Tarkvara protsessid ja kvaliteet

Osaline lühiülevaade

Jaak Tepandi

Versioon 21.09.2021, https://tepandi.ee/tks-loeng.pdf

Lisamaterjalid on õpikeskkonnas https://moodle.taltech.ee/

Tallinn

Sisukord

1. Põhimõisted

Meie ümber on kõrgtehnoloogia ja sellesse kuuluvad tarkvarasüsteemid. Mida edasi, seda enam sõltuvad tarkvarast nii meie heaolu, elu kui ka ühiskond tervikuna. Süsteemide keerukuse ja vastutuse arenguga koos arenevad ka nõudmised ja kasvavad ootused, tarkvara probleeme nähakse nii kasutajate, tellijate kui ka tegijate poolelt.

Kasutajad pole rahul süsteemidega, mis teevad valet asja, on aeglased, ebaturvalised või ebamugavad. Tellijad ja tegijad kulutavad liialt vahendeid tarkvara parandamiseks ja hoolduseks, nurisevad kasutajate teadmatuse üle ja kannatavad viletsat tööd tegevate allapakkujate pärast. Käesolev lühikonspekt aitab loodetavasti olukorda parandada ... või vähemalt pakub selleks vahendeid.

1.1. Tarkvara kvaliteet: inimesed, 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 oma töös. Seepärast on oluline, et tarkvara töötaks kvaliteetselt. Mida see tähendab?

Väga lühidalt on kvaliteet toote vastavus mitmesugustele nõuetele. Keerukate toodete puhul tuleb vastavuse hindamisel arvesse võtta ka toote loomise protsessi. Kvaliteeti luuakse inimeste jaoks ja nende koostöös. Lisaks on olulised meetodid, vahendid, keskkonnad jne. Seega seob kvaliteet *toote, nõuded* tootele, tootmise *protsessi, inimesed / osapooled* ja muud komponendid. Vaatame neid lähemalt.

Meie *toode* on praegusel juhul tarkvara. See sisaldab palju komponente - kursuses kasutame tarkvara mõistet sageli tarkvarasüsteemi sünonüümina, lülitades sinna dokumentatsiooni, metoodikaid, vahendeid jne. *Nõuded* tulenevad vajadustest, lepingutest, ootustest, seadustest, heast tavast, eetikast jne; standardimine on vahend nõuete fikseerimiseks ja kvaliteedi edendamiseks, standardid esindavad universaalseid nõuete klasse. *Tootmisprotsessi* - tarkvara elutsüklit - võib mitmeti valida ja esitada (näiteks, XP, V-mudel, prototüüpimine, testipõhine arendus jpt). *Osapooled* mõjutavad kvaliteeti erinevalt; lihtsustatult võib öelda, et tellija defineerib kvaliteedi, arendaja realiseerib selle, testija hindab seda ning hooldaja arendab ja täiustab kvaliteeti edasi - vrd ka plaanimine-teostus-kontroll-tegutsemine (PDCA) metoodika.

Kvaliteedi mõiste näitlikustamiseks kasutada lauset "meie tahame teha seda" - "meie (osapooled) tahame (nõuded) teha (protsessid) seda (toode)".

Kvaliteeti ei saa sisse testida. Oskamatult 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 materjal ei saa kõiki teemasid hõlmata, nii käsitletakse vähemal määral tarkvara analüüsi, disaini ja kodeerimise teemasid, personalitööd jms. Rohkem on kaetud mitmesugused testimise, kontrolli ja hindamise meetodid ning kvaliteedi ja protsesside laiemad mõisted, mida teised kursused katavad vähem.

Mitmesuguseid kontrolli meetodeid võib kasutada enamikus arendusprotsessi tegevustes. Kvaliteedihaldust võib kasutada ka nende tegevuste eneste paremaks muutmisel. Kontrolli rakendatakse seega enamasti otsesele tulemusele (näiteks arendatav tarkvara), kvaliteedihaldust - ka arendusprotsessidele, -meetoditele ja -vahenditele.

Kvaliteet on üks komponent "projektijuhtimise kolmnurgas" (*project management triangle*), mille tippudes on kitsendustena kvaliteet, aeg ja maksumus. Kui pöörame ülemäärast tähelepanu mingile ühele omadusele, võivad teised kannatada (alati ei pruugi). Lühidalt: "Hea, kiire, odav. Vali kaks."

Kvaliteedist rääkides mõeldakse tihti erinevaid asju.

- Kvaliteet kui ideaal (arvamus ühelt arutelult: "tarkvara kvaliteeti pole olemas, kogu aeg on kiirustamine ja pole aega ühte asja valmis saada, juba tuleb järgmine") - ideaali poole tasub püüelda, samas ideaalset kvaliteeti ei pruugi alati olemas olla; tihti on see ka mitte saavutatav või ebaotstarbekas.
- Kvaliteet kui mingi tootja või kaubamärgiga kaasas käiv omadus ("see on kvaliteettoode") - selline teadmine võib anda kasulikku infot hankimisel.
- Kvaliteet kui suhe toote, nõuete, protsesside vahel ("hinnataset arvestades oli hotell väga hea") - selline suhe on üldjuhul olemas ja abiks hinnaefektiivsel tegutsemisel kõigis rollides.
- Kvaliteet kui mõõt - mõõt on alati olemas (näiteks ka siis, kui "selle süsteemi kvaliteet on väga madal").

Käesolev kursus tegeleb eelkõige kahe viimase mõistega. Samas annab ta teadmisi ja oskusi, mida võib kasutada kõigi toodud kvaliteedi mõistete puhul.

1.2. Kursus

Kursuse eesmärk on anda arusaamine kvaliteetse tarkvara arendamise väljakutsetest ja meetoditest. Käsitletakse nelja omavahel seotud tarkvaratehnika valdkonda: hange, arendus, kontroll, hooldus, mis esitatakse vastavalt tellija, arendaja, testija ja hooldaja vaadete kaudu. Kursuse edukalt läbinud üliõpilane:

- teab kvaliteetse tarkvara, tarkvaraarenduse ja tarkvaraprotsesside mõisteid;
- omab ülevaadet tarkvaranõuete koostamise, arendusprotsessi ja testimise meetoditest, korraldustest ning dokumenteerimisest;
- tunneb praktikas kasutatavaid agiilse tarkvaranõuete koostamise, arendusprotsessi ja testimise meetodeid, korraldust ning dokumenteerimise tavasid;
- oskab arendada kvaliteetset tarkvara ning organiseerida ja korraldada kvaliteetse tarkvara arendamist.

Kursuse koostamisel on arvestatud tarkvaraarenduse inseneri, tase 7 kutsestandardi põhimõtetega. Samuti arvestatakse *ACM/IEEE Computing Curricula* ja selle edasiarenduse *Computing Curricula 2020 (CC2020)* ideid. Siia kuuluvad tarkvaratehnikat käsitlevad *Software*

Engineering, Computer Science ja muud väljaanded. Meie kursus on seotud eelkõige valdkondadega *"Software Quality", "Software Requirements", "Software Process and Life Cycle"* ning *"Software Testing"*, samuti valikuliselt valdkondadega *"Software Construction" ja "Software Sustainment"*.

Kursus on kasulik

- tellijale, ostjale, et olla kursis tarkvarale esitatavate nõuetega, osata paremini koostada pakkumiskutset ning valida toodet, jälgida arendusprotsessi ja hinnata tulemust
- süsteemi ja tarkvara arendajale, testijale ja hooldajale, et luua paremat toodet, arendada seda edasi ja toetada, saavutada tellija rahulolu
- kasutajale, et osata küsida ostetava tarkvara omadusi ja teada, mida võib tarkvaratootelt oodata
- juhile, kes õpib, mida tellijalt, arendajalt, testijalt ja hooldajalt nõuda

Konkreetse semestri õppetöö korralduse materjalid (hindamine, ajakava, projektid jne) on kursuse veebis. Näited ja detailsemad selgitused antakse loengul ja harjutustes. Käesolev materjal on kursuse osaline lühiülevaade, mis ei asenda iseseisvat tööd kursuse veebis ning peatükkide lõpus toodud materjalidega. Loengute sisu ja järjestus ei pruugi langeda kokku failis oleva tekstiga. Materjal täieneb ja muutub töö käigus, semestri ja eksami lõplik variant on olemas enne eksamisessiooni algust.

Kuidas lugeda?

Sõltub huvidest. Tellija, juht või kasutaja eelistab võib-olla alustada protsesside, nõuete ja tarkvaratoote peatükkidest, mis võivad anda häid ideid pakkumiskutse koostamiseks või toote valikuks. Arendaja, testija või toetaja saavad valida teemasid vastavalt oma spetsiifikale, ka võib aluseks võtta materjali järjestuse. Kõigile võib soovitada kogu materjali läbilehitsemist, et näha, mis veel võib huvi pakkuda.

Kui soovida lisaks teadmistele/oskustele kursuse eest ka hinnet saada (loodetavasti ei osaleta ainult hinde saamiseks), tuleb kindlasti lugeda lisamaterjale, 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.

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.

Iga mõiste kohta võiks teada selle motivatsiooni, ideed, eeltingimusi, eeliseid, puudusi, tulemusi, seoseid teiste meetoditega, hinnangut, vahendeid.

Üks võimalus on küsida iga mõiste kohta küsimusi, sealhulgas: Milleks? Mis? Millal? Kes? Kus? Kuidas?

Autor tänab kolleege, kursuse kuulajaid ja redigeerijaid paljude kasulike märkuste ja ettepanekute eest. Edasised kommentaarid ja märkused selle materjali kohta on oodatud, palun saata need aadressil jaak.tepandi (at) ttu.ee või anda edasi loengul/praktikumis.

Küsimusi ja materjale

Kontrollküsimusi ja ülesandeid

- Miks on tarkvara kvaliteedi teema oluline? Tarkvara kvaliteet, selle komponendid ja osapooled. Kvaliteet on suhteline ja sõltub kontekstist (miks?). Kvaliteet kui kompromiss (projektijuhtimise kolmnurk). Neli erinevat vaadet kvaliteedile.
- Pakkuge ise kvaliteedi mõiste, võrrelge ülal pakutud mõistega
- Näited tarkvara probleemidest tulenenud rasketest intsidentidest. Tarkvara probleemide ulatuse ja maksumuse hinnangud
- Kas tarkvara kvaliteedi määratlus erineb teiste toodete kvaliteedi määratlusest? Miks?
- Millal võib kvaliteedi määratluses piirduda vaid tootega? Vaid toote ja nõuetega?
- 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 / millal on võimalik, et kvaliteeti pole?

Materjale iseseisvaks tööks (näited)

- Ian Sommerville. Software Engineering. Tenth Edition. Addison-Wesley, Chapter 1, 24
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE. Chapter 10, Section 1.
- Daniel Galin, Software Quality assurance from theory to implementation, Pearson - Addison-Wesley. Chapters 1,2.
- Stefan Wagner. Software Product Quality Control. Springer, www.it-ebooks.info.
- Certified Tester Foundation Level Syllabus, ISTQB. Chapter 1.1.4.

2. Tarkvara protsessid

2.1. Protsessid, elutsükli mudelid, arendus- ja protsessiraamistikud

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 korrastada ja struktureerida. Seda saab teha protsesside, elutsükli mudelite ja protsessiraamistike abil.

Protsess on kogum omavahel seotud või suhtlevaid tegevusi, mis muudab protsessi sisendid väljunditeks. Näiteid tarkvaraga seotud protsessidest: hankimine, arendus, verifitseerimine, hooldus, tarnimine, ekspluatatsioon, konfiguratsioonihaldus, muudatuste haldus jne.

Protsesside vajadus sõltub olukorrast, näiteks ettevõtte tüübist, süsteemidest, töötajate ja asukohtade arvust, ettevõtte arengustaadiumist jne. Nii ei võimalda ebapiisavad protsessid keeruka struktuuri ja paljude kasutajate korral tööd toimivalt korraldada. Teisest küljest, liigsed protsessid muudavad töö bürokraatlikumaks ja kahandavad loovust. Praktikas võib arendus olla mõnikord vähem reguleeritud kui muud valdkonnad, näiteks hooldus ja kasutamine.

Näeme, et programmi kirjutamine, mis esimesel pilgul võib näida tarkvara protsessides põhilisena, on oluline, kuid tegelikes rakendustes siiski vaid üks osa ühest põhiprotsessist (arendus).

Tarkvara elutsükli all mõistetakse tarkvara arengut selle algatamisest kuni mahavõtuni.

