Test on komplekt sisendandmeid ja sisendandmetele vastav oodatud komplekt väljundandmeid.
Testimine on programmi täitmine eesmärgiga leida vigu.
Viga on programmi käitumine spetsifikatsioonile mittevastavalt.
Hea test on test, mis avastab võimalikult palju vigu.
Edukas test on test, mille tulemusena leiti viga/vigu.
Silumine on leitud vea kõrvaldamine.
Verifitseerimine püüab näidata, et järgmise etapi produkt vastab eelneva etapi määratlusele.
Valideerimine püüab näidata, et oleme teinud seda, mida vaja.
Sertifitseerimine on kolmanda osapoole tegevus, mis püüab näidata, et antud toode vastab vajalikele standarditele ja normdokumentidele.
Testimine programmi teksti põhjal: püütakse teste tekitada programmi teksti põhjal. Vaadates programmi ainult "väljastpoolt", ei pruugi leida kõiki harusid.
Lauseadekvaatsus nõuab, et testimise tulemusena peab programmis iga lause olema vähemalt üks kord töötanud. LA (nagu ka muid) kriteeriume pole alati võimalik täita: võib juhtuda, et programmi loogika ei luba mingit lauset üldse läbida. Eelised: programm on süstemaatiliselt katsetatud, vähe teste, lihtne. Puudused: ei anna andmete teste, ei leia puuduvaid harusid. Hinnang: kui aega vähe, siis kasuta.
Haruadekvaatsus läbib kõik harud, ka tühjad. Seega on täielikum kui LA.
Elementaartingimuste adekvaatsus. Kui If-lause tingimus on loogiline avaldis, siis tekivad selle avaldise läbimisel sisuliselt programmi harud, kuigi väliselt selliseid harusid ei ole. HA-st tugevam nõue on, et ka need harud oleksid testide käigus läbitud.
Teeadekvaatsus. Kõik programmi teed peavad olema läbitud. Idee on hea, kuid reaalsete programmide puhul on vajalike testide maht enamasti liiga suur.
Kuidas valida sisendeid? Valida sellised, et kõik programmi osad töötaksid vähemalt korra.
Kuidas hinnata väljundit? Väljundid võetakse ülesande püstitusest.
Millal testimine lõpetada? Testimine lõpetatakse siis, kui on rahuldatud tarkvara veakindluse nõuded või juhul kui tekkinud veasituatsiooni tõttu pole edaspidine testimine võimalik.
Kes on testijaiks? Kõige halvem testija on programmi looja, kõige parem tulevane kasutaja.
Testimise ja verifitseerimise vahekord: nad täiendavad üksteist.
Kas teeadekvaatsus garanteerib programmi korrektsuse? Peaks garanteerima.
Programmi keerukus. McCabe'i programmi keerukuse mõõt V(G) põhineb programmi hargnemistel. V(G)= programmi graafi tsüklomaatiline keerukus = graafi regioonide arv = E-N+2 (E-kaared, N-tipud) = P+1 (P-predikaadid). V(G) annab haruadekvaatsete testide arvu.
Silmuste testimine Ühekordse silmuse puhul (max n läbimist) võib testida vastavalt vajadusele m<n, n-1, n, n+1 läbimist. Mitmetasemelise silmuse puhul (k taset) võib kasutada sellist protseduuri: 1) Kõige seesmisel tasemel ühekordse silmuse testid, välised tasemed minimaalse läbimiste arvuga, 2) 2 ,...., k taseme puhul: k-ndal tasemel lihtsa silmuse testid; seesmistel tasemetel m< n läbimisega; välised tasemed minimaalse läbimiste arvuga. Kui programmis järjestikku asuvad silmused on omavahel sõltuvuses, testitakse neid nagu mitmekordset silmust; kui ei - nagu ühekordset silmust.
