SE311 - Projektovanje i arhitektura softvera

# Lekcija 1: Osnove projektovanja softvera

1. **Šta omogućava apstrakcija unutar projektovanja softvera?**

Apstrakcija omogućava uklanjanje detalja iz opisa problema uz zadržavanje suštinskih osobina svoje strukture. Apstrakcija se koristi u procesu rešavanju problema za razdvajanje logičkih od fizičkih aspekata procesa projektovanja.

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

Rešavanje problema u procesu projektovanja softvera može biti izvršeno primenom različitih pristupa – korišćenjem puta nauke i korišćenjem puta inženjeringa. Kada je u pitanju put nauke, posmatranje i ekperimentisanje su dve glavne aktivnosti. Ključne aktivnosti puta inženjeringa su izgradnja i evaluacija. Aktivnosti su usmerene ka ostvarenju jasno definisanog cilja, gde se najpre pretpostavlja krajnji rezultat, a nakon toga sledi definisanje načina za realizaciju tog rezultata.

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

Faza održavanja podrazumeva 3 tipa održavanja softvera:

* Perfective maintenance – proširivanje i unapređenje operativnog softvera, uz dodavanje novih funkcionalnosti.
* Adaptive maintenance – sprovođenje nametnutih promena izvan specifikacije zahteva.

- Corrective maintenance – ispravka identifikovanih grešaka u toku operativnog rada sistema.

* Perfective maintenance se najčešće koristi u praksi. Ovaj tip održavanja projektantu softvera sužava prostor za rešenje, povećava početni trošak, a smanjuje dugoročne troškove.

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

* 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?**

Automatizacija,testiranje u raz okruzenjima, integraciono, testiranje koriscnickog iskustva

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

Pod razmenom znanja projektanata sa drugim učesnicima se podrazumeva razumevanje i

podela relevantnih informacija, definisanje tih informacija, realizovanje zamisli, analiza i projektovanje prototipa i na kraju testiranje, koje omogućava nove ideje i redefinisanje postojećih.

7. **Šta su kanali komunikacije projektanta softvera?**

Kanali komunikacije projektanta softvera su sredstva kojima se projektanti koriste za razmjenu informacija i razmjenu mišljenja tokom procesa projektovanja softvera. Neki od najčešćih kanala komunikacije su:

1. E-mail: projektanti se često koriste e-mailom za razmjenu informacija i dokumenata, kao i za planiranje sastanaka i razmjenu mišljenja.
2. Chat: projektanti se često koriste chat-om (instant messaging) za brzu razmjenu informacija i razmjenu mišljenja, posebno u slučaju kada se nalaze na različitim lokacijama.

8. **Koja je uloga ograničenja u procesu projektovanja softvera?**

Uloga ograničenja u procesu projektovanja softvera je da pomažu projektantima da se fokusiraju na ključne funkcionalnosti i da izbjegnu probleme koji mogu uticati na kvalitet i rokove projekta. Ograničenja se mogu odnositi na vrijeme, budžet, resurse, tehnologiju i druge faktore koji utiču na projekat.

9. **Šta je metoda prepoznavanja?**

To 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?**

Proces projektovanja unapređuje razvoj softvera opisom načina koji rešava identifikovani problem.

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

Dugoročni plan izmena je plan koji se koristi za razvoj, implementaciju i održavanje sistema u dužem vremenskom periodu. On obično sadrži detaljne informacije o kratkoročnim i dugoročnim ciljevima, aktivnostima, resursima, rokovima i budžetima. Dugoročni plan izmena se koristi kada se razvija i održava softver ili IT sistem koji se koristi dugoročno, a koji se očekuje da će se menjati i razvijati tokom vremena.

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

Proces projektovanja pojednostavljuje razvoj softvera opisom načina koji rešava identifikovani problem.

13. **Šta je model rešenja?**

Model rešenja je opis sistema koji se koristi za rješavanje određenog problema ili zadatka. On obično sadrži detaljne informacije o arhitekturi, komponentama, interfejsima, protokolima i algoritmima koji se koriste u sistemu. Model rešenja se koristi kao vodič za projektante i developer-e prilikom razvoja i implementacije sistema, kao i za testiranje i održavanje sistema.

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

Neusaglašenost u procesu projektovanja softvera može nastati usled neodrživosti projektovanja i zahteva, neodrživosti projektovanja i specifikacije, kao i nesamoodrživog projektovanja koje se kao takvo ne može primeniti sa uspešnim ishodom. Neusaglašenost se može otkriti u toku prerađivanja određenih informacija i upoređivanja informacija dobijenih iz različitih pogleda na softver, tokom procesa razvoja softvera, kao i tokom implementacije softvera.

# Lekcija 2: Arhitektonske strukture, pogledi i stilovi

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

Ukoliko je potrebno ostvariti visoke performanse softvera potrebno je:

* ispitati potencijalni paralelizam kroz dekomponovanje sistema i sinhronizaciju sistema
* identifikovati potencijalna uska grla performansi

2**. Šta predstavlja cilj pisanja softverske dokumentacije?**

Cilj je obezbediti dovoljno informacija tako da arhitektura sistema može biti analizirana i da su iz navedene analize vidljive mogućnosti sistema da ispuni zahteve atributa kvaliteta.

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

Korisnici dokumentacije softverske arhitekture su članovi tima koji imaju različite uloge u timu, korisnici softvera, naručioci softvera.

4. **Šta omogućavaju različiti pogledi na softversku arhitekturu?**

Različiti pogledi na softversku arhitekturu 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. Različiti pogledi omogućavaju različite ciljeve i različite upotrebe arhitekture. Treba naglasiti da nijedan pogled ne može u potpunosti predstaviti arhitekturu.

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

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. Svaki pogled naglašava određene aspekte sistema dok ignoriše druge aspekte, a sve u interesu rešavanja problema.

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

Najčešće za implementaciju, reinženjering i testiranje. \*

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

Primena različitih stilova na softversku arhitekturu omogućava kombinovanje najboljih opcija odabranih stilova. \*

8. **Šta je element softverske arhitekture?**

Svaki **element softverske arhitekture** predstavlja rezultat pažljivog i preciznog projektovanja u cilju ispunjavanja zahteva za kvalitet softvera, a i zahteva nametnutih od strane logike poslovanja organizacije za koju se softver projektuje.

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

Zato što je baš čitalac krajnji korisnik dokumentacije. 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. Čitalac softverske dokumentacije koji primeti da je dokument pisan tako da njemu bude razumljiv, čitaće dokument svaki put kada mu bude potrebna pomoć u razumevanju softvera, dok ukoliko čitalac primeti da je dokument nerazumljiv neće ga koristiti naredni put. Dokumentacija napisana za čitaoca biće čitana dok dokumentacija napisana tako da bude pogodna za pisca neće.

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

Primenom pravila za pisanje softverske dokumentacije postiže se kvalitetna, konzistentna i potpuna dokumentacija.

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

Jedan od zadataka u dokumentovanju pogleda na arihtekturu softvera je odlučivanje koje osobine elemenata treba dokumentovati. Svaki stil ima sopstveni rečnik (tipova elemenata i odnosa). Svojstva gotovo uvek uključuju ime elementa, kratak opis i ulogu ili odgovornost u arhitekturi softvera. Često se može desiti da i kombinacija stilova korišćenih u određenoj softverskoj arhitekturi može omogućiti različit pogled. Zajedno predstavljeni stilovi treba da budu dosledno opisani i spremni za upoređivanje i selekciju.

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

Dokumentacija se održava aktuelnom i potpunom, što je vrlo bitno.

13. **U kojim slučajevima se koristi formalna notacija?**

Formalne notacije **-** Pogledi su opisani u notaciji koja ima preciznu (matematički zasnovanu) semantiku. Formalna analiza je u tom slučaju moguća. 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?** Nepotrebno ponavljanje, dvosmislenost, pisanje dokumentacije koja nije fokusirana na čitaoca, zastarela ili nepotpuna dokumentacija, strukturno neorganizovana dokumentacija...

# Lekcija 3: Stilovi softverskih modula

1. **Kada se koristi izraz modul softvera?**

Izraz modul se koristi za objašnjavanje strukture softvera koja uključuje jedinice programskog jezika poput: C, Java ili C# klase, PL/SQL procedure ili opšte grupacije programskog koda kao što su Java paketi. Modulima je moguće dodeliti svojstva, definisati odgovornost u softveru i analizirati karakteristike. Moduli se mogu dodatno razložiti na nove module ili spojiti u jedan modul.

2. **Šta omogućavaju različiti pogledi na softversku arhitekturu?**

Pogledi na softversku arhitekturu omogućavaju izvršavanje nekih od mogućih ciljeva kao što su: obrazovanje budućih članova razvojnog tima, korisnika, naručioca softvera, komunikaciju između različitih korisnika sistema, osnovu za izgradnju, analizu performansi i pouzdanost.

3. **Koja je osnovna relacija stila razlaganja?**

Osnovna relacija stila razlaganja (dekompozicije) je „is part of“ – „je deo od“.

4. **Na koji način je stil razlaganja pogodan za nove članove razvojnog tima?**

**Pregled stila razlaganja** je pogodan za proces učenja o sistemu za nove članove razvojnog tima. Softverski projektant može izvršiti podelu radnih zadataka članovima tima po modulima dobijenim u stilu razlaganja. Novi članovi se mogu usredsrediti samo na određeni deo sistema koji je značajan za njih, a zatim mogu pratiti veze sa ostalim identifikovanim modulima sistema. Ne moraju znati celokupan sistem ili sve funkcionalnosti sistema da bi mogli da rade na implementaciji ili modifikaciji određenog modula.

5. **Po čemu se stil upotrebe razlikuje od stila razlaganja?**

**Stil razlaganja** se koristi za podelu sistema na implementacione jedinice i opisuje organizaciju programskog koda u vidu modula i podmodula, prikazivajući kako su odgovornosti i funkcionalnosti, dok stil upotrebe prikazuje na koji način moduli zavise jedan od drugog, pomeže u planiranju i definisanjju podskupova sistema koji se razvija. Relacija stila razlaganja je „is part of“ – „je deo od“, dok je relacija stila upotrebe „depends on“ – „zavisi od“. Kod stila razlaganja nije dozvoljena upotreba petlji, dok stil upotrebe podržava pravilno korišćenje petlji ili zavisnosti da bi se održao inkrementalni prikaz.

6. **Šta podrazumevaju inkrementalni podskupovi modula?**

Inkrementalni podskupovi modula podrazumevaju da moduli moraju biti pravog nivoa granularnosti. Stil upotrebe je pogodan za planiranje inkrementalnog razvoja.

7. **Koje su prednosti stila generalizacije na raspolaganju projektantu softvera?**

Stil generalizacije identifikuje module na osnovu zajedničkih osobina i koristi se u objektno-orijentisanim sistemima uz korišćenje nasleđivanja i povezivanja između modula.

8. **Na koji način je omogućeno "produžavanje" modula primenom stila generalizacije?**

Uglavnom je lakše razumeti modul koji je drugačiji od drugog poznatog modula nego razumeti potpuno nov modul od početka. Generalizacija je mehanizam za proizvodnju inkrementalnih opisa u cilju izgradnje potpunog opisa modula.

9. **Kako se prikazuje softverska arhitektura kroz stil slojeva?**

Slojeviti stil 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. Relacija ovog stila je "dozvoljeno da koristi" - "allowed to use“. Softverska arhitektura se kroz slojeviti stil prikazuje uglavnom odozgo nadole. Promena na donjem sloju softverske arhitekture može biti sakrivena iza njegovog interfejsa i ne utiče na slojeve iznad u hijerarhiji.

10.  **Koji stilovi softverske arhitekture su pogodni za korišćenje uz stil slojeva?**

Stilom slojeva može da se prikaže i sloj koji se sastoji od više modula kroz stil razlaganja. Višeslojni stil je ujedno i stil komponenata i konektora, jer se skupljaju i prikazuju komponente izvršavanja.

11. **Šta su aspekti?**

Aspekti su specijalni moduli odgovorni za primenu unakrsnih koncepata. Potiču iz terminologije aspektno-orijentisanog programiranja (AOP).

Prikaz aspekata treba da sadrži informacije o svakom modulu i načinu povezivanja sa drugim modulima koji zahtevaju funkciju unakrsnog kretanja. Stil aspekta je primenjiv kroz nasleđivanje klasa i interfejsa, korisničkih biblioteka ili drugih alternativa. Primenom aspekata omogućeno je poboljšanje izmene modula koji spadaju u funkcionalnosti poslovnog domena sistema.