Tarkvara elutsükli mudel kirjeldab elutsüklit puudutavaid protsesse ja tegevusi. Näiteid tarkvara elutsükli mudelitest: ekstreemprogrammeerimine, kosemudel.

Keeruka ja vastutusrikka toote tegemisel on üldjuhul vaja vähemalt:

- aru saada, mida tuleb teha (sh nõuete kirjeldamine)
- teha see toode valmis (sh projekteerimine ja arendus)
- hinnata seda (sh kontroll)
- hooldada ja muuta seda (sh hooldus)

Praktikas lisandub veel tegevusi või protsesse. Kõiki neid võib teha järjest, iteratiivselt või muud moodi organiseeritult. Koos nad moodustavad tarkvara elutsükli mudeli. Elutsükli mudelite näiteid:

Waterfall - http://www.nouveautech.co.uk/images/developmentprocess.gif

V-model - http://en.wikipedia.org/wiki/V-Model_(software_development)

Spiral - https://en.wikipedia.org/wiki/Spiral_model

RUP – https://en.wikipedia.org/wiki/Rational_Unified_Process#/media/File:Development-iterative.png

Extreme Programming - http://www.extremeprogramming.org/map/project.html

TDD - https://en.wikipedia.org/wiki/Test-driven_development#/media/File:TDD_Global_Lifecycle.png

Samuti: Scrum - https://en.wikipedia.org/wiki/File:Scrum_process.svg

Samuti: http://agilemanifesto.org/

Samuti: http://en.wikipedia.org/wiki/Software_development_methodology

Samuti: https://youtu.be/xtpyjPrpyX8

Elutsükli mudeli või protsessi valik sõltub olukorrast - igas olukorras õigeid või valesid protsessi mudeleid ei ole. Lihtne reegel (raske rakendada): nii vähe fikseeritud protsesse kui võimalik, aga mitte vähem.

Raamistikud võivad hõlmata põhimõtteid, protsesse, praktikaid ja muid komponente. Arendusraamistikud kirjeldavad tarkvara või muu tehise arendust. Näiteid: Scrum, RUP, SAFe. Üks ja sama termin võib mõnikord tähistada protsessi (kitsamas mõttes) või raamistikku (laiemas mõttes).

Protsessiraamistikud kirjeldavad paljusid või kõiki mingi valdkonna protsesse. Näited: ISO/IEC 12207, CMMI, COBIT, ITIL. Selliste raamistike mahukus tuleneb eelkõige sellest, et nad hõlmavad väga mitmesuguseid protsesse, mitte ainult tarkvara arendust.

Levinud tarkvara protsessiraamistikud lubavad oma arendusprotsessis kasutada erinevaid elutsükli mudeleid. Seepärast ei ole vastuolu näiteks protsessiraamistike ja agiilsete arendusmeetodite vahel – üldjuhul võivad esimesed sisaldada teisi.

2.2. Agiilne tarkvaraarendus (välearendus)

Agiilne tarkvaraarendus (välearendus) on kompleks põhimõtteid, metoodikaid, tehnikaid ja raamistikke, mis seavad fookusse inimeste vahelise koostöö, töötava tarkvara, nõuete muudetavuse vastavalt kliendi vajadustele ja paindliku olukordadega kohanemise. Välearenduse põhimõtted on sõnastatud agiilse tarkvaraarenduse manifestis (https://agilemanifesto.org/iso/et/manifesto.html) ja selle põhimõtetes (https://agilemanifesto.org/iso/et/principles.html). Agiilse tarkvaraarenduse manifest rõhutab:

- osapoolte vaatest - inimesi ja nendevahelist suhtlust,
- toote vaatest - töötavat tarkvara,
- nõuete vaatest - koostööd kliendiga ja valmisolekut muudatusteks,
- protsesside vaatest - paindlikkust vastavalt olukorrale.

Põhjalikku ülevaadet välearendusest võib lugeda näiteks aadressilt https://wiki.itcollege.ee/index.php/V%C3%A4ledad_tarkvaraarenduse_mudelid.

Välearenduses rõhutatud väärtused ja põhimõtted on tuletatud paljudest tarkvaraarenduse raamistikest ning toetavad neid. Järgnevalt on toodud mõned välearenduse raamistike, tehnikate ja meetodite näited.

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 agiilsete metoodikate väljatöötamisel.

Ekstreemprogrammeerimise (XP, Extreme Programming, nt http://www.extremeprogramming.org/) üheks eesmärgiks on suurem paindlikkus muutuvate nõuete tingimustes. XP põhiideed pakkus välja Kent Beck 1996.a., selle seoseid testimisega on kirjeldatud kontrolli korralduse osas.

Testipõhisel arendusel (test driven development, nt http://en.wikipedia.org/wiki/Test-driven_development) luuakse testid enne realiseerimist kliendi kasutuslugude põhjal. Saab kasutada väga erineva taseme teste, alates ühiktestidest (rohkem kasutusel, programmeerijalt, kohe enne realiseerimist) kuni vastuvõtmise testideni (harvem, Tellijalt, funktsionaalsed).

Scrum on iteratiivne ja inkrementaalne (tarkvara)projektide haldamise meetod, mille põhiideed pakuti välja aastatel 1986-1995 (nt http://www.scrum.org/Scrum-Guides).

Timmitud tarkvaraarenduses (lean software development) kohandatakse timmitud tootmise (lean manufacturing) põhimõtteid tarkvara arendusele. Näited põhimõtetest: kõrvaldage raiskamine, otsustage nii hilja kui võimalik, soodustage õppimist jne. Näiteid raiskamisest: osaliselt tehtud töö, asjatud lisafunktsioonid, ümberõpe jne. Kanban (https://www.infoq.com/minibooks/priming-kanban-jesper-boeg/) on timmitud tootmise meetod tiimide töö juhtimiseks ja parendamiseks. Selle lähenemisviisi kaks esmast praktikat on tegevuste visualiseerimine ja käimasoleva töö ulatuse piiramine.

Kui algselt rakendati agiilseid meetodeid eelkõige ühe tiimi ulatuses, siis praeguseks on tekkinud mitmeid lähenemisi, mis seda rakendusala laiendavad. Näiteks Scaled Agile Framework (SAFe, https://www.scaledagileframework.com/) on meetodite ja põhimõtete raamistik timmitud praktikate laiendamiseks paljude tiimide vahelisele töökorraldusele.

Paljusid raamistikke, meetodeid ja tehnikaid saab omavahel kombineerida, näiteks XPd ja testipõhist arendust.

Kasulik on eristada üldotstarbelisi ja tarkvara-spetsiifilisi agiilseid meetodeid ja tehnikaid. Üldotstarbelisi (nt Scrum) saab kasutada paljudes valdkondades, tarkvara-spetsiifilisi (nt testipõhine arendus) - vaid tarkvara arenduses.

Alguseks ja väikeste projektide puhul on põhjalikud raamistikud väga mahukad ja keerukad. Tegelikult nõuab aga iga mittetriviaalse raamistiku süstemaatilisele kasutusele võtmine pingutust.

2.3. Haldamisraamistikud ja arendusmetoodikad

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

Ülalmainitud standardid võimaldavad rakendada mitmesuguseid arendusmetoodikaid. Näiteks on ISO/IEC 12207 rakendusjuhendis pakutud standardi evitamiseks kolme võimalikku elutsükli mudelit: koskmudel, inkrementmudel (arendamine koosneb mitmest iteratsioonist) ja

evolutsioonmudel (arendamine koosneb mitmest iteratsioonist, sealhulgas ka nõuded selgitatakse arenduse käigus). Neist viimane (võib-olla ka teine, sõltub XP esitusest) vastab XP metoodikale. On leitud ka, et XP metoodikat saab kasutada mitmete CMMI teise ja kolmanda taseme praktikate rakendamiseks (juba teisel tasemel jääb osa CMMI protsesse siiski XP-st välja)[1].

Erinevus on siis pigem selles, et arendusmetoodikad 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, ekspluatatsiooni, tarne, hoolduse, koolituse ja mitmete teiste protsessidega - see ei ole lihtsalt XP teema. Pigem võiks seega rääkida haldamisraamistikkudest (ISO/IEC 12207, CMM jne) ning arendusmetoodikatest (XP, Scrum ja teised), millel on erinev ulatus, aga mis ei vastandu omavahel. On loomulik, et protsessiraamistikud (käsitlevad paljusid protsesse) on suurema mahuga kui arendusmetoodikad (käsitlevad ühte protsessi).

Kombinatsioon organisatsiooni tegevusele suunatud laiemast raamistikust, mille raames kasutatakse agiilset arendust, on kasutusel mitmetes Eesti ettevõtetes.

2.4. Kvaliteet, arendus ja kontroll

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

- äri- ja töökeskkonnast, inimestest, juhtimisest, organisatoorsetest vahenditest jne
- tarkvara arenduse raamistikust, protsessidest, tegevustest, kvaliteedihaldusest
- riistvara vahenditest: piisavalt võimas keskkond, töökindlust toetavad vahendid jne
- tarkvara arenduse vahenditest: arenduskeskkonnad, programmeerimiskeeled jne

Kvaliteedihaldust on tarkvarasüsteemi elutsüklisse mitmeti ühendatud:

- kui tarkvara arendus alles hakkas tekkima, tegeldi peaasjalikult programmeerimisega ja kvaliteedist ei räägitud
- üsna kiiresti leiti, et vigane tarkvara võib olla väga ohtlik kasutajate varale ja tervisele; seepärast hakati tegelema testimisega, järgnesid inspektsioonid, 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.

[1] http://www.hristov.com/andrey/fht-stuttgart/xp-cmm-paper.pdf

2.5. Hankija tegevused

Hankija (nagu ka muude osapoolte) tegevused sõltuvad olukorrast. Üldjuhul 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. Näiteks pakub ISO/IEC 12207 järgmised hankimise tegevused, mida saab vastavalt vajadusele valida või kohandada:

- 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.

Toodud tegevustes sisalduvad näiteks ka nõuete muudatuste haldus ja vastuvõtmise plaani koostamine. Igast tegevusest võib siseneda mõnesse muude protsessi, näiteks hankimise ettevalmistamine võib eeldada nõuete haldamise protsessi kasutamist.

CMMI materjal "*CMMI for Acquisition (CMMI-ACQ)*" (http://www.sei.cmu.edu/library/abstracts/reports/10tr032.cfm) annab põhjaliku ülevaate hankimistegevustest. Konkreetsel hanke puhul võib neid tegevusi kasutada kontrollimaks, et midagi 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.

Muuhulgas, muudatuste halduse kohta annab soovitusi CMMI nõuete halduse (*Requirements management*) valdkonna eripraktika SP 1.3 (*Manage Requirements Changes,* see on vajalik juba teisel küpsustasemel). Süsteemi vastuvõtmise kohta annavad soovitusi mitmed CMMI valdkonna praktikad, näiteks hanke valideerimise (*Acquisition validation*) valdkonna alla kuuluvad eripraktikad.

Teenuste hankimisel on otstarbekas kasutada standardite seeriat ISO/IEC 20000 ja/ või ITILit (IT Infrastructure Library, https://www.axelos.com/best-practice-solutions/itil).

Hankija tegevused on põhjalikult kirjeldatud COBIT raamistikus (https://cobitonline.isaca.org/about).

Viidatud standardid ei eelda, et kõiki tegevusi tuleb täita, vaid pakuvad võimaluse teha tegevustest valik vastavalt olukorrale. Näiteid erinevate hankeprotsesside kohta on toodud ka kursuse õpikeskkonna materjalides. Valitud tegevuste sisu ja ulatuse põhjal võib prognoosida hankija töömahtu hankes.

Küsimusi ja materjale

Kontrollküsimusi ja ülesandeid

- Kuidas erinevad tarkvara protsessiraamistikud ja elutsükli mudelid?
- Näited erinevate elutsükli mudelite kohta: XP, Scrum, testipõhine arendus, kosemudel, V-mudel, spiraalmudel
- Millal pole mõtet rääkida tarkvara elutsüklist?
- Milline elutsükli mudel on parim? Kuidas valida elutsükli mudelit?
- Millised on hankija tegevused, kuidas nad sõltuvad arendusprotsessist ja hankijast?
- Kuidas on seotud teenused, tarkvara, nõuded ja protsessid?
- Tarkvara protsessiraamistike ja protsesside näited: ISO/IEC 12207, CMMI, COBIT jt
- Kas tarkvara protsessiraamistikke ja agiilseid meetodeid saab koos kasutada?

