1. **Što je dizajn arhitekture i kada se radi**
   Proces identifikacije podsustava sustava, utvrđivanje okvira unutar kojih će ti podsustavi komunicirati i s čim će sve upravljati se naziva dizajn arhitekture. Radi se u ranoj fazi, često paralelno s nekim drugim aktivnostima.
2. **Prednosti eksplicitnog dizajniranja arhitekture (3)**
   Komunikacija sa zainteresiranim osobama, Analiza sustava, Ponovno korištenje
3. **Arhitekture mogu biti orjentirane na (5)**
   Performanse, Zaštitu, Sigurnost, Dostupnost, Održivost
4. **Odluke prilikom dizajna arhitekture (6+2)**
   Postoji li generička arhitektura koja može poslužiti kao predložak, Kako će sustava biti distribuiran, Kakvi su stilovi arhitekture pogodni, Kako će dizajn arhitekture biti vrednovan, Koji će biti fundamentalni pristup sustavu, Kakva će biti dokumentacija
5. **Koje modele arhitekture razlikujemo (5).**
   Statički strukturalni model, Model dinamičkih procesa, Model sučelja, Modeli odnosa, Distribucijski model
6. **Što je ADL(Architectural descritption languages)**
   Jezici za opis arhitekture. Kao osnovne elemente koristi komponente i konektore. Teško ih je analizirati jer su teški za razumijevanje.
7. **Organizacija sustava**
   Odražava osnovnu strategiju korištenu prilikom strukturiranja sustava.
8. **Organizacijski stilovi (3)**
   Model skladišta, klijent-server model, slojeviti model.
9. **Model skladišta (prednosti, mane)**
   Model sustava koji se temelji na dijeljenoj bazi podataka.
   Prednosti: dijeljenje velike količine podataka, sigurnost, kontrola pristupa i opravljanje od grešaka su centralizirani, jednostavna integracija novih podsustava.
   Mane: što se svi podsustavi moraju „složiti" oko modela skladišta, mora doći do kompromisa između, potreba podsustava, različiti podsustavi možda imaju različite zahtjeve za sigurnost, kontrolu pristupa itd.
10. **Klijent-server model (prednosti i mane)**
    Sustav je organiziran kao skup usluga, pridruženih servera i klijenata koji koriste te usluge.
    Prednosti: distribuirana arhitektura, jednostavno dodavati nove servere, raditi unaprjeđenje postojećih servera.
    Mane: redudantno upravljanje na svakom serveru, ne postoji središnji zapisnik servera i usluga koje nude.
11. **Slojeviti model (prednosti i mane)**
    Sustav je organiziran po slojevima koji pružaju skup usluga.
    Prednosti: pomaže kod inkrementalnog razvoja, arhitektura se jednostavno mijenja.
    Mane: podjela na slojeve može biti problematična, problemi s performansama jer svi podaci prolaze kroz više slojeva, viši slojevi mogu komunicirati sa nižim slojevima (narušena ideja da sloj komunicira samo sa susjednim slojevima)
12. **Stilovi razlaganja na module**
    • Objektni model - sustav se dijeli na skup objekata s dobro definiranim sučeljima koji međusobno komuniciraju.
    Prednosti: struktura sustava dobro razumljiva, objektne jednostavno ponovno koristiti.

Mane: objekti moraju eksplictno referencirati imena i sučelja drugih objekata, kompleksne entitete teško objektno prikazati

- Pipeline ili dataflow model - sustav se dijeli na skup funkcionalnih modula koji prihvaćaju podatke na ulazu i transformiraju ih u izlazne podatke.

Prednosti: manje apstraktan, podržava ponovno korištenje operacija, intuitivan, nadogradnja sustava je jednostavna.

Mane: mora postojati zajednički format za prijenos podataka, teško modelirati interaktivne sustave.

13. **Stilovi kontrole**

Kontrolni modeli se bave tokom kontrole između podsustava.

