1. Paradigme arhitekture softvera

Cilj je da ustanove promene u tretiranju aplikativnog softvera u poslednjih 50 god. Ove promene se mogu uspešno predstaviti evolucionim modelom sa malim brojem međusobno povezanih paradigmi. Ova analiza pomaže otkrivanju trenda na osnovu kojega se može govoriti sa više izvesnosti o budućoj evoluciji arhitekture softvera. Ova analiza takođe treba da pomogne da se na što bolji način reši problem adekvatnog korišćenja računara u automatizaciji poslovanja.Osnovni razlozi promene arhitekture softvera su oni koji utiču na troškove razvoja i eksploatacije sistema. Najveća posledica navedenog razvoja je promena koncepta korišćena softvera. Striktna primena objektno orijentisanih pristupa briše podelu na kod i podatke jer su sad aplikacije učaurene sa podatcima.

2. Monolitna programska podrška

Ovo je prva faza evolucije arhitekture softvera ('46 - '65). Dominantne k-ke IS zasnovanih na primeni računara bile su pre svega određene tadašnjim hardverom. Hardver je bio izuzetno skup, personal je u odnosu na cenu hardvera bio jako jeftin, pa je interfejs između hardvera i korisnika bio prilagođen hardveru. Korisnik mora da misli kao mašina. Izvršni kod je srastao sa pogonskim rutinama U/I uređaja i monitorskim programima. Izvršni kod je pisan na asemblerskom ili mašinskom jeziku. Korisnik je bila osoba školovana za rad sa konkretnim sistemom. U prvoj fazi evolucije korisnički interfejs se dizajnirao u stilu spartanskog komfora. Cena interfejsa bila je zanemarljiva u odnosu na cenu hardvera, znači interfejs je bio obična električna mašina u funkciji konzole.

16. Kategorija objekata

Postoje glavne 3 kategorije objekata: Poslovni, Tehnološki, Aplikacioni. Svaka kategorija ima specifičnu namenu i može se primeniti smo u svom domenu pri projektovanju IS. Poslovni objekti predstavljaju autorne resurse iz poslovnog domena i u sebi obuhvataju naziv, definiciju, atribute, ponašanje, veze sa okolinom i ograničenja. Oni omogućavaju da se pravi podaci spregnu sa pravim procedurama na pravom mestu što se naziva Semantička normalizacija. Tehnološki objekti predstavljaju programske ili tehnološke koncepte koji sačinjavaju aplikacije i primenjene poslovne objekte. One su komponente IS i aplikativnog okruženja. Primeri tehnoloških objekata su komponente grafičkog korisničkog interfejsa, programske konstrukcije koje realizuju usluge mrežne distribucije objekata, baze podataka itd. Tehnološki objekti su tako struktuirani da omogućavaju punu slobodu pri kreiranju aplikacija, olakšavaju njihovo održavanje, a selekcija ovih objekata se vrši u skladu sa vodećim standardima. Poštovanje standarda omogućava integraciju i kompatibilnost proizvoda različitih proizvođača. Aplikativni objekti su programi koji prezentuju zahtevanu informaciju i omogućuju komunikaciju krajnih korisnika sa IS. Oni predstavljaju rešenja određenih poslovnih rešenja, angažujući odgovarajuće skupove poslovnih objekata u cilju realizacije specifičnih zadataka. Aplikativni objekti su sastavljeni od poslovnih i tehnoloških objekata koji uz pomoć programskog koda daju određena aplikativna rešenja. Aplikativnim objektima se automatizuju određeni poslovni procesi i mogu služiti kao aplikativna podrška odlučivanju.
Postoje 3 vrste poslovnih objekata: Entiteti, Događaji, Procesi.

17. Spajanje Klijent/ Server arhitekture (K/SA) i objektne tehnologije

K/SA IS se može analizirati sa 2 aspekta: - Prvi se odnosi na arhitekturu aplikacije (prezentacije, funkcionalna logika i podaci); - Drugi način se odnosi na konfiguraciju hardverskih komponenti IS. Prezentacija se odvija na liniji korisničkih interfejsa i on upravlja dešavanjima na ekranu korisnika. Funkcionalna logika preslikava način odvijanja automatizovanih poslovnih procesa. Funkcionalna logika je obuhvaćena programskim kodom i obuhvata odgovarajuće usluge za izvršavanje tog koda. Deo aplikacije koji se odnosi na podatke obezbeđuje smeštanje svih relevantnih podataka poslovnog sistema kao i usluge neophodne za pristup i upravljanje podacima. Kod klasične konfiguracije hardverskih komponenti se uočavaju 2 nivoa (TWO TIER model) koji dele odgovornosti za realizaciju aplikacije: **1.** Nivo servera (odgovoran za upravljanje podacima); **2.** Nivo klijenta (Odgovoran za izvršavanje logike i prezentaciju dobijenih podataka). Kod ovog modela klijent i server komuniciraju direktno bez posrednika pomoću komunikacionih mehanizama. Svaka promena bilo u šemi baze podataka ili u logici funkcionisanja kod ovog modela podrazumeva modifikaciju kompletne aplikacije. Zbog toga ovaj model IS je skup za održavanje i ne poseduje potrebnu fleksibilnost (nema reinženjering procesa). Sledeća K/SA je THREE TIER model gde se između klijenta i servera instalira 3 nivo međuservera koji realizuje logiku funkcionisanja aplikacije i usluge u distribuiranom okruženju. Korisničke radne stanice obavljaju zadatke vezane za prezentaciju aplikacije, a linija krajnjeg servera vrši obradu podataka. Kod ovog modela delovi aplikacije koji se odnose na prezentaciju, funkcionalnu logiku i podatke su nezavisne komponente aplikacije.