Materjale iseseisvaks tööks (näited)

- Ian Sommerville. Software Engineering. Tenth Edition. Addison-Wesley, Ch 2,3.
- Manifesto for Agile Software Development, http://agilemanifesto.org/
- Extreme Programming: A gentle introduction, http://www.extremeprogramming.org/
- Scrum, http://www.scrum.org/Scrum-Guides
- Daniel Galin, Software Quality assurance from theory to implementation, Pearson - Addison-Wesley. Chapter 7.
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE. Chapter 8.
- Teenustaseme lepped ja mittefunktsionaalsed nõuded, https://www.itpedia.nl/2017/07/14/devops-plan-slas-and-non-functional-requirements/

3. Nõuded

3.1. Nõuete allikad ja liigid

Nõuded tulenevad erinevatest allikatest. Eetikanormidest, seadustest, eriala parimatest praktikatest tulenevad paljud nõuded, mida mõnikord nõuetena ei fikseeritagi, kuid mis võivad osapoolte konfliktide puhul olla väga olulised. Standardid esindavad suuri nõuete klasse kindlates valdkondades. Konkreetse rakenduse nõuded on täpsemalt kirjeldatud tarkvaraprojekti dokumentatsioonis ja lepingutes.

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 juhendmaterjalis "Guide to the Software Engineering Body of Knowledge" (SWEBOK, https://www.computer.org/education/bodies-of-knowledge/software-engineering) eristatakse toote ja protsessi (sh arendusprotsessi) 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).

Tarkvara kvaliteedimudelite standardis ISO/IEC 25010 esitatakse tootekvaliteedi, andmekvaliteedi ja kasutuskvaliteedi mudelid. Kuna kvaliteedi määrab vastavus nõuetele, saab eristada toote-, andme- ja kasutusnõudeid.

Ärinõuded võivad lisaks sisaldada strateegilisi, keskkonna, maksumuse ja muid piiranguid.

Eri tüüpi nõuded võivad olla omavahel sõltuvuses. Toote nõuded tulenevad enamasti ärinõuetest, (arendus)protsessi nõuded nii äri- kui ka toote nõuetest.

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 raske ettevõtmine.

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

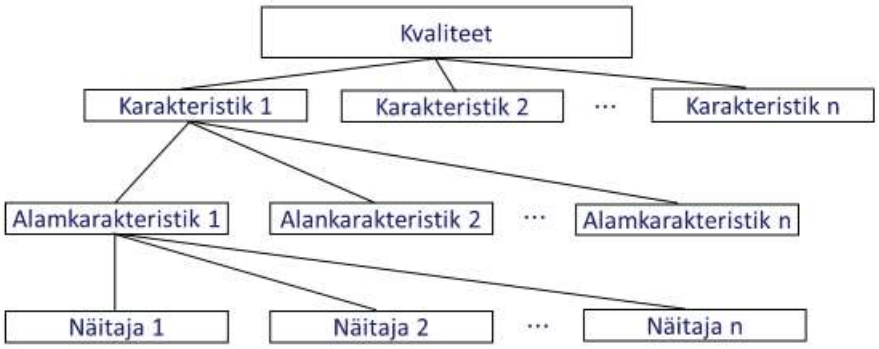
Teine võimalus nõuete üldisemaks määratluseks on kitsendada vaadeldava tarkvara klassi, ette andes selle funktsionaalsuse, rakenduspiirkonna või muud parameetrid. Väikeettevõtete raamatupidamissüsteemidele võib näiteks seada ühiseid nõudmisi. Riigiasutustes kasutatavatele rakendustele võib samuti seada mõistlikke üldisi tingimusi, mida iga konkreetse tarkvara puhul kitsendatakse.

3.2. Kvaliteediatribuutide süsteem

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

Üks selline süsteem on nõuete esitamine mitmetasemelise kvaliteediatribuutide puu kaudu (vt. joonis allpool). Puu ülemisel tasemel on kvaliteedikarakteristikud (kvaliteedifaktorid), mis annavad üldise ettekujutuse toote olulistest omadustest. Kvaliteedikarakteristikud kui üldise taseme atribuudid on määratud pigem juhtkonnale, tellijale, süsteemi kasutajale.

ISO/IEC 25000 kvaliteedimudelites kasutatav struktuur



Selliseid omadusi on raske kvantifitseerida, seepärast jaotatakse iga karakteristik alamkarakteristikuteks, mis moodustavad puu järgmise taseme või tasemed. Alamkarakteristikud on detailsemad ja rohkem arendusele orienteeritud, neid saab kasutada näiteks väljatöötaja või kasutajapoolne projektijuht.

Iga alamkarakteristik jaguneb omakorda näitajateks (*metrics*), mida saab otse mõõta. Sellised näitajad võivad huvi pakkuda kõigile osapooltele. Näiteks on näitajaid, mida peab teadma kasutaja (seisakute kogukestvus või tõrgete arv ajaühikus kindla tööprofiili puhul), on arendusse ja hooldusse puutuvaid näitajaid. Näitajaid võib sätestada teenustaseme lepetes.

Sellise süsteemi puhul saab näitajatest 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 alamkarakteristikuid karakteristikuteks enam ei koondata, kuid võimatu see pole. Viimasel juhul, valides sobivad kaalud või muud integreerimismeetodit toetavad objektid, saab alustada näitajatest 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 karakteristikute osas ei

esinenud ühtki tõsist viga) ja hinnatakse nende täitmist nagu näiteks testimise standardis IEEE Std 829.

Teenustaseme lepetes sätestatud näitajaid jälgitakse, vajadusel võetakse meetmed.

Selliseid kvaliteediatribuutide süsteeme on mitmeid, allpool vaatame kahte rahvusvahelisest standardit.

3.3. Kvaliteedinäitajad standardiseerias ISO/IEC 25000 ja mujal

Üks levinud tarkvara kvaliteedinäitajate skeem on esitatud rahvusvahelises ISO/IEC 25000 standardiseerias. Selle seeria olulisemaid osi on kvaliteedimudelite standard ISO/IEC 25010 *Software engineering: Software product Quality Requirements and Evaluation (SQuaRE) — Quality model,* mille eestikeelne tõlge on standard EVS-ISO/IEC 25010:2011.

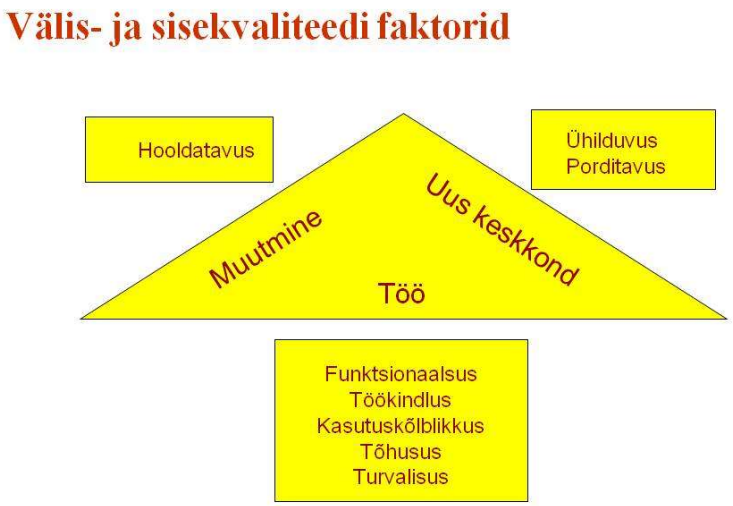
Standardi ISO/IEC 25010 toote kvaliteedi mudel on toodud järgnevas tabelis (vt ka ISO terminite andmebaas https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en).

Süsteemi / tarkvaratoote kvaliteet (EVS-ISO/IEC 25010:2011põhjal)

Funktsio-naalne sobivus	Soorituse tõhusus	Töökindlus	Ühilduvus	Kasutatavus	Turvalisus	Hooldatavus	Porditavus
Funktsio-naalne täielikkus Funktsio-naalne õigsus Funktsio-naalne kohasus	Ajaline käitumine Ressursi-kasutus Suutvus	Küpsus Tõrketaluvus Taastuvus Töökindluse vastavus	Koosolu-võime Koostalitlus-võime	Kohasuse äratuntavus Õpitavus Käsitsetavus Eksituskindlus Kasutajaliidese esteetika Hõlpsus	Konfident-siaalsus Terviklus Salgamatus Jälitatavus Autentsus	Modulaarsus Taaskasuta-tavus Analüüsitavus Modifitsee-ritavus Testitavus	Sobitatavus Installeeri-tavus Asendatavus

Näitajate loogika on lihtsustatult järgmine. Süsteemi kasutatakse, muudetakse ja kantakse üle teistele riist- ja tarkvaraplatvormidele. Kasutamisel peaks süsteem täitma vajalikke funktsioone ning olema töökindel, kasutatav, efektiivne ja turvaline; muutmisel on vaja head hooldatavust; teise keskkonda ülekandmisel on vaja ühilduvust ja porditavust (vt joonis).

Kõiki näitajaid ei ole tavaliselt vaja arvestada, süsteemi arendamisel valitakse toodud mudelist vajalikud nõuded.



Üks ISO/IEC 25000 standardiseeria eellasi oli ISO/IEC 9126. Viimase sisu võib olla paremini avalikult kättesaadav ja pakub ka praegu päris hea kvaliteediatribuutide nimekirja, mida saab vaadata veendumaks, et midagi olulist pole jäänud arvestamata. Standard võtab arvesse kasutus-, välis-, sise- ja protsessi kvaliteeti. Elutsükli protsesside kvaliteet soodustab toote kvaliteedi (sise- ja väliskvaliteedi) parandamist, toote kvaliteet omakorda soodustab kasutuskvaliteedi tõstmist. Jaotises 7.3 antakse ülevaade mõlema standardi toote ja kasutuskvalitedi mudelist.

Toodud kvaliteedinäitajate komplektid ei ole ainuvõimalikud. Lisaks ülalmainitud ISO/IEC 25000 seeria standarditele võiks vaadata ka näiteks riigi IT koosvõime raamistikku (https://www.mkm.ee/et/riigi-infosusteemi-koosvoime-raamistik), aga ka veebisaidi kasutatavuse nõudeid maailmas (https://www.w3.org/WAI/standards-guidelines/wcag/), USAs (https://en.wikipedia.org/wiki/Section_508_Amendment_to_the_Rehabilitation_Act_of_1973) ja Eestis (https://mkm.ee/et/riigi-infosusteemi-koosvoime-raamistik).

3.4. Funktsionaalsed ja mittefunktsionaalsed nõuded

Nõudeid saab esitada kvaliteedinäitajate raamistikus. Ülaltoodud kvaliteedinäitajaid, seega ka nõudeid võib jaotada funktsionaalseteks ja mittefunktsionaalseteks.

Funktsionaalsed nõuded vastavad küsimusele "Mida peab tarkvara tegema?" (näiteks, süsteem peab võimaldama kauba tellimist). Ülaltoodud tabelis määravad funktsionaalsuse põhiliselt funktsionaalse sobivuse näitajad.

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 põhiliselt esimest faktorit (funktsionaalsus), püüdes kaardistada tarkvara funktsionaalsust. Mittefunktsionaalsed nõuded jäetakse kaardistamata, mis võib viia süsteemi arenduses suurte probleemideni - näiteks võivad käideldavuse erinevad nõutud tasemed mõjutada süsteemi arhitektuuri ja muuta suurel määral süsteemi maksumust.

Üks põhjus selleks on, et funktsionaalsuse analüüs üks kõige töömahukamaid osasid nõuete analüüsist.

3.5. Testitavad / mittetestitavad, reaalsed / ebareaalsed nõuded jne

Nõuete püstitamisel on oluline, et nad oleksid testitavad. Funktsionaalsuse 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. Siin võib tekkida probleeme (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, sest meil ei ole kriteeriume hindamaks, kas mingi konkreetne töökindluse tase vastab sellele nõudele või ei.

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 osutuda, et spetsifikatsioon ei ole piisav (kas alati?).

Testide spetsifitseerimine varases arenduse staadiumis, paralleelselt tarkvara analüüsiga, aitab kaasa kvaliteetsemate nõuete püstitamisele.

Nõue võib olla testitav, kuid ebareaalne, ebamõistlik, ebapiisavalt spetsifitseeritud jne. Näide: "süsteemi vastuse aeg peab jääma alla 3 sekundi". Mis on sellise nõude probleemid? Kui sellised nõuded on lepingus, on see tavaliselt ühe poole (millise?) lisarisk. Kuidas sellist nõuet realistlikult esitada?

