

Tarkvara kvaliteet ja standardid (IDX5721, IDX5722)

Version 23.11.2011

Jaak Tepandi

Tallinna Tehnikaülikooli Informaatikainstituut

*Materjali viimane versioon: **[jaak.tepandi.ee](http://www.tepandi.ee)***

(või täpsemalt <http://www.ttu.ee/users/tepandi/we-tk.html>)

Tallinn

Sisukord

1. MILLEKS JA KELLELE	4
1.1. KURSUS JA MATERJALID	4
1.2. KUIDAS LUGEDA?	5
1.3. EKSA MIKÜSIMUSED JA OLULISEMAD TEEMAD	5
2. PÕHIMÕISTED	7
2.1. TARKVARA KVALITEET = TOODE + NÕUDED + PROTSESSID	7
2.1.1. <i>Kontrollküsimusi ja ülesandeid</i>	7
2.2. TARKVARATOODE	8
2.2.1. <i>Kontrollküsimusi ja ülesandeid</i>	8
2.3. NÕUDED	8
2.3.1. <i>Nõuete liigid</i>	8
2.3.2. <i>Kvaliteediatribuutide süsteem</i>	9
2.3.3. <i>Kvaliteedinäitajad standardi ISO/IEC 9126 / 25010 põhjal</i>	10
2.3.4. <i>Funktsionaalsed ja mittefunktsionaalsed nõuded</i>	13
2.3.5. <i>Testitavad ja mittestitavad nõuded</i>	13
2.3.6. <i>Kontrollküsimusi ja ülesandeid</i>	13
2.4. TARKVARA ARENDUSE PROTSESSID	14
2.4.1. <i>Protsessid, elutsüklid ja nende mudelid</i>	14
2.4.2. <i>Tarkvara elutsükli mudelite näited</i>	14
2.4.3. <i>Hankija tegevused</i>	17
2.4.4. <i>Kontrollküsimusi ja ülesandeid</i>	18
3. TARKVARA KONTROLL: MEETODID JA KORRALDUS	19
3.1. TESTIMINE	19
3.1.1. <i>Testimise põhimõtteid</i>	19
3.1.2. <i>Riski- ja ekspertteadmiste põhine, uuriv jm testimine</i>	21
3.1.3. <i>Spetsifikatsiooni põhine testimine</i>	23
3.1.4. <i>Testimine programmi teksti põhjal</i>	27
3.1.5. <i>Juhuslikud sisendid ja lisatud vead</i>	31
3.1.6. <i>Mittefunktsionaalsete nõuete testimine: kasutuskõlblikkus</i>	32
3.1.7. <i>Testimise maht ja lõpetamine</i>	33
3.1.8. <i>Kontrollküsimusi ja ülesandeid</i>	34
3.2. STAATILISED MEETODID	35
3.2.1. <i>Küsimustikud</i>	36
3.2.2. <i>Läbivaatused ja hindamised</i>	39
3.2.3. <i>Kontrollküsimusi ja ülesandeid</i>	42
3.3. TÖÖKINDLUSE OLULINE SUURENDAMINE JA SELLE RAKENDUSI	42
3.3.1. <i>Töökindluse parendamise vahendeid - ülevaade</i>	42
3.3.2. <i>N-versiooniline programmeerimine (N-version programming)</i>	43
3.3.3. <i>Veapuu analüüs</i>	43
3.3.4. <i>Formaalne verifitseerimine/tõestamine</i>	43
3.3.5. <i>Tarkvaraarendus puhtas/kontrollitud keskkonnas</i>	45
3.3.6. <i>Common Criteria</i>	45
3.3.7. <i>Võrdlus riistvaraga</i>	46
3.3.8. <i>Kontrollküsimusi ja ülesandeid</i>	46
3.4. KONTROLLI KORRALDUS	46
3.4.1. <i>Lihtsamaid skeeme, vajadus keerukamaks korralduseks</i>	47
3.4.2. <i>Kontrolli meetodite efektiivsus ja valik</i>	48
3.4.3. <i>Elutsükli V-mudel: Arendusprotsessi integreeritud kontroll</i>	49
3.4.4. <i>RUP testimise filosoofia</i>	52
3.4.5. <i>XP: Näide arendusprotsessi integreeritud kontrollist</i>	52

3.4.6.	Testimisprotsessi standardid ja täiustamine (TPI metoodika).....	53
3.4.7.	Tarkvara kontrolli automatiseerimine	54
3.4.8.	Pidev testimine ja arendus	57
3.4.9.	Kontrollküsimusi ja ülesandeid.....	57
4.	KVALITEEDIHALDUS, STANDARDID, NORMID, AUDIT	58
4.1.	KVALITEET	58
4.1.1.	Kvaliteet ja kvaliteedihaldus: mõisted	58
4.1.2.	Miks levib ebakvaliteetne (laiatarbe)tarkvara?	59
4.1.3.	Kvaliteedihalduse mitteformaalseid meetodeid.....	60
4.1.4.	Kvaliteedi hindamise süsteeme ja auhindu	61
4.1.5.	ISO 9000 seeria standardid	62
4.1.6.	Kvaliteedihalduse protsessid - esimesed sammud ettevõttes.....	62
4.1.7.	Kontrollküsimusi ja ülesandeid.....	63
4.2.	TARKVARA ARENDUS JA KVALITEEDIHALDUS (PROTSESSI KVALITEET).....	63
4.2.1.	Arendus ja kvaliteedihaldus	63
4.2.2.	Oma metoodika näide: tarkvaraarenduse 10 põhikomponenti	64
4.2.3.	EVS-ISO/IEC 12207 Infotehnoloogia - tarkvara elutsükli protsessid.....	66
4.2.4.	CMMI.....	68
4.2.5.	ITIL ja ISO/IEC 20000.....	68
4.2.6.	ISKE ja IT etalonturbe käsiraamat	68
4.2.7.	Kas "tardmetoodikad vs paindmetoodikad" või "haldamisraamistikud vs arendusmetoodikad"?	69
4.2.8.	IT standardid ja metoodikad: üldpilt	70
4.2.9.	Millest alustada?.....	70
4.2.10.	Kontrollküsimusi ja ülesandeid.....	71
4.3.	KVALITEEDI MÕÕTMINE: MEETRIKAD	71
4.3.1.	Ülevaade tarkvara meetrikatest	71
4.3.2.	Tarkvara arendusmaksumuse prognoos	73
4.3.3.	Kontrollküsimusi ja ülesandeid.....	74
4.4.	STANDARDID	74
4.4.1.	Standardi mõiste	74
4.4.2.	Standardimine Eestis.....	75
4.4.3.	Ülevaade tarkvara standarditest	76
4.4.4.	Tarkvara dokumenteerimine	78
4.4.5.	Kontrollküsimusi ja ülesandeid.....	78
4.5.	INFOSÜSTEEMI AUDIT.....	79
4.5.1.	Audit, selle objekt ja läbiviijad.....	79
4.5.2.	Auditi korraldus	80
4.5.3.	Organisatsioonid ja sertifitseerimine.....	80
4.5.4.	COBIT.....	81
4.5.5.	Kontrollküsimusi ja ülesandeid.....	82
5.	ÜLEVAATEID STANDARDITEST	84
5.1.	STANDARDIST "ANSI/IEEE STD 829 STANDARD FOR SOFTWARE TEST DOCUMENTATION"	84
5.2.	STANDARDIST "IEEE STD 830-1998 IEEE RECOMMENDED PRACTICE FOR SOFTWARE REQUIREMENTS SPECIFICATIONS"	89
5.3.	STANDARDIST EVS - ISO/IEC 9126-1:2003 TARKVARATEHNIKA. TOOTE KVALITEET. OSA1: KVALITEEDIMUDEL.....	90
6.	LISAD	93
6.1.	MATERJALE	93
6.2.	SÕNASTIK	95
6.3.	KASUTATUD LÜHENDEID.....	98

1. Milleks ja kellele

Igal pool meie ümber on kõrgtehnoloogia ja sellesse kuuluvad tarkvarasüsteemid. Arenevad ka nõudmised ja kasvavad ootused, nurinat ebakvaliteetse tarkvara aadressil on nii kasutajate kui ka tegijate poolelt.

Kasutajad pole rahul süsteemidega, mis teevad valet asja, on aeglased või ebaturvalised. Tegijad kulutavad liialt vahendeid tarkvara parandamiseks ja hoolduseks, nurisevad tellijate ning kasutajate teadmatuse üle ja kannatavad viletsat tööd tegevate allpakkujate pärast. Käesolev materjal aitab loodetavasti olukorda parandada ... või vähemalt pakub selleks vahendeid.

1.1. Kursus ja materjalid

Käesolev konspekt on aluseks kursusele "Tarkvara kvaliteet ja standardid" ning selle ainetöole. Kursuse konspekti viimane versioon on kättesaadav veebilehe jaak.tepandi.ee kaudu, samast on kättesaadav ka korralduse materjal (hindamine, iseseisvad tööd ja ainetööd jne).

Kursuse (ja vastavalt ka materjalide) loogika on ülevaateks üksikule ja üksikult üldisele - peale põhimõisteid esitatakse (kontrolli) meetodid, siis kvaliteet ja standardid üldiselt. Osalt on see tingitud eeldatavast arusaamise loogikast (näiteks on testimise meetodid konkreetseid ja kohe kasutatavad), osalt kursuse ja ainetöö seostest (meetodeid saab kohe kasutada ainetööde tegemiseks).

Kursus käsitleb kahte omavahel seotud, kuid siiski erinevat valdkonda. Esimene valdkond, tarkvara kontroll (sõna "kontroll" kasutatakse kursuses fraasi "testimine, verifitseerimine ja valideerimine" sünonüümina) on üks osa tarkvara elutsüklist. Ta ei pea olema eraldi etapp, kuulub pigem soovitatavalt kõigisse tegevustesse. Teine valdkond, tarkvara kvaliteedihaldus ja standardid, on üldisem ja katab organisatsiooni, protseduurid, meetodid jne.

Üks tuntumaid rahvusvahelisi arvutiteaduse ja infotehnoloogia õppeprogrammide initsiatiive on *ACM/IEEE Computing Curricula*, millesse kuulub tarkvaratehnikat käsitlev [Software Engineering Volume](#). Viimase seisukohast on materjal seotud põhiliselt valdkondadega "*Software Verification and Validation*" ja "*Software Quality*". Seega on kontroll ja kvaliteedihaldus kaks erinevat teemat - viimane ei piirdu mingil juhul ainult kontrolliga. Esimeses peatükis täpsustatakse neid seoseid ja esitatakse põhimõisted, mida on vaja kontrolli meetodite paremaks omandamiseks.

Kursuse eesmärk on anda ülevaade tarkvara kontrolli ja kvaliteedijuhtimise probleemidest ning meetoditest, tutvustada lähemalt testimise meetodeid ja korraldust ning 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 paremini koostada pakkumiskutset ning valida toodet, jälgida arendusprotsessi ja hinnata tulemust
- kasutajale, et osata küsida ostetava tarkvara omadusi ja teada, mida võib tarkvaratootelt oodata

- juhile, kes õpib, mida tellijalt ja arendajalt nõuda

Käesolev materjal on kursuse lühiülevaade, mis ei asenda loengutes ja harjutustes osalemist ning iseseisvat tööd materjalidega. Näited ja detailsemad selgitused antakse loengul. Materjal on osaliselt kaetud väljaandes "J. Tepandi. Tarkvara kvaliteet ja standardid. TTÜ kirjastus, 1999". Failis olev tekst võib olla loengutest erinevalt struktureeritud (jaotisteks liigendatud, pealkirjastatud) ja järjestatud. Materjal täieneb ja muutub töö käigus, semestri ja eksami lõplik variant on olemas enne eksamisesiooni algust.

Autor tänab kolleege, kursuse kuulajaid ja redigeerijaid paljude kasulike märkuste ja ettepanekute eest. Eelmise sügise versiooni jaoks tegid kasulikke ettepanekuid Raul Adler ja Alan Vask. Edasised kommentaarid ja märkused selle materjali kohta on oodatud, palun saata need aadressil jaak (at) tepandi.ee või anda edasi loengul/praktikumis.

1.2. Kuidas lugeda?

Alustuseks on soovitatav tutvuda järgmise osaga (põhimõisted), edasi sõltub huvidest.

Süsteemiarendaja valib arvatavasti oma eelistuste põhjal, näiteks võib aluseks võtta materjali järjestuse. Tellija või kasutaja eelistab võib-olla alustada kvaliteedihalduse osast - kui aega on väga vähe, siis kvaliteedi kriteeriumide ja meetrikate ning Eesti tarkvara standardite peatükkidest, mis võivad anda häid ideid pakkumiskutse koostamiseks või toote valikuks. Juht eelistab arvatavasti samuti alustada kvaliteedihalduse osast. Kõigile võib soovitada kogu materjali läbilehitsemist, et näha, mis veel võib huvi pakkuda.

Kes soovib lisaks teadmistele/oskustele kursuse eest ka hinnet saada (loodetavasti ei loeta ainult hinde saamiseks), võiks kasutada materjali viimast versiooni, vastata peatükkide lõpus antud kordamisküsimustele (mis on ka eksamiküsimusteks) ning osaleda loengutes ja praktikumides, kus antakse põhjalikumaid selgitusi. Kordamisküsimustele võib tihti anda erinevaid vastuseid. Oma tõlgendused on soovitatavad, samas tuleb teada kursuse käsitlust ja osata põhjendada erinevusi.

1.3. Eksamiküsimused ja olulisemad teemad

Kordamisküsimused on antud iga peatüki lõpus. Need on ühtlasi eksamiküsimusteks.

Minimaalselt võiks ette võtta käesoleva materjali sisukorra ja kontrollida, et iga alateema kohta on teada vähemalt põhilised mõisted.

Praktilisest seisukohast rõhutame järgmisi teemasid ja ülesandeid, mis on esimeses järjekorras olulised ka eksamil.

1. Tarkvaratoode – mis siia kuulub?
2. Näited tarkvara probleemidest tulenenud rasketest intsidentidest. Tarkvara probleemide ulatuse ja maksumuse hinnangud
3. Nõuded - näide nõuete (kvaliteediatribuutide) süsteemist, kahe faktori kriteeriumid
4. Funktsionaalsed ja mittefunktsionaalsed nõuded, testitavad ja mittetestitavad nõuded
5. Tarkvara elutsükli mudelid ja protsessimudelid – mis on erinevus? Ülevaade protsessimudelitest ja tarkvara elutsükli mudelitest.
6. Tarkvara kvaliteet

7. Läbivaatus/inspektsioon, selle plussid, miinused, eeltingimused, osavõtjad, korraldus
8. Riskipõhine, ekspertteadmiste põhine, uuriv, suitsu- jm testimine
9. Funktsionaalne testimine, ekvivalentsiklasside analüüs, piirulukorrad, otsustustabelid, veaotsing, andmepõhine testimine. Antud nõuded, pakkuda testid
10. Testimine programmi teksti põhjal, adekvaatsuse (katvuse) kriteeriumid, nende võimsuse suhe. Antud programm, pakkuda testid
11. Mittefunktsionaalsete nõuete testimine
12. Testimise ja kontrollimeetodite efektiivsuse võrdlus ja soovitused kasutamiseks (mida kasutada kõigepealt, mida teises järjekorras?)
13. Saavutatav töökindluse tase seniste meetoditega, selle oluline suurendamine. Verifitseerimine
14. Tarkvara kontrolli korralduse lihtsamaid skeeme. Milline neist on parim?
15. XP, V-mudel, arendusprotsessi integreeritud kontroll, TPI mudel
16. Testimise etapid: ühiku-/mooduli-, integratsiooni-, valideerimise- ja süsteemitestid
17. Testimise automatiseerimine, eeltingimused, eelised, puudused, näited.
18. Millal tasub kasutada testimise automatiseerimise süsteeme? Oskus kasutada vähemalt kahte testimise automatiseerimise vahendit
19. Kvaliteet, standard, tarkvara, kvaliteedihalduse ja arenduse seos
20. Verifitseerimine ja valideerimine, protsessi ja toote kvaliteedihaldus, arendus- ja kvaliteedihalduse protsessid - võrdlus ja standardid
21. Kvaliteedihalduse käsitlusi, teese, süsteeme ja auhindu, ISO 9000 seeria
22. Mida peaks süsteemi arendaja (tellij, kasutaja, hooldaja, tarnija, firmajuht) teadma kvaliteedist ja standarditest?
23. Millest alustada tarkvaraprotsessi kvaliteedihalduse kavandamist?
24. Võrdlus: protsessihalduse raamistikud ja arendusmetoodikad. ISO 9000 seeria, ISO 90003, EVS-ISO/IEC 12207, CMMI, ITIL ja ISO/IEC 20000, RUP, XP, IT Grundschutzhandbuch / ISKE/ ISO 27000 seeria – sisu ja omavaheline võrdlus
25. Kvaliteedimeetrikate näiteid, väärtuse määramispiirkond, parim väärtus
26. Tarkvara mahu ja arendusmaksumuse prognoos, selle vahendid, usaldatavus
27. IT standardite ülevaade. Standardimine Eestis, Eesti IT alased standardid
28. Infosüsteemi audit, selle eesmärgid, auditeeritavad objektid, maht, audiitor, riskid, korraldus, infrastruktuur, COBIT

2. Põhimõisted

Jaotises esitatakse põhilised mõisted, mida täpsustatakse järgnevates peatükkides.

2.1. Tarkvara kvaliteet = toode + nõuded + protsessid

Me kasutame iga päev süsteeme, mille üks komponent on tarkvara. Tarkvarast võib sõltuda meie elu lennureisil, vara pangas või sissetulek, kui kasutame tarkvaravahendeid tööks või müüme tarkvara. Oluline on, et tarkvara töötaks.

Töökindluse tõstmise üks võimalusi on kvaliteedihaldus. Väga lühidalt on kvaliteet toote vastavus nõuetele. Keerukate toodete puhul tuleb vastavuse hindamisel arvesse võtta ka toote loomise protsessi. Seega seob kvaliteet *toote*, *nõuded* tootele ja tootmise *protsessi*. Vaatame neid lähemalt.

Meie *toode* on praegusel juhul tarkvara. See sisaldab palju komponente - kursuses kasutame tarkvara mõistet tarkvarasüsteemi sünonüümina, lülitades sinna dokumentatsiooni, meetodikaid, vahendeid jne. *Tootmisprotsessi* - tarkvara elutsükli - võib mitmeti valida ja esitada (näiteks, V-mudel, XP, prototüüpimine, testipõhine arendus jpt). Standardimine on vahend *nõuete* fikseerimiseks ja kvaliteedi edendamiseks, standardid esindavad universaalseid nõuete klasse.

Kvaliteeti ei saa sisse testida. Halvasti arendatud süsteemi pole võimalik kontrolli abil heaks muuta. Lõpptulemus sõltub kogu arendusprotsessist, sealhulgas vajadustele vastavast riistvarast, tarkvara arenduse meetoditest ja vahenditest, projekti- ja kvaliteedihaldusest, organisatsioonist ja standarditest. Käesolev kursus ei saa kõiki teemasid hõlmata, nii peavad kohe välja jääma näiteks tarkvara analüüsi, disaini ja kodeerimise teemad, mida antakse teistes ainetes. Sisse on võetud mitmesugused testimise, kontrolli ja hindamise meetodid ning kvaliteedi ja standardite laiemad mõisted, mida teised kursused enamasti ei kata. Eriti kontrolli korralduse osas vaadeldakse kaasaegseid arendusmeetodikaid, mis väga tugevasti integreerivad kontrolli tegevusi (XP, testipõhine arendus jne).

Erinevaid kontrolli meetodeid võib kasutada enamikus arendusprotsessi tegevustes. Kvaliteedihaldust võib kasutada ka nende tegevuste eneste parendamiseks. Kontrolli rakendatakse seega enamasti otsesele tulemusele (näiteks arendatav tarkvara), kvaliteedihaldust üldiselt - ka arendusprotsessidele, -meetoditele ja -vahenditele. Selle materjali kontrolli osas räägitakse siis rohkem toote parendamisest, kvaliteedihalduse osas - ka kogu organisatsiooni ja töökorralduse parendamisest.

Kvaliteedist rääkides mõeldakse tihti erinevaid asju: kvaliteet kui ideaal ("see on kvaliteettoode"); kvaliteet kui suhe toote, nõuete, protsesside vahel ("sealset elatustaset arvestades oli hotelli kvaliteet väga hea"); kvaliteet kui mõõt ("selle süsteemi kvaliteet on madal"). Käesoleva kursuse teemaks on kaks viimast mõistet.

2.1.1. Kontrollküsimusi ja ülesandeid

- Pakkuge ise kvaliteedi mõiste, võrrelge ülal pakutud mõistega
- Kas tarkvara kvaliteedi määratlus erineb teiste toodete kvaliteedi määratlusest? Miks?
- Millal võib kvaliteedi määratluses piirduda vaid tootega? Vaid toote ja nõudmistega?

- Kuidas suhtuda väitesse "Tarkvara kvaliteeti pole olemas, kogu aeg on kiirustamine ja pole aega ühte asja valmis saada, juba tuleb järgmine"? Millist kvaliteedi mõistet siin arvestatakse? Kas on võimalik, et kvaliteeti pole?

2.2. Tarkvaratoode

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

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

2.2.1. Kontrollküsimusi ja ülesandeid

- Tarkvara arenduse tulemid
- Millal võib tarkvaraarenduse tulemitest rääkides piirduda vaid programmiga? Millisega?

2.3. Nõuded

2.3.1. Nõuete liigid

Erinevatel osapooltel on erinevad nõuded. Omanik võib nõuda, et süsteem oleks kuluefektiivne; kasutaja võib soovida loetavat kirja ekraanil; hooldaja näeks heameelega arusaadavat koodi; jne. Tarkvaratehnika IEEE juhendmaterjal "Guide to the Software Engineering Body of Knowledge" (SWEBOK, <http://www.computer.org/portal/web/swbok/v3guide>) eristatakse toote ja (arendus)protsessi nõudeid. Toote nõuded spetsifitseerivad, milliseid funktsioone peab süsteem realiseerima (funktsionaalsed nõuded) ja kuidas neid funktsioone täidetakse (mittefunktsionaalsed nõuded). Protsessinõuded määravad arenduse kitsendused (näiteks, nõuded arhitektuurile, vahenditele või keskkonnale). Ärinõuded võivad lisaks sisaldada strateegilisi, keskkonna, maksumuse ja muid piiranguid.

Eri tüüpi nõuded võivad olla omavahel sõltuvuses. Enamasti tulenevad toote ja protsessi nõuded ärinõuetest. (Arendus)protsessi nõuded võivad suures osas tuleneda toote nõuetest. Käesolevas kursuses keskendutakse eelkõige toote, kursuse teises osas ka protsessi nõuetele.

Lõpptulemusena oleme huvitatud sellest, et meile vajalik toode (näiteks tarkvara või tarkvarateenus) oleks sobiva funktsionaalsusega, töökindel, turvaline, mugav ja nii edasi - ühe sõnaga, kvaliteetne. Aga millised on üldised tarkvaratoote kvaliteedi nõuded? Kui hakata neid põhjalikumalt arutama, selgub üsna pea, et kõigile ühtseid nõudeid ei ole ega saagi olla. Kasutusvaldkonnad, kasutajad, kriitilisus on selleks liiga erinevad. Väikeettevõtte kliendihaldustarkvara ja ülemaailmse piletite reserveerimise süsteemi võrdlemine on üsna lootusetu ettevõtmine.

Kui me ei tea, mida tahame, siis me seda enamasti ei saa. Kuidas siis ikkagi nõudmisi püstitada? Üks võimalus on anda ette võimalike nõudmiste nimekirjad, mida vastavalt vajadusele muudetakse. Kuna selliseid nimekirju võib teha ja on tehtud väga erinevaid, kirjeldame allpool kõigepealt üldise põhimõttena kvaliteedi atribuutide ja meetrikate süsteemi. Anname ka mõned konkreetseid atribuutide süsteemi näited, mida saab kasutada tarkvara projekteerimisel või hankel.

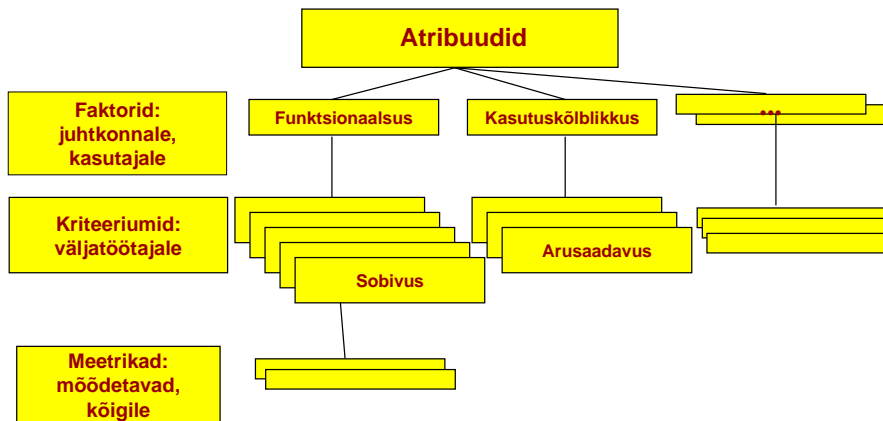
Teine võimalus nõuete üldisemaks määratluseks on kitsendada vaadeldava tarkvara klassi, ette andes selle funktsionaalsuse, rakenduspiirkonna või muud parameetrid. Väikeettevõtte raamatupidamissüsteemidele võib näiteks seada ühiseid nõudmisi. Riigiasutustes kasutatavale tarkvarale võib samuti seada mõistlikke tingimusi.

2.3.2. Kvaliteediattribuutide süsteem

On pakutud sadu võimalikke tarkvara kvaliteediomadusi (atribuute). Süstematiseerimata on neid raske kasutada. Seepärast on loodud mitmeid kvaliteediattribuutide esitamise süsteeme.

Üks selline süsteem on nõuete esitamine mitmetasemelise kvaliteediattribuutide puu kaudu (vt. joonis). 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 arendusele orienteeritud, 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 (seisakute kogukestvus või tõrgete arv ajaühikus kindla tööprofiili puhul), on arendusse ja hooldusse puutuvaid meetrikaid.

Kvaliteedinäitajate süsteem



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 mingi taseme jaoks, siis võib jätkata samamoodi ülespoole (juhul kui kriteeriume oli mitu taset). 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 raske põhjendatult 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 näiteks testimise standardis IEEE Std 829.

Selliseid süsteeme on mitmeid, allpool vaatame ühte rahvusvahelisest standardit.

2.3.3. Kvaliteedinäitajad standardi ISO/IEC 9126 / 25010 põhjal

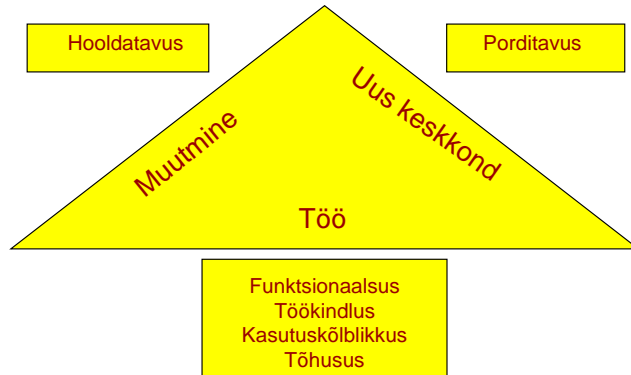
Üks levinud kvaliteedinäitajate skeem on esitatud rahvusvahelises standardis ISO/IEC 9126, mis on üle võetud ka Eesti standardiks (EVS - ISO/IEC 9126-1:2003 Tarkvaratehnika. Toote kvaliteet. Osa1: Kvaliteedimudel). Standard võtab arvesse kasutus-, välis-, sise- ja protsessi kvaliteeti. Elutsükli protsesside kvaliteet soodustab toote sise- ja väliskvaliteedi tõstmist, toote kvaliteet aga soodustab kasutuskvaliteedi tõstmist.

Kasutuskvaliteedi atribuudid on tõhusus, tööviljakus, ohutus ja rahuldus (vt jaotis 5.3).

Välis- ja sisekvaliteediatribuutide loogika on lihtsustatult järgmine. Süsteemi kasutatakse, muudetakse ja kantakse üle teistele riist- ja tarkvaraplatvormidele. Kasutamisel peaks süsteem

täitma vajalikke funktsioone ning olema töökindel, efektiivne ja kasutajasõbralik (vt joonis).

Kvaliteedifaktorid (ISO/IEC 9126)



See loogika viib järgmiste välis- ja sisekvaliteedi faktorite ja atribuutideni.