18. Prednosti THREE TIER modela

1.Arhitektura fizičke konfiguracije je u saglasnosti sa arhitekturom aplikacije što povećava performanse IS. Ovo znači da svaka od komponenti može biti modifikovana ili zamenjena i da to ne zahteva dodatne intervencije u ostatku aplikacije. **2.** Modul aplikacije moguće je koristiti na različitim platformama pošto su delovi aplikacije nezavisni moduli povezani odgovarajućim interfejsom. **3.** Usluge i f-je koje procesiraju među serveri mogu da se koriste od strane mnogih aplikacija. Treća faza evolucija K/SA je preobražaj THREE TIER K/SA u distribuirano objektno okruženje (DOC). Ova transformacija se sprovodi implementacijom tehnoloških aplikativnih i poslovnih objekata. Prelazak na distribuiranu TREE TIER aplikaciju omogućava pretragu prostora stanja i sistema i traganje za optimalnim stanjima. Na ovaj način IS dobija obeležje sistema za podršku odlučivanju.

19. GUIDS metoda

Ovo je metoda za OBJEKTNO ORJENTISANO projektovanje sastoji se iz 5 faza: Projektovanje na osnovu ciljeva (GOAL), Sprega između korisnika i aplikacije (USER INTERFACE), Projektovanje implementacije (IMPLEMENTATION), Projektovanje podataka (DATA DESIGN) , Strategije za konstruisanje (STRATEGES FOR KONSTRUCTION). Nakon OBJEKTNO ORJENTISANOG projektovanja sledi OBJEKTNO ORJENTISANO programiranje gde su objekti povezani uglavnom preko tri osnovna tipa relacija: Pod klase, Kontejneri, Kolaboratori. Osnovni elementi OBJEKTNO ORJENTISANOG sistema su: Apstrakcija,Inkapsulacija, Nasleđivanje, Polimorfizam. Današnje OBJEKTNO ORJENTISANE metode koriste apstrakciju da se obezbedi pristup koji omogućava učestvovanje korisnika tokom razvoja aplikacije. Apstrakcija je usmerena na ono što je važan za aplikaciju a ne na samu implementaciju. Ovo omogućava da se aplikacije projektuju jezikom određenog posla a programerskom tehnologijom. Inkapsualizacija uvod i skrivanje privatnih svojstava i implementacije ponašanja. Njime se pristupa preko zadatog javnog interfejsa zbog toga se objekti mogu koristiti bez poznavanja njegove implementacije. Nasleđivanjem se obezbeđuje izbegavanje ponavaljanja sličnosti i razlika među objektima. Polimorfizam dozvoljava da javni interfejs objekta koji imaju slične funkcije imaju ista imena procedura ali drugačiju implementaciju. Ne mora se znati klasa objekta da bi se izvršilo polimorfno ponašanje.

20. Proces razvoja softvera

Uključuje sledeće faze: Ideja, Zahtevi, Plan i raspored, Arhitektura, Izgradnja, kontrola. Prvi korak u ostvarivanju neke ideje je proširenje ideje na skup specifičnih zahteva koji definišu projekat i očekivanja korisnika i tehničkog osoblja. Projektant definiše zahteve korisnika u saradnji sa njima. Ovi zahtevi se dokumentuju i služe kao osnova projekta aplikacije. Pristup na osnovu ciljeva u saradnji sa korisnicima definiše prave ciljeve koji proizilaze iz njihovih potreba. U ovom slučaju rade se sledeći koraci: -Formira se projektni tim; - Urade se domaći zadaci; - Postave se ciljevi; - Odredi se obim; - Indentifikuju se potrebe; - Pretvaraju se potrebe u zahteve; - Određuju se prioriteti zahteva.

21. Planiranje i pravljenje rasporeda projekta

Plan projekta definiše kako će se projekat razvijati da bi zadovoljio zahteve definisane od strane korisnika. Prvo se pravi grub raspored projekta da bi se isplanirao vremenski period. Ovaj raspored se bazira na pretpostavkama ,a ne na detaljima projekta.

22. Razvijanje arhitekture

Metodologija za OBJEKTNO ORJENTISANO projektovane arhitekture (GUIDS) sastoji se od: Projektovanje na osnovu ciljeva, Projektovanje korisničkog interfejsa, Projektovanje implementacije, Projektovanje podataka, Strategije koje određuju pristup u pravljenju aplikacije (standardi i konvencije za programiranje, upravljanje, testiranje, i plan i raspored implementacije).

23. Izgradnja i kontrola

Projekat arhitekture koji je razvijen nekom OO metodologijom služi kao osnova za razvoj aplikacije. Kada budu završeni neki delovi programa počinje postupak kontrole. Kontrola može da obuhvati proveru programskih linija, ispitivanje programskih celina, ispitivanje njihove integrisanosti i ispitivanje sistema. Cilj kontrole je pronalaženje grešaka u programu što ranije u procesu razvoja. Gotova aplikacija zahteva održavanje koje može uključiti ispravljanje grešaka, proširivanje programa, promene pravila poslovanja itd.

3. Dihotomija aplikacija - sistemski softver

Obuhvata period od '65 - '85 god. Proizvođači hardvera su uz hardver počeli da isporučuju i određeni softver. Ovaj deo programske podrške poznat je pod imenom sistemski softver i lociran je kao interfejsni sloj između aplikacije koju razvija korisnik i hardvera za izračunavanje. Sistemski softver se deli na: Operativni sistem, Pomoćni programi
. Programeri više nisu morali biti upoznati sa k-kama hardvera, već pre svega sa načinom logičkog rešavanja problema na nekom višem programskom jeziku. Naj važniji k-ka ove faze je da aplikacija napisana na višem programskom jeziku se može prenositi i izvršavati na drugim računarskim sistemima. Prenosiva aplikacija stiče svojstvo gotove robe. Vizuelno alfanumeričko komuniciranje posredstvom ekranskog terminala postaje opšte prihvaćeni standard. U ovoj fazi korisnici se dele u 2 kategorije: Programeri, Pravi korisnici.