Üks oluline nõuete omadus on jälitatavus (*traceability*) - nõue peaks olema jälgitav kuni selle algallikani. Mitmed konfliktid tulenevad sellest, et osapooltel on nõuetest, nende kohustuslikkusest ja kehtivusest erinev arusaamine. Kui nõue on jälitatav algallikani, on konflikte lihtsam lahendada.

Nõuetelt võib oodata ka muid omadusi. Need kõik on siis "nõuded nõuetele" ehk metanõuded. Populaarne on näiteks SMART kriteeriumide kompleks eesmärkide ja nõuete hindamiseks, mida võib lahti kirjutada kui "*Specific, Measurable, Agreed, Realistic and Time bound*". Mis on siit puudu? SMART kriteeriumit võib lahti kirjutada ka teistmoodi. Kas näiteks "*Specific, Measurable, Achievable, Realistic and Time bound*" oleks hea komplekt nõudeid nõuetele (teisisõnu, kas ka metanõuetele võib seada nõuded - näiteks, et nad ei dubleeriks üksteist)?

3.6. Teenused ja teenustaseme lepped: tarkvara, nõuded, protsessid jm

Paljudel juhtudel on hankija (klient, kasutaja) peamiselt huvitatud vajalike tulemuste saavutamisest, mitte tarkvarast kui sellisest - ehk, hangitakse pigem teenust kui tarkvara. Ka sellisel juhul võib tarkvaral olla teenuse osutamisel oluline roll, kuid teenuse osutamiseks tuleb tavaliselt kaasata ka mitmeid muid komponente - riistvara, inimesi, keskkondi jne; need peavad pidevalt toimima (protsessid); seda toimimist tuleb mõõta (näitajad). Seega võib teenus hõlmata muuhulgas nii inimesi, tarkvara, riistvara, protsesse kui ka keskkondi ja on pigem välise süsteemi või organisatsiooni taseme mõiste, mitte niivõrd tarkvara taseme mõiste.

Teenuste kvaliteedi nõuded määratletakse teenustaseme leppes (TTL; inglise keeles *Service level agreement, SLA*). See on teenuseosutaja ja kliendi vaheline kirjalik kokkulepe, mis määratleb teenused ja teenuste sihttasemed. Võib ka öelda, et teenustaseme lepe kirjeldab mittefunktsionaalsed nõuded lähtuvalt ärivajadustest ning fikseerib need dokumenteeritud lepingutena.

Kvaliteedinäitajate hierarhilise süsteemi puhul kirjeldatakse teenustaseme leppes üldjuhul ülemise taseme kvaliteedinõuded. Nende täitmiseks peavad tavaliselt nii tarkvara, protsessid, organisatsioon kui ka keskkond omakorda rahuldama nendele kehtestatud nõudeid.

Sõltuvalt teenusest võib teenustaseme lepetes sätestada väga erinevaid nõudeid. Näiteks võib tarkvarateenuste puhul kokku leppida garanteeritud käideldavuse mingi ajaperioodi jooksul või reageerimise aja tõrke korral. Vastavalt standardile EVS-ISO/IEC 20000-2:2013 peaks teenustaseme lepe sisaldama spetsiifilisi lepingu punkte (nt TTLi muudatuste ohje) ja teenuse kvaliteeti sätestavaid punkte. Viimased on muuhulgas:

- teenuse aeg, näiteks 08.00 kuni 18.00 ning erandlikud kuupäevad, nagu nädalavahetused, riigipühad, kriitilised äriperioodid, samuti teenuse osutamine väljaspool teenuse aega;
- kavandatud ja kokku lepitud teenuste katkestused, sealhulgas nende etteteatamise aeg ja arv perioodi kohta;
- kliendi vastutused, näiteks süsteemide õige kasutamine ja infoturbe poliitikast kinnipidamine;
- teenuseosutaja vastutus ja kohustused, näiteks turvalisus;
- mõju ja prioriseerimise juhised;
- eskalatsiooni ja teavitamise protsess;
- kaebuste menetlemise protseduur;
- teenuse sihttasemed;
- töökoormuse ülemised ja alumised piirid, näiteks süsteemi võime toetada kokkulepitud kasutajate arvu / töö mahtu, süsteemi tootlikkus;
- teenustaseme leppes viidatakse tavaliselt ka meetmetele, mida tuleb võtta teenusekatkestuse korral, sealhulgas nii intsidentide kui ka avariide puhul.

Teenustaseme lepped peaksid keskenduma kliendi ja teenuseosutaja jaoks kõige olulisematele teemadele. Üleliigsed teenustaseme lepped võivad tekitada segadust ning liigseid kulutusi, andmata vajalikku efekti.

Küsimusi ja materjale

Kontrollküsimusi ja ülesandeid

- Nõuded kui kvaliteedi olemuslik komponent. Nõuete allikad. Kvaliteedimudelitest kvaliteedinõueteni. Funktsionaalsed ja mittefunktsionaalsed nõuded, testitavad ja mittetestitavad nõuded, realistlikud ja ebarealistlikud nõuded, jälitatavad ja mittejälitatavad nõuded. Riskid. ISO/IEC 25000 seeria. Kvaliteedi karakteristikud ja alamkarakteristikud (näited). Toote-, kasutus- ja andmekvaliteedi mudelid. Toote kvaliteedimudeli ülesehitus ja komponendid.
- Tarkvara nõuete liigitusi
- Tarkvara kvaliteedi atribuutide struktuuri vajalikkus, näide struktuurist, integreerimine
- Näide kvaliteediatribuutide süsteemist, kahe kvaliteedikarakteristiku kriteeriumid
- Tooge näiteid funktsionaalsete ja mittefunktsionaalsete nõuete kohta
- Tooge näiteid testitavate ja mittetestitavate, reaalsete ja ebareaalsete nõuete kohta
- Kas saab testida, kui nõuded puuduvad?
- Antud mittetestitav nõue, muuta testitavaks
- Kuidas on seotud teenused, tarkvara, nõuded ja protsessid? Tarkvaranõuded ja teenuse sihttasemed? Milleks on vaja teenustaseme leppeid ja mida nad sisaldavad?

Materjale iseseisvaks tööks (näited)

- Ian Sommerville. Software Engineering. Tenth Edition. Addison-Wesley, Ch 4.
- Daniel Galin, Software Quality assurance from theory to implementation, Pearson - Addison-Wesley. Chapter 3.
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE. Chapter 1 and Chapter 10 Section 1.3.
- Terms and definitions from ISO/IEC 25010, https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en.
- Web Content Accessibility Guidelines (WCAG) 2.0, http://www.w3.org/TR/WCAG20/.

4. Tarkvaratoode

4.1. Tarkvara ja süsteem

Kõigis protsessides (sh arendus, testimine, hindamine, kasutamine) on otstarbekas teha vahet tarkvaral ja süsteemil, mille koosseisu tarkvara kuulub.

Tarkvara koosseisu kuuluvad infotöötlussüsteemi programmid, protseduurid, reeglid ja nendega seotud dokumentatsioon või osa neist. Tarkvara arenduse tulem (toode, teenus) võib hõlmata ka mitmesuguseid muid komponente, mis kõik võivad olla kvaliteedihalduse objektid, näiteks

- arenduse käigus hangitud infotehnoloogiavahendid: riistvara, standardtarkvara, sideseadmed
- arenduse käigus tehtud töö: täitja arendatud tarkvara (sealhulgas lähtekood, objektkood, täitmiskood jm); installatsioonid, kohandamised, muudatused; andmehõive
- muudatused tellija organisatsioonis, protsessides, töökorralduses...
- projektdokumentatsioon kasutamise kohta (sh kasutajajuhendid); objektsüsteemi (nt organisatsioon, mille jaoks tarkvara arendatakse) kohta; loodavate objektide kohta (programmi/testimise dokumentatsioon); installeerimise ja seadistamise kohta; arenduse (sh testimise) kohta
- metoodika: tulemuste kasutamine; tulemuste edasiarendamine; uute arenduste tegemine
- vahendid hoolduseks, muudatusteks, arenduseks
- teadmised projekti tulemuste kasutamisest (nt läbi viidud koolitused); objektsüsteemist (süsteemianalüüs või vajalikud muudatused seadusandluses); projektist; arendusest
- õigused kasutamiseks, arendamiseks, levitamiseks
- andmed

Süsteem - interakteeruvate elementide ühend, mis on korraldatud saavutama üht või mitut deklareeritud eesmärki (ISO/IEC 12207).

Tarkvarasüsteemi arendus hõlmab üldjuhul sihttarkvara (seda arendatakse) ja sihivälist tarkvara (seda kasutatakse arenduse käigus). Analoogilised mõisted on sihtandmed ja sihivälised andmed.

Seotud mõiste on teenus - tootega seotud tegevuste või töö sooritamine või ülesannete täitmine (ISO/IEC 12207).

Sõltuvalt sellest, kas hangime / arendame / testime / haldame toodet, süsteemi või teenust, võib olla vaja erinevaid mõisteid, meetodeid, tööriistu.

Süsteemi keerukus ja komponentide koosvõime vajadused sõltuvad oluliselt sellest, kas suhtlus toimub erinevate komponentide, asutuste, riikide jne tasemel.

4.2. Tarkvara arhitektuur

Arhitektuur räägib olulistest komponentidest ja valikutest. Ralph Johnson: "*Architecture is about the important stuff. Whatever that is*".

Mis on oluline? Üks võimalus on mõõta olulisust muutmise maksumusega. Grady Booch: "*Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change*".

Standardi ISO/IEC/IEEE 42010:2011 järgi on arhitektuur süsteemi aluskorraldus, mida kehastavad süsteemi komponendid, komponentide seosed üksteisega ja keskkonnaga ning süsteemi kavandamise ja arenduse põhimõtted (https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:42010:ed-1:v1:en).

Arhitektuuri loomisel pakuvad huvi eelkõige arhitektuuriliselt olulised nõuded (*Architecturally Significant Requirements*), s.o need nõuded, millel on mõõdetav mõju tarkvarasüsteemi arhitektuurile. Olulisust mõõdetakse siinkohal kavandatud arhitektuuri muutuste kõrge hinnaga. Arhitektuuriliselt olulised nõuded võivad olla nii funktsionaalsed kui ka mittefunktsionaalsed (nt turvalisus, töökindlus, tõhusus).

Arhitektuuri võib kavandada ettevõtte, süsteemi, tarkvara ja muudel tasemetel. Tarkvara arhitektuuri loomist võib vaadata eraldi tarkvara arenduse etapina või tarkvara projekteerimise ühe osana.

Näiteks pakub standard ISO/IEC 12207 võimaluse võtta kasutusele eraldi süsteemi või tarkvara projekteerimise protsessid. Kui arhitektuuri arendamine on projekteerimise osa, siis vaadatakse tihti eraldi arhitektuuri projekteerimist (*architectural design, Architecture Definition process*) ja tarkvara detailprojekteerimist (*detailed design, Design Definition process*).

Arhitektuuri kirjeldusi võib kasutada osapoolte vahelises kommunikatsiooonis, süsteemi analüüsis ja projekteerimisel, süsteemi dokumenteerimisel, süsteemi mõistmiseks ja koolituseks ning süsteemi toetamisel.

Arhitektuuri vaade kujutab mingit ühte süsteemi aspekti, näiteks kuidas süsteem jaguneb mooduliteks või kuidas süsteemi komponendid on võrgus jaotatud. Nii projekteerimise kui ka dokumentatsiooni jaoks on tavaliselt vaja esitada mitu arhitektuuri vaadet.

Arhitektuuri loomisel võib toetuda arhitektuuri mustritele (*architectural pattern*). Arhitektuuri muster on stiliseeritud hea projekteerimise tava kirjeldus, mida on proovitud ja katsetatud erinevates keskkondades. Mustreid saab kasutada teadmiste esitamiseks, jagamiseks ja taaskasutamiseks. Mustrid peaksid sisaldama teavet selle kohta, millal nad on ja millal pole kasulikud. Arhitektuuri mustreid võib esitada tabelina ja graafiliselt. Näiteid arhitektuuri mustrite kohta:

- Mudel-vaade-kontroller (*Model-View-Controller, MVC*) muster
- Kihiline arhitektuurimuster (*Layered architecture pattern*)
- Hoidla muster (*Repository pattern*)
- Klient-server arhitektuur (*Client-server architecture*)
- Toru-filter arhitektuur (*Pipe and filter architecture*)

- Hajutatud süsteemide arhitektuurimustrid (Architectural patterns for distributed systems).

