoda prepoznavanja?
1.10. Na koji način proces projektovanja softvera unapređuje proces razvoj softvera?
1.11. Šta je dugoročni plan izmena? U kojim slučajevima se koristi?
1.12. Objasniti da li proces projektovanja usložnjava razvoj softvera ili ga čini jednostavnijim?
1.13. Šta je model rešenja?
1.14. Kako može doći do neusaglašenosti u procesu projektovanja softvera?

Lekcija 2: Arhitektonske strukture, pogledi i stilovi

2.1. Kada je potrebno ostvariti visoke performanse softvera, na koji način je to moguće izvršiti kroz primenu softverske arhitekture?
2.2. Šta predstavlja cilj pisanja softverske dokumentacije?
2.3. Na koji način se određuju korisnici dokumentacije softverske arhitekture?
2.4. Šta omogućavaju različiti pogledi na softversku arhitekturu?
2.5. Šta se postiže dokumentovanjem različitih pogleda na arhitekturu softvera?
2.6. Za koje aktivnosti u toku procesa razvoja softvera koristimo dokumentaciju softverske arhitekture?
2.7. Koji koraci su potrebni da bi se izvršilo kombinovanje različitih stilova softverske arhitekture?
2.8. Šta je element softverske arhitekture?
2.9. Zašto je potrebno pisati softversku dokumentaciju iz ugla čitaoca?
2.10. Kako primena pravila za pisanje softverske dokumentacije utiče na proces pisanja softverske dokumentacije?
2.11. Na koji način se vrši odabir elemenata koji se predstavljaju u softverskoj dokumentaciji?
2.12. Koje su prednosti pisanja dokumentacije u toku procesa razvoja softvera?
2.13. U kojim slučajevima se koristi formalna notacija?
2.14. Da li možete identifikovati neke nedostatke pisanja softverske dokumentacije?

Lekcija 3: Stilovi softverskih modula

3.1. Kada se koristi izraz modul softvera?
3.2. Šta omogućavaju različiti pogledi na softversku arhitekturu?
3.3. Koja je osnovna relacija stila razlaganja?
3.4. Na koji način je stil razlaganja pogodan za nove članove razvojnog tima?
3.5. Po čemu se stil upotrebe razlikuje od stila razlaganja?
3.6. Šta podrazumevaju inkrementalni podskupovi modula?
3.7. Koje su prednosti stila generalizacije na raspolaganju projektantu softvera?
3.8. Na koji način je omogućeno "produžavanje" modula primenom stila generalizacije?
3.9. Kako se prikazuje softverska arhitektura kroz stil slojeva?
3.10. Koji stilovi softverske arhitekture su pogodni za korišćenje uz stil slojeva?
3.11. Šta su aspekti?
3.12. Koje informacije sadrži prikaz aspekta u softverskoj dokumentaciji?
3.13. Kada se koristi model podataka?
3.14. Koji su tipovi modela podataka?
3.15. Da li primena stilova arhitekture usložnjava ili olakšava proces projektovanja softvera projektantu softvera?

Lekcija 4: Stilovi povezivanja softverskih komponenata

4.1. Na koji način se u neformalnoj notaciji predstavljaju komponente i konektori?
4.2. Koja svojstva ima elemenat komponenta u pogledu komponente i konektora softverske arhitekture?
4.3. Kada se koristi stil toka podataka?
4.4. Koja je osnovna razlika između klijent-server stila i stila peer-to-peer?
4.5. Šta podrazumeva grafičko predstavljanje pogleda?
4.6. Koja je svrha elementa konektor u pogledu komponente i konektora?
4.7. Koja su svojstva koja se koriste u pogledu komponente i konektora?
4.8. Kada se koristi stil komponente i konektora za predstavljanje softverske arhitekture?

4.9. Koji pogledi na softversku arhteikturu mogu biti kombinovani sa pogledom komponente i konektora?
4.10. Koji stilovi softverske arhitekture spadaju u pogled komponente i konektora?
4.11. Šta je pružalac usluga? Koja su njegova svojstva?
4.12. Koji su elementi sistema koji koristi klijent-server softversku arhitekturu?

Lekcija 5: Stilovi alokacije i hibridni stilovi

5.1. Šta se dobija specijalizacijom stilova alokacije?
5.2. U kojim stilovima alokacije je potrebno prikazivati hardverske elemente okruženja softvera?
5.3. Koja je osnovna relacija stila alokacije?
5.4. Na koji način se vrši specijalizacija stila raspoređivanja?
5.5. Zašto je neophodno kombinovati različite poglede na softversku arhitekturu?
5.6. Da li je moguće kombinovati bilo koji pogled softverske arhitekture sa nekim drugim pogledom?
5.7. Kada se primenjuje stil dodeljivanja radnih zadataka? Na koji način se vrši primena stila dodeljivanja radnih zadataka?
5.8. Koja je razlika između predstavljanja softverske arhitekture kroz stilove alokacije i kroz 4+1 Kručtenove poglede?
5.9. Šta omogućava primena 4+1 Kručtenovih pogleda na softversku arhitekturu?
5.10. Zašto je potrebno kombinovati stil raspoređivanja i stil instalacije?
5.11. Šta je specijalizacija stilova alokacije?
5.12. Šta su kombinovani pogledi na softversku arhitekturu? Koji su načini za kreiranje kombinovanje pogleda?

Lekcija 6: Upotreba šablona projektovanja softvera

6.1. Navedite primer kada je moguće koristiti šablon Abstraction-Occurrence.
6.2. Koja su ograničenja u primeni šablona Abstraction-Occurrence?
6.3. Da li šablon General Hierarchy koristi nasleđivanje?
6.4. Koja asocijacija se koristi u šablonu General Hierarchy?
6.5. U kojim situacijama je potrebno koristiti šablon Player-Role?
6.6. Šta predstavlja "uloga" (role) u primeni šablona Player-Role?

6.7. Da li primena šablona Delegation omogućava uštedu resursa potrebnih za razvoj softvera?
6.8. Da si primenom šablona Delegation duplira programski kod?
6.9. Šta predstavlja nepromenljiv objekat?
6.10. Koji su srodni šabloni šablonu Immutable?
6.11. Kada se koristi šablon Read-Only Interface?
6.12. Koje su prednosti korišćenja šablona Read-Only Interface?
6.13. Šta je klasa "teške kategorije"? (heavyweight class)?
6.14. Koji su srodni šabloni šablonu Proxy?
6.15. Šta predstavlja primetan sloj?
6.16. Koji su glavni nedostaci primene šablona projektovanja?
6.17. Na koji način primena šablona omogućava povećanje fleksibilnosti projektovanog sistema.

Lekacija 7: Strategije i metodi projektivanja softvera

7.1. Na koji način se dobija deklarativno znanje o korišćenju metoda?
7.2. Kako skup heuristika utiče na aktivnosti u procesu projektovanja softvera?
7.3. Navedite primer tehničkih problema koji mogu nastati primenom metoda projektovanja softvera.
7.4. Šta je dizajn virtuelna mašina i u kojim slučajevima se koristi?
7.5. Šta su dekompozicione metode? Kako se primenjuju?
7.6. Na koji način je moguće primeniti proceduralni model procesa projektovanja softvera?
7.7. Na koji način se vrši projektovanje po kompoziciji?
7.8. Koji su nedostaci u primeni metoda projektovanja softvera?
7.9. Kako se izvršava transformacija u procesu projektovanja softvera?
7.10. Kako faktori organizacije utiču na proces projektovanja softvera?
7.11. Kako faza elaboracije i transformacije utiče na proces projektovanja softvera?
7.12. Koje informacije projektant dobija iz faza procesa projektovanja?

Lekcija 8: Tradicionalni metodi projektovanja softvera

8.1. Šta čitalac dokumentacije softverske arhitekture može da vidi iz "P-Spec" specifikacije procesa?
8.2. Šta je rezultat strukturnog projektovanja softvera?

8.3. Šta je cilj strukturne sistemske analize?
8.4. Koje informacije projektant softvera beleži u rečnik podataka?
8.5. Na koji način se predstavlja strukturni način projektovanja?
8.6. Kako se vrši proces transformacije u SSA?
8.7. Koje su aktivnosti koje je potrebno izvršiti u koraku "Analiza transakcije"?
8.8. Kojim dijagramom se predstavlja JSP?
8.9. Šta je JSP proces?
8.10. Šta je predstavljeno dijagramom specifikacije sistema?
8.11. Šta je entitet u dijagramu strukture entiteta?
8.12. U kom slučaju se koristi JSD? Navesti i objasniti primer primene JSD.
8.13. Šta je analiza transakcije u okviru procesa transformacije?
8.14. Koje su operacije u okviru analize transformacije?

Lekcija 9: Ponovna upotreba softvera

9.1. Zašto se pri razvoju softvera koristi i stari softver (ranije razvijene softverske komponente pripremljene za ponovnu upotrebu i dr.)?
9.2. Koje su koristi od ponovne upotrebe ranije razvijenih softverskih jedinica?
9.3. Zašto ponovna upotreba softvera povećava pouzdanost softvera?
9.4. Zašto ponovna upotreba softverskih jedinica smanjuje rizik razvoja novog softvera?
9.5. Zašto ponovna upotreba softverskih jedinica povećava efektivnost upotrebe softverskih specijalista?
9.6. Zašto ponovna upotreba softverskih jedinica povećava usaglašenost sa standardima?
9.7. Zašto ponovna upotreba softverskih jedinica ubrzava razvoj softvera?
9.8. Koji se problemi mogu javiti pri razvoju softvera uz upotrebu ranije razvijenih softverskih jedinica? Objasni svaki od ovih problema.
9.9. Koje bi faktore imali u vidu pri izboru softvera za ponovnu upotrebu?
9.10. Šta je radni okvir (framework) za razvoj softvera? Koje su tri kategorije radnog okvira?
9.11. Objasni radni okvir za razvoj veb aplikacija (MVC šablon). Koju funkcionalnost obezbeđuje MVC šablon?
9.12. Šta su povratni poyivi (callbacks)?