- Centralizirana kontrola -jedan podsustav kontrolira druge podsustave. Dvije kategorije: Pozovi-vrati model(kontrola na vrhu i ide prema dnu), Model upravitelja(upravitelj komponenta upravlja procesima).
- Kontrola temeljena na događajima - svaki podsustav može odgovoriti na događaje generirane izvana. Dvije kategorije: Prijenosni model(događaj se prenosi na sve podsustave), Model upravljan prekidima

14. **Referentne arhitekture**

Često se za sustave iz iste aplikacijske domene mogu koristiti slične arhitekture - arhitekture specifične za domenu *(domain-specific architectures).* Razlikujemo dva tipa takvih arhitektura:

- Generički modeli - apstrakcije izvučene iz velikog broja realnih sustava koje enkapsuliraju osnovne karakteristike tih sustava.
- Referentni modeli - apstraktniji su i opisuju Šire klase sustava. Predstavljaju idealiziranu strukturu koja uključuje sve mogućnosti koje sustav treba ispuniti.

15. **Distribuirani sustav(prednosti, mane) (5,4)**

Sustav kod kojeg je obrada informacija podijeljena na više računala.

Prednosti: Dijeljenje resursa, otovorenost, istovremenost, tolerancija greške

Mane: Kompleksnost, zaštita, upravljivost, nepredvidljivost

16. **Dva generička tipa arhitektura distribuiranih sustava**

- Klijent-server arhitektura
- Arhitektura distribuiranih sustava

Ove arhitekture se uglavnom koriste kod sustava koji funkcioniraju unutar jedne organizacije. Za sustave koji funkcioniraju između više organizacija se koriste: p2p peer-to-peer i servisno orijentirane arhitekture.

17. **Middleware(međuprogram)**

Upravlja različitim dijelovima distribuiranog sustava.

18. **Višeprocesorske arhitekture**

Najjednostavniji model distribuiranog sustava kod kojeg se softverski sustav sastoji od niza procesa koji se izvode na različitim procesorima.

19. **Klijent-server arhitektura**

Aplikacija je napravljena tako da se sastoji od skupa servera koji nude određene usluge i skupa klijenata koji te usluge koriste. Klijenti moraju imati određeno znanje o serverima (gdje su i koje usluge pružaju).

20. **Two-tier i three tier klijent server arhitekture(+primjeri)**

Najjednostavnija klijent-server arhitektura se naziva ‚Jwo-tier client-server" arhitektura

kod koje je aplikacija organizirana kao server i skup klijenata. Razlikujemo dva oblika:

- Model s jednostavnim klijentima (thin client model) — sva obrada i upravljanje podacima se odvija na serveru.
- Model sa složenim klijentima {fat client model) - server upravlja podacima. Klijentski softver implementira aplikacijsku logiku i interakciju s korisnikom.

Problem s ovakvom arhitekturom je što tri logička sloja (sloj upravljanja podacima, sloj obrade i prezentacijski sloj) moraju biti implementirani na dva sustava. Da bi se izbjegli ovakvi problemi koristi se three-tier client-server arhitektura kod koje su tri sloja logički odvojeni procesi koji se izvode na različitim procesorima.

21. **Arhitektura distribuiranih objekata**

Kod ovakve arhitekture osnovne komponente sustava su objekti koji pružaju sučelje prema setu usluga koje nude. Drugi objekti koriste te usluge pri čemu ne postoji logička podjela na klijente i servere. ORB poznaje sve objekte sustava i njihova sučelja i njegova glavna uloga je da pruži neprimjetno sučelje između objekata;

Prednosti: arhitektura je otvorena i omogućava jednostavno dodavanje novih resursa; sustav je fleksibilan

Mane: kompleksnost — teže ih je dizajnirati i razumjeti od klijent-server sustava.