Kuna ettevõtetel on palju ühist, on ka nende rakendussüsteemidel sagedasti ühine arhitektuur, mis kajastab rakenduse nõudeid. Üldine rakendusearhitektuur on teatud tüüpi tarkvarasüsteemide üldistatud arhitektuur, mida saab konfigureerida ja kohandada konkreetsetele nõuetele vastava süsteemi loomiseks. Sama rakendusarhitektuuri saab kujutada erinevate arhitektuurimustrite abil. Erinevate rakendusarhitektuuride näiteid:

- Tehingutöötluse rakendused (*Transaction processing applications*)
- Kihiline infosüsteemi arhitektuur (*Layered information system architecture*)
- Keele töötlemise süsteem (*Language processing system*)
- Andmete ühekordse küsimise tugiarhitektuur (*Once Only Reference architecture*)
- Tervishoiu tugiarhitektuur (*Healthcare Reference Architecture*)

Süsteemi esmane arhitektuur kavandatakse ka agiilse arenduse puhul üldjuhul arenduse varajastes etappides, sest selle muutmine on kulukas. Arhitektuuri edasine arendus võib toimuda kogu süsteemi elutsükli vältel ja see sõltub valitud elutsüklist.

Arhitektuuri projekteerimise tulemused võivad olla:

- Tuvastatud osapoolte probleemid
- Välja töötatud arhitektuuri seisukohad
- Määratletud süsteemi taust, piirid ja välised liidesed
- Välja töötatud süsteemi arhitektuurilised vaated ja mudelid
- Arhitektuuri komponentide jaoks formuleeritud olulised mõisted, omadused, omadused, käitumine, funktsioonid või piirangud
- Tuvastatud süsteemi elemendid ja nende liidesed
- Hinnatud arhitektuuri variandid
- Arendatud realiseeritavate protsesside arhitektuuriline alus kogu elutsükli jooksul
- Saavutatud arhitektuuri vastavus nõuetele ja kitsendustele
- Kättesaadavaks tehtud kõik arhitektuuri määratlemiseks vajalikud võimaldavad süsteemid või teenused

Arhitektuuri kirjeldamiseks võib kasutada:

- Lihtsaid blokkskeeme, mis esitavad olemeid ja nende vahelisi suhteid
- Arhitektuuri kirjeldamise keeli, nt *ArchiMate Enterprise Architecture Modeling Language*
- Muid vahendeid, mis võimaldavad iseloomustada süsteemi komponente ja nende vahelisi seoseid

Ettevõtte arhitektuuri arenduses võib kasutada TOGAF (The Open Group Architecture Framework) raamistiku ja selle arhitektuuri arendamise meetodi (Architecture Development Method, ADM) põhimõtteid ja komponente.

Tarkvara ülesehitusest annab esmase ülevaate lahenduse arhitektuur (*solution stack, https://en.wikipedia.org/wiki/Solution_stack).*

Küsimusi ja materjale

Kontrollküsimusi ja ülesandeid

- Tarkvara - võimalikud mõisted. Tarkvaratoode. Tarkvara, süsteem, teenus. Sihttarkvara ja sihiväline tarkvara. Sihtandmed ja sihivälised andmed. Keerukuse / koosvõime tasemed. Lahenduse arhitektuur (*solution stack*).
- Tarkvara arenduse tulemid
- Millal võib tarkvaraarenduse tulemitest rääkides piirduda vaid programmiga? Millisega?
- Tooge näiteid arhitektuuriliselt oluliste ja mitteoluliste nõuete kohta
- Tooge näiteid arhitektuuriliselt oluliste funktsionaalsete ja mittefunktsionaalsete nõuete kohta
- Millistel tasemetel võib arhitektuuri kavandada?
- Kuidas paikneb arhitektuuri kavandamine tarkvara elutsüklis?
- Milleks on vaja arhitektuurilisi vaateid?
- Mis on arhitektuuri mustrid?
- Tooge näiteid rakendusarhitektuuride kohta
- Mis võivad olla arhitektuuri projekteerimise tulemused?
- Kuidas kirjeldada arhitektuuri?

Materjale iseseisvaks tööks (näited)

- Ian Sommerville. Software Engineering. Tenth Edition. Addison-Wesley, Chapter 1, 6, 24
- Daniel Galin, Software Quality assurance from theory to implementation, Pearson - Addison-Wesley. Chapter 2.
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE. Chapter 13, Section 8; Chapter 2/3.
- Certified Tester Foundation Level Syllabus, ISTQB. Chapter 1.1.
- Software, https://en.wikipedia.org/wiki/Software
- TOGAF®, http://pubs.opengroup.org/architecture/togaf9-doc/arch/index.html
- The ArchiMate Enterprise Architecture Modeling Language, https://www.opengroup.org/archimate-forum/archimate-overview
- Robert C. Martin, Clean Architecture, Prentice-Hall. Chapter 22.

- ISO/IEC/IEEE 42010:2011(en). Systems and software engineering — Architecture description
- Zachman Framework, https://en.wikipedia.org/wiki/Zachman_Framework

5. Tarkvara kontroll: meetodid ja korraldus

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

5.1. Testimine

Igaüks, 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.

5.1.1. 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 vigu[2]. 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 omadusi[3]. Testimist võib laias mõttes määratleda ka kui protsessi, mis koosneb kõikidest elutsükli tegevustest (nii staatilistest kui ka dünaamilistest) mis puudutavad 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 defekte[4].

Testitakse tarkvara arenduse erinevatel etappidel ja süsteemiarenduse erinevatel tasemetel. Testimise meetodid nendel tasemetel võivad olla erinevad: näiteks üksuste testimisel (*unit testing*) ja süsteemi testimisel kasutatakse üldjuhul erinevaid meetodeid. Üks näide arenduse ja testimise tasemete vastavuse kohta on antud V-mudelis (vt ülal).

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).

Testjuhu sisendandmeid võib tekitada väga erinevalt, näiteks tuginedes nõuetele või programmi tekstile. Mitmete testimise meetodite eesmärk ongi testide jaoks selliste sisendandmete

---

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

[3] Standardist "ANSI/IEE Std 829 Standard for Software Test Documentation"

[4] International Software Testing Qualifications Board, ISTQB, http://istqb.org.

tekitamine, mis suurendavad vigade leidmise tõenäosust ja tänu sellele minimeerivad testimise mahtu.

Oodatud väljundid saadakse ülesande püstitusest, mitte programmist (ka programmi teksti põhise testimise puhul - miks?). Testimine on väga kokkuvõtlikult programmi käivitamine testi sisendandmetega ja tulemuste võrdlemine oodatud väljundandmetega.

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 süsteemi olemasolust. On vaja teada nõudeid ja protsesse, mis omakorda tulenevad ülesande püstitusest, organisatsioonist, seadusandlusest, standarditest, riskianalüüsist, headest tavadest, kasutajatest, kasutusviisist jne.

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?

On pakutud arvukalt tarkvara testimise ja arendamise 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 (suitsutestimine, *Hello world*).
2. Kas teeb kõige vajalikumaid asju?
3. Kas hädavajalikud väljundid on olemas?
4. Ekspertteadmiste põhised testid.
5. Kas läheb kergesti rivist välja?
6. Kas kõik komponendid toimivad õigesti?
7. Kas võimalikud sisendite tüübid toimivad igas valikus?
8. Kas süsteem teeb kõike, mida vaja?

9. Katsetame tööd piirjuhtudel.
10. Otsustustabelid - testide sisestamisel arvestada ka sisendite vahelisi sõltuvusi.
11. Kuidas töötab maksimaalsel koormusel?
12. Kasutame formaalseid meetodeid.
13. Ja nii edasi sõltuvalt vajadustest.

Valikud 1,2,3 testivad (minimaalset) funktsionaalsust, millest kasutaja kõige rohkem huvitatud võib olla. Valikuid 4,5 võib iseloomustada kui eksperdi testimist ja veaotsingut. Valikud 6,7 lähenevad testimisele rohkem programmi või süsteemi seisukohast; programmipõhist testimist vaadatakse järgmistes jaotistes. Valikud 8,9,10 (mingil määral ka 6,7) üritavad süstemaatilist läbitestimist, mille levinud liik on ekvivalentsiklasside ja piirjuhtude analüüs. Valik 11 on näide mittefunktsionaalsete nõuete testimisest.

Testimisele võib esitada vastuväiteid.

- Testimine ei tõesta niikuinii õigsust, parem juba tõestada programmi. - Programmide formaalne tõestamine on vastutusrikastel juhtudel vajalik, 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? - Mitte eriti süstemaatilised kontrollid ja hindamised (nt suitsutestimine, veaotsing ja uuriv testimine) on samuti testimise meetodid, kuigi ei kata üldjuhul põhjalikult kogu testitavat tarkvara. Vastutusrikkad ülesanded nõuavad tihti siiski süstemaatilisemat testimist. Sellise testimise tulemusena teame, et kõiki komponente on katsetatud teatud etteantud nivool, mis sõltub ressurssidest ja vajadustest. Mõni testimismeetod annab ka töökindluse ja vigade arvu prognoosi. Kõik need meetodid täiendavad teineteist.

Veel mõisteid (sisulised määratlused).

- Valideerimine - tegevus, mis püüab näidata, et tehtud on seda, mis vaja (kas tulemus vastab soovitule / kas tegime õiget asja?).
- Verifitseerimine - tegevus, mis püüab näidata, et järgmise etapi tulemus vastab eelnenud etapi määratlusele (kas spetsifitseeritud nõuded on täidetud / kas tegime õigesti?).
- Silumine - leitud vea kõrvaldamine.
- Sertifitseerimine - kolmanda osapoole tegevus, mis hindab toote, teenuse või protsessi vastavust standarditele ja normdokumentidele ja väljastab vastavuse korral (vastavus)sertifikaadi.

5.1.2. Ad hoc, suitsu-, eksperdi, uuriv, riskipõhine testimine

Enne kui hakata spetsifikatsiooni- või programmipõhiselt süstemaatiliselt testima (või koos sellise testimisega), võib olla otstarbekas kasutada mitmesuguseid vähem süstemaatilisi teste. Allpool on toodud mõned näited selleks sobivatest meetoditest.

Sellised meetodid kuuluvad tihti tarkvara arenduse metoodikate koosseisu ning neid saab kasutada lisaks süstemaatilistele testidele.

Neid meetodeid on mõtet kasutada muuhulgas 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.

Ad hoc ja suitsutestimine

Ad hoc testimisel testitakse nii, kuidas parasjagu parem tundub. Ei lähtuta etteantud meetoditest või kavadest, tulemusi võidakse mitte vormistada või analüüsida jne.

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

Ekspertteadmiste põhine testimine

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

- üldised teadmised
- teadmised konkreetse rakendusvaldkonna kohta
- teadmised riist- või tarkvarakeskkonna (näiteks konkreetse programmeerimiskeele) kohta
- teadmised mingi vigade tüübi kohta (näiteks tüüpilised infoturbega seotud kodeerimise vead)
- 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 tüüpvigade analüüs võib olla iseseisva (bakalaureuse-, magistri-) töö teema. Näiteks võib staatiliste meetodite jaotises toodud küsimustikke kasutada kui programmeerimiskeeltele orienteeritud veaotsingu metoodikat.

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

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

Testimine on loominguline tegevus ja intuitiivne ekspert on tavaliselt tulemuslikum kui algaja testija (tegutsegu algaja siis süstemaatiliselt või mitte). Intuitsiooni on kursusega raske edasi anda, see tekib mingil määral (loodetavasti) töö/õppimise käigus. Paljud selle kursuse kuulajad on eksperdid oma 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).

Riskipõhine testimine

Risk on määramatuse toime eesmärkidele (ISO/IEC 27005:2014). See toime võib olla positiivne või negatiivne kõrvalekalle oodatavast. Negatiivsete kõrvalekallete korral püütakse riske vähendada või vältida, positiivsete kõrvalekallete korral - säilitada või suurendada. Edaspidises räägitakse riskidest kui negatiivsetest kõrvalekalletest (see on tavaline käsitlus ja positiivsete kõrvalekallete puhul rakendatavad meetodid on tihti üsna erinevad).

Riskipõhise testimise idee on testida esmajärjekorras tootega seotud kriitilisi riske. Selleks on vaja välja selgitada riskid, omistada neile prioriteedid, testida kõige prioriteetsemaid 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 (kõik nendest 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 alati olla otseselt seotud põhifunktsionaalsusega.
- Tehnilised omadused, nt süsteemi uudsus ja keerukus, suured andmemahud, osapoolte arvukus jne.
- Arendusprotsessiga seotud riskid, nt kiired tähtajad, tellija või täitja ressursside vähesus jne.

