1. Kodėl programinės įrangos inžinerijos kursas yra skaitomas universitetuose?
   Programinės įrangos projektai dažnai vėluodavo, kainuodavo daugiau negu buvo numatyta, neveikė taip kaip buvo tikimasi. Todėl buvo nutarta, kad programinė įranga turi būti projektuojama remiantis inžineriniais metodais ir tam skirti kursai turi būti skaitomi universitetuose.
2. Koks yra programinės įrangos inžinerijos tikslas?
   Nagrinėti visus programinės įrangos kūrimo aspektus ir visą programinės įrangos gyvavimo ciklą, kurti programinę įrangą mažiausiomis sąnaudomis. Programinės įrangos inžinieriai turėtų taikyti sisteminį ir organizuotą požiūrį į darbą, naudoti priemones ir metodus, priklausomai nuo sprendžiamos problemos, keliamų apribojimų ir turimų resursų.
3. Kas yra programinė įranga?
   Tai yra kompiuterių programos ir su jomis susijusi dokumentacija (dar duomenys). Programinės įrangos produktai gali būti kuriami konkrečiam vartotojui arba bendrai rinkai.
4. Kuo skiriasi programinės įrangos inžinerija (software engineering) ir kompiuterių mokslas (Computer science, Informatics)?
   Kompiuterių mokslas apima teoriją ir pagrindus. Programinės įrangos inžinerija apima praktinę programinės įrangos kūrimo ir naudojimo pusę. Kompiuterių mokslo teorijos nėra pakankamai išsamios, kad būtų laikomos tinkamu programinės įrangos inžinerijos pagrindu.
5. Kas yra modelis?
   Modelis – originalo, proceso atvaizdas, tapatus pasirinktu struktūros lygmeniu arba pasirinktomis funkcijomis. Sistema gali įtraukti daug modelių atspindinčius skirtingus požiūrius arba supaprastinimus. Modelis gali būti aprašymas, matematinės išraiškos, grafinis atvaizdavimas, imitacinė programa.
6. Koks skirtumas tarp „modeling“ ir „simuliation“?
   Modeling (modeliavimas) - tai modelio sudarymo procesas. Simulation – imitacinis modeliavimas (simuliavimas) tai objekto veiklos imitavimas, naudojant modelį, kuris gali būti kompiuterinė programa. Imitacinio modeliavimo metu pagal duotus duomenis yra suskaičiuojami rezultatai. Norint panaudoti rezultatus, reikia atsakyti į klausimą, ar modelis yra adekvatus realiam objektui.
7. Kas yra programinės įrangos kūrimo proceso modelis?
   Supaprastintas programinės įrangos proceso pristatymas iš tam tikros perspektyvos (tam tikru atžvilgiu). Perspektyvos: Darbų srautas – veiksmų seka. Duomenų srautas – tai informacijos judėjimas. Rolės/ veiksmo eiga – kas ką atlieka.
8. Kokie yra programinės įrangos inžinerijos metodai?
   Sisteminis požiūris į programinės įrangos kūrimą, kuris apima sistemos modelius (grafiniai modeliai, kurie turėtų būti padaryti), žymėjimus, taisykles (sistemos modeliams taikomi apribojimai), patarimus projektavimui (praktiniai gero projektavimo patarimai) ir vadovavimas procesui (kokius veiksmus, kokia tvarka atlikti).
9. Kokie yra geros programinės įrangos požymiai?
   Programinė įranga turėtų pasižymėti vartotojui reikiamu funkcionalumu ir greitaeigiškumu, turėtų būti tinkama eksploatuoti (prižiūrėti), turėtų kelti pasitikėjimą ir būti efektyvi bei tinkama naudoti.
10. Kokias esmines problemas tenka spręsti programavimo inžinerijoje?
    Reikia pajėgti susidoroti su pasenusiomis sistemomis, augančia sistemų įvairove bei realizavimo galimybėmis ir su reikalavimais trumpinti sistemų kūrimo laiką.
11. Kodėl taip ilgai užtrunka užbaigti programinės įrangos darbus?
    Kūrimo priemonės neveikia taip, kaip buvo tikėtasi, užsakovas reikalauja įvertinti naujus reikalavimus, dėl kurių tenka daug ką perdaryti, netikėtai pasikeičia valstybinis reguliavimas, suderinamumas su kitoms sistemomis reikalauja daugiau pastangų negu buvo tikėtasi, projekto rizikos valdymas reikalauja daugiau laiko negu buvo tikėtasi.