Funktsionaalne ja andmepõhine testimine, erinevus programmipõhisest testimisest. FT puhul vaatame programmi kui "musta kasti" - me ei tea tema sisemist ehitust, teame vaid sisendeid ja vastavaid väljundeid. Erinevus on selles, mille põhjal leitakse testide sisendid. FT-st vaatame ekvivalentsiklasside, piirjuhtude ja ve otsingu meetodeid.
Ekvivalentsiklasside analüüs. EKA püüab leida selliseid sisend- ja väljundandmete klasse, et (1) kui mingite andmetega klassist K leitakse viga, siis leitakse sama viga ka teistsuguste andmetega klassist K ja (2) kui mingite andmetega klassist K viga ei ilmne, siis ei aita ka teistsugused andmed klassist K. Piisab siis sellest, et valida mingi üks punkt (test) klassist K, et testida kõik andmed samast klassist. Kui ekvivalentsiklassid on valitud, tuleb nende põhjal koostada testolukorrad ja siis testid.
Piirolukorrad. On leitud, et vigu esineb palju EK piiridel, seega tasub teha PO teste. Näiteid: *Kui piir on reaalarv R, siis tasub teha teste sellel piiril, sellest veidi suuremal ja väiksemal väärtusel - s.t. väärtustel R, R-e, R+e (e valitakse vähim väärtus, mis on arvuti seisukohast eristatav), *Kui ülemist (alumist) piiri pole antud, tasub testida väga suure positiivse (negatiivse) arvuga, * Kui piir on täisarv N, tasub testida väärtusi N, N-1, N+1 jne.
Vea otsing on testimine eksperteadmiste põhjal. Kogenud arendaja oskab tõenäolisi vea kohti ette aimata. Vigu aitavad leida: *üldised teadmised, *teadmised konkreetse rakendusvaldkonna kohta, *teadmised riist- või tarkvarakeskkonna (näiteks mingi konkreetse keele) kohta, *teadmised arendusmetoodika kohta, *teadmised konkreetse arendaja või tellija kohta, jne. Vea otsing on väga efektiivne vahend, seda võib kasutada süsteemselt või intuitiivselt.
Funktsionaalse testimise korraldus ja hinnang. FT võib etappideks jagada järgmiselt: 1.eristada sisend- ja väljundandmete ekvivalentsiklassid, 2. leida igas klassis, nende piiridel ja kombinatsioonidel esinevad testolukorrad, 3. leida igale testolukorrale vastavad sisend-väljundandmed, 4. identifitseerida testid, koostada testimise plaan, 5. testida, võrrelda tulemusi, hinnata. Kokkuvõte FT-st: *idee: süstemaatiline testimine sisendi/väljundi põhjal, *eeltingimus: spetsifikatsioon, *eelis: kasutajale arusaadavam, *puudused: kui ekvivalentsiklassid on sõltuvuses, võib testimine olla mahukas. Spetsifikatsiooni pole alati olemas, *hinnang: hea.
Andmepõhine testimine. Erinevus FT-ga on selles, et niiid võivad nt. piirolukorrad tekkida programmis või arenduskeskkonnas antud kitsendustest, nagu massiivi lubatud pikkus, reaalarvu võimalik väärtus vms. Väljundid võetakse ikka ülesande püstitusest.
Testimine juhuslike andmetega. Kes testimisest eriti palju ei tea, just nii testibki - proovib mingite andmetega, kuidas töö läheb. Nii võivad jääda olulised vead avastamata. Samas võib see testimise meetod olla vägagi efektiivne teatud tingimustel: *tuleb teada milliste andmetega seda tarkvara kasutatakse?, *tuleb genereerida matemaatiliselt juhuslikke andmeid, *tuleb kasutada ka piirolukordade teste. Sellise testimise puhul saab prognoosida vigade arvu.