9.13. Šta je linija proizvoda (software product line)? Kako se projektuje jezgro softverskog sistema?
9.14. Koja je razlika između aplikacionog radnog okvira i linje softverskog proizvoda?
9.15. Kako se može razviti specijalizovani tip softverskog proizvoda primenom lije softverskog proizvoda.
9.16. Kako se vrši konfigurisanje softverskog sistema tokom procesa njegovog razvoja?
9.17. Kako se vrši konfigurisanje softverskog sistema u vreme njegovog raporeĎivanja 'stavljanja u upotrebu)? Koji su nivoi konfigurisanja sistema?
9.18. Šta je komercijalni gotov proizvod (a commercial-off-the-shelf ili COTS)? Koje su prednosti njihove primene?
9.19. Koji se problemi mogu javiti pri primeni gotovih komercijalnih softverskih proizvoda?
9.20. Objasni razliku između COTS-sistemskih rešenja (COTS-solution systems) i COTS-integrisanih sistema (COTS-integrated systems)?
9.21. Opiši tipičnu arhitekturu ERP sistema. Koja su ključna svojstva ove arhitekture?
9.22. Navedi uobičajene aktivnosti konfigurisanja COTS-sistema, kao što su ERP sistemi.
9.23. Kako bi izabrao COTS sistem? Koje faktore bi uzeo u obzir?
9.24. Koji se problemi mogu javiti pri razvoju softvera integracijom više COTS proizvoda?

Lekcija 10: Projektovana softvera primenom komponenata

10.1. Šta je softverska komponenta? Nevedite pet karakteristika softverskih komponenata i objasnite ih.
10.2. Objasnite dve kritične komponente softverske komponente: nezavisnost i nuđene servisa posredstvom interfejsa komponente? Koja je uloga interfejsa u slučaju primene softverskih kompšonenata.
10.3. Postoje dve vrste interfejsa kompšpnenata. Koje su to vrste i objasnite razliku između njih.
10.4. Šta je model komponente? Koji su osnovni elementi modela komponenata? Šta oni sadrže?
10.5. Koja je razlika između servisa platforme i servisa podrške?
10.6. Procesi softverskog inženjerstva zasnovanim na komponenatama omogućavaju projektovanje softvera primenom komponenata (PSPK) ili, na engleskom, CBSE procesi (Component-Based Software Engineering). Postije dva tipa PSPK: Razvoj za ponovnu upotrebu (development for reuse) i Razvoj sa ponovnom upotrebom (development with reuse). Koja je razlika između ova dva tipa projektovanja softvera primenom komponenata?

10.7. Razvili ste softver i razmišljate da li da na osnovu njega napravite nekoliko kompoenata koje bi kasnije mogli da koristiti pri razvoju softvera. Kako ćete odlučiti? Koje faktore ćete uzeti u obzir prilikom odlučivanja?

10.8. Pri korišćenju softverskih komponenata, postavlja se pitanja rada sa izuzecima.

10.9. Šta se podrazumeva pod upravljanjem komponentama?

10.10. Navedite aktivnosti procesa projektovanja softvera primenom komponenti (PSPK). U čemu je specifičnost ovog procesa u odnosu na projektovanje softvrea bey primene komponenti?

10.11. Postoje tri vrste spajanja (sastavljanja) komponenata pri razvoju softvera. Nevedite te tri vrste i ukratko opišite.

10.12. Pri povezivanju komponenata, često se suočavamo sa nekompatibilnim interfejsima komponenata. Koje su to nekompatibilnosti interfejsa?

10.13. Šta su adaptori i čemu oni služe pri spajanju komponenata?

10.14. Navedite najčešće odluke koje projektanti treba da donesu kada koriste komponente pri rayvoju softvera.

Lekcija 11: Projektovane distribuiranih softverskih sistema

11.1. Šta su distribuisani softverski sistemi? Zašto se oni koriste? Koje su prednosti koje njihova primena dovodi?

11.2. Šta utiče na performanse distribuiranih softverskih sistema?

11.3. Projektan softvera treba d abude svestan problema upravljivosti distribuiranim sistemom. Na koja pitanja projektan treba da obrati pažnju pri projektovanju sistema, kako bi oni bolje upravljiv? Objasnite zašto su ta pitanja relevantna.

11.4. Šta je posrednički softver, tj. middlver (middleware)?

11.5. Šta su otvoreni distribuirani sistemI? Šta podrazumeva njihova «Otvorenost»?

11.6. Šta je proširivost (scalability) distribuiranih sistema? Koje su tr i dimenzije proširivosti sistem? Objasnite ih.

11.7. Šta je razlika između prišireNja sistema (scaling-up) i povećanja sistema (scaling-out)?

11.8. Distribuirani sistem su ranjive na napade, te su manje bezbedni od centralizovanih sistema. Zašto? Nevdite i objasnite vreste napada kojima su distribuirani sistemi izloženi.

11.9. Šta je kvalitet servisa? Koji problemi prate zahteve za većim kvalitetom servisa? Navedite neki primer.

11.10. Kako se upravlja otkazima u slučaju distriburanih softverskih sistema?

11.11. Postoje dva osnovna modela interakcija između čvorova distribuiranog sistema. Koja su to dva modela i objasnite ih.

11.12. Šta je proceduralni poziv (RPC)? Šta je tačka povezivanja (stub)? Gde se nalazi udaljena procedura i koja je njena funkcija?

11.13. Šta je interakcija sa porukeama? Oja je učaga midlvera u tome?

11.14. Koja je razlika proceduralnog poziva (RPC) i razmene poruka? Objasnite..

11.15. Koje su najčešće funkcije midlvera? Kako midlver podržava interakciju čvorova?

11.16. Navedite i objasnite nivoe slojevite arhitekture distribuiranih klijent-server sistema. Koji su problemi dvoslojne rahitekture?

11.17. Koja je razlika klijent-server arhitekture sa «debelim» i «tankim» klijentom? Navedite prednosti i nedostatke primene «tankih» a posebno, «debelih» klijenata?

11.18. Koje su predmosti višeslojne arhitekture? A koji su njeni problemi?

11.19. Koja je korist od primene sistema sa distribuiranim komponentama?

11.20. Koji su nedostaci sistema sa distribuiranim komponentama?

11.21. Uporedite servisno-orijentisanu arhitekturu sa arhitekturom sistema sa distribuiranim komponentama. Koja ima bolje preformase? Zašto?

11.22. Na kojim principa rade sistemi ravnopravnih računara (peer-to-peer, or p2p)? Kako su njeniu čvorovi povezani? Kada se koristi p2p arhitektura?

11.23. Šta je polucentralizovana arhitektura sa ravnopravnim računarima? U čemu je njena prednost? Da li ona ima problem bezbednosti?

11.24. Šta e «softver-kao-servis» (SaaS) ? Koji su ključni elementi SaaS koncepta? Koje su njegove prednosti?

11.25. Koji su problemi primene SaaS koncepta?

11.26. Koja je razlika SaaS i SOA?

11.27. Na koje faktore morate d aobratite pažnjupri primeni SaaS?

11.28. Šta je dinamičko konfigurisanje softvera? Šta dozvoljava dinamičko konfigurisanje?

11.29. Šta je proširivost SaaS sistema? Koje su preoruke za realizaciju prišireniа SaaS sistema?

Lekcija 12: Servisno-orijentisano softversko inženjerstvo

12.1. Šta je veb servis? Koji tip interfejsa koristi veb servis?

Veb servis se definiše kao labavo povezana, ponovo upotrebljiva softverska komponenta koja sadrži diskretnu funkcionalnost, koja može biti distribuirana i programski pristupačna.

Interfejs veb servisa je interfejs koji obezbeđuje (provides) i definiše funkcionalnost servisa i parametre.

12.2. Šta su servisno-orijentisani softverski sistemi?

Servisno-orijentisani sistemi su sistemi koji primenjuju ponovno upotrebljive softverske komponente kojima se pristupa preko drugih programa, a ne direktno povezivanjem korisnika na sam servis.

12.3. Šta je koncept «softver kao servis»? Koja je korist od preimena ovog koncepta?

Sofver kao servis znači ponudu funkcionalnosti softvera udaljenim korisnicima na Internetu, umesto preko aplikacija instaliranih na računaru korisnika.

12.4. Kakva je arhitektura servisno-orijentisanih softverskih sistema? Ko su akteri u servisno-orijentisanim sistemima? Koja je njihova uloga?

Servisno-orijentisane arhitekture (SOA) je način razvoja distribuiranih sistema u kojima komponente sistema predstavljaju samostalne servise koji se izvršavaju na udaljenim računarima.

Korisnici servisa su klijenti koji traže servis koji im je potreban, i preko registra nalaze potreban servis i servis provajdera koji ga nudi.

Oni mogu da povežu svoju aplikaciju sa pronađenim servisom i da sa njim komuniciraju primenom standardnih servisnih protokola.

12.5. Šta je XML? Zašta se on koristi?

XML, jezik koji razume i čovek i računar, a koji definiše strukturu podataka, koristeći označen (tagovan) tekst, tj. koji koristi identifikator za određenim značenjem, koristi se za definiciju semene podataka.