12. Kodėl programuotojams svarbi profesinė atsakomybė ir etika?
    Programinės įrangos inžinerija pasižymi didesne atsakomybe nei vien techninių gebėjimų pritaikymas, programinės įrangos inžinieriai privalo elgtis garbingai ir etiškai, jei jie nori būti gerbiami kaip profesionalai, etiškas elgesys yra daugiau nei tik paisyti įstatymų, nes įstatymais ne viską galima numatyti.
13. Kokie yra keturi profesinės atsakomybės aspektai?
    Programuotojai turėtų gerbti užsakovų ar klientų konfidencialumą, nepriklausomai nuo to, ar buvo pasirašytas oficialus konfidencialumo susitarimas. Programuotojai turi objektyviai vertinti savo kompetencijos lygį ir neturėtų sąmoningai imtis darbo, kuriam įgyvendinti jie stokoja kompetencijos. Programuotojai turi būti dėmesingi - užtikrinti, kad užsakovų ir klientų intelektinė nuosavybė yra apsaugota. Programinės įrangos inžinieriai neturėtų naudotis techniniais sugebėjimais, siekdami piktavališkai pasinaudoti kitų žmonių kompiuteriais.
14. Kas tai yra sistema?
    Tai tikslingas tarpusavyje susietų komponentų rinkinys, kurie veikia kartu, tam, kad pasiektų kokį nors bendrą tikslą. Sistema gali būti sudaryta iš programinės, mechaninės, elektrinės ir elektroninės aparatūrinės įrangos. Vienas iš sistemos elementų gali būti žmogus (socio-techninė sistema). Sistemos komponentai yra priklausomi vieni nuo kitų. Sistemos komponentų savybės bei elgsena yra neatskiriamai susiję.
15. Kodėl liktines sistemas brangu modifikuoti?
    Dažnai programuojama nenuosekliai, nes skirtingas dalis realizuoja skirtingos kūrėjų grupės. Sistema gali naudoti pasenusią programavimo kalbą. Sistemos dokumentacija dažnai yra pasenusi. Sistemos struktūra gali būti sugadinta metai iš metų vykdomų palaikymo procedūrų. Gali būti panaudoti veikimo spartos arba atmintinės taupymo optimizavimo metodai, dėl kurių suprantamumas suprastėja.
16. Kas yra programinės įrangos kūrimo procesas ir jo modelis?
    Kūrimo procesas - tai suderintų veiksmų seka, specifikuojant, projektuojant, realizuojant ir testuojant programinės įrangos sistemas. Programinės įrangos inžinerijos proceso modelis yra abstraktus proceso atvaizdavimas. Jis parodo darbų organizavimą. Programinės įrangos kokybę lemia procesas.
17. Kokie yra bendriausi programinės įrangos inžinerijos proceso modeliai?
    Krioklio modelis, kai yra aiškiai atskirtos specifikacijos ir kūrimo fazės. Evoliucinis kūrimas, kai specifikavimas ir kūrimas persidengia ir tobulinimas atliekamas palaipsniui. Formalus sistemų kūrimas, kai matematinis sistemos modelis formaliai transformuojamas į realizaciją. Kūrimas yra grindžiamas pakartotiniu panaudojimu, kai sistema yra sukomplektuojama iš jau egzistuojančių komponentų.
18. Kokie yra projektavimo proceso veiksmai?
    Projektavimo procesą sudaro architektūrinis projektavimas, abstraktus specifikavimas, vartotojo sąsajos projektavimas, komponentų projektavimas, duomenų struktūros projektavimas, algoritmų projektavimas.
19. Kaip klasifikuojamos programinės įrangos darbų automatizavimo priemonės?
    Įrankiai (tools), palaiko individualias proceso užduotis, tokias kaip projektavimo suderinamumo tikrinimas, teksto redagavimas ir kt. Paketai (workbench) palaiko visą proceso fazę, tokią kaip specifikacija ar projektavimas; paprastai apima kelis suderintus įrankius. Aplinkos palaiko visą programinės įrangos kūrimo procesą ir paprastai susideda iš kelių suderintų paketų.
20. Ką skelbia lanksčių (agile) metodų manifestas?
    Individai ir bendravimas yra svarbiau negu procesai ir įrankiai. Veikianti programinė įranga yra svarbiau negu išsami dokumentacija. Bendravimas su užsakovu yra svarbesnis nei derybos dėl kontrakto. Atlikti pakeitimus yra svarbiau nei juos planuoti.
21. Kokie lanksčių (agile) metodų principai?
    Užsakovai turi būti įtraukti į kūrimo procesą. Programinės įrangos kūrimas susideda iš etapų, kuomet užsakovas apibrėžia reikalavimus kiekvienam žingsniui. Kūrimo komandos sugebėjimai turi būti pripažinti ir išnaudoti, reikia leisti komandos nariams dirbti savo būdu, neatsižvelgiant į priskirtą procesą. Turi būti tikimasi, kad sistemos reikalavimai

keisis, tad sistema projektuojama, numatant keitimus. Reikia koncentruotis į kūrimo proceso ir kuriamos programinės įrangos paprastumą, visur kur tik galima stengtis eliminuoti sistemos sudėtingumą.

22. Kokia ekstremalaus programavimo praktika?
    Pažingsninis planavimas. Mažos versijos. Pirmiausia testavimas. Visi kūrėjai turėtų perdaryti kodą nuolat ir kuo greičiau, vos radus kodo patobulinimus. Kūrėjai dirba poromis, tikrindami vienas kito darbą ir teikdami paramą, kad darbas būtų visada gerai atliktas. Kūrėjų poros dirba visose sistemos srityse, norint išvengti vadinamųjų „žinių salų“. Visi kūrėjai prisiima atsakomybę už visą kodą, kiekvienas gali bet ką pakeisti. Kai tik užduotis baigiama, atliktas darbas yra integruojamas į visą sistemą ir kuomet tokia integracija įvyksta, visi sistemos vienetų testavimai turi būti atlikti. Dideli viršvalandžiai nėra leistini, nes galutinis rezultatas yra sumažėjusi kodo kokybė ir dirbančiojo darbo našumas. Kūrimo komanda turi turėti galimybę bet kuriame etape kilus klausimų susisiekti su galutinio naudotojo sistemos (užsakovo) atstovu.
23. Kuo pasižymi testavimas ekstremaliam programavime?
    Tekstai rašomi prieš įgyvendinant kodą. Testai rašomi kaip programos, o ne kaip duomenys, kad būtų galima juos įvykdyti automatiškai ir patikrinti ar jie įvykdyti korektiškai. Pažingsninis testo kūrimas iš scenarijaus. Vartotojai įtraukiami į testų sudarymo ir atestavimo procesus. Visi ankstesni ir nauji testai paleidžiami automatiškai, kai yra pridedama nauja funkcija, kad būtų patikrinta, ar nauja funkcija neįnešė klaidų.
24. Kas tai yra kritinės sistemos?
    Kritinės yra tos sistemos, kai sutrikus sistemos darbui gali žūti žmonės, būti užteršta aplinka arba patirti didžiuliai ekonominiai nuostoliai.
25. Kodėl apie kritines sistemas plačiai kalbama programavimo inžinerijoje?
    Vis daugiau sistemų atsakingų mazgų realizuoja programinė įranga. Kritinėms sistemoms realizuoti reikia taikyti specialius metodus. Reikia ugdyti atsakomybės už projektavimą jausmą, nes lanksti programinė įranga gali suklaidinti dėl galimybės pasitaisyti.
26. Ką charakterizuoja sistemų pasikliautinumas?
    Sistemos pasikliautinumas atspindi vartotojų pasitikėjimo sistema laipsnį. Šis laipsnis priklauso nuo to, ar sistema dirbs taip, kaip tikisi vartotojas, ar ji veiks be klaidų naudojant ją įprastai.
27. Kokias savybes sujungia pasikliautinumas?
    Parengtumas, patikimumas, sauga ir apsauga.
28. Kuo skiriasi sauga nuo apsaugos?
    Sauga (safety) yra sistemos savybė, kuri atspindi sistemos galimybę veikti (normaliai arba nenormaliai), nesukeliant pavojaus ir žalos žmonių sveikatai, nesibaigtų asmens mirtimi, nedarytų neigiamos įtakos aplinkai. Beveik visi nelaimingi atsitikimai įvyksta dėl kombinacijų, kurių visų neįmanoma įvertinti, sutrikimų. Apsauga (security) yra sistemos savybė, kuri atspindi sistemos sugebėjimą apsiginti nuo atsitiktinių ar apgalvotų išorinių užpuolimų.
29. Kokia defektų ir sutrikimų įtaka?
    Sutrikimai paprastai įvyksta dėl sistemos klaidų, kurias sukelia sistemos defektai. Tačiau defektai nebūtinai sąlygoja sistemos klaidas (defektuota būsena trumpalaikė, nespėja iššaukti klaidos). Klaidos nebūtinai lemia sistemos sutrikimus (klaida gali būti ištaisyta).
30. Kaip pasiekiamas norimas pasikliautinumas ir kas daro įtaką kaštų augimui didinant pasikliautinumą?
    Norint pasiekti reikiamą pasikliautinumą, reikia siekti vengti klaidų, jas nustatyti ir ištaisyti bei apriboti sutrikimo daromą žalą. Pasikliautinumo didinimo kaštai auga eksponentiškai. Tam daro įtaką brangesnių, labiau išvystytų technologijų naudojimas ir aparatūrinės įrangos bei padidintas testavimas ir sistemos atestavimas (validavimas), kurie reikalingi siekiant įtikinti sistemos klientus dėl pasikliautinumo.
31. Kodėl svarbu susipažinti su programinės įrangos projektų valdymu?
    Programinės įrangos projektai turi išskirtinių savybių, taip pat kam nors gali tekti dirbti tokį darbą. Tai būtina norint suprasti projekto vadovų priimamus sprendimus.