Lisatud vead. Sõltumatu isiku või grupi poolt lisatakse süsteemile juhuslikke vigu. Testimise käigus avastatakse nii lisatud kui ka tegelikke vigu. Eeldades, et vigade avastamise protsent on mõlemal juhul sama (kehtib ainult teatavatel tingimustel) saab prognoosida süsteemi peale testimist jäänud vigade arvu. Puuduseks on see, et lisatud ja tegelike vigade avastamise protsent ei pruugi olla same.
Testimise maht ja lõpetamine. Testimise maht peaks sõltuma tarkvara veakindluse nõuetest - testitakse seni, kuni need on rahuldatud. Praktiliselt siiski tehakse nii vaid tõesti kriitiliste rakenduste korral. Testimise maht: *testimine peab vastama haruadekvaatsuse kriteeriumile, *kõik ekvivalentsiklassid (piirjuhud) peavad olema testitud, *olulisemad andmekombinatsioonid peavad olema testitud, *andmepõhise testimise piirjuhud peavad olema testitud, *V% lisatud vigadest peavad olema avastatud; *tarkvara töökindlus peab olema P%. Nagu teame, võivad vead kõigi nende kriteeriumide puhul ikkagi sisse jääda.
Juhuslik katsetamine ja süstemaatiline testimine: elised, puudused, millal üht või teist kasutada. Juhuslik katsetamine võib olla vägagi tulemuslik, kui seda teha oskuslikult. Lisaks saab sellise testimise puhul hinnata tarkvara töökindlust ja prognoosida vigade arvu. Puudusi: tulemuste hindamine võib olla väga mahukas; olulised "erilised" punktid võivad jääda katsetamata, testimise protsess võib olla väga aeganõudev. Süstemaatilise testimise eelis on see, et me teame, et kõiki asju on katsetatud (teataval nivool, mis sõltub ressurssidest ja vajadustest).
Arutelude ja hindamiste liike: * töö analüüs autori poolt, *läbivaatus, *audit, *programmeerija hindamine.
Läbivaatus, selle plussid, miinused, eeltingimused, osavõtjad, korraldus, mängud, suhe juhtkonda. Läbivaatuse idee on toote (spetsifikatsioon, projekt, dokumentatsioon,...) rühmapoolne ühisarutelu kindlate reeglite järgi. Pluss: *vigu saab leida varastel arenduse etappidel. Miinus: *rühma liikmed võivad olla väga erinevad inimesed. Korralduse: *nii palju ettevalmistusi kui vaja, nii vähe kui võimalik. Eeltingimus: *Kõigil grupi liikmetel ettekujutus sellest, mida neilt oodatakse, *soovit. igaühel üks positiivne ja üks negatiivne kommentaar. Osavõtjad: *Esitaja, *Koordinaator, *Sekretär, *Liikmed (sh. kasutaja esindaja). Otsese olemuse viibimine läbivaatusel on ebasoovitav. Läbivaatusi võivad häirida osavõtjate vahelised "mängud". Nähtes korduvalt kasutatud sõnaühend "Jah, aga ... " viitab mängule. Läbivaatuse tulemusteks peaks olema leitud ja parandatud vead ning parem süsteem, samuti allakirjutatud protokoll.
Audit, selle eesmärgid, auditeeritavad objektid, maht, auditor, riskid, planeerimine. IS audit = ülevaade, hinnang ja erinevad ettepanekud. Eesmärgid: *hinnata süsteemide ja infotöö vastavust ettevõtte (äri)huvidele, *hinnata ettevõttega seotud kolmandate osapoolte nõuete rahuldatust, *hinnata firma tegevusele eluliselt vajaliku info usaldatavust, kättesaadavust ja kaitstust, *hinnata süsteemide või infotöö korralduse kvaliteeti, turvet ja töökindlust jne. IS audit maht sõltub ülesandest. Valida auditor, kes: *on sõltumatu auditeeritavast rakendusest, *on ekspert infotehnoloogia vastavas valdkonnas, * jälgib auditeerimise head tava ja reegleid. Auditi riskid: *tegevusrisk - IS toimimisel/arendamisel, *sisekontrolli risk, *avastamisrisk - auditior ei avasta olulisi vigu. Auditi planeerimine: *määratletakse kriitilised valdkonnad, *töö õiges järjekorras, koordineritult, *tõendusmaterjalid, *kontrollimetoodika, *raportid, *tähtajad, töömaht
Töö analüüs autori poolt. On kõige odavam meetod ja tavaliselt leiab vigu, seega soovitav kasutada. Meetod võib siiski olla ebapiisav, kuna *autori motivatsioon on lõpetada töö, mitte leida vigu, *autor pole piisavalt enesekriitiline, *autor jälgib oma kujunenud loogikat ega suuda prognoosida kasutaja võimalikke tegevusi.