12.6. Šta je SOAP? Čemu služi?

Ovo je standard za razmenu poruka koji podržava komunikaciju između servisa. On definiše neophodne i opcione komponente poruka koje razmenjuju servisi. Servisi u servisno-orijentisanim arhitekturama se ponekad nazivaju SOAP servisi.

12.7. Šta je WSDL? Čemu služi?

Web Service Dscription Language (WSDL) je standard za definisanje interfejsa servisa. On određuje kako se definišu servisne operacije (nazivi operacija, parametri, i njihovi tipovi) i veze servisa

12.8. Šta je WS-BPEL? Čemu služi?

Ovaj standard definiše jezik za specifikaciju radnog toka (workflow), tj. precesno orijentisane programe koji koriste nekoliko različitih servisa.

12.9. Šta je UDDI? Čemu služi?

Standard pretraživanja definiše komponente specifikacije servisa koje pomažu potencijalnim korisnicima servisa da ga otkriju na Internetu.

12.10. Navedite ključne standarde sa veb SOA?

1. WS-Reliability Messaging
2. WS-Security
3. WS-Addressing
4. WS-Transactions

12.11. Koji su nedostaci veb servisa?

Njihova primena zahteva značajne obrade podataka pri kreiranju, prenosu i interpretiranju XML poruka

12.12. Do čega je dovela primena nove paradigme u softverskom inženjerstvu – primena servisno-orijentisanog sistema. Šta radi servis?

Ova promena paradigme je sada ubrzana primenom inženjerstvom oblaka (cloud computing) u kome se servisi nude sa računarske infrastrukture instalirane kod provajdera, kao što su Google i Amazon.

12.13. Šta je URI? Čemu služi?

Da bi koristili veb servis, morate da znate gde se on nalazi na Internetu, tj. da znate njegov URI (Uniform Resource Identifier) i detalje njegovog interfejsa. Ovi detalji se zovu WSDL

12.14. Šta je WSDL? Koja tri aspekta podržava WSDL?

Detalji URI interfejsa, Šta servis radi, Kako servis komunicira, Gde se servis nalazi

12.15. Koji su elementi WSDL modela?

Elementi modela: Uvod, opis tipova podataka, opis interfejsa servisa, opis ulaznih i izlaznih poruka, protokol poruka, i specifikacija krajnje tačke, tj. lokaciju servisa (URI).

12.16. Šta je resurs u kontekstu servisno-orijentisanih sistema? Koje su osnovne operacije koje se mogu izvršiti nad jednim resursom? Koje četori akcije obezbeđuju HTTP i HTTPS?

Resursi imaju jedinstven identifikator, kao što je URL. Resursi su objekti koji upotrebljavu bitove, Kreiraj, Citaj, Azuriraj, Obrisi.

POST,GET,PUT,DELETE

12.17. Šta je RESTful servis? Koji princip projektovanja on podržava?

Osnovni element RESTful arhitekture je resurs, RESRTful servis ne mora da koristi XML, koriste HTTP I HTTPS protokole, koristi JSON,

12.18. Koja je primena RESTful servisa u kod servisa «oblaka», tj. klaud servisa?

Kod servisa <oblaka> tj. klaud servisa (cloud servises) mogu se istovremeno koristiti i RESTful i SOAP servisi za slučaj istog servisa ili istog resursa. To sada koriste provajderi , kao što su Microsoft, Google i Amazon. U ovom slučaju, korisnici servisa mogu da izaberu metod za pristup servisu koji im najviše odgovara.

12.19. Koji su problemi pri primeni *RESTFul servisa?

1. Kada neki servis ima složeni interfejs i kada on nije jednostavan resurs, može biti teško da se projektuje set (skup) RESTful servisa koji predstavljaju taj interfejs.

2. Ne postoji standard za opis RESTful interfejsa, te korisnik interfejsa mora da koristi samo neformalnu dokumentaciju radi rezumevanja interfejsa.

3. Kada koristite RESTful servise, morate upotrebiti sopstvenu infrastrukturu i upravljanje kvalitetom servisa i pouzdanošću servisa. U slučaju primene SOAP servis postoje standardi za dodatnu infrastrukturu kao što su WS-Reliability i WSTransaction.

12.20. Šta je inženjerstvo servisa? Koje zahteve on mora da podrži?

Inženjerstvo servisa je proces razvoja servisa za korišćenje od strane servisno-orijentisanih aplikacija. Servis mora da bude ponovno upotrebljiva apstrakcija koja može da bude od koristi u različitim sistemima.

12.21. Koje su tri faze inženjering servisa?

1. Utvrđivanje mogućih servisa, kada utvrđujete moguće servise koje bi mogli da iskoristite i kada definišete zahteve za servis.

2. Projektovanje servisa, kada projektujete logičke i WSDL interfejse servisa.

3. Implementacija i raspoređivanje servisa, kada razvijete i testirate programski kod komponeneta koje obezbeđuju servise i kada ga činite dostupnim korisnicima.

12.22. Koja su tri osnovna tipa mogućih servisa? Objasnite ih.

1. Uslužni servisi (utility services): Ovi servisi primenjuju neku opštu funkcionalnost koja se može koristiti u različitim poslovnim procesima. Primer uslužnog servisa je servis za konverziju valuta (npr. evra u dolare).

2. Osnovni servisi: Ovi servisi su povezani sa specifičnim poslovnim funkcijama. Na primer, registracija studenata za neku izborni predmet.

3. Servisi koordinacije ili procesa: Ovo su servisi koji podržavaju uopšteniji poslovni proces u kome obično učestvuje više aktera i aktivnosti. Na primer, sistem za podršku nabavki (roba, usluga, nalaženje isporučioca, plaćanje).

12.23. Koja je razlika servisa za zadatke od servisa za entitete?

Servisi za zadatke: Povezani su sa nekom aktivnošću.

Servisis za entitete: Povezani su sa nekim poslovnim entitetom

12.24. Kako bi birali servis preko Interneta? Na koja pitanja bi obratili pažnu?

1. U slučaju servisa entiteta, da li je servis (usluga) povezana sa jednoim logičkim entitetom koji se koristi u više poslovnih procesa? Koje se operacije obično sprovode nad navedenim entitetom, koji se treba podržati servisom?

2. Za slučaj servisa za zadatke, da li zadatak izvršava više ljudi u organizaciji? Da li su oni spremni da prihvate neophodnu standardizaciju koju uvodi servis koji treba da podržava ovaj zadatak?

3.Da li je servis nezavisan, tj. u kojoj meri zavisi od drugih servisa

4. Da li pri svom radu servis mora da održava svoje stanje? Servisi su bez stanja, što znači da oni ne održavaju svoje stanje. Ako se traži informacija o stanju, onda se mora koristiti baza podataka, a to može da ograniči ponovnu upotrebljivost servisa.

5. Da li servis mogu da koriste klijenti van organizacije? Na primer, servis koji obezbeđuje katalog proizvoda, može da ima i unutrašnje i spoljnje korisnike.

6. Da li se može desiti da različiti korisnici servisa imaju različite nefunkcionalne zahteve. Ako imaju, da li to znači da servis mora da nudi loše verzija servisa?

12.25. Koje su faze projektovanja interfejsa servisa? Navedite operacije koje taj interfejs treba da podrži.

1. Logičko projektovanje interfejsa, kada utvrđujete operacije koje su vezane sa servisom, njihove ulaze i izlaze, kao i izuzetke koje su povezane sa ovim operacijama.

2. Projektovanje poruka, kada vi projektujete strukturu poruka koje se šalju i primaju od strane servisa.

3. Izrada WSDL dokumenta, kada prevodite vaše logičko projektno rešenje i projektno rešenje poruka u apstraktan opis interfejsa napisan u WSDL.

12.26. Šta je katalog intefejsa servisa? Šta on sadrži?

MakeCatalog, Lookup, Search, Compare, CheckDelivery, MakeVirtualOreder

12.27. Kako se vrši implementacija i testiranje interfejsa?

Nema odgovor na ovo pitanje u ovoj lekciji

12.28. Kako se vrši raspoređivanje servisa, tj. omogućavanje korišćenja servisa? Koje informacije o servisu treba objaviti?

Raspoređivanje servisa omogućava korišćenje servisa, na veb serveru, od strane njegovih krajnjih korisnika. Za to je dovoljno postaviti fajl sa izvršnim softverom u određeni diretorijum veb servera. Potrebno je, onda, da obezbedite informaciju o raspoloživom servisu svim potencijalnim spoljnim korisnicima servisa, kako bi oni odlučili da li im servis odgovara za njihove servisne aplikacije. Informacija koju treba da uključite u opis servisa bi mogla da sadrži sledeće:

1. Informaciju o vašem biznisu, kontakt detaljima i dr. Ovo je potrebno radi izgradnje poverenja. Korisnici servisa treba da budu uvereni da se servis neće pogrešno ponašati.

2. Neformalna informacija o funkcionalnosti koju obezbeđuje servis. Ovo omogućava potencijalnim korisnicima da odluče da li im servis zadovoljava njihove potrebe. Kako je ova informasija pisana običnim jezikom, ona ne daje potpuno precizan semantički opis mogućnosti servisa.

3. Detaljni opis tipova u interfejsu i njihove semantike.

4. Informacija o prijavljivanju novih korisnika servisa, radi registracije koja im obezbeđuje dobijanje informacije o novim verzijam servisa.