- Funktsionaalsus – atribuudid:
 - sobivus - kas kõik funktsioonid on olemas
 - õigsus
 - koostalitlusvõime teiste süsteemidega
 - turvalisus
 - funktsionaalsuse vastavus normidele
- Töökindlus – atribuudid:
 - küpsus - kui tihti esineb tõrkeid
 - tõrketaluvus - kuidas süsteem reageerib väliskeskkonna vigadele
 - taastuvus - kui raske on peale tõrget uuesti tööd alustada
 - töökindluse vastavus normidele
- Kasutuskõlblikkus – atribuudid:
 - arusaadavus
 - õpitavus
 - käsitletavus
 - meeldivus
 - kasutuskõlblikkuse vastavus normidele
- Tõhusus – atribuudid:

- ajaline käitumine
- ressursikasutus
- tõhususe vastavus normidele
- Hooldatavus – atribuudid:
 - analüüsitavus - kui raske on leida muutmise kohta
 - muudetavus - kui raske on muuta
 - stabiilsus - kui tugevalt muudatused mõjutavad süsteemi
 - testitavus
- hooldatavuse vastavus normidele
- Porditavus – atribuudid:
 - sobitatavus - kas süsteemi saab üle kanda teise keskkonda, näiteks teisele riistvara- või tarkvaraplatvormile
 - installeeritavus - kui raske on ülekanne
 - koosolvõime – tarkvara suutlikkus eksisteerida koos muu sõltumatu tarkvaraga ühises keskkonnas, kasutades koos ühisressusse
 - vahetatavus - tarkvaratoote suutlikkus olla kasutatav teise spetsifitseeritud tarkvaratoote asemel samaks otstarbeks ja samas keskkonnas
- porditavuse vastavus normidele

Täpsemad määratlused on toodud materjali lõpus. Välis- ja sisekvaliteedi faktorid ja atribuudid on ülevahtlikult toodud järgnevas tabelis.

Välis- ja sisekvaliteet

Funktsionaalsus	Töökindlus	Kasutus-kõlblikkus	Tõhusus	Hooldatavus	Porditavus
Sobivus Õigsus Koostalitlusvõime Turvalisus Funktsionaalsuse vastavus	Küpsus Tõrketaluvus Taastuvus Töökindluse vastavus	Arusaadavus Õpitavus Käsitsetavus Meeldivus Kasutus-kõlblikkuse vastavus	Ajaline käitumine Ressursikasutus Tõhususe vastavus	Analüüsitavus Muudetavus Stabiilsus Testitavus Hooldatavuse vastavus	Sobitatavus Installeeritavus Koosolvõime Vahetatavus Porditavuse vastavus

ISO/ IEC 9126 asendatakse ISO/ IEC 25000 seeriaga, milles on lisandunud süsteemi ja tarkvara kvaliteedi tasemed, uute faktoritena on sisse toodud turvalisus ja ühilduvus, on hakatud sisse tooma andmekvaliteedi atribuute jne. ISO/IEC 9126 pakub siiski ka praegu päris hea kvaliteediatribuutide nimekirja, mida saab vaadata veendumaks, et midagi olulist pole jäänud arvestamata.

Toodud nõuete komplektid ei ole ainuvõimalikud, lisaks võiks vaadata ka näiteks riigi IT koosvõime raamistikku (<http://www.riso.ee/et/koosvoime/raamistik>); mittefunktsionaalsete nõuete kirjeldamise juhendit avalikus sektoris (http://www.ria.ee/public/Mittefunk_nouded.doc); veebisaidi kasutatavuse nõudeid maailmas (<http://www.w3.org/TR/WCAG20/>), USAs

(http://en.wikipedia.org/wiki/Section_508_Amendment_to_the_Rehabilitation_Act_of_1973) ja Eestis (<http://www.riso.ee/et/koosvoime/internet>); jne.

2.3.4. Funktsionaalsed ja mittefunktsionaalsed nõuded

Ülaltoodud nõudeid võib jaotada funktsionaalseteks ja mittefunktsionaalseteks.

Funktsionaalsed nõuded vastavad küsimusele "Mida peab tarkvara tegema?" (näiteks, süsteem peab väljastama teatud aruanded laoseisu kohta). Ülaltoodud nimistus määravad funktsionaalsuse põhiliselt sobivuse ja õigsuse atribuudid. Mõnikord pannakse funktsionaalsete nõuete alla ka ülejäänud atribuudid funktsionaalsuse faktorist (koostalitlusvõime teiste süsteemidega; turvalisus; funktsionaalsuse vastavus normidele).

Mittefunktsionaalsed nõuded vastavad küsimusele "Kuidas tarkvara peab vajalikke funktsioone täitma?". Näiteks, süsteemi vastuse aeg peab jääma etteantud piiridesse (tõhusus); süsteem peab teatud ajavahemike jooksul tõrgeteta töötama (töökindlus) jne.

Tarkvara arenduse kursused käsitlevad tihti enamaltjaolt esimese faktori (funktsionaalsus) esimest atribuuti (sobivus), püüdes kaardistada tarkvara funktsionaalsust. Teised nõuded (õigsus, mittefunktsionaalsed nõuded) jäetakse kaardistamata, mis võib viia süsteemi arenduses suurte probleemideni. Üks põhjus selleks on, et sobivus üks kõige töömahukamaid osasid nõuetest.

2.3.5. Testitavad ja mittetestitavad nõuded

Nõuete püstitamisel on oluline, et nad oleksid testitavad. Funktsionaalsuse (täpsemalt, sobivuse) korral on see enamasti nii. Tõepoolest, kui näiteks püstitatakse nõue "süsteem peab väljastama jooksva hetke laoseisu", siis on võimalik seda testida – kontrollime, kas selline laoseisu väljastamise võimalus on olemas ja kas laoseis väljastatakse õigesti. Tegelikuses on olukord veidi keerukam (näidake, miks toodud nõude testimine võib olla raskendatud), kuid selliste nõuete testimine on mingil määral enamasti võimalik.

Mittefunktsionaalsete nõuete puhul on asi keerukam. Võtame näiteks nõude "süsteem peab olema töökindel". Kuidas seda nõuet testida? Ilmselt on see nõue sõnastatud ebapiisava detailsusega, meil ei ole kriteeriume hindamaks, kas mingi töökindluse tase on rahuldav või mitte.

Kui üldisi nõudeid täpsustada, muutuvad need (paremini) testitavateks. Näiteks, kas nõue "Süsteemi töö võib kuu aja pikkuses ekspluatatsioonis keskkonnas XYZ, kasutusaktiivsuse UVW ja kasutuslaadi NML korral olla häiritud maksimaalselt ühe tunni jooksul" on testitav? Kuidas? Kas selline täpsus on piisav? Tooge näiteid olukordadest, kus sellest täpsusest piisab ja kus mitte.

Otstarbekas on püstitada testitavad nõuded, muidu ei saa nende täidetust hinnata. Nõuete testitavuse hinnang on omalaadne spetsifikatsiooni test – kui nõuded ei ole testitavad, võib osutada, et spetsifikatsioon ei ole piisav (kas alati?).

Eesmärkide ja nõuete hindamiseks võib kasutada ka laiemat SMART kriteeriumit (*Specific, Measurable, Agreed, Realistic and Time bound*). Mis on siit puudu?

2.3.6. Kontrollküsimusi ja ülesandeid

- Tarkvara nõuete liigitusi
- Tarkvara kvaliteedi atribuutide struktuuri vajalikkus, näide struktuurist, integreerimine

- Näide kvaliteediatribuutide süsteemist, kahe faktori kriteeriumid
- Tooge näiteid funktsionaalsete ja mittefunktsionaalsete nõuete kohta
- Tooge näiteid testitavate ja mittetestitavate nõuete kohta
- Kas saab testida, kui nõuded puuduvad?
- Antud mittetestitav nõue, muuta testitavaks

2.4. Tarkvara arenduse protsessid

2.4.1. Protsessid, elutsükli ja nende mudelid

Tarkvara modelleerib tegelikkust ja võib olla väga keerukas, samuti võib olla väga keerukas selle arendus. Et selle keerukusega hakkama saada, on tarkvaraga seonduvaid tegevusi, tulemeid, dokumentatsiooni jne mõistlik kuidagi struktureerida.

Tuleks eristada tarkvara elutsükli mudeleid ja protsessiraamistikke. Tarkvara elutsükli mudelid, mida vaadatakse järgmises alajaotises, iseloomustavad esmajoones arendusprotsessi, protsessiraamistikud – kõiki protsesse.

Tänapäeval on pakutud mitmeid erinevad tarkvara protsesside raamistikke ja standardeid (nt ISO/IEC 12207, CMMI, COBIT, ITIL), mida vaadatakse materjali neljandas osas. Selliste raamistike mahukus tuleneb eelkõige sellest, et nad hõlmavad väga mitmesuguseid protsesse, mitte ainult tarkvara arendust. Näiteid protsessidest: hankimine, tarnimine, ekspluatatsioon, hooldus, konfiguratsiooni haldus, muudatuste haldus jne.

Üks varasemaid näiteid, mis sisaldab nii elutsükli mudeli kui ka protsessiraamistiku elemente, on Zachmani ettevõtte arhitektuuri raamistik (the Zachman Framework for Enterprise Architecture, http://en.wikipedia.org/wiki/Zachman_framework). Raamistik on kujutatav 6*6 tabelina, milles read sisaldavad erinevad osapooli või vaateid (Planner, Owner, Designer, Builder, Sub-Contractor, Functioning Enterprise) ja veerud – erinevaid süsteemi aspekte/küsimusi (What, How, Where, Who, When, Why). Muide, neid kuut aspekti saab kasutada väga mitmesuguste teemade korral.

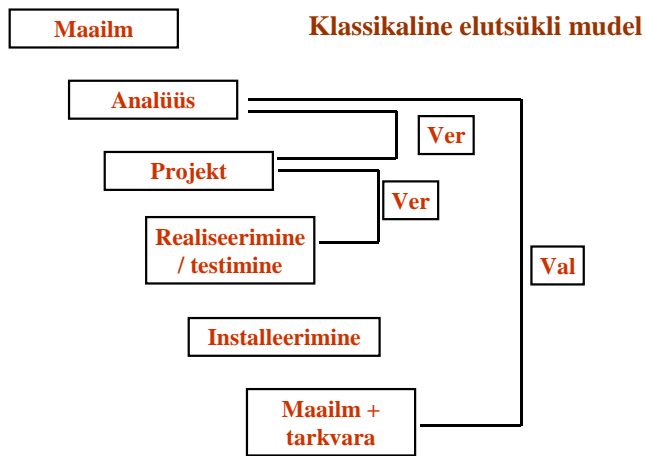
2.4.2. Tarkvara elutsükli mudelite näited

Tarkvara elutsükli mudelid hõlmavad esmajoones tarkvara arendust, mis on üks protsess paljudest. Toome näiteid erinevate tarkvara elutsükli mudelite kohta.

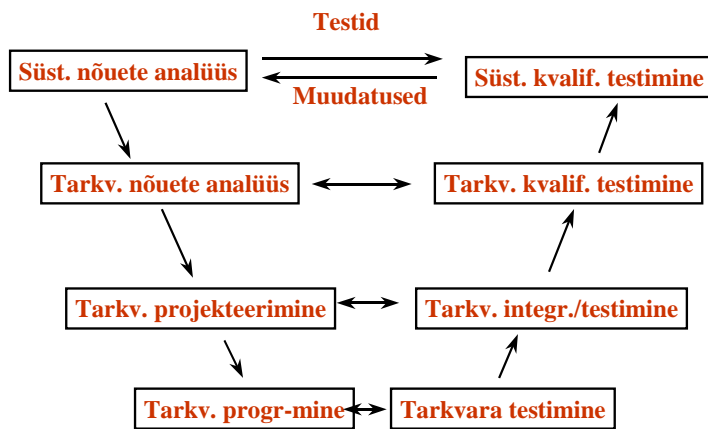
Rõhutame, et tarkvara protsesside raamistikud (nt. EVS-ISO/IEC 12207 või CMMI) ei eelda mingite kindlate etteantud elutsükli mudelite kasutamist.

Järgneval joonisel on klassikaline elutsükli mudel ("kosemudel"), millel on ära toodud ka verifitseerimise ja valideerimise tegevused. Esimene kosemudeli kirjeldus pärineb 1970-test aastatest. Kosemudeli idee pärineb tööstusest, kus muudatuste tegemine eelmise etapi tulemitesse võib olla väga kulukas või võimatu. Ülekantuna tarkvara arendusse tähendab see, et "puhta" kosemudeli puhul peaksid eelmised etapid olema lõpetatud enne, kui minna täitma järgmist etappi - tagasipöördumist eelmisse etappi ei ole. Võrreldes hoopis ilma struktuurita programmeerimisega (ülesanne -> programm) aitab kosemudel süsteemi paremini kavandada, võimaldab jagada tööd etappideks, parandab dokumentatsiooni. Samal ajal on süsteemi nõudeid raske täpselt ette kavandada, kosemudel viib tihti liiga jäiga ja bürokraatliku arendustsükli. Seepärast ei kasutata kosemudelit selle puhtal kujul tänapäeval eriti tihti (kuid siiski kasutatakse

– miks ja kus?). Samas selle mudeli ideed ja modifikatsioonid (näiteks, et arendus tuleks jagada etappideks; et sisuliselt võib olla vaja teha analüüsi, disaini ja hindamise tegevusi, kuigi võib-olla teisiti organiseeritult) on kasutusel mitmetes teistes elutsükli mudelites. Näiteks, kui lubada tagasipöördumist eelmistesse etappidesse, tekib iteratsioon.



Järgnev joonis esitab lihtsa V- mudeli. V-mudel is kulgevad arenduse ja kontrolli tegevused sümmeetriliselt - igale arendustegevusele vastab kontrolli tegevus. Valides erinevaid arenduse ja kontrolli tegevusi saame erinevaid V-mudelid. Joonisel olevas mudelis on sees tagasipöördumised eelmistesse etappidesse; kõigis V-mudeli versioonides seda ei ole.

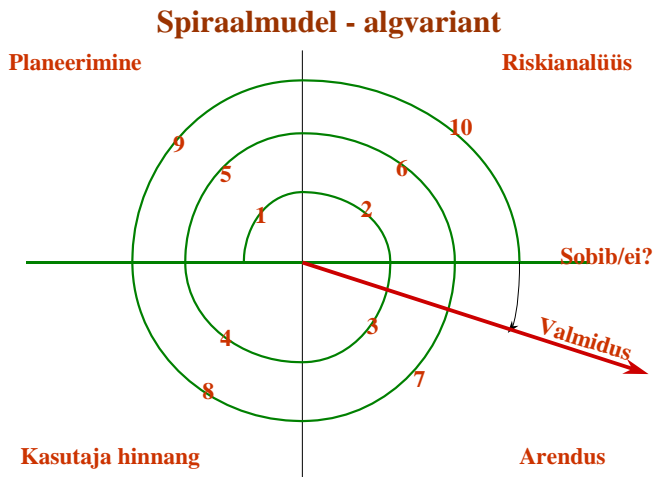


Tänapäeval on laialt kasutatavad mitmesugused iteratiivsed arenduse elutsüklid (näiteks XP meetoodika), mille allikaid võib kujutada alltoodud spiraalmudeli näitel. Spiraalmudeli pakkus välja Barry Boehm 1988.a. Mudel sisaldab nelja põhietappi, mis omakorda jagunevad alametappideks.

Esimene põhietapp on eesmärkide määratlemine ja planeerimine. Teisel etapil toimub valikute hindamine ja riskianalüüs. Kolmas etapp on tootearendus, mis sisaldab disaini, kodeerimist,

testimist ja integreerimist. Neljandal etapil saadakse kasutaja hinnang ning minnakse üle järgmiste etappide planeerimisele. Spiraalmudeli põhjal toimivas arenduses luuakse tavaliselt mitu tarkvaraversiooni. Esimene on prototüüp, järgmised võivad olla juba lõplikud, sõltuvalt kasutaja hinnangust.

Detailne spiraalmudel koos kõigi alametappidega on liiga kulukas väikeste projektide jaoks, kuid selle üldist loogikat saab rakendada ka väikesemahulistes projektides ning seda on rakendatud ka erinevate hilisemate meetodikate väljatöötamisel.

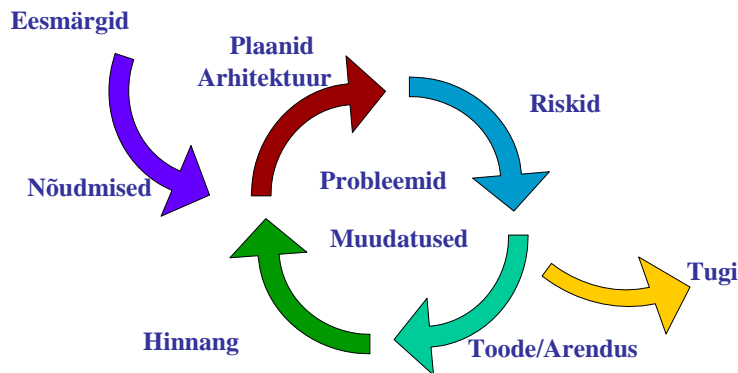


XP (Extreme Programming) ideid ning seoseid testimisega on kirjeldatud materjali teises osas.

Alguseks ja väikeste projektide puhul on põhjalikud raamistikud väga mahukad ja keerukad. Tegelikult nõuab siiski iga mittetriviaalse raamistiku süstemaatilisele kasutusele võtmine pingutust. Kas oleks võimalik sellistest raamistikkudest loobuda ja pigem püüda määratleda, millised aga võiksid olla tarkvara arenduse ja juhtimise põhikomponendid? Siis võiksime nendest ise oma tarkvara elutsükli mudeli koostada.

Ega ühtset loetelu ei ole, erinevates meetodikates on need erinevad (seepärast muide ongi universaalsed raamistikud keerukad, et nad püüavad hõlmata kõiki olukordi). Näiteks võib osaliselt Rational Unified Process (RUP) meetodikale tuginedes pakkuda valiku järgmistest tarkvaraprotsessi põhikomponentidest: eesmärkide püstitamine ja ärivaate loomine, nõudmiste spetsifitseerimine, projektiplaani koostamine, arhitektuuri planeerimine, riskide ja nende maandamise võimaluste hindamine, arendus, toote hindamine, probleemide haldamine, muudatuste haldamine, tugi (vt. joonis). Seda valikut võib oma vajaduste järgi modifitseerida.

Spiraalmudel - 10 põhikomponenti (näide)



Näeme, et programmi kirjutamine, mis esimesel pilgul võib näida tarkvara protsessides põhilisena, on oluline, kuid tegelikes rakendustes siiski vaid üks osa ühest põhikomponendist (arendusprotsess). Muuhulgas, ka kursuse esimeses osas vaadeldavad kontrolli meetodid on kasutatavad arenduses ja mõnedes muudes komponentides.

2.4.3. Hankija tegevused

On oluline, et hankija osaleks hankes aktiivselt. Keeruka tarkvara hange on võrreldav maja ehitamisega – kui loota, et ehitaja teeb ilma tellija osaluseta valmis sobiva maja, võib oodata mõlema poole pettumust.

Erinevates elutsükli mudelites on hankija eeldatav osavõtt erinev, kuid mingi põhiline tegevuste komplekt on hädavajalik. Näiteks pakub ISO/IEC 12207 järgmised hankimise tegevused:

- hankimise ettevalmistamine (sh vajadus, süsteemi- ja tarkvaranõuded, hankimisplaan, vastuvõtustrateegia ja –tingimused, protsessid)
- hanke väljakuulutamine
- tarnija valimine (sh valiku protseduurid)
- lepingu sõlmimine (sh õigused, muudatuste ohje)
- leppe seire (kasutades teisi protsesse)
- hankijapoolne vastuvõtmine (sh vastuvõtuläbivaatus ja vastuvõtutestimine)
- sulgemine.

CMMI materjal "CMMI for Acquisition (CMMI-ACQ)" (<http://www.sei.cmu.edu/library/abstracts/reports/10tr032.cfm>) annab põhjaliku ülevaate hankimistegevustest. Konkreetset hange puhul võib neid tegevusi kasutada kontrollimaks, et mindagi olulist pole unustatud. Tegevusi tuleb kohandada erinevatele elutsükli mudelitele, näiteks agiilsete meetodite puhul on selleks antud soovitusi materjalis <http://www.sei.cmu.edu/reports/10tn002.pdf>.

Hankija tegevused on põhjalikult kirjeldatud COBIT raamistikus (<http://www.isaca.org/Knowledge-Center/cobit/Pages/Downloads.aspx>).

Toodud tegevustes sisalduvad ka näiteks nõuete muudatuste haldus ja vastuvõtmise plaani koostamine.

Viidatud standardid ei eelda, et kõiki tegevusi tuleb täita, vaid pakuvad võimaluse teha tegevustest valik vastavalt olukorrale. Valitud tegevuste sisu ja ulatuse põhjal võib prognoosida hankija töömahtu hankes.

2.4.4. Kontrollküsimusi ja ülesandeid

- Näited erinevate elutsükli mudelite kohta
- Kosemudel
- V-mudel
- Spiraalmudel
- XP
- Millal pole mõtet rääkida tarkvara elutsüklist?
- Milline elutsükli mudel on parim?
- Tarkvara elutsükli mudelid ja protsesside mudelid – mis on vahe?
- Millised on hankija tegevused, kuidas nad sõltuvad arendusprotsessist ja hankijast?

3. Tarkvara kontroll: meetodid ja korraldus

Tarkvara kvaliteeti saab parendada mitmete kontrolli meetoditega, näiteks tarkvara testimise ja läbivaatustega. Need kontrolli meetodid on praktilised ja kohe, sealhulgas ka iseseisvates töödes, kasutatavad. Seepärast vaadatakse neid alguses, jättes üldisemad kvaliteedihalduse teemad järgmisse osasse.

3.1. Testimine

Igäüks, kes on koostanud mingi programmi, on seda ka arvatavasti käivitanud ja katsetanud. Vastutusrikaste süsteemide puhul peavad sellised katsetused (testid) olema süstemaatilised, põhjalikud ja andma hinnangu toote omadustele, nt töökindlusele või tootesse jäänud vigade arvule. Allpool vaadatakse testimise põhimõtteid ja meetodeid.

3.1.1. Testimise põhimõtteid

Testimine on levinud tegevus ning sellele on antud erinevaid definitsioone. Kitsamas mõttes on testimine tarkvara täitmine / käivitamine kontrollimaks, kas ta vastab ettenähtud nõuetele ning leidmaks vigu¹. Käesolevas materjalis on enamasti kasutatud kitsamat testimise mõistet, et eristada testimist ning teisi kontrolli tegevusi (näiteks läbivaatused ja tõestamine).

Laiemas mõttes on testimine tarkvara analüüsi protsess eesmärgiga leida erinevusi olemasolevate ja nõutud tingimuste vahel ning hinnata tarkvara omadusi². Testimist võib laias mõttes määratleda ka kui kõikidest elutsükli tegevustest (nii staatilistest kui ka dünaamilistest) koosnevat protsessi, mis puudutab tarkvara ja sellega seotud toodete planeerimist, ettevalmistust ja hindamist ning mille eesmärk on kindlaks teha toodete vastavus spetsifitseeritud nõuetele, näidata et nad vastavad eesmärgile ning leida defekte³.

Ka mõistet "test" ning sellega seonduvaid mõisteid on defineeritud erinevalt, näiteks: test on komplekt ühest või mitmest testjuhust (ANSI/IEEE 829). Testjuht (test case) on lihtsaimal juhul komplekt sisendandmeid, täitmise tingimusi ning sisendandmetele vastavaid oodatavaid väljundandmeid (ANSI/IEEE 829). Testjuhtu võib aga defineerida ka kui komplekti sisendandmeid, täitmise eeltingimusi, oodatavaid väljundandmeid ning täitmise järeltingimusi, mis on arendatud kindlal eesmärgil, näiteks selleks, et täita etteantud programmi haru või kontrollida vastavust teatud nõudele (IEEE, ISTQB).

Sisendandmeid saadakse väga erinevalt (nt tuginedes nõuetele või programmi tekstile). Mitmete testimise meetodite eesmärk ongi sisendandmete tekitamine. Testide sisendid püütakse leida nii, et suurendada vigade leidmise tõenäosust ja minimeerida testide mahtu.

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

¹ British Computer Society Specialist Interest Group in Software Testing, BCS SIGiST, http://www.testingstandards.co.uk/living_glossary.htm#T

² Standardist "ANSI/IEE Std 829 Standard for Software Test Documentation"

³ International Software Testing Qualifications Board, ISTQB, <http://istqb.org>.

Programm on vigane, kui tegelik väljund ei vasta oodatud väljundile. Kui leitakse viga, siis oli test edukas (kas on võimalik ka teistsugune seisukoht?). Hea on test, mis avastab palju vigu. Testimine ei tõesta, et programmis vigu pole (miks? kuidas võiks seda tõestada?) - ta võib vaid näidata, et programmis on vigu.

Arendajad viivad läbi esimesed testid, millest üldjuhul ei piisa - autor jälgib oma loogikat ega suuda prognoosida kasutaja võimalikke tegevusi, autor pole alati piisavalt enesekriitiline ja ei taha oma tööd "lõhkuda", autori otsene motivatsioon on lõpetada töö jne. Häid tulemusi võib oodata süsteemi tulevastelt kasutajatelt, kes ei ole arendusega seotud. Vastutusrikkamatel juhtudel rakendatakse eraldi testijaid.

Efektiivseks testimiseks ei piisa vaid süsteemist. On vaja teada nõudeid ja protsesse, mis omakorda tulenevad ülesande püstitusest, organisatsioonist, seadusandlusest, standarditest, riskianalüüsist, headest tavadest, kasutajatest, kasutusviisist jne (vt materjali teist osa ja kursuse korralduse failis iseseisvate tööde kirjeldust).

Testide projekteerimisel tekib hulk küsimusi, näiteks:

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

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

Kuidas testimise meetodid on tekkinud? Alustame algusest: 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 neist on proovimine, seda tehakse kasvõi huvi pärast. Valikud 2,3,5 testivad (minimaalset) funktsionaalsust, millest kasutaja kõige rohkem huvitatud võib olla. Valikut 4 võib iseloomustada kui eksperdi testimist ja veaotsingut. Valik 6 läheneb testimisele rohkem programmi/süsteemi seisukohast, programmipõhist testimist vaadatakse järgmistes jaotistes. Valikud 7 ja 8 (mingil määral ka 5 ja 6) üritavad süstemaatilist läbitestimist, mille levinud liik on ekvivalentsiklasside ja piirjuhtude analüüs.

Testimisele võib esitada vastuväiteid.

- Testimine ei tõesta niikuinii õigsust, parem juba tõestada programmi. - Programmide formaalne tõestamine on mõnes mõttes hea, kuid väga töömahukas ega pruugi avastada nt spetsifikatsiooni vigu (vt järgmistes jaotistes). Testimine ja programmi tõestamine täiendavad teineteist.
- Mis kasu on neist meetoditest, kui nad ikkagi ei garanteeri veakindlust, ma parem proovin niisama vigu leida? - Veaotsing, uuriv ja suitsutestimine on samuti testimise meetodid. Käsitletavad meetodid nõuavad üldjuhul süstemaatilist testimist. Tulemusena teame, et kõiki asju on katsetatud teatud nivool, mis sõltub ressurssidest ja vajadustest. Mõni testimismeetod annab ka töökindluse ja vigade arvu prognoosi. Need meetodid täiendavad teineteist.

Veel mõisteid (sisulised määratlused).

- Valideerimine - tegevus, mis püüab näidata, et tehtud on seda, mis vaja.
- Verifitseerimine - tegevus, mis püüab näidata, et järgmise etapi tulemus vastab eelnenud etapi määratlusele.
- Silumine - leitud vea kõrvaldamine.
- Sertifitseerimine - kolmanda osapoole tegevus, mis hindab toote, teenuse või protsessi vastavust standarditele ja normdokumentidele ja väljastab vastavuse korral (vastavus)sertifikaadi.

3.1.2. Riski- ja ekspertteadmiste põhine, uuriv jm testimine

Enne kui hakata spetsifikatsiooni- või programmipõhiselt süstemaatiliselt testimata, võib olla otstarbekas läbi viia esialgne testimine, mis selgitab, kas on mõtet põhjalikumaid meetodeid rakendada. Allpool on toodud mõned näited selleks sobivatest meetoditest.

Samuti kuuluvad sellised meetodid mitmete arendusmeetodite koosseisu ning neid saab kasutada lisaks süstemaatilistele testidele.

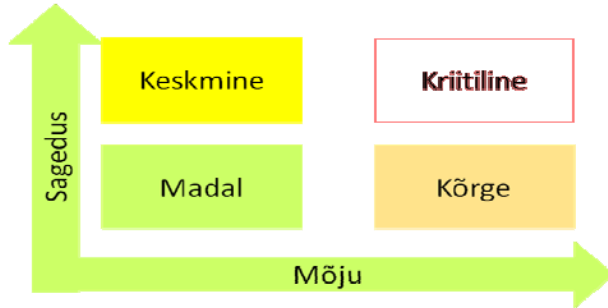
Neid meetodeid on mõtet kasutada ka siis, kui ülesanne pole väga kriitiline, testimise ressursid on napid või süstemaatilise testimise oskus puudub.

Toodud meetodid on tüüpiliselt kiired ja kuluefektiivsed, aga mitte väga süstemaatilised ja detailsed.

Riskipõhine testimine

Riskipõhise testimise idee on testida esmalt tootega seotud kriitilisi riske. Selleks on vaja identifitseerida riskid, omistada neile prioriteedid, testida kõige prioriteetsemad riske, informeerida teisi osapooli tulemustest ning võtta vastu otsused edasise kohta.

Riski prioriteeti saab iseloomustada mõju ja sageduse (tõenäosuse) kombinatsiooniga, näide on toodud järgnevas tabelis.



Riske võib otsida näiteks järgnevast (osa neist ei pruugi olla testitavad).