Programmeerija hindamine. Meetodi idee on anda programmeerijaile arusaamine oma tugevaist ja nõrkadest külgedest. On anonüümne, ei mõjuta grupi liikmete palka ja käekäiku. Tulemusena saab iga osaleja hinnangu oma kvalifikatsioonile.
Küsimustikkude liigitusi, programmeerimisele orienteeritud küsimustiku struktuur
Küsimustikke võivad kasutada autor, läbivaataja, analüüsija jne. Nad sisaldavad momente, millele muidu ei oskagi nagu tähelepanu pöörata. Küsimustikke on väga mitmesuguseid ja erineva mahuga. Programmeerimine: *Andmete kirjeldamine, *Pöördumine andmete poole, *Arvutuste vead, *Võrdluste vead, *Juhtimise vead, *Alamprogrammi, *Sisend ja väljund
Töökindluse saavutatav tase seniste meetoditega, selle olulise suurendamise võimalusi. Testimisega on saavutatav tase 10-4 viga tunnis ehk u. üks viga aastas. Teatavates valdkondades nõutakse tunduvalt suuremat veakindlust, näiteks NASA 10-10 viga tunnis (ehk üks viga miljoni aasta jooksul) kümnetunnisel lennul. Vahendid töökindluse parendamiseks: *dubleerimine - kui pannakse paralleelselt käima mitu programmi ja võrreldakse tulemusi, emam levinud vastused loetakse õigeiks (tarkvara puhul hästi ei sobi), *veapuu analüüs - ehitatakse ja/või puu. Alustatakse mingist suurest veast, mida tahetakse vältida, vaadatakse selle vea eeltingimusi, eeltingimuste eeltingimusi ja nii edasi (tarkvara puhul ei sobi), *formaalsed meetodid - võimaldavad taset 10-7 ... 10-8 viga tunnis - mitte piisav, aga parem kui testimine.
Programmide tõestamine, selle ajalugu, probleeme, võimalusi jne. Tekkis 1960.ndatel, kui leiti, et programmide töökindlust on vaja tõsta. Ülevaatlikult võib iseloomustada järgmiselt: *siht (näidata, et programm vastab spetsifikatsioonile), *idee (luuakse loogiline arvutus, selle arvutuse terminites kujutatakse spetsifikatsioon ning programm. Tõestatakse, et lähtudes antud sisenditest ja kasutades antud programmi jõutakse nõutud väljunditeni), *eeltingimused (spetsifikatsioon, programm, loogikavahendid, ressursid), *eelised (suurem töökindlus, formaalne korrektsus), *puudused (töömahukas, ei välista spetsifikatsiooni vigu), *tulemused (programm tõestatakse spetsifikatsiooni suhtes).
Tõestamise tasemed: spetsifikatsioon (läbivaatus), projekt (projekti tõestus), programmi kood (programmi tõestamine), tõestusmeetodid (loogika), objektkood (programmi või kompilaatori tõestamine), täitmine (mikroprogrammi/riistvara tõestamine).