12. **Koje informacije sadrži prikaz aspekta u softverskoj dokumentaciji?**

Prikaz aspekata treba da obuhvati detalje o svakom modulu i načinu povezivanja sa drugim modulima koji zahtevaju funkciju unakrsnog kretanja.

13. **Kada se koristi model podataka?**

Model podataka opisuje strukturu podataka unutar sistema kao i veze između entiteta. Pomaže u procesu implementacije i omogućava unapređenje performansi i modifikacija sistema. Model podataka se često crta upotrebom ERD dijagrama ili UML klasnog dijagrama.

14. **Koji su tipovi modela podataka?**

Konceptualni model podataka – fokus je na entitetima i njihovim vezama.

Logički model podataka - evolucija konceptualnog modela podataka preko tehnologije upravljanja podacima (poput relacionih baza podataka).

Fizički model podataka - implementacija entiteta podataka. Omogućava optimizaciju, kreiranje primarnih ključeva i indeksa i vrši optimizaciju performansi.

15. **Da li primena stilova arhitekture usložnjava ili olakšava proces projektovanja softvera projektantu softvera?**

Primena odgovarajućih stilova arhitekture može znatno da olakša proces projektovanja softvera projektantu softvera. Svaki stil nudi svoje pogodnosti, ali sa sobom nosi i ograničenja. Stilovi se ne trebaju primenjivati po svaku cenu. Pre primene određenog stila treba dobro razmisliti da li ima smisla koristiti ga za odabrani sistem.

# Lekcija 4: Stilovi povezivanja softverskih komponenata

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

Komponente i konektore je moguće prikazati neformalnom notacijom. Pored naziva komponenti potrebno je precizirati i njihovo značenje. To je moguće izvršiti u dodatnom opisu na dijagramu (korišćenjem ključa koji objašnjava svaki nacrtani simbol na dijagramu).

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

Komponente su glavne procesne jedinice i skladišta podataka. Komponenta ima skup portova preko kojih se ostvaruje komunikacija sa drugim komponentama (preko konektora). Ima ime i portove, koji predstavljaju potencijalne interakcije komponente sa okruženjem. Može biti povezana samo za konektor, a ne za druge komponente.

3. **Kada se koristi stil toka podataka?**

Ovaj stil se primenjuje uokviru domena gde se obavlja obrada podataka kroz nekoliko transformacionih koraka. 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?**

Kod peer-to-peer stila interakcija može biti inicirana od bilo koje komponente, jer svaka komponenta može delovati kao klijent ili server.

Kod klijent – server stila, svaki server odgovara na servise zatražene od strane klijenta.

5. **Šta podrazumeva grafičko predstavljanje pogleda? Tekstualni opis svih identifikovanih elemenata.**

Grafičko predstavljanje i dokumentovanje pogleda komponenta i konektora:

* 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 složeni oblici interakcija kao što su recimo kanal komunikacije orijentisan ka transakcijama između servera baze podataka i klijenta. Primeri konektora su: poziv za servisiranje, asinhrone poruke, tokovi podataka.

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

Pouzdanost (utvrđivanje pouzdanosti sistema), performanse (analiza mogućnosti, testiranje vremena potrebnog za odgovor), bezbednost (određivanje potencijalnih ranjivih tačaka sistema), funkcionalnost (razumevanje funkcija), konzistentnost (proces analize), nivo (definisanje procedure izgradnje sistema i raspoređivanje komponenti), potrebni resursi (potrebni hardver sistema).

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

* Prikaz načina funkcionisanja sistema
* Rukovođenje procesom razvoja softvera kroz specifikaciju strukture i ponašanja elemenata u toku izvršavanja programa
* Pomoćno sredstvo za projektovanje softvera radi razumevanja atributa kvaliteta (performanse, bezbednost, dostupnost...) u toku izvršavanja programa

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

Pogled komponente i konektora može biti kombinovan sa pogledom na module softverske arhitekture i stilom cevi i filtera.

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

* + 1. Stil toka podataka
    2. Stil poziv-povratak
    3. Stil peer-to-peer
    4. Stil softverske arhitekture zasnovane na događajima
    5. Stil cevi i filtera
    6. Stil klijent-server
    7. Servisno-orjentisani stil
    8. Publish-subscribe stil
    9. Stil deljenih podataka

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

Komponente pružaoci usluga su serveri. Može postojati jedan pružalac usluga ili više pružaoca usluga koji pružaju skup usluga preko jednog ili više portova. Primer: Web aplikacija

(korisnici koriste web browser za korišćenje aplikacije, a komponente se pokreću na Tomcat/Glassfish serveru)

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

Klijent – komponenta koja poziva servise serverske komponente

Server – omogućava servise klijentu

Konektor – koji koristi klijent za pozivanje servisa na serveru (upit/odgovor)

# Lekcija 5: Stilovi alokacije i hibridni stilovi

1. **Šta se dobija specijalizacijom stilova alokacije?**

Mogućnost ponovnog korišćenja stila u različitim delovima određenog sistema.

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

* Stil raspoređivanja
* Stil instalacije

3. **Koja je osnovna relacija stila alokacije?**

Relacija koja se koristi u stilu alokacije je dodeljen prema ("allocated to").

4. **Na koji način se vrši specijalizacija stila raspoređivanja?**

Specijalizacija stila raspoređivanja – „Tiered Distribution“ šablon, koji propisuje raspodelu softverskih komponenti u višeslojnoj arhitekturi na hardverske elemente koji se koriste za pokretanje, je razvio „Microsoft“. On omogućava generički stil primene.

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

Kombinovani pogled najčešće smanjuje broj pogleda na softversku arhitekturu u dokumentu, jer zamenjuje više različitih pogleda.

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

Da. Moguće je kombinovati različite komponenta i konektor poglede, pogled raspoređivanja sa servisno-orijentisanim pogledom ili komunikaciono-procesnim pogledima, pogled raspoređivanja i pogled instalacije, pogled razlaganja i dodeljivanja radnih zadataka, generalizaciju i aspekte.

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

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

Stil dodeljivanja radnog zadatka pomaže u planiranju i upravljanju resursima timova, u dodeljivanju odgovornosti za izgradnju i za objašnjanjavanje strukture projekta.

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

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

Primena Kručtenovih pogleda omogućava sagledavanje arhitekture softvera iz logičkog (klasni dijagram, dijagram stanja), procesnog (dijagram aktivnosti), fizičkog (dijagram komponenti), pogleda raspoređivanja (dijagram komponenti) i scenarija (use case) softverskih komponenti.

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

Kombinovanje pogleda prikazuje instalacione fajlove i hardverske elemente na koje se postavljaju instalacioni fajlovi.

11. **Šta je specijalizacija stilova alokacije?**

Specijalizacija stila alokacije se vrši u slučaju potrebe za ponovnim korišćenjem stila u različitim delovima određenog sistema.

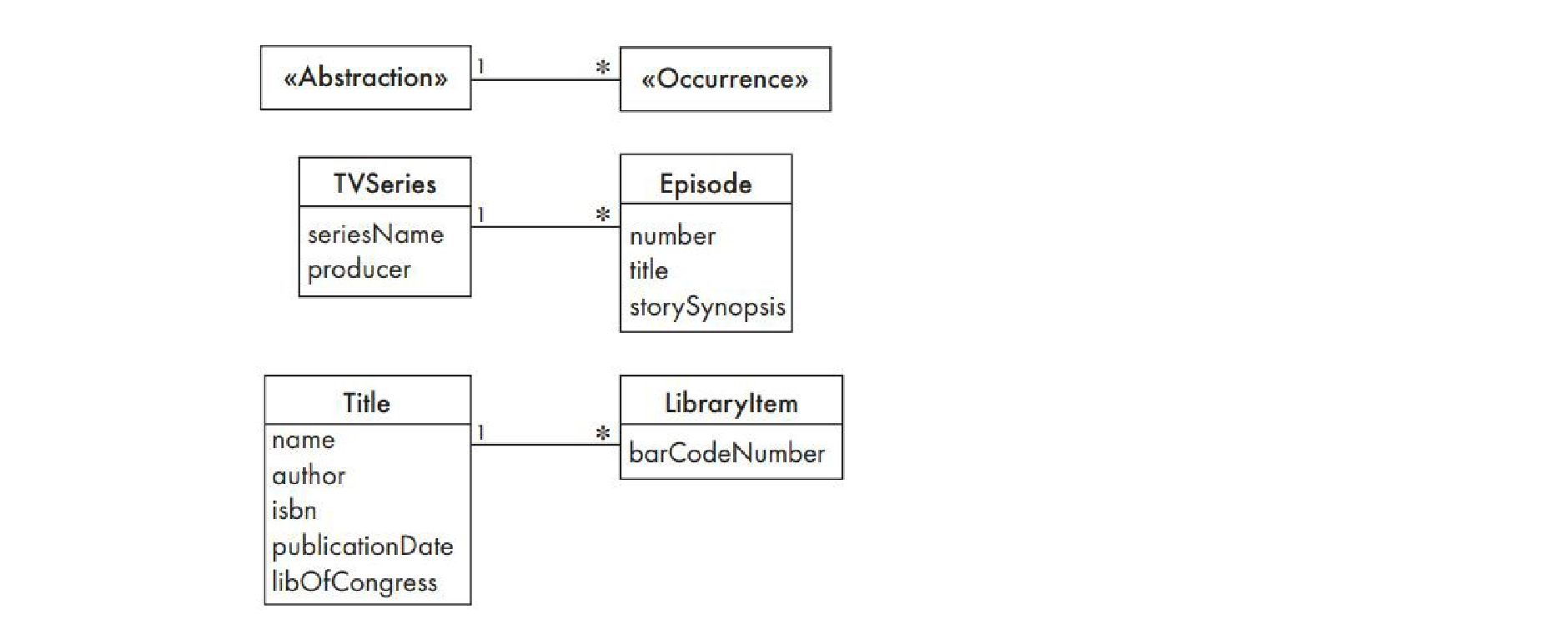
12. **Šta su kombinovani pogledi na softversku arhitekturu? Koji su načini za kreiranje kombinovanje pogleda?**

Kombinovani pogled najčešće smanjuje broj pogleda na softversku arhitekturu u dokumentu, jer zamenjuje više različitih pogleda. Povezivanje različitih pogleda na softversku arhitekturu moguće je korišćenjem asocijacija. Najjednostavniji način za prikaz asocijacije između dva pogleda je da se pogledi grupišu u jedan kombinovani pogled.

# Lekcija 6: Upotreba šablona projektovanja softvera

1. **Navedite primer kada je moguće koristiti šablon Abstraction-Occurrence.**

Sve epizode televizijske serije. Epizode imaju istog producenta, naziv serije, a različite priče po epizodi.



2. **Koja su ograničenja u primeni šablona Abstraction-Occurrence?**

Potrebno je prikazati članove svakog skupa pojava bez dupliranja zajedničkih informacija i izbeći rizik od nekonzistentnosti koja bi rezultovala promenu zajedničkih informacija samo u određenim objektima. Potrebno je modelovati rešenje koje omogućava maksimalnu fleksibilnost sistema.

3. **Da li šablon General Hierarchy koristi nasleđivanje?**

Upotreba nasleđivanja predstavlja anti-šablon. Ne podrazumeva se da svaka hijerarhija mora biti hijerarhija nasleđivanja.

4. **Koja asocijacija se koristi u šablonu General Hierarchy?**

Asocijacija "podređen" može biti opciono na više ("optional-to-many") ili više na više ("many- to-many"). Ako je asocijacija više na više, u takvoj hijerarhiji čvor može imati mnogo objekata ispod sebe.

5. **U kojim situacijama je potrebno koristiti šablon Player-Role?**

Primenom šablona Player-Role moguće je definisati različite uloge koje objekat određena klasa može

izvršavati.

6. **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.

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

Da. Ponovno korišćenje metode u klasi iz druge klase minimizira razvojne troškove.

8. **Da li primena šablona Delegation duplira programski kod?**

Ne. Dupliranje programskog koda spada u anti-šablone.

9. **Šta predstavlja nepromenljiv objekat?**

Nepromenljiv objekat je onaj koji ima stanje koje se nikad ne menja nakon njegovog kreiranja. Omogućava ga šablon Immutable.

10. **Koji su srodni šabloni šablonu Immutable?**

„Read-only interface“ pruža iste mogućnosit kao Immutable.