4. Paradigma korisniku bliskog sistema

Od '85 pa na dalje pojavom PC računara dolazi do velikog skoka u razvoju IS. Dolazi do potpuno nove arhitekture programske podrške. Tipičan korisnik je vrlo malo upoznat sa k-kama hardvera (tehničke k-ke i mogućnosti računara). Interakcija između korisnika i PC - ja postaje ključni problem jer je sada cena eksploatacije računara manja od cene rada korisnika, pa zbog toga aplikacija mora da ispunjava želju korisnika. Veći deo hardvera i softvera se javlja u ulozi interfejsa između korisnika i aplikacije. U fazi razvoja IS na tržištu se jasno razlikuju proizvođači hardvera, sistemskog softvera i aplikativnog softvera.

5. Računarska mreža

Iako nije bio zamišljen za realizaciju složenih IS, PC kao radna stanica u lokalnoj računarskoj mreži postao je moćno sredstvo za realizaciju IS firmi umerene veličine. Razmena podataka sa drugim entitetima obavlja se preko računarske mreže koja može biti sastavljena od različitog hardvera kao i od raznih operativnih sistema. Problem konektivnosti (povezivanje heterogenih računara) rešen je razvojem komunikacionih protokola ('70 god.) kao što je TCP/IP. Internet takođe koristi TCP/IP protokol što omogućava realizaciju globalnih IS u kojem svaka informacija postaje dostupna svim korisnicima sistema.

6. Koncept upravljana projektima

Za realizaciju složenih programa i projekata stvorena je koncepcija upravljanja projektom (Project menager). Koncept je razvijen da bi se u složenim projektima postigli planirani ciljevi u planiranom vremenu i sa planiranim troškovima. Koncept se bazira na uspostavljanju takve organizacione forme koja na najbolji način omogućava iskorišćenje raspoložive metode planiranja i kontrole za efikasnu realizaciju projekta.

7. Definisanje koncepta upravljanja projektima (UP)

Prepisi celo 6. pitanje .plus ovo: UP u sebi sadrži: - Postavljanje cilja (rezultat, vreme i troškovi); - Planiranje (određivanje strukture, izračunavanje vremena, utvrđivanje budžeta i planiranje resursa); - Organizaciju (funkcije, kompletiranje personala, neophodne instrukcije, uzajamna povezanost); - Kontrola (rezultat, vreme i troškovi). Osnovne k-ke koncepta UP su: - Definisanje i korišćenje najpogodnije organizacije za upravljanje realizacijom projekta; - Formiranje i korišćenje odgovarajućeg IS za upravljanje realizacijom projekta pomoću odgovarajućeg softvera; - Korišćenje tehnike mrežnog planiranja i gantograme u planiranju, praćenju i kontroli realizaciji projekta. Uspešno UP u sebi sadrži 8 funkcionalnih oblasti: - Upravljanje klimom projekta ; -

Upravljanje troškovima
; -
Upravljanje vremenom
; -
Upravljanje kvalitetom
; -
Upravljanje ljudskim resursima
; -
Upravljanje komunikacijama
; -
Upravljanje ugovaranjem
; -
Upravljanje rizikom
. Standardni programi za UP su: Primavera, Super project, Projacs, PMCS/66, OPTIMA.

8. Upravljane softverskim projektima

Kvalitetni razvojni softverski paketi mogu se realizovati jedino kombinovanjem 3 kritična faktora: Dobrih kadrova, Upravljačkih tehnika, Tehnologija. Sprovođenje upravljačkih tehnika sprečava rasipanje ljudske energije (primena standarda ISO9000 i SEI). Najbolje organizacije fokusiraju se na sveobuhvatnu metriku procesa i usavršavanje, a prosečne na formalizaciju testiranja, upravljanje konfiguracijom, tehnike revizije koda kao i na formalizaciju odgovarajućih metoda softverskog inženjerstva za sistemsku analizu, dizajn i implementaciju. Za dobro upravljanje softverskim projektima potrebno je jasno definisati uloge i odgovornosti unutar definisanog procesa. Definisani procesi se jasno ažuriraju i usavršavaju kroz pilot testove i analizu troškova.Potrebna je da menadžeri prate kvalitet softverskih proizvoda i zadovoljstvo korisnika. Ovakav okvir upravljanja softverskim projektima predstavlja CMM model (model zrelosti projekta) koji trasira put za postizanje kontrole razvoja i održavanja softvera. CMM čini 5 nivoa zrelosti procesa: - Inicijalni nivo, k- še kao povremen čak kao haotični, samo su neki procesi definisani a uspeh zavisi od pojedinačnih napora; - Pon
avljajući nivo
, vrši se planiranje i praćenje projekta. Ranija uspešna iskustva se dosledno primenjuju; -
Definisani nivo
, i upravljačke i upravljane aktivnosti su dokumentovane i standaradizovane u standardni softverski proces organizacije; -
Upravljivi nivo
, softverski proces

se procenjuju i kontrolišu; -
Opimizirajući nivo
, konstantno se poboljšava softverski proces što se postiže povratnom spregom koja uključuje sam proces i realizaciju novih ideja i tehnoloških inovacija.

