Tarkvara kvaliteet ja standardid

Jaak Tepandi
TTÜ informaatikainstituut

Me kasutame iga päev süsteeme, mille üks komponent on tarkvara (kursuses kasutame mõistet tarkvarasüsteemi sünonüümina, lülitades sinna dokumentatsiooni, metoodikaid, vahendeid jne). Mõnikord oleneb sellest meie sissetulek, kui kasutame tarkvaravahendeid tööks või müüme tarkvara, meie vara pangas või elu lennureisil. Oluline on, et tarkvara töötaks.

Töökindluse tõstmise üks võimalusi on kvaliteedijuhtimine. Väga lühidalt on kvaliteet vastavus nõuetele. Keerukate toodete puhul saab vastavust hinnata, võttes arvesse kogu toote loomise protsessi. Seega seob kvaliteet ***toote***, ***nõuded*** tootele ja tootmise ***protsessi***.

Vaadeldav ***toode*** (tarkvara) sisaldab palju komponente. ***Tootmisprotsessi*** - tarkvara elutsüklit - võib mitmeti valida ja esitada (kosemudel, V-mudel, prototüüpimine jne). Standardimine on vahend ***nõuete*** fikseerimiseks ja kvaliteedi edendamiseks, standardid esindavad universaalseid nõuete klasse.

Kvaliteedijuhtimist võib ühendada tarkvarasüsteemi elutsüklisse mitmeti, alates kvaliteediprobleemide ignoreerimisest ja lõpetades kogu arendusprotsessi ja kvaliteedijuhtimise samastamisega. Kursuses on valitud kesktee - kvaliteedijuhtimine on integreeritud elutsükli kõigisse etappidesse. Kursuse alguses antakse konkreetseid testimisalaseid teadmisi ja oskusi, mida saab iseseisva töö käigus kohe kasutada; kvaliteedijuhtimise üldtemaatika jääb kursuse teise ossa.

Kursuse eesmärk on

- anda ülevaade tarkvara kvaliteedi arendamise ja kontrolli probleemidest ning meetoditest
- tutvustada lähemalt testimise meetodeid ja korraldust
- anda praktilisi kogemusi testimise projekteerimisest, läbiviimisest ja dokumenteerimisest

Kursus on kasulik

- süsteemiarendajatele, laiemalt süsteemse töö täitjale, et luua paremat toodet, saavutada tellija rahulolu
- tellijale, ostjale, et olla kursis tarkvarale esitatavate nõuetega, osata valida toodet, jälgida arendusprotsessi ja hinnata tulemust
- kasutajale, kes saab teada, mida võib tarkvaratootelt oodata
- juhile, kes õpib, mida tellijalt ja arendajalt nõuda

Konspekt on kursuse lühiülevaade. Näited ja detailsemad selgitused antakse loengul.

Esimene osa sisaldab testimise meetodeid. Teine osa käsitleb kvaliteedi ja standardite mõisteid laiemalt. Lõpus on kirjanduse loetelu.

Autor tänab kolleege, kursuse kuulajaid ja redigeerijaid, eriti A. Tamboomi, M. Kandelini, M. Tõnissoni, L. Tepandit ja J. Nurmet kasulike märkuste ja abi eest.

Käesolev materjal täieneb ja muutub töö käigus. Praeguses variandis on esialgne osaline tekst. Lõplik variant on olemas enne eksamisessiooni algust. Failis olev tekst võib olla loengutest erinevalt struktureeritud (jaotisteks liigendatud, pealkirjastatud), kuid ta maht ja sisu on samad.

Kommentaarid ja märkused selle konspekti kohta on oodatud, palun saata need aadressil tepandi@cc.ttu.ee või anda edasi loengul/praktikumis.

Sisukord

1. TARKVARA KONTROLL: MEETODID JA KORRALDUS

1.1. TESTIMINE
1.1.1. Testimise põhimõtteid
1.1.2. Testimine programmi teksti põhjal: juhtimine
1.1.3. Funktsionaalne ja andmepõhine testimine
1.1.4. Muid testimise meetodeid
1.1.5. Testimise maht ja lõpetamine
1.1.6. Kontrollküsimusi ja ülesandeid
1.2. STAATILISED MEETODID
1.2.1. Küsimustikud
1.2.2. Läbivaatused ja hindamised
1.2.3. Kontrollküsimusi ja ülesandeid
1.3. TÖÖKINDLUSE OLULINE SUURENDAMINE
1.3.1. Kontrollküsimusi ja ülesandeid
1.4. KONTROLLI KORRALDUS
1.4.1. Kontrolli korralduse võimalusi: ülevaade
1.4.2. Elutsükli V-mudel: Arendusprotsessi integreeritud kontroll
1.4.3. Keerukamad korralduse raamistikud
1.4.4. Kontrollküsimusi ja ülesandeid

2. KVALITEEDIJUHTIMINE, STANDARDID, NORMID

2.1. KVALITEET
2.1.1. Kvaliteedi mõiste
2.1.2. Kvaliteedijuhtimise põhimõisteid
2.1.3. Kvaliteedi hindamise süsteeme
2.1.4. Kontrollküsimusi ja ülesandeid
2.2. STANDARDID
2.2.1. Standardi mõiste
2.2.2. Ülevaade tarkvara standarditest
2.2.3. Kontrollküsimusi ja ülesandeid
2.3. TARKVARA KVALITEEDI KRITEERIUMID JA MEETRIKAD
2.3.1. Kvaliteediatribuutide süsteem
2.3.2. Atribuudid ISO/IEC 9126
2.3.3. Meetrikad
2.3.4. Kontrollküsimusi ja ülesandeid
2.4. INFOSÜSTEEMI AUDIT
2.4.1. Kontrollküsimusi ja ülesandeid

3. LISAINFO JA KIRJANDUS

1. Tarkvara kontroll: meetodid ja korraldus

Tarkvara kvaliteeti saab tõsta mitmeti; näiteks tarkvara testimise ja läbivaatusega. Need kontrollmeetodid on praktilised ja kohe, sealhulgas ka iseseisvates töödes, kasutatavad. Seepärast vaadatakse neid alguses, jättes üldisemad kvaliteedijuhtimise teemad teise ossa.

1.1. Testimine

Igaüks, kes on koostanud mingi programmi, on seda ka arvatavasti katsetanud. Vastutusrikaste süsteemide puhul peavad sellised katsetused (testid) olema süstemaatilised, põhjalikud ja andma ülevaate toote omadustest, nt töökindlusest või tootesse jäänud vigade arvust. Allpool vaadatakse testimise meetodeid.

1.1.1. Testimise põhimõtteid

Testimine on programmi täitmine eesmärgiga leida vigu.

Test on komplekt sisendandmeid ja sisendandmetele vastav oodatud komplekt väljundandmeid.

Testimine on programmi käivitamine testi sisendandmetega ja tulemuste võrdlemine oodatud väljundandmetega. Oodatud väljundid saadakse ülesande püstitusest, mitte programmist.

Programm on vigane, kui tegelik väljund ei vasta oodatud väljundile. Kui leitakse viga (vigu), siis oli test edukas. Hea on test, mis avastab palju vigu. Testimine ei tõesta, et programmis vigu pole - ta näitab vaid, et programmis on vigu.

Arendajad viivad läbi esimesed testid, millest tavaliselt ei piisa. Paremaid tulemusi võib oodata süsteemi tulevastelt kasutajatelt, kes ei ole arendusega seotud.

Testimise projekteerimisel tekib hulk küsimusi:

- kuidas valida sisendeid?
- kuidas hinnata väljundit?
- millal testimine lõpetada?
- kes on testijad?
- millal, kas ja kuidas viia läbi kordustestimine?

Välja on mõeldud rohkesti tarkvara testimiseks ja arendamiseks ette nähtud meetodeid. Näiteks musta ja valge kasti meetodid, kus esimesel juhul ei süveneta programmi teksti, vaid testitakse tema funktsionaalsust; teisel juhul vastupidi uuritakse tähelepanelikult koodi ja püütakse teha teste programmi teksti alusel.

Testimisele võib esitada vastuväiteid.

- Testimine ei tõesta niikuinii õigsust, parem juba verifitseerida. Verifitseerimine on mõnes mõttes hea, kuid väga töömahukas ega pruugi avastada nt

spetsifikatsiooni vigu (vt allpool). Testimine ja verifitseerimine täiendavad teineteist

- Mis kasu on neist meetoditest, kui nad ikkagi ei garanteeri veakindlust, ma parem proovin niisama vigu leida? - Käsitletavad meetodid nõuavad üldjuhul süstemaatilist testimist. Me teame, et kõiki asju on katsetatud teatud nivool, mis sõltub ressurssidest ja vajadustest. Mõni testimismeetod annab ka töökindluse ja vigade arvu prognoosi. Veaotsing, kui seda teeb ekspert, on üks testimise viise. Need meetodid täiendavad teineteist.

Veel mõisteid

- Silumine -- leitud vea kõrvaldamine
- Verifitseerimine -- tegevus, mis püüab näidata, et järgmise etapi tulemus vastab eelnenud etapi määratlusele
- Valideerimine -- tegevus, mis püüab näidata, et tehtud on seda, mis vaja
- Sertifitseerimine -- kolmanda osapoole tegevus, mis püüab näidata, et toode vastab standarditele ja normdokumentidele

1.1.2. Testimine programmi teksti põhjal: juhtimine

Tekstipõhise testimise puhul võetakse ette programm ja püütakse teste luua selle teksti põhjal. See on vajalik, sest funktsionaalsest testimisest ei piisa. Tõepoolest, vaadates programmi ainult väljastpoolt selle käitumise seisukohast, me ei leia näiteks harusid, mida tavalisel täitmisel ei läbita. Tekstipõhisel (läbipaistva/valge kasti) testimisel võib lähtuda juhtimisest või andmetest. Juhtimisest lähtuv tekstipõhine testimine püüab süstemaatiliselt läbida programmi mingeid osasid, näiteks lauseid, harusid, teid. Vastavalt sellele, milliste osade läbimist nõutakse, eristatakse lause-, haru-, teeadekvaatsuse ja muid kriteeriume. Vaatame kõigepealt lauseadekvaatsuse kriteeriumit.

Ka selle meetodi puhul saadakse testi tulemuse hinnang lähtudes ülesandest, mitte programmist.

1.1.2.1. Lauseadekvaatsuse kriteerium

See kriteerium nõuab, et testimise tulemusena peab programmis iga lause olema vähemalt üks kord töötanud. Lauseadekvaatsuse nagu ka muid kriteeriume pole alati võimalik täita. Näiteks võib juhtuda, et programmi loogika ei luba lauset üldse läbida. Samuti ei pruugi programmi testimine selle kriteeriumi põhjal (nagu ka testimine üldse) olla piisav. Sel juhul tuleb kasutada ka teisi meetodeid.

Selle kriteeriumi puhul võib muuhulgas küsida, mitu testi on vaja programmi lauseadekvaatseks kontrollimiseks. Näiteks, kui programmis pole kordust ja on üks *if-then-else*, siis on vaja vähemalt kaht testi, sest ühega ei läbita mõlemat haru.

Kokkuvõte lauseadekvaatsuse kriteeriumist

- Idee: kõik programmi osad on töötanud
- Eeltingimused: programmi tekst on olemas, selle analüüs

- Eelised: programmi on süstemaatiliselt katsetatud. Vähe teste. Selge idee
- Puudused: ei anna andmete teste. Ei leia puuduvaid harusid. Programmi teksti pole alati olemas. Mitte alati saavutatav
- Tulemused: testikomplekt, mis katab programmi
- Suhe teistesse: kasutada koos teiste meetoditega
- Hinnang: kui aega on vähe, siis kasuta
- Vahendid: on olemas vahendid, mis aitavad leida lauseadekvaatset testikomplekti

1.1.2.2. Haruadekvaatsus ja muud

1.1.2.2.1.Haruadekvaatsus

Lauseadekvaatsuse puhul läbitakse kõik laused, kuid harud, milles lauseid pole, jäetakse läbimata. Haruadekvaatsuse nõue eeldab ka tühjade harude läbimist. Seega on ta täielikum: *Lauseadekvaatsus ≤ Haruadekvaatsus*. Haruadekvaatsuse nõuet illustreerib programmi graaf, mille kõik kaared on läbitud. Programmi graafis vastab igale hargnemisele graafi tipp; järgnevad täidetavad hargnemiseta laused võib ühendada üheks tipuks.

Programmi graafil põhineb ka üks esimesi programmi meetrikaid - McCabe (1976) programmi keerukuse mõõt. Meetrikate ülesanne on midagi, antud juhul programmi, mõõta. Meetrikad võimaldavad ka hinnata nt programmi maksumust ja tegemise ajakulu. McCabe programmi keerukuse mõõt põhineb programmi hargnemistel ja seda saab mõõta neljal moel.

Programmi keerukus (kõik mõõdud on samaväärsed) V(G)=

= programmi graafi tsüklomaatiline keerukus (*cyclomatic complexity*) V(G)

= graafi regioonide arv

= E-N+2 (E-kaared, N-tipud)

= P+1 (P-predikaadid)

Kasutamine:

- V(G) annab haruadekvaatsete testide soovitatava arvu
- tekib keerukuse ja arendusaja hinnang
- saab hinnata juba projekti staadiumis, s.t enne programmi tegelikku koostamist
- saab kasutada arenduses hindamiskriteeriumina, ühe mooduli V(G) mõõt peaks siis olema ≤10

1.1.2.2.2.Elementaartingimuste adekvaatsus

Kui If-lause tingimus on loogiline avaldis, siis tekivad selle avaldise läbimisel programmi harud, kuigi väliselt selliseid harusid ei ole. Näiteks võidakse disjunktsiooni puhul hinnata tingimus tõeseks juba esimese komponendi tõesuse korral; järgmisi komponente siis enam ei hinnata ega testita. Neid harusid saab programmi graafis kujutada. Järgmine haruadekvaatsusest tugevam nõue on, et ka

need harud oleksid testide käigus läbitud. Ka see on üsna mõistlik kriteerium ja praktikas kasutusel.