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

Riskipõhine testimine on osa laiemast organisatsiooni IT riskihaldusest. IT riskihaldus on infotehnoloogilise süsteemi ressursse mõjutada võivate määramatute sündmuste tuvastuse, ohje ja välistamise või minimeerimise protsess tervikuna. Riskihalduse põhitegevused:

- Riskianalüüs (riskide tuvastamine, hindamine ja meetmeid vajavate alade tuvastus). Milline on vastuvõetav riskitase? Milline on olemasolev tase?
- Riskihalduse strateegia väljatöötamine. Kuidas katta vahe olemasoleva ja vajaliku taseme vahel?
- Riskide minimeerimine või välistamine. Meetmete rakendamine, et jõuda vajaliku tasemeni. "Riskihalduse 4 T": treat, transfer, tolerate, terminate.

Riskihalduseks ja riskianalüüsiks võib kasutada mitmeid standardeid (nt ISO/IEC 27005) ja metoodikaid (nt COBIT, FMEA).

5.1.3. Spetsifikatsiooni põhine testimine

Spetsifikatsiooni põhise testimise puhul me ei tea programmi sisemist ehitust või ei analüüsi seda. Testid koostatakse vaid nõuete põhjal. Kasutatakse ka nimetust "testimine musta kasti meetodil" ("*black box testing*" - programm kui "must / läbipaistmatu kast"). Selline testimine võib olla funktsionaalne (kasutatakse funktsionaalseid nõudeid) või mittefunktsionaalne (kasutatakse mittefunktsionaalseid nõudeid). Selles alajaotises vaatame funktsionaalset spetsifikatsiooni põhist testimist.

Võrdluseks, programmipõhise testimise puhul arvestame testide koostamisel programmi struktuuri - programm kui "valge / läbipaistev kast" ("*white box testing*"). Erinevus nende vahel on selles, mille põhjal leitakse testide sisendid (kas spetsifikatsiooni või programmi teksti põhjal); väljundid tekitatakse mõlemal juhul spetsifikatsiooni alusel.

Kahe eelmise vahepeal on testimise meetodid, mille puhul programmi sisemine ehitus on vaid osaliselt teada - nt algoritmid, andmestruktuurid, arhitektuur vms (programm kui "hall kast"). Nende meetodite korral saab olemasolevat infot testimisel kasutada, kuid see info ei ole nii põhjalik kui programmipõhise testimise puhul.

Spetsifikatsiooni võib testimiseks kasutada mitmeti, näiteks tehes mingi kriteeriumi järgi (sh riskid) olulisemaid teste spetsifikatsiooni põhjal, püüdes katta vähemalt igat spetsifikatsiooni komponenti (nt kasutusjuhtu), kattes ekvivalentsiklasse, kattes piirjuhte, kattes nii ekvivalentsiklasse kui ka piirjuhte jne.

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

1. Kogu sisendite ala jaotatakse klassideks (seejuures võib ühte klassi erandjuhul kuuluda vaid üks väärtus, nt piirjuhtude puhul), mille kohta eeldatakse, et süsteem käitub kõigi samasse klassi kuuluvate andmete testimisel ühtemoodi - kui annab vea ühtede andmete korral sellest klassist, siis annab vea kõikide andmete puhul sellest klassist ja kui ei anna viga mingite andmete korral, siis ei anna viga ka teiste andmetega sellest klassist. See lihtsustus võimaldab lõpmatu arvu sisendite asemel vaadelda lõplikku hulka klasse.
2. Testimisel püütakse katta kõik klassid. Kuna kõigi klasside kombineerimine võib ikkagi viia 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, olekuskeemide, jadadiagrammide ja mõnede teiste protsessile orienteeritud formalismide puhul saame testimisel kasutada ka struktuurse (programmipõhise) testimise meetodite analooge.

Ekvivalentsiklassid

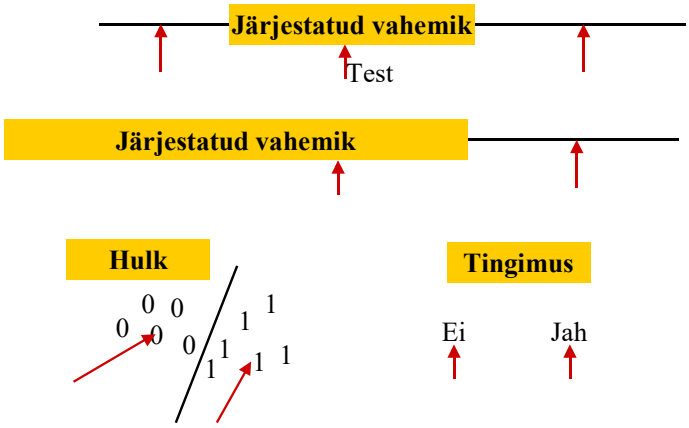
Ekvivalentsiklasside analüüsi idee on kasutada ära asjaolu, et sisendandmed jaotuvad töötluse suhtes enamasti rühmadesse, nii et ühes rühmas asuvaid andmeid töödeldakse ühtemoodi (vt. joonis).

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

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 on täidetud/täitmata)

Ekvivalentsiklassid (ka väljundile)



Ülaltoodud tegevused käisid ühe sisendi kohta. Kuna sisendeid võib olla palju, tuleb ekvivalentsiklassid koostada ja testandmed tuleb valida kõigi sisendite jaoks.

Testandmed kombineeritakse testidesse kõiki sisendeid arvesse võttes nii, et kõik valitud testandmed oleksid kaetud. Õigete / normaalse töö andmete erinevaid ekvivalentsiklasse püütakse ühte testi sisse panna korraga maksimaalsel arvul, 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?).

Mitmete ülesannete puhul on otstarbekas vaadelda sisendandmete vahelisi sõltuvusi - mingid spetsiifilised olukorrad tekivad teatava sisendandmete kombinatsiooni puhul. Sisuliselt on need kombinatsioonid sellisel juhul vaadeldavad ekvivalentsiklassidena.

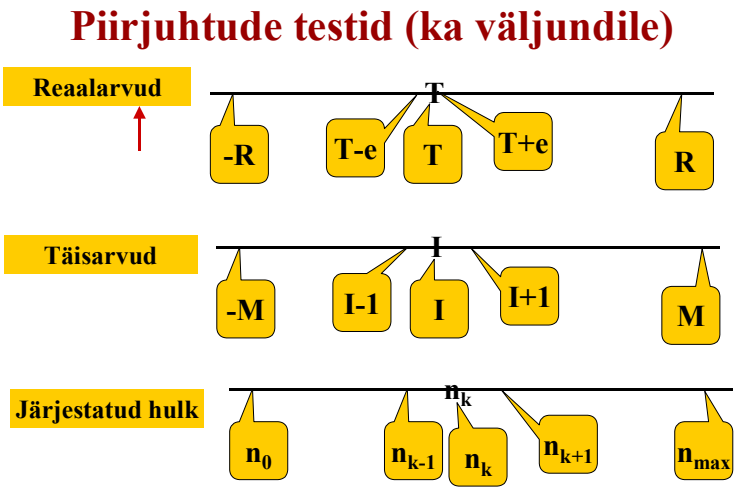
Ekvivalentsiklassid tekitatakse nii sisendite kui ka väljundite jaoks, püüdes kavandada neid klasse katvaid teste. Väljundite ekvivalentsiklasside puhul võib see mõnikord olla keerukas või võimatu.

Piirjuhud

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 järjestatud jada n_0, n_1, ..., n_K, ...n_M puhul, kus n_K on piirjuht, testitakse väärtusi n_0, n_1, n_{K-1}, n_K, n_{K+1}, n_{M-1}, n_M

- Kui sisendandmeteks on järjestamata hulgad, siis piirolukordi ei teki (miks?). Tavaliselt küll mingi järjestus leidub, iseasi kas see on oluline testitava ülesande seisukohast.



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

Ekvivalentsiklasside ja piirjuhtude põhise testimise analüüs ja etapid

Ühe sisendi korral, kus lubatavad väärtused on antud intervallina, tuleb siis kokku kolm ekvivalentsiklassi ja kaheksa piirolukorda. Kui sisendeid on palju ja nende ekvivalentsiklasse tuleb omavahel kombineerida, võib testide (vajalike sisendandmete kombinatsioonide) arv olla väga suur. Eespool vaatasime juba, kuidas vähendada testide üldarvu erinevate sisendite ekvivalentsiklasside omavahelise kombineerimise teel (valides õigete sisendandmete erinevaid klasse ühte testi maksimaalselt, veaolukordi ühekaupa). 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 veaolukordi võib esitada ühe ekvivalentsiklassina (kui jah, saab piirduda veaolukorra klassis ühtede testandmetega)?

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

Funktsionaalse testimise võib etappideks jagada järgmiselt:

1) eristada sisend- ja väljundandmete ekvivalentsiklassid, sealhulgas vajadusel 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.

Kuna piiripealsetes andmetes on tavaliselt rohkem vigu, on funktsionaalsetest meetoditest hinnaefektiivseim (vea leidmise maksumus on väikseim) piirjuhtude meetod, kuigi sellest enamasti ei piisa (miks?).

Kokkuvõte funktsionaalsest testimisest ekvivalentsiklasside ja piirjuhtude põhjal.

- Idee: süstemaatiline testimine sisendi/väljundi (spetsifikatsiooni, funktsionaalsuse) põhjal
- Eeltingimused: vajadus; spetsifikatsioon on mingil kujul olemas; selle analüüs on teostatav
- Eelised: kasutatav funktsionaalsus on süstemaatiliselt testitud; võib olla kasutajale arusaadavam kui programmi teksti põhine testimine; õigel arendamisel koostatakse testid juba spetsifikatsiooni koostamise ajal, mis võimaldab ühtlasi testida spetsifikatsiooni
- 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). Teste võib defineerida reeglite, eeltingimuste või tegevuste alusel.

Otsustustabeleid saab kasutada ka ekvivalentsiklasside ja piirjuhtude korral, eriti kui väljundid sõltuvad mingitest spetsiifilistest sisendandmete kombinatsioonidest. Kui selliseid sõltuvusi pole, kas siis on mõtet otsustustabelit kasutada?

Kasutusjuhud (kasutusmallid, use case)

Kui süsteem 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 osutuda vajalikuks teha ühe kasutusjuhu kohta arvukalt teste.

Kasutusjuhtude põhist testimist saab teha erineva põhjalikkusega, järgnevalt mõned näited.

- Iga kasutusjuhu põhistsenaariumi kohta tehakse vähemalt üks test.

- Iga kasutusjuhu põhistsenaariumi ja iga alternatiivse stsenaariumi tehakse vähemalt üks test.
- Kasutusjuhtude sisendid ja väljundid testitakse, kattes nii ekvivalentsiklassid kui ka piirjuhud.

Olekuskeemid

Testjuhtude koostamiseks saab kasutada väga erinevaid spetsifikatsioonivahendeid. Nii esitab olekuskeem (*state transition diagram*) süsteemi võimaliku oleku ja näitab sündmusi või asjaolusid, mis põhjustavad siirdumist ühest olekust teise või tulenevad sellest. Olekuskeemi (nt UML olekumasina, http://en.wikipedia.org/wiki/UML_state_machine) põhjal saab koostada teste erinevatest põhimõtetest lähtudes.

- Testidega kaetakse tüüpilised olekute järjestused.
- Testidega kaetakse kõik olekud.
- Testidega kaetakse kõik üleminekud ühest olekust teise.
- Testidega kaetakse ettemääratud olekute jadad.
- Testitakse lubamatuid üleminekuid.

5.1.4. Testimine programmi teksti põhjal

Programmi teksti põhise testimise puhul võetakse ette programm (nt lähtekood, struktuuri kirjeldus, liideste kirjeldused vms) ja püütakse luua teste selle põhjal. Inglise keeles kasutatakse selle meetodi puhul mitmesuguseid nimetusi, nt w*hite-box testing, glass box testing, transparent box testing, structural testing*. Eesti keeles on kasutatud veel vasteid "programmipõhine testimine", "lähtekoodipõhine testimine", "valge kasti testimine", "struktuuripõhine testimine" jm. Kõik need nimetused kajastavad seda, et me teame midagi programmi või süsteemi sisemise ehituse kohta ning kasutame seda teadmist testimisel.

Selline testimine on vajalik, sest ainuüksi funktsionaalsest testimisest ei piisa (samuti nagu ei piisa ainult programmipõhisest testimisest – mis jääb puudu?). Tõepoolest, vaadates programmi ainult väljastpoolt, programmi käitumise seisukohast, ei leia me suure tõenäosusega näiteks harusid, millele pole spetsifikatsioonis vastet või mida tavalisel täitmisel ei läbita.