12.29. Kako se vrši sastavljanje i konfigursanje servisa?

Osnovni princip servisno-orijentisanog softverskog inženjerstva je sastavljanje i konfigurisanje servisa prilikom kreiranja novog, složenijeg servisa. Ovo se može integrisati sa korisničkim interfejsom na veb pretraživaču, ili se može koristiti kao komponenta koja se koristi pri kreiranju nekog drugog složenijeg servisa. I RESTful i SAOP servisi se mogu koristiti za kreiranje novih, složenijih (kompozitnih) servisa sa proširenom funkcijonlnošću.

12.30. Navedi faze konstruisanja sistema sastavljanjem više servisa

1. Formulisanje šeme radnog toka: U ovoj početnoj fazi projektovanja sistema, koristite zahteve za složeni servis kao bazu za kreiranje idealnog projektnog rešenja servisa. Trebalo bi da kreirate vrlo apstraktnio projektno rešenje u ovoj fazi, sa namerom da dodajete detalje kada znate više informacija o raspoloživim servisima.

2. Otkrivanje servisa: U ovoj fazi vi vi tražite postojeće servise da bi ih eventualno uključili u sastav vašeg, složenijeg servisa. U prkasi, najčešće se koriste postojeći servisi koje organizacija već ima, te vi pretražujete karaloge lokalnih servisa. Naravno, ovde možete i da pretražujete servise provajdera kojima verujete, kao što su Oracle i Microsoft.

3. Izbor mogućih servisa: Vršite izbor servisa iz skupa prethodno utvrđenih mogućih servisa, da bi ih koristili u aktivnostima vašeg radnog toka. Glavni kriterijum izbora je funkcionalsnost servisa , ali možete uzeti u obzir trošak korišćenja servisa, kao i kvalitet servisa koji se nude.

4. Prerada radnog toka: Na osnovu informacija o servisima koje ste izabrali, sada treba da preradite radni tok dodavanjem detalja ili izbacivanjem pojedinih aktivnosti. Ovo se može iterativno ponavljati.

5. Kreiranje programa radnog toka: Apstraktno definisan radni tok se transformiše u izvršni program (primenom Jave, C# ili BPMN). Razvijate i veb-orijentisane korisničke interfejse da bi se omogućio pristup novom servisu sa bilo kog veb pretraživača.

6. Testiranje postavljenog servisa ili aplikacije: Testiranje postavljenog složenog servisa je složenije nego testiranje komponenata u slučaju da se koriste spoljni servisi.

12.31. Kako se vrši testiranje sastavlkenog servisa? Koji su problemo ovog testiranja?

Za inspekciju i za testiranje, koristi sa izvorni program kao osnov za planiranje testiranja. Međutim, kada se servisi nude od spoljnih provajdera, izvorni kod nije raspoloživ. Zbog toga testiranje servisno-orijentisanih sistema ne može da koristi poznate metode testiranje koje se zasnivaju na izvornom kodu.Sledeći problemi se javljaju pri testiranju sastavljenih (kompozitnih) servisa:

1.Spoljni servisi su pod kontrolom provajder servisa, a ne korisnika servisa. Servis provajder može ukinuti ove servise kada hoće ili da izvrši izmene na njima, što čini neprimenljivim sva obavljena testiranja servisa pre tih promena. Ovi problemi su vezani za softverske komponente i za njihove verzije. Međutim, trenutno ne postoje standardi koje rade se verzijama servisa.

2. Dugovečne verzije SOA za servise se dinamički povezane sa servisno-orijentisanim aplikacijama Ovo znači da neka aplikacija ne mora uvek da upotrebljava isti servis uvek kada

se izvršava. Zbog toga, testovi mogu biti uspešni kada je aplikacija povezana sa odrđenim servisom, li se ne može garantovati da će servis biti u upotrebi za vreme stvarnog izvršenja sistema.

3. Nefunkcionalno ponašanje servisa ne zavisi samo od aplikacije koja se testira. Servis može lepo da radi kada nije pod velikim opterecenjem. U praksi, ponašanje servisa može biti različito zbog zahteva koje stižu od drugih servisa.

4. Model placanja servisa može da doprinese visokoj ceni testiranja. Postoje različiti modeli placanja servisa: a) besplatan servis, b) plaćanje preko pretplate, i c) plaćanje prema upotrebi. Ako je servis besplatan, onda provajder servisa nece želeti da postavi servise koji su u testiranju. Ako je potrebna pretplata, onda korisnik servisa nije voljan da se pretplati na servis pre nego što on bude testiran. I konacno, ako se placa korišcenje, korisnici servisa mogu da ne prihvate trošak testiranja.

5. Postoji problem testiranja softverskog servisa sa akcijama kompenzacije. Oni mogu da zaviste od otkaza drugih servisa. Obezbediti da ovi servisi padnu za vreme testiranje je dosta teško.

Lekcija 13: Inženjerstvo softvera u realnom vremenu

13.1. Šta su sistemi u realnom vremenu? U čemu je razlika ovih sistema u odnosi na ostale sofverske sisteme? Šta su “meki” i “tvrdi” sistemi u realnom vremenu?

Softverski sistem u realnom vremenu je sistem čiji ispravan rad zavisi i od rezultata koji proizvodi i od vremena u kome se ti rezultati proizvedu.

Ugrađeni sistema u realnom vremenu su reaktivni sistemi. Oni reaguju na događaje u svom okruženju. Vremena odziva često zavise od fizičkih zakona.

Pod "mekim" sistemima u realnom vremenu podrazumevaju se sistemi koji neispravno rade ako se rezultati ne proizvode u skladu sa specificiranim vremenskim zahtevima. Pod "tvrdim" sistemima u realnom vremenu se podrazumevaju sistemi koji ne proizvode rezultate u skladu sa specifikacijom vremena reagovanja, i u tom slučaju, sistem je neuspešan.

13.2. Šta je podsticaj, a šta je ponašanje sistema u realnom vremenu? Koje vrste podsticaje postoje? Objsnite ih.

Podsticaj je događaj koji se javlja u okruženju softverskog sistem koji prouzrukuje da sistem reaguje na neki način – a to je signal ili poruka koje sistem šalje u svoje okruženje.

1. Periodički podsticaji: Javljaju se u određenim vremenskim intervalima. Na primer, sistem, uzima stanje sa senzora na svakih 50 milisekundi i pokreće akciju zavisno od vrednosti koju daje senzor.

2. Aperiodički podsticaji: Javlja se neregularno i nepredvidljivo i obično je utvrđen radom prekidnog mehanizma računara. Na primer, prekid koji pokazuje da je unos podataka završen i da su podaci raspoloživi u baferu.

Definisanje ponašanja sistema u realnom vremenu se može definisati listom podsticaja koje sistem prima, odgovarajućih odgovora sistema, i sa vremenom potrebnim da se proizvede odgovor sistema.

13.3. Koja je funkcija senzora, a koja- pokretača (actuator)?

Za svaki tip senzora postoji proces upravljanja senzorom koji prikuplja podatke sa ovih senzora. Procesi obrade podataka računaju zahtevane odgovore za podsticaje koje sistem prima.

Procesi upravljanja pokretača su povezani sa svakim pokretačem i upravljaju radom pokretača. Ovaj model omogućava brzo prikupljanje podataka sa senzora i omogućava obradu tih podataka i odgovarajući odgovor pokretača (actuator) do koga dolazi na kraju.

13.4. Navedite aktivnosti procesa projektovanja sistema u realnom vremenu.

1. Izbor platforme: Birate izvršnu platformu koju čine hardver i operativni sistem u realnom vremenu. Faktori za izbor: vremenska ograničenja, ograničenja napajanja, iskustvo tima za razvoj i ciljna cena sistema koji se razvija.

2. Identifikacija podsticaja/odgovora: Utvrđuju se podsticaji koje sistem treba da obra]uje i odgovori na svaki od njih.

3. Analiza vremena: Za svaki podsticaj i odgovor određujete vremensko ograničenje koje se odnosi i na podsticaj i na obradu odgovora. Na ovaj način se postavljaju rokovi za izvršenje procesa sistema.

4. Projektovanje procesa: Projektovanje procesa predstavlja objedinjavanje podsticaja i obradu odgovora u određeni broj konkuretnih procesa. Projektovanje počinjete izborm odgovarajućeg šablona projektovanja, a onda optimizujete arhitekturu procesa u skladu sa specifičnim zahtevima sistema koji projektujete.

5. Projektovanje algoritma: Za svaki podsticaj i odgovor, projektujete odgovarajući algoritam za realizaciju potrebnih obračuna. Određivanje algoritama se radi u ranoj fazi procesa projektovanja kako bi se odredio obim obrade podataka i potrebno vreme obrade.

6. Projektovanje podataka: Određujete informacije koje se razmenjuju između procesa i događaje koji koordinišu ovu razmenu informacija. Projektujete i strukturu podataka za upravljanje ovom razmenom informacija. Nekoliko konkurentnih procesa obično deli ove strukture podataka.

7. Planiranje procesa: Potrebno je da projektujete sistem planiranja koji će obezbediti da procesi počinju na vreme i da ostvaruju definisane rokove za završetak svog posla.

13.5. Zašto je potrebna sinhronizacija procesa? Šta je cilj sinhronizacije? Šta je kružni bafer? Cemu služe Get i Put operacije?

U radu sistema u realnom vremenu više procesa paralelno radi i deli pojedine resurse, vrlo je važno obezbediti adekvatnu koordinaciju njihovog rada. Kako u radu sistema u realnom vremenu više procesa paralelno radi i deli pojedine rasurse, vrlo je važno obezbediti adekvatnu koordinaciju njihovog rada. U tom cilju koriste se semfori, monitori, i kritični regioni. Ove mehanizme za sinhronizaciju procesa obezbeđuju operativni sistemi koje koristimo.