22. **CORBA - standard za ORB**

Međuprogram koji podržava rad s distribuiranim objektima. Potreban je na dvije razine:

- Na razini logičke komunikacije • Na razini komponenti

23. **P2P (peer-to-peer)**

P2P sustavi su decentralizirau sustavi kod kojih operacije može izvršavati bilo koji čvor mreže. Razlikujemo:

• Decentraliziranu arhitekturu - Čvorovi su organizirani u područja s time da neki čvorovi služe kao mostovi s drugim područjima.

• Polu-centralizirana arhitektura - Neki čvorovi služe kao serveri koji omogućavaju mrežnu komunikaciju.

Nedostatci p2p su zaštita i povjerljivost.

24. **Servisno orijentirane arhitekture**

Se temelje na ideji pružanja vanjskih usluga putem web servisa. Organizacije koje žele omogućiti drugim programima pristup njihovim informacijama definiraju i objavljuju sučelje web servisa. Razlike između servisnog i distribuiranog objektnog modela: Servise može nuditi bilo tko unutar ili van organizacije.

25. **Četiri generička tipa arhitektura(+primjena)**

Sustavi za obradu podatka, Sustavi za obradu događaja, Sustavi za obradu transakcija, Sustavi za obradu jezika

Generičke arhitekture se koristi kao polazište za dizajn arhitekture; kao osnova za podjelu radnih zadatka.

26. **Sustavi za obradu podataka**

Aplikacije za obradu podataka su aplikacije koje obrađuju podatke u velikim količinama bez korisnikovog upletanja tijekom same obrade. Arhitektura ovakvih aplikacija ima tri glavne komponente: ulaz, obradu i izlaz.

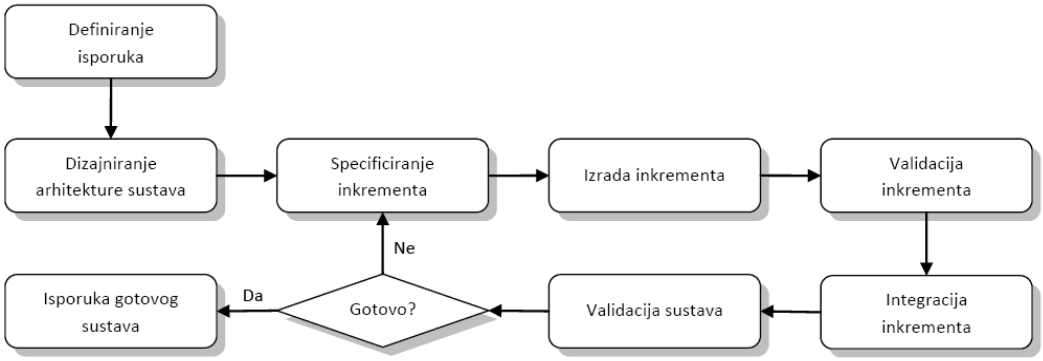
27. **Sustavi za obradu transakcija**

Sustavi za obradu transakcija su dizajnirani da obrađuju zahtjeve za informacijama i

zahtjeve za dodavanje informacija u bazu. To su obično interaktivni sustavi kod kojih korisnici asinkrono traže usluge.