Võrdlus: töökindluse saavutamine riistvara ja tarkvara puhul. Üks vahend töökindluse parendamiseks on dubleerimine. See meetod ülgustab ennast väga hästi riistvara puhul, kuid ei ole sama kasulik tarkvara korral (üks põhjusi - imimlik loogika jälgib tihti samu radu ja paralleelsetes arendustes tehakse samu vigu). Töökindluse saavutamine riistvara puhul on niisiis lihtsam.
Lihtsamaid skeeme tarkvara kontrolli korralduseks. Milline neist on parim? 1)Kasutaja ja arendaja on sama isik - kui toode pole vastutusrikas, toimib "ise teen, ise testin" korralduse. 2)Kui kasutaja ja arendaja on erinevad ja toode pole vastutusrikas, siis võib kasutada järgmist skeemi: *tellija annab tellimuse, *programmeerija annab tellijale arendatud tarkvara, *tellija katsetab tarkvara ja annab programmeerijale leitud vigade kirjelduse, *programmeerija parandab vead ja annab üle parandatud toote. 3)Vastutusrikkama tarkvara puhul võib kasutada sellist skeemi: *arendajapoolne testimine, *rakenduse ja testimise eksperdi poole testimine, *kasutajate grupi testimine. Neist variantidest tuleb antud olukorras sobivaim korraldus.
Vajadusi keerukamaks korralduseks. *keerukas toode - sellisel juhul on otstarbekas alustada kontrolli arenduse algetappidest ja jaotada see hallatavateks osadeks, *töökindluse poolest kriitiline toode - testimine tuleb jaotada etappideks, kasutada erimeetodeid, *organisatsiooniliselt lahutatud arendusetapid - näiteks peab sel juhul olema korraldatud analüüsi, projekteerijate ja programmeerijate vaheline koostöö, *organisatsioonilisel keerukas kasutaja - nõuab erinevate kasutajagruppide koostööd, *mitmeetapiline mahukam testimine, mis vaheldub arendustegevustega - nõuab vigade ja nende paranduste jälgimist ning regressioonitestimist (kas vanad testid töötavad peale parandusi?)
Kontrolli projekteerimise loogikac olukord ja testitavad objektid - meetodid – tegevused. Kontrollid projekteeritakse vastavalt nõudmistele ja tarkvara keerukusele. Vt. eelmine ja üleeelmine punkt.
V-mudel, arendusprotsessi integreeritud kontroll. Kaasaegses süsteemiarendusprotsessis alustatakse kontrollitegevustega võimalikult varakult - mida varem vead leitakse, seda odavam on neid parandada. Seepärast projekteeritakse iga arendusetapi käigus ka testid. Tegevuses vastavad eri arendusetappidele eri tüüpi testid, tekib nn. tarkvara elutsükli V-mudel. Üks levinumaid on järgmine: *süsteemi üldprojektile vastab süsteemitestimine, *nõudmiste spetsifikatsioonile vastab valideerimistestimine, *realisatsiooni projektile vastab integratsioonitestimine, *kodeerimisele vastab mooduli testimine. Testimiste väljunditeks on leitud vead ja töökindluse prognoos.
Ühiku/mooduli test. Enamus eelmistes jaotistes kirjeldatud meetodeid on kasutatavad mooduli tasemel testimiseks. Seejuures on kasulik jälgida testimise hinnaefektiivsust (kui palju maksab ühe vea leidmine ja parandamine erinevate meetoditega?). Ilmselt on otstarbekas teha kõigepealt hinnaefektiivsemaid teste.