1.1.2.2.3.Teeadekvaatsus

Viimase kriteeriumina sellest klassist vaatame nõuet, et kõik programmi teed peavad olema läbitud. Idee on ilus, kuid reaalsete programmide puhul on vajalike testide maht enamasti liiga suur, mis ei luba kriteeriumit kasutada.

1.1.2.3. Silmuste testimine

Silmuste (korduste, iteratsioonide, tsüklite) testimiseks on mainitud meetodid nõrgad, sest tüüpiliselt piisab lause- või haruadekvaatsuse kriteeriumi täitmiseks silmuse ühekordsest läbimisest. Siin tuleb kasutada teisi meetodeid.

Silmuste testimisel võib eristada silmuse testimist mingi testimiskriteeriumi või -meetodi põhjal ja silmuse kui juhtimiskonstruktsiooni testimist. Esimesel juhul koostame vajalikke teste valitud meetodite põhjal. Väikest segadust võib tekitada testide arv. Kui ühe korduse jooksul täidame kõik testid, kas see on siis üks test või mitu. Vastus sõltub testimise ülesandest ja arveldamise vajadusest, kuid igal juhul tuleb selgitada, kuidas testide arv saadi. Teisel juhul lähtume silmusest kui sellisest ja testime sisuliselt andmepõhiselt juhtparameetri(te) järgi, milleks on pakutud järgmist meetodit.

Ühekordse silmuse puhul (maksimaalselt n läbimist) võib testida vastavalt vajadusele 0, 1, 2, m<n, n-1, n, n+1 läbimist.

Mitmetasemelise silmuse puhul (k taset) võib kasutada järgmist protseduuri.

1. Esimesel (kõige seesmisel) tasemel tehakse ühekordse silmuse testid, välised tasemed testitakse minimaalse läbimiste arvuga.
2. 2 ,…., k taseme puhul: k-ndal tasemel tehakse lihtsa silmuse testid; seesmistel tasemetel m < n läbimisega; välised tasemed minimaalse läbimiste arvuga.

Toodud protseduur vähendab oluliselt testide arvu mitmetasemeliste silmuste puhul.

Programmis järjestikku asuvate silmuste testimine sõltub sellest, kas nad on omavahel sõltuvuses. Kui jah, testitakse neid koos nagu mitmekordset silmust; kui ei, siis eraldi nagu mitut ühekordset silmust.

Struktureerimata silmused tuleb enne muuta struktuurseiks.

1.1.3. Funktsionaalne ja andmepõhine testimine

Funktsionaalse testimise puhul vaatame programmi kui musta kasti, sest me ei tea tema siseehitust, teame vaid sisendeid ja väljundeid. Erinevus on selles, mille põhjal leitakse testide sisendid; väljundid tekitatakse mõlemal juhul spetsifikatsiooni alusel.

Funktsionaalsest testimisest vaatame ekvivalentsiklasside, piirjuhtude ja veaotsingu meetodeid.

Sama tehnikat, mida rakendatakse funktsionaalse testimise juures (testitava ala jaotamine piirkondadeks), saab kasutada ka programmipõhisel testimisel. Sellisel juhul tekitatakse vastavaid piirkondi mitte sisendi/väljundi spetsifikatsiooni, vaid programmi põhjal.

1.1.3.1. Ekvivalentsiklasside analüüs

Kui katsetame programmi, teeme tegelikus elus midagi järgmist (lihtsamalt keerukamale).

1. Proovime, kas programm üldse midagi teeb (*Hello world*).
2. Kas teeb kõige vajalikumaid asju?
3. Kas hädavajalikud väljundid on olemas?
4. Eksperttestid. Kas läheb kergesti rivist välja?
5. Kas teeb kõike, mida vaja?
6. Kas kõik olemasolevad asjad toimivad?
7. Kas võimalikud sisendite tüübid toimivad igas valikus?
8. Katsetame tööd piirjuhtudel.
9. Otsustustabelid - testide sisestamisel arvestada ka sisendite vahelisi sõltuvusi.
10. Kuidas töötab maksimaalsel koormusel?
11. Kasutame formaalseid meetodeid.
12. Ja nii edasi sõltuvalt vajadustest.

Esimene on proovimine, kaks järgmist osaliselt samuti, 4. valikut võib iseloomustada kui veaotsingut. Valik 7 (mingil määral ka 5 ja 6) üritab süstemaatilist läbitestimist, mille levinud liik on ekvivalentsiklasside analüüs.

Ekvivalentsiklasside analüüsi idee on selles, et sisendandmed jaotuvad töötluse suhtes enamasti rühmadesse, nii et ühes rühmas asuvaid andmeid töödeldakse ühtemoodi. See meetod püüabki leida selliseid sisend- ja väljundandmete klasse, et (1) kui klassist K leitakse viga, siis leitakse sama viga ka teistsuguste andmetega klassist K ja (2) kui klassis K viga ei ilmne, siis ei aita ka teistsugused andmed klassist K. Seega piisab sellest, kui valida üks punkt (test) klassist K, et testida kõiki samast klassist andmeid.

Kui ekvivalentsiklassid on valitud, tuleb nende põhjal koostada testolukorrad. Seda võib teha järgmiste põhimõtete alusel. Kui ekvivalentsiklass on

- järjestatud tõkestatud vahemik, siis võtta testid vahemiku seest, enne vahemikku ja pärast vahemikku
- järjestatud tõkestamata vahemik, siis võtta testid vahemiku seest ja väljast
- hulk, siis võtta testid hulga seest ja väljast
- tingimus, siis proovida mõlemaid variante (tingimus täidetud/täitmata)

Iga testolukorra jaoks koostatakse test. Õigeid klasse püütakse testolukorda sisse panna maksimaalselt, vigu ühekaupa. Testimise järjekord on selline: ekvivalentsiklassid, testolukorrad, testid.

Paljude ülesannete puhul on otstarbekas vaadelda sisendandmete vahelisi sõltuvusi - mingid olukorrad tekivad teataval sisendandmete kombinatsioonil. Sisuliselt on need kombinatsioonid sellisel juhul vaadeldavad ekvivalentsiklassidena.

Ekvivalentsiklassid tekitatakse nii sisendite kui ka väljundite jaoks.

1.1.3.2. Piirolukorrad

On leitud, et vigu esineb palju ekvivalentsiklasside piiridel, seega tasub teha piirolukordade teste. Selliste testide 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 –ks valitakse vähim väärtus, mis on arvuti või rakenduse 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
- Järjestatud väärtuste jada puhul testitakse piirjuhtusid ja nende ümbrust. Näiteks jada n_0, n_1, ..., n_K, ...n_M puhul, kus n_K on piirjuht, testitakse väärtusi n_0, n_1, n_{K-1}, n_K, n_{K+1}, n_{M-1}, n_M
- Kui sisendandmeteks on järjestamata hulgad, siis piirolukordi ei teki (miks?). Tavaliselt küll mingi järjestus leidub, iseasi kas see on oluline testitava ülesande seisukohast

Nagu ekvivalentsiklasside puhul tuleb piirjuhtusid vaadata nii sisendite kui ka väljundite jaoks.

1.1.3.3. Veaotsing (testimine eksperditeadmiste põhjal)

Kogenud arendaja oskab tõenäolisi vea kohti ette aimata. Tõenäoliste vigade leidmine võib sõltuda mitut sorti eelteadmistest, milleks on

- üldised teadmised
- teadmised konkreetse rakendusvaldkonna kohta
- teadmised riist- või tarkvarakeskkonna (näiteks konkreetse keele) kohta
- teadmised arendusmetoodika kohta
- teadmised konkreetse arendaja või tellija kohta jne

Kuna sellised teadmised kipuvad sõltuma konkreetsetest olukordadest (rakendusest, arenduskeskkonnast vms), siis ei saa neid hästi formaliseerida ega ka selles kursuses ammendavalt esitada. Valdkonna tüüpvigade analüüs võib olla iseseisva (diplomi-, magistri-) töö teema. Järgmises jaotises antavad küsimustikud võiksid olla näiteks programmeerimiskeeltele orienteeritud veaotsingu metoodikast.

Veaotsing on väga efektiivne vahend, mida võib kasutada süsteemselt või intuitiivselt. Esimesel juhul on ees küsimustikud kindla valdkonna kohta, mida süstemaatiliselt läbi vaadatakse. Sellisena on meetod kasutatav ka iseseisvates töödes. Teisel juhul jääb alles mittesüsteemsete meetodite puudus - testimine kipub olema juhuslikku laadi; eeliseks on aja kokkuhoid.

Kommentaar: testimine on loominguline tegevus ja intuitiivne ekspert on tavaliselt tulemuslikum kui algaja testija (tegutsegu algaja siis süstemaatiliselt või mitte). Intuitsiooni on kursusega raske edasi anda, see tekib mingil määral (loodetavasti)

töö/õppimise käigus. Paljud selle kursuse kuulajad on eksperdid oma paketi puhul või omas rakendusvaldkonnas.

1.1.3.4. Funktsionaalse testimise korraldus ja hinnang

Funktsionaalse testimise võib etappideks jagada järgmiselt:

1) eristada sisend- ja väljundandmete ekvivalentsiklassid, sealhulgas erinevate sisendite/väljundite kombinatsioonid,
2) leida igas klassis, nende piiridel ja vajadusel kombinatsioonidel esinevad testolukorrad,
3) leida igale testolukorrale vastavad sisend-väljundandmed,
4) identifitseerida testid, koostada testimise plaan,
5) testida, võrrelda tulemusi, hinnata.

Vaadeldud meetodite efektiivsuse järjestus (selles mõttes, et vea leidmise maksumus on väikseim; alates efektiivseimast) on tavaliselt järgmine: vea-otsing, piirjuhud, ekvivalentsiklassid, programmipõhine testimine.

Kokkuvõte funktsionaalsest testimisest.

- Idee: süstemaatiline testimine sisendi/väljundi (spetsifikatsiooni) põhjal
- Eeltingimused: vajadus; spetsifikatsioon on olemas; selle analüüs on teostatav
- Eelised: kasutajale arusaadavam; õigel arendamisel koostatakse testid juba spetsifikatsiooni koostamise ajal, mis võimaldab ühtlasi testida spetsifikatsiooni; kasutatav funktsionaalsus on süstemaatiliselt testitud
- Puudused: kui ekvivalentsiklassid on sõltuvuses, võib testimine olla mahukas. Spetsifikatsiooni pole alati olemas
- Tulemused: testikomplekt, mis katab funktsionaalsuse
- Suhe teistesse: võib kasutada iseseisvalt või koos teiste meetoditega
- Hinnang: hea
- Vahendid sõltuvad spetsifikatsiooni formalismidest. Kuna need pole unifitseeritud, on selle meetodi jaoks vähe üldlevinud testigeneraatoreid

1.1.3.5. Andmepõhine testimine

Ekvivalentsiklasside, piirjuhtude ja veaotsingu ideid saab kasutada ka programmipõhisel testimisel. Sel juhul tekitatakse sisendandmed programmi tekstis antud andmestruktuuride alusel, eristades siingi ekvivalentsiklasse ja piirolukordi. Erinevus funktsionaalsest testimisest on selles, et nüüd võivad nt piirolukorrad tekkida programmis või arenduskeskkonnas antud kitsendustest, nagu massiivi lubatud pikkus, reaalarvu võimalik väärtus vms. Väljundid võetakse nagu ikka ülesande püstitusest.

1.1.4. Muid testimise meetodeid

Eelmistes jaotistes toodud meetodid on lihtsad ja enim kasutatavad. Testimise meetodeid on siiski palju ja kriitilisemate ülesannete puhul tuleb otsida tugevamaid.

Vaadeldud meetodite üheks puuduseks on, et nad ei võimalda hästi hinnata tarkvara töökindlust, vigade arvu jne. Järgnevad näited võimaldavad seda mingil määral teha.

1.1.4.1. Testimine juhuslike andmetega

Võib öelda, et kes testimisest põhjalikumalt kuulnud pole, see just nii testibki - proovib mingite andmetega, kuidas töö läheb. Programmeerimise algusaastatel, mõnikümmend aastat tagasi oli see levinud meetod. Siis leiti, et nii jäävad olulised vead avastamata. Seepärast jäi kauaks püsima seisukoht, et juhuslike andmetega testimine ei ole soovitatav. Kaheksakümnendate aastate lõpus uuriti asja ja leiti, et see testimismeetod võib olla vägagi efektiivne, kuid ainult kindlatel tingimustel, nimelt

- tuleb hinnata tarkvara kasutusprofiili (milliste andmetega ja kui tihti seda tarkvara kasutatakse?)
- tuleb profiili raames genereerida juhuslikke (matemaatiliselt juhuslikke, mitte testija poolt “juhuslikult” valitud) andmeid vastavalt profiili sagedustele
- tuleb lisaks juhuslikele testidele kasutada ka teisi (vähemalt piirolukordade) teste

Nimetatud tingimustel annab juhuslike andmetega testimine häid tulemusi. Lisaks saab sellise testimise puhul hinnata tarkvara töökindlust ja prognoosida vigade arvu, mida eespool vaadatud meetodid hästi ei võimalda.

Meetodi puudusi: tulemuste hindamine võib olla mahukas; olulised “erilised” punktid võivad jääda katsetamata, nt on vähe tõenäoline, et 0 tuleks juhusliku arvuna. Selle vältimiseks rakendada kolmandat punkti ülalt.

Et meetodit efektiivselt kasutada, on soovitav süsteemi paralleelne realisatsioon (eelmine versioon, prototüüp, matemaatiline mudel vms), mis võimaldab kiiresti leida oodatavaid väljundeid. Vastasel juhul võib testimine nõuda palju aega.