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

Programmipõhist testimist saab kasutada erinevatel elutsükli etappidel (nt programmeerimine, sõltumatu testimine) ning erinevates protsessides (nt arendus, hooldus). Teadmist süsteemi struktuuri kohta saab kasutada integratsiooni- ja valiidsustestimisel. Programmeerimisel võib see meetod olla esmaselt rakendatav ja kõrge prioriteediga, näiteks testipõhise arenduse puhul. Hankija jaoks võivad muud meetodid olla odavamad ja prioriteetsemad ning programmipõhist testimist kui suhteliselt kulukat rakendatakse pigem suurema kriitilisusega tarkvaratoodete puhul.

Programmi teksti põhisel testimisel võib lähtuda juhtimisest või andmetest. Juhtimisest lähtuv tekstipõhine testimine püüab süstemaatiliselt läbida programmi mingeid osasid, näiteks lauseid,

harusid, teid. Vastavalt sellele, milliste osade läbimist nõutakse, eristatakse lause-, haru-, teeadekvaatsuse ja muid kriteeriume.

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

Kaasaegsel süsteemiarendusel võib olla palju erinevaid süsteemi kirjelduse tasemeid (nt vajadused, nõuded, arhitektuur, detailne disain, kood jm). Testide kavandamist võib vaadata kui protsessi, mis võtab sisendiks ühe või mitme sellise taseme kirjeldused ning annab väljundiks kavandatud testid. Spetsifikatsiooni- ja programmipõhise testimise erinevus ei ole sellise vaate puhul enam nii selge. Siiski on need kaks hästi arusaadavat testimise eriliiki, mida tasub õppida ja kasutada.

Kui programmipõhisest testimisest jääb väheseks, on järgmine programmi teksti kasutav tase formaalne tõestamine, mida vaadatakse allpool ja mille elemente kasutatakse toote kontrollimiseks puhta ruumi tarkvaraarenduses (cleanroom software development) ja infoturbe ühiskriteeriumide (Common Criteria) puhul.

Vaatame kõigepealt lauseadekvaatsuse kriteeriumit.

Lauseadekvaatsuse kriteerium

See kriteerium nõuab, et testimise tulemusena peab programmi 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 mõlemas harus olevaid lauseid.

Kokkuvõte lauseadekvaatsuse kriteeriumist

- Idee: kõik programmi osad on töötanud
- Eeltingimused: programmi tekst on olemas, seda saab analüüsida
- Eelised: programmi on süstemaatiliselt katsetatud. Vähe teste. Selge idee
- Puudused: mitte eriti tugev kriteerium. Ei anna andmete teste. Ei leia puuduvaid harusid. Programmi tekst pole alati kättesaadav. Mitte alati saavutatav
- Tulemused: testikomplekt, mis katab programmi
- Suhe teistesse: kasutada koos teiste meetoditega
- Vahendid: on olemas vahendid, mis mõõdavad programmi kaetust testidega või aitavad leida lauseadekvaatset testikomplekti, kui see on olemas

Haruadekvaatsus ja muud

Haruadekvaatsus

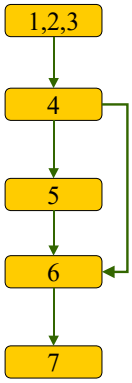
Lauseadekvaatsuse puhul läbitakse kõik laused, kuid harud, milles lauseid pole, jäetakse läbimata. Haruadekvaatsuse nõue eeldab ka tühjade harude läbimist, seega on ta põhjalikum, *Lauseadekvaatsus* $\preceq$ *Haruadekvaatsus*.

Haruadekvaatsust saab illustreerida programmi graafil. Sellel vastab igale hargnemisele graafi tipp, millest väljub rohkem kui üks kaar. Üksteisele järgnevad täidetavad hargnemiseta laused võib ühendada üheks tipuks. Haruadekvaatsuse nõuet võib sõnastada järgmiselt: testimise käigus peavad programmi graafi kõik kaared olema läbitud. Milline on lauseadekvaatsuse kriteeriumi sõnastus programmi graafil?

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 testi 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.

Haruadekvaatsus 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 näitajaid - McCabe programmi keerukuse mõõt. Näitajate ülesanne on midagi (antud juhul programmi) mõõta. Näitajad võimaldavad ka midagi hinnata, antud juhul 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, seades programmi keerukusele ülemise piiri

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 rühmast 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 teeadekvaatsuse kriteeriumit selliste programmide testimisel kasutada. Ilma tsükliteta programmi puhul võib see olla hea kriteerium.

Andmepõhine testimine

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

Tsüklite testimine

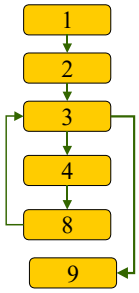
Tsüklite (korduste, iteratsioonide, silmuste) testimisel võib eristada testimist mingi testimiskriteeriumi või -meetodi põhjal ja tsükli kui eraldi konstruktsiooni testimist. Esimesel juhul koostame vajalikke teste valitud meetodite või kriteeriumide põhjal (näiteks haruadekvaatsuse kriteerium). Segadust 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 testide arvu hindamise vajadusest, kuid igal juhul tuleks selgitada, kuidas testide arv saadi.

Tsüklite testimiseks võivad eeltoodud meetodid olla nõrgad, sest vead võivad olla seotud tsükli läbimiste arvuga. Teisel juhul lähtumegi tsüklist kui eraldi konstruktsioonist ja testime 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 (ekvivalentsiklassi esindav väärtus - võimalusel arv, mis asub 2 ja n-1 vahel), 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 ,…., k taseme puhul: k-ndal tasemel tehakse ühekordse tsükli testid; seesmistel tasemetel kasutatakse m < n läbimise arvu; välised tasemed testitakse minimaalse läbimiste arvuga.