9. Metodologija - organizacioni aspekti

Osnova ove metodologije zasniva se ne sugestijama da projekat treba da realizuje projektni tim koji se nalazi pod nadzorom upravljačkog tima. Upravljački tim izrađuje nacrt projekta i formira projektni tim. Rukovodeći tim nadgleda posao, obezbeđuje potrebne resurse rešava eventualne problema i služi kao iterfejs između projeketnog tima i ostatka organizacije proizvođača. Rukovodeći tim ne upravlja projektnim timom već to čini šef proj. tima. Projektni tim sadrži vodećeg dizajnera koji je i vođa tima on odgovara i podnosi izveštaje rukovodećem timu. Ostali članovi rukovodećeg tima su implementator (programer koji piše kod) i tehnički recenzent (programer koji ocenjuje tehničke aspekte implementacije). Ukoliko se radi o složenom projektu on se deli na podprojekte i stvara se više projektnih timova. Među njima je potrebno ostvariti koordinaciju što je uloga koordinatora koji je ujedno i vođa projekta. U procesu planiranja identifikuju se standardi za kodiranje testiranje i upravljanje konfiguracijom, vođenje projektne dokumetacije itd.

10. Metodologija - procesni aspekti (PADRE metodologija)

Proces na nivou objekta sadrži: 1. Postavljanje projekta: - organizacija izvođača (formiranje upravljačkog tima); - Upravljački tim (izrađuje nacrt projekta i formira projektni tim).
2.
Planiranje projekta, u ovoj fazi projektni tim deli projekat na etape vrši ustanovljavanje standarda i procedura za postizanje kvaliteta dobija potvrdu za planove od strane upravljačkog tima.
3.

Realizacija projekta, projektni tim prati proces na nivou etapa, a upravljački tim otklanja prepreke i obezbeđuje resurse.
4.
Ocena projekta, oba tima rade na Pronalaženju načina za poboljšanje proizvoda projekta, planiranje projekta i načina njegove realizacije.

Proces na nivou etape sadrži: 1. Planiranje etape, etapa se deli na module i vrši se dodela svakom članu tima, te se dobija potvrda za planove od rukovodećeg tima.
2.
Realizacija etape, projektni tim prati proces na nivou modula i koriguje proizvod etape. Vođa tima otklanja prepreke i obezbeđuje resurse, i prati rezultate provere koju vrši korisnik.
3.
Ocena etape, vrši je projektni tim.

Proces na nivou modula sadrži: 1. Planiranje modula, programer ili vodeći programer razvija dizajn implementacije i testiranje. **2.** Realizacija modula, programer vrši programiranje modula prateći detaljno dizajn, vodi evidenciju o napredovanju odgovarajućim razvojnim alatom i koriguje implementaciju prema rezultatima provere koju vrši projektni tim. **3.** Ocen
a modula, vođa tima i programer rade na pronalaženju načina za poboljšanje proizvoda modula.

Na svakom nivou proces uključuje sledeće korake: **1.** Planiranje (plan). **2.** Realizacija (Do). **3.** Ocena (Evaluate). Svaki nivo takođe uključuje potvrdu planiranog koraka (Approval) i iterativnu proveru (Review) i korekciju (Revise) rezultata koraka realizacije.

11. Planiranje projekta i etapa

Planiranje se izvodi od vrha ka dnu dok se posao unutar etapa ne podeli na manje delove (module). To nisu moduli u obliku podprograma već jedinice posla. Svaki član projektnog tima koristi proces planiranja na nivou modula za upravljanjem razvoja svog modula, dok vođa tima upravlja serijom modula, što omogućava lakše praćenje progresa u poslu, rano otkrivanje problema itd. Rezultat planiranja etape se koristi za planiranje modula. To daje detaljan plan za

etapu koji se sastoji iz Pert dijagrama, kao dodatak ovome dodaje se i opis svakog modula što predstavlja preliminarni dizajn.

12. Proces na nivou modula

Svaki modul prolazi kroz mini životni ciklus sa 5 kontrolnih tačaka: Dizajn (20%), Potvrda (5%), Kodiranje (40%), Provera (15%), Korekcija (20%
).

Na svakom nivou modul prolazi sledeće korake:
1.
Planiranje (plan).
2.
Realizacija (Do).
3.
Ocena (Evaluate). Svaki nivo takođe uključuje potvrdu planiranog koraka (Approval) i iterativnu proveru (Review) i korekciju (Revise) rezultata koraka realizacije. Što je u suštini PADRE metodologija od vrha ka dnu. Ovakav životni ciklus u softverskom inžinjerstvu naziva se spiralni životni ciklus.

13. Iterativna strategija

Kompleksne sisteme je teško formalno specificirati. Nemoguće ih je specificirati u prvom ciklusu, a retko i drugom. Zbog toga se zahtevi kod ovakvih sistema menjaju tokom razvoja. Zbog toga se prihvata iterativna strategija u razvoju softvera. Po PADRE metodologiji postoji 5 tehnika za borbu protiv kompleksnosti softvera: **1.** Planiranje i realizacija, vrši se razbijanjem projekta preko etapa na dvonedeljne radne celine, module koji se realizuju kroz mini životni ciklus. **2.** Objektna tehnologija
koja je vrlo pogodna za iterativni razvoj.

3.
Alati za realizaciju prototipova
, omogućuju jednostavnu realizaciju dizajna aplikacija. Realizacija prototipa i korisničkog interfejsa pomaže dizajnerima da svoje ideje iskažu korisnicima.
4.
Inkrementalno testiranje
, svaki modul se potpuno testira pre uključenja u postojeći sistem. Programer piše test procedure kojima testira svoj modul.
5.
Spiralni životni ciklus.