28. **Sustavi za obradu događaja**
    Sustavi za obradu događaja odgovaraju na događaje ili iz okoliša sustava ili sa korisničkog sučelja. Glavna karakteristika ovakvih sustava je neodređenost trenutka javljanja događaja.
    Obično su jedno korisnički sustavi.
    Moraju izrazito brzo reagirati na korisničke naredbe što znači da moraju raditi na podacima koji se nalaze u memoriji.
29. **Sustavi za obradu jezika**
    Sustavi za obradu jezika kao ulaz prihvaćaju neki prirodni ili umjetni jezik, a kao izlaz generiraju neki drugi oblik tog istog jezika. Najrašireniji takvi sustavi su kompajleri.
30. **Faze objektno orijentiranog razvoja**
    Analiza, dizajn, programiranje
31. **Objektno-orijentirani dizajn**
    OO dizajn se bavi razvojem OO modela softverskog sustava koji implementira zahtjeve. OO sustavi se lakše mijenjaju, objekti se mogu modificirati kao samostalni entiteti, mijenjanje implementacije jednog objekta ne utječe na druge objekte (u teoriji)
32. **Proces objektno orijentiranog dizajna**
    Proces objektno orijentiranog dizajna ima niz isprepletenih faza:
    Razumijevanje i definiranje konteksta i načina korištenja sustava, Dizajniranje arhitekture sustava, Identifikacija osnovnih objekata sustava, Razvoj modela dizajna, Specificiranje sučelja objekata
33. **Okolina sustava (system context) i modeli korištenja (models of use)**
    Predstavljaju dva nadopunjavajuča modela odnosa sustava i okoline. Okolina sustava je statički model koji opisuje okolinu sustava i može se prikazati koristeći udruživanje („associations"). Model korištenja sustava je dinamički model koji opisuje kako sustav međudjeluje s okolinom.
34. **Identifikacija objekata**
    Najteži dio objektno orijentiranog dizajna.
    • Koristeći gramatičku analizu opisa sustava, • Koristeći opipljive entitete, • Koristeći pristup kod kojeg dizajner prvo razumije ponašanje sustava., • Koristeći analizu temeljenu na scenarijima
35. **Modeli dizajna**
    Pokazuju objekte i klase sustava, kao i odnose između entiteta. Razlikujemo dva tipa:
    Statički modeli - opisuju statičku strukturu sustava koristeći klase i odnose između klasa. Dokumentiraju se važni odnosi kao npr: nasljeđivanje, korištenje.
    Dinamički modeli - opisuju dinamičku strukturu i međudjelovanje objekata sustava. Dokumentira se slijed zahtjeva za uslugama, način na koji je stanje sustava povezano s međudjelovanjem sustava.
36. **Model poretka (Sequence model)**
    Dinamički model koji za svaki način međudjelovanja pokazuje interakciju objekata. Objekti koji međudjeluju su poredani horizontalno. Svaki objekt je povezan s vertikalnom linijom. Vrijeme je prikazano vertikalno (od gore prema dolje).

37. **Zašto je došlo do razvoja metoda rapid softvvare development,karakt.?**
Zbog stalnih promjena okruženja, često je nemoguće doći do stabilnih, konzistentnih korisničkih zahtjeva. Zbog toga je model vodopada (,,waterfall model") nepraktičan i potreban je pristup razvoju temeljen na iterativnoj specifikaciji i dostavi softvera. Procesi ubrzanog razvoja softvera su napravljeni tako da omoguće što bržu proizvodnju korisnog softvera. Svi pristupi ubrzanom razvoju dijele zajedničke karakteristike:
• Procesi specifikacije, dizajna i implementacije se odvijaju istovremeno., • Sustav se razvija u nizu inkremenata., • Koriste interaktivne sustave za razvoj koji omogućavaju
38. **Prednosti i mane iterativnog razvoja. (2,4)**
Glavne prednosti iterativnog razvoja softvera su: • Ubrzana isporuka, • Uključivanje korisnika
Glavni problemi kod iterativnog razvoja su: • Problemi s upravljanjem, • Problemi s ugovorima, • Problemi s validacijom, • Problemi s održavanjem
39. **Nacrtaj sliku modela iterativnog razvoja**