11. **Kada se koristi šablon Read-Only Interface?**

Primenom šablona Read-Only Interface moguće je definisati klase sistema koje mogu da modifikuju atribute objekata koji su nepromenljivi. Ovaj šablon je usko povezan sa šablonom Immutable.

12. **Koje su prednosti korišćenja šablona Read-Only Interface?**

Primenom šablona Read-Only Interface moguće je definisati klase sistema koje mogu da modifikuju atribute objekata koji su nepromenljivi. Ovaj šablon je usko povezan sa šablonom Immutable.

13. **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.

14. **Koji su srodni šabloni šablonu Proxy?** Šablon „Delegation“.

15. **Šta predstavlja primetan sloj?**

To je drugo proširenje objektnog klijent-server okvira (OCSF).

16. **Koji su glavni nedostaci primene šablona projektovanja?**

Potrebno je razumeti prednosti šablona i pažljivo donositi odluke o projektovanju softvera primenom šablona. Često se primenom šablona mogu doneti nepravilne odluke koje mogu proces projektovanja odvesti u pogrešnom smeru.

17. **Na koji način primena šablona omogućava povećanje fleksibilnosti projektovanog sistema?**

Vrlo je bitno da se šabloni ne primenjuju tek tako, jer u određenim slučajevima mogu izazvati više štete, nego koristi. Potrebno je modelovati rešenje koje omogućava maksimalnu fleksibilnost sistema, primenom odgovarajućeg šablona. Svaki šablon je potrebno detaljno dokumentovati i specificirati kako bi se održao u kasnijim fazama razvoja softvera. Šablon Facade je samo jedan od šablona koji poboljšava fleksibilnost sistema.

# Lekacija 7: Strategije i metodi projektivanja softvera

1. **Na koji način se dobija deklarativno znanje o korišćenju metoda?**

Deklarativno znanje je opis zadataka koje treba rešiti po fazama procesa projektovanja. Predstavlja se opisom: „potrebno je uraditi ovo, a onda uraditi to“.

2. **Kako skup heuristika utiče na aktivnosti u procesu projektovanja softvera?**

Skup heuristika: daje smernice o definisanju određenih aktivnosti unutar procesnog dela i organizacije određenih klasa problema. Bazirani su na iskustvu i prethodnoj upotrebi metode sa specifičnim domenom problema. Ubraja se u proceduralno znanje. Generišu se u određenom vremenskom periodu, a iskustvo primene heuristika se stiče korišćenjem metode u širem problemskom domenu.

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

Nastaju usled spoljnih faktora kao što je npr. struktura razvojne organizacije.Tehnički problemi koji mogu nastati primenom metoda projektovanja softvera:

* Mehanizam prenosa znanja
* Korišćenje metode projektovanja
* Korišćenje metode

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

Svaka metoda projektovanja softvera nudi formu "Dizajn virtuelne mašine" (DVM). DVM predstavlja okvir koji projektant softvera treba da razvije i pridržava se u toku projektovanja. Nudi sloj apstrakcije iznad fizičke mašine.

5. **Šta su dekompozicione metode? Kako se primenjuju?**

Dekompozicione metode su jedna od grupa metoda projektovanja softvera. Primenjuju pogled „topdown“ u procesu projektovanja softvera sa ciljem razvijanja modela projektovanja kroz proces podele sistema. Strategija dekompozicije je podela glavnog zadatka programa na manje zadatke (potprograme).

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

Korišćenjem pravougaonika za označavanje oblika reprezentacije, ovalnog oblika za označavanje proceduralnog koraka i luka za označavanje koraka u sekvenci. Svaki proceduralni korak može biti klasifikovan u korak transformacije ili korak elaboracije.

7. **Na koji način se vrši projektovanje po kompoziciji?**

Suprotan pristup od dekompozicije. Predstavlja osnovni nivo dizajna za definisanje i predstavljanje entiteta. Primenjuje se strategija „top down“ za identifikovanje operacija koje sistem mora izvršiti. Relevantni elementi se identifikuju analizom početnog opisa problema. Sledeća faza predstavlja razvoj kompletnog modela rešenja kroz razradu opisa entiteta i interakcija između njih. Iako vrsta i oblik entiteta i interakcija mogu značajno da se razlikuju, celokupna strategija modela projektovanja je u suštini ista. Iako su metode kompozicionog projektovanja složene i manje intuitivne, od metoda pristupa "odozgo na dole", može se reći za njih da su stabilnije i da njihova upotreba dovodi do sličnih rešenja problema.

8. **Koji su nedostaci u primeni metoda projektovanja softvera?**

Metod projektovanja softvera ne omogućava automatsko uklanjanje svih problema.

Ne postoje smernice specifične za problem.

Deo procesa projektovanja ne čini model kompletnog procesa projektovanja.

9. **Kako se izvršava transformacija u procesu projektovanja softvera?**

Faza transformacije nakon faze elaboracije obuhvata preraspodelu i modelovanje strukturnog grafikona potprograma.

10. **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: Međunarodne agencije, organi centralne i lokalne samouprave koji su glavni kupci softverskih sistema.

11. **Kako faza elaboracije i transformacije utiče na proces projektovanja softvera?**

Projektant proširuje dijagram toka podataka kroz restrukturiranje ilustrovanog jednog balona u tri balona. Projektant softvera, nakon faze elaboracije, uzima kreirani dijagram i vrši restrukturiranje u strukturni grafikon, pri čemu svaki balon postaje potprogram.

12. **Koje informacije projektant dobija iz faza procesa projektovanja?**

Informacije o modelu projektovanja i informacije o ograničenjima.

**Lekcija 8: Tradicionalni metodi projektovanja softvera**

1. **Šta čitalac dokumentacije softverske arhitekture može da vidi iz "P-Spec" specifikacije procesa?**

Opis procesa kroz:

* naziv procesa
* opis procesa
* ulazno/izlazne podatke iz procesa
* proceduralne zadatke (sekvence, selekcije i iteracije)

2. **Šta je rezultat strukturnog projektovanja softvera?**

Pseudo kod i aspekti koji rešavaju problem (detaljni dizajn).

3. **Šta je cilj strukturne sistemske analize?**

Cilj je modelovanje funkcija koje su usmerene ka problemu i definisanje skupa opisnih formi koje se mogu koristiti u projektovanju arhitekture

4. **Koje informacije projektant softvera beleži u rečnik podataka?**

One koje se odnose na tok podataka u sistemu. Opisuju se podaci prikazani na dijagramu toka podataka i podaci pomenuti u „P-Spec“.

5. **Na koji način se predstavlja strukturni način projektovanja?**

Opisivanjem funkcija potprograma i prikazom hijerarhije poziva.

6. **Kako se vrši proces transformacije u SSA?**

Pet osnovnih koraka procesa transformacije su:

1. Konstruisati dijagram toka podataka (kroz prikaz visokog nivoa)
2. Razraditi slojevitu hijerarhiju na osnovu dijagrama toka podataka
3. Primeniti analizu transakcija i podeliti dijagram u izdvojene jedinice
4. Izvršiti analizu transakcija kreiranjem dijagrama za svaku transakciju u cilju dobijanjastrukturnog grafičkog prikaza za svaku identifikovanu transakciju
5. Spojiti dobijene strukturne grafike u cilju dobijanja inmplementacionog plana, kontrolisanja grešaka i drugih mogućih nedostataka

7. **Koje su aktivnosti koje je potrebno izvršiti u koraku "Analiza transakcije"?**

Dogadjaj, stimulus, aktivnost, odgovor, efekat.

8. **Kojim dijagramom se predstavlja JSP?** Strukturnim dijagramom.

9. **Šta je JSP proces?**

JSP proces obuhvata pet glavnih koraka koji se izvršavaju u nizu (uz određene iteracije između njih).

Zadaci (koraci) su sledeći:

1. Crtanje strukturnog dijagrama koji opisuje svaki ulazni i izlazni podatak
2. Spajanje u programski strukturni dijagram
3. Navođenje operacija koje treba da budu izvršene od programa uz dodeljivanjeoperacija elementima u strukturnom dijagramu
4. Pretvaranje programa u tekst bez specifičnih uslova za određene tačke odluka
5. Dodavanje uslova korišćenja za svaku operaciju iteracije i selekcije

10. **Šta je predstavljeno dijagramom specifikacije sistema?**

Dijagram specifikacije sistema je mrežni dijagram koji vrši identifikovanje interakcije između entiteta modela sistema. Dva osnovna mehanizma interakcije: tok podataka i vektor stanja.

11. **Šta je entitet u dijagramu strukture entiteta?**

Entitet, u ovom kontekstu, predstavlja "aktivan" element koji se identifikuje kroz operacije procesa modelovanja.

12. **U kom slučaju se koristi JSD? Navesti i objasniti primer primene JSD.**

JSD metoda obuhvata analizu i aktivnosti projektovanja koje su usmerene ka izgradnji modela sistema kroz skup interaktivnih opisa procesa. Bavi se modelovanjem procesa, a ne objekata. Sastoji se od 3 faze:

* + - Faza modelovanja
    - Faza mreže
    - Faza implementacije

13. **Šta je analiza transakcije u okviru procesa transformacije?**

Glavna akcija unutar ovog koraka je da komponente "velikog projektovanja sistema" budu razdvojene na mrežu međusobno povezanih podsistema i transakcija između njih.

14. **Koje su operacije u okviru analize transformacije?**

Identifikovanje operacija koje deluju kao centralne transformacije u dijagramu toka podataka. Sledeća je dodavanje još jednog kruga na mesto centralne transformacije, kada nije moguće identifikovati određeni proces kao centralnu transformaciju. Naredna operacija je kreiranje hijerarhije.

# Lekcija 9: Ponovna upotreba softvera

1. **Zašto se pri razvoju softvera koristi i stari softver (ranije razvijene softverske komponente pripremljene za ponovnu upotrebu i dr.)?**

Primena postojećeg softvera u delovima novog softvera smanjuje ukupni trošak razvoja i održavanja softvera, jer se manji broj komponenata mora razvijati. + 9.2

2. **Koje su koristi od ponovne upotrebe ranije razvijenih softverskih jedinica?**

* Povećana pouzdanost
* Smanjen rizik procesa
* Efektivna upotreba specijalista
* Usaglašenost sa standardima
* Ubrzan razvoj

3. **Zašto ponovna upotreba softvera povećava pouzdanost softvera?**

Softver koji je prethodno korišćen i testiran je pouzdaniji od novog softvera. Njegove greške su utvrđene i otklonjene.

4. **Zašto ponovna upotreba softverskih jedinica smanjuje rizik razvoja novog softvera?**

Za razliku od novog softvera, cena postojećeg softvera je poznata. Ovo smanjuje marginu greške pri proceni troškova projekta. Ovo je naročito važno pri razvoju velikih sistema sa velikim komponentama.

5. **Zašto ponovna upotreba softverskih jedinica povećava efektivnost upotrebe softverskih specijalista?**

Umesto da ponovo razvijaju već razvijeno, softver specijalisti se usmeravaju da razviju komponente za ponovnu upotrebu.

6. **Zašto ponovna upotreba softverskih jedinica povećava usaglašenost sa standardima?**

Neki standardi, npr. za GUI, se mogu koristiti za ponovnu upotrebu. Aplikacije koje koriste standardne interfejse su pouzdaniji, jer ih korisnici poznaju i prave manje grešaka u korišćenju.

7. **Zašto ponovna upotreba softverskih jedinica ubrzava razvoj softvera?**

Često je važnije lansirati softver na tržište što pre, nego troškovi njegovog razvoja. Softver sa ponovnom upotrebom komponenti se brže razvija, jer je vreme ispitivanja i razvoja smanjeno.

8. **Koji se problemi mogu javiti pri razvoju softvera uz upotrebu ranije razvijenih softverskih jedinica? Objasni svaki od ovih problema.**

* **Povećani troškovi održavanja** – Ako izvorni kod komponente za ponovnu upotrebu nije na raspolaganju, povećavaju se troškovi održavanja
* **Nedostatak alata za podršku** – Alatai za podršku često ne podrzavaju razvoj softvera sa bibliotekom komponenata.
* **Sindrom “nije razvijeno kod nas” –** Neki sofverski Inženjeri radije razvijaju novi softver nego što upotrebljavaju stari.
* **Kreiranje, održavanje I upotreba biblioteke komponenata –** Održavanje biblioteke komponenata može biti vrlo skupo.
* **Nalaženje, razumeti, prilagoditi za ponovnu upotrebu –** Potrebne komponente treba pronaći, razumeti, i prilagoditi potrebi.