32. Kuo išsiskiria programinės įrangos projektų valdymas?
    Produktas yra neapčiuopiamas ir unikaliai lankstus. Programinės įrangos kūrimo procesas yra nestandartizuotas. Nevizualios (nematomos) programinės įrangos dalys kelia problemų valdymui.
33. Kam reikalingas programinės įrangos projekto valdymas?
    Kad kokybiška programinė įranga būtų sukuriama laiku ir pagal tvarkaraštį, kad būtų laikomasi biudžeto apribojimų. Geras vadovavimas projektui yra pagrindinis faktorius, lemiantis projekto sėkmę.
34. Kokias veiklas vykdo programinės įrangos projekto vadovas?
    Rašo pasiūlymus, planuoja projektą ir sudaro darbų tvarkaraštį, numato projekto išlaidas. Jis taip pat atsakingas už projekto stebėjimą ir peržiūrą, personalo atranką ir įvertinimą. Vedovas rengia ir pristato ataskaitas.
35. Kas būdinga programinės įrangos projekto planavimui?
    Tai tęstinė veikla nuo pradinės koncepcijos iki sistemos pateikimo. Planai privalo būti reguliariai atnaujinami, kai tik nauja informacija tampa prieinama. Projekto atskaitos taškai bei pateiktys turi būti organizuoti taip, kad būtų pristatyti apčiuopiami rezultatai, leidžiantys įvertinti progresą.
36. Ką reikia daryti, sudarant projekto darbų tvarkaraštį?
    Suskirstyti projektą į užduotis ir apskaičiuoti laiką ir resursus, reikalingus kiekvienai užduočiai užbaigti. Organizuoti, kad užduotys būtų vykdomos lygiagrečiai, siekiant, optimaliai panaudoti darbo jėgą. Privalu minimizuoti užduočių priklausomybes tam, kad būtų išvengta uždelsimų, atsiradusių vienai užduočiai laukiant kitos pabaigos. Reikia atsižvelgti į galimas problemas ir kad produktyvumas nėra proporcingas skaičiui žmonių, dirbančių su užduotimi.
37. Ką rodo kalendorinės diagramos ir tinklinis darbų grafikas?
    Tinklinis grafikas parodo projekto suskirstymą į užduotis, užduočių tarpusavio priklausomumą ir kritinius kelius. Kalendorinės diagramos rodo darbų tvarkaraštį kalendorinio laiko atžvilgiu.
38. Kas tai yra programinės įrangos projekto rizika ir jos valdymas?
    Rizika – tai tikimybė, kad kils kažkokių nepalankių aplinkybių. Rizikos valdymas **–** tai rizikos įvertinimas ir kūrimas planų, kurie minimizuotų rizikos poveikį projektui. Projekto rizika daro įtaką tvarkaraščiui arba resursams. Produkto rizika daro įtaką kokybei arba programinės įrangos kūrimo našumui.
39. Kaip suprantami reikalavimai?
    Tai gali būti aprašymai (nuo labai abstrakčių teiginių iki detalizuotų matematinių funkcijų). Reikalavimai gali atlikti dvigubą funkciją, gali būti kaip pagrindas pasiūlymui (konkursui),todėl turi būti atviri interpretacijai ir sudaryti pagrindą pačiai užsakymo sutarčiai – taigi reikalavimai turi būti aprašyti gana smulkiai.
40. Kuo skiriasi reikalavimas nuo tikslo?
    Reikalavimas turi būti objektyviai išmatuojamas, kad būtų įsitikinta, ar jis įgyvendinamas. Jeigu teiginis nepatikrinamas, jis negali būti reikalavimas. Tikslas atspindi tik vartotojų ketinimus. Tikslai taip pat naudingi kūrėjams.
41. Kokios problemos kyla užrašant vartotojo reikalavimus naturalia kalba?
    Sunku pasiekti tikslumą kad nebūtų sunku dokumentą skaityti. Funkciniai ir nefunkciniai reikalavimai turi tendenciją susimaišyti. Keletas skirtingų reikalavimų gali būti išreikšti kartu. Sunku išvengti dviprasmiškumo. Natūrali kalba netinka struktūrizuoti reikalavimus.
42. Kuo skiriasi funkciniai ir nefunkciniai reikalavimai?
    Funkciniai reikalavimai - tai sistemos paslaugų aprašymai, kurie dar turėtų paaiškinti, kaip sistema turėtų reaguoti į ypatingai įvedamus duomenis ir kaip sistema elgsis ypatingose situacijose. Nefunkciniai reikalavimai - tai sistemos paslaugų arba funkcijų apribojimai, tokie kaip laiko apribojimai, kūrimo proceso apribojimai, standartai, ir pan.
43. Kas atliekama reikalavimų inžinerijos proceso metu?
    Reikalavimų inžinerijos procesas apima galimybių tyrimą, reikalavimų išgavimą ir analizę, reikalavimų specifikavimą ir reikalavimų vadybą.