Kruzni bafer, obično se primenjuje kružna organizacija sa kružnim redosledom, koja upotrebljava strukturu podataka u vidu liste. Razlike u brzinama rada procesa se prilagođavaju bez kašnjenja u izvršenju procesa.

Put operaciju koristi proces koj proizvodi informaciju, a Get operaciju koristi proces koji koji koristi informaciju, tj. koji je preuzima iz bafera.

13.6. Šta je cilj statičke anaize sistema?

Kada ste odabrali platformu za izvršenje sistema, završili projektovanje arhitekture procesa i kada ste se odlučili za način planiranja izvršenja procesa, treba da proverite da li vaš sistem zadovoljava postavljene zahteve. To radite statičkom analizom sistema koristeći korišćenjem poznate vremenske konstante ponašanja komponenti sistema ili koristite simulaciju rada sistema.

13.7. Koji su problemi programiranja sistema u realnom vremenu?

Zbog korišćenja funkcija u operativnom sistemu, aplikacioni kod je teži za razumevanja, jer ne pokazuje sve operacije koje se odvijaju u realnom vremenu. Pored razumevanja programa, korisnik mora da razume i kako operativni sistem podržava rad u realnom vremenu preko sistemskih poziva. Zbog vremenskih ograničenja, pri razvoju sistema u realnom vremenu, ne možete da primenite objektno-orijentisan razvoj softvera u slučaju "teških" sistema u realnom vremenu. Primena OO programiranja podrazumeva pristup podacima unutar objekata u vidu atributa, a korišćenjem njegovih metoda za pristup atributima objekta. Ovaj dodatni kod utiče na snižanje performansi sistema, zbog čega se najšešće ne postižu zahtevane performanse sistema ako se koristi objektno-orijentisano programiranje.

13.8. Koji se prolemi javljaju pri programiranja sofvera u realnom vremenu, ako se primenjuje objektno-orijentisani programerski jezik? Šta se postiže primenom Java SE Real-Time Systems, verzijom Jave za programiranje sistema u realnom vremenu?

Primena OO programiranja podrazumeva pristup podacima unutar objekata u vidu atributa, a korišćenjem njegovih metoda za pristup atributima objekta. Ovaj dodatni kod utiče na snižanje performansi sistema, zbog čega se najšešće ne postižu zahtevane performanse sistema ako se koristi objektno-orijentisano programiranje.

Posebna verzija Jave je razvojena za razvoj ugrađenih sistema (Java SE Real-Time Systems) , i ona uključuje modifikovane mehanizme sa nitima, koji dozvoljavaju specifikaciju niti koje neće biti prekidane prikupljanje iskorišćenih podataka (garbage collection).

13.9. Zašto je potrebno koićenje posebnih šablona projektovanja sistema u realnm vremenu? Da li se šabloni mogu kombinovati prilikom projektovanja softverskog sistema?

U njih je ugrađeno znanje o organizaciji arhitekture sistema, o tome kada se ova arhitektura koristi, i o prednostima i nedostacima predložene arhitekture. Vi treba da koristite arhitektonske šablone da bi razumeli arhitekturu sistema i da bi imali početnu tačku kreiranja vašeg sopstvenog projektnog rešenja arhitekture sistema koji razvijate.

Mogu se kombinovati

13.10. Opiši šablon «Osmatraj i reaguj» (Observe and React), navodeći i njegove stimulanse, odgovore, procesi i upotrebu.

Osmatraj i reaguj (Observe and React): Ovaj se šablon koristi u slučajevima kada imate set senzora moji se rutinski osmatraju i prikazuju. Kada senzori jave neki događaj (npr. poziv na mobilnom telefonu), sistem reaguje aktiviranjem odgovarajućeg procesa koji radi sa tim događajem.

13.11. Opiši šablon «Kontrola okruženja» (Environmental Control), navodeći i njegove stimulanse, odgovore, procesi i upotrebu.

Upravljanje okruženjem (Environment Control): Ovaj šablon se koristi kada sistem koristi senzore koji obezbeđuju informacije o okruženju i pokretače (actuators) koji menjaju okruženje. Kao rekcija na promene u okruženju detektovane senzorima, šalju se upravljački signali pokretačima sistema.

13.12. Opiši šablon «Procesni kanal» (Process pipline) navodeći i njegove stimulanse, odgovore, procesi i upotrebu.

Procesni kanal (Process pipline): Ovaj šablon se koristi kada se podaci kada je potrebna transformacija podataka iz jednog oblika u drugi pre nego što se izvrši njihova obrada. Transformacija podataka se vrši sekvencijalnim izvršenjem više procesnih koraka koji se mogu i konkurentno (paralelno) izvršavati. Ovo omogućava vrlo brzu obradu podataka, jer posebno jezgro procesora ili poseban procesor izvršava svaku transformaciju.

13.13. Šta je analiza vremena i u čemu je njen značaj? Koja su tri ključna faktora projektovanja ugrađenih sistema?

U toj analizi vi proračunavate koliko često svaki proces mora da se izvrši da bi se obezbedila obrada svih ulaza i svih sistemskih odgovora tako da budu proizvedeni na vreme. Rezultati analize vremena se koriste kod odlučivanja sa kojom će frekvencijom svaki proces izvršavati svoje operacije i kako se ovi procesi moraju da planiraju od strane operativnog sistema u realnom vremenu.

Rokovi, Frekvencija, Vreme izvrsenja

13.14. Zašto se primenjuju posebni operativni sistemi u realnom vremenu pri primeni softvera u realnom vremenu? Zašto se ne koriste uobičajeni operativni sistemi?

Operativni sistem u realnom vremenu upravlja procesima i vrši dodeljivanje resursa za potrebe sistema u realnom vremenu. Da bi mogli efikasno da rade.

Linix I windows su izvrsne platforme sistema, danas se ugrađeni sistemi u realnom vremenu razvijaju uz korišćenje operativnih sistema u realnom vremenu (real-time operating systems – RTOS) kako bi mogli efikasno da rade. Primeri operativnih sistema u realnom vremenu su: Windows Embedded Compact, VxWorks i RTLinux

13.15. Koje su funkcije OS u realnom vremenu?

1. Časovnik u realnom vremenu (real-time clock), koji obezbeđuje informaciju koja je periodično neophodna procesu planiranja.

2. Program za obradu prekida (interrupt handler), koja upravlja aperiodičnim zahtevima sa servisom.

3. Planer (sheduler), koji je odgovoran za ispitivanje procesa koji se mogu izvršavati i za izbor jednog od tih procesa za izvršenje.

4. Menadžer resursa (resource manager), koji raspoređuje odgovarajuće memorijske i procesorske resurse procesima koji su planirani za izvršenje.

5.Dispečer (dispatcher), koji je odgovoran za startovanje izvršenja procesa.

13.16. Navedite dva nivoa prioriteta upravljanje procesima i objasnite ih.

1. Nivo časovnika (clock level), tj. nivo prioriteta koji se dodeljuje periodičnim procesima.

2. Nivo prekida (interrupt level), tj. najviši nivo prioriteta. On se dodeljuje procesima koji treba da obezbede vrlo brz odziv, tj. odgovor na neki podsticaj. Jedan od tih procesa je proces časovnika u realnom vremenu. Ovaj proces se ne zahteva ukoliko sistem ne podržava prekide.

13.17. Šta su periodički procesi?

Periodički procesi se moraju izvršiti u specificiranim vremenskim intervalima planiranim za prikupljanje podataka (data aquisition) i za kontrolu pokretača (actuator control). U skladu sa vremenskim zahtevima koje aplikacioni program određuje, operativni sistem u realnom vremenu (RTOS) uređuje izvršenje periodičnih procesa tako da oni mogu da zadovolje svoje rokove.

13.18. Šta su procesi vođeni prekidima?

Procesi koji moraju da brzo odgovore na asihrone događaje, se često vode prekidima (interrupt-driven). Mehanizam za računara rad sa prekidima vrši kontrolu prenosa određene memorijske lokacije. Ova lokacija ima instrukciju da skoči na jednostavan i brzi servis RTOS za rad sa prekidima.

13.19. Šta je planiranje bez predpražnjenja?

Planiranje bez predpražnjenja, (nonpreemptive scheduling): Posle postavljanja plana za izvršenje on se izvršava do kraja ili dok se ne blokira iz nekog razloga, kao što se dešava kada čeka na ulaz. To može da napravi probleme u slučaju da postoje procesi sa različitim prioritetima izvršavanja jer i procesi sa visokim prioritetom treba da čekaju završetak procesa sa niskim prioritetom. Znači, ovde nema procesa sa prečim pravom izvršenja.

13.20. Šta je planiranje sa predpražnjenjem?

Planiranje sa pretpraživanjenjem (preemptive scheduling): Izvršenje nekog procesa se može zaustaviti u slučaju da neki proces sa većim prioritetom zahteva servis. Proces sa većim prioritetom stiće preče pravo izvršenja od procesa sa nižim prioriteto, te se dodeljuje procesoru.

13.21. Kako se vrši upravljanje resursima u operativnim sistemima za rad u realnom vremenu?

Proces onda prelazi u listu spremnih procesa tj. u <listu spremnih> (<ready list>). Kada procesor završi izvršenje ovog procesa i postane raspoloživ za druge procese, aktivira se dispečer. On gleda <listu spremnih> procesa, da bi odredio sledeći proces za izvršenje na ovom procesoru.