9. **Koje bi faktore imali u vidu pri izboru softvera za ponovnu upotrebu?**

* Očekivano vreme trajanja softvera
* Znanje, iskustvo i veštine razvojnog tima
* Kritičnost softvera i njegovi nefunkcionalni zahtevi
* Domen primene
* Platforma na kojoj će sistem raditi

10. **Šta je radni okvir (framework) za razvoj softvera? Koje su tri kategorije radnog okvira?**

Radni okvir (framework) je opšta (generička) struktura koja se proširuje radi kreiranja specifičnog podsistema ili aplikacije. Radni okviri obezbeđuju opšte funkcije koje se najčešće koriste u svim aplikacijama sličnog tipa. Tri kategorije radnog okvira:

1. Radni okviri infrastrukture sistema
2. Radni okviri za integraciju posredničkih softvera
3. Radni okviri za aplikacije preduzeća

11. **Objasni radni okvir za razvoj veb aplikacija (MVC šablon). Koju funkcionalnost obezbeđuje MVC šablon?**

Radni okviri veb aplikacija (Web application frameworks - WAFs) podržavaju razvoj dinamičkih veb sajtova. Oni najčeše koriste MVC šablon. MVC šablon odvaja stanje aplikacije od korisničkog interfejsa. MVC radni okvir podržava predstavljanje podataka na različite načine i dozvoljava međusobnu interakciju ovih različitih predstavljanja. Kada dođe do promene podatka u jednom predstavljanju, dolazi do promene modela sistema, te i do promene tog podatka u svim drugim predstavljanjima.

12. **Šta su povratni pozivi (callbacks)?**

Povratni pozivi (callbacks) su metodi koji se pozivaju kao reakcija na događaje koje prepoznaje radni okvir.

13. **Šta je linija proizvoda (software product line)? Kako se projektuje jezgro softverskog sistema?**

Linija softverskog proizvoda (software product line) je skup aplikacija sa zajedničkom arhitekturom i deljivim komponentama, pri čemu je svaka aplikacija specijalizovana prema zadovoljenju različitih zahteva. Jezgro sistema je projektovano tako da može da se konfiguriše i prilagodi potrebama različitih korisnika sistema.

14. **Koja je razlika između aplikacionog radnog okvira i linje softverskog proizvoda?**

1. Aplikacioni radni okviri zavise od OO svojstava, kao što je nasleđivanje i polomorfizam radi proširenja radnog okvira. Programski kod radnog okvira se ne menja. Kod linija softverskih proizvoda, njihov kod se menja.
2. Aplikacioni okviri primarno obezbeđuju tehničku podršku razvoju, umestodomenske podrške. Linije softverskog proizvoda obično sadrže i domenske i platformske informacije.
3. Linije softverskih proizvoda obično kontrolišu neku opremu. Aplikacioni radni okviri retko daju podršku vezivanja hardvera.
4. Liniju softevrskog proizvoda čini familija sličnih aplikacija organizacije. Ne koriste opšte jezgro neke aplikacije, već menjaju jednu od aplikacija linije, koja se najmanje razlikuje.

15. **Kako se može razviti specijalizovani tip softverskog proizvoda primenom linije softverskog proizvoda?**

Kreira se jezgro linije softverskog proizvoda proširenjem radnog okvira sa domenskim specifičnostima određene kompanije i ugradnjom njenih mehanizama. U drugoj fazi razvoja, kreiraju se različite verzije sistema za različite kupce. Na ovaj način, mogu da se razviju različiti specijalizovani tipovi softverskog proizvoda.

16. **Kako se vrši konfigurisanje softverskog sistema tokom procesa njegovog razvoja?**

Organizacija koja razvija softver menja zajedničko jezgro linije proizvoda razvijajući ga, biranjem ili prilagođavanjem komponenata radi kreiranja novog sistema za kupca.

17. **Kako se vrši konfigurisanje softverskog sistema u vreme njegovog rapoređivanja 'stavljanja u upotrebu)? Koji su nivoi konfigurisanja sistema?**

Generički sistem je projektovan za konfigurisanje od strane kupca ili konsultanata koji rade sa kupcem, a

to znači u vreme. U sistem se ugrađuje znanje kupčevih specifičnih zahteva i sistemskog operativnog

okruženja.

Postoji nekoliko nivoa konfigurisanja sistema u vreme njegovog raspoređivanja kod kupaca:

1. Izbor komponenata, kada birate module u sistemu koji obezbeđuju zahtevanufunkcionalnost.
2. Definisanje radnog toka (workflow) i pravila, kada definišete radni tok (kakose obrađuje informacija, od faze do faze) i primenjujete pravila validacije koja se primenjuju na informacije koje unose korisnici ili koje sistem generiše.
3. Defisanje parametara, kada definišete vrednosti parametara specifičnogsistema koji odražavaju aplikaciju koju kreirate.

18. **Šta je komercijalni gotov proizvod ( a commercial-off-the-shelf ili COTS)? Koje su prednosti njihove primene?**

Komercijalni gotov proizvod (a commercial-off-the-shelf ili COTS) je softverski sistem koji se može dalje prilagođavati potrebama različitih kupaca bez promene njegovog izvornog koda. Prednosti:

1. Omogućeno je brzo raspoređivanje pouzdanih sistema.
2. Moguće je videti funkcionalnost koja je obezbeđena, što olakšava procenu da linam proizvod odgovara ili ne.
3. Izbegnuti su neki rizici razvoja upotrebom postojećeg softvera.
4. Poslovanje se se može usresrediti na ključnu aktivnost bez odvajanja mnogoresursa na razvoj IT sistema.
5. Kako se izvršne platforme usavršavaju, ugradnja novih tehnologija postajeuprošćena jer to obezbeđuje proizvođač COTS proizvoda, umesto da to radi kupac.

19. **Koji se problemi mogu javiti pri primeni gotovih komercijalnih softverskih proizvoda?**

Problemi koji se mogu javiti pri primeni gotovih komercijalnih softverskih proizvoda:

1. Zahtevi se često moraju prilagođavati funkcionalnostima i načinu rada COTS proizvoda. To može da dovede do naglih promena u postojećim poslovnim procesima.
2. COTS proizvod se može bazirati na pretpostavkama koje je kasnije praktično nemoguće promeniti.
3. Izbor odgovarajućeg COTS sistema za neko preduzeće može biti težak proces,naročito ako postoji mnogo COTS proizvoda koji nisu dobro dokumentovani. Izbor pogrešnog COTS proizvoda može dovesti do toga i da novi sistem ne radi, kako je zahtevano.
4. Može da postoji nedostatak lokalnog znanja za razvoj sistema, te kupac ondapostaje zavistan od proizvođača i spoljnih konsultanata radi saveta u vezi razvoja. Ovi saveti mogu biti pristrasni i podređeni cilju prodaje proizvoda i servisa, umesto da pomognu kupcu da zadovolji svoje zahteve.
5. Proizvođač COTS proizvoda kontroliše podršku sistema i njegovu evoluciju.

Mogu da nestanu iz biznisa, ili prodati, što sve može da dovede do teškoća kod kupaca.

20. **Objasni razliku između COTS-sistemskih rešenja (COTS-solution systems) i COTS-integrisanih sistema (COTS-integrated systems)?**

21. **Opiši tipičnu arhitekturu ERP sistema. Koja su ključna svojstva ove arhitekture?**

ERP sistemi, kao na primer SAP, su veliki integrisani sistemi, koji su projektovani da obezbeđuju podršku poslovanju, kao što je naručivanje i fakturisanje, upravljanje zalihama, planiranje i terminiranje proizvodnje. Generički ERP sistem uključuje veći broj modula koji su kreirani na različite načine.

Svojstva ove arhitekture:

* + - Primena većeg broja modula za podršku različitih poslovnih funkcija.
    - Definisani skup poslovnih procesa, povezanih sa svakim modulom i koji sadržeaktivnosti svakog modula.
    - Zajednička baza podataka koja održava informacije o poslovnim funkcijamapreduzeća.
    - Skup poslovnih pravila koja se primenjuju na sve podatke u bazi podataka sistema. To obezbeđuje njihovu konsistentnost, tj. prihvatljivost od strane svih modula sistema.

22. **Navedi uobičajene aktivnosti konfigurisanja COTS-sistema, kao što su ERP sistemi.**

1. Izbor zahtevanih funkcija od sistema (npr. izbor odgovarajućih modula) 2. Uspostavljanje modela podataka koji definiše kako se podaci organizacije strukturišu u bazi podataka sistema.

1. Definisanje poslovnih pravila koji se odnose na ove podatke.
2. Definisanje očekivanih interakcija sa spoljnim sistemima.
3. Dizajniranje ulaznih formi i izlaznih izveštaja generisanih od strane sistema.
4. Projektovanje novih poslovnih procesa koji su usaglašeni sa procesima definisanihod strane sistema.
5. Postavljanje parametara koji definišu kako se sistem raspoređuje na njegovojplatformi.

23. **Kako bi izabrao COTS sistem? Koje faktore bi uzeo u obzir?**

COTS sistem koji bi izabrao bi morao, pre svega, biti dobro dokumentovan. Trudio bih se da izaberem onaj sistem koji se ne bazira na pretpostavkama koje je kasnije nemoguće izmeniti. Nastojao bih da proverim da li postoji neko lokalno znanje razvoja sistema koje bi me direktno vezivalo za proizvođača i spoljne konsultante. Zatim bih proverio reputaciju i istoriju poslovanja proizvođača COTS sistema, kako bih bio siguran da će COTS sistem imati podršku i mogućost evolucije u budućnosti.

24. **Koji se problemi mogu javiti pri razvoju softvera integracijom više COTS proizvoda?**

1. Nedostatak kontrole nad funkcionalnošću i performansama. Interfejsi mogu da imaju nisku efikasnost.
2. Problemi sa interoperatibilnošću COTS sistema. COTS proizvodi primenju različitestandarde, npr., primenjuju različite modele događaja.
3. Nema kontrole nad evolucijom sistema. Proizvođači COTS proizvoda usmeravajunjihov dalji razvoj, te je i budućnost sistema koji ih integrše u njihovim rukama.

# Lekcija 10: Projektovana softvera primenom komponenata

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.

2. **Objasnite dve kritične komponente softverske komponente: nezavisnost i nuđenje servisa posredstvom interfejsa komponente? Koja je uloga interfejsa u slučaju primene softverskih komponenata.**

1. Komponenta je nezavisni izvršni entitet definisan svojim interfejsima. Ne morate da znate njen izvorni kod, niti da li se koristi kao servis ili je uključena direktno u program, tj. u sistem koji se razvija.
2. Servisi koje nudi komponenta su raspoložive preko interfejsa i sve interakcije se odvijaju samo prekotog interfejsa. Interfejs komponente se izražava pomoću parametrizovanih operacija i ne prikazuje svoje unutrašnje stanje.

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

**Interfejs "obezbeđuje"** ("provides") definiše servise koje obezbeđuje komponenta. Ovaj interfejs je API komponente. On definiše metode koji se mogu pozivati od strane korisnika komponente. U UML dijagramu komponenata, intervejsi koje komponenta obezbeđuje označavaju se punim krugom, povezan linijom sa komponentom.

**Interfejs "potražuje"** ("requires") specificira servise koje treba da joj obezbede druge komponente da bi komponenta ispravno radila. U UML-u, interfejs koji potražuje servise označava se otvorenim polukrugom i linjom spajanja sa interfejsom.

Kritična razlika između komponente gledane kao spoljnog servisa i komponente kao programskog elementa (ugrađen u sistem), je u tome što su servisi potpuno nezavisni entiteti.

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

Model komponente je standard za implementaciju, dokumentaciju i raspoređivanje komponente. Ovi standardi obezbeđuju interoperabilnost komponenata.

Osnovni elementi modela komponenata:

* Interfejsi – definicija interfejsa, kompozicija i specifični interfejsi
* Informacija o korišćenju – konvencija imena, pristup meta podacima i prilagođavanje
* Raspoređivanje i upotreba – pakovanje, dokumentacija i podrška evoluciji

5. **Koja je razlika između servisa platforme i servisa podrške?**

1. Servisi platforme, koji omogućavaju komponentama da komuniciraju i međusobno usaglašeno rade udistribuiranom okruženju (instalirani su u različitim serverima i čvorovima na mreži). Ovo su osnovni servisi koji moraju biti ponuđeni svim sistemima koji koriste komponente.
2. Servisi podrške, koji se nude kao zajednički servisi koji eventualno mogu biti korišćeni od strane nekih različitih komponenata.