1.1.4.2. Lisatud vead

Seda testimise meetodit võib iseloomustada järgmiselt.

- Idee: Meetodi ülesanne on prognoosida süsteemi jäänud vigu. Selleks lisab sõltumatu isik või rühm süsteemile juhuslikke, kuid mitte süntaktilisi vigu. Testimise käigus avastatakse nii lisatud kui ka tegelikke vigu. Eeldades, et vigade avastamise protsent on mõlemal juhul sama (kehtib ainult teatud tingimustel), saab prognoosida vigade arvu, mis jäid süsteemi peale testimist
- Eeltingimused: vajadus; olemas on programmi tekst, sõltumatu rühm/isik, programmi teksti analüüsi ja muutmise võimalused
- Eelised: vigade arvu prognoos
- Puudused: lisatud ja tegelike vigade avastamise protsent ei pruugi olla sama. Võib rikkuda tarkvara funktsionaalsust. Tehniliselt tülikas
- Tulemused: vigade arvu prognoos
- Suhe teistesse: koos teiste meetoditega
- Hinnang: kasutada erijuhtudel
- Vahendid: on tehtud juhuslike vigade generaatoreid, kuid need pole levinud

1.1.5. Testimise maht ja lõpetamine

Testimise resultatiivsust pole kerge hinnata, erinevalt näiteks programmeerimisest - testimine ei anna kergesti nähtavat tulemust. Seepärast pole ka testimise mahu üle otsustamine kerge (erinevalt jällegi programmeerimisest, kus teatud ülesanded peavad olema realiseeritud ja seda saab suhteliselt lihtsalt kontrollida).

Ideaalselt peaks maht sõltuma tarkvarale esitatud nõuetest. Testitakse seni, kuni need on rahuldatud. Praktiliselt tehakse nii vaid tõesti kriitiliste rakenduste korral (vähemalt loodetavasti tehakse ... mõeldes eelseisvale lennu- või laevareisile). Põhjusi on palju, näiteks nõudeid ja nende rahuldatust on raske hinnata; töö tuleb kiiresti üle anda; on olemas eelnev kogemus ja see määrab testimise mahu; rakendus ei ole kriitiline (kui midagi juhtub, keegi eriliselt ei kannata) jne. Seega kasutatakse testimise mahu määramisel ja lõpetamise otsustamisel tihti järgmisi kriteeriume:

- esimesed testid jooksid läbi
- kasutaja ei oska rohkem tahta
- testimiseks eraldatud aeg või raha on läbi
- eelmine kord testisime samapalju ja oli hea küll
- süsteemi üleandmise tähtaeg on käes
- paistab, et edasine testimine ei anna uusi vigu
- tahaks midagi muud teha
- ja nii edasi

Sõltuvalt olukorrast on sellised kriteeriumid mõnikord õigustatud, aga kaugeltki mitte alati. Näiteks kui testimiseks on eraldatud piisavalt aega ja vahendeid ning testijad on professionaalsed ning kohusetundlikud, siis võib kriteerium "testimiseks eraldatud aeg või raha on läbi" olla kasulik. Raskus on muidugi aja planeerimisel. Testimiseks vajalike ressursside hinnang kõigub vahemikus 20-80% programmi maksumusest. Mõned näited: hinnatakse väga ligikaudselt, et keskmise vastutusega süsteemide puhul võiks projekteerimise, programmeerimise ja testimise mahud olla sama suurusjärku. Seni suurima arvutiprojekti - aasta 2000 probleemi puhul hinnatakse testimise mahtu umbes 50 protsendile kogu projekti mahust.

Vaadatud testimismeetodid annavad lisavõimalusi testimise mahu määramiseks, näiteks:

- 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 sisse jääda. Testimise meetodid parandavad hea kasutamise korral efektiivsust (samade kulutuste juures leitakse rohkem vigu), süstemaatilisust (väldivad olukorda, kus mingi osa on põhjalikult testitud, mõni osa aga jäänud kahe silma vahele) ja hinnatavust (kasutaja või asjasse puutuvad muud osapooled, näiteks avalikkus, võivad vajadusel kontrollida, kas nende huvid on kaitstud).

1.1.6. Kontrollküsimusi ja ülesandeid

- Mis on test, testimine, viga, hea test, edukas test, silumine, verifitseerimine, valideerimine, sertifitseerimine?
- Mis on testimine programmi teksti põhjal? Mis on adekvaatsustingimused? Milline on viimaste võimsuse suhe, kuidas neid kasutada?
- Anda kokkuvõte meetoditest: idee, eeltingimused, eelised, puudused, tulemused, suhe teistesse, hinnang, vahendid
- Kuidas valida sisendeid?
- Kuidas hinnata väljundit?
- Millal testimine lõpetada?
- Kes on testijad?
- Missugune on testimise ja verifitseerimise vahekord?
- Kas teeadekvaatsus garanteerib programmi korrektsuse?
- Anda hinnang programmi keerukusele
- Kuidas toimub silmuste testimine?
- Funktsionaalne ja andmepõhine testimine, nende erinevus programmipõhisest testimisest, ekvivalentsiklasside analüüs, piirolukorrad, veaotsing
- Funktsionaalse testimise korraldus ja hinnang
- Andmepõhine testimine, testimine juhuslike andmetega, lisatud vead
- Testimise maht ja lõpetamine
- Mitu testi on vaja antud programmi lause- (haru-, tee- jne) adekvaatseks testimiseks?
- Millised on antud programmi puhul ekvivalentsiklassid, testitavad olukorrad, testid?
- Juhuslik katsetamine ja süstemaatiline testimine: eelised ja puudused. Millal üht või teist kasutada?
- Antud tarkvara ja selle rakendamise olukord. Kas ja milliseid testimise meetodeid kasutada?

1.2. Staatilised meetodid

Tavaliselt staatiliste meetodite puhul programmi/süsteemi ei täideta, pigem uuritakse selle teksti, spetsifikatsiooni, dokumentatsiooni või muud. Staatilised meetodid võivad hõlmata väga mitmesuguseid teemasid. Kogu kvaliteedijuhtimine - koos mitmesuguste protsessidega, nagu hange, tarne, hooldus ja nii edasi - võiks siia alla käia. Selles jaotises vaadeldakse siiski konkreetseid arendusele orienteeritud meetodeid. Lisaks võiks staatiliste meetodite alla liigitada ka programmide tõestamise ja infosüsteemi auditi. Oma eripära ja olulisuse tõttu on neid siiski vaadeldud edaspidistes jaotistes eraldi.

Mitme sellise meetodi esimeseks heaks küljeks on, et vigu saab leida juba enne programmi valmimist projekteerimise ja spetsifitseerimise ajal. Just neil etappidel tehtud vead maksavad hiljem väga valusalt kätte. Teiseks, kuna kõiki olukordi ei õnnestu tekitada ja vahetult testida, näiteks katastroofe või kauget tulevikku, siis on vaja mingil muul moel veenduda, et tarkvara nendes olukordades siiski töötab. Kolmandaks, testimisega saavutataval töökindlusel on piir. Kui töökindlus peab olema sellest piirist parem, tuleb rakendada teisi meetodeid. Neljandaks, mitmeid süsteemi

olulisi omadusi (näiteks hooldatavus - analüüsitavus ja muud hooldatavuse kriteeriumid) on raske testimise teel hinnata, rohkem sobivad selleks staatilised meetodid.

1.2.1. Küsimustikud

Eespool oli juttu veaotsingust - meetodist, mille puhul kasutatakse eksperditeadmisi mingilt alalt (programmeerimiskeskkonnast, ainevaldkonnast…) tõenäoliste vigade kiireks leidmiseks. Programmeerimiskeelte küsimustikud pakuvad võimalike vigade hinnanguid, mida võib kasutada autor, läbivaataja, analüüsija jne. Küsimustikud sisaldavad momente, millele muidu ei osata tähelepanu osutada. Küsimustikke on mitmesuguseid ja erineva mahuga, näiteks ühes NASA küsimustikus on üle tuhande küsimuse.

Küsimustikud on kasulikud nii vigade leidmisel kui ka selliste omaduste hindamisel, mida on raske testida (näiteks, hooldatavus või kasutajasõbralikkus). Paljud standardid esitavad sisuliselt küsimustikke standardi ala kohta.

Allpool on toodud väiksema mahuga küsimustik programmide kohta. Mõni küsimus on üldisem, mõni kitsam, mõne keele puhul langeb osa küsimusi sootuks ära. Küsimused on jaotatud rühmadesse.

Andmete kirjeldamine

- Kas muutujad on ilmutatult kirjeldatud (sõltub keelest, paljud translaatorid kontrollivad seda)?
- Kas vaikimisi atribuudid on antud õieti?
- Kas initsialiseeritud on õieti (eriti massiivid)?
- Kas muutujad on sarnaste või segadusse ajavate nimetustega, näiteks VOLT ja VOLTS, I1 ja O0?

Pöördumine andmete poole

- Kas pöördumisel on muutujal väärtus olemas?
- Kas indeks võib minna väljapoole lubatud piire?
- Kas indeksile omistatakse täisarv?
- Kas viida muutujale vastav mäluväli on määratud?
- Kas ühismälu kasutatakse õieti?
- Kas adresseerimine sõna piiril on korrektne?
- Kas ühised andmestruktuurid on kirjeldatud alamprogrammides ühte moodi?

Arvutusvead

- Kas arvutusi tehakse lubamatute tüüpidega?
- Kas koos kasutatakse eri tüüpi andmeid?
- Kas juhtub üle-/alatäitumist (tulemus kaob väiksuse tõttu ära)?
- Kas toimub jagamist nulliga?
- Kas muutujad on väljaspool sisulisi piire? Näiteks tõenäosus >1.
- Kas ebatäpsuste kuhjumisel võib tekkida oluline viga?
- Kas täisarvuline aritmeetika on õige (kas sulud on õieti pandud?)?
- Kas tehete prioriteedid on õiged?

Võrdluste vead

- Kas võrreldakse sobimatuid asju, näiteks teksti ja numbrit?
- Kas sobivaid, aga eri tüüpi muutujaid võrreldakse õieti?
- Kas spetsifikatsioon on võrdlustehetesse õieti tõlgitud, näiteks 'suurim' , 'mitte väiksem kui'?
- Kas võrdlustehete prioriteedid on antud õieti?
- Kas võrdlustehte tulemus sõltub kompilaatorist?
- Kas võrreldakse reaalarvulist muutujat mingi kindla arvuga?

Juhtimise vead

- Kas igale BEGIN-lausele vastab END?
- Kas programm või moodul lõpetab töö?
- Kas tsükkel või kordus lõpetab töö?
- Mis juhtub, kui tsükli tingimus on kohe vale?
- Kui FOR alumine raja on suurem kui ülemine, kas siis tulemus vastab oodatule?
- Kas on vaadatud kõrvalekaldumisi (väga suured korduste arvud)?

Alamprogramm

- Kas alamprogrammi väljakutsel muutujate arv (atribuudid) sobib?
- Kas mõõtühikud väljakutsutavas ja väljakutsuvas programmis on samad?
- Kas parameetrid on sisuliselt samas järjekorras?
- Kui sisendpunkte on mitu, kas siis kõigile muutujatele antakse igal juhul väärtused?
- Kas alamprogramm muudab sisendparameetreid?
- Kas konstante kantakse üle?
- Kas globaalsete muutujate atribuudid on igas moodulis samad?

Sisend ja väljund

- Kas faili atribuudid on õiged?
- Kas avamine/sulgemine on õieti korraldatud?
- Kas kõik vajalikud failid on avatud/suletud?
- Kas pöördumine ühtib kirjeldustega?
- Kas sisend-/ väljundpuhvrite pikkused on õiged?
- Kas faili lõputingimus on antud korrektselt?
- Kas veaolukorrad/katkestused on õieti käsitletud?
- Kas tekstides on sisulisi/grammatilisi vigu?

Mitmesugust

- Kas translaatori hoiatusi ja teateid on uuritud?
- Kas vigaseid sisendeid on proovitud?
- Kas kõik funktsioonid on olemas?
- Kas testida veel? Kui leiti viga, võib veel leida; kui ei leitud, siis on võib-olla halvasti testitud

1.2.2. Läbivaatused ja hindamised

Paljudest võimalikest vaatame töö analüüsi autori poolt, läbivaatust, programmeerija hindamist.

1.2.2.1. Läbivaatus

Kvaliteedijuhtimises on levinud meetodiks mitut tüüpi ühised arutelud. Standard *ANSI/IEEE Std 1028-1988 IEEE Standard for Softvare Reviews and Audits* eristab juhtkonnapoolset hindamist (*management review*), tehnilist hindamist (*technical review*), tarkvara inspektsiooni (*software inspection*), läbivaatust (*walkthrough*) ja auditit (*audit*), mida teeb sõltumatu osapool, kes jälgib vastavust kehtestatud nõuetele. Kõigil meetoditel on see ühine omadus, et nad peavad efektiivseks läbiviimiseks olema teataval määral planeeritud, organiseeritud ja juhitud. Vaatame detailsemalt läbivaatust, mille ette lisatakse tavaliselt sõna “struktuurne” (*structured walkthrough*).

Läbivaatus on suunatud toote kvaliteedi parandamisele, selle tulemused ei tohiks mõjutada töötajate palka, heaolu, ametikõrgendust vms. Läbivaatuse idee on toote (spetsifikatsioon, projekt, dokumentatsioon,...) ühisarutelu kindlate reeglite järgi. Sõna “struktuurne” rõhutab esiteks seda, et efektiivsuse tagamiseks peab läbivaatus olema organiseeritud, ja teiseks seda, et struktureerimata süsteeme ei paranda ükski läbivaatus.