Lekcija 14: Projektovanje pouzdanog proizvoda

14.1. Navedite i objasnite četiri kategorije grešaka.
14.2. Navedite greške koje ne dobodi do otkaza rada (pada) sistema.
14.3. Navedite pristupe koje vode poboljšanju pouzdanosti softvera.
14.4. Šta je pouzdanost (reliability), a šta je dostupnost (availability) sistema. Koja je povezanost pouzdanosti i dostupnosti sistema?
14.5. Šta je otkaz (kvar) sistema? Koja su dva problema koja često dovode do kvarova u sistemu?
14.6. Šta je pouzdanost sistema? Koja je metrika za pouzdanost?
14.7. Šta je verovatnoća otkaza na zahtev – POFOD? Dajte primer. Kada se upotrebljava POFOD?
14.8. Šta je stopa javljanja otkaza sistema – ROCOF? Dajte primer. Kada se koristi ROCOF?
14.9. Šta je srednje vreme otkaza – MTTF? Dajte primer. Kada se koristi MTTF?
14.10. Šta je dostupnost (availabity – AVAIL)? Dajte primer.
14.11. Šta definišu nepfunkcionalni zahtevi pouzdanosti?
14.12. Koja je korist od kvantifikovane pouzdanosti?
14.13. Navedite tri preporuke (uputstva) za specifikaciju pouzdanosti sistema.
14.14. Kako se postiže visok nivo funkcionalne pouzdanosti sistema? Šta obuhvata specifikacija funkcionalne pouzdanosti?
14.15. Navedite četiri četiri tipa zahteva funkcionalne pouzdanosti.
14.16. Šta je tolerancija grešaka (fault tolerance)?
14.17. Šta sadrži arhitektura sistema koji je otporan na greške?
14.18. Šta su zaštitni sistemi? Šta je cilj dejstva zaštitog sistema? Opišite funkcionalnost zaštitnog sistema. Kako zaštitni sistem radi?
14.19. Šta je samoosmatrajuća arhitektura? Kako se ostvaruje samoosmatrajuća arhitektura?
14.20. Šta je trostruka modularna redudantnost-TMR?
14.21. Šta je programiranje N verzija softvera? Kda ga primeniti?
14.22. Koja su moguća dodatnia pravila kompanija?
14.23. Kako dolazi do pogrešne interpretacije specifikacije?
14.24. Koja su mogućna rešenja problema pogrešne interpretacije specifikacije?
14.25. Koje su preporuke za programiranje za pouzdanost?
14.26. Koji su potrebni podaci za utvrđivanje pouzdanosti softvera?
14.27. Koje su faze statističkog testiranja radi merenja pouzdanosti?
14.28. Koje su teškoće u statitskičkom testiranju?
14.29. Šta je profil rada?

Lekcija 15: Analiza i ocena kvaliteta projektnog rešenja softvera

15.1. Šta je cilj upravljanja kvalitetom softvera?

15.2. Kada se primenjuje razvoj vođen planom, tj. formalizovano upravljanje kvalitetom (quality management – QM)? Šta je specifično za formalizovano upravljanje kvalitetom na organizacionom i na projektnom nivou?

15.3. Koja je razlika upravljanja kvalitetom i kontrole kvaliteta softvera?

15.4. Koja je uloga QM tima (tima koji upravlja kvalitetom projekta).

15.5. Šta je plan kvaliteta? Koja je stuktura plana kvaliteta?

15.6. Zašto je subjektivnost prisutna u oceni kvaliteta softvera?

15.7. Na koje odgovore QM tim traži odgovore pri pravljanju kvalitetom softvera?

15.8. Navedite atribute kvaliteta softverskog proizvoda.

15.9. Opiši proces ocenjivanja kvaliteta softvera.

15.10. U slučaju fizičkih proizvoda, kvalitet proizvoda se postiže ako se poštuje specifikacija proizvodnog procesa. Međutim, kod softverskih proizvoda to nije tako. Zbog čega? Šta je specifičnost procesa razvoja softverskog proizvoda?

15.11. Koje su teškože u određivanju atributa kvaliteta?

15.12. Šta je to «kultura kvaliteta»? Kada možemo da kažemo da se u nekoj organizaciji neguje «kultura kvaliteta»

15.13. Šta definišu standardi proizvoda, a šta definišu standardi procesa?

15.14. Šta standardi softverskog inženjerstva definišu?

15.15. Inženjeri obično ne vole da primenjuju standarde. Zašto? Kako bi ubedio svoje kolege da primenjuju standarde?

15.16. Da li menadžer projekta može da modifikuje standard koji želi da primeni? Obrazložite svoj stav.

15.17. Šta je ISO 9001? Koji su ključni procesi koje obuhvata standard ISO 9001?

15.18. Šta je uputstvo o kvalitetu?

15.19. Navedite sitandarde koji zu od značaja z akvalitet softvera.

15.20. Šta je recenzija kvaliteta? Šta je sve predmet recenzije? Koja je korist od recenzije?

15.21. Navedite aktivnosti procesa recenzzije softvera.

15.22. Šta je kontrola softvera? Kako se vrši kontrola programa?

15.23. Klasifikujte greške koje se javljaju pri razvoju softvera.

15.24. Kako se vrši kontrolu kvaliteta softvera i donosi odluke o kvalitetu softvera.

15.25. Pri primeni Scrum metode razvoja softvera, programiranje se vrši u paru. Zašto? Međutim, pored dobrih strana, programiranje u paru ima i negative spekte. Koji su problemi kontrole pri programiranja u paru?

15.26. Kako se vrši merenje softvera? Šta su metrike softvera? Koje su dve vrste metrike softvera?

15.27. Navedite metrike kontrole procesa. Koja je razlika metrike predviđanja i metrike kontrole?

15.28. Šta su metrike softverskog proizvoda? Navedite primere

15.29. Merenje softverskog sistema se može koristiti na dva načina. Koja su ta dva načina? Opišite ih.

15.30. Navedite spoljne i unutrašnje atribute kvaliteta. Koji su uslovi predviđanja spoljnjih atribita?

15.31. Opišite kako bi izvršili analizu kvaliteta komponente.

15.32. Koja je razlika dinamičkih i statičkih metrika?

Lekcija 14: Projektovanje pouzdanog proizvoda

1. Navedite i objasnite četiri kategorije grešaka.

1. Ljudske greške (human error): Ponašanje ljudi koje dovodi do unošenja grešaka u softverski sistem.

2. Sistemske greške (system fault): Karakteristika softverskog sistema koja dovodi do greške.

3. Greška u sistemu (system error): Pogrešno stanje sistema za vreme izvršavanja koje vodi od ponašanja sistema koje korisnici sistema ne očekuju.

4. Otkaz sistema (system failure): Događaj koji se pojavljuje u nekom vremenskom trenutku kada sistem ne isporučuje servis koji korisnici očekuju.

2. Navedite greške koje ne dobodi do otkaza rada (pada) sistema.

1. Ne izvršava se ceo program. Deo programa koji sadrži grešku ne izvršava se zbog uslova korišćenja sistema, koji ne dozvoljava korišćenje dela softvera koji sadrži grešku. Zbog toga, iako softver ima grešku, ona ne proizvodi nikakav efekat.

2. Greška je privremena. Stanje promenljive sistema ima pogrešnu vrednost prouzrokovanu izvršenjem programa sa greškom. Međutim, pre nego što dođe do korišćenja promenljive sa pogrešnom vrednošću koja dovodi do pada sistema, neki drugi ulazi u sistem mogu da svojom obradom dovedu do resetovanja stanja sa pogrešnom vrednošću promenljive, te ona opet ima ispravnu vrednost. Zbog toga, pogrešna vrednost promenljive, nije imala nikakav praktičan efekat u ovom slučaju.

3. Sistem sadrži mehanizam detekcije grešaka i zaštite. Zbog primene ovih mehanizama, pogrešno ponašanje sistema je otkriveno i otklonjeno pre nego što je došlo do ugrađavanja servisa sistema.

3. Navedite pristupe koje vode poboljšanju pouzdanosti softvera.

1. Izbegavanje greške (fault avoidance): Proces projektovanja i implementacije softvera treba da primeni pristupe u razvoju softvera koji pomažu u izbegavanju greške u projektovanju i u programiranju, te se na taj način smanjuje broj grešaka u sistemu.

2. Detekcija grešaka i njihovo otklanjanje (fault detection and correction): Procesi verifikacije (provere) i validacije (overavanja) se projektuju tako da otkriju i uklone greške u programu, pre njegove upotrebe.

3. Tolerancija grešaka (fault tolerance): Sistem je tako projektovan da se otkrivaju greške ili neočekivano ponašanje sistema u vreme izvršavanja ali se njime tako upravlja da ne dolazi do pada sistema.

4. Šta je pouzdanost (reliability), a šta je dostupnost (availability) sistema. Koja je povezanost pouzdanosti i dostupnosti sistema?

1. Pouzdanost (reliability): Verovatnoća rada bez grešaka u toku određenog vremena, i datom okruženju, a radi ostvarenja određene svrhe.

2. Dostupnost (availability): Verovatnoća da sistem, u određenom vremenskom trenutku, radi i isporučuje zahtevane servise.

Pouzdanost sistema nije neka apsolutna vrednost, jer ona zavisi i od toga i gde i kako se sistem koristi. Zato se pouzdanost sistema uvek razmatra u odnosu na okolnosti i okruženja njegove upotrebe.

5. Šta je otkaz (kvar) sistema? Koja su dva problema koja često dovode do kvarova u sistemu?