44. Kokios pakopos sudaro reikalavimų analizę?
    Reikalavimų analizę sudaro šios periodiškai pasikartojančios pakopos: srities supratimas, reikalavimų surinkimas, klasifikavimas, struktūrizavimas, prioritetų išskyrimas ir atestavimas.
45. Kokios yra reikalavimų analizės problemos?
    Suinteresuoti asmenys dažnai nežino, ko jie iš tiesų nori. Suinteresuoti asmenys išreiškia savo reikalavimus savais terminais. Skirtingi suinteresuoti asmenys gali pateikti tarpusavyje nesuderinamų reikalavimų. Organizaciniai ir politiniai faktoriai gali daryti įtaką reikalavimams, keliamiems sistemai. Reikalavimai keičiasi analizės proceso metu. Gali atsirasti naujų suinteresuotų asmenų, ir tai pakeis verslo aplinką.
46. Kokie principai taikomi struktūriniams sistemos modeliams sudaryti?
    Padalinimas (dekompozicija) - identifikuoja struktūrinius ryšius tarp esybių. Apibendrinimas (abstrakcija) -identifikuoja bendrumus tarp esybių. Projekcija (perspektyva) - identifikuoja skirtingus požiūrius į problemą.
47. Kas tikrinama reikalavimų peržiūros metu?
    Patikrinamumas. Ar reikalavimą tikrai galima patikrinti (testuoti)? *Išsamumas*. Ar reikalavimai suprasti teisingai? *Sekamumas*. Ar reikalavimų kilmė yra aiškiai suformuluota? *Prisitaikomumas*. Ar gali reikalavimas pasikeisti be didelio poveikio kitiems reikalavimams?
48. Kuo naudingos UML diagramos ?
    UML standartizuoja grafinius žymėjimus. Patogi priemonė aptarti programinės įrangos reikalavimus. Kaip ir formalios specifikacijos mažina dviprasmiško interpretavimo galimybes. Sudaro galimybes aptikti prieštaravimus tarp modelių. Sudaro prielaidas automatiškai transformuoti modelius į veikiančią programą
49. Kas tai yra modelis ir modeliavimas ?
    Modelis – tai sistemos atvaizdas iš vienos perspektyvos, leidžiantis geriau suvokti visą sistemą. Sistema gali turėti daug modelių.
    Modeliavimas – tai sudarymas modelių rinkinio, leidžiančio suvokti ir nagrinėti sistemą įvairiais aspektais.
50. Kas tai yra - imitacinis modeliavimas ?
    Imitacinis modeliavimas (simulation) – tai sistemos elgsenos imitavimas leidžiantis pasinaudoti modeliavimo rezultatais. Imitacinio modeliavimo atveju reikia atsakyti į klausimą ar modelis adekvatus realiam objektui. Prototipas tai yra supaprastintas imitacinis modelis
51. Kas būdinga UML diagramoms?
    Diagramos gali būti skirtingos paskirties ir skirtingo detalumo lygio. Mokantis ir bendraujant svarbiausia abstraktus mąstymas ir dėmesys reikiamoms detalėms, nekreipiant dėmesio į tuo momentu nesvarbius dalykus. Priklausomai nuo sistemos sudėtingumo ir prigimties, naudojami skirtingi modeliai ir diagramos.
52. Kaip gaunamas ir kam reikalingas programinės įrangos architektūrinis projektas?
    Architektūrinis projektas paprastai išreiškiamas kaip blokinė diagrama, vaizduojanti bendrą sistemos struktūrą. Projektavimas apima sistemos padalinimą į bendraujančias posistemes. Labiau specifiniai modeliai rodo, kaip posistemės dalinasi duomenimis ir kokias jų tarpusavio sąsajos. Architektūra gali būti panaudota suinteresuotiems asmenims (stakeholders) komunikuoti. Ta pati architektūra gali būti pritaikyta daugeliui sistemų.
53. Kokias sistemos charakteristikas lemia architektūra?
    Sistemos eksploatacinės savybes, apsaugą, saugą ir parengtumą.
54. Kokie naudojami būdai sistemai organizuoti?
    Bendros duomenų saugyklos, bendrų paslaugų ir serverių, abstraktaus automato arba sluoksniavimo būdai.
55. Kokie yra sistemų modulinio skaldymo būdai?
    Objektinis būdas, kai sistema susideda iš sąveikaujančių objektų ir duomenų srautų; būdas, kai sistema skaldoma į funkcinius modulius, kurie transformuoja įėjimus į išėjimus.

56. Kokie yra būdai programinės įrangos sistemai valdyti?
    Centralizuotas valdymas - kai viena posistemė atsakinga už kontrolę ir startuoja bei stabdo kitas posistemes, bei įvykiais paremtas valdymas, kai kiekviena posistemė savarankiškai atsako į įvykius iš kitų posistemių arba sistemos aplinkos.
57. Kokie yra paskirstytų sistemų privalumai ir trūkumai?
    Privalumai - dalijimasis resursais, atvirumas, lygiagretiškumas, išplečiamumas, klaidų toleravimas, skaidrumas. Trūkumai – sudėtingumas, saugos užtikrinimas, valdymo problemos, nenuspėjamumas.
58. Kokios lygiagrečių skaičiavimų galimybės?
    Procesorių greitaeigos didinimo galimybės beveik išnaudos, todėl plačiai naudojami lygiagrečių skaičiavimų klasteriai procesų bendravimui užtikrinti, klasterių gridai, virtualizavimas, debesų kompiuterija.
59. Kokius sluoksnius naudoja programų sistemų sluoksniuota architektūra?
    Atvaizdavimo sluoksnis, skaičiavimų vykdymo sluoksnis, duomenų valdymo sluoksnis.
60. Kuo skiriasi paskirstytų sistemų lengvo ir sunkaus kliento modeliai?
    Lengvo kliento modelyje visų programų vykdymas ir duomenų valdymas vyksta serveryje. Klientas yra atsakingas tik už atvaizdavimo programinės įrangos veikimą. Sunkaus kliento modelyje serveris yra atsakingas tik už duomenų valdymą. Kliento programinė įranga įgyvendina taikymus ir sąveiką su sistemos vartotoju.
61. Kas būdinga paskirstytų objektų architektūrai?
    Paskirstytų objektų architektūroje tarp klientų ir serverių nėra skirtumų. Kiekviena paskirstoma esybė yra objektas, kuris tiekia paslaugas kitiems objektams ir priima paslaugas iš kitų objektų. Objektai bendrauja per tarpines programines priemones, vadinamas objekto užklausos tarpininku (programinės įrangos magistralė), tačiau juos sudėtingiau projektuoti nei kliento /serverio sistemas.
62. Kokie yra internetinių paslaugų privalumai?
    Nepriklausomas tiekimas, viešas informavimas apie teikiamas paslaugas, naujų paslaugų formavimas komponavimo būdu, galimybė sumokėti už paslaugas, centralizuota paslaugos priežiūra ir pritaikymas, teikiamos paslaugos bet kokioje platformoje ir parašytos bet kuria programavimo kalba.
63. Kaip apibūdinamos realaus laiko sistemos?
    Tai sistemos, kurios stebi ir valdo savo aplinką. Šios sistemos neišvengiamai susijusios su aparatine įranga. Jutikliai (sensors) surenka duomenis iš sistemos aplinkos. Judikliai (actuators) pakeičia (tam tikru būdu) sistemos aplinką. Laikas tokioms sistemoms yra esminis kriterijus. Realaus laiko sistemos PRIVALO reaguoti per tam tikrą laiką.
64. Kas būdinga realaus laiko sistemų projektavimui?
    Atskirai projektuojama aparatinė ir programinė įrangos susijusios su sistema, priskiriant funkcijas aparatinei ir programinei įrangai. Projektuojama atsižvelgiant į nefunkcinius sistemos reikalavimus. Aparatinė įranga veikia greičau, bet jos kūrimas trunka ilgiau ir yra mažiau galimybių ją pakeisti.
65. Kaip skiriasi aparatinės ir programinės įrangos projektavimas?
    Aparatūros ir programinės įrangos funkcijas aprašo panašios algoritminės kalbos. Sukompiliuotą programą galima iš karto vykdyti, kai tuo tarpu aparatūros kompiliatorius paruošia gamybos šablonus ir aparatūros gamyba užtrunka tam tikrą laiką. Egzistuoja programuojamos matricos, kur aparatūra betarpiškai konfiguruojama vykdymui, bet šiuo atveju sumažėja aparatūros greitaeiga, bet jis būna didesnis negu programinės realizacijos atveju.
66. Kokie etapai sudaro realaus laiko sistemų projektavimą?
    Reikia nustatyti apdorojamą poveikį ir įvardyti reakcijas į jį, taip pat nustatyti laiko apribojimus kiekvienam poveikiui ir reakcijai. Privalu apjungti poveikio ir reakcijos apdorojimą į lygiagrečius procesus, susijusius su kiekviena poveikio ir reakcijos klase, suprojektuoti algoritmus kiekvienai poveikio ir reakcijos klasei apdoroti, nustatant laiko apribojimus. Reikia suprojektuoti planavimo sistemą, kuri užtikrintų, kad proceso vykdymo pradžia būtų parinkta taip, jog būtų spėta juos užbaigti iki nustatyto termino. Procesai turi būti apjungti naudojant realaus laiko vykdiklius ir operacinę sistemą.