14. Objektno orjentisana metodologija i distribuirani računarski sistemi

Pri projektovanju IS koji će preslikati ponašanje i strukturu poslovnih sistema i ujedno omogućiti transformaciju i rast sistema, dolazilo je do nezadovoljavajućih rezultata u domenu fizičke arhitekture IS i domenu arhitekture aplikacija koje opisuju poslovne procese. Osnovni problem programera je kompleksnost ovakvog sistema. Poslednji odgovor na izazov komleksnosti je objektni pristup u sagledavanju i modeliranju modernog sveta. Ovaj pristup rezultira u paradigmi koja se u ambijentu funkcionisanja poslovnih sistema naziva objektnom tehnologijom.

15. Objektno orjentisana tehnologija (OOT)

OOT je način modeliranja realnih sistema. Počinje OBJEKTNO ORJENTISANOM analizom koja je osnov za OBJEKTNO ORJENTISANO dizajniranje modela i preko OBJEKTNO ORJENTISANOG programiranja rezultira u softveru sa željenim osobinama. Analitičaru je ostavljeno da odredi nivo apstrakcije na kome ga interesuje ponašanje realnog sveta i da na osnovu toga projektuje osnovne funkcionalne elemente posmatranog sistema u obliku klasa. Na osnovu klasa analitičar će nasleđivanjem osnovnih klasa će stvoriti čitavu hijerarhiju elemenata posmatranog sistema koje preslikavaju njihove atribute ponašanje i interakcije. Objekt prezentuje objekt iz realnog sistema obuhvatajući njegove atribute, funkcije i ograničenja. Svaki

objekat poseduje identitet i komunicira sa okruženjem pomoću svojih metoda. Klase su tipovi podataka u nekom od programskih jezika koji podržavaju OBJEKTNO ORJENTISANU metodologiju. **Enkapsulacija** ili skrivanje podataka, je zaštita unutrašnje realizacije tipa od pristupa spolja. Ovo podrazumeva strogo kontrolisan pristup podacima unutar tipa. Ova osobina obezbeđuje autonomnost tipa što eliminiše rizik pojave greške u ostatku modela jer se ni jedan drugi objekat ne oslanja na podatke koji mu nisu dostupni. **Nasleđivanje** , podrazumeva izgradnju novih, izvedenih klasa koje nasleđuju podatke i funkcije od već definisanih klasa i još im se dodaju nove f-je i podaci što rezultira u evoluciji određene hijerarhije klasa
. Polimorfizam
je k-ka koja podrazumeva mogućnost pojavljivanja u više oblika. OBJEKTNO ORJENTISANI programski jezici omogućavaju pomoću definisanja virtualnih funkcija da se jedna ista virtualna funkcija na različite načine koristi u različitoj hijerarhiji.

Svi problemi iz inženjeringa poslovnih procesa postaju tipovi podataka u domenu softverskog inženjerstva. Objekti su potpuno ne zavisni od vrste programskih jezika u kojima su kodirani, od alata koji je kreirao aplikaciju i od platforme na kojima ta aplikacija funkcioniše. OBJEKTNO ORJENTISANI pristup razvoju softvera se preslikava u domen modeliranog sistema čime se u modelu IS i modeliranom poslovnom sistemu otvara mogućnost upravljanja rastom postojećih rešenja uz poštovanje izabranih kriterijuma. Svako novo rešenje je rezultat evolucije prethodnog, što smanjuje razvojni ciklus i time pruža brži odgovor na neophodnost konstantnih izmena.

24. Neke prednosti korišćenja OO pristupa

Apstraktni objekti omogućuju standardni jezik timu projektanata i bolji poslovni model. GUIDS metoda daje puteve za uključivanje korisnika u fazi stvaranja arhitekture. OO projektovanje smanjuje komleksnost programiranja, obezbeđuje logiku i red u načinu na koji se delovi aplikacije uklapaju. Ovo proces razvoja čine efikasnijim jer se zna ko, šta i treba da radi. Segmenti aplikacije moraju biti nezavisni. OO pristup pravi zidove između segmenata definišući nezavisne zatvorene objekte. Interakcija ovih objekata se obavlja preko javnih interfejsa. Klase u OO projektovanju opisuju dobro definisane i nezavisne komponente. Inkapsulacija klasa čini implementaciju nezavisno od ostatka sistema, što pojednostavljuje izmene i testiranje aplikacije. OO projektovanje dozvoljava da se prave nezavisne komponente koje se mogu kombinovati za probne verzije. OO projektovanje dozvoljava da se definišu sve komponente aplikacije i da se uključe u raspored kao zadaci koji se mogu definisati i proceniti.

25. Određivanje standarda kvaliteta

Standardi kvaliteta određuju neke opšte uslove za zahtevane kvalitete aplikacije. Pri definisanju standarda kvaliteta treba razdvojiti sledeće faktore: - Jednostavnost upotrebe; - Usaglašenost sa ustanovljenim konvencijama za korisnički interfejs; - Mogućnos održavanja; - Pouzdanost; - Performanse; - Prihvatljivi nivo programskih grešaka;

- Kompatibilnost sa drugim srodnim sistemima
. Definisanjem standarda kvaliteta u fazi preduslova za projekat postiže se prilagođenost tokom celog razvoja. U protivnom umesto da bude primarni projektni zahtev, standardi kvaliteta postaju spisak bagova. Norme za projektovanje softvera uključuju: Minimalna hardverska konfiguracija, Preporučena hardverska konfiguracija, Operativni sistem, Mrežna konfiguracija, Jezici, Baze podataka, Prenosivost, Mogućnost ponovnog korišćenja (REUSABILITY), Projektovani broj korisnika, Sigurnosni zahtevi, Veza prema drugim sistemima, tekući tehnički standardi i programerske konvencije, Vreme dooglašavanja, Internacionalizacija.