- Tellijale või kasutajale kõige suuremat väärtust andvad ja sagedamini kasutatavad süsteemi funktsioonid (risk on siis, et need ei toimi).
- Mittefunktsionaalsete nõuetega seotud riskid: töökindlus, efektiivsus, turve, kasutatavus jne.
- Kõige suuremad ohud, nt valesti teostatud makse, katla plahvatus jne (risk on siis, et need ohud realiseeruvad). Ohud ei pruugi olla seotud põhifunktsionaalsusega.
- Tehnilised omadused, nt süsteemi uudsus ja keerukus, suured andmemahud, osapoolte arvukus jne.
- Arendusprotsessiga seotud riskid, nt kiired tähtjad, tellija või täitja ressurside vähesus jne.

Riskide analüüsiks võib kasutada mitmeid meetodikaid (nt FMEA) ja standardeid (nt ISO/IEC 27005).

Riskipõhise testimise tulemuste põhjal võib võtta vastu otsused riskide vähendamiseks, aktsepteerimiseks või jagamiseks, tegevuse lõpetamiseks jne.

Ekspertteadmiste põhine testimine

Kogenud arendaja või testija 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 programmeerimiskeele) 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. Mingi konkreetse valdkonna tüüpigade analüüs võib olla iseseisva (diplomi-, magistri-) töö teema. Staatiliste meetodite jaotises toodud küsimustikud võiksid olla näiteks programmeerimiskeeltele orienteeritud veaotsingu meetodikast.

Veatsing on väga efektiivne vahend, mida võib kasutada süsteemselt või intuiitiivselt. Esimesel juhul on ees küsimustikud kindla valdkonna kohta, mida süstemaatiliselt läbi vaadatakse. Sellisena on meetod kasutatav ka iseseisvates töödes. Intuiitiivset laadi veatsingu juures jääb alles mittedüsteemsete meetodite puudus - testimine kipub olema juhuslikku laadi; eeliseks on tugeva eksperdi puhul head testid ja aja kokkuhoid.

Selle meetodi eeldus on kas eksperdi või abivahendite (nt küsimustikkude) olemasolu.

Kommentaari: testimine on loominguiline tegevus ja intuiitiivne 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 tarkvara puhul või omas rakendusvaldkonnas.

Uuriv testimine

Uuriv testimine (exploratory testing) on mitteformaalne tarkvara testimise tehnika, mille puhul testija hindab testide kavandamist nende täitmise käigus ning kasutab saadud informatsiooni uute ja paremate testide projekteerimiseks (www.istqb.org).

Suitsutestimine

Suitsutestimisel (smoke testing) täidetakse alamhulk kõigist testidest selgitamas, kas põhilised funktsioonid töötavad. Nimetus tuleneb elektroonikatööstusest (seadme esmane sisselülitamine).

3.1.3. Spetsifikatsiooni põhine testimine

Spetsifikatsiooni põhise testimise puhul vaatame programmi kui musta kasti, sest me ei tea tema siseehitust, teame vaid sisendeid ja väljundeid (spetsifikatsiooni). Erinevus programmipõhise testimisega ongi selles, mille põhjal leitakse testide sisendid (spetsifikatsioon või programmi tekst); väljundid tekitatakse mõlemal juhul spetsifikatsiooni alusel.

Spetsifikatsiooni põhisest testimisest vaatame ekvivalentsiklassidel, piirjuhtudel, otsustustabelitel ja kasutusjuhtudel (kasutusmallidel) põhinevaid meetodeid. Kõik need meetodid tuginevad vähemalt kahel lihtsustusel.

- Kogu sisendite ala jaotatakse klassideks (ühte klassi võib kuuluda vaid üks väärtus, nt piirjuhtude puhul), mille kohta eeldatakse, et süsteem käitub kõigi samasse klassi kuuluvate andmete testimisel ühtemoodi. See lihtsustus võimaldab lõpmatu arvu sisendite asemel vaadelda lõplikku hulka klasse.
- Testimisel püütakse katta kõik klassid. Kuna kõigi klasside kombineerimine viib ikkagi liiga suure arvu testideni, siis rakendatakse teist lihtsustust: püütakse klasse testidesse panna nii, et testide arv oleks minimaalne.

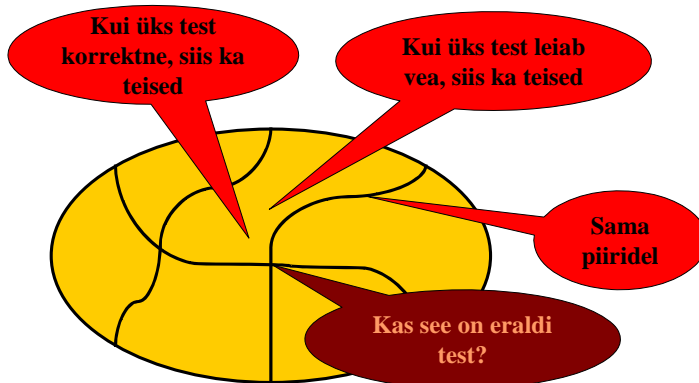
Erinevate meetodite puhul rakendatakse seejuures erinevaid testide üleskirjutuse ja süstematiseerimise variante.

Kasutusjuhtude, olekudiagrammide, jadadiagrammide ja mõnede teiste protsessile orienteeritud formalismide puhul saame testimisel kasutada ka struktuurse (programmipõhise) testimise meetodite analooge.

Ekvivalentsiklasside analüüs

Ekvivalentsiklasside analüüsi idee on selles, et sisendandmed jaotuvad töötuse suhtes enamasti rühmadesse, nii et ühes rühmas asuvad andmeid töödeldakse ühtemoodi (vt. joonis).

Ekvivalentsiklassid ja piirjuhud

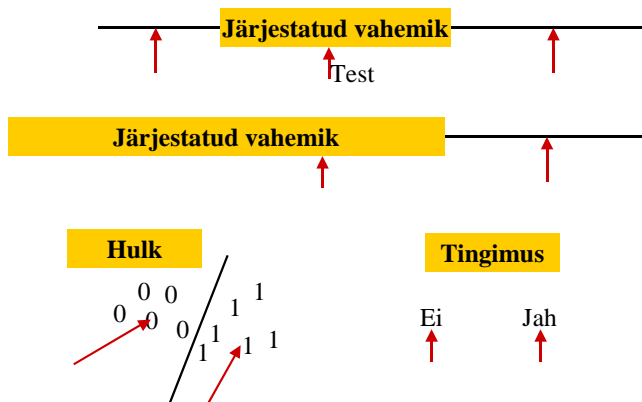


See meetod püüabki leida selliseid sisend- ja väljundandmete klasse, et (1) kui klassist K leitakse viga, siis leitakse sama viga ka teistsuguste andmetega klassist K ja (2) kui klassis K viga ei ilmne, siis ei ilmne viga ka teistsuguste andmete puhul klassist K. Seega piisab sellest, kui valida üks punkt (testandmed) klassist K, et testida kõiki samast klassist andmeid.

Kui ekvivalentsiklassid on valitud, tuleb nende põhjal valida testandmed. Seda võib teha järgmiste põhimõtete alusel. Kui ekvivalentsiklass on (vt. joonis):

- järjestatud tõkestatud vahemik, siis võtta testandmed vahemiku seest, enne vahemikku ja pärast vahemikku
- järjestatud tõkestamata vahemik, siis võtta testandmed vahemiku seest ja väljast
- ühte moodi käituvate elementidega hulk, siis võtta testandmed hulga seest ja väljast
- erinevalt käituvate elementidega hulk, siis testida kõiki elemente
- tingimus, siis proovida mõlemaid variante (tingimus täidetud/täitmata)

Ekvivalentsiklassid (ka väljundile)



Ülaltoodud tegevused käisid ühe sisendi kohta. Sisendeid võib olla palju ja ka testandmed tuleb valida kõigi sisendite kõikide ekvivalentsiklasside jaoks.

Testandmed kombineeritakse kõiki sisendeid arvesse võttes testidesse nii, et kõik valitud testandmed oleksid kaetud. Õigete andmete klasse püütakse testi sisse panna korraga maksimaalselt, vigaseid sisendeid ühekaupa. Põhjus on selles, et peale ühe vigase sisendi töötlemist võib programmi töö lõppeda ning teisi vigaseid sisendeid enam ei analüüsita. Seega mitme üheaegse vigase sisendi korral me ei ole kindlad, kas kõiki neid töödeldi korrektselt. Sama kehtib mõnikord ka piirjuhtude korral (millal?).

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

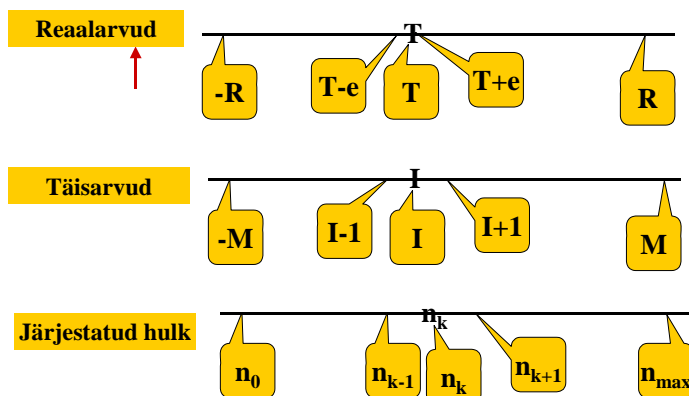
Ekvivalentsiklassid tekitatakse nii sisendite kui ka väljundite jaoks.

Piirolukorrad

On leitud, et vigu esineb palju ekvivalentsiklasside piiridel, seega tasub teha piirolukordade teste. Selliste testandmete näiteid (vt. joonis).

- 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 $n_0, n_1, \dots, n_k, \dots, n_M$ 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

Piirjuhtude testid (ka väljundile)



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

Ekvivalentsiklasside ja piirjuhtude põhise testimise analüüs

Ühe sisendi korral, kus lubatavad väärtused on antud intervallina, tuleb siis kokku kolm ekvivalentsiklassi ja kaheksa piirlokorda. Kui sisendeid on palju, võib testide (vajalike sisendandmete kombinatsioonide) arv olla väga suur. Eespool vaatasime juba, kuidas vähendada testide üldarvu erinevate sisendite ekvivalentsiklasside omavahelise kombineerimise teel. Lisaks võib analüüsida igat ekvivalentsiklassi ja piirjuhtu eraldi, küsides:

- Kui suured on töökindluse nõuded (milline testimine on üldse vajalik, kas võib testide arvu vähendada)?
- Kas meid huvitab täpne piirist üleminek (kui ei, võib kaaluda ühte piiril olevat või sellele lähedast väärtust, vt ka järgmine punkt)?
- Kas vealokordi võib esitada ühe ekvivalentsiklassina (kui jah, saab piirduda vealokorra klassis ühtede testandmetega)?

Sõltuvalt nõudmistest võib siis näiteks ühe intervallina antud sisendi korral testväärtuste arv väheneda (milline on väikseim võimalik testide arv?), tulemusena väheneb testide üldarv.

Funktsionaalse testimise võib etappideks jagada järgmiselt:

- 1) eristada sisend- ja väljundandmete ekvivalentsiklassid, sealhulgas erinevate sisendite/väljundite kombinatsioonid,
- 2) määrata igas klassis, nende piiridel ja vajadusel kombinatsioonidel testandmed,
- 3) ühendada testandmed testidesse, määrates ka vastavad väljundandmed (või sisendid, kui analüüsiti väljundi ekvivalentsiklasse),
- 4) identifitseerida testid, koostada testimise plaan,
- 5) testida, võrrelda tulemusi, hinnata.

Leitakse, et funktsionaalne testimine on hinnaefektiivsem kui programmipõhine (vea leidmise maksumus on väiksem). Funktsionaalsetest meetoditest on omakorda efektiivsem piirjuhtude meetod, kuigi sellest enamasti ei piisa (miks?).

Kokkuvõtte funktsionaalsest testimisest ekvivalentsiklasside ja piirjuhtude põhjal.

- Idee: süstemaatiline testimine sisendi/väljundi (spetsifikatsiooni) põhjal
- Eeltingimused: vajadus; spetsifikatsioon on mingil kujul olemas; selle analüüs on teostatav
- Eelised: kasutatav funktsionaalsus on süstemaatiliselt testitud; kasutajale arusaadavam kui programmi teksti põhine testimine; õigel arendamisel koostatakse testid juba spetsifikatsiooni koostamise ajal, mis võimaldab ühtlasi testida spetsifikatsiooni; hinnaefektiivsem kui programmipõhine testimine
- Puudused: spetsifikatsiooni pole alati olemas. Ei pruugi avastada funktsionaalsusega mitte seotud koodi. Kui ekvivalentsiklassid on sõltuvuses, võib testimine olla mahukas
- 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

Otsustustabelid

Kui sisendid on omavahelises sõltuvuses, siis võib olla kasulik otsustustabelite põhine testimine. Otsustustabel sisaldab eeltingimusi, tegevusi ja reegleid (vt nt http://en.wikipedia.org/wiki/Decision_table). Iga reegli kohta saab defineerida testi.

Otsustustabeleid saab kasutada ka ekvivalentsiklasside ja piirjuhtude korral, eriti kui sisendmuutujad on sõltuvuses. Kui sõltuvusi pole, kas siis on mõtet otsustustabelit kasutada?

Kasutusjuhud (kasutusmallid, use case)

Kui süsteemi on spetsifitseeritud kasutusjuhtude abil, saab nende abil kavandada ka teste. Kuna kasutusjuhud on tihti ülataseme spetsifikatsioonid, mis määratlevad süsteemi üldist käitumist, võib selline testimine olla väga kasulik. Samas võivad kasutusjuhud sisaldada mitmeid alternatiivseid stsenaariume või olla väga üldised. Sellisel puhul võib osutada vajalikuks teha ühe kasutusjuhu kohta arvukalt teste.

3.1.4. Testimine programmi teksti põhjal

Programmi teksti põhise testimise puhul võetakse ette programm ja püütakse teste luua selle teksti põhjal. See on vajalik, sest ainuüksi funktsionaalsest testimisest ei piisa (samuti nagu ei piisa ainult programmpõhisest testimisest – mis jääb puudu?). Tõepoolest, vaadates programmi ainult väljastpoolt selle käitumise seisukohast, me ei leia näiteks harusid, mida tavalisel täitmisel ei läbita.

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

Programmi teksti põhisel (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-, teadekvaatsuse ja muid kriteeriume.

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

Vaatame kõigepealt lauseadekvaatsuse kriteeriumit.

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, mitut testi on vaja programmi lauseadekvaatseks kontrollimiseks. Näiteks, kui programmis pole kordust ja on üks *if-then-else*, siis on vaja vähemalt kaht testi, sest ühe testiga ei ole võimalik läbida mõlemat haru.

Kokkuvõtte 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: mitte eriti tugev kriteerium. 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
- Vahendid: on olemas vahendid, mis aitavad leida lauseadekvaatset testikomplekti

Haruadekvaatus ja muud

Haruadekvaatus

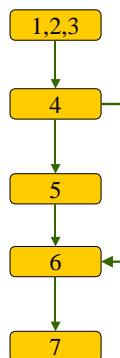
Lauseadekvaatsuse puhul läbitakse kõik laused, kuid harud, milles lauseid pole, jäetakse läbimata. Haruadekvaatsuse nõue eeldab ka tühjade harude läbimist, seega on ta täielikum, $Lauseadekvaatus \leq Haruadekvaatus$. Haruadekvaatsust saab illustreerida programmi graafil. Sellel vastab igale hargnemisele graafi tipp, millest väljub rohkem kui üks haru. Üksteisele järgnevad täidetavad hargnemiseta laused võib ühendada üheks tipuks. Haruadekvaatsuse nõuet võib sõnastada järgmiselt: testimise käigus peavad programmi graafi kõik kaared olema läbitud.

Järgneval joonisel on kujutatud lihtne programm ja sellele vastav graaf. Graafis on kolm esimest lauset ühendatud üheks tipuks. Selle programmi lauseadekvaatseks testimiseks piisab ühest testist, mis läbib lause 5 (tooge näide). Samas haruadekvaatseks testimiseks tuleb teha vähemalt kaks testi - eelmisele lisaks ka test, mis läbib tühja else-haru. Kui selles harus oleks mingi lause, siis oleks nii lause- kui ka haruadekvaatseks testimiseks vaja teha vähemalt kaks testi.

Haruadekvaatus ja programmi graaf: Leida maksimum

```

1 Function max (a,b)
2   Read a,b
3   max := a
4   If b>max
5     Then max := b
6   End If
7 End Function
    
```



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 selle koostamise 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
- keerukust saab hinnata juba projekti staadiumis, s.t enne programmi tegelikku koostamist
- saab kasutada arenduses oleva mooduli hindamiskriteeriumina, ühe mooduli $V(G)$ mõõt peaks olema ≤ 10

Elementaartingimuste adekvaatsus

Kui If-lause tingimus on loogiline avaldis, siis tekivad selle avaldise läbimisel sisuliselt programmi harud, kuigi näiliselt 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 ongi **elementaartingimuste adekvaatsus** - ka loogilise avaldise harud peavad olema testide käigus läbitud. Ka see on üsna mõistlik kriteerium ja praktikas kasutusel.

Teeadekvaatsus

Viimase kriteeriumina sellest klassist vaatame nõuet, et programmi testimisel peavad kõikvõimalikud teed programmi graafis olema läbitud. Idee on selge ja kena, kuid tsükleid sisaldavate reaalsete programmide puhul on vajalike testide maht enamasti väga suur ja see ei luba teadekvaatsuse kriteeriumit selliste programmide testimisel kasutada. Ilma tsükliteta programmi puhul võib see olla hea kriteerium.

Andmepõhine testimine

Ekvivalentsiklasside, piirjuhtude ja veaotsingu ideid saab kasutada ka programmpõhisel 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.

Tsüklite testimine

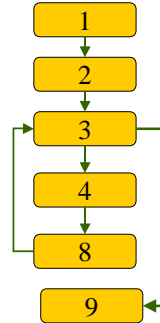
Tsüklite (korduste, iteratsioonide, silmuste) testimisel võib eristada testimist mingi testimiskriteeriumi või -meetodi põhjal ja tsükli kui eraldi konstruktsiooni testimist. Esimesel juhul koostame vajalikke teste valitud meetodite või kriteeriumide põhjal (näiteks haruadekvaatsuse kriteerium). Väikest segadust võib tekitada testide arv. Näiteks kui ühe programmi käivitamise puhul täidetakse tsükkel mitu korda, läbides kokkuvõttes kõik harud, siis võib tekkida küsimus, kas see on üks test või mitu. Vastus sõltub testimise ülesandest ja arveldamise vajadusest, kuid igal juhul tuleb selgitada, kuidas testide arv saadi.

Tsükli testimiseks võivad eeltoodud meetodid olla nõrgad, sest vead võivad olla seotud tsükli läbimiste arvuga. Teisel juhul lähtumegi tsüklist kui eraldi konstruktsioonist ja testimise sisuliselt andmepõhiselt juhtparameetri(te) järgi. Selleks on pakutud järgmist meetodit.

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

Ühetasemeline tsükkel

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



Mitmetasemelise tsükli puhul (k taset) võib kasutada järgmist protseduuri (vt. joonis)

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

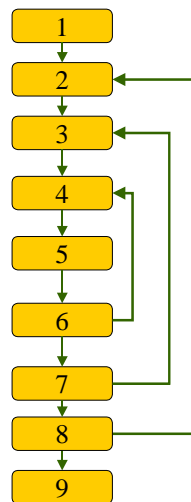
Mitmetasemeline tsükkel

```

.....
While....
Begin.....
  ....
  While...
  Begin
    ....
    End While
  ....
End While
.....

```

Näide:
 Aastate,
 Osakondade,
 Inimeste...
 lõikes



Toodud protseduur vähendab oluliselt testide arvu mitmetasemeliste tsüklite puhul.

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

Struktureerimata tsüklid tuleb enne muuta struktuurseiks.

3.1.5. Juhuslikud sisendid ja lisatud vead

Eelmistes jaotistes toodud meetodid on lihtsad ja enim kasutatavad. Testimise meetodeid on siiski palju ja kriitilisemate ülesannete puhul tuleb otsida tugevamaid. Ülalvaadeldud meetodite üheks puuduseks on, et nad ei võimalda hästi hinnata tarkvara töökindlust, vigade arvu jne. Järgnevad näited võimaldavad seda mingil määral teha.

Testimine juhuslike andmetega

Esimesel pilgul näib, et kes testimisest põhjalikumalt kuulnud pole, see just nii testibki - proovib mingite andmetega, kuidas töö läheb. Programmeerimise algusaastatel oli selline testimine levinud meetod. Siis leiti, et praktiliselt jäävad nii olulised vead avastamata. Seepärast jäi kauaks püsima seisukoht, et juhuslike andmetega testimine ei ole soovitatav.

1980-te aastate lõpus uuriti teemat täpsemalt ja leiti, et see testimismeetod võib olla vägagi efektiivne, kuid ainult kindlatel tingimustel:

- 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 piirulukordade) 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:

- testide arv võib olla väga suur
- testide tulemuste hindamine võib olla töömahukas
- olulised "erilised" andmete väärtused võivad jääda katsetamata, nt on vähe tõenäoline, et 0 tuleks juhusliku arvuna (selle vältimiseks tuleb rakendada kolmandat punkti ülalt)

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

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; on olemas 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: leitud vead, vigade arvu prognoos
- Suhe teistesse: koos teiste meetoditega
- Hinnang: kasutada erijuhtudel
- Vahendid: on tehtud juhuslike vigade generaatoreid, kuid need pole levinud

Lisatud vigu on programmeerijad praktikas kasutanud ka testijate töö kvaliteedi hindamiseks. Kuidas veel saaks testijate tööd hinnata? Kui on teada mõni konkreetne näide lisatud vigade kohta, siis kas need vead oleksid tulnud välja näiteks süstemaatilisel funktsionaalsel või programmipõhisel testimisel?

3.1.6. Mittefunktsionaalsete nõuete testimine: kasutuskõlblikkus

Nagu eespool märgitud, on mittefunktsionaalsete nõuete puhul enamasti raskem tagada nende testitavust. Lisaks on neid tihti ka raskem testida, isegi kui nõuded ise on testitavad. Toome mõned näited sellise testimise kohta.

Kui nõue on testitavalt sõnastatud, viitab ta ise sellele, kuidas testimist põhimõtteliselt läbi viia. Näiteks nõuet "Süsteemi töö võib kuu aja pikkuses eksploatatsioonis keskkonnas XYZ, kasutusaktiivsuse UVW ja kasutuslaadi NML korral olla häiritud maksimaalselt ühe tunni jooksul" saaks testida nii, et installime süsteemi keskkonda XYZ, kasutame seda kasutusaktiivsusega UVW ja kasutuslaadiga NML kuu aega ning mõõdame, kui kaua aega oli süsteemi töö häiritud. Selliseid teste tuleks teha korduvalt, et saada statistiliselt põhjendatud hinnanguid.

Kui tegu on kontorisyüsteemiga, võib seda vast isegi teha (oletades, et leiame ressursid, kasutajad, keskkonna jne), kui aga näiteks lennuki juhtimisega? Või kui ajavahemik on aastates? Mõnikord on sellised testid tehtavad pigem simulatsioonivahendite abil või rakendades staatilisi (süsteemi läbivaatuse jne) meetodeid, mida vaadatakse järgmises jaotises.

Mittefunktsionaalsete nõuete testimine võib nõuda mahukaid eksperimente. Vaatame näiteks kasutuskõlblikkuse testimist.

Kasutuskõlblikkuse testimine

Märgime kõigepealt, et mitmed kvaliteedifaktorid (eelkõige funktsionaalsus, tõhusus, töökindlus) mõjutavad samuti kasutuskõlblikkust. Tõepoolest, süsteemi millel puudub vajalik funktsionaalsus, mis kukub tihti kokku või mille reaktsiooniaeg on aeglane, võib vaevalt lugeda kasutajasõbralikuks.

Kasutuskõlblikkuse analüüs võib sisaldada mitmeid staatilise analüüsi komponente. Näiteks uuring, kas veebisaidi värvid on sobivalt valitud võib põhineda psühholoogiliste uuringute

tulemustel, mitte otsestel testidel. Kasutuskõlblikkuse kohta on tehtud mitmeid küsimustikke. Samas on küsimustikud tihti omavahel vasturääkivad. See on ka arusaadav – inimeste põhimõtted, kultuuriline taust, vajadused ja maitse on erinevad.

Kasutuskõlblikkus võib suurel määral sõltuda seadustest ja teistest regulatsioonidest – näiteks nõue, et avaliku sektori veebisait vastaks puuetega inimeste vajadustele, on fikseeritud mitme riigi seadusandluses.

Võimalik testimismeetod on süsteemi andmine katsetamiseks esinduslikule tulevaste kasutajate grupele ning selle grupi rahulolu mõõtmine. Tekivad kohe küsimused, milline on testimise meetodika, kuidas valida teste, kui suur peaks see grupp olema, kuidas valida grupi liikmeid, kui tihti tuleks testida jne.

Kasutuskõlblikkuse testimiseks tuleks valida realistlik olukord ja probleem, millede puhul kasutaja peab tegema süsteemiga kindlaid tegevusi, et probleemi lahendada. Vaatlejad jälgivad kasutaja tegevusi ja teevad märkmeid; tegevusi võidakse salvestada. Samuti kasutatakse kulutatud aja ja tegevuste mahu hinnanguid, silma liikumise jälgimist, kasutaja poolset oma tegevuste valjusti kommenteerimist, testi eel- ja järelküsimustikke jne.

Üks tuntud autor veebi kasutuskõlblikkuse teemal on Jakob Nielsen (<http://www.useit.com/>), kes soovib tihti teostatavaid testimisi väikese grupiga. Väide on, et enamik probleeme tuleb välja juba viie inimesega testimisel ("five users is enough"). Selle meetodi kriitikud väidavad, et suure ja mittehomoogeense kasutajaskonna puhul ei ole viieinimeseline rühm piisav.

3.1.7. Testimise maht ja lõpetamine

Testimise resultatiivsust pole kerge hinnata, erinevalt näiteks programmeerimisest - testimine ei anna alati selgelt 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
- testimise (või halvemal juhul süsteemiarenduse) 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
- pole enam huvitav, 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 siinjuures aja ja maksumuse planeerimisel. Testimiseks vajalike ressursside hinnang kõigub tegelikku kasutusse minevate programmide puhul vahemikus 20-80% programmi maksumusest. Mõned näited: hinnatakse väga ligikaudselt, et keskmise vastutusega süsteemide puhul võiks projekteerimise, programmeerimise ja testimise mahud olla sama suurusjärku. Samas vastutusrikaste reaalarjasüsteemide testimise maht võib olla tunduvalt suurem kui muude tegevuste maht kokku.

Vaadatud testimismeetodid annavad lisavõimalusi testimise mahu määramiseks, näiteks:

- kõik ekvivalentsiklassid (piirjuhud) peavad olema testitud
- testimine peab vastama haruadekvaatsuse kriteeriumile
- 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), süstemaatilist (võivad 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).

Paneme tähele, et selliste kriteeriumide puhul me vaid lükkame otsustuse testimise mahu kohta kõrgemale tasemele. Me ei vaatle üksikuid teste, vaid üldisemaid kriteeriumeid, mis omakorda määravad mahu. Otsustus mahu kohta tuleb siiski teha. Sellel tasemel on see nüüd otsustus, millist meetodit või kriteeriumit valida kasutamiseks või kui palju ressursse testimisele tuleb kulutada. Sellist otsustust on lihtsam teha, ta sõltub näiteks vähem testimise objekti mahust. Üheseid otsustuseeskirju niisuguseks otsustuseks ei ole, testimise korraldust käsitlevas peatükis anname mõned soovitusel.

Vaadeldud meetodite hinnaefektiivsuse järjestus (selles mõttes, et vea leidmise maksumus on väikseim; alates efektiivseimast) on tavaliselt järgmine: suitsutestimine, testimine kasutaja andmetega, riskipõhine testimine, uuriv testimine, ekspertteadmiste põhine testimine, piirjuhud, ekvivalentsiklassid, programmpõhine testimine, muud meetodid. Ühe meetodi hinnaefektiivsus ei tähenda, et teisi meetodeid ei tuleks kasutada. Meetod võib olla efektiivne ja leida esialgu kiiresti vigu, kuid kõiki vigu ei leia ükski meetod. Seega võib vastavalt programmi kriitilisuse astmele olla vaja kasutada ka kallimaid meetodeid.