Integratsioonitest. Kui mooduli taseme testid töötavad, tuleb moodulid kokku panna ja testida. Selleks on mitu meetodit: *"Kõik kokku ja test" - võib töötada lihtsamate süsteemide korral; keerukamate puhul on raske vigu lokaliseerida, *alt üles integreerimistestid (väljakutsuvad moodulid asendatakse draiveritega. Eelis: korraga saavad töötada paljud, igaüks testib sõltumatult oma osa. Puudus: tervikpilti süsteemist ei teki viimase hetkeni), * ülalt alla integreerimistestid (väljakutsuvad moodulid asendatakse lühistega. Eelis: midagi on juba algusest saadik olemas, parem side kasutajaga.), * segameetod ("võileib"; hakatakse liikuma nii ülalt alla, kasutades lühiseid, kui ka alt üles, kasutades draiveried. Võimaldab nii saada pilti arendusest kui ka paljudel arendajatel korraga töötada)
Valideerimistest. Valideerimine vastab küsimusele" kas ehitasime õige tarkvara?" - tarkvara katsetatakse vastavuse suhtes nõudmistele. Valideerimise etappi võivad kuuluda nn. alfa-testimine (kasutaja poolt/andmetega arendaja keskkonnas) ja beeta-testimine (kasutaja poolt/andmetega kasutaja keskkonnas).
Süsteemi testimine. Üldjuhul kuulub tarkvara mingi süsteemi koosseisu. Ka nõudmised tarkvarale võivad olla vigased, seega ei anna valideerimistestid õigsuse garantiid - tuleb testida kogu süsteemi tervikuna. Süsteemi testide hulka kuuluvad peale tavaliste tööreziimide testide tihti ka jõudluse, taluvuse, taastamise ja turbe testid. On olemas ka erilisi olukordi, mida on kulukas või võimatu testida. Sellistel puhkudel saab rakendada näiteks: *läbivaatusi, *tõestamist, *töökindluse mudeleid, *simuleerimist, *vigade arvu prognoosi. Näiteks võib süsteemitestimine tuua kaasa hinnaliste seadmete kasutamist ja hävimist. Mida hilisem testimise etapp, seda rohkem ressursse see nõuab, seega tuleks võimalikult palju tööd ära teha alguses.
Testimise ja kontrolli meetodite efektiivsuse võrdlus ja soovitused kasutamiseks
mida teha, kui testida ei saa? Vt. eelmine punkt.

Kvaliteet on toote, teenuse või protsessi omaduste kogum, mis rahuldab määratud või eeldatavaid vajadusi. Lühidalt on kvaliteet vastavus nõudmistele.
Standard on teatud organisatsiooni poolt kinnitatud normdokument, mis on suunatud standardimiseesmärkide saavutamiseks.
Tarkvara, nõudmiste liigitusi, elutsükkel, elutsükli liigid. Vajadusi ja nõudmisi võib liigitada mitmeti. Kõigepealt, nad võivad olla määratud (sõnastatud, lepingus esitatud jne.) või eeldatud ("tahaksin seda", "tavaliselt tehakse nii"). Nõudmised tulenevad mitmetest allikatest: spetsifikatsioon, standardid, normid, seadused, hea tava, eetika jne. Infosüsteemi areng on pidev protsess, mida tuleb jaotada (alam)projektideks ja etappideks, et seda protsessi paremini hallata.
Kvaliteedijuhtimine - üldise juhtimisfunktsiooni osa, mis määrab kindlaks ja rakendab kvaliteedipoliitika.
Kvaliteedipoliitika - organisatsiooni tippjuhtkonna ametlikult esitatud kvaliteedialased üldised eesmärgid ja juhtnöörid.
Kvaliteedisüsteem - organisatsiooniline struktuur, vastutused, protseduurid ja vahendid kvaliteedi juhtimiseks.
Kvaliteediõendus - tegevuste kogum, mis on vajalik piisava usaldatavuse tagamiseks, et toode või protsess rahuldab esitatud kvaliteedinõudeid.
Kvaliteet ja organisatsiooni juhtimine. USA ja Euroopa ning Jaapani arusaamad kvaliteedist on olnud mõnevõrra teistsugused. USA, Euroopa - kvaliteedijuhtimine on orienteeritud tulemusele, oluline on lõpptulemuse , mitte protsessi kvaliteet. Paraku ei taha see põhimõte keerukate toodete puhul hästi töötada. Jaapan - kvaliteedijuhtimine on orienteeritud protsessile, oluliseks peetakse, et kogu tootmine oleks kvaliteetne.