26. Analiza kvaliteta softvera primenom ISHIKAWA analize

Pošto na kvalitet softvera utiču mnogi faktori te faktore je potrebno grupisati putem neke metode primer ISHIKAWA metode. Kod ove metode se koristi dijagram za prikaz i razmatranje odnosa između date posledice i njenih mogućih posledica. Dijagram se konstruiše tako da se svi mogući uzroci razmatraju i organizuju u kategorije i podkategorije shodno međusobnim hijerarhiskim odnosima. ISHIKAWA dijagram omogućuje: Analizu i prikaz uzročno poslediičnih odnosa, Olakšava rešavanje problema počevši od utvrđivanja siptoma preko posledиica pa do rešenja. U osnovne uzroke koji utiču da se dobije dobar softver spadaju: Čovek, Merljivost, Sistem podataka (Menadžment podataka IS), Materijal, Metode, Okruženje, Oprema. Uticaj čoveka na razvoj softvera izražava se kroz njegovu motivaciju okruženje sticanje novih znanja i dobro školovanje. Merljivost se odnosi na brzinu izvršavanja aplikacije, stepen jednostavnosti upotrebe softvera, osobine kompajlera, debagovanje i BP. Sistem podataka dobrog softvera sadrži zahteve korisnika, Podatke iz domena, razne primere i literaturu za posmatrani problem. Materijal se odnosi na literaturu vezanu za posmatrani problem. Kod metoda vodi se računa o metodama projektovanja i testiranja. Od metoda za razvoj softvera razlikuju se modularni i OO pristup. Pod okruženjem se misli na troškove, budžet i rokove projekta. Oprema podrazumeva odgovarajuću hardversku konfiguraciju, Operativi sistem i razvojni alat koji se koristi. Projektni

zahtevi koji su osnov izrade svake aplikacije, treba da omoguće jednostavnu izradu i održavanje aplikacije. U okviru ovih zahteva mogu se izdvojiti: - Interakcija sa drugim programima koja se ogleda u radu sa raznim formatima dokumenata; - Iskorišćenost mogućnosti OS; - Stabilnost softvera; - Prenosivost koda na različite platforme i OS; - Rad u mreži; - Jednostavnost korisničkog interfejsa; - Hardverski zahtevi aplikacije određuju računar na kojem aplikacija može da se razvija. Primenom komparativne ISHIKAWA metode svakoj grani dijagrama može se dodeliti koeficijent od 1 do 10, koji opisuje značaj te grane. Postoje 2 načina zadavanja koeficijenata: - Top-Down, polazi od definisanja koeficijenata na najnižem nivou koju zadaje stručnjak ili tim koji ima najviše znanja iz tog domena. Na osnovu koeficijenta pretka, potomcima se dodeljuje težinski koeficijent; - Bottom-Up, nema nikakvih izračunavanja koeficijenata već se koeficijenti mogu uneti za svaku granu posebno bez obzira da li ima potomke ili ne. Moguće je upoređivanje 2 ISHIKAWA dijagrama tako što će se formirati dijagrami iste strukture kojima će se pored težinskog koficijenta pridružiti i neki novi koeficijenti: - Zahtevni koeficijent i –te grane n – tog nivoa (Isto što i težinski koeficijent); - Ostvareni koeficijent i – te grane n - tog nivoa (opsisuje koliko je ostvaren uticaj uzroka na posledicu); - Koeficijent relativnog značaja i – grane n – tog nivoa (koeficijent procentualnog učenja); - Ukupni koeficijent. ISHIKAWA metoda pomaže u poboljšanju kvaliteta softvera i odabiranja prave razvojne platforme.

27. Multimedije i ekspertni sistemi

Razvoj tehnologije ekspertnih sistema najvećim delom je bio nezavisan od razvoja multimedijalnih sistema. Ovaj razvoj je doveo do pojave niza multimedijalnih uređaja i njihovog povezivanja sa ekspertnim sistemima. Ova generacija se javlja zbog problema koji multimedijalni nisu mogli sami da prevaziđu. Kao što su interaktivan pristup podatcima, efikasno selektovanje i prikaz datih podataka, učenje i poređenje koristeći obimne informacije i akumulirano znanje. Integrisanjem ove dve tehnologije dobijamo novo nastali sistem koji pored dobrih osobina i jednog i drugog dobija i nov kvalitet koji nastaje na osnovu komplementarnosti obilja tehnologija. Ekspertni sistemi pored reda sa alfa- numeričkim podacima dobijaju dobijaju mogućnost rada sa slikom, govorom, animacijama, vide zapisom itd. Uključivanjem multimedije u ekspertne sisteme dobija se bolja komunikacija korisnika sa sistemom, učenje korisnika postaje efikasnije, a donošenje odluka bolje. Multimedija može da omogući i modeliranje šireg opsega znanja, predstavljanje nove kategorije modela znanja i poboljšanje dela ekspertnog sistema za obrazloženje tipa ¨zašto?¨. Multimedijalni sistem koristi skupljena znanja u ekspertnom sistemu radi poboljšanja korisničkog interfejsa tj. Stvaranje inteligentnog multimedijalnog korisničkog interfejsa. Proces integracije ovih sistema dovodi do generičke arhitekture interaktivnog multimedijalnog ekspertnog sistema (IMMES).