Otkaz sistema (failure) je spoljni događaj koji utiče na korisnike sistema.

1. Specifikacije softvera su često nekompletne iili netačne, i ostavljaju inženjerima softvera prostor za slobodnu interpretaciju kako sistem treba da se ponaša

2. Niko sem inženjera razvoja softvera ne čita specifikaciju softvera

6. Šta je pouzdanost sistema? Koja je metrika za pouzdanost?

1. Pouzdanost (reliability): Verovatnoća rada bez grešaka u toku određenog vremena, i datom okruženju, a radi ostvarenja određene svrhe.

Metrika Pouzdanosti se može specificirati kao verovatnoća javljanja pada sistema kada je on radi u okviru specificiranog okruženja njegovog rada.

7. Šta je verovatnoća otkaza na zahtev – POFOD? Dajte primer. Kada se upotrebljava POFOD?

Ako koristite ovu metriku, vi definišete verovatnoću otkaza sistema kada zahtevate neki njegov servis. Na primer, POFOD= 0,001 znači da postoja 1/1000 šanse da sa javi otkaz pri postavljenjanju tražnje za servisom.

8. Šta je stopa javljanja otkaza sistema – ROCOF? Dajte primer. Kada se koristi ROCOF?

Ova metrika definiše verovatnoću javljanja broja otkaza sistema relativno u odnosu na određeni vremenski period (na primer, na sat).

9. Šta je srednje vreme otkaza – MTTF? Dajte primer. Kada se koristi MTTF?

Recipročna vrednost od ROCOF je srednje vreme otkazivanja (mean time to failure - MTTF), koja se povremeno koristi kao metrika pouzdanosti. MTTF je prosečan broj vremenskih jedinica između otkaza sistema

10. Šta je dostupnost (availabity – AVAIL)? Dajte primer.

Dostupnost (availabity – AVAIL) je verovatnoća da če sistem raditi kada neko zahteva njegov servis. Na primer, dostupnost od 0,9999 znači da je sistem, u proseku, dostupan 99,99% u toku vremena rada

11. Šta definišu nepfunkcionalni zahtevi pouzdanosti?

Nefunkcionalni zahtevi pouzdanosti su specifikacije o zahtevanoj pouzadanosti i dostupnosti sistema upotrebom jedne od metrika pouzdanostu

12. Koja je korist od kvantifikovane pouzdanosti?

Specificirana kvantifikovana pouzdanost je korisna iz sledećih razloga:

1. Proces odlučivanja o zahtevanom nivou pouzdanosti pomaže razjašnjavanju šta su stvarne potrebe aktera sistema
2. Predstavlja osnovu za ocenjivanje kada treba prestati sa testiranjem sistema
3. To je sredstvo za ocenjivanje više strategija projektovanja s ciljem da se poveća pouzdanost sistema
4. Ako regulator treba da poboljša sistem pre nego što bude pušten u rad, onda je evidentiranje dostizanja zahtevane pouzdanosti važna za sertifikaciju sistema.

13. Navedite tri preporuke (uputstva) za specifikaciju pouzdanosti sistema.

1. Specificirajte zahteve dostupnosti i pouzdanosti za različite tipove otkaza

2. Specificirajte zahteve dostupnosti i pouzdanosti za različite tipove servisa sistema

3. Razmislite da li vam je visoka pouzdanost zaista potrebna.

14. Kako se postiže visok nivo funkcionalne pouzdanosti sistema? Šta obuhvata specifikacija funkcionalne pouzdanosti?

Visoki nivo pouzdanosti i dostupnosti softverskog sistema se postiže kombinacijom tehnika, kao što su:

1.izbegavanje grešaka (fault-avoidance)

2.detekcija grešaka (fault-detection)

3.tolerancija grešaka (fault-tolerance).

Ovi zahtevi funkcionalne pouzdanosti treba da specificiraju otkrivanje grešaka i akcije koje treba preuzeti koje treba da spreče pad sistema zbog tih grešaka

15. Navedite četiri četiri tipa zahteva funkcionalne pouzdanosti.

1. Zahtevi provere: Ovi zahtevi utvrđuju provere ulaza u sistem radi obezbeđenja otkrivanje nepravilnih ulaza, ili ulaznih podataka čije vrsu vrednosti van dozvoljenih opsega vrednosti, i to pre nego što počne obrada podataka.

2. Zahtevi oporavka: Ovi zahtevi pomažu da se sistem oporavi posle pada sistema. Oni se uglavnom svode na održavanje kopije sistema i njegovih podataka, i na specifikaciju kako treba povratiti servis sistema posle otkaza.

3. Zahtevi redudantnosti: Specificiraju redundantna (ponovljena) svojstva sistema koji obezbeđuju da pad jedne komponente ne dovede do pada celog sistema.

4. Zahtevi procesa: Ovo su zahtevi koji dovode do izbegavanja grešaka, kojim se primenjuje dobra praksa u procesu razvoja. Primena dobre prakse ima za cilj da smanji broj grešaka u sistemu.

16. Šta je tolerancija grešaka (fault tolerance)?

Tolerancija grešaka (fault tolerance) je pristup pouzdanosti u fazi izvršenja sistema koji uključuje mehanizme za nastavak rada i kada dođe do pojave grešaka u softveru ili hardveru, i kada je sistem u stanju greške.

17. Šta sadrži arhitektura sistema koji je otporan na greške?

Da bi se obezbedila tolerancija grešaka, mora da se projektuje arhitektura sistema tako da uključi redudantne (ponovljive) i različite hardverske i softverske komponente.

18. Šta su zaštitni sistemi? Šta je cilj dejstva zaštitog sistema? Opišite funkcionalnost zaštitnog sistema. Kako zaštitni sistem radi?

Zaštitni sistem (protection system) je specijalizovani sistem koji je povezan sa nekim drugim sistemom. On je najčešće kontrolni sistem nekog procesa, kao što su proizvodni procesi, ili sistemi za kontrolu opreme, kao što su vozovi bez mašinovođe.

Zaštitni sistem sadrži samokritičnu funkcionalnost koja je neophodna da sistem prebaci iz nebezbednog u bezbedno stanje.

19. Šta je samoosmatrajuća arhitektura? Kako se ostvaruje samoosmatrajuća arhitektura?

Samoosmatrajuća arhitektura (self-monitoring architecture) je arhitektura u kojoj je sistem projektovan da osmatra sam svoj rad i da preduzima akcije ako je otkriven neki problem.

Računanje se vrši posebnim kanalima, i rezultati ovih računanja sa upoređuju. Ako su rezultati identični i ako su istovremeno raspoloživi, onda je to indikator da sistem ispravno radi. Ako su rezultati različiti, onda je verovatno došlo do neke greške u sistemu. U tom slučaju, sistem aktivira izuzetak za pad sistema na izlazu. Ovo označava da je potrebno da se kontrola prebaci nekom drugom sistemu.

20. Šta je trostruka modularna redudantnost-TMR?

Kod TMP sistema hardverske komponente se repliciraju tri puta. Izlaz iz svake jedinice se šalje u komparator izlaza koji se implementira u obliku sistema za glasanje

21. Šta je programiranje N verzija softvera? Kada ga primeniti?

Upotrebom iste specifikacije, različiti timovi razvijaju isti softverski sistem. Ovako razvijene verzije istog softvera se izvršavaju na različitim kompjuterima

Zato se ovaj pristup koristi samo u slučajevima kada je nepraktično obezbeđivanje zaštitnog sistema koji štiti sistem od grešaka opasnih po bezbednost.

22. Kako dolazi do pogrešne interpretacije specifikacije?

U praksi je nemoguće ostvariti potpunu nezavisnost kanala u sistemu. I različiti timovi prave iste greške ili vrše pogrešnu interpretaciju delova projektne specifikacije. Postoji nekoliko razloga za ovu pogrešnu interpretaciju delova specifikacije:

1. Članovi različitih timova imaju istu kulturnu pozadinu ili imaju isto obrazovanje koristeći isti pristup i iste udžbenike

2. Ako su zahtevi netačni ili se oslanjaju na nerazumevanja okruženja sistema onda će te greške da se odražavaju u svakoj implementaciji sistema.

3. Detaljna specifikacija kritičkih sistema koja se dobija na osnovu sistemskih zahteva trebalo bi da obezbedi jedinstvenu definiciju sistemskog ponašanja.

23. Koja su mogućna rešenja problema pogrešne interpretacije specifikacije?

Jedan od načina da se izbegnu greške sa specifikacijom je da se razviju nezavisne detaljne specifikacije sistema i da se napišu u različitim jezicima. Jedan razvojni tim može da koristi formalnu specifikaciju, a drugi da koristi model baziran na stanjima, a treći – da koristi specifikaciju napisanu primenom prirodnih jezika. Ovaj način otklanja neke greške u tumačenu specifikacije.

24. Koje su preporuke za programiranje za pouzdanost?

1. Ograničite vidljivost informacija u programu.

2. Proverite ispravnost svih ulaza

25. Koji su potrebni podaci za utvrđivanje pouzdanosti softvera?

1. Broj otkaza sistema posle određenog broja zahteva za korišćenje servisa softvera.

2. Vreme ili broj transakcija između dva pada sistema plus ukupno proteklo vreme ili ukupan broj transakcija

3. Vreme popravke ili restartovanja sistema posle njegovog pada koje vodi ka gubljenju njegovog servisa

26. Koje su faze statističkog testiranja radi merenja pouzdanosti?

1. Analizaraju se postojeći sistemi istog tipa da bi se razumelo njihovo korišćenje u praksi.

2. Napravite skup test podataka koji odgovaraju utvrđenom profilu rada (operational profile).

3. Testirate sistem upotrebom ovih podataka i brojite broj i vrste otkaza do kojih dolazi.

4. Kada ste utvrdili statistički značajan broj kvarova (pada) sistema, računate pouzdanost softvera i određujete odgovarajuće vrednosti metrike pouzdanosti.