Kvaliteedijuhtimise ja arendusmeetodite seos. Veel on erinevad olnud ka arengumeetodid: kui USA -s ja Euroopas tehakse läbimurdvaid, kardinaalselt uusi otsustusi ja samme (innovatsioon), siis Jaapan on osanud areneda pidevalt, ilma suurte hüpete või suunamuutusteta.
Kvaliteedijuhtimise integreerimine elutsüklisse. Eelpooltoodust tulenevalt on USA ja Euroopa kvaliteedijuhtimine olnud ajalooliselt enam orienteeritud statistilistele meetoditele, nt. partiide kontrollile. Jaapanis on see seevastu rohkem iga töötaja ja iga hetke küsimus (kaizen).
Kvaliteedijuhtimise käsitlusi, teese, süsteeme ja auhindu. Philip B. Crosby on sõnastanud kolm populaarset teesi: *Tee kohe õieti, *Ei ühtegi vigast toodet, *Kvaliteet on tasuta. Philip B. Crosby kvaliteedijuhtimise koostisosad: 1. Juhtkonna toetus 2. Kvaliteedigrupp 3. Kvaliteedi mõõtmine 4. Kvaliteedi maksumuse hinnang 5. Töötajate kvaliteeditaseme tõstmine 6. Nulldefekti komitee 7. Nulldefekti päev 8. Selgelt formuleeritud sihid näiteks 30, 60, 90 päeva peale 9. Vea põhjuste kõrvaldamine 10. Parimate tunnustamine 11. Kõike seda tuleb teha korduvalt Armand V. Feigenbaum on sõnastanud järgmised kvaliteedijuhtimise tegevused: 1. Püstita kvaliteedistandardid 2. Hinda vastavust standardeile 3. Tegutse kui standardeid rikutakse 4. Arenda standardeid edasi.
Miks peaks süsteemi arendaja teadma kvaliteedist ja standarditest? Tellija? Kasutaja? Hooldaja? Tarnija? Firmajuht?

Standardimiskeham, konsensus. Keham - juriidiline või haldusüksus, millel on kindel ülesanne ja koosseis, nt. organisatsioon, firma, ettevõte, ametkond, fond vms. Konsensus - standard loob üldjuhul eelise teatud organisatsioonile.
Standard, selle kohustuslikkus, näiteid, liigitusi, kasutusvõimalusi. Liigitus: *Ametlikud standardid (vastu võetud tunnustatud kahelu poolt). *Tööstus e. de facto standardid (ei pruugi olla ametlikult kinnitatud, kuid on üldlevinud). *Tehnilised raamstandardid (standardite kogumikud, juhendid nende kasutamiseks) *Riiklikud soovitused (tulenevad kliendi poolt) *Firmasisesed standardid. Kohustuslik? *Ei, kuid seda võib muuta seaduste abil.
Standardimise eeliseid, puudusi, maht Eelised/eesmärgid: *Kulutuste minimeerimine. *Komponentide vahetatavus. *Komponentide koostöökulu minimeerimine. *Protseduuride lihtsustamine/parendamine (saab teistelt ettevõtteilt kindlalt töötavaid protseduure üle kanda).
*Kvaliteedijuhtimine. *Standardid võimaldavad reguleerida tootjate omavahelist konkurentsi. Puudused: *Lisakulu,-aeg,-bürokraatia *Takistab innovatsiooni? Kohustuslik? *Ei, kuid seda võib muuta seaduste abil. Maht: * Nii palju kui vajalik *Nii vähe kui võimalik. Millal: Nii vara kui vajalik *Nii hilja kui võimalik.
Standardiinstitutsioonid *ISO - International Organisation for Standardisation (1947), *IEC - International Electrotechnical Commission (1906), *ECMA - European Computer Manufacturers Association, *ANSI - American National Standards Institute.