40. **Kod kojih aplikacija se ne koristi ubrzani razvoj?**
Postoje sustavi kod kojih iterativni razvoj nije najbolje rješenje, npr. kod velikih sustava koji uključuje rad različitih timova na različitim mjestima, kod kritičnih sustava, kod sustava koji ovise o razvoju posebnog hardvera. Koristi se hibridni proces kod kojeg se razvija prototip sustava koji se koristi za eksperimentiranje sa zahtjevima i dizajnom.
41. **Kod kojih aplikacija se ne koriste agilni metode?**
Agilne metode se ne koriste kod razvoja velikih sustava kod kojih ima više razvojnih timova na različitim mjestima, gdje postoji kompleksna interakcija s drugim hardverskim i softverskim sustavima. Agilne metode se ne bi trebale koristit za razvoj kritičkih sustava gdje je potrebna detaljna analiza sistemskih zahtjeva da bi se razumjeli utjecaji na sigurnost.
42. **Zašto nije toliko bitno da se proizvod isporuči bez grešaka?**
Prioritet ubrzanog razvoja softvera je brzina isporuke, a ne eventualne greške. Bitno je da klijent što prije dobije najvažnije funkcionalnosti. Eventualne greške se ispravljaju u sljedećim iteracijama.
43. **Principi agilnih metoda.**
Sudjelovanje korisnika, Isporuka inkremenata, Važni su ljudi, a ne procesi, Prihvaćanje promjena, Održavanje jednostavnosti
44. **Zašto su se počele koristi agilne metode i nabroji neke.**
Agilne metode omogućavaju razvojnom timu da se fokusira na sam softver, a ne na dizajn i dokumentaciju (često oduzimaju puno vremena). Temelje se na iterativnom razvoju i namijenjene su razvoju malih do srednjih poslovnih sustava kod kojih se zahtjevi brzo

mijenjaju. Neke agilne metode su: ekstremno programiranje, SCRUM, Crystal, Adaptive Software Development.

45. **Što je ekstremno programiranje?**
Ekstremno programiranje je jedna od najpoznatijih i najčešće korištenih agilnih metoda. Kod ekstremnog programiranja svi zahtjevi se izražavaju kao scenariji, koji se implementiraju kao nizovi zadataka. Programeri rade u parovima i razvijaju testove za sve zadatke prije nego što počnu s programiranjem.

46. **Principi ekstremnog programiranja.**
Inkrementalno planiranje, Mala izdanja, Jednostavan dizajn, Unaprijeđeni razvoj testova (test-first development), Programiranje u paru (pair-programming), Zajedničko vlasništvo nad kodom, Stalna integracija, Održivi ritam, Uključivanje klijenta

47. **Koje novosti uvodi ekstremno programiranje u proces testiranja.**
Ekstremnu programiranje slavlja veći naglasak na testiranje nego druge agilne metode. Ovakav pristup smanjuje vjerojatnost da će novi inkrementi uzrokovati nove greške. Glavne osobine testiranja su:
• Razvoj kod kojeg se prvo definiraju testovi (najvažnija inovacija XP-a) • Inkrementalni razvoj testova iz scenarija • Uključivanje korisnika u razvoj i validaciju testova • Automatsko provjeravanje testova

48. **Kako test-first development pomaže kod razumijevanja zahtjeva.**
Pisanje testova implicitno definira sučelje i ponašanje komponenti sustava. Smanjuju se problemi nerazumijevanja zahtjeva i sučelja.

49. **Koji su problemi kod test-first developmenta.**
Problemi: programerima je draže programiranje od testiranja, pa ponekad pišu nepotpune testove, neke testove je teško napisati, teško je procijeniti potpunost niza testova.

50. **Zašto je programiranje u paru dobro?**
Dva programera razvijaju softver na istom računalu. Ne radi se uvijek u istim parovima. Programiranje u paru ima niz prednosti: • Podržava ideju zajedničkog vlasništva i zajedničke odgovornosti. • Služi kao neslužbeni proces pregleda koda • Podržava refactoring.

51. **Prototipovi**
Prototip je početna verzija sustava i koristi se za prikaz koncepata, isprobavanje dizajna, otkrivanje i validaciju zahtjeva, testiranje.
Prototipovi omogućuju korisnicima saznanje o tome koliko sustav podržava njihov rad.
Prednosti: poboljšava se korisnost sustava, sustav više odgovara potrebama korisnika.
Mane: način korištenja prototipa možda ne odgovara korištenju sustava, vrijeme treniranja za korištenje prototipa može biti prekratko...