Läbivaatusel on palju soodsaid külgi:

- vigu saab leida varastel arenduse etappidel, millal nad on kõige kallimad
- ta on parim viis vigade vähendamiseks (suurusjärk)
- kontaktid rühmas paranevad
- tootlikkus ja kvaliteet paranevad
- ühe osavõtja lahkumisel saab teda asendada

Läbivaatuse probleeme:

- rühma liikmed võivad olla erinevatest osakondadest
- rühma liikmed võivad olla erinevad: kõrge IQ-ga, kannatamatud, konservatiivsed, vähe huvitatud "reaalsest maailmast", eelistavad eraldatust jne
- kellelegi ei meeldi, kui teda kritiseeritakse

Eeltingimused:

- kõigil rühma liikmetel olgu ettekujutus sellest, mida neilt oodatakse
- koostööõhkkond
- materjalid on ette valmistatud ja kätte jagatud
- osavõtjad on nendega tutvunud, nt igaühel on üks positiivne ja üks negatiivne kommentaar

Osavõtjad ja nende rollid:

- esitaja (ei pruugi olla toote autor)
- koordinaator (juhataja)
- sekretär
- liikmed: juurutamise ja standardite eksperdid, kasutaja esindaja (eriti alguses ja lõpus)

Mainitud rollid võivad olla ühendatud. Otsese ülemuse viibimine läbivaatusel pole soovitav.

Läbivaatuse korraldus:

- nii palju ettevalmistusi (tekstid, dokumendid) kui vaja ja nii vähe kui võimalik
- rusikareegel: ettekande pikkus on 30..60 min

Igat liiki arutelusid, eriti aga läbivaatuse tüüpi demokraatlikke kogunemisi, võivad häirida osavõtjatevahelised teadvustatud või teadvustamata mängud. Mäng on tegevuste jada, millel on kaks eesmärki: varjatud ja avalik. Näiteks korduvalt kasutatud sõnaühend "Jaa, aga ... " viitab mängule. Kuna mängud rikuvad normaalset arutelu, siis on oluline

- teada, et mängud on olemas
- osata aru saada, et mäng käib
- osata see õigel ajal ära lõpetada

Läbivaatuse tulemusteks peaksid olema leitud ja parandatud vead ning parem süsteem, samuti allakirjutatud protokoll.

Juhtkond võiks läbivaatuste puhul jälgida, et peetaks kinni järgmistest reeglitest:

- soodustada arutelusid
- jälgida, et osavõtjad on ette valmistunud
- usaldada rühma (lubada lobisemist)
- jälgida ajalimiiti
- jälgida vastutust (nt allkirjastamine)
- jälgida formaalseid nõudeid
- veenduda, et teatakse standardeid
- vältida võimalusel juhtkonna esindajate viibimist arutelul

1.2.2.2. Autori analüüs

Autori analüüsi teeb enamik programmeerijaid ja arendusrühmi (firmasid). See on suhteliselt odav meetod, mis tavaliselt leiab vigu. Seega on meetodit soovitav kasutada. Kui seda veel ei tehta, on soovitav, et firma meetodit tutvustaks.

Meetod võib siiski olla ebapiisav. Nagu autori sooritatud testimise puhul võib siingi olla mitu põhjust:

- autori (otsene) motivatsioon on lõpetada töö, mitte leida vigu
- autor pole piisavalt enesekriitiline, ta ei taha oma tööd “lõhkuda”
- autor jälgib oma loogikat ega suuda prognoosida kasutaja võimalikke tegevusi

1.2.2.3. Programmeerija hindamine

Meetodi idee on anda programmeerijaile arusaamine tema tugevatest ja nõrkadest külgedest. Hindamine on anonüümne, ei mõjuta rühma liikmete palka ja käekäiku. On vaja rühma, kuhu kuuluks 6…12 inimest. Igaüks neist valib enda tehtud töödest oma arvates parima ja halvima toote, näiteks programmi, projekti, ütlemata, kumb on halvem ja kumb parem. Igaüks hindab teiste pakutud kahte toodet. Tagatud peab olema hinnatavate anonüümsus. Hinnata tuleks toodete arusaadavust (ka projekti

puhul), vastavust projektile või spetsifikatsioonile, muudetavust. Kas hindaja oleks ise rahul, kui see oleks tema töö?

Meetodi eeltingimus on rühma ja programmi olemasolu. Kuna protseduur nõuab aega, peab selle järele olema ka vajadus. Tarvis on initsiaatoreid, kes hindamisega tegelevad. Tulemusena saab iga osaleja hinnangu oma kvalifikatsioonile.

1.2.3. Kontrollküsimusi ja ülesandeid

- Anda kokkuvõte meetoditest: idee, eeltingimused, eelised, puudused, tulemused, suhe teistesse, hinnang, vahendid
- Küsimustikkude liigitusi, programmeerimisele orienteeritud küsimustiku struktuur
- Loetleda arutelude ja hindamiste liike
- Läbivaatus, selle plussid, miinused, eeltingimused, osavõtjad, korraldus, mängud, suhe juhtkonda
- Töö analüüs autori poolt, programmeerija hindamine

1.3. Töökindluse oluline suurendamine

Mõnes valdkonnas nõutakse suurt töökindlust, näiteks NASA lennujuhtimisprogrammide veakindluse nõue on kümnetunnisel lennul 10^{-10} viga tunnis (ehk üks viga miljoni aasta jooksul). Kuidas seda saavutada, sest süsteemi ei saa ju miljon aastat katsetada? On leitud, et testimisega on saavutatav tase 10^{-4} viga tunnis ehk umbes üks viga aastas. Kõik sellised hinnangud on ligikaudsed. Töökindluse parendamise vahendeid on palju, näiteks:

- dubleerimine. Paralleelselt pannakse käima mitu programmi ja võrreldakse tulemusi, enam levinud vastused loetakse õigeks. Meetod, millelt palju loodeti ja mis õigustab ennast hästi riistvara puhul, ei ole niisama kasulik tarkvara korral (üks põhjus on selles, et inimlik loogika jälgib tihti samu radu ja paralleelsetes arendustes tehakse ühesuguseid vigu)
- veapuu analüüs - ehitatakse ja/või puu. Alustatakse suurest veast, mida tahetakse vältida, vaadatakse selle vea eeltingimusi, eeltingimuste eeltingimusi ja nii edasi. Meetod toimib hästi tehniliste süsteemide puhul, aga selle kasulikkus tarkvara puhul pole selge
- formaalsed meetodid, sealhulgas programmide tõestamine

On hinnatud, et formaalsed tõestusmeetodid võimaldavad taset 10^{-7} … 10^{-8} viga tunnis, mis pole piisav, aga on siiski parem kui testimine.

Ajalooliselt tekkisid tõestusmeetodid kuuekümnendatel aastatel, kui leiti, et programmide töökindlust on vaja parandada. Avaldati arvukalt töid, kus tõestati lihtsamaid programme. Siis aga leiti, et (1) ka toodud tõestustes on vigu, (2) tõestus on väga töömahukas, (3) testimine, küsimustikud, hea projekteerimine jne aitavad töökindlust oluliselt tõsta. Tulemusena pöörati tõestamisele vähem tähelepanu, kuni 90. alguses järgnes uus tõus eriti seepärast, et (1) olid tekkinud kõrgendatud töökindlusnõuetega rakendused, (2) tõestusmeetodid olid arenenud praktilise kasutatavuseni, (3) oli tekkinud tõestust toetav tarkvara.

Programmide tõestamine

- Siht. Näidata, et programm vastab spetsifikatsioonile
- Idee. Luuakse loogiline arvutus (valemid, aksioomid, tuletusreeglid). Selle arvutuse terminites kujutatakse spetsifikatsiooni (sisendid ja väljundid) ning programmi. Tõestatakse, et lähtudes antud sisenditest ja kasutades antud programmi jõutakse nõutud väljunditeni (tavaliselt ka, et programm lõpetab töö)
- Eeltingimused. Spetsifikatsioon, programm, loogikavahendid, vajadus, ressursid, soovitavalt toetav tarkvara
- Eelised. Suurem töökindlus, formaalne korrektsus, ainuke viis tsüklite korrektsuse formaalseks põhjendamiseks
- Puudused. Töömahukas, sobib halvasti suurte süsteemide jaoks, ei välista spetsifikatsiooni vigu (samuti arenduskeskkonna ja muid vigu), tsükleid on tehniliselt raske tõestada
- Tulemused. Programm tõestatakse spetsifikatsiooni suhtes
- Suhe teistesse. Kasutatakse koos teiste meetoditega
- Hinnang. Kasutada erijuhtudel. Vähekriitiliste puhul pole otstarbekas
- Vahendid. On tehtud tõestamist toetavat tarkvara, kuid see pole levinud

Töömahu vähendamiseks on pakutud jaotada süsteem tuumaks, mida tõestatakse, ja ümbruseks, mida ei tõestata. Tuum kindlustab, et halvad asjad ei toimu, aga ei kindlusta, et head asjad toimuvad. Ümbrus ei ole nii kriitiline töökindluse nõuete suhtes.

Nagu mainitud, on tõestamise üheks probleemiks, et selle objekt (üleminek spetsifikatsioonilt programmile) on ainult üks etapp arenduses. Arendusprotsessis on ka teisi etappe ja kui nende töökindlus on madal, pole ühe etapi tõestamisest palju kasu. Illustreerime mõningaid etappe tabeliga.

TASE	**VAHENDID**	**MÄRKUSI**	**RASKUSASTE**
Spetsifikatsioon	Läbivaatus	Ei ole formaalset eellast, seega pole millegi suhtes tõestada	Spetsifikatsiooni korrektsus on peamisi probleeme
Realisatsiooni-projekt	Projekti tõestus	Sisulist kontrolli saab teha vaid siis, kui spetsifikatsioon oli formaalne, tavaliselt mõne aspekti suhtes. Saab ka kontrollida, et poleks vasturääkivusi	Sõltub formalismist
Programmi kood	Programmi tõestamine	Väga mahukas	Raske

Tõestusmeetodid	Loogika	Leidub häid meetodeid ja formalisme. Ei sõltu üksikutest programmidest	Keskmine Ühekordne töö
Objekti kood	Programmi või kompilaatori tõestamine		Raske
Täitmine	Mikroprogrammi/ riistvara tõestamine		Ühekordne töö

Lõpuks kommentaar selle kohta, kuidas saavutatakse 10^{-10} töökindlus riistvara puhul (miks see ei toimi tarkvara korral?).

- Dubleerimine / hääletamine (vt ülal)
- Kõrge töökindlusega seadmed
- Lihtsad funktsioonid

1.3.1. Kontrollküsimusi ja ülesandeid

- Teha kokkuvõte meetoditest: idee, eeltingimused, eelised, puudused, tulemused, suhe teistesse, hinnang, vahendid
- Saavutatav töökindluse tase seniste meetoditega, selle olulise suurendamise võimalusi
- Programmide tõestamine, ajalugu, probleeme, võimalusi jne
- Tõestamise tasemed
- Võrdlus: töökindluse saavutamine riist- ja tarkvara puhul

1.4. Kontrolli korraldus

Seni oleme vaadelnud kontrolli meetodeid. Neid võib arenduse käigus mitmeti kasutada. Selles jaotises vaatame erinevaid viise, kuidas kontrolli korraldada, sealhulgas kontrollimeetodite kasutamist.

1.4.1. Kontrolli korralduse võimalusi: ülevaade

Alustame ülevaatega sellest, kuidas praktiliselt tihti kontroll on korraldatud. Vaatame olukordi, millal lihtsamad skeemid ei toimi.

1.4.1.1. Lihtsamaid skeeme

Kontrolli korraldus nagu ka kasutatavad meetodid sõltuvad nõudmistest arendatavale süsteemile, firma üldisest töökorraldusest, firma ambitsioonidest, näiteks soovist saada uusi tellimusi jne. Järgnevalt mõni näide.

Kui kasutaja ja arendaja on sama isik (võib olla firmas erinevates rollides) ja toode pole vastutusrikas, toimib “ise teen, ise testin” korraldus.

Kui kasutaja ja arendaja küll erinevad, mistõttu eelmine skeem enam ei toimi, kuid arendatavale süsteemile ei esitata suuremaid nõudmisi ja ka firma töös pole korraldusele erilist tähelepanu pööratud, siis võib toimida tellija (tellija esindaja) ja programmeerija vahel järgmine suhtlus:

- 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

Vastutusrikka tarkvara puhul, mida hakkab kasutama palju kasutajaid, eelmine skeem enam ei toimi, sest seda peavad katsetama erinevad kasutajad. Reaalselt on kasutatud sellist skeemi:

- arendajapoolne testimine
- rakendus- ja testimiseksperdi(nt toetusrühma-)poolne testimine
- kasutajate rühma testimine
- igalt etapilt võib minna tagasi eelmistele etappidele, kui leiti vigu

Milline meetod on parim? Nagu mainitud (kvaliteet on suhe nõuete, toote ja protsessi vahel), ei saa sellele küsimusele midagi muud teadmata vastata. Tuleb valida antud olukorras sobivaim korraldus.

1.4.1.2. Vajadus keerukaks korralduseks

Ilmselt olid eelmises jaotises toodud meetodid reastatud kasvava tugevuse järjekorras. Kas viimasest meetodist piisab ka kriitilistes rakendustes? Osutub, et paljudes olukordades mitte. Põhjusteks, mis sunnivad kasutama keerukamat töökorraldust, võivad olla

- keerukas toode. Sellisel juhul on otstarbekas alustada kontrolli arenduse algetappidest ja jaotada see hallatavateks osadeks
- kriitilise töökindlusega toode. Testimine tuleb jaotada etappideks, kasutada erimeetodeid
- organisatsiooniliselt lahutatud arendusetapid. Sel juhul peab näiteks olema korraldatud analüüsijate, projekteerijate ja programmeerijate koostöö
- organisatsiooniliselt keerukas kasutaja. See nõuab erinevate kasutajarühmade koostööd
- mitmeetapiline mahukas testimine, mis vaheldub arendustegevustega. See nõuab vigade ja nende paranduste jälgimist ning regressioontestimist (kas vanad testid töötavad peale parandusi?)
- ja nii edasi

Need tegurid esinevad tihti koos (püüdke tuua näiteid). Mõnel juhul piisab eelmises jaotises toodud korraldusskeemide detailiseerimisest. Vaatleme korraldust, mis püüab lahendada toote ja arenduse keerukusega seonduvat.