67. Kaip įgyvendinami laikui keliami apribojimai, projektuojant realaus laiko sistemas?
    Gali prireikti plataus modeliavimo ir eksperimentavimo, kad būtų užtikrinta, jog sistema atitinka laiko apribojimus. Tai gali reikšti, kad tam tikros projektavimo strategijos, tokios kaip objektiškai orientuotas projektavimas, negali būti naudojamos dėl greičiui keliamų reikalavimų. Dėl šios priežasties gali prireikti vykdymą realizuoti žemo lygio programavimo kalba.
68. Kas tai yra realaus laiko vykdikliai?
    Realaus laiko vykdikliai yra specializuotos operacinės sistemos, kurios valdo realaus laiko sistemų procesus. Vykdikliai atsakingi už procesų valdymą ir resursų (procesoriaus ir atminties) paskirstymą. Gali būti sudaryti pagal standartinį realaus laiko vykdiklių branduolį, kuris naudojamas nepakeistas arba modifikuotas pagal konkrečią taikomąją programą. Nepalaiko failų valdymo.
69. Kas tai yra duomenų surinkimo sistemos?
    Šios sistemos surenka duomenis iš jutiklių tolimesniam apdorojimui ir analizei. Duomenų surinkimo ir apdorojimo procesai gali apimti skirtingus periodus ir terminus (deadlines). Duomenų surinkimas gali būti greitesnis negu jų apdorojimas, pvz., informacijos surinkimas apie sprogimą. Ciklinis arba žiedinis buferis yra mechanizmas, suvienodinantis greičio skirtumus.
70. Kodėl svarbi vartotojo sąsaja?
    Sistemos vartotojas dažnai sprendžia apie sistemą pagal jos sąsają, o ne pagal sistemos funkcionalumą. Prastai suprojektuota sąsaja gali būti katastrofiškų vartotojo klaidų priežastimi. Nekokybiška vartotojo sąsaja yra pagrindinė priežastis, dėl kurios daugelis programinės įrangos sistemų yra nenaudojamos.
71. Kokie vartotojo sąsajos pagrindiniai reikalavimai?
    Projektavimas turi būti orientuotas į vartotoją, t.y. vartotojo poreikiai yra svarbiausi ir vartotojas yra įtraukiamas į projektavimo procesą. Vartotojo sąsajos projektavimas turi vertinti sistemos vartotojo poreikius, patirtį ir sugebėjimus. Projektuotojai turi žinoti žmonių fizinius ir mentalinius (protinius) apribojimus (pvz. ribota trumpalaikė atmintis) ir turi suprasti, kad žmonės daro klaidų.
72. Kokie yra vartotojo sąsajos projektavimo principai?
    Sąsaja turi būti pagrįsta vartotojo terminais ir koncepcijomis, o ne kompiuterinėmis koncepcijomis. Komandos ir meniu būti to paties formato, komandų skyryba turi būti panaši ir t.t. Jei komanda veikia žinomu būdu, vartotojas turi sugebėti iš anksto nustatyti panašios komandos veiksmus. Sistema turi būti tampri (atspari) vartotojo klaidoms ir turi leisti vartotojui ištaisyti klaidas. Turi būti pateikiami vartotojo gidai tokie kaip pagalbos sistemos, on-line vartotojo vadovai ir t.t. Turi būti skirtingos sąsajos skirtingiems vartotojų tipams.
73. Kokios rekomendacijos dėl spalvų naudojimo vartotojų sąsajose?
    Nenaudoti per daug spalvų. Taikyti spalvų kodavimą ir leisti vartotojui jį keisti. Projektuoti, naudojant vieną spalvą ir tik tada pridėti kitas spalvas, spalvinį kodavimą naudoti nuosekliai. Vengti spalvų porų, kurios vargina akis. Naudoti spalvos pasikeitimus pasikeitusiai būsenai parodyti.
74. Kaip apibūdinamas sistemos klaidų pranešimų projektavimas?
    Klaidų pranešimų projektavimas yra labai svarbus. Silpni klaidų pranešimai gali reikšti, kad vartotojas greičiau atmes sistemą negu priims. Pranešimai turi būti mandagūs, glausti, nuoseklūs ir konstruktyvūs. Vartotojo patirtis ir įgūdžių lygis turi būti lemiami faktoriai projektuojant pranešimus.
75. Kokie programinės įrangos be defektų kūrimo požymiai?
    Reikalinga tiksli (pageidautina formali) specifikacija. Reikalingos organizacinės priemonės kokybei pasiekti. Informacijos slėpimas ir inkapsuliacija yra būtini dalykai projektuojant programinę įrangą . Turėtų būti naudojama programavimo kalba su griežta kintamųjų tipų kontrole ir klaidų tikrinimu, kuomet rašomas kodas tikrinantis galimas būsenas. Turėtų būti vengiama klaidas sąlygojančių struktūrų. Patikimas ir kartotinis kūrimo procesas.