3.1.8. Kontrollküsimusi ja ülesandeid

- Kontrollküsimusi.
- Mis on test, testimine, viga, hea test, edukas test, silumine, verifitseerimine, valideerimine, sertifitseerimine?
- Suitsutestimine, veaotsing, uuriv testimine, riskipõhine testimine

- Anda kokkuvõtte meetoditest: idee, eeltingimused, eelised, puudused, tulemused, suhteistesse, hinnang, vahendid
- Kuidas valida sisendeid? Kuidas hinnata väljundit? Millal testimine lõpetada? Kes on testijad? Missugune on testimise ja verifitseerimise vahekord?
- Funktsionaalne testimine, erinevus programmipõhisest testimisest, ekvivalentsiklasside analüüs, piirilukorrad, otsustustabelid, kasutusjuhud. Funktsionaalse testimise korraldus ja hinnang
- Mis on testimine programmi teksti põhjal? Mis on adekvaatsuskriteeriumid? Milline on viimaste võimsuse suhe, kuidas neid kasutada? Kas teadekvaatus garanteerib programmi korrektsuse? Anda hinnang programmi keerukusele. Kuidas toimub tsüklite testimine?
- Spetsifikatsiooni põhjal tuleks teatavat sisendandmete vahemikku töödelda ühte moodi. Realisatsioonis töödeldakse osa sellest vahemikust ühte moodi ja teist osa teisiti, kusjuures väljund on õige. Kas see on viga? Kuidas selline olukord võib tekkida? Kas see tekitab raskusi funktsionaalsel testimisel?
- Andmepõhine testimine, testimine juhuslike andmetega, lisatud vead
- Kuidas testida mittefunktsionaalseid nõudeid? Mis on siin erinev funktsionaalsete nõuete testimisest? Tooge näiteid.
- Kuidas testida kasutuskõlblikkust? Kas võib kasutuskõlblikkuse testimisena vaadelda ka ekspertide või kasutajate poolset hääletamist (näiteks kodulehekülgede konkursse)?
- Testimise maht ja lõpetamine. Kuidas hinnata testimise meetodi efektiivsust, millised meetodid on hinnaefektiivsemad, kas kõige hinnaefektiivsemad meetodid on piisavad ja miks?
- Ülesandeid.
- Millised on antud programmi puhul ekvivalentsiklassid, testitavad olukorrad, testid?
- Mitu testi on vaja antud programmi lause- (haru-, tee- jne) adekvaatseks testimiseks?
- 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?
- Antud programmi tekst, millistele järgnevatest küsimustest saab vastata: "Mitu testi on vaja antud programmi testimiseks?"; "Mitu testi on vaja antud programmi haruadekvaatseks testimiseks?"; "Millised on haruadekvaatsed testid?"?

3.2. Staatilised meetodid

Testimiseks (nagu eespool mainitud, kasutame testimise mõistet selle kitsamas mõttes) peab programm üldjuhul olema olemas. Staatiliste meetodite puhul programmi/süsteemi tavaliselt ei täideta, pigem analüüsitakse selle teksti, spetsifikatsiooni, dokumentatsiooni või muid objekte.

Staatiliste meetodite 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 ja on kõige kallimad parandada. 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 - analüüsivõime ja muud hooldatavuse kriteeriumid) on raske testimise teel hinnata, rohkem sobivad selleks staatilised meetodid.

Tänu neile eelistele hinnatakse mitmeid staatilisi meetodeid, näiteks läbivaatusi, testimisest hinnaefektiivsemateks (sama kulutatud aja või maksumuse kohta saadakse parem tulemus). Samas on selge, et staatilised meetodid testimist siiski ei asenda, isegi tõestamine ei garanteeri programmi töötamist. Kursuses on testimise meetodid eespool nende kohese rakendatavuse tõttu ainetöodes.

Staatilised meetodid võivad hõlmata väga mitmesuguseid teemasid. Kogu kvaliteedihaldus - 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 osalt ka programmide tõestamise ja infosüsteemi auditi. Oma eripära ja olulisuse tõttu on neid siiski vaadeldud edaspidistes jaotistes eraldi.

3.2.1. Küsimustikud

Eespool oli juttu veaotsingust - meetodist, mille puhul kasutatakse ekspertteadmisi mingilt alalt (programmeerimiskeskonnast, 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 või olla tähelepanu osutada. Küsimustikke on mitmesuguseid ja erineva mahuga, tavaliselt kümnetest kuni sadade küsimusteni.

Küsimustikud on kasulikud nii testide konstrueerimisel, vigade leidmisel kui ka selliste omaduste hindamisel, mida on raske testida (näiteks, hooldatavus või kasutajasõbralikkus). Ka standardeid võib kasutada kui laiaulatuslikke ja üldisi küsimustikke.

Näide: programmeerimise küsimustik

Allpool on toodud väiksema mahuga küsimustik programmide kohta. Mõni küsimus on üldisem, mõni kitsam, mõne keele puhul langeb osa küsimusi sootuks ära (näiteks, translaator kontrollib vastavat omadust). Küsimused on jaotatud rühmadesse.

Andmete kirjeldamine

- Kas muutujad on ilmutatult kirjeldatud (sõltub keelest, paljud translaatorid kontrollivad seda)?
- Kas vaikimisi atribuudid on antud õigesti?
- Kas initsialiseeritud on õigesti?
- 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 ühised andmestruktuurid on kirjeldatud alamprogrammides ühte moodi?

Arvutusvead

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

Võrdluste vead

- Kas võrreldakse sobimatuid asju, näiteks teksti ja numbrit?
- Kas sobivaid, aga eri tüüpi muutujaid võrreldakse õigesti?
- Kas spetsifikatsioon on võrdlustehetesse õigesti tõlgitud, näiteks 'suurim', 'mitte väiksem kui'?
- Kas võrdlustehete prioriteedid on antud õigesti?
- Kas võrdlustehete 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 ja atribuudid on korrektsed?
- 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 pöördumisel antakse üle konstante?
- Kas globaalsete muutujate atribuudid on igas moodulis samad?

Sisend ja väljund

- Kas faili atribuudid on õiged?
- Kas avamine/sulgemine on õigesti korraldatud?
- Kas kõik vajalikud failid on avatud/suletud?
- Kas pöördumine ühtib kirjeldustega?
- Kas sisend-/ väljundpuhvrte pikkused on õiged?
- Kas faili lõputingimus on antud korrektselt?
- Kas veaolukorrad/katkestused on õigesti 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

"Nähtamatud kasutajad" - Lisaks inimkasutajale suhtleb programm tüüpiliselt paljude "nähtamatute kasutajatega". Kas on uuritud, mis juhtub sellise suhtluse probleemide korral?

- Operatsioonisüsteem
- API-liides
- Failisüsteem

Lisafunktsioonid - Tavaliselt uuritakse, kas tarkvara teeb seda, mida vaja. Mõnikord (eriti turvalisuse testimisel) on rohkemgi olulised tarkvara kasutaja poolt mitte eeldatud, spetsifitseerimata ja dokumenteerimata omadused ning reaktsioonid - lisaks "teeb, mida vaja" on oluline ka "ei tee, mida ei ole vaja".

- Kas on testitud tarkvara spetsifitseerimata käitumist ja lisafunktsioone?

Standard kui küsimustik

Paljud standardid esitavad sisuliselt küsimustikke standardi ala kohta. Näiteks, kursuse teises osas vaadeldava standardi EVS-ISO/IEC 12207. "Infotehnoloogia – Tarkvara elutsükli protsessid" punktis 7.1.5 (tarkvara konstrueerimise protsess) öeldakse, et teostaja peab hindama tarkvara koodi ja testimistulemusi, arvestades alljärgnevaid kriteeriume. Hindamiste tulemused tuleb dokumenteerida.

- tarkvaraelemendi jälitatavus nõuete ja lahenduseni,
- väline kooskõla tarkvaraelemendile esitatud nõuete ja lahendusega,

- sisemine kooskõla üksusenoete vahel,
- üksuste kaetus testimisega,
- kasutatud kodeerimismeetodite ja -standardite sobivus,
- tarkvara integreerimise ja testimise teostatavus,
- käituse ja hoolduse teostatavus."

Seda punkti on kerge sõnastada küsimustikuna, näiteks nii:

"Kas teostaja on hinnanud tarkvara koodi ja testimistulemusi, arvestades alljärgnevaid kriteeriume? Kas hindamiste tulemused on dokumenteeritud? Kas on arvestatud:

- tarkvaraelemendi jälitatavust nõuete ja lahenduseni?
- välist kooskõla tarkvaraelemendile esitatud nõuete ja lahendusega?
- sisemist kooskõla üksusenoete vahel?
- üksuste kaetust testimisega?
- kasutatud kodeerimismeetodite ja -standardite sobivust?
- tarkvara integreerimise ja testimise teostatavust?
- eksploatatsiooni ja hoolduse teostatavust?"

3.2.2. Läbivaatused ja hindamised

Paljudest võimalikest vaatame töö analüüsi autori poolt, läbivaatust, programmeerija hindamist. Ka näiteks kaasüliõpilase töö analüüs kuulub siia alla.

Läbivaatus

Kvaliteedihalduses on levinud meetodiks mitut tüüpi ühised arutelud. Standard *ANSI/IEEE Std 1028-1988 IEEE Standard for Software Reviews and Audits* eristab juhtkonnapoolset hindamist (*management review*), tehnilist hindamist (*technical review*), tarkvara inspeksiooni (*software inspection*), läbivaatust (*walkthrough*) ja auditit (*audit*); viimast 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 on raske kuitahes heade meetoditega parandada.

Läbivaatusel on palju soodsaid külgi:

- viga saab leida varastel arenduse etappidel. Mida varasemal etapil viga on tehtud, seda suurema maksumusega on selle negatiivsed tagajärjed, kui viga jääb avastamata. Kuna vea leidmine läbivaatuse abil on suhteliselt odav, siis kokkuvõttes on läbivaatuse efektiivsus varastel etappidel suur. See on ka läbivaatuse oluline eelis testimise ees (miks?)
- 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

Kuidas võivad läbivaatused nurjuda?

- Pole arusaamist, mida on vaja läbi vaadata, läbivaatused korraldatakse arenduse lõpupoole (efekt on väiksem)
- Puuduvad ühised sihid, kvaliteedikriteeriumid, arusaamine läbivaatuse protsessist
- Läbivaatuse sisendid ja väljundid on kontrollimata
- Leitud vigade parandamist ei jälgita
- Leitakse vaid antud toote / dokumendi probleeme, ei püüta leida vigade algpõhjust (kogu protsessi vigu)
- Keskendutakse inimeste, mitte toote probleemidele
- Keskendutakse vaid tootele, ignoreeritakse inimeste probleeme

Eeltingimused:

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

Osavõtjad ja nende rollid:

- esitaja (läbivaatusel üldjuhul 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

Soovitusi läbivaatuseks:

- analüüsi toodet, mitte autorit
- koosta kava ja jälgi seda

- luba vaidlusi, kuid piira neid vajadusel
- ära püüa kõiki probleeme lahendada
- tee kirjalikke märkmeid
- piiratud osavõtjate arv, ettevalmistused
- valmista ette küsimustik iga analüüsitava toote jaoks
- varu ressursse, sh aega
- koolita osavõtjaid
- vaata läbi ka (varasemaid) läbivaatusi

Igat liiki arutelusid, eriti aga läbivaatuse tüüpi demokraatlikke kogunemisi, võivad häirida osavõtjatevahelised teadvustatud või teadvustamata mängud. Mängu mõistetakse siinkohal tegevuste jadana, millel on kaks eesmärki: varjatud ja avalik. Näiteks korduvalt kasutatud sõnaühend "Jah, aga ..." viitab mängule. Mängud võivad olla arutelu eesmärgi saavutamist soodustavad või häirivad. Mäng ei pruugi olla lõbus tegevus. Kuna mängud rikuvad normaalset arutelu, siis on oluline:

- teada, et mängud on olemas
- osata aru saada, et mäng käib
- osata see vajadusel 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 ka lobisemist)
- jälgida ajalimiiti
- jälgida vastutust (nt allkirjastamine)
- jälgida formaalseid nõudeid
- veenduda, et teatakse standardeid
- vältida võimalusel juhtkonna esindajate viibimist arutelul

Autoripoolne analüüs

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

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 või projekti, ütlemata kumb on halvem ja kumb parem. Igaüks hindab teiste pakutud kahte toodet. Tagatud peab olema hinnatavate anonüümsus. Hinnata tuleks vastavalt vajadusele toodete arusaadavust (ka projekti puhul), vastavust projektile või spetsifikatsioonile, muudetavust, muid tegureid. 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 tunnetatud vajadus. Tarvis on initsiaatoreid, kes hindamisega tegelevad. Tulemusena saab iga osaleja hinnangu oma kvalifikatsioonile ja soovitusi enese arendamiseks.

3.2.3. Kontrollküsimusi ja ülesandeid

- Anda kokkuvõtte meetoditest: idee, eeltingimused, eelised, puudused, tulemused, suhe teistesse, hinnang, vahendid
- Küsimustikkude liigitusi, programmeerimisele orienteeritud küsimustiku struktuur, standard kui küsimustik
- Millised arutelude ja hindamiste liigid on kasutusel?
- Läbivaatus, selle plussid, miinused, eeltingimused, osavõtjad, korraldus, mängud, suhe juhtkonda
- Töö analüüs autori poolt, programmeerija hindamine

3.3. Töökindluse oluline suurendamine ja selle rakendusi

3.3.1. Töökindluse parendamise vahendeid - ülevaade

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 ja kontrollida - 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. Mõningaid neist vaadatakse allpool: N-versiooniline programmeerimine (N-version programming); veapuu analüüs; formaalsed meetodid, sealhulgas programmide tõestamine; teatud kvaliteediomadustele, nt töökindlusele või turvalisusele orienteeritud arendusmeetodid (sh Cleanroom development, Common Criteria).

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

3.3.2. N-versiooniline programmeerimine (N-version programming)

N-versioonilise programmeerimise (N-version programming) idee on, et paralleelselt arendatakse ja kasutatakse mitut programmi versiooni. Kasutamisel võrreldakse tulemusi, enam levinud vastused loetakse õigeaks (hääletamine).

Matemaatilisel võib tõestada, et sellisel meetodil – rakendades paralleelseid seadmeid ning kasutades hääletamist, vajadusel hierarhiliselt - saab teatud tingimustel viia riistvara töökindluse kuitahes suure soovitava väärtuseni.

Meetod, millelt palju loodeti ja mis õigustab ennast hästi riistvara puhul, on kasutatav, kuid ei anna sama häid tulemusi tarkvara korral. Üks põhjus on selles, et inimlik loogika jälgib tihti samu radu ja paralleelsetes arendustes tehakse ühesuguseid vigu.

3.3.3. Veapuu analüüs

Veapuu analüüsi puhul ehitatakse ja/või veapuu. Alustatakse suurest veast, mida tahetakse vältida, vaadatakse selle vea eeltingimusi, eeltingimuste eeltingimusi ja nii edasi. Kui iga puu lehega seostada eeltingimuse tõenäosuse hinnang, saab lehtedest puu tipu suunas liikudes kätte analüüsitava suure vea tõenäosuse.

Meetod toimib hästi tehniliste süsteemide puhul. Süsteemi või programmi veapuu analüüs võimaldab lokaliseerida ja analüüsida süsteemi kriitilisi komponente, analüüsida kriitiliste komponentide vahelisi seoseid ning paremini kvantifitseerida riske. NB! Võib juhtuda, et süsteemi normaalse funktsionaalsuse mittetoimimine ei ole "suur viga, mida tahetakse vältida" (millal?)

3.3.4. Formaalne verifitseerimine/tõestamine

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) tõestamine 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 tõestamist toetav tarkvara.

Algoritmide või programmide tõestamist võib käsitleda ühe staatilise meetodina. Lühikokkuvõtte tõestamisest:

- Siht. Näidata, et programm vastab spetsifikatsioonile
- Idee. Luuakse loogiline arvutus (valemid, aksioomid, tuletusreeglid, tõestused, teoreemid). 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, soovitatavalt toetav tarkvara
- Eelised. Suurem töökindlus, formaalne korrektsus, ainuke viis tsükli 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 vajadusel. Vähekriitiliste süsteemide puhul pole otstarbekas
- Vahendid. On tehtud tõestamist toetavat tarkvara, kuid see pole levinud

Töömahu vähendamiseks võib jaotada süsteemi tuumaks, mida tõestatakse, ja ümbruseks, mida ei tõestata. Tuum kindlustab, et väga halvad asjad ei toimu, aga ei pruugi kindlustada, et "head" asjad toimuvad (vrd veapuu analüüs). 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 teostamise tase on madal, pole ühe etapi tulemuste tõestamisest palju kasu. Illustreerime seda järgneva tabeliga.

Tõestatav tase	Vahendid	Märkusi	Raskusaste
Spetsifikatsioon	Läbivaatus	Ei ole formaalset eellast, seega pole millegi suhtes tõestada	Spetsifikatsiooni korrektsus on peamisi probleeme
Realisatsiooni-projekt	Projekti tõestus	Tõestust saab teha vaid siis, kui spetsifikatsioon oli formaalne. Erijuhused - tõestatakse mõne aspekti suhtes (nt vasturääkivus, liiasus jne)	Sõltub formalismist ja tõestamise laadist
Programmi kood	Programmi tõestamine	Väga mahukas	Raske
Tõestus-meetodid	Loogika	Leidub häid meetodeid ja formalisme. Ei sõltu üksikutest programmidest	Palju uuritud. Ühekordne töö
Objektkood	Programmi või kompilaatori tõestamine		Raske. Ühekordne töö
Täitmine	Mikroprogrammi/riistvara tõestamine		Ühekordne töö

Tõestamise teema lõpetamiseks veel tsitaat klassikult. D. E. Knuth: "Beware of bugs in the above code; I have only proved it correct, not tried it." (<http://sunburn.stanford.edu/~knuth/faq.html>).

3.3.5. Tarkvaraarendus puhtas/kontrollitud keskkonnas

Üks komplekssetest meetodikatest, mis rakendab eelpooltoodud meetodeid, on tarkvaraarendus puhtas/kontrollitud keskkonnas (Cleanroom software engineering). Anname selle lühikese ülevaate.

Eesmärk: kõrge kvaliteediga kontrollitud töökindlusega tarkvara arendus (sõna cleanroom tuleb elektroonikatööstusest, kus kasutatakse füüsiliselt väga puhast keskkonda, et vältida vigaseid detaile). Rõhk on vigade vältimisel ja sertifitseeritud töökindlusel.

Allikad: Harlan Mills (1987), edasi arendatud 1990-tel, praegu tööstuslikult kasutusel.

Meetodid: lisaks tavaliselt kasutatavatele arendusmeetoditele rakendatakse (1) mitmeid formaalseid meetodeid arenduses ja testimisel ning (2) kasutusprofiili spetsifitseerimist.

- Püütakse tagada korreksust arenduse algfaasides (nt. tõestada, et disain on korrektne)
- Rakendatakse versioonikaupa arendust (incremental development), kus iga versiooni tulemust võrreldakse etteantud kriteeriumidega ning tehakse uus iteratsioon eelmiste testide kordamisega, kui tulemus ei rahulda
- Rakendatakse matemaatilisi meetodeid. Vaadatakse koodi kui funktsiooni, mis tuleb defineerida spetsifikatsioonis ja verifitseeritakse, et disain realiseerib selle funktsiooni
- Programmi vastavust disainile kontrollitakse läbivaatustega
- Testimine statistiliselt esinduslike juhuslike andmetega, põhineb kasutusprofiili spetsifikatsioonil ning statistilisel analüüsil

Eelised: Tunduv töökindluse, korreksuse ja arusaadavuse paranemine. Hinnangud töökindluse ning vigade arvu kohta.

Kasutajad (näited): IBM, Ericsson, õhujõud, maismaa väed.

Suhe teiste meetoditega: objekt-orienteeritud meetodid, CMM.

Vahendid: Toetavad süsteemid ja keskkonnad on olemas.

3.3.6. Common Criteria

Näitena selle kohta, millistel juhtudel rakendatakse tarkvara arenduse ja kontrolli formaalseid meetodeid, toome standardi ISO/IEC 15408. Evaluation Criteria for IT security (kasutatakse nimetusi ja lühendeid "*Common Criteria*", "*The Common Criteria for Information Technology Security Evaluation*", CCITSE, CC, "Ühiskriteeriumid"). Standardi pakutav raamistik võimaldab tarkvara kasutajatel spetsifitseerida tarkvara turvanõudeid, arendajatel neid nõudeid realiseerida ning sõltumatutel testimislaboritel nõudeid hinnata.

Standardis esitatakse seitse turvataset:

1. Funktsionaalsuse ja liideste spetsifikatsioon + funktsionaalselt testitud (sõltumatu osapoole vastavustestimine)
2. Struktuurselt testitud (lisaks eelmisele nõutud nt arendajate poolne kaastöö koodipõhisel testimisel ja vigade otsingul)
3. Meetoodiliselt testitud ja kontrollitud (lisaks eelmistele tuleb nt rahuldada spetsifikatsiooni adekvaatsuse/katvuse kriteeriumid ning testida disaini)
4. Meetoodiliselt disainitud, testitud ja läbi vaadatud (lisaks eelmistele mitmesugused analüüsid ja läbivaatused)

5. Poolformaalselt disainitud ja testitud (lisaks eelmistele tuleb osaliselt rakendada formaalseid disaini meetodeid)
6. Poolformaalselt verifitseeritud disain ja testitud (lisaks eelmistele nõutakse osalises või täismahus formaalsete disaini ja testimise meetodite kasutamist)
7. Formaalselt verifitseeritud disain ja testitud (lisaks eelmistele nõutakse täiendavate formaalsete disaini ja verifitseerimise meetodite kasutamist)

Neist tasemetest on esimene kõige nõrgem ja seitsmes kõige tugevam. Nõudmised formaalsete arenduse meetodite olemasolule esitatakse alates viiendast turvaklassist. Esimese nelja klassi kohta on ühtlustatud arusaamine ja tunnustamise kokkulepped mitmete maade vahel. Viiendast seitsmenda klassini ei ole praegu üldist kokkulepet turvaklasside sertifitseerimise tunnustamise kohta.

Standard on allalaaditav ISO veebist (<http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>). Infot sertifitseerimise, sealhulgas sertifitseeritud toodete kohta on aadressil <http://www.commoncriteriaportal.org/>.

3.3.7. Võrdlus riistvaraga

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

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

3.3.8. Kontrollküsimusi ja ülesandeid

- Saavutatav töökindluse tase seniste meetoditega, selle olulise suurendamise võimalusi
- Teha kokkuvõtte meetoditest: idee, eeltingimused, eelised, puudused, tulemused, suhe teistesse, hinnang, vahendid
- Arenduse dubleerimine
- Veapuu analüüs
- Programmide tõestamine, idee, ajalugu, probleeme, võimalusi jne. Tõestamise tasemed
- Cleanroom development: suhe formaalsete meetoditega, testimise korraldus, ülevaade
- Common Criteria: suhe formaalsete meetoditega, tasemete võrdlus, ülevaade
- Võrdlus: töökindluse saavutamine riist- ja tarkvara puhul

3.4. **Kontrolli korraldus**

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

Sellesse jaotisse võiks põhimõtteliselt lülitada ka eespool formaalsete meetodite rakendusnäitena toodud tarkvaraarenduse puhtas/kontrollitud keskkonnas.

3.4.1. Lihtsamaid skeeme, vajadus keerukamaks korralduseks

Kontrolli korraldus nagu ka kasutatavad meetodid sõltuvad nõuetest 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 on küll erinevad (seepärast 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

Ülaltoodud tegevusi võib korrata üks või rohkem kordi.

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

- arendus ja 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 vastata, teadmata olukorda. Tuleb valida antud olukorras sobivaim korraldus.

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

- tarkvara on süsteemi sisse ehitatud, süsteemi testimine on kallid - tekib vajadus lahutada tarkvara ja süsteemi test
- keerukas toode - 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 - nõuab erinevate kasutajarühmade koostööd
- mitmeetapiline mahukas testimine, mis vaheldub arendustegevustega - 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. Allpool vaadeldakse keerukamaid skeeme.

3.4.2. Kontrolli meetodite efektiivsus ja valik

Eriti lihtsamate korralduse skeemide puhul on otstarbekas jälgida kontrolli meetodite hinnaefektiivsust (kui palju maksab ühe vea leidmine ja parandamine erinevate meetoditega?) ja alustada efektiivsematega. On leitud, et suureneva maksumuse järjekorras võib meetodid reastada järgmiselt:

- arendaja poolne esmane testimine (saab teha kiirelt arenduse käigus, on mitmete arendusmetoodikate komponent)
- suitsutestimine (väga kiire esmane testimine)
- riskipõhine testimine (katsetab kõige kriitilisemaid omadusi)
- uuriv testimine
- testimine kasutaja andmetega (peavad kindlasti töötama), selleks võib kasutada ka normaalse töö ekvivalentsiklasside teste
- läbivaatused (avastavad vigu vara)
- vea otsing (kui testija on ekspert)
- piirjuhud (vigade kuhjumise kohad)
- ekvivalentsiklassid (sealhulgas veaolukorrad)
- programmi põhjal testimine
- lisatud vead
- tõestamine

Järgnevas tabelis on soovitusel kontrolli meetodi valikuks lähtudes rakenduse töökindluse ja kriitilisuse nõuetest. Üldine loogika põhineb meetodite efektiivsuse ja maksumuse hinnangul, eriti nõrgemate nõudmiste puhul alustatakse kõige odavamatest ja efektiivsematest meetoditest. Tabeli kasutamisel tuleb silmas pidada, et erinevatel alamsüsteemidel võivad olla erinevad kriitilisuse nõuded.

Nõutav töökindluse tase	Programmi vigade võimalik mõju	Soovitavad meetodid
Väga madal	Pole märgatavaid kahjulikke tagajärgi	Arendaja testimine, suitsutestimine, testimine kasutaja andmetega
Madal	Suhteliselt väike rahaline kahju või ajakadu	+ läbivaatus, vea otsing, uuriv testimine, piirjuhud
Keskmine	Märgatav rahaline kahju, ebamugavused	+ Üldised kvaliteedihalduse ja töökorralduse meetodid, (protsessi)standardite kasutamine, küsimustikud, funktsionaalne testimine,

		haruadekvaatus, andmepõhine testimine
Kõrge	Suured rahalised kahjud, firma võimalik häving, vigastused, keskkonnareostus	+ muud programmipõhised meetodid, juhuslikud testid, testimise automatiseerimine, meetrikad
Väga kõrge	Inimelud, paljusid mõjutav finantskahju, suur keskkonnareostus	+ tõestamine, dubleerimine, muud meetodid

3.4.3. 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. Lihtsustatult võib öelda, et 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 realiseerimisprojektile vastab integratsioonitestimine
- tarkvara kodeerimisele vastab mooduli testimine

Tegemist on nelja erineva testimisetapiga, mis vastavad neljale süsteemiarenduse olulisele etapile. Igal arendusetapil luuakse oma testiprojekt (dokumenteeritud või dokumenteerimata). Iga testimisetapi sisend on tarkvara konfiguratsioon (sh vajadusel dokumentatsioon) ja testimise konfiguratsioon (sh vajadusel testimise dokumentatsioon). Testimise väljundiks on leitud vead ja töökindluse prognoos. Kui leiti suuri probleeme, võib igalt testimisetapilt tagasi minna vastavale arendusetapile - mudel on iteratiivne ja hästi tagasisidestatud.

Võimalik raamistik selliseks arenduseks on antud standardis IEEE Std 1012 IEEE Standard for Software Verification and Validation Plans. Kui IEEE 829 käsitleb põhiliselt testide dokumenteerimist, siis IEEE 1012 annab rohkem juhiseid kontrolli korralduseks ja selle integreerimiseks arendusprotsessi.

V-mudeli aluseks on lihtne idee, millel põhinevad mitmed teised meetodid. Näiteks kasutatakse seda enamikus kaasaegsetes elutsükli iteratiivsetes mudelites. Seda on ka laiendatud, näiteks tuues sisse kasutusprofiili ("Dotted U-mudel").

Vaatame testimise etappe lähemalt, alustades madalama taseme testidest.

Mooduli testimine

Mooduli tasemel testimiseks on kasutatavad programmipõhised meetodid, tõestamine, mitmed testimise automatiseerimise vahendid jne. Seejuures on kasulik jälgida testimise hinnaefektiivsust (vt. ülal).

Integratsioonitestimine

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

- “pane kõik kokku ja testi” - võib töötada lihtsate süsteemide korral; keerukate puhul on vigu raske lokaliseerida
- alt üles integreerimistestid – kõigepealt luuakse alumise taseme moodulid, nendest lähtudes liigutakse ülataseme moodulite suunas
- ülalt alla integreerimistestid – kõigepealt luuakse ülataseme moodulid, nendest lähtudes liigutakse alumise taseme moodulite suunas
- segameetod ("võileib") – eelmise kahe kombinatsioon

Puuduvate komponentide asendamiseks kasutatakse sellisel testimisel komponendi reaalsel käitumist imiteerivaid objekte (mock objects), milles on realiseeritud vaid teatud komponendi omadused – näiteks kasutajaliides.

Alt üles integreerimise puhul asendatakse osa ülemise taseme väljakutsuvaid moduleid 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 koostestitavate moodulite kogumid (klastrid)
- kirjuta klastrite jaoks draiverid (draiveri tegevuste näited: kutsub vaid mooduli välja; saadab parameetri; prindib parameetri; saadab ja prindib)
- testi
- asenda draiverid tegelike moodulitega

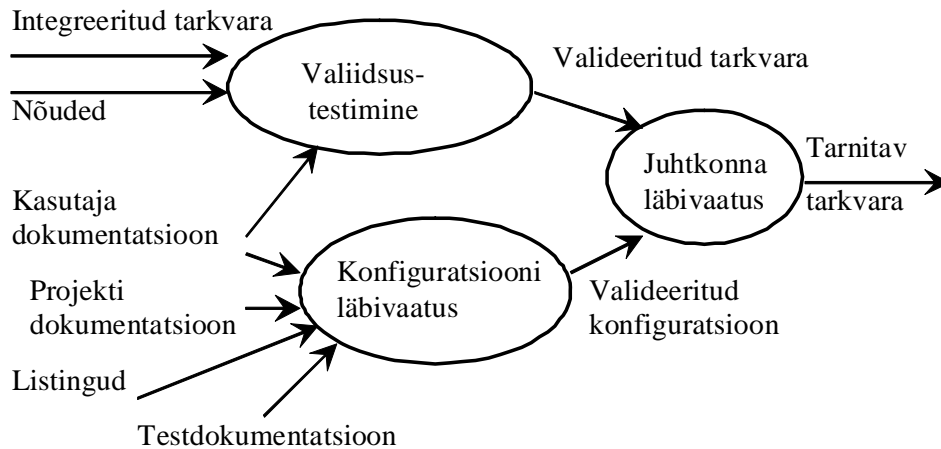
Ülalt alla integreerimistestide puhul kirjutatakse väljakutsutavate moodulite asemel nn lühised (asendavad programmid; näited: prindi mooduli nimi; prindi üleantav parameeter; tagasta vastusparameeter; otsi tabelist vastus ja tagasta väljund vastavalt sisendile). Testimise käik: ülataseme moodul - lühised - test – lühiste asendamine - regressioonitestimine. Eelis on see, et mingi kasutaja jaoks nähtav esialgne süsteem on juba algusest saadik olemas, side kasutajaga on parem. Üldpilt süsteemist on kasulik ka arendusjuhtidele, kellel on kogu aeg ettekujutus arenduse käigust.