Tarkvara standardite liigitusi, võimalik klassifikatsioon. Liigitusi: *üldised / organisatsioonilised / tehnilised *rahvusvahelised / riiklikud *ametlikud / mitteametlikud *ettevõttesisesed / -välised
standardimine Eestis: korraldus, olemasolevad standardid,ute tekitamise protseduurid, plaanid, standardimisalad Korraldus: *Loodud Infotehnoloogia standardimise tehniline komitee *Eesti Standardiamet *Informaatikafond *Eesti on juba ka ISO/IEC liige. Standardid:
*Infotehnoloogia reeglid Eesti keele ja kultuuri keskkonnas, *Eestikeelse elektronposti profiil
Standardite loomise protseduur: *Ekspertide töögrupid => standardiprojektid *Huvigruppide hinnangud projektile *Korrigeerimine, kooskõlastamine *Registreerimine
tarkvara vajaduste ja nõudmiste liigitusi
tarkvara kvaliteedi atribuutide struktuuri vajalikkus, näide struktuurist, integreerimine
näide kvaliteedifaktorite süsteemist, kahe faktori kriteeriumid
meetrikate idee, omadusi, efekt, probleemid, liigitusi, näiteid, kasutamine
kvaliteedi meetrikate näiteid, väärtuse määramispiirkond, parim väärtus
ISO 9000 efekte
ISO 9000 standardite seeria sisaldab järgmisi olulisemaid standardeid: *ISO 8402 - Kvaliteedisõnastik *ISO 9000 - Ülevaade järgnevast 4-st standardist, kasutusjuhised *ISO 9001 - Standard kvaliteedi juhtimise jaoks süsteemidele, mis hõlmavad disaini või projekteerimise etappi (tarkvaratootmisel kui elutsükkel). *ISO 9002 - Hõlmab süsteeme, milles põhiline tegevus on tootmine ja installeerimine. *ISO 9003 - Masstootmise standard, lõpliku katsetamise ja inspektsiooni standardid. *ISO 9004 - Üldpõhimõtted kvaliteedijuhtimise süsteemidest.
toote elutsükkel ISO 9000 keskkonnas (Tootja) projekt->väljatöötamine->tootmine->kontroll->pakkimine (Kasutaja) realiseermine->montaaz->kasutamine->hooldus->hävitamine
ISO 9001, 9002, 9003 suhe
ISO 9000-3 struktuur *annab aluse kogu firma töö korralduseks, *annab aluse tarkvara kvaliteedikäitluseks, *on aluseks võetav kogu tarkvara standardite temaatikale, *on kasulik kui orienteerutakse ISO 9000 sertifitseerimisele, *on kasulik, kui orienteerutakse Euroopa Liidu turule, *on kasulik kui PR vahend.
kvaliteedisüsteemi ümbrus (raamistik): tellija vastutus, kvaliteedisüsteem
infosüsteemi elutsükli etappidega seotud KJ tegevusi: näiteid, tootja läbivaatused, kvaliteet lepingus,
toetavad (elutsükli etappidest sõltumatud) kvaliteedijuhtimise tegevused: näiteid, konfiguratsiooni- ja dokumendijuhtimine
kvaliteedijuhtimise ja -tõenduse etapid süsteemi elutsüklis
standardid, sertifitseerimine, akrediteerimine
skeem kvaliteetegevuste alustamiseks
millist tarkvara elutsüklit toetab ISO 9000-3?

ÜL.:Mitu testi on vaja antud programmi lause- (haru-, tee- jne.)adekvaatseks testimiseks?
ÜL.:Millised on antud programmi puhul ekvivalentsiklassid? Testitavad olukorrad? Testid?
ÜL.:Antud tarkvara ja selle rakendamise olukord. Kas ja milliseid testimise meetodeid kasutada?