76. Kokios pasikliautino programinės įrangos kūrimo proceso charakteristikos?
    Procesas turi būti gerai dokumentuotas, patvirtintas audito ir atsparus individualioms klaidoms, turi būti išsamūs programinės įrangos kūrimo standartai ir naudojamas detalus verifikavimas bei testavimas.
77. Kokios yra klaidas sąlygojančios struktūros?
    Slankaus taško skaičiai, rodyklės, dinaminis atminties paskirstymas, lygiagretūs skaičiavimai, rekursija, pertraukimai, paveldimumas. Šių struktūrų neturėtų būti vengiama, bet jas naudoti reikėtų labai atidžiai.
78. Kas tai yra gynybinis programavimas?
    Programuotojai įtaria, kad gali būti defektų sistemos kode, todėl įdeda į jį perteklinį kodą, tikrinantį sistemos būseną skaičiavimo metu.
79. Kaip užtikrinamas aparatūrinės įrangos tolerantiškumas defektams?
    Yra naudojamos 3 identiškos komponentų kopijos, kurios priima tokius pačius duomenis ir kurių rezultatai yra palyginami. Jeigu vienas iš trijų rezultatų yra kitoks, jis yra ignoruojamas, ir prieinama prie išvados, kad įvyko komponento trikis. Remiasi maža tikimybe, kad komponentai dėl gedimų funkcionuos neteisingai vienu metu.
80. Kaip užtikrinamas programinės įrangos tolerantiškumas klaidoms?
    Programinei įrangai svarbios klaidos, bet ne gedimai. Todėl galima remtis tik projektavimo įvairove, tikintis, kad skirtingose programinės įrangos versijose nebus tų pačių klaidų. Yra taikomas N-versijų programavimas, kai lyginami skirtingų versijų rezultatai ir teisingas rezultatas parenkamas daugumos balsavimu.
81. Kaip pasiekiama projektavimo įvairovė toleruojant programinės įrangos klaidas?
    Realizavimas skirtingomis programavimo kalbomis. Skirtingų įrankių ir vystymo aplinkų naudojimas. Realizacijai naudojami skirtingi algoritmai.
82. Kokių kyla problemų dėl projektavimo įvairovės, toleruojant programinės įrangos klaidas?
    Darbo grupės dažnai nėra kultūriškai skirtingos, mokėsi tuose pačiuose universitetuose, taigi jos linkusios spręsti problemas tuo pačiu būdu. Jei jau specifikacijoje yra klaida, tai matoma visose realizacijose. Tai gali būti išspręsta , naudojant įvairias specifikacijų reprezentacijas.
83. Ar visuomet programinės įrangos pertekliškumas yra tikslingas?
    Priešingai nei aparatinės įrangos atveju programinės įrangos klaidos nėra neišvengiamas realaus pasaulio padarinys. Todėl kai kurie žmonės mano, kad aukštesnis patikimumo ir tinkamumo lygis gali būti pasiektas, stengiantis mažinti programinės įrangos sudėtingumą. Perteklinė programinė įranga yra žymiai sudėtingesnė, taigi dėl klaidas toleruojančių kontrolerių atsiranda papildomų klaidų, kurios veikia sistemos patikimumą.
84. Kokie yra pakartotinio programinės įrangos panaudojimo privalumai ir trūkumai?
    Privalumai: didesnis patikimumas, mažesnė proceso rizika. Efektyvus specialistų gebėjimų panaudojimas. Atitikimas standartams. Greičiau kuriama programinė įranga. Trūkumai: didesni palaikymo kaštai, “čia neišrasta” sindromas, komponentų paieška ir pritaikymas.
85. Kaip apibūdinami programų generatoriai?
    Programų generatoriai pakartotinai naudoja standartinius šablonus ir algoritmus. Jie yra įterpti į generatorius ir parametrizuoti vartotojo komandose. Programa yra generuojama automatiškai. Generatoriais paremtas pakartotinis panaudojimas yra labai efektyvus kainos atžvilgiu, tačiau taikomumą apriboja nedidelis taikymo sričių skaičius. Galutiniam vartotojui lengviau sukurti programas, naudojant generatorius lyginant su komponentų pakartotiniu panaudojimu.
86. Kokias savybes turi turėti pakartotinio panaudojimo komponentai?
    Atspindėti stabilias srities abstrakcijas, paslėpti būsenos atvaizdavimą, būti kuo labiau nepriklausomi, skelbti išimtis per komponentų sąsają.
87. Koks ryšys tarp komponentų naudojamumo ir pakartotinio panaudojamumo?
    Kuo bendresnė sąsaja, tuo geresnis pakartotinas panaudojamumas, bet tada sąsaja sudėtingesnė, vadinasi, gali būti mažiau naudojama.

88. Kodėl naudojamą programinę įrangą neišvengiamai reikia keisti?
    Naudojant programinę įrangą iškyla naujų reikalavimų. Pasikeičia verslo aplinka. Tenka taisyti klaidas. Reikia panaudoti pasirodžiusios naujos aparatūros privalumus. Gali tekti gerinti vykdymo spartą arba patikimumą.
89. Kokie yra Lehmano dėsningumai?
    Realioje aplinkoje naudojama programa turi būti atnaujinama, nes priešingu atveju jos naudingumas toje aplinkoje vis mažės. Programą pastoviai atnaujinant, jos struktūra tampa vis sudėtingesnė. Norint, kad struktūra išliktų paprasta arba ją supaprastinti, reikia papildomų investicijų. Programos evoliucija yra savireguliuojantis procesas. Per sistemos gyvavimo laiką visų atnaujinimų apimtis būna panaši.
90. Kas tai yra programinės įrangos priežiūra?
    Programinės įrangos modifikavimas po to, kai ji buvo pateikta naudojimui. Palaikymo procedūrų metu dideli sistemos architektūros pakeitimai dažniausiai nevykdomi. Atnaujinimas realizuojamas arba modifikuojant jau egzistuojančius sistemos komponentus, arba pridedant naujus.
91. Kas apibūdina programinės įrangos palaikomumą?
    Prašymų taisyti klaidas skaičius. Vidutinis laikas, reikalingas poveikio analizei atlikti. Vidutinė pakeitimo realizavimo trukmė. Laukiančių prašymų atnaujinti skaičius. Jeigu vienas arba visi šie rodikliai auga, tai gali rodyti mažėjantį palaikomumą.
92. Kas tai yra sistemos perdarymo inžinerija?
    Perstruktūrizavimas arba perrašymas tik dalies arba visos sistemos, nekeičiant jos funkcionalumo. Pritaikoma ten, kur kelios ar visos didelės sistemos posistemės reikalauja dažnų pataisymų. Perdarymo inžinerija reikalauja pastangų kad sistema tamptų lengviau palaikoma. Sistemą gali reikėti perstruktūrizuoti arba pakeisti dokumentaciją.
93. Kokios yra programinės įrangos sistemos perdarymo proceso veiklos?
    Išeities kodo transliavimas į naują kalbą. Atstatymo inžinerija, apimanti programos analizę, siekiant ją suprasti. Automatinis išeities kodo struktūros gerinimas, siekiant padidinti suprantamumą. Programos struktūros reorganizavimas. Duomenų perdarymas ir restruktūrizavimas.
94. Kokios yra liktinių sistemų vystymo strategijos?
    Visiškai išmesti sistemą ir pakeisti verslo procesus, taip kad sistemos nebereikėtų, toliau palaikyti liktinę sistemą, sistemą perprojektuoti, kad pagerėtų jos palaikomumas, keisti sistemą nauja.
95. Kaip skiriasi patikra nuo atestavimo?
    Patikra nustato, ar programa atitinka specifikaciją, o atestavimas nustato, ar programa atitinka vartotojo norus.
96. Kuo skiriasi statinė ir dinaminė patikra?
    Statinė patikra tai programinės įrangos peržiūra, inspektavimas siekiant atskleisti anomalijas ir gal būt su tuo susjjusias klaidas. Statinės patikros metu programa nevykdoma ir gali būti analizuojama ne tik programa, bet ir dokumentacija, projektas, specifikacijos.
    Dinaminės patikra tai programinės įrangos testavimas susjjęs su programos vykdymu su parinktais duomenimis ir rezultatų stebėjimu siekiant nustatyti galimas klaidas.
97. Kuo skiriasi testavimas ir derinimas?
    Klaidų testavimas ir derinimas yra skirtingi procesai. Testavimas yra skirtas nustatyti klaidą programoje. Derinimas yra skirtas nustatyti klaidos vietą ir ją ištaisyti. Derinimo metu formuojamos hipotezės apie programos veikimą, po to, testuojant šias hipotezes, randamos sistemos klaidos.
98. Kas būdinga programinės įrangos peržiūrai?
    Siejasi su žmonėmis, nagrinėjančiais išeities tekstus, siekiant atrasti anomalijas ir klaidas. Nereikalauja vykdyti sistemos, todėl gali būti naudojama prieš realizaciją. Gali būti taikoma bet kokiems sistemos elementams (reikalavimams, projektui, testų duomenims ir t.t.).Tai labai efektyvi metodika klaidoms surasti.