1.4.2. Elutsükli V-mudel: Arendusprotsessi integreeritud kontroll

Nüüdisaegses süsteemiarendusprotsessis alustatakse kontrolliga võimalikult varakult. Mida varem vead leitakse, seda odavam on neid parandada. Seepärast projekteeritakse iga arendusetapi käigus ka testid. Arendusetappidele vastavad eri tüüpi testid, tekib nn tarkvara elutsükli V-mudel. Kuna elutsüklid erinevad, erinevad ka V-mudelid. Üks levinum on järgmine:

- süsteemi üldprojektile (sealhulgas tarkvara) vastab süsteemitestimine
- tarkvara nõudmiste spetsifikatsioonile vastab valideerimistestimine
- tarkvara realisatsiooniprojektile vastab integratsioonitestimine
- tarkvara kodeerimisele vastab mooduli testimine

Tegemist on nelja erineva testimisetapiga, mis vastavad neljale süsteemiarenduse olulisele etapile. Igal arendusetapil luuakse oma testiprojekt. Iga testimisetapi sisend on tarkvara konfiguratsioon (sh dokumentatsioon) ja testimise konfiguratsioon (sh testimise dokumentatsioon). Testimise väljundiks on leitud vead ja töökindluse prognoos.

Vaatame testimise etappe lähemalt, alustades madalama taseme testidest.

1.4.2.1. Mooduli testimine

Enamik 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?). On leitud, et suureneva maksumuse järjekorras võib meetodid reastada järgmiselt:

- testimine kasutaja andmetega (peavad kindlasti töötama)
- läbivaatused (avastavad vigu vara)
- vea oletus (kui testija on ekspert)
- piirjuhud (vigade kuhjumise kohad)
- ekvivalentsiklassid (lähtuvad kasutamisest)
- programmi põhjal testimine
- lisatud vead
- tõestamine

Ilmselt on otstarbekas teha kõigepealt hinnaefektiivsemaid teste.

1.4.2.2. Integratsioonitestimine

Kui mooduli taseme testid töötavad, tuleb moodulid kokku panna ja testida. Selleks on mitu meetodit:

- “Kõik kokku ja test” võib töötada lihtsate süsteemide korral; keerukate puhul on vigu raske lokaliseerida
- alt üles integreerimistestid
- ülalt alla integreerimistestid
- segameetod (võileib)

Alt üles integreerimise puhul asendatakse osa väljakutsuvaid mooduleid nn draiveritega. Eeliseks on see, et korraga saab töötada palju inimesi (rühm), igaüks testib sõltumatult oma osa. Puudus: süsteemist ei teki tervikpilti viimase hetkeni. Tegevuse järjekord:

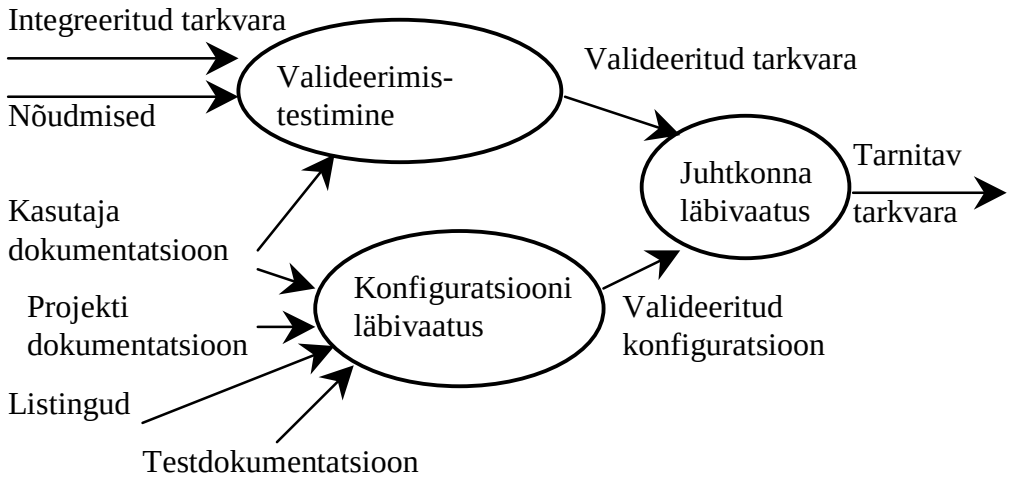
- moodusta üheskoos testitavate moodulite kogumid (klastrid)
- kirjuta klastrite jaoks draiverid (draiver võib näiteks välja kutsuda ainult mooduli, saata parameetri, printida parameetri, saata ja printida)
- testi
- asenda draiverid tegelike moodulitega

Ülalt alla integreerimistestide puhul kirjutab juhtmoodul väljakutsutavate moodulite asemel nn lühised (asendavad programmid; näiteid: prindi mooduli nimi; prindi üleantav parameeter; tagasta vastusparameeter; otsi tabelist vastus ja tagasta väljund vastavalt sisendile). Testimise käik: põhimoodul - lühised - test - asenda - regressioonitest. Eelis on see, et midagi on juba algusest saadik olemas, side kasutajaga on parem. Süsteemist saab kohe üldpildi, mis on eriti kasulik arendusjuhtidele, kellel on kogu aeg ettekujutus olukorrast olemas.

Segameetodi korral hakatakse liikuma nii ülalt alla, kasutades lühiseid, kui alt üles, kasutades draivereid. Selleks võetakse mingi vahenivoo ja testitakse selle tasemini ülalt alla ja alt üles (tase tuleks valida selline, kus hakkab selguma juba sisulisi tegevusi). Võimaldab nii arendusest pildi saamist kui ka paljude arendajate koostööd.

1.4.2.3. Valideerimistestimine

Valideerimine vastab küsimusele: ”Kas ehitasime õige tarkvara?”. Tarkvara katsetatakse nõudmistele vastavuse suhtes. Valideerimistestimist võib kujutada järgmiselt:



Valideerimise etappi võivad kuuluda alfa-testimine (kasutaja poolt/andmetega arendaja keskkonnas) ja beeta-testimine (kasutaja poolt/andmetega kasutaja keskkonnas).

1.4.2.4. Süsteemi testimine

Üldjuhul kuulub tarkvara süsteemi koosseisu. Ka nõuded tarkvarale võivad olla vigased. Seega ei anna valideerimistestid õigsuse garantiid, vaid testida tuleb süsteemi tervikuna. Süsteemitestide hulka kuuluvad peale tavalise tööre□iimi testide tihti ka jõudlus-, taluvus-, taastamis- ja turbetestid.

Jõudlustestid testivad maksimumkoormusi, mida süsteem peab taluma.

Taluvustestides katsetatakse süsteemi käitumist üle normaalse ulatuva koormuse korral. Süsteem peab ära hoidma suured õnnetused. Lubatud on mõne funktsiooni või kasutaja blokeerimine.

Taastamistestid näitavad, kuidas süsteemi töö taastub peale katkestusi (nt katkestused peale taluvusteste, voolukatkestust, mõne seadme riket jne).

Turbetestid näitavad, kuidas süsteem on kaitstud hooletuste ja rünnete vastu.

Korralduse probleemidest märgitagu, et süsteemi ei saa testida, kui pole spetsifikatsiooni. On ka eriolukordi, mida on kulukas või võimatu testida. Näiteks võib olla väga kulukas reaalselt testida laeva avariisüsteeme. Sellisel puhul saab rakendada

- läbivaatusi
- tõestamist
- töökindluse mudeleid
- simuleerimist
- vigade arvu prognoosi

Tehtud ülevaade peaks andma vastuse ka küsimusele, miks peaks testimise jagama etappideks (kihtideks). Näiteks võib süsteemi testimine nõuda hinnalisi seadmeid ja testimise tulemuseks võib olla nende häving. Mida hilisem testimise etapp, seda rohkem ressursse see nõuab. Seega tuleks võimalikult palju tööd teha ära alguses. Etappideks jagamine annab organisatsiooni, mis sellise korralduse efektiivselt ja hallatavalt teostab.

1.4.3. Keerukamad korralduse raamistikud

V-mudel väga kasulik ja soovitatav kasutamiseks esimese põhjalikuma korralduse elemendina. Vastutusrikastel juhtudel on ta siiski ebapiisav, kuna (1) ta käsitleb peamiselt arendust, kontroll on aga vajalik kõigis tarkvara protsessides, (2) ta ei käsitle testprotsessi enese arengut ja hindamist ja (3) ta ei ole piisavalt detailne. Kogu testimisprotsessi haldamist tervikuna ja selle parendamist käsitlevad mitmed kursuse teises osas käsitletavad standardid, näiteks EVS-ISO/IEC 12207:1998. Infotehnoloogia – Tarkvara elutsükli protsessid. See standardid annab soovitusi testimise haldamiseks kõigis tarkvara elutsükli protsessides. Toodud soovitused on paratamatult üsna üldsõnalised.

Spetsiifiliselt testimisele orienteeritud lähenemise näitena toome testprotsessi täiustamise (Test Process Improvement, TPI) mudeli ja metoodika, mis on avaldatud raamatus (Koomen and Pol 1999). Raamat on praktiline juhend TPI mudeli kasutamiseks. Mudeli eesmärk on analüüsida testimise olemasolevat protsessi ja

näidata selle tugevaid ning nõrku külgi. Seda võib rakendada tarkvarale, aga ka laiemalt infotehnoloogiasüsteemidele. Mudeli põhilised ideed on järgnevad.

Testimisprotsess jaotatakse kahekümneks võtmealaks. Võtmealad on näiteks testimise strateegia, meetrikad ja nii edasi. Et jälgida küpsustaset, on iga võtmealaga seostatud selle tasemed ja kontrollküsimused. Mudel sisaldab ka soovitusi ja instruktsioone teatava tasemeni jõudmiseks. Lisaks nendele põhikomponentidele sisaldab mudel ka metodoloogia tarkvaratoodete struktuurseks testimiseks (TMap) ja testimise küpsuse maatriksi (Test Maturity Matrix). Viimane aitab näidata seoseid ja taseme erinevusi erinevate võtmealade vahel.

Mudeli ülevaade on antud aadressidel http://www.iquip.nl/tpi ja http://www.iquip.nl/testen/index_testen_engels.html inglise, saksa ja hollandi keeles.

Ettevõte, kes soovib parendada oma tooteid ja teenuseid, peaks kaaluma, kas TPI mudel eraldi on piisav seatud sihtide saavutamiseks. Mudel keskendub testimisele, mis on üks paljudest olulistest protsessidest. Näiteks esitab rahvusvaheline standard EVS-ISO/IEC 12207:1998 kokku seitseteist tarkvara elutsükli protsessi. Kindlasti saab TPI mudelit kasutada väärtusliku täiendusena ja täpsustusena sellistele laiaulatuslikele standarditele.

1.4.4. Kontrollküsimusi ja ülesandeid

- Iseloomustage tarkvara kontrolli korralduse lihtsamaid skeeme. Milline neist on parim?
- Millal tekib vajadus keerukamaks korralduseks?
- Iseloomustage kontrolli projekteerimise loogikat: olukord ja testitavad objektid, meetodid, tegevused
- V-mudel, arendusprotsessi integreeritud kontroll
- Testimise etapid: ühiku-/mooduli-, integratsiooni-, valideerimise- ja süsteemitestid
- Testimise ja kontrollimeetodite efektiivsuse võrdlus ja soovitused kasutamiseks
- Mida teha, kui testida ei saa?
- TPI mudeli põhilised komponendid
- Võrdlus: V-mudel, EVS-ISO/IEC 12207:1998 ja TPI mudel
- Kokkuvõte meetoditest: idee, eeltingimused, eelised, puudused, tulemused, suhe teistesse, hinnang, vahendid

2. Kvaliteedijuhtimine, standardid, normid

Esimeses osas vaadeldi tarkvara kontrolli meetodeid (testimine, staatilised meetodid jm) ja korraldust. Kvaliteedijuhtimises on nendel oma koht, kuid on ka omad piirangud. Kvaliteeti ei saa sisse testida. Halvasti arendatud süsteemi pole võimalik kontrolli abil heaks muuta. Lõpptulemus sõltub kogu arendusprotsessist, sealhulgas sellistest asjadest nagu

- tarkvara arenduse meetodid
  - üldised: struktuurne analüüs, disain…
  - spetsiaalsed: veakindel programmeerimine, n-versiooniline programmeerimine, transaktsioonide haldamine, taastamisvahendid…
- riistvara vahendid (soodustavad)
  - võimas keskkond
  - töökindlust toetavad vahendid
- tarkvara arenduse vahendid
  - CASE, hea keel, dokumenteerimisvahendid...
- läbivaatused, verifitseerimine, testimine, valideerimine ...
- organisatoorsed vahendid
  - üldine kvaliteedijuhtimine, aruandlus, sertifitseerimine…
- tarkvaraspetsiifiline kvaliteedijuhtimine
  - atribuudid, meetrikad, mõõtmine, ISO 9000-3, SEI CMM
- standardid
- elutsüklid, analüüs, disain, testimine, turve…

Teises osa loogika on järgmine. On vaja töötada paremini, teenida rohkem. Peame pakkuma kvaliteetsemat toodet/ teenust. Tekivad küsimused

- Mis on kvaliteet? Kuidas seda juhtida (jaotis 2.1)?
- Nõudmised: standardid (2.2), kvaliteedikriteeriumid (2.3)
- Audit: kvaliteedi sõltumatu tugi (2.4)

2.1. Kvaliteet

Kvaliteet näib olevat midagi, mida enamik inimesi heameelega tahaks näha (“süsteem võiks paremini töötada”), kuid mis näib laiali valguvat niipea, kui püütakse sellest täpsemalt rääkida. Kvaliteeti võibki käsitleda mitut moodi. Kirjeldan allpool ühte enam levinud kvaliteedi mõistet ja sellele vastavaid kvaliteedijuhtimise tegevusi, samuti süsteeme kvaliteedi hindamiseks.