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.

Valiidsustestimine

Eelkirjeldatud testimise etapid lähtuvad tihti projektist (nt tarkvara spetsifikatsioonist, disainist) ja püüavad vastata küsimusele "Kas ehitasime tarkvara õigesti?" (verifitseerimistest, verifitseerimine). Projekt võib aga olla vigane. Valiidsustestimine vastab küsimusele: "Kas ehitasime õige tarkvara?" - tarkvara katsetatakse selle vastavuse suhtes tarkvara lähtenõuetele.

Valiidsustestimise rolli arenduses võib kujutada järgmiselt.



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

Süsteemi testimine

Üldjuhul kuulub tarkvara süsteemi koosseisu. Spetsifitseeritud nõuded tarkvarale võivad olla vigased. Seega ei anna ka tarkvara valiidsustestid õigsuse garantiid, vaid testida tuleb süsteemi tervikuna lähtudes süsteemi nõuetest. Süsteemi nõuded spetsifitseeritakse enamasti juba süsteemi, mitte tarkvara mõistetes (nt sõiduki töökindlus teatud kiiruste puhul).

Süsteemitestide hulka kuuluvad peale tavalise töörežiimi testide tihti ka jõudlus-, taluvus-, taastumis-, turbe- ja muud testid.

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.

Taastumistestid 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 märgime, et süsteemi on raske (kuigi mitte võimatu) testida, kui pole spetsifikatsiooni. On ka eriolukordi, mida on kulukas või võimatu testida. Nii võib olla väga kulukas ja ohtlik reaalselt testida laeva avarisüsteeme. Sellisel puhul saab rakendada näiteks läbivaatusi, tõestamist, töökindluse mudeleid, simuleerimist või 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.

3.4.4. RUP testimise filosoofia

RUP testimise filosoofia (Testing: The RUP Philosophy. Paul Szymkowiak, Philippe Kruchten. Rational Software) võib kokku võtta järgmiselt:

- Iteratiivne arendus. Missioon igal iteratsioonil
- Esialgne dokumenteerimise aste on madal. Detailne testimise plaan igal iteratsioonil, Master Plan on suhteliselt lakooniline
- Terviklik lähenemine. Testid põhinevad spetsifikatsioonil ja muudel allikatel
- Automatiseerimine. Testandmete genereerimiseks, testide läbimiseks ja tulemuste analüüsiks

Iteratsiooni missioon - näiteid

- Find as many defects as possible.
- Find important problems fast.
- Assess perceived quality risks.
- Advise about perceived project risks.
- Advise about perceived quality.
- Certify to a given standard.
- Assess conformance to a specification (requirements, design, or product claims).

Testimise artefaktid:

- Test Evaluation Summary
- Test Plan (and sometimes a Master Test Plan)
- Test Ideas, and Test-Idea List
- Test Suites and Test Cases
- Defect and Defect List
- Workload Model

Neli testimisega seotud rolli (mitte tingimata töökoha nimetused):

- Test Manager
- Test Analyst
- Test Designer
- Tester

3.4.5. XP: Näide arendusprotsessi integreeritud kontrollist

XP (Extreme Programming) üheks eesmärgiks on suurem paindlikkus muutuvate nõuete tingimustes. XP põhiideed pakkus välja Kent Beck 1996.a. XP ja testimise vahekorda võib lühidalt iseloomustada järgmiselt (<http://www.extremeprogramming.org>, www.xprogramming.com, <http://c2.com/cgi/wiki?CodeUnitTestFirst>):

- Põhimõtted: Tellija-orienteeritus, tiimitöö, V-mudel (testimine paaris arendusega), prototüüpimine, lihtsuse püüd
- Testipõhine arendus (test driven development): testid luuakse enne realiseerimist kliendi lugude (stories) põhjal: ühiku testid (programmeerijalt, kohe enne realiseerimist) ja vastuvõtmise testid (Tellijalt, funktsionaalsed)
- Läbivaatuste asemel koostöö

XP & testimine, näide - uue funktsionaalsuse lisamine:

- Find out what you have to do.
- Write a unit test for the desired new capability. Pick the smallest increment of new capability you can think of.
- Run the unit test. If it succeeds, you're done. Go to step 1, or if you are completely finished, go home.
- Fix the immediate problem: maybe it's the fact that you didn't write the new method yet. Maybe the method doesn't quite work. Fix whatever it is. Go to step 3.

3.4.6. Testimisprotsessi standardid ja täiustamine (TPI meetodika)

Eelpooltoodud meetodikad (sealhulgas ka eelmises peatükis kirjeldatud tarkvaraarendus puhtas/kontrollitud keskkonnas) on kasulikud ja soovitatavad kasutamiseks esimeste põhjalikumate korralduse raamistikena. Kogu testimisprotsessi haldamist tervikuna ja selle parendamist käsitlevad mitmed kursuse teises osas käsitletavat meetodikad ja standardid, näiteks CMMI ja EVS-ISO/IEC 12207. Infotehnoloogia – Tarkvara elutsükli protsessid. Viimased annavad ka soovitusi testimise haldamiseks kõigis tarkvara elutsükli protsessides.

Spetsiifiliselt testimisele orienteeritud lähenemise näitena toome testprotsessi täiustamise (Test Process Improvement, TPI) mudeli ja meetodika, 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, testide spetsifitseerimise meetodid, testimise vahendid, meetrikad ja nii edasi. Et jälgida organisatsiooni küpsuse taset, on iga võtmealaga seostatud selle tasemed ja kontrollküsimused. Mudel sisaldab ka soovitusi ja instruksioone teatava tasemeni jõudmiseks. Lisaks nendele põhikomponentidele sisaldab mudel metodoloogia tarkvaratoodete struktuurseks testimiseks (TMap) ja testimise küpsuse maatriksi (Test Maturity Matrix). Viimane aitab näidata seoseid ja taseme erinevusi erinevate võtmealade vahel.

Ettevõtte, 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 (või osa protsessist). Näiteks esitab rahvusvaheline standard EVS-ISO/IEC 12207 kokku nelikümmend kolm süsteemikonteksti ja tarkvaraspetsiifilist protsessi. Kindlasti saab TPI mudelit kasutada väärtusliku täiendusena ja täpsustusena sellele ja teistele laiaulatuslikele standarditele.

On kättesaadavad nende meetodikate järgmised versioonid (TPI®NEXT AND TMAP®NEXT).

3.4.7. Tarkvara kontrolli automatiseerimine

Tarkvara kontrolli tegevused, näiteks testide läbiviimine, aga vähemal määral ka testide projekteerimine, sisaldavad sageli palju rutiinset käsitööd. Eriti suur on sellise käsitöö osa regressioonitestimisel (korduvtestimisel). Sellise testimise automatiseerimine näib lihtne ja seda on ahvatlev teha. Testimise automatiseerimise vahendid on mitmete eelpool vaadeldud kaasaegsete arendusmetoodikate (XP, testipõhine arendus, Cleanroom development jne) hädavajalik komponent. Testimise automatiseerimise vahendi näide on JUnit (<http://www.junit.org/index.htm>) koos arvukate xUnit modifikatsioonidega.

Kontrolli automatiseerimiseks on mitmeid võimalusi ja tarkvaravahendeid:

- testide genereerimise automatiseerimine: näiteks, on vahendeid, mis genereerivad automaatselt lause- või haruadekvaatsed testid, samuti vahendeid mis genereerivad teste mitmesuguste funktsionaalsete kirjelduste põhjal
- GUI testidraiverid ning testide salvestamise ja korduvtestimise vahendid lubavad teste salvestada ning uuesti korrata peale tarkvara muudatusi
- testide skriptide salvestamine ja korduvtestimine: tekitades, automaatselt genereerides või täitmise ajal salvestades testimise skripte ning neid uuesti täites
- koormustestimise vahendid võimaldavad tekitada suure töökoormuse serverite ja/või veebirakenduste testimiseks
- mitmesugused vahendid testimise lihtsustamiseks: näiteks tarkvara, mis lihtsustab draiverite või lühiste loomist integratsioonitestimisel, juhuslike vigade generaatorid ja nii edasi
- vahendid testimise kvaliteedi hindamiseks, näiteks kontrollimaks, millised programmi komponendid on testidega läbimata
- staatilise analüüsi vahendid, näiteks programmi meetrikate hindamise tarkvara
- vigade ja paranduste haldamise abivahendid
- tarkvara testimisprotsessi kavandamiseks, testimise ressursside hindamiseks jne
- veebisaitide testimise vahendid jne

Mõnikord reklaamitakse testimise automatiseerimise vahendeid kui kiiresti tasuvaid ja lihtsalt rakendatavaid. Tegelikkus pole siiski nii lihtne. Testimise (laiemalt, tarkvara kontrolli) automatiseerimine nõuab mitmesuguseid ressursse:

- vahendite hange või arendus (mõnikord üsna kulukas)
- vahendite alane koolitus (vahendid võivad olla keerukad)
- võib olla vajalik arendustegevuse ümberkorraldus, näiteks arendajate ja testijate suhtlemise osas (sageli kaugemas perspektiivis kasulik tegevus)
- testide kogumite loomine iga testitava rakenduse jaoks, mis nõuab testide dokumenteerimist (kui korduvtestimist ei tehta, pole sellel mõtet)
- testide täitmine
- testide kogumi uuendamine, kui testitav toode muutub (võib olla üsna töömahukas)

- testide ülekanne, kui muutub arendusplatvorm
- muud ressursid, näiteks testide jagamiseks eri arendusgruppide vahel

Soovitusi testimise automatiseerimiseks (Bach, 1999):

- tehke vahet testimisprotsessi ja selle automatiseerimise (vahendite) vahel, ärge neid samastage - testimine peab jääma läbipaistvaks ka automatiseerimise korral
- vaadake automatiseeritud testimist kui täiendust kogu testimisprotsessile - mitte selle asendust
- valige vahendid hoolikalt, tutvuge nendega enne, koguge infot
- valige hoolega testide kogumi struktuuri, et see oleks arusaadav ja toetaks testimisprotsessi
- kindlustage, et iga automatiseeritud testimise seanss annaks tulemuses raporti, mis kirjeldab täidetud teste ja leitud vigu - see aitab analüüsida testimisvahendite otstarbekust
- veenduge, et testitav toode on piisavalt väljaarendatud - siis on lootust, et testide automatiseerimisest saadavad tulud ületavad testide muutmisele tehtud kulutusi

Eelnevast tulenevad ka testimise vahendite eelised ja puudused. Eelisteks on, et need vahendid võimaldavad kokku hoida hulga käsitööd. Puudused - palju lisatööd, eriti muutuva tarkvara või selle keskkonna puhul.

Automatiseeritud vahendite näitena on praktikumides vaadatud mitmeid süsteeme (vt kursuse veebist jaak.tepandi.ee).

Lisaks vaatame lähemalt veebisaitide testimist. Üks Fortune 100 hulka kuuluvate firmade veebisaitide audit näitas, et nende saitide ligi kolmsada tuhat HTML lehekülge sisaldasid kokku 84,302 puuduvat linki - üks puuduv link kolme-nelja lehekülje kohta. Ainult seitsme ettevõtte saitidel vaadeldud grupist ei olnud ühtegi puuduvat linki. Samad leheküljed sisaldasid üle kolme ja poole miljoni HTML kodeerimisvea - igal leheküljel seega üle tosina vea, kusjuures päris ilma kodeerimisvigadeta ei olnud ühegi firma veebisait.

Veebisaidi kvaliteediomadusi on käsitletud palju, seejuures varieeruvad nii vaadeldavate saitide tüübid, käsitletavat kvaliteediatribuudid kui ka testimise meetodid. Üsna palju on uuritud saidi kasutusomadusi, näiteks disaini mõju kasutaja tegevusele. Vähem on käsitletud saitide testimise probleeme.

Testi planeerides valivad arendajad lähtuvalt ülesande omadustest vajalikud kvaliteedikriteeriumid. Veebisaitide puhul tavaliselt testitavad omadused saab hästi paigutada ISO/IEC 9126 kvaliteediatribuutide skeemi. Lihtsamate saitide puhul testitakse tüüpiliselt funktsionaalsust, töökindlust, efektiivsust ja kasutatavust, sealhulgas järgmisi omadusi:

- kasutatavus, eriti kasutusmugavus
- funktsionaalsus, eriti vastavus, turvalisus ja täpsus (näiteks, failide avastamine, mida pole muudetud teatud arv päevi)
- töökindlus, eriti valmidus - HTML kodeerimisvead, puuduvasse faili või URLi viitavate linkide leidmine, jne.
- efektiivsus - lihtsad jõudlustestid, nagu näiteks saitide laadimise aeg; mittekasutatavate failide või failiruumi leidmine

- efektiivsus - kompleksed laadimis- ja koormustestid, nagu näiteks testimine teatud arvu samaaegsete kasutajatega
- koostalitlusvõime – näiteks töötamine erinevate brauseritega

Veebisaitide või veebiliideste testimiseks on üle saja eri vahendi, sealhulgas ka prii- ja jaosvara. Vahend võib olla iseseisev või integreeritud arenduskeskkonda. Funktsionaalsuselt võib testimise tarkvara klassifitseerida järgmiselt:

- vastavuse kontrolli vahend - toetab selliseid tegevusi nagu süntaksikontroll, puuduvate linkide otsing, laadimistestid jne
- üldotstarbeline testimisvahend - võimaldab teha lisaks eelmistele ka funktsionaalseid, koormus- ja muid teste
- turvalisuse testimisvahendid
- eriotstarbelised, mitmesuguseid funktsioone realiseerivad vahendid, mis võimaldavad veebi omadusi parandada või testivad eritüüpi omadusi (nt puuetega inimeste juurdepääs)

Nii iseseisvad kui ka sisseehitatud testimisvahendid võivad anda häid tulemusi. Sisseehitatud vahend on käepärasem, kui sait on ka loodud sama vahendiga. Võib siiski osutada, et redaktorisse sisseehitatud testimisvahendid testivad vaid selle redaktoriga loodud saite.

Vahendid, mis leiavad ainult HTML standardile mittevastavusi, võivad näidata arvukaid vigu, mis praktiliselt tegelikku tööd ei häiri. Eriotstarbelised vahendid on kasulikud vastavate erinõudmiste puhul.

Järgnevas tabelis 1 on toodud mõnede testimisvahendite parameetrid (vt ka näiteks <http://www.softwareqatest.com/qatweb1.html>, <http://www.webaim.org/articles/freetools/> jt).

Testimis- vahend	Tüüp	WWW-aadress	Mida teeb
W3C HTML Validation Service	Vastavus	http://validator.w3.org/	Kontrollib vastavust W3C HTML ja XHTML jt kodeerimis-standarditele. Saab kontrollida etteantud faili selle aadressi järgi või üles laaditud faili.
W3C Link Checker	Puuduvate linkide kontroll	http://validator.w3.org/checklink	Kontrollib puuduvaid linke etteantud aadressi (URI) põhjal. Märkus: ei kontrollita, kui saidi puhul on kasutatud robotitõrje vahendeid (nt fail robots.txt)
CSS Validation Service	CSS kontroll	http://jigsaw.w3.org/css-validator/	Kontrollib CSS (Cascading Style Sheets) või neid sisaldavaid veebilehti
WDG HTML Validator	Vastavus	http://www.htmlhelp.com/tools/validator/	Kontrollib vastavust W3C HTML kodeerimis-standarditele. Saab kontrollida etteantud faili selle aadressi järgi või üles laaditud faili.
A Real Validator	Vastavus	http://arealvalidator.com/	Kontrollib vastavust W3C HTML kodeerimis-standardile, kasutades SGML parserit. Saab kontrollida etteantud faili selle aadressi järgi või üles laaditud faili. Vt ka http://arealvalidator.com/real-validation.html
CynthiaSays	Erinõuded	http://www.cynthiasays.com/	Ligipääsetavuse (kontrollib vastavust puuetega kasutaja vajadustele - nt. piltide ja video tekstiallkirjade olemasolu, graafikute kokkuvõtted jne.), kvaliteedi, privaatsuse kontrollid vastavalt WCAG (ja USA) nõuetele
Canoo WebTest	Üld- otstarbe- line	http://webtest.canoo.com	Veebirakenduste testimisvahend: testide skriptid ja korduvtätimine (vabavara)
Rational Robot jt	Üld- otstarbe- line	http://www.rational.com/	Enamik veebi kontrolli / testimise funktsioone: testimise skriptid, koormustestid, puuduvate linkide ja orbfaillide leidmine jne.
Parasoft WebKing	Üld- otstarbe- line	http://www.parasoft.com/jsp/products/home.jsp?products	Enamik veebi kontrolli / testimise funktsioone: vastavus standarditele, koormustestimine, turvalisuse jm testid

		ct=WebKing&	
SeleniumHQ	Üld-otstarbe-line	http://seleniumhq.org/	Veebirakenduste testimisvahend
Apache JMeter	Koormustestimine	http://jakarta.apache.org/jmeter/	Palju komponente

3.4.8. Pidev testimine ja arendus

Testimise ja laiemalt kvaliteedihalduse ülesanded muutuvad keerukamaks kaasaegsete teenus-orienteeritud arhitektuuride puhul, kus võivad olla kehtestatud teenustaseme lepingud ja nõutav töökindluse tase võib olla väga kõrge (nt „viis üheksat” – süsteem on kättesaadav 99.999% ajast). Hajusarhitektuuride ja pideva arengu puhul võib olla rakendatud „alalise beeta” põhimõte (nt http://en.wikipedia.org/wiki/Perpetual_beta) – süsteem on pidevas arengus ja testimises, ka kasutajate poolt. Selline arenduslaad ei ole sobiv kõrgete töökindlusnõuete puhul. Sellisel juhul on pakutud põhimõtet „testitakse kõike, kõik testivad, testitakse pidevalt” (nt <http://www.itko.com/site/resources/difference.jsp>), mis nõuab testimise automatiseerimist ja kõikide osapoolte poolt teostatavate testide pidevat täitmist ja haldamist.

3.4.9. Kontrollküsimusi ja ülesandeid

- Iseloomustage tarkvara kontrolli korralduse lihtsamaid skeeme. Milline neist on parim? Millal tekib vajadus keerukamaks korralduseks?
- Kontrolli meetodite efektiivsuse võrdlus ja soovitusel kasutamiseks
- Iseloomustage kontrolli korralduse loogikat: olukord ja testitavad objektid, meetodid, tegevused
- V-mudel, arendusprotsessi integreeritud kontroll
- Testimise etapid: ühiku-/mooduli-, integratsiooni-, valideerimise- ja süsteemitestid
- Mida teha, kui testida ei saa?
- Võrdlus: lihtsad skeemid, V-mudel, RUP ja testimine, XP, TPI mudel
- Tooge näiteid erinevatest missioonidest sama arenduse erinevatel iteratsioonidel.
- Millised rollid võiksid vastutada milliste testimise tulemitest eest?
- TPI mudeli põhilised komponendid
- Kontrolli testimise automatiseerimise vahendite liigitus, vajalikud ressursid, eelised, puudused, näited. Oskus kasutada vähemalt kahte testimise automatiseerimise vahendit
- Millal tasub kasutada testimise automatiseerimise süsteeme?
- Veebi testimise kriteeriumid, vahendite liigitus, vahendite näited
- Kuidas areneb testimine / testimise automatiseerimine uutes hajus- / teenus-orienteeritud arhitektuurides?
- Kokkuvõtte meetoditest: idee, eeltingimused, eelised, puudused, tulemused, suhteistesse, hinnang, vahendid
-

4. Kvaliteedihaldus, standardid, normid, audit

Eelmises jaotises vaadeldi tarkvara kontrolli meetodeid (testimine, staatilised meetodid jm) ja korraldust. Need on orienteeritud enamasti toote kvaliteedi parendamisele, kuid sellest alati ei piisa. Näiteks, kui inimeste koostöö lonkab või arendusvahendid ei vasta vajadustele, ei lahenda testimine olulisi probleeme. On vaja midagi, mis analüüsiks ja mõjutaks organisatsiooni, protsesse, vahendeid ja muid tegureid, mis kujundavad tulemuse kvaliteeti. See "midagi" ongi kvaliteedihaldus.

Kursuse teise poole loogika on siis järgmine. Tahame töötada paremini, teenida rohkem [muide, kas tahame ja kui, siis miks?] => tahame pakkuda kvaliteetsemat toodet/ teenust. Tekivad järgmised küsimused, millele allpool lühidalt vastatakse:

- Mis on kvaliteet ja kvaliteedisüsteem? Kuidas neid hinnata, planeerida, hallata?
- Kuidas parandada tarkvaraprotsessi kvaliteeti? Kuidas ühendada kvaliteedihaldus tarkvaraprotsessiga?
- Kuidas kvaliteeti mõõta?
- Kuidas kasutada standardites olevat teadmist?
- Kuidas saada sõltumatut toetust ja hinnangut tarkvara arendusele?

4.1. Kvaliteet

Kvaliteet tundub 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.

Eespool teises jaotises on toodud tarkvara kvaliteediga seotud põhimõisted. Kirjeldame allpool lisaks üldisemalt ühte enam levinud kvaliteedi käsitlust ja sellele vastavaid kvaliteedihalduse meetodeid, süsteeme kvaliteedi hindamiseks, standardeid, protsesse ning tegevusi.

4.1.1. Kvaliteet ja kvaliteedihaldus: 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 ongi kvaliteeti mõnikord määratletud kui juhtimist (A. Feigenbaum: *Quality - a way of managing the organisation*). Et selline käsitlus venitaks teema laiaks, piirdume kvaliteedi aspektidega, mis on levinud Euroopas ja leidnud kajastamist ISO 9000 standardite seerias.

Tarkvara arenduse tulem (toode, teenus) hõlmab mitmesuguseid komponente, mis kõik võivad olla kvaliteedihalduse objektid. Neid vaadeldi eespool, käesoleva materjali teises jaotises.

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. Tarkvara nõuete (kvaliteediatribuutide) süsteeme on kirjeldatud jaotistes 2 ja 5. Nõuded võivad olla orienteeritud kõikidele süsteemiarenduse 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 tihti selliseid etappe nagu spetsifitseerimine, analüüs, disain jne. Etappideks jaotus ja etappide nimetused, samuti nende organisatsioon (XP, 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 võivad mitmete etappide tegevusteks olla süsteemitöö, projektijuhtimine, kvaliteedihaldus ja koolitus). Teises osas kirjeldati mõningaid tarkvaraprotsesse ja tarkvara elutsükli mudeleid.

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

Kvaliteedipoliitika - organisatsiooni tippjuhtkonna ametlikult esitatud kvaliteedialased üldeesmärgid ja juhtnõõrid.

Kvaliteedihaldus - 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.

4.1.2. Miks levib ebakvaliteetne (laiatarbe)tarkvara?

Võib küsida, et kui kvaliteedil ja kvaliteetsel tootel on eelised, miks siis on nii palju ebakvaliteetset (näiteks eaturvalist) tarkvara tootmises ja kasutuses. Majandusseadused peaksid ju välja filtreerima ebaotstarbekad (ebakvaliteetsed) lahendused, järele peaksid jääma kvaliteetsed. Uurimused on näidanud, et paljudel juhtudel on see tõesti nii, samas näiteks laiatarbetarkvara tootjate osas pole asi nii lihtne.

Järgnevalt on toodud mõned põhjused, miks laiatarbetarkvara tootjad ja ostjad võivad eelistada vähemkvaliteetseid tooteid – ja järelikult, miks viletsamad tooted võivad olla tootjale tulusamad.

- Võrgumajandus (Metcalfi seadus – võrgu väärtus on võrdne selle osaliste vaheliste kommunikatsiooniühenduste arvuga, ehk siis liikmete arvu ruuduga) toob kaasa "võitja võtab kõik" efekti. Võitja on esimene, sest tema saab kaasa kõige suurema kasutajate võrgu. Seega võib majanduslikult kasulik olla vigase toote turule toomine ja selle edasine parandamine. Samas viib see lähenemine ebakvaliteetsete toodeteni, sest tagantjärele on ebareaalne kvaliteeti efektiivselt tootesse sisse viia.
- Asümmeetrilise informatsiooni tingimustes ei pruugi turg olla efektiivne (http://en.wikipedia.org/wiki/The_Market_for_Lemons). Antud juhul, kuna kasutajatel on raske vahet teha kvaliteetsete ja ebakvaliteetsete tarkvaratoodete vahel, siis nad

võivad valida odavamad, ning kvaliteedile tehtavad kulutused võivad tootjale olla majanduslikult ebasoodsad.

- Monopoolse jõuga ettevõtted (eriti tarkvaraturul) kaitsevad oma positsioone mitmesuguste meetoditega (näiteks patendid) ning muudavad kasutajate jaoks ümberlülitamise teistele süsteemidele raskeks.
- Kvaliteedi mõiste võib erinevatele kasutajatele tähendada erinevaid asju, seepärast on ebareaalne toota tarkvara, mis kõiki rahuldaks.

Eelnev ei tähenda muidugi, et tootjad ei peaks püüdma toota paremat tarkvara ning et kasutajad ei peaks oma õiguste eest seisma. Olukord on parem tellimustööna loodava tarkvara puhul, kus tellija saab tingimusi küllalt suurel määral ette anda. Lugemist kvaliteedi, eriti turbe majanduslike aspektide kohta: Ross Anderson, Why Computer Security is Hard, <http://www.acsac.org/2001/papers/110.pdf>.

4.1.3. Kvaliteedihalduse mitteformaalseid meetodeid

USA, Euroopa ning Jaapani arusaamad kvaliteedist on olnud mõnevõrra erinevad. Viimastel aastakümnetel on need ühtlustunud. USA ja Euroopa kvaliteedihaldus on olnud rohkem orienteeritud tulemusele - oluline on lõpptulemuse, vähem protsessi kvaliteet. Paraku ei taha see põhimõte keerukate toodete puhul hästi toimida. Jaapanis on kvaliteedihaldus traditsiooniliselt 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 kvaliteedihaldus olnud ajalooliselt enam orienteeritud statistilistele meetoditele, näiteks tootepartiide kontrollile. Jaapanis on see aga pigem iga töötaja ja iga hetke tähelepanu küsimus (*kaizen*).

Philip B. Crosby on sõnastanud kolm populaarset teesi.

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

Philip B. Crosby kvaliteedihalduse 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 kvaliteedihalduse tegevused:

1. Püstita kvaliteedistandardid.
2. Hinda vastavust standardeile.
3. Tegutse, kui standardeid rikutakse.
4. Arenda standardeid edasi.

4.1.4. Kvaliteedi hindamise süsteeme ja auhindu

Kvaliteedi hindamiseks on loodud mitmesuguseid süsteeme. Selliste süsteemide alla võib paigutada ka kvaliteedi konkursid ja auhinnad, näiteks Euroopas EFQM (*The European Quality Award*), Eestis Eesti juhtimiskvaliteedi auhinna ja USA-s Malcolm Baldrige auhinna (*The Malcolm Baldrige National Award for Quality*).

EFQM mõõdab järgmisi kvaliteedi komponente:

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

Ka Eesti juhtimiskvaliteedi auhinna puhul rakendatakse EFQM põhimõtteid.

Malcolm Baldrige 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).

Maksimaalne võimalik kogusumma on tuhat, senised maksimaalsed hinnangud on olnud 700 piires.

4.1.5. ISO 9000 seeria standardid

ISO 9000 seeria: võimalused

- Protsesside parendamine ettevõttes
- Süstemaatiline (IT alane) kvaliteedihaldus
- Sertifikaadi taotlemine ja ettevõtte taseme teadvustamine avalikkusele
- Kasulik kui orienteerutakse ekspordile (EL, aga ka mujale)

ISO 9000 seeria: hetkeseis

- esimene versioon 1987, uuendatud 1994 (u. 20 standardit), uuendatud 2000 ("ISO 9000 2000 standardid")
- ISO 9000:2005, Quality management systems. Fundamentals and Vocabulary (eelmine versioon oli aastast 2000)
- ISO 9001:2008, Quality management systems. Requirements
- ISO 9004:2000, Quality management systems. Guidance for performance improvements
- Lisaks mitmesuguseid rakendusmaterjale, muuhulgas standard *ISO/IEC 90003, Software engineering — Guidelines for the application of ISO 9001:2000 to computer software* sisaldab suuniseid ISO 9001:2000 kohaldamiseks tarkvara väljatöötamisele, tarnimisele, installeerimisele ja hooldusele

ISO 9000 seeria standardid on kasutusel kõigis tööstusharudes, olulisemad neist on tõlgitud ka eesti keelde ja üle võetud eesti standardina.