28. Modeli realizacije IMMES

IMMES se može dekomponovati na bar dve softvreske komponente, jednu kod koje je ekspertni sistem dominantan i drugu kod koje je multimedija dominantna. Postoji 5 mogućih načina povezivanja ekspertnih sistema i multimedija: - Povezivanje arhitektura nezavisnih sistema: - Integracija rešenja: - Slabo spregnuti sistemi; -Čvrsto spregnutu sistemi; - Integrisano optimizovano rešenje . Povezivanje arhitektura nezavisnih sistema je multinovo integracije ekspernih i multimedijalnih sistema. Prednosti ove aritekture su: Raspoloživost komercijalnih alata, Jednostavnost, brzina razvoja, Nezavisno održavanje. Nedostatci su: Nepostojanje bilo kakve integracije i izostavljanje sinergizma. Integracija rešenja je prvi nivo integracije EX i MM sistema. Aplikacija se sastoji od više aplikacija razvijenih pomoću različitih alata. Koristeći ovaj model za aplikaciju razvijenu sa jednim alatom neophodno je izvršiti translaciju sadržaja i strukture. Zato je potreban određeni nivo standardizacije. Prednosti ovog modela su: Raspoloživost komercijalnih alata, Jednostavnost, brzina razvoja, brzina isporuke sistema. Nedostatak: Nije ekonomično rešenje i potpuno tačna translacija može biti neizvodljiva. Slabo spregnuti sistem omogućava komunikaciju EX i MM sistema preko datoteke podataka. To znači da postoji kooperativno procesiranje preko datoteke podataka. Prednost je poboljšani singerizam. Nedostatak je što postoji dodatno procesiranje podatka u toku komunikacije. Čvrsto spregnutu sistemi realizuje komunikaciju između EX i MM sistema preko mehanizma slanja poruka ili mehanizma deljene memorije. Prednosti ovog modela integracije su: Povećana robusnost, Modularnost, Fleksibilnost projektovanja i Brže izvršavanje aplikacija u odnosu na slabo spregnute sisteme. Nedostatci: Nepostojanje komercijalnih alata, Povećanja složenost razvoja, održavanja,

redundantni parametri i procesiranje podataka i pitanja validacije i verifikacije. Integrisano optimizovano rešenje optimizuje performanse celog sistema u odnosu na aplikaciju. U ovakvom razvojnom okruženju podržani su i koncepti EX i MM sistema. Prednosti: Najviši nivo singerizma, Robusnost, Odlične performanse u toku izvršavanja, Eliminisana redudantnost, Poboljšano rešavanje problema. Nedostatci: Povećano vreme razvoje i Nedostatak komercijalnih alata. Ovaj model je najbolje rešenje za IMMES.

29. Generička arhitektura IMMES-a

Korisnički interfejs postaje i audio i vizuelni interfejs. IMMES u ovom slučaju koristi tri baze podataka: Alfanumeričku u kojoj se nalaze standardni alfanumerički podaci, MM BP koja sadrži slike i audio-vizuelne podatke i Bazu podataka znanja koja generiše znanje. Kod ovakve arhitekture IMMES najvažniji je podsistem za upravljanje podacima. Glavni zadatak podsistema za upravljane podacima je da garantuje efikasan pristup podacima bez obzira na veličine datoteka u kojima se podaci nalaze. Operatori koje ovaj podsistem treba da podržava su: INSERT, DEL, UPDATE, SELECT, JOIN, CASH JOIN i SORT. Podsistem za upravljanje objektima sadrži model objekata, programski prevodilac, biblioteke potrebne za rad u toku izvršavanja programa i interpreter.

30. Mogući načini za poboljšanje performansi

Poboljšanje ili pronalaženje novih algoritama struktura podataka za pristup podacima (u praksi se koristi više verzija B+ stabla) **je način za** poboljšanje performansi sistema. B+ struktura ima podjednako vreme pristupa bilo kom slogu na osnovu vrednosti ključa, ima odličnu brzinu pristupa, dobar način održavanja indeksne strukture i opseg slogova. Upravljanje B+ stablom garantuje balansiranje stabla (svi listovi se nalaze na istoj dubini). Jedina posledica je nešto veće zauzeće memorijskog prostora.

31. Nove arhitekture i pristupi OO razvoju softvera

U savremenim pristupima razvoju softverskih sistema jasno se razlikuju sledeći koncepti: - Funkcionalna specifikacija sistema
koja definiše šta sistem treba da radi; -
Arhitektura sistema
koja opisuje šta treba da se izgradi; - Modeli sistema koji opisuju sistem sa različitih aspekata; -

Metodologija
koja definiše redosled aktivnosti koje treba preduzeti da bi se željeni sistem izgradio i koristio.

32. Funkcionalna specifikacija sistema

Formalno definiše zahteve korisnika koji će biti u neposrednoj interakciji sa budućim sistemom.U savremenim OO pristupima funkcionalna specifikacija se zadaje pomoću slučajeva korišćenja. Svaki kasniji korak u razvoju sistema treba da ima jasan trag do odgovarajućeg slučaja korišćenja. Odgovarajući slučajevi korišćenja definišu buduću arhitekturu sistema. Međutim detaljna specifikacija novog slučaja korišćenja zavisi i od usvojene arhitekture sistema. Zbog toga se specifikacija slučaja korišćenja i arhitekture sistema zajednički ugrađuju u procesu razvoja sistema što utiče na karakteristike životnog ciklusa ovog proizvoda.

33. Arhitektura sistema

Arhitektura sistema opisuje šta treba da se izgradi. Pored zahteva korisnika na arhitekturu utiču okruženje u kome se sistem razvija. Postojeći softver i gotove softverske komponente: aplikacioni kosturi i mustre. Savremena razvojna okruženja predstavljaju arhitekture koje u sebi sadrže mnoštvo blokova koji se mogu direktno koristiti, specializovati ili dograditi u procesu razvoja softverskog sistema.