52. **Back to back testiranje**
Isti test se provodi i na sustavu i na prototipu. Ako oba sustava daju isti odgovor test najvjerojatnije nije naišao na pogrešku, a ako se rezultati razlikuju znači daje možda došlo do greške u sustavu.

53. Proces razvoja prototipa

54. **U kojim fazama razvoja se koriste prototipovi?**
Prototipovi se mogu koristiti u:
- Fazi specificiranja zahtjeva - pomaže u otkrivanju i validaciji zahtjeva

- Fazi dizajna - za istraživanje određenih rješenja i za razvoj Ul-a
- Fazi testiranja - za obavljanje back-to-back testova.

55. **Što je verifikacija, a što validacija?**
Verifikacija provjerava je li sustav napravljen prema specifikaciji (ispunjava li sve funkcionalne i nefunkcionalne zahtjeve). Validacija provjerava ispunjava li sustav korisnikova očekivanja. Glavni cilj V&V (verifikacije i validacije) je otkrivanje odgovara li sustav namjeni.

56. **Zašto je validacija teška?**
Zbog eventualnih korisnikovih nerealnih očekivanja,

57. **Koje su statičke a koje dinamičke metode V&V?**
Pregled softvera i automatizirane analize spadaju u statičke V&V tehnike. Statičke tehnike mogu samo provjeriti koliko softver odgovara specifikaciji. Mana je što ne mogu provjeriti koliko je softver stvarno koristan u rješavanju problema i zadovoljava li izvirujuća svojstva {emergentproperties). Testiranje je dinamička V&V tehnika.

58. **Razlika između V&V i debuggiranja**
Namjena V&V je utvrđivanje postojanja greški softverskog sustava. Debugging se bavi lociranjem i ispravljanjem tih pogrešaka.

59. **Testni plan (7)**
Planiranje testova se bavi utvrđivanjem standarda za testiranje procesa. Planiranje testova pomaže upraviteljima s alokacijom resursa i s predviđanjem rasporeda testiranja. Glavne komponente testnog plana su: • Testni proces - opis glavnih faza., • Praćenje zahtjeva - testiranje se planira tako da se svaki zahtjev posebno testira., • Testirane stavke — specificiraju se proizvodi softverskog procesa koji će se testirati. • Ograničenja - koja ograničenja utječu na testni proces (npr. nedostatak osoblja).

60. **Pregled softvera**
Pregled softvera je statički V&V proces kod kojeg se softverski sustav pregledava s ciljem pronalaženja grešaka i anomalija. Pregled se najčešće fokusira na kod ali se mogu pregledavati i zahtjevi i modeli dizajna.
Glavne prednosti pregleda nad testiranjem:
- Za vrijeme testiranja greške mogu prekrivati druge greške. Nakon otkrića greške ne može se biti siguran da li su greške u izlazu uzrokovane novom greškom ili su nuspojava originalne greške. Kod pregleda sustava mogu se otkriti mnoge greške sustava.
- Nepotpune verzije sustava se mogu pregledati bez dodatnih troškova.

61. **Proces pregleda programa**

62. **Uloge u procesu pregleda**
Autor ili vlasnik koda, inspektor (pronalazi pogreške, nedosljednosti u kodu ili dokumentu), čitač, pisar (zapisuje rezultate pregleda), moderator (upravlja procesom, odgovara glavnom moderatoru), glavni moderator (odgovoran za poboljšanja procesa pregleda, standarda razvoja...).

63. **Formalne metode**
Formalne metode se bave matematičkom analizom specifikacije, transformiranjem specifikacije u detaljniju.
Prednosti: izrada matematičke specifikacije zahtjeva detaljnu analizu zahtjeva i vrlo vjerojatnome otkriti pogreške, usporedbom programa i specifikacije mogu se otkriti pogreške u implementaciji prije testiranja.

Mane: zahtjeva posebnu notaciju koju ne mogu razumjeti stručnjaci iz područja kojim se bavi aplikacija

64. **Timovi**

• Specifikacijski tim — odgovorni za razvoj i održavanje specifikacije sustava. • Razvojni tim - odgovorni za razvoj i verifikaciju softvera. • Certifikacijski tim - odgovorni za razvoj statističkih testova.