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

4.1.6. Kvaliteedihalduse protsessid - esimesed sammud ettevõttes

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

Kõigepealt tuleb lähtudes ettevõtte strateegiast planeerida kvaliteedihalduse eesmärgid (näiteks, kas soovitakse ainult töökorralduse parendamist või lisaks sellele 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 tööloik, kus kvaliteedihaldus annab suurima efekti. Kui valitud meetod õigustab ennast selles lõigus, võib üle minna järgmistele kriitilistele valdkondadele. Vajadusel korrigeeritakse meetodit või kvaliteedipoliitikat.

ISO 9000 seeria ja teised kvaliteedihalduse standardid sisaldavad detailseid juhiseid kvaliteedisüsteemi ja kvaliteedihalduse protsesside kavandamiseks, kvaliteedihalduseks, vajalike

ressursside ja infrastruktuuri planeerimiseks ja haldamiseks, kvaliteedisüsteemi realiseerimiseks ja uuendamiseks.

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 tööloik
5. parandage seda tööloiku vastavalt valitud meetodikale
6. kui põhimõtted ja meetodika on ennast õigustanud ennast, minge tagasi sammule 4
7. kui meetodika või kvaliteedipoliitika vajavad ümbervaatamist, minge tagasi sammudele 3, 2 või 1

4.1.7. Kontrollküsimusi ja ülesandeid

- Kvaliteet, tarkvara, nõuete liigitusi, elutsükkel, elutsükli liigid
- Kvaliteedihaldus, -poliitika, -süsteem, -tõendus
- Miks levib ebakvaliteetne tarkvara?
- Kvaliteet ja organisatsiooni juhtimine
- Kvaliteedihalduse ja arendusmeetodite seos
- Kvaliteedihalduse integreerimine elutsükklisse
- Kvaliteedihalduse käsitlusi, teese, süsteeme ja auhindu
- ISO 9000 seeria, ISO 90003
- Mida peaks süsteemi arendaja (tellija, kasutaja, hooldaja, tarnija, firmajuht) teadma kvaliteedist ja standarditest (vt ka kursuse sihtrühmade ülevaade ning lugemissoovitusi materjali algusest)?

4.2. Tarkvara arendus ja kvaliteedihaldus (protsessi kvaliteet)

Nagu ülal märgitud, peab keeruka toote puhul (tegelikult ka paljudel muudel juhtudel) hea tulemuse saamiseks olema kvaliteetsed nii toode kui ka tootmise protsess. Tarkvara puhul räägime tarkvara elutsüklit ja selle protsessidest.

4.2.1. Arendus ja kvaliteedihaldus

Kvaliteeti ei saa sisse testida. Halvasti arendatud süsteemi pole võimalik kontrolli abil heaks muuta. Lõpptulemus sõltub kogu arenduskeskkonnast ja -protsessidest, sealhulgas:

- tarkvara arenduse meetoditest: üldised (struktuurne analüüs, disain ...), spetsiaalsed (veakindel programmeerimine, n-versiooniline programmeerimine, transaktsioonide

haldamine, taastamisvahendid ...), kontrolli meetodid (läbivaatused, verifitseerimine, testimine, valideerimine ...) ja muud

- riistvara vahenditest: piisavalt võimas keskkond, töökindlust toetavad vahendid jne
- tarkvara arenduse vahenditest: arenduskeskkonnad, programmeerimiskeeled, dokumenteerimisvahendid jne
- tarkvaraspetsiifilisest kvaliteedihaldusest: elutsükli haldus, atribuudid, meetrikad, mõõtmine, ISO 90003, SEI CMMI jne
- kasutatavatest standarditest: elutsükliid, analüüs, disain, testimine, turve jne
- äri- ja töökeskkonnast, juhtimisest, organisatoorsest vahenditest jne

Kvaliteedihaldust on tarkvarasüsteemi elutsükklisse mitmeti ühendatud:

- kui tarkvara arendus alles hakkas tekkima, tegeldi peaaesjalikult programmeerimisega ja kvaliteedist ei räägitud
- üsna kiiresti leiti, et vigane tarkvara võib olla väga ohtlik kasutajate varale ja tervisele; seepärast hakati tegelema testimisega, järgnesid inspeksioonid, tõestamine ja muud meetodid. Esialgu moodustasid need ühe etapi arendusest
- selgus, et ka sellest ei piisa, kõik etapid ja tegevused peaksid olema vajalikul tasemel - kvaliteedihaldus on integreeritud elutsükli kõigisse etappidesse
- on ka samastatud kogu arendusprotsessi ja kvaliteedihaldust

Kursuses on valitud kesktee - kvaliteedihaldus on integreeritud elutsükli kõigisse etappidesse, muuhulgas on ta kõikide elutsükli komponentide loomulik osa.

4.2.2. Oma metoodika näide: tarkvaraarenduse 10 põhikomponenti

Tarkvara modelleerib tegelikkust ja võib olla väga keerukas, samuti võib olla väga keerukas selle arendus. Käesolevas materjalis on toodud näiteid erineva taseme tarkvara elutsükli mudelite ja protsessiraamistikkude kohta, näiteks (liikudes arenduse mudelistest protsessimudelite suunas): triviaalsed elutsükli mudelid, kosemudel, V-mudel, spiraalmudel, Rational Unified Process, agiilsed raamistikud (nt XP), Zachmani raamistik, tarkvara elutsükli protsesside standardid, CMMI, SPICE, ISO/IEC TR 15504.

Alguseks ja väikeste projektide puhul on detailsed raamistikud mahukad ja keerukad. Tegelikult nõuab siiski iga mittetriviaalne raamistik kasutuselevõtuks pingutust. Kas oleks võimalik sellistest raamistikkudest loobuda ja pigem püüda määratleda, millised aga võiksid olla tarkvara arenduse ja juhtimise põhikomponendid? Siis võiksime nendest ise oma metoodikaid koostada.

Ühtset loetelu põhikomponentidest ei ole, erinevates rakendustes on need erinevad (seepärast muide ongi universaalsed raamistikud keerukad, et nad püüavad hõlmata kõiki olukordi). RUP metoodikale tuginedes on pakutud näiteks valik järgmistest tarkvaraprotsessi kõige olulisematest komponentidest.

1. **Eesmärgid, äriavaade.** Laiem taust, eesmärgid, tasuvuse analüüs. Milleks seda tarkvara üldse tehakse, mis probleeme lahendatakse? Kas ta tasub ennast ära?

- 2. Nõuded.** Nõuded võivad olla formaalselt või mitteformaalselt esitatud. Mis probleeme lahendatakse? Kes on huvipooled, millised on nende ootused? Kes on kasutajad, millised on nende vajadused? Millised on põhiterminid/mõisted? Mida tarkvara peab tegema, mida mitte? Millises keskkonnas (nt. seadusandlus, organisatsioon, tehnoloogia) hakatakse tarkvara kasutama? Millised on funktsionaalsed nõuded? Mittefunktsionaalsed nõuded? Kavandamise kitsendused? Selle kursuse materjalidest võib nõuete esitamiseks kasutada näiteks RUP või agiilsete tehnoloogiate vahendeid, lisas toodud standardit ANSI/IEEE Std 830-1984 nõuete vormistamiseks, standardis ISO/IEC 9126 toodud kvaliteedi atribuute, standardis EVS-ISO/IEC 12207 toodud elutsükli protsesside kirjeldusi, ISKE raamistikku jne.
- 3. Plaan(id).** Näiteks projektorganisatsiooni, projektijuhtimise, ajakavade, nõudmiste haldamise, kvaliteedihalduse, konfiguratsioonijuhtimise, testimise, hindamise, vastuvõtmise ja muud plaanid. Minimaalsel tasemel võivad plaanid olla mitteformaalsed ja lühikesed. Selle kursuse materjalidest võib kasutada näiteks standardis EVS-ISO/IEC 12207 toodud elutsükli protsesside kirjeldusi, standardit ANSI/IEE Std 829 testide planeerimiseks ja dokumenteerimiseks või erinevaid infoturbe standardeid turvapoliitika ja -meetmete kavandamiseks. Vastuvõtmise plaani koostamiseks võib kasutada funktsionaalse testimise meetodeid (nt. riskipõhine testimine, ekvivalentsiklassid, piirjuhud), vormistades neid vajadusel standardi ANSI/IEE Std 829 põhjal.
- 4. Riskid ja nende maandamise võimalused.** Kuidas projekt võib nurjuda ja milliseid meetmeid tuleks kavandada. Võib kasutada näiteks standardit EVS-ISO/IEC 27005 infoturbe riskide analüüsiks.
- 5. Probleemide haldamine.** Probleemid tuleb sellistena teadvustada (teadvustamata probleemiga on raske hakkama saada), lahendada ja neid tuleb jälgida. Haldamise käigus võib kasutada läbivaatusi ja mitmesuguseid kontrolli korralduse meetodeid, samuti ITIL-is või standardis EVS-ISO/IEC 12207 toodud protsesside kirjeldusi.
- 6. Arhitektuur.** Süsteemi komponendid, nende vahelised liidesed ja interaktsioonid. Erinevad arhitektuuri vaated, kui vaja. Selle kursuse materjalidest võib kasutada näiteks standardis EVS-ISO/IEC 12207 toodud elutsükli protsesside kirjeldusi.
- 7. Arendus/toode - näide.** Iteratiivselt kodeeri, koosta, testi. Iga iteratsiooni lõpus on olemas hinnatav versioon. Võib kasutada mitmeid selles kursuses toodud testimise ja kontrolli meetodeid ja standardeid, sealhulgas funktsionaalset testimist, testimist programmi teksti põhjal, statistilisi meetodeid, kontrolli korralduse meetodeid jne.
- 8. Hindamine.** Hinda versioone, hinda riske, otsusta vastuvõtmine. Võib kasutada standardit ANSI/IEE Std 829 testide planeerimiseks ja tehtu dokumenteerimiseks, samuti standardis EVS-ISO/IEC 12207 toodud elutsükli protsesside kirjeldusi..
- 9. Muudatuste haldamine.** Tarkvara muutub vältimatult (miks?). Muudatuste haldamine võib sisaldada vajaduste/päringute tuvastamist, lahenduste kavandamist, muudatuste sisseviimist, muudatuste jälgimist. Haldamise käigus võib kasutada läbivaatusi ja mitmesuguseid kontrolli korralduse meetodeid, samuti ITIL-is või standardis EVS-ISO/IEC 12207 toodud protsesside kirjeldusi.
- 10. Tugi.** Tuleb vastata kasutaja küsimustele, lahendada tema probleeme, koolitada, analüüsida lisavajadusi jm. Selle kursuse materjalidest võib kasutada näiteks ITIL-is või standardis EVS-ISO/IEC 12207 toodud protsesside kirjeldusi.

Näeme, et programmi kirjutamine, mis tarkvara arendusega tutvumisel võib esmapilgul näida ainukese tegevusena, on väga oluline, kuid tegelikes rakendustes üks osa ühest põhikomponendist. Kvaliteedihaldus kuulub kõikidesse nendesse tegevustesse.

4.2.3. EVS-ISO/IEC 12207 Infotehnoloogia - tarkvara elutsükli protsessid

Standardist ISO/IEC 12207 on olemas mitu järjestikust versiooni. Viimane neist on ISO/IEC 12207:2008 "Süsteemi- ja tarkvaratehnika. Tarkvara elutsükli protsessid" (eestikeelne versioon on EVS ISO/IEC 12207:2009). See ühendab süsteemi- ja tarkvara elutsükli protsessid ning sisaldab 43 protsessi - oluliselt rohkem kui eelnevad versioonid. Tarkvara protsessidega tegelemiseks on lihtsam alustada eelmiste versioonidega, mida tutvustatakse allpool.

Standard EVS-ISO/IEC 12207 võib olla aluseks ettevõtte tarkvaraprotsesside haldusele. Selle evitamise hõlbustamiseks on välja töötatud juhend ISO/IEC TR 24748-3 (eelmine versioon oli ISO/IEC TR 15271), mida on kasulik järgida.

Oluline on teada, et ISO/IEC 12207 ei kirjuta ette konkreetset elutsükli mudelit ega tarkvara arendusmeetodit. Standardiga sobivad väga mitmesugused elutsükli mudelid, sealhulgas agiilsed. Standardit järgivate huvipoolte hooleks jääb elutsükli mudeli valimine tarkvaraprojekti tarbeks ning standardi protsesside, tegevuste ja tööde peegeldamine selles mudelis.

Standard jagab tarkvara elutsükli protsessid kolme gruppi:

- Primaarprotsessid (5): Hankimine- Tarnimine - Arendus - Eksploatatsioon - Hooldus
- Abiprotsessid (8): Dokumenteerimine - Konfiguratsiooni haldus - Kvaliteedi tagamine - Verifitseerimine - Valideerimine - Ühisläbivaatus - Auditeerimine - Probleemide lahendamine
- Organisatsioonilised protsessid (4): Haldus - Infrastruktuur - Täiustamine - Koolitus

Iga protsess koosneb tegevustest, tegevused omakorda töödest.

Kokku on 23 protsessi, millega on seotud 95 tegevust. Toome järgnevalt mõnede protsesside tegevuste loetelu.

Hankimine

- algatamine
- pakkumiskutse koostamine
- lepingu sõlmimine ja uuendamine
- tarnija seire
- vastuvõtmine ja lõpetamine

Tarnimine

- algatamine
- vastuse koostamine
- leping
- plaanimine
- täitmine ja juhtimine
- läbivaatus ja hindamine
- üleandmine ja lõpuleviimine.

Arendus

- protsessi teostamine
- süsteeminõuete analüüs
- süsteemi arhitektuuri projekteerimine
- tarkvaranõuete analüüs
- tarkvara arhitektuuri projekteerimine
- tarkvara detailprojekteerimine
- tarkvara programmeerimine ja testimine
- tarkvara integreerimine
- tarkvara kvalifitseerimistestimine
- süsteemi integreerimine
- süsteemi kvalifitseerimistestimine
- tarkvara installeerimine
- tarkvara vastuvõtmise toetamine

Ekspluatatsioon

- protsessi teostamine
- katseekspluatatsioon
- süsteemi ekspluatatsioon
- kasutaja toetamine

Hooldus

- protsessi teostamine
- probleemide ja muutmise analüüs
- muudatuste teostamine
- hoolduse läbivaatus ja kinnitamine
- migratsioon
- tarkvara mahavõtt

Kvaliteedi tagamine

- protsessi teostamine
- toodete tagamine
- protsesside tagamine
- kvaliteedisüsteemide tagamine

ISO /IEC 12207 evitamine sisaldab soovitatavalt järgmisi samme.

- evituse plaanimine
- ISO/IEC 12207 kohandamine
- pilootprojekti(de) läbiviimine
- meetodika formaliseerimine
- meetodika ametlikustamine

Harilikult on soovitatav mitte püüda evitada kogu ISO/IEC 12207 korraga, vaid alustada neist protsessidest, millega saavutatakse kõige tunduvad kasulikud tulemid.

ISO/IEC 12207 ei määratle protsesside, tegevuste ja tööde järjestust ega kirjuta ette mingit konkreetset tarkvara elutsükli mudelit.

Selles järgus on kasulik projitseerida praegused protsessid, menetlused ja/või meetodid ISO/IEC 12207 protsessidele, tegevustele ja töödele.

Sellist projektsiooni võib kasutada meetodika täielikkuse kontrolliks, st praeguse olukorra ja sihtolukorra (kus kasutatakse ISO/IEC 12207 protsesse) vaheliste lõhede väljaselgitamiseks.

4.2.4. CMMI

Väljatöötaja: Software Engineering Institute, Carnegie Mellon University.

Link: <http://www.sei.cmu.edu/cmmi/>

Sisu:

- sisaldab mitut mudelit (hange, arendus, teenused, inimesed)
- laiendab ja kombineerib eelnevaid mudeleid - the Capability Maturity Model for Software (SW-CMM), the Systems Engineering Capability Model and the Integrated Product Development Capability Maturity Model.
- tarkvara arenduse, hoolduse ja muude protsesside head tavad
- enesehindamine, võrdlus teistega
- 5 taset: Level 1 (initial), 2 (managed), 3 (defined), 4 (predictable) and 5 (optimizing)
- taseme- ja jätkuv mudel
- protsessid ütlevad, mida tehakse
- tasemed ütlevad, kui hästi seda tehakse

Tugevused:

- Võib valida sobiva mudeli
- Detailne
- Spetsiaalselt tarkvara arendavatele organisatsioonidele
- Rõhub pidevale arengule, mitte vaid sertifitseerimisele
- Võib kasutada nii sertifitseerimise, täiendamise kui ka enesehindamise vahendina

4.2.5. ITIL ja ISO/IEC 20000

ITIL (IT Infrastructure Library) on IT teenuste haldamise parima praktika kogum. ISO/IEC 20000 on kaheosaline standard, mis ühildub ITIL-iga (Part 1: Specification for information technology service management; Part 2: Code of practice for information technology service management). ITIL kohta saab teavet näiteks aadressilt <http://www.itil-officialsite.com/AboutITIL/WhatisITIL.aspx>, ISO/IEC 20000 kohta – aadressilt www.iso.org.

4.2.6. ISKE ja IT etalonturbe käsiraamat

Infosüsteemide kolmeastmeline etalonturbe süsteem (ISKE) põhineb infovarade turbeastme määramisel, varade kirjeldamisel tüüpmodulite abil ning turvameetmete valikul vastavalt tüüpmodulitele ning turbeastmele. ISKE rakendamine sisaldab järgmisi tegevusi:

- Infovarade inventuur ja spetsifitseerimine
- Andmekogude turvaklasside määramine

- Muude infovarade turvaklasside määramine
- Turvaklassiga infovarade turbeastme määramine
- Tsoonide vajaduse analüüs, asutuse tsoneerimine vajadusel
- Tüüpmodulite märkimine infovarade spetsifikatsioonidesse
- Turbehalduse meetmete loetelu koostamine
- Turvameetmete rakendamise plaani koostamine
- Turvameetmete rakendamine
- Tegelik turvaolukorra kontroll, vajadusel täiendavate meetmete rakendamine

ISKE põhineb Saksamaa Infoturbeameti (Bundesamt für Sicherheit in der Informationstechnik, BSI) poolt publitseeritaval IT etaloniturbe käsiraamatul (IT Grundschutzhandbuch'il). BSI süsteem on väga ulatuslikult ja detailselt dokumenteeritud, on kooskõlas ISO/IEC 27000 seeria infoturbe standardite perega ning seda täiendatakse regulaarselt kord aastas.

ISKE materjalid koos linkidega IT etaloniturbe käsiraamatule on kättesaadavad aadressil <http://www.ria.ee/ISKE>.

4.2.7. Kas "tardmeetodid vs paindmeetodid" või "haldamisraamistikud vs arendusmeetodid"?

On ilmunud mitmeid artikleid, kus vastandatakse omavahel uusi meetodikaid, nagu näiteks XP, sellistele standarditele nagu näiteks ISO/IEC 12207, CMM/CMMI või SPICE. Väide on enamasti, et esimesed on "paind-" või "kerged" meetodid ning sobivad igapäevaelu, kus on vaja kiiresti arendada muutuvate nõudmiste tingimustes, teised aga "tard-" või "jäigad" meetodid ning sobivad missioonikriitilistesse rakendustesse. Lähemal tutvumisel mõlemate meetodikatega selgub siiski, et selline vastandamine võib olla tekkinud pigem standardite ebapiisavast tundmisest, traditsioonidest ning uute meetodite loojate eristumispüüdest.

Üldalmainitud standardid võimaldavad rakendada mitmesuguseid arendusmeetodikaid. Näiteks on ISO/IEC 12207 rakendusjuhendis ISO/IEC TR 15271:1998 pakutud standardi evitamiseks kolme võimalikku elutsükli mudelit: koskmudel, inkrementmudel (arendamine koosneb mitmest iteratsioonist) ja evolutsioonmudel (ka nõuded selgitatakse arenduse käigus). Neist viimane (võib-olla ka teine, sõltub XP esitusest) vastab XP meetodikale. On leitud ka, et XP meetodika on sobiv CMMI teise taseme saavutamiseks (NB! Juba teisel tasemel jääb suur osa CMMI protsesse siiski XP-st välja).

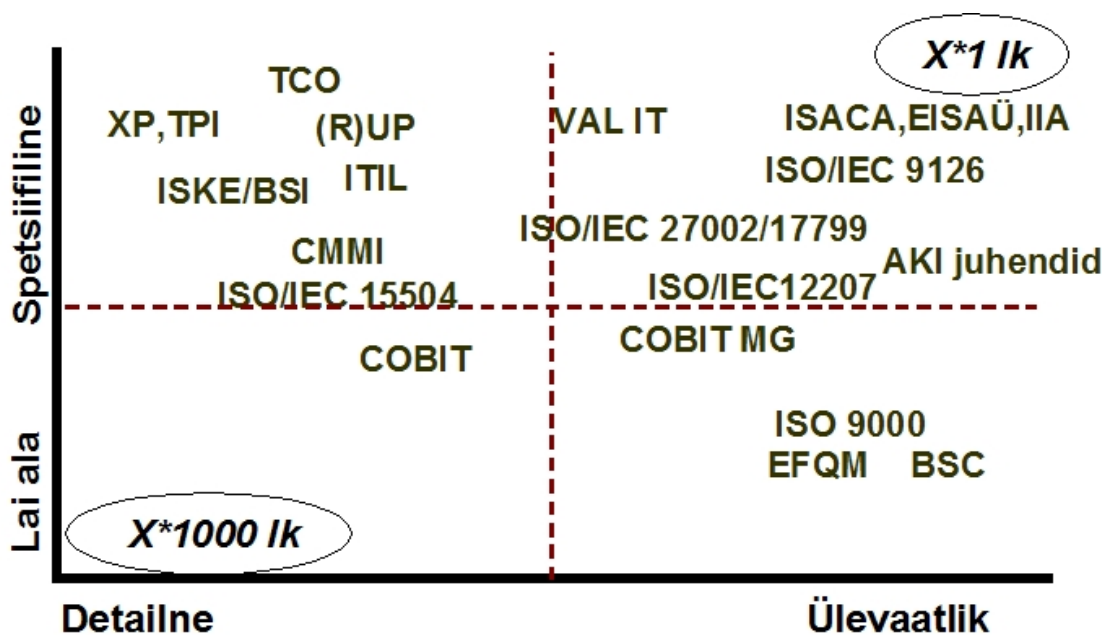
Erinevus on siis pigem selles, et arendusmeetodid räägivadki põhiliselt tarkvara arendusest ja sellega seotud protsessidest, kuna aga laiemad standardid käsitlevad kõiki tarkvara elutsükli protsesse. Tõepoolest, XP ei tegele näiteks standardis ISO/IEC 12207 määratletud hanke, eksploatatsiooni, tarne, hoolduse, koolituse ja mitmete teiste protsessidega - see ei ole lihtsalt XP teema. Pigem võiks seega rääkida haldamisraamistikudest (ISO/IEC 12207, CMM jne) ning arendusmeetodikatest (XP ja teised), millel on erinev ulatus, aga mis ei vastandu omavahel.

"Agiilsete" meetodite, nagu XP ja tema analoogide loojate teene on evolutsioonmudeli detailne väljatöötamine, mitmete inspireerivate detailide lisamine, nende meetodikate aktiivne ning tulemuslik kasutuselevõtt ning nende propageerimine. Kombinatsioon organisatsiooni tegevusele

suunatud laiemast raamistikust, mille raames kasutatakse evolutsioonimudelit, nagu seda on näiteks XP, võiks olla ahvatlev mitmetele Eesti ettevõtetele.

4.2.8. IT standardid ja meetodikad: üldpilt

Võimalike meetodikate ja raamistike valik on siis üsna suur. Mida valida, sõltub vajadustest. Järgneval joonisel on kujutatud valik ITstandardeid ja meetodikaid kahe dimensiooni lõikes. Vertikaalteljel on hallatava valdkonna ulatus – laiast spetsiifiliseni. Horisontaalteljel on detailsuse aste – detailsest ülevaatlikuni. All vasakul paikneksid siis väga laiad ja väga detailed materjalid – nende maht on tuhandetes ja enamgi lehekülgedes, selliseid on vähe. Ülal paremal on väga spetsiifilised ja üldised materjalid – nende maht võib olla lehekülgedes. Selliseid materjale on palju, kuid oma spetsiifilisuse ja üldisuse tõttu pole nad üldjuhul väga tuntud. Seepärast paiknevad enamtuntud materjalid joonisel peamiselt piki langevat diagonaali. Osa joonisel toodud materjalidest on kirjanduse loetelus või tekstis viidatud, osa on kergesti leitavad veebist.



4.2.9. Millest alustada?

Eelmistes jaotistes on toodud erinevaid lähenemisi (tarkvara) kvaliteedihaldusele: lihtsate põhimõtete rakendamine, kvaliteedi auhinnad, ISO 9000 seeria (sealhulgas tarkvarale orienteeritud ISO 90003). Infotehnoloogiaettevõtete kvaliteedihalduse aluseks võib võtta ka standardi EVS-ISO/IEC 12207, meetrikad, COBIT meetodika, CMMI, tarkvaraprotsesside täiustamise ja võimete määramine (SPICE, vt. <http://www.sqi.gu.edu.au/SPICE/>), standardi *ISO/IEC TR 15504 Information technology — Software process assessment*, ITIL (IT Infrastructure Library), TCO (Total Cost of Ownership). Üldisematest kvaliteedijuhtimise raamistikudest tulevad veel kõne alla näiteks Six Sigma ja tasakaalus tulemuskaart.

Arvestada tasub kindlasti agiilseid meetodikaid (nt XP), tarkvarafirmade meetodikaid (enamikul suurematest firmadest on oma arendus- või vähemalt projektijuhtimise meetodika) ning võimalust luua ise oma protsessimudel.

Laias laastus võib kõiki neid raamistikke jagada infotehnoloogiale orienteerituteks ja üldisteks, viimaste hulka kuuluvad eelmainitute ISO 9000, Six Sigma, tasakaalus tulemuskaart. Samuti eristub laiapõhjaliste (paljudele elutsükli protsessidele orienteeritud) raamistikkude rühm: EVS-ISO/IEC 12207, ISO 90003, SPICE, CMMI, ISO/IEC TR 15504.

Nii EVS-ISO/IEC 12207 kui ka EVS-EN ISO 9000 on vastu võetud eesti standardina, mis hõlbustab nende kasutamist. Esimesel puudub sertifitseerimine, teisel on olemas rahvusvaheline tunnustamise meetodika ja infrastruktuur. Esimene on protsesside juhtimisele orienteeritud, näiteks eksploatatsioon 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 ühega mainitud standarditest.

Algatus alustamiseks võib tulla mitmest kohast, näiteks ISO 9000, CMMI, COBIT rakendamise puhul – klientidelt või partneritelt; ISO 9000, kvaliteediauhinna, Six Sigma puhul – emafirmalt või juhtkonnalt; COBIT puhul – kõigilt eelpoolmainitult, aga ka audiitoritelt; IT reeglistikkude, EVS-ISO/IEC 12207, CMMI puhul – IT osakonnast.

Igal juhul on mõtet alustada. Nagu ka kvaliteedijuhtimise puhul, peaks kõigepealt rakendama tervet mõistust. Erinevaid raamistikke võibki vaadata kui rahvusvahelise kogemuse süstematiseeritud esitust, enamaltjaolt nad ei vastandu, neid võib kasutada (ja kasutatakse) koos.

4.2.10. Kontrollküsimusi ja ülesandeid

- Millest alustada tarkvaraprotsessi kvaliteedihalduse kavandamisel?
- Tarkvaraarenduse olulised komponendid ja protsessid. Kuidas on kvaliteedihaldus seotud nende protsessidega?
- Võrdlus: protsessihalduse raamistikud ja arendusmeetodikad
- Ülevaade tarkvara elutsükli mudelistest
- Põhimõisted, eesmärgid, ülevaade, kuidas kasutada – EVS-ISO/IEC 12207, CMMI, RUP, ITIL, ISKE
- Võrdlus: ISO 9000 seeria, ISO 90003, EVS-ISO/IEC 12207, CMMI, ITIL, ISKE. ISO/IEC 20000, RUP, XP.

4.3. *Kvaliteedi mõõtmine: meetrikad*

Tarkvara arendusprotsessi (arendusprotsessi kvaliteeti) on vaja mõõta või hinnata mitmel põhjustel - näiteks selleks, et teha korrektset pakkumist, et otsustada toote vastuvõtmise üle või et osata arendajate tööd rahaliselt väärtustada. Selliseks mõõtmiseks kasutatakse tarkvara meetrikaid - arvatavaid või hinnatavaid suursi, mis iseloomustavad olulisi omadusi ja võivad olla seotud ülesande, projekti, dokumendi või ettevõttega. Tarkvara meetrikate kasutamine on efektiivne vahend arendusprotsessi parandamiseks, kuid sellega peavad kaasnema teised kvaliteedihalduse tegevused ning võrdlevad hinnangud teiste arendajatega.

4.3.1. Ülevaade tarkvara meetrikatest

Võimalikke 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. Tuuakse näiteid ka teistsuguste meetrikate kohta.

Üks ideelt selgem ja mitmeti kasutatav meetrika on koodi ridade arv. Kuid 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

Programmeerimismeetrikad hindavad / iseloomustavad programmi (mahtu, keerukust), hindavad kvaliteeti, toetavad arendusprotsessi (prognoosivad töö mahtu, ridade arvu).