2.1.1. Kvaliteet ja kvaliteedijuhtimine: mõisted

Kvaliteet on toote, teenuse või protsessi omaduste kogum, mis rahuldab määratletud või eeldatavaid vajadusi. Kvaliteeti saab juhtida, kui vajadused on määratletud (on olemas nõuded; siis on kvaliteet vastavus nõuetele). Keerukate toodete puhul - nende hulka kuulub ka tarkvara - on toote/teenuse ja nõuete vastavust raske kontrollida,

seepärast on siin oluline, et arendusprotsess oleks jälgitav ja vastaks samuti kindlatele nõuetele. Seega võib öelda, et kvaliteet on toote/teenuse, nõuete ja protsessi vaheline suhe.

Selline kvaliteedi mõiste on nii lai, et hõlmab suurt hulka juhtimistegevusi. Tõepoolest, eriti USA-s ja Jaapanis (A. Feigenbaum *Quality - a way of managing the organisation*) ongi kvaliteeti määratletud kui juhtimist. Et selline käsitlus venitaks teema laiaks, piirdume kvaliteedi aspektidega, mis on levinud Euroopas ja leidnud kajastamist ISO 9000 standardite seerias (sh tarkvara käsitlevas standardis ISO 9000-3).

Tarkvara arenduse tulem (toode, teenus) hõlmab mitmesuguseid komponente, mis kõik võivad olla kvaliteedijuhtimise objektid, näiteks

- arenduse käigus hangitud infotehnoloogiavahendid: riistvara, standardtarkvara, sideseadmed
- arenduse käigus tehtud töö: täitja arendatud tarkvara; installatsioonid, kohandamised, muudatused; andmehõive
- muudatused tellija organisatsioonis, töökorralduses...
- projektdokumentatsioon kasutamise kohta (kasutajajuhendid); objektsüsteemi kohta; loodavate objektide kohta (programmi/testimise dokumentatsioon); arenduse kohta
- metoodika: tulemuste kasutamine; tulemuste edasiarendamine; uute arenduste tegemine
- vahendid hoolduseks, muudatusteks, arenduseks
- teadmised projekti tulemuste kasutamisest; objektsüsteemist (infovoogude analüüs või vajalikud muudatused seadusandluses); projektist; arendusest
- õigused tööks, arendamiseks, levitamiseks

Vajadusi ja nõudeid võib liigitada mitmeti. Kõigepealt võivad nad olla määratud (sõnastatud, lepingus esitatud jne) või eeldatud (“tahaksin seda”, “tavaliselt tehakse nii”). Nõuded tulenevad mitmest allikast: spetsifikatsioonist, standardist, normist, seadusest, heast tavast, eetikast jne. Nõuded võivad olla orienteeritud kõikidele eelmises jaotises antud IS arenduse tulemitele ja ka arendusprotsessile. Sel juhul on otstarbekas eristada sisemist ja välist klienti.

Infosüsteemi areng on pidev protsess, mida tuleb jaotada (alam)projektideks ja etappideks, et seda protsessi paremini hallata. Üksikutes projektides eristatakse tüüpiliselt selliseid etappe nagu spetsifitseerimine, analüüs, disain jne. Etappideks jaotus ja etappide nimetused, samuti etappide organisatsioon (lineaarne, spiraalne, V-kujuline vms) võivad mudeliti erineda. Antud teema jaoks on oluline, et sellised etapid on olemas ja igale etapile vastaksid kindlad tegevused, näiteks süsteemitöö, projektijuhtimine, kvaliteedijuhtimine ja koolitus.

Osa süsteemiarenduse tegevusi ei sõltu projekti etappidest (konfiguratsiooni juhtimine). Ka nende tegevustega kaasnevad kvaliteedinõuded. Lõpuks on kogu kvaliteedijuhtimise protsessis iseseisvaid, konkreetsetest projektidest sõltumatuid komponente (kvaliteedipoliitika formuleerimine). Toome järgnevalt olulisemad kvaliteedijuhtimise mõisted.

Kvaliteedipoliitika - organisatsiooni tippjuhtkonna ametlikult esitatud kvaliteedialased üldeesmärgid ja juhtnöörid.

Kvaliteedijuhtimine - üldise juhtimisfunktsiooni osa, mis määrab kindlaks ja rakendab kvaliteedipoliitikat.

Kvaliteedisüsteem - organisatsiooniline struktuur, vastutus, protseduurid ja vahendid kvaliteedi juhtimiseks.

Kvaliteeditõendus (*quality assurance*) - tegevuste kogum, mis on vajalik piisava usaldatavuse tagamiseks, et toode või protsess rahuldaks kvaliteedinõudeid.

2.1.2.Kvaliteedijuhtimise mitteformaalseid meetodeid

USA, Euroopa ning Jaapani arusaamad kvaliteedist on olnud mõnevõrra erinevad. Viimasel ajal on need siiski ühtlustumas. USA ja Euroopa kvaliteedijuhtimine on olnud rohkem orienteeritud tulemusele - oluline on lõpptulemuse, mitte protsessi kvaliteet. Paraku ei taha see põhimõte keerukate toodete puhul hästi toimida. Jaapanis on kvaliteedijuhtimine orienteeritud protsessile, peetakse oluliseks, et kogu tootmine oleks kvaliteetne.

Erinevad on olnud ka arengumeetodid: kui USA-s ja Euroopas püütakse edu saavutada läbimurdeliste, kardinaalselt uute väljatöötluste ja otsustustega (innovatsioon), siis Jaapanis eelistatakse pidevat arengut ilma suurte hüpete või muutusteta.

USA-s ja Euroopas on kvaliteedijuhtimine olnud ajalooliselt enam orienteeritud statistilistele meetoditele, näiteks tootepartiide kontrollile. Jaapanis on see aga pigem iga töötaja ja iga hetke küsimus (*kaizen*).

Philip B. Crosby on sõnastanud kolm populaarset teesi.

- Tee kohe õieti (do it right first time)
- Ei ühtegi vigast toodet (*zero defects*)
- Kvaliteet on tasuta (*quality is free*)

Philip B. Crosby kvaliteedijuhtimise koostisosad.

1. Juhtkonna toetus.
2. Kvaliteedirühm.
3. Kvaliteedi mõõtmine.
4. Kvaliteedi maksumuse hinnang.
5. Töötajate kvaliteediteadlikkus, selle õpetamine.
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.

2.1.3. Kvaliteedi hindamise süsteeme

Kvaliteedi hindamiseks on loodud mitmesuguseid süsteeme. Selliste süsteemide alla võib paigutada ka kvaliteedi konkursid ja auhinnad, näiteks Euroopas EFQM (*The European Quality Award*) ja USA-s Malcolm Baldridge auhind (*The Malcolm Baldridge National Award for Quality*). Esimene nendest mõõdab järgmisi kvaliteedi komponente:

1) juhtimist 10%,
2) firma poliitikat ja strateegiat 8%,
3) inimeste juhtimist 18%/2 = 9% (punktid 2 ja 7 annavad kokku 18%),
4) vahendite hankimist ja kasutamist 9%,
5) protsesse 14%,
6) kasutaja rahulolu (hinnang) 20%,
7) firma töötajate rahulolu 18%/2,
8) mõju ühiskonnale 6%,
9) äritulemusi 15%.

Malcolm Baldridge auhinna määramisel arvestatakse järgmisi tegureid (sulgudes on maksimaalne punktide arv, seega ka osatähtsus kogu kvaliteedihinnangus):

1) juhtimine (100),
2) kvaliteediinfo ja selle analüüs (60),
3) strateegiline kvaliteedi planeerimine (90),
4) töötajate osalus (150),
5) kvaliteeditõendus (150),
6) kvaliteeditulemused (150),
7) kliendi rahulolu (300).

Kogusumma on tuhat, senised maksimaalsed hinnangud on olnud 700 piires.

2.1.4.Kvaliteedijuhtimise metoodikaid ja standardeid

2.1.4.1.EVS-EN ISO 9000-3:1999

ISO 9000 seeria standardid on kasutusel kõigis tööstusharudes, enamik neist on tõlgitud ka eesti keelde ja üle võetud eesti standardina. Kui kvaliteedihaldus laieneb ka toote projekteerimisele, rakendatakse standardit ISO 9001. Meid huvitav EVS-EN ISO 9000-3:1999 sisaldab suuniseid ISO 9001:1994 kohaldamiseks tarkvara väljatöötamisele, tarnimisele, installeerimisele ja hooldusele.

ISO 9000 standardite jaoks on olemas rahvusvaheline tunnustamise infrastruktuur, mis lubab vajaliku taseme saavutanud ettevõtetel taotleda rahvusvahelist sertifikaati ning seda oma klientidele teadvustada.

ISO 9000 seeria võimalused

- ***protsesside parendamine ettevõttes***
- ***sertifikaadi taotlemine ja ettevõtte taseme teadvustamine avalikkusele***

2.1.4.2.EVS-ISO/IEC 12207 ja teised

Eelmistes jaotistes on toodud erinevaid lähenemisi kvaliteedihaldusele: lihtsate põhimõtete rakendamine, kvaliteeditribuudid, meetrikad, ISO 9000 seeria. Lisaks neile võib IT kvaliteedihalduse aluseks võtta allpool toodud standardi EVS-ISO/IEC 12207:1998 või COBIT metoodika.

Nii EVS-ISO/IEC 12207 kui ka EVS-EN ISO 9000-3 on vastu võetud eesti standardina, mis hõlbustab nende kasutamist. Esimesel puudub sertifitseerimine, teisel on olemas rahvusvaheline tunnustamise metoodika ja infrastruktuur. Esimene on protsesside juhtimisele orienteeritud, näiteks ekspluatatsioon ja hooldus on paremini kajastatud. Teine on orienteeritud kvaliteedihaldusele, näiteks käsitletakse kvaliteedipoliitikat, kontrolli- ja testimisvahendite ohjet.

Mitmed ettevõtted Eestis, kes tahavad kvaliteedihaldust seada tõhusale alusele, on alustanud mainitud standarditest. Põhjalikumad rahvusvahelised metoodikad, nagu võimete küpsuse mudelid (CMM, vt. http://www.sei.cmu.edu/cmm/cmms/cmms.html) või tarkvaraprotsesside täiustamise ja võimete määramine (SPICE, vt. http://www.sqi.gu.edu.au/SPICE/), ei mahu käesoleva materjali raamidesse.

2.1.5. Kvaliteedihaldus - esimesed sammud ettevõttes

Kvaliteedihaldus on enamasti suunatud organisatsiooni laiemate eesmärkide saavutamisele. Kui kvaliteedinõudeid ei ole püstitatud, on konkreetse töö tegijal otseselt vähe motivatsiooni kvaliteedi taotlemiseks. Seepärast nõuab kvaliteedihaldus juhtkonna algatust ja tuge.

Kõigepealt tuleb lähtudes ettevõtte strateegiast planeerida kvaliteedihalduse eesmärgid (näiteks, kas soovitakse sisemise töökorralduse parendamist või ka kvaliteedisertifikaati) ja kvaliteedipoliitika.

Vastavalt eesmärkidele valitakse kvaliteedihalduse meetod.

Kui varem ettevõttes kvaliteediküsimustega ei ole tegeldud, siis ei ole otstarbekas alata kogu ettevõtet hõlmava kvaliteedihaldusega. Pigem tasub valida kõige kriitilisem töölõik, kus kvaliteedihaldus annab suurima efekti. Kui valitud meetod õigustab ennast selles lõigus, võib üle minna järgmistele kriitilistele valdkondadele. Vajadusel korrigeeritakse meetodit või kvaliteedipoliitikat.

Kvaliteedihaldus - esimesed sammud ettevõttes

1. kindlustage juhtkonna tugi
2. lähtudes ettevõtte strateegiast planeerige kvaliteedihalduse eesmärgid ja kvaliteedipoliitika
3. valige kvaliteedihalduse meetod
4. valige ettevõtte kõige kriitilisem töölõik
5. parandage seda töölõiku vastavalt valitud metoodikale
6. kui põhimõtted ja metoodika on ennast õigustanud ennast, minge tagasi sammule 3
7. kui metoodika või kvaliteedipoliitika vajavad ümbervaatamist, minge tagasi sammudele 2 või 1

2.1.6.Kontrollküsimusi ja ülesandeid

- Kvaliteet, standard, tarkvara, nõuete liigitusi, elutsükkel, elutsükli liigid
- Kvaliteedijuhtimine, -poliitika, -süsteem, -tõendus
- Kvaliteet ja organisatsiooni juhtimine
- Kvaliteedijuhtimise ja arendusmeetodite seos
- Kvaliteedijuhtimise integreerimine elutsüklisse
- Kvaliteedijuhtimise käsitlusi, teese, süsteeme ja auhindu
- Mida peaks süsteemi arendaja teadma kvaliteedist ja standarditest (tellija, kasutaja, hooldaja, tarnija, firmajuht)?

2.2. Standardid

Anname standardi mõiste ja ülevaate tarkvara puudutavatest standarditest.

2.2.1. Standardi mõiste

Standard on konsensuse alusel koostatud ja tunnustatud kehami poolt kinnitatud normdokument, mis on suunatud standardimiseesmärkide saavutamiseks.