6. **Procesi softverskog inženjerstva zasnovanim na komponenatama omogućavaju projektovanje softvera primenom komponenata (PSPK) ili, na engleskom, CBSE procesi (Component-Based Software Engineering). Postoje 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?**

1. Korisnički zahtevi se uopštenije navode, za razliku od uobičajenog detaljnog navođenja korisničkih zahteva. Fleksibilnost u zahtevima povećava broj mogućih komponenti koje se mogu uzeti u obzir. Međutim, morate navesti ceo skup zahteva na početku projekta razvoja, kako bi pronašli što veći skup komponenata koje možete uzeti u obzir za razvoj.
2. Zahtevi se detaljišu i menjaju rano u procesu, u zavisnosti od raspoloživih komponenata.Ako ne možete pronaći nijednu komponentu koja zadovoljava vaše zahteve, onda pokušajte da analizirate zahteve koje mogu da zadovolje raspoložive komponente. Korisnici mogu biti spremni da promene svoje mišljenje, ako ih to vodi nižim troškovima ili bržoj isporuci sistema.
3. Posle postavljanja arhitekture, nastavlja se pretraga komponenata i poboljšanje projektnog rešenja. Može se pokazati da neke planirane komponente ipak nisu odgovarajuće ili da ne rade ispravno sa drugim koponentama koje su odabrane. Ovo dovodi i do daljih promena u zahtevima.
4. Projektovanje je jedan proces spajanja i integrisanja pronađenih komponenti. Ovo zahteva integraciju komponenata sa infrastrukturom modela komponente i razvoj adaptera (prilogođivača) koji koriguju interfejse nekompatibilnih komponenti. ako se jave novi zahtevi, i oni mogu biti razlog za zamenu i ubacivanje novih komponenti.

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

Treba obezbediti komponente koje zadovoljavaju funkcionalne i nefunkcionalne zahteve sistema. Takođe, treba izabrati komponente sa sastavom koji će omogućiti lakše prilagođavanje komponenata kada dođe do promene u zahtevima.

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

Problem sa radom sa izuzecima je posebano težak. Komponente ne bi trebalo da se time bave, jer svaka aplikacija ima svoje specifične zahteve za rad sa izuzecima. Umesto toga, svaka komponenta bi trebalo 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 teškorazumljiv. 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.

9. **Šta se podrazumeva pod upravljanjem komponentama?**

Dokumentovnje njihovih svojstava, skladištenje, i raspoloživost za upotrebu.

10. **Navedite aktivnosti procesa projektovanja softvera primenom komponenti (PSPK). U čemu jespecifičnost ovog procesa u odnosu na projektovanje bez primene komponenti?**

* + 1. Utvrđivanje sistemskih zahteva
    2. Utvrđivanje komponente kandidata
    3. Modifikacija zahteva u skladu sa nađenim komponentama
    4. Projektno rešenje arhitekture
    5. Projektovanje i implementiranje komponente
    6. Sastavljanje komponente i kreiranje sistema

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

1. Sekvencijalno spajanje kreira novu komponentu spajanjem 2 postojeće komponente, njihovim sekevencijalnim pozivanjem. Kako obe komponente imaju interfejs kojim nude svoje servise, neophodno je ubaciti međusoftver koji preuzima informacije i sa jednog i sa drugog interfejsa. Ovaj tip spajanja se može primeniti i kod komponenata koje su element programa i koje su servisi.
2. Hijerarhijsko spajanje se realizuje kada jedna komponenta poziva servise drugog. Pozvana komponenta onda daje tražene servise drugoj komponenti. Ako dođe do neslaganja dva interfejsa, potrebna je konverzija koda. Ovaj tip spajanja se ne koristi kod veb servisa.
3. Aditivno spajanje je slučaj kada se dve ili više komponente sastavljaju radi kreiranja nove komponente koja kombinuje nihovu funkcionalnost. Komponente se nezavisno pozivaju preko spoljnjeg interfejsa složene komponente. A i B nisu međusobno zavisne i ne pozivaju jedna drugu. Ovaj tip spajanje se može primeniti i kod komponenata koje su element programa i koje su servisi.

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

1. Nekompatibilnost parametara: Operacije na svakoj strani interfejsa imaju isto ime,ali njihovi tipovi parametara ili njihov broj je rezličit.
2. Nekompatibilnost operacija: Imena operacija i kod interfejsa koji obezbeđuje("provides") i koji zahteva ("requires") su različite.
3. Nekompletnost operacija: Interfejs komponente koji obezbeđuje servis ("provides") je podskup interfejsa koji zahteva servise ("requires") druge komponente ili suprotno.

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

Problemi nekompatibilnosti se rešavaju razvojem adaptora koji usklađuje interfejse dve komponente. Komponenta adapter konvertuje jedan interfejs u drugi. Oblik adaptora zavisi od tipa spajanja.

14. **Navedite najčešće odluke koje projektanti treba da donesu kada koriste komponente pri razvoju softvera.**

1. Koji sastav komponenata daje najveće efekte na zadovoljenje funkcionalnih inefunkcionalnih zahteva sistema?
2. Koji sastav komponenata čini sistem lakši za prilagođavanje komponenata kadadođe do promena u zahtevima?
3. Koje ća svojstva nastati u sistemu? Ova svojstva se odnose na performanse ipouzdanost. Ovo možete utvrditi tek kada je sistem kompletiran i primenjen.

# Lekcija 11: Projektovane distribuiranih softverskih sistema

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 kompjutera, za razliku od centralizovanih sistema u kome su sve komponente softvera izvršavaju u jednom kompjuteru. Svaki čvor distribuiranih sistema daje svoju specifičnu funkcionalnost sistema, i sam za sebe, predstavlja jedan sistem.

**Prednosti:** Deljenje resursa, Otvorenost, Konkurentnost, Proširljivost, Otpornost u slučaju otkaza

2. **Šta utiče na performanse distribuiranih softverskih sistema?**

Na performanse distribuiranih softverskih sistema utiču:

* Propusni kapacitet računarske mreže
* Njeno trenutno opterećenje
* Brzina svih računara distribuiranih sistema

3. **Projektan softvera treba da bude svestan problema upravljivosti distribuiranim sistemom. Na koja pitanja projektant treba da obrati pažnju pri projektovanju sistema, kako bi oni bolje upravljiv?**

**Objasnite zašto su ta pitanja relevantna.**

Zbog složenosti distribuiranih sistema, postavlja se pitanje o njegovoj kontroli. Mora se računati na nepredvidljivost ponašanja distribuiranog sistema i toga mora da bude svestan projektant. U obzir treba da uzme sledeća pitanja:

1. **Transparetnost:** 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 povećavati u skladu sa povećanjem zahteva za korišćenjem sistema (npr. povćanjem broja njegovih korisnika)?

4.**Bezbednost:** Kako definisati i primeniti bezbedonosna pravila koriščenja 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 (implementirati) da bi bio isporučen sa prihvatljivim kvalitetom servisa svim korisnicima? 6. Upravljanje otkazima: Kako se mogu otkriti otkazi u sistemu, šta učiniti da se minimizira njihov negativni efekat na druge komponente sistema i kako se mogu izvršiti popravke sistema?

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

**Posrednički softver (midlver, middleware)** se koristi za mapiranje logičkih resursa koje program poziva u stvarne fizičke resurse. On se koristi i za upravljanje interakcijama između ovih resursa sistema.Termin „midlver“ se koristi za posrednicki softver koji se nalazi u sredini, izmedju distribuiranih komponenti sistema. Softver je opste namene koji se obicno kupuje kao gotov proizvod, umesto da se posebno razvija.

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

**Otvoreni distribuirani sistemi** su sistemi koji su sagrađeni u skladu s generalno prihvaćenim standardima. To znači da se komponente različitih dobavljača integrišu sistematski da 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.

6. **Šta je proširivost (scalability) distribuiranih sistema? Koje su tri dimenzije proširivosti sistema? Objasnite ih.**

**Proširljivost (scalability)** je svojstvo softvera da obezbeđuje visoki kvalitet servisa i kada se zahtevi za korišćenjem sistema povećavaju.

1. Veličina: Trebalo bi omogućiti dodavanje resursa sistemu da bi on mogao da odgovori na povećan broj svojih korisnika.
2. Distribucija: Trebalo bi omogućiti sposobnost komponenata da se prostorno raštrkaju, a da to ne umanji njegove performanse.
3. Upravljivost: Neohodno je uspešno upravljanje sistemom i kada se povećava njegova veličina, čak i u slučajevima kada su delovi sistema locirani u nezavisnim organizacijama.

7. **Šta je razlika između proširenja sistema (scaling-up) i povećanja sistema (scaling-out)?**

**Proširenje sistema** (scaling up) podrazumeva zamenu komponenata sa moćnijim komponentama, a **povećanje sistema** (scaling out) podrazumeva povećanje broja komponenti, što je isplativije.

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

Distribuirani sistem su ranjiviji na napade i manje bezbedniji od centralizovanih sistema. Napadač, upadom u neki deo sistema "ulazi na mala vrata" u sistem, i onda može da pristupi i drugim delovima sistema.

Postoje sledeće vrste napada na koje neki distribuirani sistem mora da se odbrani:

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.

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

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

Problemi koji prate zahteve za većim kvalitetom servisa:

1. Nije prihvatljiva cena takvog sistema prilikom maksimalnog opterećenja.

2. Parametri QoS mogu međusobno da budu kontradiktorni. Primer: porast pouzdanosti može da smanji protok informacija,

10. **Kako se upravlja otkazima u slučaju distriburanih softverskih sistema?**

Upravljanje otkazima obuhvata primenu tehnika za obezbedjenje otpornosti na otkaze. Nalazi se cvor gde je doslo do otkaza i drugi cvor preuzima ulogu.

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

1. Proceduralna interakcija ukljucuje racunar koji poziva poznati servis koji nudi neki drugi racunar iobicno ceka isporuku tog servisa.
2. Interakcija porukama ukljucuje racunar koji kreira i salje poruku drugom racunaru. Poruke prenose vise informacija u jednoj interakciji nego proceduralnoj interakciji.

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

**RPC** koristi komponenta koja poziva drugu komponentu kao u slucaju poziva lokalne operacije.

**Stub** je tacka povezivanja i ona prevodi parametre procedure u standardnu formu za prenos u udaljenu proceduru i salje zahtev za izvrsenje preko midlvera.

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.

14. **Šta je interakcija sa porukama? Koja je učaga midlvera u tome?**

Interakcija porukama uključuje računar koji definiše informaciju o onome što bi trabalo da sadrži poruka, koja se onda šalje drugom računaru. Preko midlvera poruka se šalje komponenti koja treba da je primi. Komponenta koja prima poruku, čita poruku, vrši odgovarajuću obradu podataka i kreira poruku za komponentu koja je poslala poruku. Ta poruka sadrži rezultat obrade podataka (tj. rezultat servisa) i prosleđuje se preko midlvera komponenti koja je prvobitno poslala poruku.

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

RPC poziv je sinhron, a razmena poruka je asinhrona.

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

* + - Upravljanje komunikacijama
    - Rad sa bazama podataka
    - Upravljanje transakcijama
    - Konverzija podataka
    - Sadrzi kontrolu komunikacija

Midlver koordiniše interakcije između različitih omponenti sistema. Obezbeđuje lokacijsku transparentnost, što znači da komponente ne moraju da znaju fizičku lokaciju drugih komponenti. Mogu da daju podršku različitim programskim jezicima za implementaciju komponenata, da vrše detekciju događaja, obezbeđuju komunikacije i dr.

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

* Nivo prezentacije prikazuje informaciju korisniku i upravlja interakcijom korisnika.
* Nivo upravljanje podacima – koji proverava podatke, generise veb strane.
* Nivo obrade podataka aplikacije – koji implementria logiku aplikacije i obezbedjuje zahtevanu funkcionalnost krajnjim korisnicima.
* Nivo baze podataka – koji skladisti podatke i obezbedjuje upravljanje transakcijama servisa.
* Logički sloj (prezentacija, obrada podataka aplikacije, upravljanje podacima i baza podataka) mora da se mapira na dva računarska sistema - na računar klijenta i na računar servera.

Dolazi do problema proširljivosti sistema i njegovih performansi, pogotovu kada se primenjuje model sa tankim klijentom. Može nastati i problem sa upravljanjem sistema u slučaju primene debelog klijenta.