Esimesed meetrikad ilmusid seitsmekümnendate aastate algul. Tuntumad olid McCabe programmi keerukuse mõõt (1976) ja Halsteadi tarkvara teadus (1977). Toome allpool veel kvaliteedimeetrikate näiteid. Milline on väärtuse määramispiirkond, parim võimalik väärtus?

- Spetsifikatsiooni muutmise tase = $(E+M+L)/A$, kus E - eemaldatud funktsioonid, M - muudetud funktsioonid, L - lisatud funktsioonid, A - algne funktsioonide arv.
- Tehnilise ülesande ja spetsifikatsiooni vastavus = $(\text{Funktsioonide arv tehnilises ülesandes}) / (\text{Funktsioonide arv spetsifikatsioonis})$.
- Keskmine tõrgetevaheline aeg = $(\text{Funktsioneerimise kestvus}) / (\text{Tõrgete arv})$

Meetrikaid saab kasutada, võrreldes neid teada oleva soovitava tasemega või integreerides neid kvaliteedikriteeriumideks. Meetrikate 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.

Allpool on toodud valik meetodikate korduma kippuvatest küsimustest koos vastustega:

- "Mind ei huvita tehnilise ülesande ja spetsifikatsiooni vastavus, mind huvitab, kas see tarkvara teeb, mida vaja!" Lahendus - meetrikate integreerimine või võrdlemine meetrikate andmebaasidega
- "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 meetrikate väärtustamise meetodikad, kasutada võrdlusandmeid
- "Millised probleemid võivad kaasneda meetrikate sisseviimisega?" Meetrika peab olema sisukas ja oluline, tema sisseviimise võimalikud efektid peavad olema hinnatud

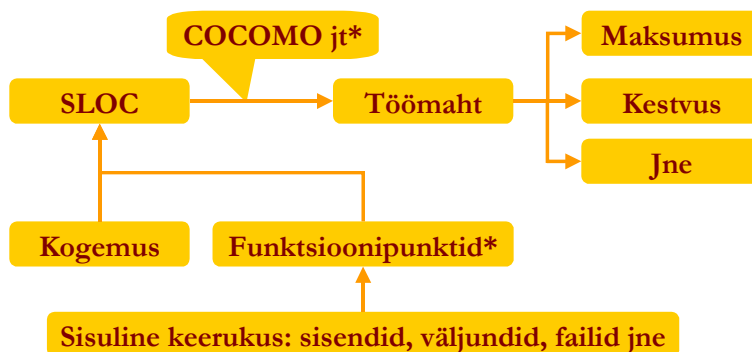
4.3.2. Tarkvara arendusmaksumuse prognoos

Meetrikate rakendamise näitena vaatame tarkvara arendusmaksumuse prognoosi. See on maailmas tunnustatud kui keerukas ülesanne. Ideaalis, selle ülesande sisendiks on tarkvara spetsifikatsioon ning väljundiks - arenduse maksumus või arenduseks vajalike inimkuude (-päevade, -aastate) arv. Kahjuks pole head spetsifikatsiooni alati olemas, ka on palju parameetreid peale funktsionaalsuse, mis maksumust mõjutavad, näiteks mittefunktsionaalsed nõuded (töökindlus, turvalisus jne), arenduskeskkond, arendusvahendid, arendajate tase ja nii edasi. Et paljusid neist suurustest ei saa lihtsal viisil mõõta, on nende hinnangud paratamatult kogemuslikku laadi. Seega peab meil olema olemas eelnev võrdlusbaas - mitmesuguste parameetritega projektide tegelikud maksumused. On loodud meetodeid ja tarkvara, mis neid ja mitmeid muid tegureid arvesse võttes ning tuginedes eelnevale kogemusele leiavad prognoositava maksumuse.

Rakendatakse mitmesuguseid meetodeid, enamlevinud on tarkvara mahu hinnang ja selle baasil maksumuse hinnang. Mahtu hinnatakse kas koodi ridade arvu või funktsioonipunktide arvu põhjal. Funktsioonipunktid (vt. nt. David Garmus, David Herron. Function Point Analysis: Measurement Practices for Successful Software Projects. Addison-Wesley Information Technology Series, 2000) iseloomustavad tarkvara sisulist keerukust ning nende arv leitakse arvestades selliseid parameetreid nagu sisendite arv, väljundite arv, failide arv jne. Jällegi on tegemist empiiriliste mudelitega eelnevate projektide kogemuste baasil.

Teades orienteeruvat tarkvara mahtu, on võimalik prognoosida arenguks vajalikku töömahtu. Selleks kasutatakse mitmesuguseid mudeleid, enamlevinud on COCOMO mudel (Boehm, B. ja teised. Software Cost Estimation with COCOMO II. Prentice-Hall, 2000) ja selle edasiarendused. Nendes mudelites kasutatakse süsteemi kalibreerimiseks ja prognoosi andmiseks olemasolevate projektide andmebaase (vt joonis). Et selliste andmebaaside tekitamine ja hooldamine ning nende baasil prognoosisüsteemide loomine on töömahukas, võivad arvestatavad tarkvara maksumuse prognoosimise süsteemid olla üsna kallid. Esimeseks katsetamiseks sobib hästi Costar (<http://www.softstarsystems.com/>) demoversioon.

Tarkvara arendusmaksumuse prognoos



- *Empiirilised mudelid eelnevate kogemuste baasil: projektide AB-d jne
- *Võimalus kalibreerida mudeleid
- *Ka kalibreeritud mudelid võivad anda tegelikest erinevaid tulemusi
- *Olemas kirjandus ja tarkvara
- SLOC - source lines of code

Tarkvara maksumuse hinnang ilma sellise võrdlusbaasita on ebarealistlik. Hindamiseks võib seega kasutada kirjanduses avaldatud kalibreeritud prognoosimeetodeid või prognoosipakette. On leitud, et saadavad prognoosid võivad ka oma kalibreerituse alas anda tulemusi, mis erinevad tegelikest (näiteks, kui arendus on olnud ebaedukas, siis edukatel projektidel põhinevad kestvuse prognoosid ei kehti). Siiski on sellised prognoosid väärtuslikud arenduse eeldatava maksumuse ja edukuse hindamisel.

Tarkvara arendusmaksumuse prognoosi kokkuvõtteks:

- Tarkvara maksumuse prognoosi enamlevinud meetodid kasutavad tarkvara mahu prognoosi ja selle baasil töökulu prognoosi
- Mõlemad prognoosid nõuavad seniste projektide andmebaase ja nende põhjal prognoosimetoodika kalibreerimist
- Tarkvara mahu prognoosi võib teha, kasutades olemasolevaid publitseeritud materjale või prognoositarkvara. Prognoositarkvara vähendab töömahtu
- Prognoosid on kasulikud, kuid võivad ka oma kalibreerituse alas anda tulemusi, mis erinevad tegelikest. Seepärast on kasulik arvestada lisaks lokaalseid, ettevõtte oma kogemusi ja andmeid

4.3.3. Kontrollküsimusi ja ülesandeid

- Meetrikate idee, omadusi, efekt, probleemid, liigitusi, kasutamine
- Kvaliteedimeetrikate näiteid, väärtuse määramispiirkond, parim väärtus
- Tarkvara mahu ja arendusmaksumuse prognoos, selle vahendid, usaldatavus

4.4. Standardid

Kogu tänapäeva tehnika ja seega ka suurem osa igapäevasest elust põhineb standarditel. Nad toimivad tagaplaanil ja me märkame seda tavaliselt alles siis, kui tekivad probleemid - näiteks, kui tahame USAs lükata oma fööni voluvõrku. Standardid püstitavad nõuded suurtele toote või rakenduse klassidele. Anname allpool standardi mõiste ja ülevaate tarkvara puudutavatest standarditest.

4.4.1. Standardi mõiste

Standard on konsensusel alusel koostatud ja tunnustatud kehami poolt kinnitatud normdokument, mis on suunatud standardimiseesmärkide saavutamiseks. Kehami all mõistetakse siinjuures juriidilist või haldusüksust, millel on kindel ülesanne ja koosseis, nt organisatsioon, firma, ettevõtte, ametkond, fond vms. Konsensus on oluline, sest on oht, et standard võib anda eeliseid mingile kindlale organisatsioonile. Standardi kehtestamiseks on vaja, et ka teised seda aktsepteeriks, standardimisprotsess põhineb kokkulepetel ja kompromissidel.

Standard ei ole üldjuhul kohustuslik, kuid seda saab seaduste ja määruste abil kohustuslikuks muuta. Standardid soovitatakse muuta kõigile kättesaadavaks. Standardimise eelised/eesmärgid on:

- Kulutuste minimeerimine

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

Puudused:

- Lisakulu, -aeg, -bürokratia
- Valesti rakendatuna võib mõnikord mõjuda pidurdavalt innovatsioonile

Standardite liigitus:

- Ametlikud standardid (vastu võetud tunnustatud kehami poolt), näiteks EVS-ISO/IEC 12207
- Tööstus- ehk de facto standardid (ei pruugi olla standardimiskehami poolt ametlikult kinnitatud, kuid on üldlevinud), näiteks operatsioonisüsteemide standardid
- Tehnilised raamstandardid (standardite kogumikud, juhendid nende kasutamiseks), näiteks mitmesugused head tavad
- Riiklikud soovitusel
- Firmasisesed standardid

Standardimiskehamite ja nende allüksuste näiteid:

- ISO - International Organisation for Standardisation, 1947
- IEC - The International Electrotechnical Commission, 1906
- ISO / IEC JTC1 - ISO / IEC ühendatud tehniline komitee (Joint Technical Committee)
- ISO / IEC alla kuulub üle 170 tehnilise komitee (TC), millel on omakorda alamkomiteed (Sub-Committees, SC) ja töögrupid (Working Group, WG)

Firmasisesed standardid peaksid olema lihtsad aru saada ja jälgida, piisavad, lünkadeta, ühesed ja ellu viidud. 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. Võib öelda, et kui standard pole ette valmistatud elluviimiseks, siis on mõttetu tema peale aega kulutada.

4.4.2. Standardimine Eestis

- Eestis on üle võetud rida IT standardeid, mis sisaldavad hinnalist rahvusvahelist kogemust. Ülevaade nende nimetustest on toodud allpool materjalide lisas. Enamus standardeid on tõlgitud. Lisaks juba vaadeldud tarkvara elutsükli protsesside ning tarkvara kvaliteedimudeli standarditele on tõlgitud mitmeid infoturbe alaseid standardeid, dokumentatsiooni koostamise ja halduse suunised jne. Samuti on koostatud infotehnoloogia reeglid eesti keele ja kultuuri keskkonnas, infotehnoloogia sõnastik (ligi 40 osa) ja muid originaalseid materjale.

Kui tahta oma tööd paremaks muuta, võiks igal juhul vaadata olemasolevaid standardeid, et mitte jalgratast leiutada. Eesti IT standardeid saab kasutada:

- tarkvaraga seotud tööprotsesside kujundamiseks, juhtimiseks ja korrastamiseks (12207, 15288)
- kvaliteedihalduseks (9126, 90003)
- infoturbe haldamiseks (13335, 17799, 13569, 27000 seeria)
- eestikeelse tarkvara loomisel (EVS 8)
- IT terminoloogia korrastamiseks (2382 jt)
- dokumenteerimiseks (6592, 9294 jt)

Standardimise korraldus Eestis

- Eesti standardiorganisatsioon (EVS)
- Erialastandardite arendamiseks on sõlmitud koostöölepingud erialaliitude ja asutustega
- Tehnilised komiteed (üle 30): ...EVS/TK 3 Telekommunikatsioonitehnika, EVS/TK 4 Infotehnoloogia...
- Üle 22000 EVS standardi, neist tõlgitud üle tuhande, originaalseid alla kolmesaja

Tehnilise normi ja standardi seadus

- jõustus 1. aprillil 1999
- standard: dokument, kasutamine vabatahtlik
- norm: seaduses, VV/ ministri määruses
- ülevõtmine ka standardiorganisatsiooni ametlikus keeles

EVS TK4 - IT standardimise tehniline komitee

- Asutatud 29.10.97
- Standardiameti juures Riigi Infosüsteemide Arenduskeskuse tehnilisel teenindamisel
- Üle 20 liikme (juriidilised isikud)
- Uued liikmed võrdõiguslikud
- Vaatlejaliige mitmetes ISO allkomiteedes

4.4.3. 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 standardid soodustavad 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)
- ettevõttesised / -välised

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

Materjali lõpus on toodud mitmete olulisemate või Eestis evitatavate standardite nimetused. Tegelikult on standardeid palju rohkem ja neid tuleb pidevalt juurde. Allpool on temaatikad, millele paljud erinevad standardimiskehameid on loonud mitmeid standardeid:

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

Standardite kasutamise võimalusi

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

4.4.4. Tarkvara dokumenteerimine

Tihti küsitakse, kas on olemas ühtsed nõuded tarkvara dokumenteerimiseks, mida saab rakendada igas olukorras - öeldes, et tehke need-ja-need dokumendid ning vormistage need sellisel-ja-sellisel moel? Vastus on ühtaegu "ei" ja "jah".

Ei - ei ole ühtset nimekirja dokumentidest, mida tarkvara arendamisel tuleb koostada. Tarkvara ja selle kasutuskeskkonnad on erinevad. Sellest tulenevalt erineb arendusprotsess, selle tegevused ja igas tegevuses koostatav dokumentatsioon. Ei saa näiteks võrrelda dokumentatsiooni, mida koostatakse väikeettevõtte isikliku klientide andmebaasi programmeerimisel ja (...loodetavasti...) lennuki juhtsüsteemide arendamisel.

Jah - on olemas standardid, praktikad ja soovitused selle kohta, millist tüüpi dokumentatsiooni luua. Nii käivad tarkvara ja muud tüüpi dokumentatsiooni kohta järgmised Eesti standardid:

- EVS-ISO/IEC 6592:2002 Infotehnoloogia. Arvutipõhiste rakendussüsteemide dokumenteerimise suunised
- EVS - ISO/IEC TR 9294:2003 Infotehnoloogia. Tarkvara dokumentatsiooni halduse suunised
- ISO 15489:2001. Dokumendihaldus
- ISO 5127. Informatsioon ja dokumentatsioon. Sõnavara

On olemas ka rida standardeid ja soovitusi selle kohta, kuidas võiks välja näha erinevate tarkvaraprotsesside ja tegevuste dokumentatsioon. Mitmed selles kursuses käsitletavat standardid, eelkõige EVS-ISO/IEC 12207, annavad hea ülevaate vajalikest dokumentidest. Osa käsitletavatid standardeid, nagu ANSI/IEEE Std 830 ja ANSI/IEE Std 829, annavad juhtnöörid konkreetse arendusetapi (spetsifitseerimine, testimine) tulemuste dokumenteerimiseks. Maailmas on loodud standardeid kõikide põhiliste elutsükli tegevuste dokumenteerimise kohta, mitmed neist on üle võetud ka Eesti standardiks.

4.4.5. Kontrollküsimusi ja ülesandeid

- Eesti IT valdkonda puudutavad standardid
- Standardi mõiste, eelised, puudused
- Standardite liigitus
- Standardimiskehamite näited
- Standardimine Eestis: korraldus, IT standardite tehniline komitee
- Tarkvara standardite liigitusi
- Standarditavad valdkonnad
- Standardite kasutamine
- Infoturbe standardid, EVS-ISO/IEC TR 13335- ülevaade
- Dokumenteerimist käsitlevad standardid

4.5. Infosüsteemi audit

Nagu iga keerulise asjaga, on ka tarkvaraga (laiemalt, infosüsteemidega) aeg-ajalt probleeme. Üks võimalus probleemide ennetamiseks, lahendamiseks või vastutusrikaste rakenduste hindamiseks on infosüsteemi audit. Siin kutsub süsteemi omadustest, probleemi ennetamisest või selle lahendamisest huvituv osapool olukorrast ülevaate saamiseks tööle kolmanda isiku - audiitori. Infosüsteemid pole selles suhtes erand, analoogiline on olukord näiteks raamatupidamises, keskkonnakaitses ja mujal.

Anname allpool ülevaate infosüsteemi auditist, selle korraldusest, organisatsioonidest ja ühest olulisemast meetodikast COBIT.

4.5.1. 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 infotehnoloogiaga. 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, meetodikaid, kasutamist, hooldust, mõõtmist, aruandlust, jälgimist (sisemist kvaliteedihaldust).

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 ja organisatsioonist, 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 meetodikat.

Eetikareeglid ütleavad 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 ebasüüdsast 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

4.5.2. Auditi korraldus

Auditi läbiviimine sisaldab tavaliselt 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
- reeglite tegeliku täitmise ulatuse hindamine
- vajadusel sisuline testimine
- hindamine, nt riskide hinnang
- hinnang ja raportid.

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

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.

4.5.3. Organisatsioonid ja sertifitseerimine

Maailmas ühendab infosüsteemide audiitoreid eelkõige Infosüsteemide Auditi ja Kontrolli Assotsiatsioon (*Information Systems Audit and Control Association, ISACA*, vt

<http://www.isaca.org>). Assotsiatsioonil on üle 95000 liikme ning tütarühingud 180-s riigis, sealhulgas Eestis (Eesti infosüsteemide audiitorühing). ISACA sertifitseerib infosüsteemide audiitoreid, avaldab IT auditi alast kirjandust, töötab välja auditi metoodikaid, korraldab koolitusi, algatab uurimis- ja arendustöid, avaldab ajakirja *IS Audit & Control Journal* ning korraldab viiel mandril rahvusvahelisi konverentse.

Üks ISACA olulisemaid tööloike 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 või mitmel kindlal päeval. Eksamil testitakse kandidaadi kogemusi infotehnoloogia auditi, kontrolli ja turbe alal, samuti tema oskusi rakendada erialaseid standardeid ja teadmisi. Selleks tuleb nelja tunni jooksul vastata kahesajale küsimusele. CISA sertifikaat eeldab lisaks eksami sooritamisele eetikanormide ja standardite tunnustamist, töökogemust, pidevkoolitust ja aastamakset. Maailmas on CISA sertifikaadi saanud üle 75000 inimese.

ISACA on praeguseks välja andnud kümneid IS auditi alaseid raamatuid. Raamatute hulgas on kogu auditi ala katvad monograafiad, eriküsimusi käsitlevad raamatud, näiteks UNIX-, COBIT-teemalised väljaanded, sissejuhatavad tekstid, CISA eksami materjalid, videod.

Eesti infosüsteemide audiitorühingusse 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.

4.5.4. COBIT

ISACA poolt välja antud info- ja sellega seotud tehnoloogia kontrolli sihid (COBIT), millest on ilmunud neli versiooni, annavad auditi korralduse üldise metoodika. Viimastel aastatel on COBIT arenenud pigem üldise protsessiraamistiku suunas.

COBITi põhitekstid on saadaval ISACA koduleheküljel. COBIT jaotab juhtimiseesmärgid nelja põhilisse alasse:

- planeerimine ja organiseerimine
- hankimine ja evitus
- tarnimine ja tugi
- seire

Nendel neljal alal on 34 IT protsessi ja laia juhtimiseesmärki. Iga IT protsessi jaoks on antud:

- detailsed juhtimiseesmärgid
- meetodid
- kontrollide hindamine
- vastavuse hindamine
- riskide hindamine

Näiteks COBIT protsessi PO1 - "Määratleda strateegiline IT plaan" jaoks on toodud kaheksa kontrolli sihti, soovitus intervjuuerida kuut isikut, soovitus hankida viit tüüpi dokumentatsioon, seitseteist kontrollküsimust metoodikate ja praktikate kohta, kümme soovitus olemasolevate praktikate testimiseks, muud juhendid.

Toome allpool nelja alasse kuuluvad protsessid. Planeerimine ja organiseerimine:

- PO1. Määratleda strateegiline IT plaan
- PO2. Määratleda infoarhitektuur
- PO3. Määratleda tehnoloogiline suund
- PO4. Määratleda IT organisatsioon ja seosed
- PO5. Hallata IT-investeeringuid
- PO6. Teavitada juhtimisstiilid ja suund
- PO7. Hallata inimressursse
- PO8. Tagada vastavus välisnõuetele
- PO9. Hinnata riskid
- PO10. Hallata projekte
- PO11. Hallata kvaliteeti

Hankimine ja evitamine:

- HE1. Tuvastada lahendused
- HE2. Hankida ja hooldada rakendustarkvara
- HE3. Hankida ja hooldada tehnoloogiaarhitektuur
- HE4. Välja töötada ja käigus hoida IT-protseduurid
- HE5. Installeerida ja akrediteerida süsteemid
- HE6. Hallata muutusi

Tarnimine ja tugi:

- TT1. Määratleda teenusetasemed
- TT2. Hallata kolmandate osapoolte teenuseid
- TT3. Hallata suutvust ja võimsust
- TT4. Tagada pidev teenus
- TT5. Tagada süsteemide turvalisus
- TT6. Tuvastada ja kinnitada kulud
- TT7. Koolitada kasutajaid
- TT8. Abistada ja nõustada IT kliente
- TT9. Hallata konfiguratsiooni
- TT10. Hallata probleeme ja intsidente
- TT11. Hallata andmeid
- TT12. Hallata ruume
- TT13. Hallata ekspluatatsiooni

Seire:

- S1. Seirata protsesse
- S2. Hinnata sisejuhtimise adekvaatsust
- S3. Saada sõltumatu kinnitus
- S4. Korraldada sõltumatu audit

4.5.5. Kontrollküsimumisi ja ülesandeid

- IS audit, selle eesmärgid, auditeeritavad objektid, maht, audiitor, riskid
- Planeerimine, korraldus

- Organisatsioonid, sertifitseerimine
- COBIT

5. Ülevaateid standarditest

5.1. Standardist "ANSI/IEE Std 829 Standard for Software Test Documentation"

Järgnevates jaotistes antakse lühiülevaade veel mõnedest standarditest, mis võivad olla kasulikud tarkvara arenduses ning mida saab kasutada ka iseseisvates ja ainetöodes.

Standard ANSI/IEE Std 829 annab struktuuri testimise dokumenteerimiseks ning kogemuse tööks rahvusvahelise standardiga, mis võib edasises kasuks tulla. Selle erinevaid komponente võib valikuliselt kasutada ka iseseisvates töodes. Eelkõige on seda standardit soovitatav kasutada neil, kellel on suuremate projektide arenduskogemus. Kui arendustöö tagapõhja ei ole, võivad standardi mitmed osad tunduda ebavajalikud, võib ka olla raske välja valida, mida siit tegelikult tasub kasutusele võtta. Mõningaid korduma kippuvaid küsimusi:

- Mõned asjad korduvad erinevates dokumentides? - Nendel dokumentidel on erinevad kasutajad
- Kas on liiga palju dokumente? - Kõiki ei pruugi lihtsamal testimisel kasutada
- Ma pean kirjeldama asju, mida ma niikuinii tean? - Jah, aga tegelikkuses loevad standarditega määratud dokumente mitmesugused grupid (nt., kasutajad, ülemused jne.), kes neid ei tea. Ka endal lähevad ajapikku asjad meelest ära
- Mõned dokumendid või nende osad tunduvad liigsed? - Suure projekti puhul võib sellisest dokumentatsioonist sõltuda süsteemi vastuvõtmine, tellija-täitja vahelised maksed ja töötajate palgad, inimeste ja firma käekäik. Kui vaadata asja sellisest vaatenurgast, on igal dokumendil oma ülesanne, loogika ja sihtrühm
- Liiga palju paberit (ainetöös)? - Väikese projekti jaoks küll. Suurt projekti ei saa jälle kursuse raames ette võtta. Idee pole mitte selles, et nii alati testida, vaid et anda mingi eeskuju ja kogemus testide dokumenteerimisest ja rahvusvahelise standardi järgi töötamisest. Sellised standardid annavad lähtepunkti, millest võib vastavalt vajadustele arendada oma standardeid

Järgnevas toome selle standardi struktuuri. Kirjeldatud dokumentidel on erinev kaal. Kõige tähtsamaks osutub tavaliselt viimane dokument, mis annab erinevatele kasutajagruppidele ülevaate testimise tulemustest ja näitab ka testijate töö kvaliteeti.

Standard kirjeldab järgmisi dokumente:

1. * TEST PLAN - testimise plaan
2. * TEST-DESIGN SPECIFICATION - testimise projekt. Kuidas? Millised testid?
3. * TEST-CASE SPECIFICATION - testi kirjeldus
4. TEST-PROCEDURE SPECIFICATION - testimise protseduuri spetsifikatsioon
5. * TEST-ITEM TRANSMITTAL REPORT - testitavate objektide üleandmise aruanne
6. TEST LOG - testimise käik (kuidas testimine toimus, kes tegi jne)

7. * TEST-INCIDENT REPORT - testprobleemi aruanne

8. * TEST-SUMMARY REPORT - testimise kokkuvõte

Vaatleme tärniga tähistatud dokumente. Inglisekeelsed pealkirjade nimetuste tõlked ei pruugi olla ainuõiged - kui tunnete, et mingi muu tõlge on oluliselt parem, kasutage seda.

Testimise plaan (Test plan)

Plaan annab testimise põhjuse, objektid, nõuded (ja mitte-nõuded), meetodite lühikirjelduse, läbimise kriteeriumid ja projektijuhtimise nõuded (ressursid, ajakava jne.).

1. Testimise plaani identifikaator (Test-plan identifier)

Identifikaator on igal dokumendil (ideaalselt igal objektil - nt., ka programmil ja selle versioonil). See peab kajastama olukorda lühidalt. Selle järgi on objekti hea ära tunda, ta on dokumendi märgend. Näiteks: RMTP-TEST-PLAN-TEST-1,TEST-2 jne.

2 Sissejuhatus (Introduction)

Sissejuhatuses selgitatakse probleemi - kes algatas, kelle jaoks, mida testitakse. See osa esitab olukorra, mis tingib testimise (ja edasi nõuded ning ka meetodid). Kui sellist olukorda pole (esitatud), siis puudub vajadus testimiseks ning pole põhjendatud ka testimise nõuded. Kui reaalselt olukorda pole, võib selle iseseisvas töös välja pakkuda.

3. Testitavad objektid (Test items)

Ülevaade testitavatest objektidest. Mida testitakse. Viited dokumentidele, kust võetakse testimise andmed.

4. Testitavad omadused (Features to be tested)

- Omadused, mida testitakse. Omadusi sellesse ja järgmisse punkti võib võtta ülaltoodud ISO/IEC 9126 kvaliteedifaktorite nimestikust.

5. Omadused, mida ei testita (Features not to be tested)

Omadused mida ei testita.

6. Meetod (Approach)

Testimise, kontrolli meetod. Milliseid objekte ja missuguse meetodiga testime. Käesolevas kursuses käsitletavatest meetoditest ja teemadest saab iseseisvates töödes kasutada näiteks järgmisi (*-ga on märgitud meetodid, mille kasutamisel iseseisvates töödes on soovitatav konsulteerida õppejõuga):

- Programmipõhine testimine: lause-, haru-, elementaartingimuste-, teadekvaatsuse kriteeriumid, silmuste testimine, andmepõhine testimine,
- Funktsionaalne testimine: ekvivalentsiklassid, piirjuhud, vea otsing*,
- Muud testimise meetodid: lisatud vead*, juhuslikud testid*,
- Testimist ja kvaliteeti toetav tarkvara
- Staatilised meetodid: küsimustikud*, struktuursed läbivaatused, programmeerija hindamine*, tõestamine*,

- Kontrolli korraldus: suurem testimise projekt* (integratsioonitestid, valideerimistestid, süsteemitestid), regressioonitestimise planeerimine ja korraldamine*,
- Meetrikate programmi planeerimine*
- ISO 9000 standardite programmi planeerimine/ettevalmistamine*

7. Testi läbimise kriteerium (Item pass/fail criteria)

Kriteeriumid mille puhul testimine on läbitud ja toode vastu võetud.

8. Testimise katkestamise ja jällealustamise tingimused (Suspension criteria and resumption requirements)

Märgitakse ära olukorrad, mille puhul pole mõtet testi jätkata ning tingimused, mille puhul .testimist peale katkestamist uuesti alustatakse.

9. Üleantavad materjalid (Test deliverables)

Testimise tulemused. Näidatakse ära materjalid, mis antakse üle testimise lõppedes.

10. Testimise ülesanded (Testing tasks)

Testimise ülesanded. Mida vaja teha, et testimise juurde asuda. (Ettevalmistused, kooskõlastused, koolitus...)

11. Nõuded testimise ümbrusele (Environmental needs)

Nõuded testimise ümbrusele, näiteks: riistvara-, operatsioonisüsteemi-, kommunikatsioonide nõuded. Võib-olla tuleb testimiseks tekitada fiktiivseid andmebaase, kasutajaid? Kui testitakse seadmeid, võib olla vaja neid seadmeid muretseda, installeerida jne.

12. Vastutused (Responsibilities)

Testimise vastutused. Näidatakse, kes millise töö osa eest vastutab. Näiteks võivad arendajad olla vastutavad vigade eest (vähemasti spetsifikatsioonile mittevastavuste puhul), Tellija keskkonna loomise eest (nt. b-testimisel).

13. Personali ja koolituse vajadus (Staffing and training needs)

Inimeste ja koolituste vajadus.

14. Ajakava (Schedule)

Ajakava. Mis millal toimuma hakkab ja millal valmis saab.

15. Riskid ja ootamatused (Risks and contingencies)

Riskid ja ootamatused. See puudutab testimist ennast. Mis juhtub siis, kui test mingil põhjusel nurjub. Antakse hinnang projekti edukuse kohta.

16. Allkirjad (Approvals)

Osapoolte allkirjad ja kooskõlastused.