65. **Razlika testiranja komoponenti i testiranja sustava(unit&system testing)**

Kod manjih sustava obično se koriste dvije osnovne testne aktivnosti:

• Testiranje komponenti - cilj je otkrivanje greški testiranjem pojedinih komponenti. Testovi se obično pišu na temelju iskustva programera i obično ih provode članovi razvojnog tima.

• Testiranje sustava - cilj je ustanoviti ispunjava li sustav funkcionalne i nefunkcionalne zahtjeve i ponaša li se nepredvidivo. Testiraju se grupe komponenti integrirane u podsustav ili sustav. Testiranje obavlja nezavisni testni tim.

66. **Defect testing**

Proces testiranja softvera ima dva cilja:

• Testiranje validacije - pokazati razvojnom timu i klijentu da softver ispunjava zahtjeve. Uspješan test pokazuje da sustav radi po specifikaciji.

• Testiranje pogrešaka {Defect testing) - otkriti pogreške u sustavu - traže se netočni proračuni, rušenja sustava... Test je uspješan ako pronađe pogrešku. Testiranje samo može pokazati da greške postoje (ne može dokazati da ne postoje).

67. **Testiranje sustava**

Testiranje sustava uključuje integriranje dviju ili više komponenti i provjeravanje integriranog sustava. Faze testiranja:

• Testiranje integracije - pronalaze se problemi i komponente koje treba debuggirati.
• Testiranje isporuka {release testing) - testira se verzija koja se može isporučiti.
• Testiranje performansi - sustav se tek nakon integracije može testirati na izvirujuća svojstva.. Često se koristi testiranje izdržljivosti (stress testing) - sustav se opterećuje iznad predviđenog opteredenja.

68. **Može li testiranje provjeriti da li sustav odgovara namjeni**

Glavni cilj V&V (verifikacije i validacije) je otkrivanje odgovara li sustav namjeni.

69. **Testiranje sučelja**

Testiranje složenih komponenti se prvenstveno bavi testiranjem sučelja pojedinih komponenti. Postoji više tipova sučelja:

• Parametarska sučelja, • Sučelja zajedničke memorije, • Proceduralna sučelja, • Sučelja za prijenos poruka

70. **Dizajn testova**

Cilj procesa dizajna testova je kreiranje skupa testova koji su učinkoviti u otkrivanju pogrešaka i dokazivanju da sustav zadovoljava specifikaciju. Prilikom dizajniranja testa, za neko svojstvo sustava odabire se skup ulaza i zabilježi se skup očekivanih izlaza.
Razlikuju se sljedeći pristupi:

• Testiranje temeljeno na zahtjevima - iz svakog zahtjeva se izvodi skup testova. Ovo testiranje spada u validaciju - dokazuje se da sustav ispunjava zahtjeve. • Testiranje dijelova (partition testing) • Strukturalno testiranje je pristup kod kojeg se testovi izvode

iz poznavanja strukture i implementacije sustava. • Testiranje putova (Path testing) - strategija kod koje je cilj provjeriti svaki nezavisni tok kroz sustav.

71. **Automatizacija testova**

Testiranje je skup proces i zbog toga su razvijeni brojni alati koji ga ubrzavaju.
Neki od testnih alata su:
• upravitelj testovima (prati podatke, očekivane rezultate), • generatori testnih podataka (mogu se koristiti ili baze podataka ili slučajno generirani podaci), • orade (predviđa rezultate testova, mogu biti ili prethodne verzije programa ili prototipovi), • file comparatorfaspoređuje rezultate programa s prethodnim testovima i označava razlike), • generator izvješća, dinamički analizator (prati koliko se puta koji dio koda izvrši), • simulator (npr. simulatori rada na korisničkom sučelju, simulatori određenih konfiguracija...).