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

Prednost primene modela tankog klijenta

* Jednostavnost upravljanja klijentima.
* Od značaja kod sistema sa velikim brojem klijenata, problem instalisanja klijentskog softvera kod velikog broja korisnika.
* Sa tankim klijentom, koji koristi samo veb pretraživač, to nije problem jer nema nikakve dodatne instalacije (npr. veb aplikacije).

Nedostaci primene modela tankog klijenta

* Cela je obrada na servereu, te može da dođe do njegovog preopterećenja.
* Veće je opterećenje mreže, jer je veća interakcija klijenta sa serverom.
* Primena modela sa tankim klijento traži kvalitetne mreže, te i veće investicije u mreže
* Problem se može ublažiti, ako se veb pretraživaču dodele neke funkcije, primenom npr., Javascript programa na veb strani kojoj pristupa veb pretraživač

Prednost primene modela debelog klijenta:

Model debelog klijenta efikasno raspodeljuje obradu podataka klijentskim računarima, ali njegovu upravljanje sistemom je složenije.

Nedostatak primene modela debelog klijenta je u tome što je cela obrada na serveru, te može da dođe do njegovog preopterećenja. Takođe, veće je opterećenje mreže, jer je veća interakcija klijenta sa serverom. Zato, primena modela sa tankim klijentom traži kvalitetne mreže, te i veće investicije u mreže.

19. **Koje su prednosti višeslojne arhitekture? A koji su njeni problemi?**

Višeslojni klijent-server sistemi distribuiraju obradu aplikacije na nekoliko servera te su više proširivi nego dvoslojne klijent-server arhitekture.

20. **Koja je korist od primene sistema sa distribuiranim komponentama?**

1. Dozvoljava projektantu sistema da odloži odluku gde i kako da se pojedini servisi obezbede.
2. Komponente koje obezbeđuju servise mogu da se izvršavaju na bilo kom čvoru mreže.
3. 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
4. Sistem je fleksibilan i proširljiv.
5. Nove ili replicirane komponente se mogu dodavati kada dolazi do porasta opterećenja sistema, bez ometanja rada ostalih delova sistema.
6. Moguće je izvršiti dinamičku rekonfiguraciju sistema sa komponentama koje se prebacuju po mreži, ako je to potrebno

20. **Koji su nedostaci sistema sa distribuiranim komponentama?**

◦ Složenije su za projektovanje nego klijent-server sistemi.

* + - 1. Višeslojna klijent-server sistemi su dosta intuitivni. Oni odražavaju mnoge ljudske transakcije kako ljudi traže i pružaju servise drugim ljudima.
      2. Arhitekture sa distribuiranim komponentama su teže za razumevanje ljudi i za vizualizaciju

▪ Standardizovani midlver za sisteme sa distribuiranim komponenatama nije prihvaćen od korisnika.

* + - 1. Umesto toga, pojedini proizvođači (Microsoft, Sun) su razvili različit i nekompatibilni. midlver.
      2. Midlver je složen sistem i oslanjanje na njih povećava ukupnu složenost sistema sa distribuiranim komponentama

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

Servisno-orjentisana arhitektura se danas gotovo najvise koristi, dok se arhitektura sistema sa distriburiranim komponentima redje koristi. Arhitektura sa distribuiranim komponentama je brza i mogu da se izvrsavaju paralelno servisi.

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

Peer-to-peer su decentralizovani sistemi u kojima obrada podataka može da se vrši u bilo kom čvoru mreže.

Svaki čvor je direktno vezan sa elementima koji su mu neophodni za rad.

P2p arhitektura se koristi:

* Kada sistem ima intezivne obrade podataka,a moguce ih je raspodeliti vecem broju racunara.
* U slučajevima kada sistem primarno se bavi razmenom informacija između pojedinačnih računara na mreži, a nema potrebe da se ta informacije centralno skladišti i upravlja

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.

Postoji problem bezbednosti. Komunikacije u p2p sistemima dovode računar u direktnu interakciju sa drugim računarima u mreži, a to znači da ovi sistemi mogu, potencijalno, da pristupaju bilo kom resursu tog računara. Da bi se ovo sprečilo, treba organizovati sistem tako da ovi resursi budu zaštičeni.

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

**Softver kao servis** (software as a service, or **SaaS**) podrazumeva instalaciju softvera u nekom udaljenom serveru koji omogućava pristup klijentima preko Interneta.

Ključni elementi SaaS koncepta:

* + - Softver je raspoređen na serveru (ili na više servera)
    - Softver je u vlasništvu provajdera softvera koji i njim upravlja
    - Korisnici plaćaju korišćenje softvera u skladu sa obimom njegovog korišćenja ili plaćanjem mesečne ili godišnje cene servisa.

Prednost je ta sto se korisnici oslobodjavaju troskova upravljanja softvera, jer tu obavezu preuzima provajder servisa.

25. **Koji su problemi primene SaaS koncepta?**

* + - Troškovima transporta podataka ka udaljenom servisu.
    - Nedostatak kontrole evolucije softvera (jer provajder može da promeni softver kada poželi).
    - Problemi sa zakonima i propisima

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.

27. **Na koje faktore morate da obratite pažnju pri primeni SaaS?**

Na konfigurabilnost, visestruki zakup i prosirljivost.

28. Š**ta je dinamičko konfigurisanje softvera? Šta dozvoljava dinamičko konfigurisanje?**

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

Dinamčko konfigurisanje dozvoljava brendiranje, poslovna pravila i radne tokove, proširenje baze podataka i kontrolu pristupa.

29. **Šta je proširivost SaaS sistema? Koje su preoruke za realizaciju priširenja SaaS sistema?**

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

Preporuke:

1. Svaka komponenta treba da bude razvijena kao jednostavni servis bez stannja koji se može izvršavati na bilokom serveru.

U okviru iste transakcije, korisnik može da radi sa delovima servisa koji su instalisani na različitim računarima.

1. Pri projektovanju koristite asinhrone komunikacije kako aplikacija ne bi čekala rezultat interakcije. Na taj način, izvršenje aplikacije se nastavlja i ya vreme dok korisnik priprema svoju interakciju
2. Upravljajte resursima, kao što su veze mreže i baza podataka, kao pul, tako da nijedan pojedinačni server ne dođe do nedostatka svojih resursa.
3. Projektujte svoju bazu tako da omogućava zaključavanje na što detaljnije nivou (fine-grain locking).

Ne zaključavajte ceo slog baze kada koristite samo jedan njegov deo.

# 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 predstavljaju način razvoja distribuiranih sistema u kome su sistemske komponente samostalni servisi, koje se izvršavaju na geografski distribuiranim računarima.

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 servisnoorijentisanim sistemima? Koja je njihova uloga?**

Servisno-orijentisani sistemi imaju labavo povezane (loosely coupled) arhitekture u kojime se veze među servisima menjaju za vreme izvršenja sistema. Akteri su klijenti i serveri. Klijenti traže upotrebu servisa, a serveri obezbeđuju tražene servise.

12.5. **Šta je XML? Za šta se on koristi?**

Predstavlja univerzalni standard za predstavljanje podataka. Njime je moguće predstavljati bilo kakve podatke. Omogućava stukturiranje, prenos i čuvanje podataka.

12.6**. Šta je SOAP? Čemu služi?**

**SOAP:** 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?**

**WSDL**: 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?**

**WS-BPEL**: 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?**

**UDDI** (Universal Description Discovery and Integration) standard pretraživanja definiše komponente specifikacije servisa koje pomažu potencijalnim korisnicima servisa da ga otkriju na Internetu.

12.10. **Navedite kljućne standarde sa veb SOA?**

1. WS-Reliability Messaging – standard za razmenu informacija koji obezbeđuje isproruku poruka, i to jednom, i samo jednom.
2. WS-Security – skup standarda koji podržavaju zaštitu veb servisa, koji uključuju standard koji specificira definiciju pravila za bezbednost i standarde koji pokrivaju upotrebu digitalnih potpisa.
3. WS-Addressing – koji definiše kako se predstavlja informacija u SOAP poruci.
4. WS-Transactions – koji definiše kako se koordinišu transakcije distribuiranih servisa.

12.11**. Koji su nedostaci veb servisa?**

* + - Dostupnost – Servis ne mora nužno biti dostupan sve vreme.
    - Specifični zahtevi – Servisi se kreiraju sa ciljem opsluženja velikog broja korisnika. Ipak, ukoliko imamo korisnike sistema koji treba da imaju jedinstveno svojstvo za potrebe obavljanja svojih zadataka, tu nastaje problem. Ako se sistem i poslovanje ne mogu uklopiti u predefinisani model, veb servis nije rešenje. Trebale bi se razmotriti druge opcije.
    - Bezbednost – Servisi su javni, koristi se http protokol. Svako može da im pristupi i da ih koristi. • Ne garantuju izvršenje. Http protokol nije najpouzdaniji protokol, pa ne može ni da garantuje siguran odgovor na poziv.
  1. **Do čega je dovale 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. Ovo će imati stalni i duboki uticaj na sistemske proizvode i poslovne procese. Primena veb servisa omogućava da servisno –orijentisana preduzeća značajno poboljšaju agilnost kompanije, brzinu izbacivanja svojih novih proizvoda i servisa na tržište (time-tomarket) i smanjivanje IT troškova i poboljšavanje efikasnosti rada.

* 1. **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), lokaciju servisa, i detalje njegovog interfejsa. Ovi detalji se nalaze u opisu njegovog servisa napisanog u XML jeziku koji se naziva WSDL (Web Service Description Language).

* 1. **Šta je WSDL? Koja tri aspekta podržava WSDL?**

**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.

WDSL specifikacija definiše tri aspekta veb servisa: šta servis radi, kako komunicira i i gde se nalazi.

* 1. **Koji su elementi WSDL modela?**

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

* 1. **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?**

Resurs je element podatka kao što je katalog ili medicinski karton, ili neki dokument i sl. Element može da se predstavlja na različite načine, tj. da mogu biti u različitim formatima. Resursi imaju jedinstven identifikator, kao što je URL. Osnovne operacije koje se mogu izvršiti nad jednim resursom su: Create, Read, Update i Delete. Četiri akcije koje obezbeđuju HTTP i HTTPS: POST, GET, PUT i DELETE.

* 1. **Šta je RESTful servis? Koji princip projektovanja on podržava?**

RESTful pristup podržava efikasanu interakciju sa servisom, ali ne podržava svojstva na organizacionom nivou, kao što je WS-Reliability i WS-Transactions.

* 1. **Koja je primena RESTful servisa kod servisa «oblaka», tj. klaud servisa?**

Servisi se nude sa računarske infrastrukture instalirane kod provajdera, kao što su Google i Amazon.

* 1. **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 iupravljanje kvalitetom servisa i pouzdanošću servisa. U slučaju primene SOAP servis postoje standardi za dodatnu infrastrukturu kao što su WS-Reliability i WSTransaction.

* 1. **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 upotrebljiv, robustan i pouzdan. Nefunkcionalni zahtevi servisa treba da definišu bezbednost, performanse i raspoloživost.

* 1. **Koje su tri faze inženjering servisa?** 
     1. Utvrđivanje mogućih servisa
     2. Projektovanje servisa
     3. Implementacija i raspoređivanje servisa

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

1. **Uslužni servisi**: 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 uopštenijiposlovni 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?**

1. Servisi za zadatke: Povezani su sa nekom aktivnošću.
2. Servisi 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 (MakeCatalog, Lookup, Search, Compare,

CheckDelivery, MakeVirtualOrder)

1. Projektovanje poruka
2. Izrada WSDL dokumenta
   1. **Šta je katalog intefejsa servisa? Šta on sadrži?**

Katalog interfejsa sadrži naziv operacije, ulaze, izlaze i izuzetke.

* 1. **Kako se vrši implementacija i testiranje interfejsa?**

Posle utvrđivanje mogućih servisa i projektovanja njihovih interfejsa, u poslednjoj fazi procesa servisnog inženjerstva se radi implementacija servisa., tj. programiranje servisa upotrebom standardnih programskih jezika.

Pre raspoređivanja servisa, neophodno je razvijeni servis testirati, a to znači ispitivati ulaze u servis, kreiranje ulaznih poruka koje su u saglsnosti sa zahtevanim kombinacijama ulaze i videti da li se dobijaju očekivani izlazi. Ukoliko ima grešaka u ulazima, neophodno je da se generišu odgovarajući izuzeci.