Testimise projekt (Test-Design Specification)

1. Testimise projekti identifikaator (Test-design-spetsification identifier)

Vt. ülal

2. Testitavad omadused (Features to be tested.)

Pannakse põhjalikult kirja, mida testitakse (kogu dokument peab olema enam-vähem iseseisvalt), näiteks kvaliteedi kriteeriumid ja alamkriteeriumid.

3. Meetodite täpsustus (Approach refinements)

Meetodite täpsustus. Millised testid, milliste meetoditega tehakse.

4. Testide identifikaatorid (Test identification)

Standardis on siin ette nähtud testide identifikaatorid koos lühikirjeldustega - testide detailed kirjeldused on eraldi dokumendis. Iseseisvas töös tavaliselt sellist eraldi dokumenti pole, seepärast kirjeldatakse siin ülevaاتlikult kogu testi: pannakse kirja testi idee, sisend ja väljund ning oodatud tulemus. Siin peaks olema ka põhjendus selle kohta, kuidas valitud testimise meetoditest testid tulenesid.

5. Omaduste vastuvõtmise ja tagasilükkamise kriteeriumid (Features pass/fail criteria)

Omaduste kaupa kirja pandud läbimise tingimused. Et kogu objekt oleks vastu võetud, peaksid selles punktis kirjeldatud kriteeriumid olema rahuldatud.

Testi kirjeldus (Test-Case Specification)

Neid kirjeldusi võib olla üks iga testi kohta.

1. Testi identifikaator (Test-case-specification identifier)

Vt. ülal

2. Testitavad objektid (Test items)

Milliseid objekte testitakse.

3. Sisendid (Input specifications)

Testiks vaja minevaid sisendandmeid

4. Väljundid (Output specifications)

Oodatavad väljundandmed

5. Nõuded ümbrusele (Environmental needs)

Vajadused ümbrusele. Tarkvara, riistvara, andmed jne.

6. Eriprotseduuride nõuded (Special procedural requirements)

7. Testide vahelised seosed (Intercase dependencies)

Mida millises järjekorras teha.

Testitavate objektide üleandmise aruanne (Test-Item Transmittal Report)

Siia pannakse kirja testitavate objektide üleandmisega seotu, et oleks selge, millises versioonis probleem leiti.

1. Üleandmise identifikaator (Transmittal-report identifier)

Vt. ülal

2. Üleantavad objektid (Transmitted items)

Üleantava objekti kirjeldus.

3. Objektide asukoht (Location)

Üleantavate objektide asukoht

4. Staatus (Status)

Lähteolukord, millises olukorras on objekt.

5. Allkirjad (Approvals)

Allkirjad.

Testprobleemi aruanne (Test-Incident Report)

Neid kirjeldusi võib olla üks iga testimisel leitud probleemi kohta.

1. Testprobleemi aruande identifikaator (Test-incident-report identifier)

Vt. ülal

2. Kokkuvõte (Summary)

Kokkuvõte problemist.

3. Probleemi kirjeldus (Incident description)

Inputs - sisend

Expected Results - oodatud tulemus

Actual Results - tegelik tulemus

Anomalies - kõrvalekalded

Date and Time - kuupäev ja kellaaeg

Procedure Step - protseduuri samm (kui testimiseks oli eraldi protseduur)

Environment - millises ümbruses tekkis

Attempts to Repeat - millised katsed võeti ette selle vea kordamiseks

Testers - kes olid testijad

Observers - kes olid tunnistajad

4. Raskusaste (Impact)

Vea mõju ja raskuse hinnang

Testimise kokkuvõte (Test-Summary Report)

Võib näida, et see ja eelmised dokumendid on dubleerivad. Mingil määral jah, kuid sellel on oma põhjus - nad on kirjutatud erinevatele adressaatidele ja on erineva detailsuse astmega. Testimise kokkuvõte on määratud (ka) juhtkonnale või investoritele, kes peaks siit kiiresti aru saama, millised olid projekti eesmärgid, kuidas kogu projekt toimus ja millised olid tulemused, sealhulgas hinnang testimise objektile.

1. Testimise kokkuvõtte identifikaator (Test-summary report identifier)

Vt. ülal

2. Kokkuvõte (Summary)

Kokkuvõte kogu testimise käigust ja tulemustest. See on ka antud dokumendi kokkuvõte, mida järgmised osad täpsustavad.

3. Kõrvalekaldumised (Variances)

Kõrvalekaldumised, mida leiti (ülevaade, laskumata detailidesse)

4. Põhjalikkuse hinnang (Comprehensiveness assessment)

Kui põhjalikult testiti, milliseid osi testiti, mis jäi testimata jne

5. Tulemuste kokkuvõte (Summary of results)

Leitud vigade ülevaade. Millised testid tehti.

6. Hinnang (Evaluation)

Hinnang testitavatele objektidele

7. Tegevuste kokkuvõte (Summary of activities)

Mida sai tehtud. Millised tööd tehti, kui palju aega kulutati.

8. Allkirjad (Approvals)

5.2. Standardist "IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications"

Ka seda standardit võib kasutada nii igapäevases töös kui ka iseseisvates töödes. Standardi põhjal võiks nõudmiste spetsifikatsioon olla näiteks järgmise struktuuri ja loogikaga.

Sisukord

1. Sissejuhatus

1.1. Otstarve (+ülesande püstitus ja kasutusvaldkond)

1.2. Ulatus

- nimi/tüüp
- mida teeb/ei tee

1.3. Definitsioonid ja kasutatavad lühendid

1.4. Viited kasutatud materjalidele

1.5. Spetsifikatsiooni ülevaade (sisu/struktuur)

2. Üldkirjeldus

2.1. Kontekst ja perspektiiv

- iseseisev/integreeritud?
- seos hõlmava süsteemi struktuuriga
- liidesed hõlmava süsteemiga
- olemasolev IT keskkond (kui vaja)

2.2. Funktsioonide ülevaade, näiteks: vastamine mingit tüüpi päringule, andmete või teadmiste uuendamine, tellimuse vormistamine, laoseisu kontroll jne.

2.3. Huvipooled ja kasutajad

2.4. Üldised kitsendused ja nõuded (on soovitatav kasutada ISO/IEC 9126 nõudmisi)

2.5. Eeldused/sõltuvused

3. Erinõuded

3.1. Funktsioonid {kirjeldatakse eraldi põhjalikult igat p. 2.2 nimetatud funktsiooni}

3.1.1. Funktsioon 1: Sissejuhatus, sisend, töötlus, väljund

.....

3.1.n. Funktsioon n: Sissejuhatus, sisend, töötlus, väljund

3.2. Välisliidesed (detailselt)

- kasutajad, riistvara, tarkvara, kommunikatsioon

3.3. Tööjõudluse nõuded

3.4. Kavandamise kitsendused

- standardid, riistvara,

3.5. Muud nõuded ja omadused

- (on soovitatav kasutada ISO/IEC 9126 nõudmiste nimekirja)
- andmebaas, operatsioonid, ülekanne, ja nii edasi

5.3. Standardist EVS - ISO/IEC 9126-1:2003 Tarkvaratehnika. Toote kvaliteet. Osa1: Kvaliteedimudel

Sise- ja väliskvaliteedi näitajate (alla joonitud) ja allnäitajate täpsemad määratlused on järgmised.

Funktsionaalsus. Tarkvaratoote suutlikkus pakkuda funktsioone, mis rahuldavad deklareeritud ja eeldatavaid vajadusi, kui seda tarkvara kasutatakse spetsifitseeritud tingimustes.

Sobivus. Tarkvaratoote suutlikkus pakkuda spetsifitseeritud ülesannete ja kasutaja eesmärkide jaoks sobivat funktsioonide kogumit.

Õigsus. Tarkvaratoote suutlikkus anda õigeid või kokkulepitud tulemeid või toimeid nõutava täpsusastmega.

Koostalitlusvõime. Tarkvara suutlikkus interakteerida ühe või mitme spetsifitseeritud süsteemiga.

Turvalisus. Tarkvaratoote suutlikkus kaitsta informatsiooni ja andmeid, nii et volitamata isikud või süsteemid ei saa neid lugeda ega muuta ning et volitatud isikute või süsteemide juurdepääsu neile ei tõkestata.

Funktsionaalsuse vastavus. Tarkvaratoote suutlikkus järgida funktsionaalsust käsitlevaid standardeid, kokkuleppeid või eeskirju seadustes ja muudes selletaolistes ettekirjutustes.

Töökindlus. Tarkvaratoote suutlikkus säilitada spetsifitseeritud sooritusvõime taset, kui teda kasutatakse spetsifitseeritud tingimustel.

Küpsus. Tarkvaratoote suutlikkus vältida tarkvara defektidest tulenevaid tõrkeid.

Tõrketaluvus. Tarkvaratoote suutlikkus säilitada spetsifitseeritud sooritusvõimetaset tarkvara defektide puhul või tootele spetsifitseeritud liidese rikkumise puhul.

Taastuvus. Tarkvaratoote suutlikkus tõrke korral ennistada spetsifitseeritud sooritusvõimetase ja taastada andmed, mida tõrge otseselt mõjutas. **MÄRKUS 2. Käideldavus** on tarkvaratoote suutlikkus olla seisundis, kus ta saab antud ajahetkel täita deklareeritud kasutamistingimustel nõutavat funktsiooni. Väliselt saab käideldavust hinnata suhtelise ajaosaga, mille kestel tarkvaratoode on talitlusvõimeline. Käideldavus on seega kombinatsioon küpsusest (millest sõltub tõrgete sagedus), tõrketaluvusest ja taastuvusest (millest sõltub igale tõrkele järgnev rikkeaeg). Seetõttu pole käideldavust eraldi allnäitajana sisse võetud.

Töökindluse vastavus. Tarkvaratoote suutlikkus järgida töökindlust käsitlevaid standardeid, kokkuleppeid või eeskirju.

Kasutuskõlblikkus. Tarkvaratoote suutlikkus olla spetsifitseeritud tingimustel kasutamisel kasutajale arusaadav, õpitav, kasutatav ja meeldiv. **MÄRKUS 1.** Kasutuskõlblikkust mõjutavad ka mõned funktsionaalsuse, töökindluse ja tõhususe aspektid, kuid ISO/IEC 9126 otstarbeks ei ole neid liigitatud kasutuskõlblikkuseks.

Arusaadavus. Tarkvaratoote suutlikkus võimaldada kasutajal aru saada, kas see tarkvara on sobiv ja kuidas teda saab kasutada konkreetsete ülesannete ja kasutamistingimuste puhul.

Õpitavus. Tarkvaratoote suutlikkus võimaldada kasutajal õppida ta rakendamist.

Käsitsetavus. Tarkvaratoote suutlikkus võimaldada kasutajal teda käsitseda ja juhtida.

Meeldivus Tarkvaratoote suutlikkus meeldida ta kasutajale.

Kasutuskõlblikkuse vastavus. Tarkvaratoote suutlikkus järgida kasutuskõlblikkust käsitlevaid standardeid, kokkuleppeid, stiilisunniseid või eeskirju.

Tõhusus. Tarkvaratoote suutlikkus deklareeritud tingimustel pakkuda kasutatavate ressursside suhtes asjakohast sooritusvõimet.

Ajaline käitumine. Tarkvara suutlikkus deklareeritud tingimustel oma funktsiooni täitmisel pakkuda asjakohaseid reaktsiooni- ja töötluaegu ning jõudlusi.

Ressursikasutus. Tarkvaratoote suutlikkus deklareeritud tingimustel oma funktsiooni täitmisel kasutada koguselt ja tüübilt asjakohaseid ressursse.

Tõhususe vastavus. Tarkvaratoote suutlikkus järgida tõhusust käsitlevaid standardeid või kokkuleppeid.

Hooldatavus. Tarkvara suutlikkus olla modifitseeritav. Modifikatsioonid võivad sisaldada parandusi, täiustusi või tarkvara sobitamist keskkonna, nõuete ja funktsionaalspetsifikatsioonide muutustega.

Analüüsitavus. Tarkvaratoote suutlikkus olla diagnoositav tarkvara puuduste või tõrkepõhjuste tuvastamiseks või modifitseerimisele kuuluvate osade leidmiseks.

Muudetavus. Tarkvaratoote suutlikkus võimaldada spetsifitseeritud modifikatsiooni teostamist.

Stabiilsus. Tarkvaratoote suutlikkus vältida tarkvara modifikatsioonidest tulenevaid ootamatuid ilminguid.

Testitavus. Tarkvaratoote suutlikkus võimaldada valideerida modifitseeritud tarkvara.

Hooldatavuse vastavus. Tarkvaratoote suutlikkus järgida hooldatavust käsitlevaid standardeid või kokkuleppeid.

Porditavus. Tarkvaratoote suutlikkus olla üle viidud ühest keskkonnast teise.

Sobitatus. Tarkvaratoote suutlikkus olla sobitatud mitmesuguste spetsifitseeritud keskkondadega, ilma et tuleks rakendada muid kui selle tarkvara jaoks selleks otstarbeks määratud toiminguid või vahendeid.

Installeeritavus. Tarkvaratoote suutlikkus olla installeeritud spetsifitseeritud keskkonda.

Koosoluvõime. Tarkvaratoote suutlikkus eksisteerida koos muu sõltumatu tarkvaraga ühises keskkonnas, kasutades koos ühisressursse.

Vahetatavus. Tarkvaratoote suutlikkus olla kasutatav teise spetsifitseeritud tarkvaratoote asemel samaks otstarbeks ja samas keskkonnas.

Porditavuse vastavus. Tarkvaratoote suutlikkus järgida porditavust käsitlevaid standardeid või kokkuleppeid.

Kasutuskvaliteedi näitajad on järgmised.

- **Toimivus.** Tarkvaratoote suutlikkus võimaldada kasutajail saavutada spetsifitseeritud eesmärged spetsifitseeritud kasutuskontekstis õigesti ja täielikult.
- **Tööviljakus.** Tarkvaratoote suutlikkus võimaldada kasutajail kulutada spetsifitseeritud kasutuskontekstis saavutatava toimivuse suhtes asjakohaseid ressursikoguseid.
- **Ohutus.** Tarkvaratoote suutlikkus saavutada spetsifitseeritud kasutuskontekstis vastuvõetavaid inimeste, äritegevuse, tarkvara, omandi või keskkonna kahjustamise riski tasemeid.
- **Rahuldus.** Tarkvaratoote suutlikkus rahuldada kasutajaid spetsifitseeritud kasutuskontekstis.

6. Lisad

6.1. Materjale

Osa materjale on viidatud tekstis vastavate teemade juures.

- Käesolev materjal
- W. Perry. Effective Methods of Software Testing. John Wiley & Sons, Incorporated
- ANSI/IEEE Std 829 Standard of software test documentation
- ISO/IEC (9126). ISO/IEC 9126, Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their Use
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE, <http://www.computer.org/portal/web/swebok>
- Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, ACM / IEEE, <http://www.acm.org/education/curricula-recommendations>
- Certified Tester Foundation Level Syllabus. International Software Testing Qualifications Board, <http://www.istqb.org/download.htm>. +Other materials from this site.
- Books including software quality, testing, and standards issues as significant portions of the text, edition after 2000 preferred. Examples: I. Sommerville, Software Engineering, Addison-Wesley; R.S. Pressman, Software Engineering, McGraw Hill; G. O'Reagan, A Practical Approach to Software Quality, Springer
- CMM - <http://www.sei.cmu.edu/cmm/cmms/cmms.html>
- Osade teemade puhul saab kasutada ka väljaannet: Tepandi, J. Tarkvara kvaliteet ja standardid. TTÜ kirjastus, 1999

Eesti IT standardeid

- EVS8:2000 Infotehnoloogia reeglid eesti keele ja kultuuri keskkonnas
- EVS-ISO/IEC 2382 Infotehnoloogia. Sõnastik (ligi 40 osa)
- EVS-ISO/IEC 12207 Infotehnoloogia. Tarkvara elutsükli protsessid
- ISO/IEC TR 15271:1999 Infotehnoloogia. ISO/IEC 12207 (Tarkvara elutsükli protsessid) juhend
- EVS - ISO/IEC TR 15504-9:2003 Infotehnoloogia. Tarkvaraprotsesside hindamine. Osa 9: Sõnastik
- EVS - ISO/IEC 9126-1:2003 Tarkvaratehnika. Toote kvaliteet. Osa1: Kvaliteedimudel
- 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 TR 9294:2003 Infotehnoloogia. Tarkvara dokumentatsiooni halduse suunised
- EVS-ISO/IEC 6592:2002 Infotehnoloogia. Arvutipõhiste rakendussüsteemide dokumenteerimise suunised
- Jõustamisteate meetodil ülevõetud Euroopa standardid
- Teised infotehnoloogia erialal kasulikud standardid, näiteks ISO 9000 seeria. ISO/IEC 90003, Software engineering — Guidelines for the application of ISO 9001:2000 to computer software.

Täiendavad materjalid

- ANSI/IEEE Std 1028. IEEE Standard for Software Reviews and Audits
- Bach J. Test Automation Snake Oil, v2.1. http://www.satisfice.com/articles/test_automation_snake_oil.pdf
- Boehm, B. ja teised. Software Cost Estimation with COCOMO II. Prentice-Hall, 2000
- Certified Information Systems Auditor Review Manual. Information Systems Audit and Control Association, Rolling Meadows, USA
- COBIT. Governance, Control and Audit for Information and Related Technology. Information Systems Audit and Control Foundation, Rolling Meadows, USA - <http://www.isaca.org>
- Costar - <http://www.softstarsystems.com/>
- Eesti kvaliteediauhind - www.eaq.ee
- EISAÜ. Infosüsteemide audiitorkontrolli eeskirjad. Eesti infosüsteemide audiitorühing. Tallinn, www.eisay.ee
- Extreme Programming ja testimine - <http://www.extremeprogramming.org>, www.xprogramming.com, <http://c2.com/cgi/wiki?CodeUnitTestFirst>
- Garmus, David, David Herron. Function Point Analysis: Measurement Practices for Successful Software Projects. Addison-Wesley Information Technology Series, 2000
- IEEE Std 829. IEEE Standard for Software Test Documentation
- IEEE Std 830. IEEE Recommended Practice for Software Requirements Specifications
- ISKE materjalid - <http://www.ria.ee/ISKE>
- ISO 5807. Information processing -- Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts
- ISO 8402. Quality Assurance - Vocabulary
- ISO/IEC TR 15504. Information technology — Software process assessment
- Koomen, T., and Pol, M. Test process improvement: a practical step-by-step guide to structured testing. Addison Wesley, 1999
- Pressman, R. Software Engineering: a Practitioners Approach, 5th edition. 2001.

- Rational Unified Process (RUP) ja muud meetodid ja vahendid - <http://www.rational.com/>
- SPICE - <http://www.sqi.gu.edu.au/SPICE/>
- Tepandi, J. Mida võtta aluseks infotehnoloogia auditeerimisel? A&A, 4, 2007, lk 19-22
- Test Process Improvement, TPI - <http://www.sogeti.nl/index.html?iospagina.cfm?uNr=150>
- Testimise automatiseerimine - vt lingid peatükis 3.4
- Testing: The RUP Philosophy. Paul Szymkowiak, Philippe Kruchten. Rational Software, 2003
- <http://tester.com.ua/> - vene- ja ingliskeelsed materjalid testimise kohta

Lisamaterjalid

Lisamaterjali saab (TTÜ) raamatukogust, WWW-st ja mujalt. 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, samuti vaadata erialaseid uudistegruppe. Näiteid:

- standardid - <http://www.ria.ee>
- testimine - <http://www.io.com/~wazmo/qa/> (palju viiteid), <http://www.soft.com/>
- kvaliteedihaldus - <http://www.soft.com/>, <http://deming.eng.clemson.edu/> (viiteid teistele kvaliteedihalduse allikatele)
- näide uudistegrupist - comp.software.testing

Ajakirjad ja konverentsid - on erialaajakirju (*Communications of the ACM, IEEE Software*), artikleid avaldatakse perioodiliselt ka muudes ajakirjades. On ka sellele teemale pühendatud üldisi konverentse, erialateemadele pühendatud konverentse, sektsioone muudel konverentsidel.

6.2. Sõnastik

Toome olulisemate kasutatud terminite jaoks ingliskeelsed vasted ja sisuseletused, põhinedes rahvusvahelistele ja Eesti terminoloogiastandarditele. Erinevates standardites on ingliskeelseid mõisteid mõnevõrra erinevalt eesti keelde tõlgitud, paralleelsete tõlgete puhul on allpool eelistatud IT standardites toodud vasteid. Sisuseletuse lõpus on võimalusel sulgudes viide standardile ja selle märksõnale, näiteks (20.01.01) viitab standardi ISO/IEC 2382-20 märksõnale 20.01.01 (süsteemiarendus).

Audit - audit. Volitatud isiku sooritatav revisjon tarkvaratoodete ja protsesside sõltumatuks hindamiseks nende nõuetekohasuse otsustamise eesmärgil (EVS-ISO/IEC 12207)

Eksitus, inimviga, viga (selles tähenduses mittesoovitav) - mistake, human error, error (deprecated in this sense). Inimese selline toiming või toimimatus, mis võib tekitada ettekatsetamatu tagajärje (14.01.09)

Elutsükli mudel - Life cycle model. Tarkvaratoote väljatöötamise, ekspluatatsiooni ja hooldusega seotud protsesse, tegevusi ja töid sisaldav raamstruktuur, mis hõlmab süsteemi eluiga alates ta nõuete määratlusest kuni ta kasutamise lõpetamiseni (EVS-ISO/IEC 12207)

Integratsioonitest - integration test. Programmide või moodulite järkjärguline linkimine ja testimine nende laitmatu talitluse tagamiseks terviksüsteemis (20.05.06)

Katseprojekt, pilootprojekt - pilot project. Projekt, mis on mõeldud infotöötlussüsteemi esialgse variandi katsetamiseks tegelikes, kuid piiratud talitlustingimustes ning mida seejärel kasutatakse süsteemi lõpliku variandi katsetamiseks (20.01.07)

Kontroll. Seda mõistet kasutatakse kursuses fraasi "verifitseerimine ja valideerimine" sünonüümina

Kvaliteedi tagamine - quality assurance, QA (lühend). Kavandatud süstemaatilised toimingud, mis on vajalikud, et tagada komponendi või süsteemi vastavust kehtestatud tehnilistele nõuetele (20.05.01)

Kvaliteedijuhtimine - quality management. Koordineeritud tegevused organisatsiooni kvaliteediga seonduvaks suunamiseks ja ohjeks (EVS-EN ISO9001:2001). Märkus: Eesti IT standardites on "quality management" tõlgitud kui "kvaliteedihaldus", sama vastet on kasutatud ka käesolevas materjalis

Kvaliteedihaldus - vt. kvaliteedijuhtimine

Kvaliteedipoliitika - quality policy. Organisatsiooni üldised kvaliteediga seonduvad taotlused ja suunad, esitatuna ametlikult tippjuhtkonna poolt (EVS-EN ISO9001:2001)

Kvaliteedijuhtimissüsteem (kvaliteedisüsteem) - quality management system. Juhtimissüsteem organisatsiooni kvaliteediga seonduvaks suunamiseks ja ohjeks (EVS-EN ISO9001:2001)

Kvaliteet - quality. Määr, milleni olemuslike karakteristikute kogum täidab nõuded (EVS-EN ISO9001:2001). Märkus: see määratlus muutub arusaadavamaks koos teiste EVS-EN ISO9001:2001 mõistete seletustega, samas ei ole kõiki neid siinkohal otstarbekas ära tuua. Materjali tekstis on toodud kvaliteedi mõistete lihtsamaid seletusi, näiteks: "Kvaliteet on vastavus nõuetele" või "Kvaliteet on mõiste, mis seob toote, nõuded tootele ja tootmise protsessi"

Läbivaatus - walkthrough. Eri vormide kohta käiv terminoloogia pole fikseeritud. Üks võimalus on tõlkida standardi *ANSI/IEEE Std 1028-1988 IEEE Standard for Software Reviews and Audits* mõisteid järgmiselt: juhtkonnapoolne hindamine (*management review*), tehniline hindamine (*technical review*), tarkvara inspeksioon (*software inspection*), läbivaatus (*walkthrough*). *Joint review* võiks olla ühisläbivaatus või ühine hindamine

Nõue - requirement. Oluline tingimus, mida süsteem peab rahuldama (20.01.02)

Prototüüp - prototype. Süsteemi projekteerimise, töönäitajate ja kasutuspotentsiaali hindamiseks või nõuete paremaks mõistmiseks või määramiseks sobiv mudel või esialgne teostus (20.01.08)

Rakendustarkvara [rakendusprogramm] - application software [program]. Tarkvara [programm], mis on spetsiifiline mingi rakendusprobleemi lahenduse mõttes (20.01.15)

Rike, defekt - fault. Ebanormaalne olukord, mis võib põhjustada *funktsionaalüksusel* nõutava funktsiooni täitmise võime kahanemise või kadumise (14.01.10)

Sertifitseerimine – certification. Kolmanda osapoole tegevus, mis hindab toote, teenuse või protsessi vastavust standarditele ja normdokumentidele ja väljastab vastavuse korral (vastavus)sertifikaadi

Silumine – debugging. Leitud vea kõrvaldamine

Spetsifikatsioon - specification. Dokumendi vormis detailne formuleering, mis annab süsteemi väljatöötamiseks ja vastuvõtukatsetusteks süsteemi määratleva kirjelduse (20.01.03)

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

Süsteemi elutsükl - system life cycle. Süsteemi arenduslike muutuste kulg süsteemi algatamisest ta kasutamise lõpuni (20.01.05)

Süsteemi hooldus - system maintenance. Süsteemi modifitseerimine defektide kõrvaldamiseks, jõudluse tõstmiseks või süsteemi sobitamiseks muutunud keskkonna või muutunud nõuetega (20.05.09)

Süsteemi teostus, teostus- implementation (of a system). Süsteemiarenduse faas, mille lõpul süsteemi riistvara, tarkvara ja protseduurid viiakse töökorra (20.04.01)

Süsteemiarendus - system development. Protsess, mis harilikult hõlmab nõuete analüüsi, süsteemi projekteerimist, teostamist, dokumenteerimist ja kvaliteedi tagamist. (20.01.01)

Süsteemiintegratsioon, integratsioon - (system) integration. Tervikliku süsteemi järkjärguline koostamine komponentidest (20.04.02)

Süsteemitarkvara - system software. Rakendusest sõltumatu tarkvara, mis toetab rakendustarkvara käitust (20.01.14)

Tarkvara - software. Infotöötlussüsteemi kõik programmid, protseduurid, reeglid ja nendega seotud dokumentatsioon või osa neist (01.01.08). Laiemas mõttes kasutatud kursuses ka infosüsteemi tähenduses

Tarkvarakomponendi test, komponenditest - unit test. Üksiku programmi või mooduli test, mille otstarve on tagada analüüsi- ja programmeerimisvigade puudumine (20.05.05)

Tarkvarapakett - software package. Täielik ja dokumenteeritud programmide komplekt, mis tarnitakse mitmele kasutajale üldise rakenduse või funktsiooni tarbeks. MÄRKUS: Mõned tarkvarapaketid on kohandatavad eri rakenduste jaoks (20.01.16)

Test, testjuht, testimine – test, test case, testing. Vt jaotis 3.1.1.

Tõrge - failure. *Funktsionaalüksuse* nõutava funktsiooni täitmise võime lakkamine (14.01.11)

Valideerimine – validation. Tegevus, mille otstarve on välja selgitada, kas teostatud süsteem vastab spetsifitseeritud nõuetele (tegevus, mis püüab näidata, et tehtud on seda, mis vaja)

Valiidsustest - validation (test). Test, mille otstarve on välja selgitada, kas teostatud süsteem vastab spetsifitseeritud nõuetele (20.05.04)

Vastuvõtutest - acceptance test. Süsteemi või funktsionaalüksuse test, mille harilikult sooritab tellija oma asukohas pärast installeerimist tarnija osavõtul, et tagada lepingunõuetele vastavust (20.05.07)

Verifitseerimine – verification. Tegevus, mis püüab näidata, et järgmise etapi tulemus vastab eelnenud etapi määratlusele

Verifitseerimistest - verification (test). Süsteemi test, mille otstarve on tõendada, et süsteem vastab vaadeldavas arendusjärgus kõigile spetsifitseeritud nõuetele (20.05.03)

Viga - error. Arvutatud, vaadeldud või mõõdetud väärtuse või oleku ning tegeliku, spetsifitseeritud või teoreetiliselt õige väärtuse või oleku lahknevus. (14.01.08)

6.3. Kasutatud lühendeid

API - Application Programming Interface

CASE – Computer-Aided Software Engineering

CISA – Certified Information Systems Auditor

CMMI – Capability Maturity Model® Integration

COBIT - Governance, Control and Audit for Information and Related Technology

EISAÜ - Eesti Infosüsteemide Audiitorite Ühing

EVS - Eesti Vabariigi Standardiorganisatsioon

EVS TK4 - Eesti Infotehnoloogia standardimise tehniline komitee

GUI - Graphical User Interface

IEC - The International Electrotechnical Commission

IEEE - The Institute of Electrical and Electronics Engineers

ISACA - Information Systems Audit and Control Association

ISKE – Infosüsteemide kolmeastmelise etalontube süsteem

ISO - International Organisation for Standardisation

IS - infosüsteem

IT - infotehnoloogia

JTC - Joint Technical Committee

RUP - Rational Unified Process

TK - tehniline komitee

XP - Extreme Programming