99. Koks santykis tarp peržiūros ir testavimo?

Peržiūra ir testavimas yra vienas kitą papildantys ir neprieštaraujantys tikrinimo metodai. Abu turėtų būti naudojami kuriant programinę įrangą. Peržiūra padeda išsiaiškinti, kaip programa atitinka specifikaciją, bet ne realius užsakovų reikalavimus. Peržiūra negali patikrinti nefunkcinių charakteristikų (tokių kaip našumas, tinkamumas naudoti ir t.t.). Tai gali būti patikrinta tik testuojant.

100. Kokios išanktinės sąlygos keliamos peržiūrai?

Turi būti prieinama tiksli sistemos specifikacija. Peržiūros komandos nariai turi būti susipažinę su organizacijos standartais. Turi būti prieinamas sintaksiškai teisingas kodas. Turi būti paruoštas klaidų tikrinimo sąrašas. Vadovybė turi susitaikyti su tuo, kad peržiūra padidina kaštus ankstyvame programinės įrangos kūrimo etape. Vadovybė neturėtų naudoti peržiūros rezultatų personalui vertinti.

101. Kokia statinių analizatorių paskirtis?

Statiniai analizatoriai – programinės įrangos įrankiai išeities tekstui apdoroti. Jie išnagrinėja programos tekstą, bando surasti potencialiai klaidingas sąlygas ir atkreipti į jas peržiūros komandos dėmesį. Labai efektyvi pagalba peržiūrai. Priedas prie peržiūros, bet ne jos pakaitalas.

102. Kuo remiasi bedefektis programinės įrangos kūrimas?

Palaipsniniu kūrimu, formalia specifikacija, statiniu tikrinimu, naudojant korektiškumo argumentus, statistiniu testavimu, nustatant programos patikimumą.

103. Kodėl formalūs kūrimo metodai nepaplito taip plačiai kaip buvo tikėtasi?

Kiti programinės įrangos inžinerijos metodai sėkmingai pagerino sistemų kokybę. Todėl formalių metodų naudojimo poreikis sumažėjo. Rinka į pirmą vietą iškėlė kūrimo rinkai laiko trukmę (Time-to-market) prieš mažų klaidų kiekio faktorių. Formalūs metodai nesutrumpina kūrimo rinkai trukmės. Formalių metodų naudojimo sritys yra ribotos. Jie nelabai tinka vartotojo sąsajai ir bendravimui specifikuoti bei analizuoti. Formalūs metodai sunkiai išplečiami didelėms sistemoms.

104. Kas būdinga programų testavimui?

Testavimas gali parodyti klaidas bet ne jų nebuvimą. Sėkmingas testas – tas, kuris atskleidžia vieną ar daugiau klaidų. Testavimas yra vienintelis patikros ir atestavimo metodas, taikomas nefunkciniams reikalavimams. Programų testavimas turi būti panaudotas apjungiant kartu su statine patikra, kad būtų pilna patikros ir atestavimo apimtis.

105. Kam tenka atsakomybė ir kas vykdo komponentų ir integravimo testavimą?

Komponentus testuoja jų kūrėjai, o integravimo testavimą dažniausiai vykdo nepriklausoma testuotojų komanda.

106. Kaip gaunami testiniai atvejai ekvivalentinio sudalinimo metodu?

Įvedami duomenys ir išvedami rezultatai suskirstomi į atskiras klases, kur visi klasių nariai yra panašūs. Kiekviena iš šių klasių yra lygiaverčio suskirstymo rezultatas, kur programos elgesys su kiekvienu klasės nariu yra toks pat (ekvivalentiškas). Testiniai atvejai turi būti parinkti kiekvienai klasei.

107. Kokių testavimo nuorodų reikia masyvams?

Testuoti programinę įrangą, kai masyvas turi tik vieną elementą. Skirtingiems testams naudoti skirtingo dydžio masyvus. Parinkti testus taip, kad būtų nagrinėjamas pirmas, vidurinis ir paskutinis masyvo elementas. Testuoti nulinio ilgio masyvus.

108. Kokie naudojami kriterijai struktūriniam testavimui?

Testai bent vieną kartą turi vykdyti kiekvieną programos operatorių. Testai bent vieną kartą turi vykdyti kiekvieną skaičiavimų šaką. Testai bent vieną kartą turi vykdyti kiekvieną skaičiavimų kelią.

109. Ką rodo testavimo pilnumas?

Testavimo pilnumo matavimas rodo, kaip įgyvendintas testavimo kriterijus. Testavimo pilnumas gali būti išreikštas procentais. Testavimo kokybę rodo pasirinktas testavimo kriterijus ir jo išpildymo pilnumas. Testavimas negali garantuoti, kad bus surastos visos klaidos.

110. Kokie yra objektų testavimo lygiai?

Testuoti operacijas susietas su objektais. Testuoti objektų klases. Testuoti bendradarbiaujančių objektų grupes. Testuoti užbaigtą objektinę sistemą.