* 1. **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. Informacije o servisu koje treba objaviti:

* + - 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 obezbeđujedobijanje informacije o novim verzijam servisa.
  1. **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.

* 1. **Navedi faze konstruisanja sistema sastavljanjem više servisa**

◦ Formulisanje šeme radnog toka

◦ Otkrivanje servisa

◦ Izbor mogućih servisa

◦ Prerada radnog toka

◦ Kreiranje programa radnog toka

◦ Testiranje postavljenog servisa ili aplikacije

* 1. **Kako se vrši testiranje sastavlenog 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 servisnoorijentisanih sistema ne može da koristi poznate metode testiranje koje se zasnivaju na izvornom kodu.

Problemi koji se mogu javiti u vezi testiranja:

1. Spoljni servisi su pod kontrolom provajder servisa, a ne korisnika servisa.
2. Dugovečne verzije SOA za servise se dinamički povezane sa servisno-orijentisanim aplikacijama
3. Nefunkcionalno ponašanje servisa ne zavisi samo od aplikacije koja se testira.
4. Model placanja servisa može da doprinese visokoj ceni testiranja.
5. Postoji problem testiranja softverskog servisa sa akcijama kompenzacije. Oni mogu da zaviste od otkaza drugih servisa.

**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 ( real-time software system ) je sistem čiji ispravan rad zavisi i od rezultata koji proizvodi i od vremena u kome se ti rezultati proizvedu. 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 sistema koji prouzrukuje da sistem reaguje na neki način – a to je signal ili poruka koje sistem šalje u svoje okruženje. 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.

Postoje dve kategorije podsticaja:

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.

* 1. **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.

* 1. **Navedite aktivnosti procesa projektovanja sistema u realnom vremenu.**

1. Izbor platforme

2. Identifikacija podsticaja/odgovora

3. Analiza vremena

4. Projektovanje procesa

5. Projektovanje algoritma

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

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. Treba obezbediti da proces koji proizvodi informaciju ne pokušava da je stavi u već pun bafer. Analogno, treba sprečiti da proces koji koristi informaciju, ne učitava podatak sa praznog bafera. Da bi se ovo obezbedilo, treba da koristite kružni bafer sa Get i Put operacijama pristupu baferu. Put operaciju koristi proces koj proizvodi informaciju, a Get operaciju koristi proces koji koji koristi informaciju, tj. koji je preuzima iz bafera. Sinhroninizacioni mehanizam koristi semafore i kritične regione da bi se obezbedilo da operacije Get i Put sinhronizovano rade svoj posao.

13.6 **Šta je cilj statičke anaize sistema?**

Cilj statičke analize sistema je provera da li sistem zadovoljava postavljene zahteve.

* 1. **Koji su problemi programiranja sistema u realnom vremenu?Ne može da se koristi OO pristup kod „teških“ sistema u realnom vremenu.**

<https://www.eventhelix.com/RealtimeMantra/IssuesInRealtimeSystemDesign.htm>

* 1. **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?**

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.

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).

* 1. **Zašto je potrebno korišćenje posebnih šablona projektovanja sistema u realnom vremenu? Da li se šabloni mogu kombinovati prilikom projektovanja softverskog sistema?**

Oni pomažu pri donošenju odluka pri detaljnijem projektovanju sistema Ovi šabloni podržavaju upravljaju izvršenjem procesa, komunikacije, alokaciju resursa, kao i bezbednost i pouzdanost sistema

Ovi se šabloni mogu kombinovati pri projektovanju konkretnog sistema u realnom vremenu.

* 1. **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.

* 1. **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.

* 1. **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. Transformacijapodatakasevršisekvencijalnimizvršenjemvišeprocesnihkorakakojisemogu i konkurentno (paralelno) izvršavati. Ovo omogućava vrlo brzu obradu podataka, jer posebno jezgro procesora ili poseban procesor izvršava svaku transformaciju.

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

Kako ispravan rad sistema u realnom vremenu zavisi od vremenskih intervala u kome se dobijuje rezultat obrade podataka i reakcije senzora i drugih uređaja na ulaze, tj. od brzine ragovanja sistema na ulaze, analiza vremena predstavlja važnu aktivnost u procesima razvoja ugrađenih sistema u realnom vremenu. 3 ključna faktora su: rokovi, frekvencija i vreme izvršenja.

* 1. **Zašto se primenjuju posebni operativni sistemi u realnom vremenu pri primeni softvera u realnom vremenu? Yašto se ne koriste uobičajeni operativni sistemi?**
  2. **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.

13.15. **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 (interruptlevel), 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.

* 1. **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).

* 1. **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 jednostavani brzi servis RTOS za rad sa prekidima.

* 1. **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.

* 1. **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 prioritetom, te se dodeljuje procesoru.

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

Svojstvo konkuretnosti i upravljanje resursima se realizuje pozivom primitiva koje obezbeđuje operativni sistema razvijen za srad sistema u realnom vremenu.

# Lekcija 14: Projektovanje pouzdanog proizvoda

14.1. **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.

14.2. **Navedite greške koje ne dovode do otkaza rada (pada) sistema**.

1. Ne izvršava se ceo program. Deo programa koji sadrži grešku ne izvršava se zbog uslova korišćenja sistema, koji nedozvoljava korišćenje dela softvera koji sadrži grešku.

Zbog toga, iako softver ima grešku, ona ne proizvodi nikakav efekat.

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

Razlog zašto ne dolazi do pada sistema je taj što su ljudi svesni greške.

14.3. **Navedite pristupe koje vode poboljšanju pouzdanosti softvera.**

* + 1. Izbegavanje greške
    2. Detekcija grešaka i njihovo otklanjanje
    3. Tolerancija grešaka

14.4**. 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.

I pouzdanost i dostupnost se mogu izraziti preko verovatnoće.

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

Događaj koji se pojavljuje u nekom vremenskom trenutku kada sistem ne isporučuje servis koji korisnici očekuju. Otkaz sistema (failure) je spoljni događaj koji utiče na korisnike sistema. S sa tehničke tačke gledano, otkaz sistema je ponašanje koje ne zadovoljava specifikaciju sistema.

Dva problema koja često dovode do kvarova u sistemu:

* + - 1. Specifikacije softvera su često nekompletne iili netačne
      2. Niko sem inženjera razvoja softvera ne čita specifikaciju softvera.

14.6. **Šta je pouzdanost sistema? Koja je metrika za pouzdanost?**

Pouzdanost softvera (reliability) znači da servis isporučuje tačne rezultate. Može se izraziti preko verovatnoće. Ako 2 ulaza, na svakih 1000, dovodi do pada softvera, onda je pouzdanost, defisana kao učestanost javljanja grešaka, 0,002.

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

Ako se koristi POFOD metrika, definiše se verovatnoća otkaza sistema kada se zahteva neki njegov servis. Ukoliko je, na primer, POFOD= 0,001, to znači da postoji 1/1000 šansi da se javi otkaz pri postavljanju tražnje za servisom. POFOD se upotrebljava u situacijama kada pad pri zahtevu korišćenja sistema vodi do ozbiljnih kvarova sistema.

14.8. **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). ROCOF = 1/1000. ROCOF se koristi u situacijama kada kvar pri pozivu dovodi do ozbiljnog pada sistema.

14.9. **Šta je srednje vreme otkaza – MTTF? Dajte primer. Kada se koristi MTTF?** MTTF je prosečan broj vremenskih jedinica između otkaza sistema. ROCOF=2 znači javljanje dva otkaza po satu, a to znači da je srednje vreme otkaza 30 minuta. Ako je važno apsolutno vreme između otkaza, onda specifirajte pouzdanost sa srednjim vremenom između dva otkaza (MTTF)

14.10. **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.

14.11. **Šta definišu nefunkcionalni zahtevi pouzdanosti?**

**Nefunkcionalni zahtevi** pouzdanosti su specifikacije o zahtevanoj pouzdanosti i dostupnosti

sistema upotrebom jedne od metrika pouzdanostu (POFOD, ROCOF ili AVAIL).

14.12. **Koja je korist od kvantifikovane pouzdanosti?**

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

14.13. **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.

14.14. **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 izbegavanje grešaka (fault-avoidance), detekcija grešaka (faultdetection) i tolerancija grešaka (fault-tolerance).

Specifikacija funkcionalne pouzdanosti obuhvata analizu nefunkcionalnih zahteva (ako treba da se specificiraju), ocenu rizika na pouzdanost i specifikaciju funkcionalnosti sistema koja se odnosi na te rizike.

14.15. **Navedite 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.
   1. **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. Mehanizmi za tolerisanje grešaka otkrivaju i ispravljaju stanje greške tako da pojava greške ne dovodi do pada sistema. Posebno su bitni kod sistema koji su kritični sa stanovišta sigurnosti i bezbednosti.

* 1. **Šta sadrži arhitektura sistema koji je otporan na greške?**

Mora da se projektuje arhitektura sistema tako da uključi redudantne (ponovljive) i različite hardverske i softverske komponente. Najjednostavnija primena pouzdane arhitekture je arhitektura sa dupliranim serverima, kada dva ili više servera obavljaju isti zadatak.

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

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

Zaštitni sistemi imaju svoj poseban sistem monitoringa okruženja. Ako se otkrije problem, on izdaje komande pokretačima da isključe sistem ili da aktiviraju druge zaštitne mehanizme.

* 1. **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. Primenjuje se različit hardver na svakom kanalu. Koristi se raličit softver u svakom kanalu.

* 1. **Šta je trostruka modularna redudantnost-TMR?**

Kod TMP sistema hardverske komponente se repliciraju tri (ili više) puta. Cilj je obezbeđivanje tolerantnosti na greške u hardveru.

* 1. **Šta je programiranje N verzija softvera? Kada ga treba primeniti?**

Upotrebom iste specifikacije, različiti timovi razvijaju isti softverski sistem. Ovako razvijene verzije istog softvera se izvršavaju na različitim kompjuterima. Jeftiniji je od arhitekture sa samoproveravanjem. Ipak, ne spada u kategoriju jeftinijih pristupa, jer je potrebno finansirati rad više timova programera, da bi se dobile različite verzije istog softvera. Ovaj pristup se koristi samo u slučajevima kada nije praktično obezbeđivanje zaštitnog sistema koji štiti sistem od grešaka opasnih po bezbednost.

* 1. **Koja su moguća dodatna pravila kompanija?**
     + Uključuje zahteve za primenom različitih metoda projektovanja.
     + Insistiranje da se pri programiranju koriste različiti programski jezici.
     + Zahtevati korišćenje različitih razvojnih alata i razvojnih okruženja za razvoj sistema.
     + Zahtevati primenu različitih algoritama u nekim delovima implementacije.

14.23. **Kako dolazi do pogrešne interpretacije specifikacije?**

1. Članovi različitih timova imaju istu kulturnu pozadinu ili imaju isto obrazovanje koristeći isti pristup i iste udžbenike. Ovo znači da imaju iste teškoće u razumevanju istih delova specifikacije i imaju iste probleme u komunikaciji sa stručnjacima u domenu rada softvera. Moguće je da i kada rade nezavisno, čine iste greške i da projektuju iste algoritme za rešavanje problema.
2. Ako su zahtevi netačni ili se oslanjaju na nerazumevanja okruženja sistema, onda će tegreške da se odražavaju u svakoj implementaciji sistema.
3. Detaljna specifikacija kritičkih sistema koja se dobija na osnovu sistemskihzahteva, trebalo bi da obezbedi jedinstvenu definiciju sistemskog ponašanja.

Međutim, ako je specifikacija dvosmislena, onda i različiti timovi ih mogu da različito tumače na isti način.

* 1. **Koja su moguća 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.

Neka istraživanja ukazuju da su sistemi sa tri kanala, 5 do 9 puta pouzdaniji od sistema sa jednim kanalom. Ipak, za neke sisteme visoka cena nije prihvatljiva, pa ta opcija nije rešenje.