34. Modeli sistema

Različiti modeli sistema izgrađuju se korišćenjem nekog formalnog i opšte prihvaćenog jezika. Takav je danas UML. On omogućuje da se opiše projektovani sistem sa različitih aspekata: - Strukturnog
(dijagrami klase i dijagrami objekata); -
Dinamičkog
(dijagrami slučajeva korišćenja, promene stanja itd.); -
Fizičkog
(dijagrami komponenti i dijagrami razmeštanja); -
Sa aspekta upravljanja modelima
(paketi, podsistemi, modeli).

35. Proces razvoja sistema

Za razliku od konvecijolnalnih metodoloških pristupa, OO pristupi omogućavaju i zagovaraju efikasan iterativno – inkrementalni pristup. U iterativno – implementacionom modelu životnog ciklusa podrazumeva se razvoj sistema malim koracima po strategiji: - Planiraj malo; -Specifiraj, projektuj i implementiraj malo; - Integriši, testiraj i koristi svaku iteraciju malo.

36. Arhitekture programskih sistema

Arhitektura softverskog sistema definiše strukturne i dinamičke karakteristike sistema. Preko ovih karakteristika treba da se iskažu upotrebljivost, funkcionalnost, performanse kao i ekonomska i tehnološka ograničenja sistema. Najznačajnija karakteristika savremenih softverskih arhitektura treba da bude njena prilagodljivost brzim promenama u poslovnom i tehnološkom okruženju. U K/SA razlikujemo dvoslojni i troslojni generički model arhitekturi. Dvoslojna K/SA učinila je nezavisnim BP i aplikacije da bi se obezbedio integritet BP nezavisno

izvan aplikacija koje nad njom rade. Jedini mehanizam potreban za komunikaciju aplikacije sa BP je SQL upit koji se prosleđuje direktno kroz računarsku mrežu. Ovakva arhitektura ima dosta nedostataka. Značajan deo aplikacije se vezuje za klijenta. Ovo prouzrokuje skupo održavanje zbog čestih izmena u tehnologiji korisničkog interfejsa. Ovako razvijen softver je nedovoljno modularan, teško prenosiv i loše podržavaju veliki broj korisnika.U troslojnom generičkom modelu jasno se odvaja upravljanje podacima, aplikaciona logika i korisnički interfejs. Suština ove arhitekture odražena je u srednjem sloju , aplikacionom serveru. Aplikacioni server omogućava transparentno povezivanje aplikacija sa različitim izvorima podataka na različitim platformama a ne samo sa jednim serverom. Poseban značaj imaju distribuirane arhitekture softverskih komponenti . Zadatak ovih arhitektura je da omogući izgradnju nove aplikacije na osnovu postojećih komponenti i postojećeg koda izgrađujući pri tome nove komponente za buduće šire korišćenje. Ovakva arhitektura rešava problem produktivnosti razvoja softvera i održavanja. Ovde nastaje problem kada treba povezati staru i dolazeću tehnologiju zbog toga što je stara tehnologija realizovana na veoma niskom nivou apstrakcije. Za rešavanje ovog problema usvojena je višeslojna arhitektura koja omogućuje uvođenje više nivoa apstrakcije na kojima se IS opisuje u okruženju definisanih poslovnih procesa i objekata. Predložene su dve arhitekture: - Prva BOA - OMG ; - Druga predložena od strane IBM. Arhitektura BOA – OMG predviđa sloj jedinstvenih poslovnih objekata i sloj poslovnih objekata specifičnih za datu organizaciju. IBM arhitektura predviđa sloj zajedničkih poslovnih objekata i osnovnih poslovnih procesa. Komponente na ovim nivoima na obe arhitekture nisu zatvorene već ih je moguće modifikovati i nadograđivati. Ove komponente mogu da budu: - Softverske komponente; - Mustre (opšte rešenje za opšte objekte); - Kosturi (složenija mustra kojom se definiše neka opšta aplikacija). Infrastrukturni nivoi oba modela omogućuju izgradnju i funkcionisanje poslovnih objekata i procesa i da stave na raspolaganje skup nekih opštih uslužnih servisa. Najniži nivoi u oba modela predstavljaju osnovni skup mehanizama za ostvarivanje arhitekture distribuiranih softverskih komponenti.

1. Paradigme arhitekture softvera

2. Monolitna programska podrška

3. Dihotomija aplikacija - sistemski softver

4. Paradigma korisniku bliskog sistema

5. Računarska mreža

6. Koncept upravljana projektima

7. Definisanje koncepta upravljanja projektima (UP)

8. Upravljane softverskim projektima

9. Metodologija - organizacioni aspekti

10. Metodologija - procesni aspekti (PADRE metodologija)

11. Planiranje projekta i etapa

12. Proces na nivou modula

13. Iterativna strategija

14. Objektno orjentisana metodologija i distribuirani računarski sistemi

15. Objektno orjentisana tehnologija (OOT)

16. Kategorija objekata

17. Spajanje Klijent/ Server arhitekture (K/SA) i objektne tehnologije

18. Prednosti THREE TIER modela

19. GUIDS metoda

20. Proces razvoja softvera

21. Planiranje i pravljenje rasporeda projekta

22. Razvijanje arhitekture

23. Izgradnja i kontrola

24. Neke prednosti korišćenja OBJEKTNO ORJENTISANOG pristupa

25. Određivanje standarda kvaliteta

26. Analiza kvaliteta softvera primenom ISHIKAWA analize

27. Multimedije i ekspertni sistemi

28. Modeli realizacije IMMES

29. Generička arhitektura IMMES-a

30. Mogući načini za poboljšanje performansi

31. Nove arhitekture i pristupi OBJEKTNO ORJENTISANOM razvoju softvera

32. Funkcionalna specifikacija sistema

33. Arhitektura sistema

34. Modeli sistema

35. Proces razvoja sistema

36. Arhitekture programskih sistema