Keham on juriidiline või haldusüksus, millel on kindel ülesanne ja koosseis, nt organisatsioon, firma, ettevõte, ametkond, fond vms.

Konsensus - standard annab üldjuhul eelise kindlale organisatsioonile. Standardi kehtestamiseks on vaja, et ka teised seda aktsepteeriksid. Standardi kehtestamine on seega kokkulepe ja kompromiss. Standard soovitatakse muuta kõigile kättesaadavaks.

Eelised/eesmärgid

- Kulutuste minimeerimine
- Komponentide vahetatavus
- Komponentide koostöökulu minimeerimine (jääb ära liideste tegemine)
- Protseduuride lihtsustamine / parandamine (saab teistelt ettevõtteilt kindlalt töötavaid protseduure üle kanda)
- Kvaliteedijuhtimine
- Standardid võimaldavad reguleerida tootjate omavahelist konkurentsi

Puudused

- Lisakulu, -aeg, -bürokraatia
- Takistab innovatsioon (?)

Standard ei ole kohustuslik, kuid seda saab seaduste ja määruste abil kohustuslikuks muuta.

Standardite mahu ja sisseviimise aja kohta kehtib soovitus: sisse viia nii palju ja nii vara kui vajalik ning nii vähe ja nii hilja kui võimalik.

Standardite liigitus

- Ametlikud standardid (vastu võtnud tunnustatud keham)
- Tööstus- ehk *de facto* standardid (ei pruugi olla ametlikult kinnitatud, kuid on üldlevinud)
- Tehnilised raamstandardid (standardite kogumikud, juhendid nende kasutamiseks)
- Riiklikud soovitused
- Firmasisesed standardid

Standardimiskehameid

- ISO International Organisation for Standardisation (1947)
- IEC the International Electrotechnical Commission (1906)
- ISO / IEC JTC1 Joint Technical Committee
- Üle 170 tehnilise komitee (TC)
- *Sub-Committees (SC)* $\Rightarrow$ *Working Group (WG)*

Firmasisesed

- Lihtsad aru saada ja jälgida
- Piisavad, lünkadeta
- Ühesed
- Ellu viidud

Võib öelda, et kui standard pole ette valmistatud elluviimiseks, siis on mõttetu tema peale aega raisata.

2.2.2. Ülevaade tarkvara standarditest

Tarkvara elemendid (ANSI/IEEE Std 1028-1988):

- plaani dokumendid
- spetsifikatsioon, realisatsiooniprojekt
- testimise dokumentatsioon
- kasutaja dokumentatsioon
- lähtekood
- tarkvara, mis on realiseeritud riistvaras
- aruanded läbivaatuse kohta
- andmed, testi tulemused

Tarkvara arendusstandard soodustab kvaliteedi parandamist, vahetatavust, efektiivsust ja mõõtmisi.

Tarkvara standardite liigitusi:

- üldised (seotud kogu elutsükliga) / organisatsioonilised / tehnilised (nt seotud elutsükli mõne etapiga)
- rahvusvahelised / riiklikud (nt ISO / BS)
- ametlikud / mitteametlikud (nt ISO standardid / andmevoo diagrammid)
- ettevõttesisesed / -välised

Tunnustatud tarkvara standardimiskehameid: ANSI/IEEE, ISO/IEC , BS, MIL, DoD, IEE, CCITT.

Konspekti lõpus on olulisemate või Eestis evitatavate standardite nimetused. Tegelikult on standardeid palju rohkem ja neid tuleb pidevalt juurde. Allpool on temaatikad, millele paljud erinevad standardimiskehamid on loonud hulga standardeid

- Terminoloogia
- Süsteemi elutsükkel
- Juhtimine / korraldus
- Kvaliteedijuhtimine
- Dokumenteerimine
- Testimine
- Konfigureerimine
- Turve, töökindlus
- Elutsükli etappidega seotud tarkvaraarenduse standardid
- Andmevahetus
- ID-kaart
- Rahvuslikud standardid

Standardite kasutamisvõimalusi

- Rahvusvahelised standardid on eeskujuks firmasisestele
- Kollektiivne kogemus: näitab, mida peetakse maailmas oluliseks
- Kirjanduse loetelust saab andmeid standardite tellimiseks
- Standardi vormistamine rahvusvahelisel tasemel

2.2.3.Kontrollküsimusi ja ülesandeid

- Standardi mõiste, eelised, puudused
- Standardite liigitus
- Standardimiskehamite näited
- Tarkvara standardite liigitusi
- Standarditavad valdkonnad
- Standardite kasutamine

2.3. Tarkvara kvaliteedi kriteeriumid ja meetrikad

Vajadusi ja nõudmisi võib liigitada mitmeti. Üks võimalus on anda neid ette kvaliteediatribuutide, sealhulgas kriteeriumide ja meetrikate näol.

2.3.1. Kvaliteediatribuutide süsteem

Tarkvara kvaliteediatribuute on palju (sadades). Süstematiseerimata on neid raske kasutada. Seepärast on loodud mitmeid kvaliteediatribuutide esitamise süsteeme.

Üks selline süsteem on nõuete esitamine mitmetasemelise kvaliteediatribuutide puu kaudu. Puu ülemisel tasemel on kvaliteedifaktorid, mis annavad üldettekujutuse toote olulistest omadustest. Kvaliteedifaktor kui üldise taseme atribuut on määratud juhtkonnale, tellijale, süsteemi kasutajale. Selliseid omadusi on raske kvantifitseerida, seepärast jaotatakse iga faktor kvaliteedikriteeriumideks, mis moodustavad puu järgmise taseme või tasemed. Kriteeriumid on detailsemad ja rohkem arendusorienteeritud, neid saab kasutada näiteks väljatöötaja või kasutajapoolne projektijuht. Iga kriteerium jaguneb omakorda meetrikateks, mida saab otse mõõta. Meetrikad võivad huvi pakkuda kõigile osapooltele, näiteks on meetrikaid, mida peab teadma kasutaja (tõrgete arv ajaühikus kindla tööprofiili puhul), ja arendusse puutuvaid meetrikaid.

Sellise süsteemi puhul saab meetrikatest alustades integreerida väärtusi ülespoole. Integreerimiseks kasutatakse mitmesuguseid tehnikaid, nagu lihtne väärtuste liitmine, kaalutud summad, hägune (*fuzzy*) loogika, otsustusmeetodid jne. Kui on saadud väärtused kriteeriumide alumise taseme jaoks (juhul kui kriteeriume oli mitu taset), siis võib jätkata samamoodi ülespoole. Tavaliselt kriteeriume faktoriteks enam ei koondata, kuid võimatu see pole. Viimasel juhul, valides sobivad kaalud või muud integreerimismeetodit toetavad objektid, saab alustada meetrikatest ja liikuda ülespoole, kuni lõpuks on olemas üks kvaliteeti iseloomustav arv. Seda võrreldakse etteantud arvuga, et otsustada, kas süsteemi kvaliteet on piisav.

Kuna kaale on põhjendatult raske valida, siis kehtestatakse kvaliteedi üle otsustamiseks siiski tavaliselt otsustusnõuded (nt võetakse süsteem vastu, kui hinnatavate kriteeriumide osas ei esinenud ühtki tõsist viga) ja hinnatakse nende täitmist nagu testimise standardis IEEE Std 829-1983.

2.3.2. Atribuudid ISO/IEC 9126

Kvaliteediatribuutide skeeme on palju, ühe levinud skeemi loogika on lihtsustatult järgmine. Süsteemi kasutatakse, hooldatakse ja kantakse üle teistele riist- ja tarkvaraplatvormidele. Kasutamisel peaks süsteem täitma vajalikke funktsioone ning

olema töökindel, efektiivne ja kasutajasõbralik. See loogika viib järgmiste kvaliteedifaktoriteni (ISO/IEC, 9126 - sulgudes on kriteeriumid):

- funktsionaalsus (vastavus ülesannetele - kas kõik funktsioonid on olemas; täpsus; koostöövõime teiste süsteemidega; vastavus normidele; turvalisus)
- töökindlus (valmidus - kui tihti esineb tõrkeid; veakindlus - kuidas reageerib väliskeskkonna vigadele; taastatavus - kui raske on peale tõrget uuesti tööd alustada)
- efektiivsus (ajaefektiivsus; ressursiefektiivsus)
- kasutatavus (kontseptuaalne selgus; õpitavus; kasutusmugavus)
- hooldatavus (analüüsitavus - kui raske on leida muutmise kohta; muudetavus - kui raske on muuta; stabiilsus - kui tugevalt muudatused mõjutavad süsteemi; testitavus)
- ülekantavus (adapteeruvus - kas saab üle kanda; installeerimise mugavus - kui raske on ülekanne; vastavus normidele; asendatavus)

2.3.3. Meetrikad

Tarkvara arendusprotsessi on vaja mõõta või hinnata kas või sellepärast, et osata seda tööd rahaliselt väärtustada. Selliseks mõõtmiseks kasutatakse tarkvara meetrikaid - arvutatavaid või hinnatavaid suurusi, mis iseloomustavad olulisi omadusi ja võivad olla seotud ülesande, projekti, dokumendi või ettevõttega. Tarkvara meetrikate kasutamine on efektiivne vahend arendusprotsessi parandamiseks, kuid sellega peab kaasnema muu kvaliteedijuhtimine ning võrdlevad hinnangud teiste arendajatega.

Meetrikaid on palju, ülevaate saamiseks võib neid klassifitseerida

- tehnoloogia järgi (nt programmeerimiskeeltele orienteeritud meetrikad, andmebaaside meetrikad jne)
- elutsükli protsessi järgi (näiteks arenduse või hoolduse meetrikad)
- rakendusala järgi (näiteks juhtimissüsteemides või panganduses kasutatavad meetrikad)

Kuna programmeerimisest peaks kõigil kuulajatel olema ülevaade, käsitletakse loengus mõnda lihtsamat ja loomulikumat programmeerimismeetrikat - koodi ridade arvu, produktiivsust, vigade tihedust, et illustreerida tekkivaid probleeme. Tuuakse näiteid ka teistsuguste meetrikate kohta.

Üks ideelt selgem ja mitmeti kasutatav meetrika on koodi ridade arv. Kuid ka selle näiliselt lihtsa meetrikaga pole kõik alati lihtne. Uurimus näitas, et hindajad võivad sama programmi ridade arvu hinnata kuni suurusjärgu võrra erinevaks. Ridade arvul põhineb mitu teist meetrikat nagu arendaja tootlikkus ja vigade tihedus. Howard Rubin annab järgmised hinnangud:

- arenduse tootlikkus: (ridu aastas) kõigub 5000 ja 60 000 vahel; USA keskmine 7500, Jaapani keskmine 12 000
- vigade arv tuhande koodirea kohta on USA-s üle nelja, Jaapanis alla kahe

Programmeerimismeetrikate liike:

- hindavad / iseloomustavad programmi (mahtu, keerukust)
- hindavad kvaliteeti

- toetavad arendusprotsessi (prognoosivad töö mahtu, ridade arvu)

Meetrikaid saab kasutada, võrreldes neid teada oleva soovitava tasemega või integreerides neid kvaliteedikriteeriumideks. Mehaaniline kasutamine, näiteks programmi ridade arvu või McCabe mõõdu kasutamine töö tasustamise hinnanguks võib tuua pigem kahju ja viia programmide kunstlikult pikemaks/keerukamaks ajamiseni; sama kehtib paljude teiste meetrikate puhul.

Esimesed meetrikad ilmusid seitsmekümnendate aastate algul. Tuntumad olid McCabe ('76) programmi keerukuse mõõt ja Halsteadi (’77) tarkvara teadus. Toome veel kvaliteedimeetrikate näiteid. Milline on väärtuse määramispiirkond, parim väärtus?

Spetsifikatsiooni muutmise tase (E+M+L)/A, kus

E - eemaldatud funktsioonid,

M- muudetud funktsioonid,

L - lisatud funktsioonid,

A - algne funktsioonide arv

(parim väärtus = 0).

Tehnilise ülesande ja spetsifikatsiooni vastavus

$$\frac{\text{Funktsioonide arv tehnilises ülesandes}}{\text{Funktsioonide arv spetsifikatsioonis}}$$

$$\text{Keskmine tõrgetevaheline aeg} = \frac{\text{Kogu funktsioneerimise aeg}}{\text{Tõrgete arv}}$$

Meetrika probleeme

- "Mind ei huvita tehnilise ülesande ja spetsifikatsiooni vastavus, mind huvitab, kas see tarkvara teeb, mida vaja!" Lahendus - meetrikate integreerimine
- Ütleme kliendile, et tehnilise ülesande ja spetsifikatsiooni vastavus on 1.3333… See arv ei ütle talle mitte midagi. Lahendus - meetrikate/projektide võrdlusandmete andmebaasid
- Suurus, mida saab mõõta, pole huvitav; suurust, mis on huvitav, ei saa mõõta. Lahendus - standardida mõõdetavad meetrikad
- Meetrika peab olema sisukas ja oluline, muidu võib see tuua pigem kahju

2.3.4. Kontrollküsimusi ja ülesandeid

- 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

- Kvaliteedimeetrikate näiteid, väärtuse määramispiirkond, parim väärtus

2.4. Infosüsteemi audit

Üks võimalik lahendus vastutusrikaste rakenduste hindamiseks on, et süsteemi omadustest huvituv osapool kutsub olukorrast ülevaate saamiseks tööle kolmanda isiku - audiitori. Infosüsteemid pole selles suhtes erand, analoogiline on olukord näiteks raamatupidamises, keskkonnakaitses ja mujal (COBIT, 1998; EISAÜ, 1997; Tepandi, 1998).

Audit, selle objekt ja läbiviijad

Infosüsteemi audit on mitmekülgne ülevaade ja hinnang auditeeritava ettevõtte, asutuse või organisatsiooni automatiseeritud infosüsteemile või selle osadele, kaasa arvatud seostele automatiseerimata protsessidega ja organisatsioonilise struktuuriga.