* 1. **Koje su preporuke za programiranje za pouzdanost?**
     + Ograničite vidljivost informacija u programu.
     + Proverite ispravnost svih ulaza (opsega, veličine...)
     + Obezbedite proceduru za obradu izuzetaka
     + Minimizirajte upotrebu programskih rešenja koja su sklona greškama
     + Obezbedite mogućnost restartovanja sistema
     + Proverite granice nizova
     + Uključite ograničenje vremena čekanja pri pozivanju spoljnih komponenti
     + Imenujte sve konstante koje predsatvljaju stvarne vrednosti

14.26. **Koji su potrebni podaci za utvrđivanje pouzdanosti softvera?**

1. Broj otkaza sistema posle određenog broja zahteva za korišćenje servisa
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 kagubljenju njegovog servisa, u cilju merenja dostupnosti.
   1. **Koje su faze statističkog testiranja radi merenja pouzdanosti?**
   2. **Koje su teškoće u statitskičkom testiranju?**
4. Neizvesnost profila rada: Profili rada koji se oslanjaju na iskustvo sa drugimsistemima ne moraju da tačno odražavaju upotrebu vašeg sistema.
5. Visoki troškovi generisanja test podataka: Može biti vrlo skupo generisati veliki broj podataka za testiranje sistema u profilu rada, u slučaju da proces nije automatizovan. 3. Statistička neizvesnost specifikacije sa visokom pouzdanošću: Morate da generišete statistički značajan broj otkaza da bi dozvolili tačno merenje pouzdanosti. Kada je softver već pouzdan, samo malo otkaza se javlja i teško je izazvati nove otkaze.

4. Prepoznavanje kvarova: Nije uvek očigledno da je došlo do kvara sistema. Ako imate specifikaciju, možete utvrditi odstupanje ponašanja sistema u odnosu na specifikaciju. Međutim, ako je ona pisana u prirodnom jeziku, mogu da nastanu nejasnoće koje testere dovode u zabunu da li je sistem u kvaru ili nije.

14.29. **Šta je profil rada?**

**Profil rada** softverskog sistema odražava način njegove primene u praksi. On sadrži specifikaciju vrste ulaza i verovatnoću njihovog korišćenja.

# Lekcija 15: Analiza i ocena kvaliteta projektnog rešenja softvera

15.1. **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.

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?**

Kada se razvijaju veliki i složeni sistema čiji razvoj može da potraje i nekoliko godina, primenjuje se razvoj vođen planom. Tada se primenjuje formalizovano upravljanje kvalitetom (quality management – QM):

1. Na organizacionom nivou, upravljanje kvalitetom se bavi postavljanjem okvira organizacionih procesa i standarda koji treba da dovedu do softvera visokog kvalteta. QM tim bi trebalo da preuzme odgovornost za definisanje procesa razvoja softera koje treba koristiti i za standarde koji bi trebalo da se primeni na softveru i odgovarajućoj dokumentaciji, uključujući sistemske zahteve, projektovanje i programiranje.
2. Na projektnom nivou, upravljanje kvalitetom uključuje primenu specifičnih procesa kvaliteta, vrši proveru primene ovih procesa i obezbeđuje da rezultati projekta zadovolje definisane standarde projekta. Upravljanje kvalitetom projekata definiše ciljeve kvaliteta projekta i definiše koji procesi i standardi treba da se primene.
   1. **Koja je razlika upravljanja kvalitetom i kontrole kvaliteta softvera?**
   2. **Koja je uloga QM tima (tima koji upravlja kvalitetom projekta).**

* QM tim bi trebalo da preuzme odgovornost za definisanje procesa razvoja softera
* QM tim proverava rezultate projekta da bi obezbedilо da oni zadovoljavaju standarde i ciljeve organizacije.
* QM takođe proverava dokumentaciju procesa
* QM tim koristi dokumentaciju da bi proverio da nisu zaboravljeni važni zadaci
* QM tim u velikim kompanijama je odgovoran upravljanje procesom završnog testiranja.
* QM tim bi trebalo da ima odgovornost na nivou organizacije za upravljanje kvalitetom.
* QM tim izveštava menadžment koji je iznad nivoa menadžera projekta.
  1. **Šta je plan kvaliteta? Koja je stuktura plana kvaliteta?**

Plan kvaliteta opisuje željene atribute kvaliteta softvera i opisuje kako se ovi atributi mogu ocenjivati. Struktura plana kvaliteta treba da sadrži:

1. Uvod o proizvodu: Opis proizvoda, ciljno tržište i očekivani kvaliteta proizvoda.
2. Planovi proizvoda: Rokovi izdavanja verzija softvera i odgovornosti za proizvod, zajedno sa planovima za distribuciju i servisiranje proizvoda.
3. Opisi procesa: Procesi razvoja i servisa, kao i standardi koji bi trebalo da se koriste u razvoju proizvoda i u upravljanju razvojem.
4. Ciljevi kvaliteta: Ciljevi kvaliteta i planovi proizvoda, uključujući utvrđivanje I proveru kritičkih atributa kvaliteta proizvoda.
5. Rizici i upravljanje rizicima: Ključni rizici koji mogu da utiču na kvalitet proizvoda I akcije koje treba preuzeti da bi se odgovorilo na ove rizike.

Preporuka je da plan kvaliteta ne bude dug, jer ga u tom slučaju ljudi ne bi čitali.

* 1. **Zašto je subjektivnost prisutna u oceni kvaliteta softvera?**

Njihova ocena zavisi od iskustva programera koji ocenjuju. Zbog ove subjektivnosti, njihovo objektivno merenje nije moguće. Da bi merili atribute kvaliteta, morate meriti neke interne atribute softvera

* 1. **Na koja pitanja 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 zadovoljavasve 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?
   1. **Navedite atribute kvaliteta softverskog proizvoda.**

Atributi kvaliteta mogu biti spoljni i unutrašnji.

* 1. **Opiši proces ocenjivanja kvaliteta softvera.**

Proces ocenjivanja kvaliteta softvera treba da bude definisan u planu kvaliteta. Predstavlja dogovoren i usvojen način procene da li je ostvaren kvalitet proizvoda sa aspekta definisanih atributa kvaliteta.

* 1. **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?**
  2. **Koje su teškoće u određivanju atributa kvaliteta?**
  3. **Šta je to «kultura kvaliteta»? Kada možemo da kažemo da se u nekoj organizaciji neguje «kultura kvaliteta»?**

Definisanje odgovarajućih procesa razvoja softvera nije dovoljno za dobijanje kvalitetnog softvera. Oraganizacije treba da stvori i održava "kulturu kvaliteta" u kojoj svi učesnici u razvoju softvera treba da doprinesu postizanju visokog nivoa kvaliteta softvera.

* 1. **Šta definišu standardi proizvoda, a šta definišu standardi procesa?**

**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.

**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 napisati za vreme ovih procesa.

* 1. **Šta standardi softverskog inženjerstva definišu?**

Oni se baziraju na znanju i najboljoj praksi kompanije. Oni su najčešće razultat dugo sticanog iskustva. Njihovom primenom se koristi prethodno stećeno znanje i iskustvo i izbegava ponavljanje nekih grešaka.

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

Inženjeri ne vole da primenjuju standarde, jer moraju da se prilagođavaju određenom radnom okviru/načinu rada. Međutim, primena istog principa rada može da bude korisna prilikom rešavanja nekog konkretnog problema, jer se ostali članovi tima mogu bez poteškoća uključiti i pomoći u rešavanju problema. Ustaljen način rada takođe omogućava lakše prilagođavanje novih inženjera.

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

Menadžer projekta može da modifikuje standarde procesa u skladu sa okolnostima njegovog projekta, pri čemu treba da pazi da promene ne naruše kvalitet proizvoda. Menadžer projekta, zajedno sa menadžerom kvaliteta, može da zaobiđe probleme vezane za primenu neodgovarajućih standarda pažljivim planiranjem kvaliteta u ranoj fazi projekta. Najpre se određuju organizacioni standardi koji će se primenjivati bez ikakvih modifikacija, zatim oni standardi koji zahtevaju određene modifikacije, kao i oni standardi koji neće biti primenjeni.

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

**ISO 9001** je radni okvir za razvoj softverskih standarda. Postavlja opšte principe kvaliteta i procedure koje treba definisati. Ključni procesi ISO 9001 standarda:

**Procesi isporuke proizvoda**: Poslovna akvizicija, Projektovanje i razvoj, Proizvodnja i isporuka, Servis i podrška, Test

**Procesi podrške:** Upravljanje poslovanjem, Upravljanje dobavljačima, upravljanje konfiguracijom, Upravljanje zalihama.

* 1. **Šta je uputstvo o kvalitetu?**

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

* 1. **Navedite standarde koji su od značaja za kvalitet softvera.**

**Standardi proizvoda**: Odnose se na softverski proizvod i obuhvataju standarde dokumentovanja rada. **Standardi procesa**: Oni definišu procese koje treba primeniti tokom razvoja softvera.

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

**Recenzija kvaliteta** 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. Recenzija nije samo provera poštovanja standarda. Ona pomaže da se otkriju problemi i propusti u softveru ili u projektnoj dokumentaciji.

* 1. **Navedite aktivnosti procesa recenzije softvera.**

**Aktivnosti pre recenzije** – pripremne aktivnosti recenzije. Definiše se tim recenzenata, vreme i mesto gde će se aktivnost recenzije obaviti i prosleđuju se dokumenti za recenziju. Članovi tima analiziraju softver, softversku dokumentaciju i standarde. Sastanak tima recenzenata - Za vreme sastanka, autori dokumenata daju pregled dokumenata timu recenzenata. Sastanak je relativno kratak, npr., do dva sata. Jedan član tima vodi sastanak, a drugi zapisuje komentare. Zapisnik sa sastanka se potpisuje sa komentarima i sa dogovorenim akcijama.

**Aktivnosti posle recenzij**e – Posle sastanka recenzenata se izvode zaključci. Pristupa se otklanjanju grešaka, vrši se refaktorizacija softvera da bi bio u skladu sa standardima ili se modifikuje dokumentacija.

* 1. **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.

* 1. **Klasifikujte greške koje se javljaju pri razvoju softvera.**
  2. **Kako se vrši kontrolu kvaliteta softvera i donosi odluke o kvalitetu softvera.**
  3. **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?**

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. Međutim, dva programera koja kreiraju kod ne mogu da budu tako objektivni kao što je spoljni tim kontrolora. Ovo su potencijalni problemi:

1. Uzajamno nerazumevanje: Oba člana para mogu da čine iste greške u razumevanju sistemskih zahteva. Diskusije mogu i da pojačaju ove greške.

2. Reputacija para: Par može da bude nevoljan da gleda na greške jer ne želi da uspori rad na realizaciji projekta.

3. Radni odnosi: Sposobnost para da otkriva greške je ugrožena u bliskoj saradnji članova para koji imaju problem da kritikuju rad partnera.

* 1. **Kako se vrši merenje softvera? Šta su metrike softvera? Koje su dve vrste metrike softvera? Merenje softvera** kvantifikuje neke atribute softverskog sistema, kao što je njegova složenost ili pouzdanost. Upoređivanje izmerenih vrednosti međusobno, a i njihovim upoređenjem sa standardima organizacije, možete zaključiti o kvalitetu softvera ili oceniti efektivnost softverskih procesa, alata i metoda.

**Metrike softvera** su karakteristike softverskog sistema, dokumentacije sistema, ili procesa razvoja koje se mogu objektivno izmeriti. Metrike kontrole I Metrike predvidjanja

* 1. **Navedite metrike kontrole procesa. Koja je razlika metrike predviđanja i metrike kontrole?**
  2. **Šta su metrike softverskog proizvoda? Navedite primere.**

**Metrike proizvoda** su metrike predviđanja upotrebljene za kvalifikovanje unutrašnjih atributa softverskog sistema. Primeri metrika proizvoda su: veličina sistema merena u broju linija koda ili brojem metoda po klasama.

Metirike možemo klasifikovati na dinamičke metrike i statične metrike.

* 1. **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.

* 1. **Navedite spoljne i unutrašnje atribute kvaliteta. Koji su uslovi predviđanja spoljnjih atribita?**
  2. **Opišite kako bi izvršili analizu kvaliteta komponente.**

Anomalna merenja, koja značajno odudaraju od norme, obično ukazuju na probleme povezane sa kvalitetom ovih komponenata.

* 1. **Koja je razlika dinamičkih i statičkih metrika?**

**Dinamičke metrike** pomažu u ocenjivanju efikasnosti i pouzdanosti softverskog sistema.

**Statičke metrike** pomažu u ocenjivanju složenosti, razumljivosti i održivosti sistema ili njegovih komponeneta.

Statičke metrike imaju samo indirektne veze sa atributima kvaliteta.

* Dinamičke metrike, koje se određuju merenjima u toku izvršenja programa. One se određuju testovima.
* Statičke metrike, koje se određuju merenjem reprezentacija proizvoda, kao što je projektno rešenje, programski kod ili dokumentacija.