111. Kodėl reikalingas automatinis testavimas?

Kuriant programą, ją derinant, užtikrinant priežiūrą tenka tą pačią programą ir tuos pačius komponentus testuoti daug kartų po kiekvieno pakeitimo. Testavimo kartojimas, jeigu jis neautomatinis, yra labai imlus darbui. Papildomos pastangos automatiniam testavimui visuomet atsiperka. Todėl testavimas turi būti vykdomas automatiškai.

112. Kokie yra būdai programavimo našumui matuoti?

Apimtimi paremtas matavimas vertina programų kūrimo proceso rezultatus. Tai gali būti pateikto išeities kodo, objektinio kodo eilučių kiekis ir pan. Funkcionalumu paremtas matavimas vertina pateiktų programų funkcionalumą. Šio tipo matavimams priskiriami ir geriausiai žinomi funkciniai taškai.

113. Kaip suskaičiuojami programos funkciniai taškai?

Remiasi programos savybėmis: įėjimų ir išėjimų skaičius, vartotojo sąveikų skaičius, išorinių sąsajų skaičius, sistemoje naudojamų failų skaičius. Funkciniai taškai skaičiuojami dauginant kiekvienos charakteristikos reikšmę iš svorio ir viską sumuojant.

114. Kokie faktoriai daro įtaką programavimo našumui?

Patirtis taikymo srityje, proceso kokybė, projekto dydis, naudojama technologija, darbo aplinka.

115. Kokie metodai yra taikomi kaštams vertinti?

Algoritminis kaštų skaičiavimas, ekspertų nuomonė, vertinimas pagal analogą, Parkinsono dėsnis, kaina laimėjimui.

116. Kaip pasirinkti kaštų vertinimo metodą?

Visi kaštų vertinimo metodai gali būti netikslūs ir neaišku, kurį pasirinkti. Todėl rekomenduojama kaštus vertinti keliais metodais ir jeigu vertinimai panašūs galima juos primti kaip teisingus. Priešingu atveju reikia tikslinti duomenis vertinimo metodams, kol vertinimai taps panašūs.

117. Kaip algoritmiškai apskaičiuojami kaštai?

Kaštai yra matematinė funkcija, kurios reikšmė priklauso produkto, projekto, ir proceso atributų, kurių reikšmes nustato projekto vadybininkas: pastangos žmogaus darbo mėnesiais = A x $Dydis^B$ x M. A yra nuo organizacijos priklausanti konstanta, B atspindi pastangų neproporcingumą dideliems projektams ir M yra daugiklis atspindintis produkto, proceso ir žmonių atributus. Programos dydis yra nurodomas tūkstančiais eilučių.

118. Kokie yra pagrindiniai produkto kokybės faktoriai?

Proceso kokybė, naudojama technologija, darbuotojų kvalifikacija, skirtos lėšos, laikas ir tvarkaraštis. Visais atvejais jei naudojamas nerealus planavimas, produkto kokybė nukentės.

119. Koks programinės įrangos kūrimo proceso tobulinimo tikslas?

Organizacijos siekia sumažinti programinės įrangos kūrimo kaštus, trukmę ir pasiekti, kad būtų mažiau klaidų programose. Paskutiniu metu vis daugiau dėmesio kreipiama į kuriamos programinės įrangos kokybę. Organizacija gali turėti ir kitų tikslų.

120. Kokie yra programinės įrangos kūrimo proceso tobulinimo etapai?

Tobulinimų įvardijimas, tobulinimų prioritetizavimas, proceso keitimas, darbuotojų apmokymas, keitimų derinimas. Keitimai turėtų būti skatinami matuojamais tikslais.

121. Kokie yra galimybių brandos modelio lygiai?

Pradinis - iš esmės nekontroliuojamas. Pakartojamas - produkto valdymo procedūros apibrėžtos ir naudojamos. Apibrėžtas - proceso valdymo procedūros apibrėžtos ir naudojamos. Valdomas - kokybės valdymo strategijos apibrėžtos ir naudojamos. Optimizuojantis - proceso tobulinimo strategijos apibrėžtos ir naudojamos.

122. Kam reikalingas įmonės galimybių brandos modelis ir kaip jis nustatomas?

Vertinimą vykdo audito firma. Audito išlaidas apmoka pati įmonė. Nustatytas galimybių brandos modelio lygis leidžia įmonei sulaukti daugiau užsakymų.

123. Kokios universalios ir užsakomos programinės įrangos kitimo tendencijos?

Užsakomosios programinės įrangos apimtys ryškiai mažėja, stengiamasi sukurti tokią programinę įrangą kuri tiktų kuo didesniam vartotojų kiekiui. Tokiu būdu mažėja kūrimo išlaidos ir programinė įranga pinga.

124. Kokią įtaką daro globalizacijos tendencijos?

Programinę įrangą kuria vis mažiau pasaulio traukos centrų, nyksta ribos tarp nacionalinių valstybių, vis labiau įsigali tarpnacionalinės korporacijos, programavimo darbai keliasi į besivystančias šalis.

125. Kokią įtaką daro internetinės paslaugos?

Vis mažiau programinės įrangos yra diegiama vartotojų kompiuteriuose. Auga kompiuterinių tinklų greitaeiga. Internetinės paslaugos lemia globalizacijos tendencijas. Kompiuterių apsaugos problemos tampa labai svarbios. Bendru atveju mažėja kūrimo ir eksploatavimo išlaidos.

126. Kaip siejasi paskirstytų sistemų taikymas ir informatikos specialistų poreikis?

Vartotojų kompiuteriai iš esmės naudojami tik rezultatams atvaizduoti. Skaičiavimai vykdomi nutolusiuose serveriuose, geriau išnaudojami bendri resursai, skaičiavimai debesyse leidžia konfiguruoti virtualius serverius. Nuotoliniai skaičiavimai, darbų koncentracija, aparatūros universalumas mažina informatikos specialistų poreikį.

127. Kokios atviro kodo naudojimo tendencijos?

Atviro kodo, kurį tobulina patys vartotojai, populiarumas mažėja dėl globalizacijos, universalumo, internetinių paslaugų tendencijų, dėl nepakankamo valdymo ir konkurencijos.

128. Kokią įtaką daro masinio naudojimo programinės priemonės?

Smulkios taikomosios programėlės plinta įvairiose srityse. Jas pasiūlyti ir pardavinėti gali atskiri programuotojai. Dėl masiškumo kaina gali būti labai nedidelė. Tokiu būdu gerai išnaudojamas kūrybinis potencialas ir darbuotojų iniciatyva.

129. Kokios tyrimo kryptys numatytos Europos H2020 programoje?

H2020 programoje numatyta kiber-fizinės sistemos, sujungiančios aparatūrą, programinę įrangą, kompiuterinius tinklus, valdymą; daiktų internetas, didelių duomenų apdorojimas, sistemų sauga ir apsauga, daugiamodalinės bendravimo su kompiuteriu sąsajos.