Toome näiteid küsimustest, millele auditi käigus püütakse vastust leida. Asutusel on probleem infolekkega. Kas saab selgitada põhjuseid ja hoida ära sellised juhtumid tulevikus? Juhtkonnale tundub, et infotehnoloogia ressursse ei kasutata hästi. Mida teha? Oluline projekt venib. Mida ette võtta?

Näiteid auditi eesmärkide kohta:

- hinnata süsteemide ja infotöö vastavust ettevõtte (äri)huvidele
- hinnata ettevõttega seotud kolmandate osapoolte (näiteks avalikkuse) nõuete rahuldatust
- hinnata firma tegevusele eluliselt vajaliku info usaldatavust, kättesaadavust ja kaitstust
- hinnata süsteemide või infotöö korralduse kvaliteeti, turvet ja töökindlust
- kaitsta tellija huve, kui tellitavas projektis on põhiline teadmine täitja poolel
- kontrollida venivaid või muus mõttes ebaedukaid projekte
- pakkuda tuge uute projektide käivitamisel

Auditeeritakse kõiki infosüsteemidega seotud objekte, tegevusi, protsesse ja valdkondi, sealhulgas planeerimist, organisatsiooni, dokumentatsiooni, hanget, projekti, projekti juhtimist, arendust, metoodikaid, kasutamist, hooldust, mõõtmist, aruandlust, jälgimist (sisemist kvaliteedijuhtimist).

Infosüsteemi audiitor on isik, kes soovitatavalt, omades kehtivat infosüsteemi audiitori sertifikaati, auditeerib auditi eesmärgist lähtudes auditeeritava organisatsiooni infosüsteemi vastavalt infosüsteemide audiitorkontrolli eeskirjadele ja järgib infosüsteemi audiitori eetikanormistikku.

Audiitorile esitatakse mitmesuguseid nõudmisi. Ta peaks olema sõltumatu auditeeritavast rakendusest, olema ekspert infosüsteemide auditeerimises ja infotehnoloogia vastavas valdkonnas, jälgima auditeerimise head tava ja reegleid, olema tuttav Eesti seadusandlusega ja standarditega ning tundma mõnda tunnustatud auditeerimise metoodikat.

Eetikareeglid ütlevad muuhulgas, et audiitor peab

- toetama infosüsteemide eeskirjade, protseduuride ja kontrollide väljatöötamist ning nende järgimist

- tegutsema hoolikalt, lojaalselt ja ausal viisil oma tööandja, ettevõtte omanike, klientide ja avalikkuse huvides ning teadlikult mitte osa võtma mis tahes seadusevastasest või ebasündsast tegevusest
- säilitama oma kohustuste täitmise käigus saadud informatsiooni konfidentsiaalsust. Informatsiooni ei tohi kasutada isikliku kasusaamise huvides ega avaldada asjasse mittepuutuvatele osapooltele
- täitma oma kohustusi sõltumatult ja objektiivsel viisil ning hoiduma tegevustest, mis ohustaksid või võiksid ohustada tema sõltumatust
- säilitama asjatundlikkust auditi ja infosüsteemide alal, arendades oma ametialaseid oskusi ning võttes osa koolitusest
- hoolikalt koguma ja dokumenteerima piisavat faktilist materjali, millel põhjal teha järeldusi ja soovitusi
- informeerima asjassepuutuvaid osapooli sooritatud auditist
- toetama juhtkonna, klientide ja avalikkuse koolitamist, et laiendada nende arusaamist auditist ja infosüsteemidest

Auditi korraldus

Auditi läbiviimine sisaldab selliseid samme nagu eelläbirääkimised, auditeerimislepingu sõlmimine, auditi planeerimine, olukorra identifitseerimine ja dokumenteerimine, näiteks kehtestatud infosüsteemi kasutamise ja arendamise poliitika; protseduurireeglid; vastavus seadusandlusele, organisatsiooni äriplaanile, rahvusvahelistele *de jure* ja *de facto* standarditele, tehnoloogilistele nõuetele ja standarditele; hindamine, nt riskide hinnang, vastavustestimine, reeglite tegeliku täitmise ulatuse hindamine, vajadusel sisuline testimine, hinnang ja raportid.

Kuna audit ei kontrolli kõike, siis jääb nagu teistegi auditi tüüpide puhul risk, et auditi käigus ei avastata ka suhteliselt olulisi vigu. Seda riski tuleb teadvustada lepingu läbirääkimistel ja sellele tuleb viidata ka auditi lepingus. Auditiga on seotud ka korralduse ja ootuste riskid. Näiteks kui vead olid enne teada ja nende parandamiseks pole soovi või ressursse, võib auditi kasu olla piiratud. Audit pole ka arendus ega jooksev vigade parandus.

Auditi planeerimise käigus määratletakse kriitilised valdkonnad, pühendatakse neile piisavalt tähelepanu ja varutakse ressursse. Lepitakse kokku töö õige järjekord ja koordineerimine, tõendusmaterjalid, kontrolli metoodika, raportid, tähtajad ja töö maht.

Auditit toetav infrastruktuur

Maailmas ühendab infosüsteemide audiitoreid eelkõige Infosüsteemide Auditi ja Kontrolli Assotsiatsioon (*Information Systems Audit and Control Association*, ISACA, vt http://www.isaca.org). Assotsiatsioonil on 18 000 liiget sajas riigis. Ta sertifitseerib infosüsteemide audiitoreid, avaldab auditialast kirjandust, töötab välja auditi metoodikaid, korraldab koolitust, algatab uurimis- ja arendustöid, avaldab ajakirja *IS Audit & Control Journal* ning korraldab viiel mandril rahvusvahelisi konverentse.

Hiljuti välja antud info- ja sellega seotud tehnoloogia kontrolli sihid (COBIT) annavad auditi korralduse üldise metoodika. COBIT eristab nelja põhilist ala: planeerimine ja organisatsioon, hange ja rakendus, ülekanne ja tugi, jälgimine. Nendel aladel on määratletud infotehnoloogia protsessi ja kontrolli üldised sihid. Iga sihi jaoks on

määratud detailsed auditi alad, meetodid, kontrollide hindamine, vastavuse hindamine, riskide hindamine.

Üks ISACA olulisemaid töölõike on audiitorite sertifitseerimine. Sertifitseeritud infosüsteemide audiitor (CISA) peab sooritama vastava eksami, tõendama pidevat koolitust, jälgima audiitori eetikanorme ja standardeid, omama töökogemust.

CISA koolitust ja eksamit arendavad ISACA ja selle tütarorganisatsioonid. Eksam korraldatakse igal aastal ühel kindlal päeval. Eksamil testitakse kandidaadi kogemusi infotehnoloogia auditi, kontrolli ja turbe alal, samuti tema oskusi rakendada erialaseid standardeid ja teadmisi. Näiteks 1997. aastal tuli selleks nelja tunni jooksul vastata kahesajale küsimusele. Eksami valdkondadel on järgmine osakaal:

- audit ja infosüsteemide (IS) turve 8%
- IS organisatsioon (sh strateegia, korraldus) ja juhtimine 15%
- IS protsess (sealhulgas infotehnoloogia) 22%
- IS terviklus, konfidentsiaalsus, käideldavus 29%
- IS arendus, hange, hooldus 26%

1997. aastal tegi CISA eksamit üle 4000 spetsialisti maailma enam kui viiekümnes riigis. Üks riik oli Soome. Helsingis sooritas kahekümne kaheksast osalejast eksami edukalt üksteist (1996. aastal oli Helsingis osalejate ja edukalt sooritajate suhe 30:10). Maailmas on praegu üle 10 000 CISA audiitori.

ISACA on praeguseks välja andnud umbes 50 IS auditialast raamatut. Raamatute hulgas on kogu auditi ala katvad monograafiad, eriküsimusi käsitlevad raamatud, näiteks UNIX-, EDI-, COBIT-teemalised väljaanded, sissejuhatavad tekstid, CISA eksami materjalid, videod.

Eestis on olemas Eesti infosüsteemide audiitoriühing, kuhu kuuluvad alast huvitatud isikud. Ühingu tegevusvaldkonnad on koolitus, avalikkuse informeerimine, koostöö teiste organisatsioonidega (ISACA), auditi standardite ülevõtmine või koostamine, infosüsteemide audiitorkontrolli eeskirjade koostamine ja arendamine.

2.4.1.Kontrollküsimusi ja ülesandeid

- Audit, selle eesmärgid, auditeeritavad objektid, maht, audiitor, riskid, planeerimine

3. Lisainfo ja kirjandus

Lisamaterjali saab (TTÜ) raamatukogust, WWW-st ja mujalt. Kõigepealt, on ilmunud rida raamatuid ja artikleid, mille pealkirjades esinevad sõnad *Information Systems, Software Engineering, Software Testing, Quality Management*, sh ka vene ja saksa keeles.

Kirjandust ja viiteid võib otsida WWW-st märksõnade nagu *Testing* ja *Quality* alt. On erialaseid uudistegruppe. Näiteid:

- standardid - http://www.eik.ee
- testimine - http://www.io.com/~wazmo/qa.html (palju viiteid), http://www.soft.com/,

- kvaliteedijuhtimine - http://www.soft.com/QualWeek/QW99, http://deming.eng.clemson.edu/ (viiteid teistele kvaliteedijuhtimise allikatele),
- näide uudistegrupist - comp.software.testing.

Ajakirjad. On erialaajakirju (*Communications of the ACM*, *IEEE Software*), artikleid avaldatakse perioodiliselt ka muudes ajakirjades.

Konverentsid. On sellele teemale pühendatud üldisi konverentse, erialateemadele pühendatud konverentse, sektsioone muudel konverentsidel.

Standardite ja kirjanduse viiteid

- ANSI/IEEE (1988). ANSI/IEEE Std 1028-1988. IEEE Standard for Software Reviews and Audits
- Certified Information Systems Auditor Review Manual. Information Systems Audit and Control Association, Rolling Meadows, USA, 1997
- COBIT (1998). Governance, Control and Audit for Information and Related Technology. Second Edition. Information Systems Audit and Control Foundation, Rolling Meadows, USA, 1998
- EISAÜ (1997). Infosüsteemide audiitorkontrolli eeskirjad. Eesti infosüsteemide audiitorühing. Tallinn, 1997
- EVS 8:1993 Infotehnoloogia reeglid eesti kultuuri ja keele keskkonnas - uustöötlus
- EVS-EN ISO 9000-3:1999 Kvaliteedijuhtimise ja kvaliteeditagamise standardid. Osa 3: Suunised ISO 9001:1994 kohaldamiseks tarkvara väljatöötusele, tarnimisele, installeerimisele ja hooldusele
- EVS-ISO/IEC 12207:1998. Infotehnoloogia – Tarkvara elutsükli protsessid
- EVS-ISO/IEC 2382-1:1998. Infotehnoloogia. Sõnastik. Osa 1: Põhiterminid
- EVS-ISO/IEC 2382-2,… :1998. Infotehnoloogia. Sõnastik. Osad 2-6, 9-13, 16, 17, 19-27
- EVS-ISO/IEC TR 13335-1:1999 Infotehnoloogia. Infoturbe halduse suunised. Osa 1: Infoturbe mõisted ja mudelid
- EVS-ISO/IEC TR 13335-2:1999 Infotehnoloogia. Infoturbe halduse suunised. Osa 2: Infoturbe haldus ja plaanimine
- EVS-ISO/IEC TR 13335-3:1999 Infotehnoloogia. Infoturbe halduse suunised. Osa 3: Infoturbe halduse meetodid
- IEEE Std. 829-1983 (Reaff 1991) IEEE Standard for Software Test Documentation
- ISO (8402). ISO 8402. Quality Assurance - Vocabulary
- ISO 5807:1985 Information processing -- Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts
- ISO 8879:1986 Information processing – Text office systems – Standard Generalized Markup Language (SGML)
- ISO/IEC (9126). ISO/IEC 9126, Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their Use, 1991
- ISO/IEC 7812:1993 Identification cards Identification of issuers Part 1: Numbering system; Part 2: Application and registration procedures koos eesti lisaga “Taotlemine ja registreerimise kord”

- ISO/IEC 7816:1994 Identification cards Integrated circuit(s) cards with contacts Part 5: Numbering system and registration procedure for application identifiers
- ISO/IEC DIS 6592 Information technology -- Guidelines for the documentation of computer-based application systems
- ISO/IEC DTR 15271 Information technology -- Guide for ISO/IEC 12207 (Software Life Cycle Processes)
- ISO/IEC TR 13335 Information technology -- Guidelines for management of IT Security Part 4: Selection of safeguards; Part 5: Safeguards for external connections
- ISO/TR 13569:1997 Banking and related financial services -- Information security guidelines
- Jarvis and Crandall (1997). Jarvis, A., Crandall, V. Inroads to Software Quality. Prentice Hall, New Jersey
- Koomen, T., and Pol, M. Test process improvement: a practical step-by-step guide to structured testing. Addison Wesley, 1999
- Littover ja Vain (1994). Littover, M., Vain, J. Infosüsteemi projekti käivitamisest. Arvutustehnika ja Andmetöötlus, 1994, nr. 5-10
- Lucas (1994). Lucas, H. C. Jr. Information Systems Concepts for Management. Mitchell McGRAW-HILL, New York, 1994
- Martin (1989). Martin, J. Information Engineering, Vol. 1, Introduction. Prentice-Hall, Englewood Cliffs, N. J.
- Pressman (1992). Pressman, R. S. Software Engineering. A Practitioner's Approach. European Edition. Adapted by D. Ince. McGraw Hill, London
- Tepandi (1998). Tepandi, J. Ülevaade infosüsteemide auditeerimisest. Arvutustehnika ja Andmetöötlus 1998, nr. 2, lk 20-22