27. Šta je profil rada?

Profil rada (operational profile) softverskog sistema odražava način njegove primene u praksi. On sadrži sepcifikaciju vrste ulaza i verovatnoću njihovog korišćenja.

Lekcija 15: Analiza i ocena kvaliteta projektnog rešenja softvera

1. Šta je cilj upravljanja kvalitetom softvera?

Upravljanje kvalitetom softvera treba da doprinese da softverski sistem zadovolji potrebe svojih korisnika, da ostvari efikasnost i pouzdanost, a da bude isporučen na vreme i u okviru planiranog budžeta.

2. Kada se primenjuje razvoj vođen planom, tj. formalizovano upravljanje kvalitetom (quality management – QM)? Šta je specifično za formalizovano upravljanje kvalitetom na organizacionom i na projektnom nivou?

U slučaju razvoja velikih i složenih sistema, čiji razvoj traje i nekoliko godina, primenjuje se razvoj vođen planom. U ovakvim slučajevima, primenjuje se formalizovano upravljanje kvalitetom.

1. Na organizacionom nivou, upravljanje kvalitetom se bavi postavljanjem okvira organizacionih procesa i standarda koji treba da dovedu do softvera visokog kvalteta.

2. Na projektnom nivou, upravljanje kvalitetom uključuje primenu specifičnih procesa kvaliteta

3. Koja je razlika upravljanja kvalitetom i kontrole kvaliteta softvera?

upravljanje kvalitetom se bavi postavljanjem okvira organizacionih procesa i standarda koji treba da dovedu do softvera visokog kvalteta, dok upravljanje kvalitetom uključuje primenu specifičnih procesa kvaliteta

4.Koja je uloga QM tima (tima koji upravlja kvalitetom projekta).

QM tim u velikim kompanijama je odgovoran upravljanje procesom završnog testiranja. On je i odgovoran za proveru da testovi sistema pokažu da sistem zadovoljava zahteve

5. Šta je plan kvaliteta? Koja je stuktura plana kvaliteta?

Plan kvaliteta i opisuje željene atribute kvaliteta softvera i opisuje kako se ovi atributi mogu ocenjivati.

Struktura :

1. Uvod o proizvodu
2. Planovi proizvoda
3. Opisi procesa
4. Ciljevi kvaliteta
5. Rizici i upravljanje rizicima

6. Zašto je subjektivnost prisutna u oceni kvaliteta softvera?

1. Teško je napisati kompletne i nedvosmislene softverske zahteve.

2. Specifikacije obično integrišu zahteve koji dolaze od različitih kategorija stejkholdera

3. Neke karakteristike kvaliteta (npr, održivost) je teško direktno meriti i ne mogu se definisati na neki nedvosmisleni način.

7. Na koje odgovore QM tim traži odgovore pri pravljanju kvalitetom softvera?

1. Da li je sofver testiran na propisani način, i da li su testovi pokazali da on zadovoljava sve zahteve?

2. Da li softver dovoljno pouzdan da bi se koristio?

3. Da li su performanse softvera prihvatljive za normalno korišćenje?

4. Da li je softver koristan?

5. Da li je softver dobro strukturisan i razumljiv?

6. Da li su standardi programiranja i dokumentovanja primenjeni u procesu razvoja?

8. Navedite atribute kvaliteta softverskog proizvoda.

Bezbednost, Zastita, Pouzdanost, Otpornost, Robustnost, Razumljivost, Ispitljivost, Prilagodljivost, Modularnost, Slozenost, Prenosljivost, Upotrebljinost, Ponovna upotrebljivost, Efikasnost, Sposobnost ucenja.

9. Šta definišu standardi proizvoda, a šta definišu standardi procesa?

Standardi proizvoda: Odnose se na softverski proizvod i obuhvataju standarde dokumentovanja rada.

Standardi procesa: Oni definišu procese koje treba primeniti tokom razvoja softvera.

10. Šta standardi softverskog inženjerstva definišu?

1. Standardi proizvoda: Odnose se na softverski proizvod i obuhvataju standarde dokumentovanja rada, kao što je struktura dokumenata za definisanje zahteva, za specifikaciju klasa objekata, kao i standardi programiranja, koji upućuje na način korišćenja nekog programskog jezika.

2. Standardi procesa: Oni definišu procese koje treba primeniti tokom razvoja softvera. Oni obuhvataju dobru praksu razvoja. Uključuju definisanje specifikacije, procese projektovanja i validacije, proces alata za podršku i opis dokumenata koje treba napisti za vreme ovih procesa.

11. Da li menadžer projekta može da modifikuje standard koji želi da primeni? Obrazložite svoj stav.

Svaki menadžer projekta može da modifikuje standarde procesa u skladu sa okolnostima njegovog projekta.

12. Šta je ISO 9001? Koji su ključni procesi koje obuhvata standard ISO 9001?

ISO 9001 je više radni okvir za razvoj softverskih standarda. On postavlja opšte principe kvaliteta i procedure koje treba definisati

13. Šta je uputstvo o kvalitetu?

Uputstvo o kvalitetu treba da opiše ove relevantne procese i procesne podatke koji se prikupljaju i održavaju.

14. Šta je recenzija kvaliteta? Šta je sve predmet recenzije? Koja je korist od recenzije?

Recenzija (review) i kontrola (inspection) su aktivnosti obezbeđenja kvaliteta kojima se proverava rezultat projekta. Obuhvataju proveru softvera, softverske dokumentacije i zapisa otkrivanja grešaka i propusta, kao i nepoštovanja standarda.

Recenzija kvaliteta (quality review) se bazira na dokumentaciji koja je proizvedena u procesu razvoja softvera. Vrši se recenzija specifikacija softvera, projektnih rešenja, programa, modela procesa, planova testiranja, procedura upravljanja konfiguracijom softvera, standarda procesa i korisničkih uputstava. Recenzija treba da proveri konzistentnost i kompletnost dokumenata ili programskog koda, i da obezbedi poštovanje standarda, ako su definisani.

15. Navedite aktivnosti procesa recenzzije softvera.

Aktivnosti pre recenzije, sastanak tima recenzenata, i aktivnosti posle recenzije

16. Šta je kontrola softvera? Kako se vrši kontrola programa?

Kontrola programa je kolegijalna recenzija u kojoj članovi tima sarađuju u traženju grešaka u programu koji je u razvoju

Kontrola programa obuhvata članove timove iz različitih oblasti koji onda pažljivo analiziraju izvršni kod, liniju po liniju. Traže nedostatke i probleme, i onda ih opisuju na sastanki tima za kontrolu

17. Pri primeni Scrum metode razvoja softvera, programiranje se vrši u paru. Zašto? Međutim, pored dobrih strana, programiranje u paru ima i negative spekte. Koji su problemi kontrole pri programiranja u paru?

Kod koji razvija jedan član para, stalno proverava i ispravlja drugi član para. Kontroliše se svaka linija koda pre nego bude prihvaćena. Programiranje u paru zahteva duboko znanje programa, jer oba programera treba da do detalja razumeju program da bi nastavili razvoj. To duboko razumevanje i znanje je teško postići kod drugih procesa kontrole. Zbog toga, programiranje u paru može da pronađe greške u kodu koje ne bi otkrila formalna kotrola.

Ovo su potencijalni problemi: 1. Uzajamno nerazumevanje 2. Reputacija para 3. Radni odnosi

18. Navedite metrike kontrole procesa. Koja je razlika metrike predviđanja i metrike kontrole?

Metrike kontrole podržavaju upravljanje procesom, a metrike predviđanja pomažu u predviđanju karakteristika softvera

Postoje tri vrste metrika kontrole (procesa) koje se mogu da koriste:

1. Vreme potrebno za završetak posebnog procesa.

2. Resursi koji su potrebni za određeni proces

3. Broj javljanja određenog događaja

19. Šta su metrike softverskog proizvoda?

Broj ulaza/izlaza, Duzina koda, Ciklomaticka slozenost, Duzina identifikatora, Duzina povezivanja uslova, fogov indeks.

20. Merenje softverskog sistema se može koristiti na dva načina. Koja su ta dva načina? Opišite ih.

1. Radi određivanja vrednosti atributa kvaliteta sistema: Merenjem karakteristika komponenata sistema, i njihovim sabiranjem , možete da ocenite atribute kvaliteta sistema, kao što je, na primer, održivost.

2. Radi utvrđivanja komponenata sistema čiji je kvalitet bitan: Merenja mogu da utvrede komponente čije karakteristike odstupaju od norme. Na primer, utvrđujete najsloženije komponente. One imaju veću verovatnoću da sadrže greške, jer složenost povećava verovatnoću greške programera.

21. Koji su uslovi predviđanja spoljnjih atribita?

1. Unutrašnji atribut mora tačno da se meri.

2. Mora da postoji zavisnost između unutrašnjih (koji se mere) i spoljnjih atributa (koji su naš cilj)

3. Veza spoljnjih i unutrašnjih atributa mora da bude razumljiva, proverena, i izražena u vidu neke formule ili modela.

23. Koja je razlika dinamičkih i statičkih metrika?

Dinamičke metrike, koje se određuju merenjima u toku izvršenja programa

Statičke metrike, koje se određuju merenjem reprezentacija proizvoda