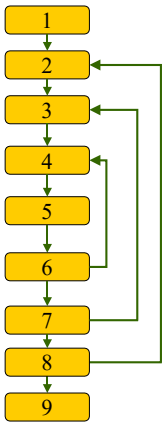
Mitmetasemeline tsükkel

```
…..
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 testimist muuta struktuurseiks.

5.1.5. Juhuslikud sisendid ja lisatud vead – MF nõuete testimine (1)

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 testida mittefunktsionaalseid nõudeid - tarkvara töökindlust, lubatud vigade arvu jne. Järgnevad meetodid 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 selle tarkvara erinevaid funktsioone kasutatakse?) ning testida vastavalt kasutusprofiilile
- tuleb profiili raames genereerida juhuslikke (matemaatiliselt juhuslikke, mitte testija poolt "juhuslikult" valitud) andmeid vastavalt profiili sagedustele
- tuleb lisaks juhuslikele testidele kasutada ka muid (vähemalt piirolukordade) teste
- tuleb testimist perioodiliselt korrata ja analüüsida, kuidas muutub leitud vigade arv ajas ning milliseid prognoose vigade arvu aegrea alusel saab teha

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 oodatavate tulemuste hindamine võib olla töömahukas
- olulised "erilised" andmete väärtused võivad jääda katsetamata, nt on vähe tõenäoline, et 0 tuleks juhusliku arvuna (selle vältimiseks tuleb kasutada piirolukordade teste)

Et seda meetodit efektiivselt rakendada, on võimalik kasutada süsteemi paralleelset realisatsiooni (eelmine versioon, prototüüp, matemaatiline mudel vms), mis võimaldab kiiresti leida oodatavaid väljundeid. Vastasel juhul võib testide koostamine 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 (see eeldus kehtib küll 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. Meetod 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 praktikas kasutatud 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?

5.1.6. Mittefunktsionaalsete nõuete testimine (2)

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.

Eelmises alajaotises on kaks testimise meetodit mittefunktsionaalsete nõuete testimiseks. Toome veel mõned näited sellise testimise kohta.

Kui nõue on testitavalt sõnastatud, viitab ta ise tihti sellele, kuidas testimist põhimõtteliselt läbi viia. Näiteks nõuet "Süsteemi töö võib kuu aja pikkuses ekspluatatsioonis 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 kontorisüsteemiga, võib seda üldjuhul teha (eeldades, et leiame ressursid, kasutajad, keskkonna jne), kui aga näiteks lennuki juhtimisega ja tegutsemisega avariiolukordades? Või kui ajavahemik on aastates? Sellistel juhtudel saab süsteemi hinnata pigem simulatsioonivahendite abil või rakendades staatilisi (süsteemi läbivaatuse jne) meetodeid, mida vaadatakse järgmises jaotises. Ka juhusliku testimise ja lisatud vigade meetodid võimaldavad testida töökindlust.

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

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

Kasutatavuse analüüs võib sisaldada mitmeid staatilise analüüsi komponente. Näiteks uuring, kas veebisaidi värvid on sobivalt valitud võib põhineda psühholoogiliste uuringute tulemustel, mitte otsestel testidel. Kasutatavuse kohta on tehtud mitmeid küsimustikke. Samas on küsimustikud tihti omavahel vasturääkivad. See on ka arusaadav – inimeste põhimõtted, kultuuriline taust, vajadused ja maitsed on erinevad.

Kasutatavus võib suurel määral sõltuda seadustest ja teistest regulatsioonidest – näiteks hõlpsuse (*accessibility*) nõue, et avaliku sektori veebisait vastaks puuetega inimeste vajadustele, on fikseeritud W3C juhendites (*Web Content Accessibility Guidelines,* https://www.w3.org/WAI/intro/wcag), standardis ISO/IEC 40500:2012 ning mitme riigi, sh USA

seadusandluses. Vastavust nendele nõuetele saab hinnata näiteks veebisaidil http://www.cynthiasays.com/.

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

Kasutatavuse 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.

A/B testimise korral püütakse hinnata, milline kahest realisatsioonist, näiteks kahest veebisaidist, annab parema tulemuse. Tavaliselt erinevad need realisatsioonid vaid väheste aspektide poolest, näiteks mingi elemendi paigutus, värv vms. Testimise tulemusena saadakse kasutajate eelistused selle konkreetse aspekti esituse kohta.

Üks tuntud autor veebi kasutatavuse teemal on Jakob Nielsen (http://www.useit.com/), kes soovitab 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 mittehomogeense kasutajaskonna puhul ei ole nii väike rühm piisav.

5.1.7. Testimise maht ja lõpetamine

Testimise resultatiivsust pole kerge hinnata, erinevalt näiteks programmeerimisest - testimine ei anna alati selgelt nähtavat tulemust ning kõiki vigu avastada pole üldjuhul võimalik. 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 testimise maht sõltuma kasutaja vajadustest ja tarkvarale esitatud nõuetest - testitakse seni, kuni need on rahuldatud. Praktiliselt tehakse nii kriitiliste rakenduste korral (või 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 eriti ei kannata) jne. Seega kasutatakse testimise mahu määramisel ja lõpetamise otsustamisel mõnikord 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 võivad sellised kriteeriumid olla 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 töömahud olla sama suurusjärku. Samas vastutusrikaste reaalajasüsteemide testimise maht võib olla tunduvalt suurem kui muude tegevuste mahud kokku.

Eelpool vaadatud (testimis)meetodid annavad lisavõimalusi testimise mahu määramiseks, näiteks:

- olulisemad riskid peavad olema testitud
- kõik funktsionaalsed ja mittefunktsionaalsed nõuded peavad olema mingis ulatuses testidega kaetud (nõudeid saab üldjuhul testida erineva põhjalikkusega)
- kõik ekvivalentsiklassid (piirjuhud) peavad olema testitud
- testimine peab vastama haruadekvaatsuse kriteeriumile (või X % harudest peab olema läbitud vms)
- 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 arenduse efektiivsust (samade kulutuste juures leitakse rohkem vigu), süstemaatilisust (väldivad olukorda, kus mingi süsteemi 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 korral 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 või kui palju ressursse testimisele tuleb kulutada. Sellist otsustust on lihtsam teha, ta sõltub näiteks vähem testimise objektist. Üheseid otsustuseeskirju niisuguseks otsustuseks ei ole, testimise korraldust käsitlevas peatükis anname mõned soovitused.

Seni vaadeldud meetodite hinnaefektiivsuse järjestus (selles mõttes, et vea leidmise maksumus on väikseim; alates efektiivseimast; jättes välja staatilised meetodid) on üldjuhul järgmine: programmeerija poolne testimine, suitsutestimine, kasutaja poolne testimine, riskipõhine testimine, uuriv testimine, ekspertteadmiste põhine testimine, piirjuhud, ekvivalentsiklassid, eraldi osapoole teostatud programmipõhine testimine, muud meetodid. Ühe meetodi kõrge

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.

Küsimusi ja materjale

Kontrollküsimusi ja ülesandeid.

- Mis on test, testimine, viga, hea test, edukas test, silumine, verifitseerimine, valideerimine, sertifitseerimine?
- Riskipõhine, ekspertteadmiste põhine, uuriv, suitsu- jm testimine
- Kuidas valida sisendeid? Kuidas hinnata väljundit? Millal testimine lõpetada? Kes on testijad? Missugune on testimise ja verifitseerimise vahekord?
- Mis määrab riskiastme? Kust otsida riske? Riskihalduse 4T
- Erinevus süstemaatiliste ja mittesüstemaatiliste testimise meetodite vahel
- Funktsionaalne testimine, erinevus programmipõhisest testimisest, ekvivalentsiklasside analüüs, piirolukorrad, 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 teeadekvaatsus garanteerib programmi korrektsuse? Anda hinnang programmi keerukusele. Kuidas toimub tsüklite testimine? Antud programm, pakkuda testid
- Andmepõhine testimine, testimine juhuslike andmetega, lisatud vead
- Kuidas testida mittefunktsionaalseid nõudeid? Mis on siin erinev funktsionaalsete nõuete testimisest? Tooge näiteid.
- Kuidas testida kasutatavust? Kas võib kasutatavuse testimisena vaadelda ka ekspertide või kasutajate poolset hääletamist (näiteks koduleheküljegede konkursse)?
- Testimise maht ja lõpetamine. Kuidas hinnata testimise meetodi efektiivsust, millised meetodid on hinnaefektiivsemad, kas kõige hinnaefektiivsemad meetodid on piisavad ja miks, mida kasutada kõigepealt ja mida teises järjekorras?
- Anda kokkuvõte meetoditest: idee, eeltingimused, eelised, puudused, tulemused, suhe teistesse, hinnang, vahendid

Ülesandeid.

- Spetsifikatsiooni põhjal tuleks teatavat sisendandmete vahemikku töödelda ühte moodi. Realiseeritud süsteemis töödeldakse osa sellest vahemikust ühte moodi ja teist osa teisiti, kusjuures väljund on kogu vahemikus õige. Kas see on viga? Kuidas selline olukord võib tekkida? Kas see tekitab raskusi funktsionaalsel testimisel?
- Millised on antud programmi puhul ekvivalentsiklassid, testitavad olukorrad, testid?
- Mitu testi on vaja antud programmi lause- (haru-, tee- jne) adekvaatseks testimiseks?

- Juhuslik katsetamine, mittesüstemaatilised testimismeetodid ja süstemaatiline testimine: eelised ja puudused. Millal mingit neist 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?"?

Materjale iseseisvaks tööks (näited)

- Ian Sommerville. Software Engineering. Tenth Edition. Addison-Wesley, Ch 8.
- Daniel Galin, Software Quality assurance from theory to implementation, Pearson - Addison-Wesley. Chapter 9.
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE. Chapter 4.
- Certified Tester Foundation Level Syllabus, ISTQB. Chapter 2.
- Christian Johansen, Test-Driven JavaScript Development. Addison-Wesley, 2010, http://it-ebooks.info/

5.2. Staatilised meetodid

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

Staatilistel meetoditel on mitu head omadust. Esiteks, nende puhul saab vigu leida juba enne programmi valmimist spetsifitseerimise ja projekteerimise 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 - analüüsitavus ja muud hooldatavuse kriteeriumid) on raske testimise teel hinnata, rohkem sobivad selleks staatilised meetodid. Viiendaks, võib paraneda inimeste vaheline kommunikatsioon, eri rühmade töötajad saaavad üksteisest ja eesmärkidest paremini aru, töötajaid on kergem asendada jne.

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 projektides.

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 saaks staatiliste meetodite alla liigitada ka programmide tõestamise ja infosüsteemi auditi. Oma eripära ja olulisuse tõttu on neid siiski vaadeldud edaspidistes jaotistes eraldi.

5.2.1. Läbivaatused ja hindamised

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

Autoripoolne analüüs

Autoripoolset analüüsi teeb enamik programmeerijaid ja arendusrühmi (firmasid). See on suhteliselt odav meetod, mis tavaliselt leiab vigu. Seega on meetodit soovitav kasutada. Autoripoolne analüüs on mitmete, eriti agiilsete, arendusmeetodite koostisosa. Kui seda veel ei tehta, on soovitav, et firma meetodit tutvustaks.

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

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

Läbivaatus

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

See on ka enamuse staatiliste meetodite eeldus, näiteks sisaldab Scrum enam kui viit erinevat tüüpi kohtumisi, mille jaoks on määratud eesmärgid, teemad, osavõtjad, maksimaalne kestvus jne.

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:

- vigu 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 väga hea viis vigade arvu vähendamiseks
- kontaktid rühmas võivad paraneda
- 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, läbivaatusi korraldatakse vaid 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 (sh 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 igaühel 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
- analüüsi tehtud läbivaatusi, et neid tulevikus paremini teha

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

Programmeerija hindamine

Meetodi idee on anda programmeerijaile arusaamine nende 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. Vastavalt hindamise eesmärkidele tuleks hinnata 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 toodete 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.

5.2.2. Küsimustikud ja tööriistad

Eespool oli juttu veaotsingust - meetodist, mille puhul kasutatakse ekspertteadmisi mingilt alalt (programmeerimiskeskkonnast, ainevaldkonnast…) tõenäoliste vigade kiireks leidmiseks. Programmeerimiskeelte küsimustikud pakuvad võimalike vigade ideid, mida võib kasutada autor, läbivaataja, analüüsija jne. Küsimustikud sisaldavad momente, millele muidu võib-olla ei osata 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). Muuhulgas, küsimustikke saab koostada disaini mustrite põhjal ja standardeid võib kasutada kui laiaulatuslikke ja üldisi küsimustikke.

Näide: programmeerimise küsimustik

Allpool on küsimustiku näitena toodud lihtne 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.

<u>Andmete kirjeldamine</u>

Kas muutujad on ilmutatult kirjeldatud (sõltub keelest, paljud translaatorid kontrollivad seda)?

Kas vaikimisi atribuudid on antud õigesti?

Kas muutujad on õigesti initsialiseeritud?

Kas muutujad on sarnaste või segadusse ajavate nimetustega, näiteks VOLT ja VOLTS, I1 ja O0?

<u>Pöördumine andmete poole</u>

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 tüüpidega?

Kas kasutatakse koos 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 võrdlustehte tulemus sõltub kompilaatorist?

Kas võrreldakse reaalarvulist muutujat mingi kindla arvuga?

Juhtimise vead

Kas igale BEGIN-lausele vastab END?

Kas programm või moodul lõpetab töö?

Kas tsükkel või kordus lõpetab töö?

Mis juhtub, kui tsükli tingimus on kohe vale?

Kui FOR alumine raja on suurem kui ülemine, kas siis tulemus vastab oodatule?

Kas on vaadatud kõrvalekaldumisi (nt 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-/ väljundpuhvrite 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 ebapiisavalt testitud

"Nähtamatud kasutajad" ja spetsifitseerimata käitumine

Lisaks inimkasutajale suhtleb programm tüüpiliselt paljude "nähtamatute kasutajatega" (operatsioonisüsteem, API-liides, failisüsteem jne). Kas on uuritud, mis juhtub sellise suhtluse probleemide korral?

Siia kuuluvad ka mitmed turvalise programmeerimise teemad. 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 nõudele "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, kusjuures hindamiste tulemused tuleb dokumenteerida:

- tarkvaraelemendi jälitatavus nõuete ja lahenduseni,
- väline kooskõla tarkvaraelemendile esitatud nõuete ja lahendusega,
- sisemine kooskõla üksusenõuete vahel,
- üksuste kaetus testimisega,
- kasutatud kodeerimismeetodite ja -standardite sobivus,
- tarkvara integreerimise ja testimise teostatavus,
- käituse ja hoolduse teostatavus.

Seda punkti saaks 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 üksusenõuete vahel?
- üksuste kaetust testimisega?
- kasutatud kodeerimismeetodite ja -standardite sobivust?
- tarkvara integreerimise ja testimise teostatavust?
- käituse ja hoolduse teostatavust?"

Tööriistad

Tarkvara staatiliseks analüüsiks võib kasutada mitmesuguse funktsionaalsusega tööriistu.

- Üldotstarbelised koodi analüsaatorid paljude programmeerimiskeelte jaoks (nt Java, .NET, JavaScript, Python).
- Koodi analüüs turvaaukude, nõrkuste jm infoturbe nõuetele mittevastavuste suhtes.
- Kasutamise hõlpsuse (accessibility) analüüs ja hindamine.
- Disaini verifitseerimine, nt vasturääkivuste või soovitatud disainimustrite suhtes.
- Jõudluse probleemide analüüs.
- Tõestamise ja verifitseerimise tugi.
- Analüüsi, disaini jm tegevuste korraldus.

Küsimusi ja materjale

Kontrollküsimusi ja ülesandeid

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

Materjale iseseisvaks tööks (näited)

- Ian Sommerville. Software Engineering. Tenth Edition. Addison-Wesley, Ch 24.
- Daniel Galin, Software Quality assurance from theory to implementation, Pearson - Addison-Wesley. Chapters 5,8.

- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE. Chapter 10 Section 2.3.
- Robert C. Martin, Clean Code, Prentice-Hall. Chapter 17.
- Certified Tester Foundation Level Syllabus, ISTQB. Chapter 3.

5.3. Töökindluse oluline suurendamine ja selle rakendusi

5.3.1. Töökindluse parendamise vahendeid - ülevaade

Mõnes valdkonnas nõutakse suurt töökindlust, näiteks USA *Federal Aviation Adminstration* veakindluse nõue on antud lennukitüübi jaoks tüüpilisel lennul 10^{-9} viga tunnis (ehk üks viga saja tuhande aasta jooksul). Kuidas seda saavutada ja kontrollida - süsteemi ei saa ju sada tuhat 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 arenduse ja halduse meetodid (sh *Cleanroom development, Common Criteria*, ka ISKE mõned moodulid).

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

5.3.2. 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 õigeks (hääletamine).

Matemaatiliselt võib tõestada, et põhimõtteliselt saab seda tüüpi meetoditega – rakendades paralleelseid seadmeid ning kasutades hääletamist, vajadusel hierarhiliselt - 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.

5.3.3. 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. Seejuures on võimalik, et

süsteemi normaalse funktsionaalsuse mittetoimimine ei ole "suur viga, mida tahetakse vältida" (millal?)

5.3.4. Formaalne verifitseerimine/tõestamine

Ajalooliselt tekkisid tõestusmeetodid 1960-tel 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) hea projekteerimine, staatilised meetodid, testimine 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. Lühikokkuvõte tõestamisest:

- Siht. Näidata, et programm vastab spetsifikatsioonile
- Idee. Luuakse loogiline arvutus (valemid, aksioomid, tuletusreeglid, tõestused, teoreemid). Selle arvutuse terminites kirjeldatakse spetsifikatsioon (sisendid ja väljundid) ning programm. Tõestatakse, et lähtudes antud sisenditest ja kasutades antud programmi jõutakse nõutud väljunditeni (tavaliselt ka, et programm lõpetab töö)
- Eeltingimused. Spetsifikatsioon, programm, loogikavahendid, vajadus, ressursid, soovitavalt toetav tarkvara
- Eelised. Suurem töökindlus, formaalne korrektsus, ainuke viis tsüklite korrektsuse formaalseks põhjendamiseks
- Puudused. Töömahukas, sobib halvasti suurte süsteemide jaoks, ei välista spetsifikatsiooni vigu (samuti arenduskeskkonna ja muid vigu), tsükleid on tehniliselt raske tõestada
- Tulemused. Programm tõestatakse spetsifikatsiooni suhtes
- Suhe teistesse. Kasutatakse koos teiste meetoditega
- Hinnang. Kasutada vajadusel. Vähekriitiliste süsteemide puhul pole otstarbekas
- Vahendid. On tehtud tõestamist toetavaid tarkvaravahendeid, kuid need pole laialt 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 ebapiisav, pole ühe etapi tulemuste tõestamisest palju kasu. Illustreerime seda järgneva tabeliga.

Tõestatav objekt	**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. Erijuhus - 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
Programmi täitmine	Mikroprogrammi/ riistvara tõestamine		Ühekordne töö
Programmi kasutamine	Tootmis-keskkonna meetmed	Väga erinevad, üldjuhul ei saa tõestada	

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://www-cs-faculty.stanford.edu/~uno/faq.html).

5.3.5. Tarkvaraarendus puhtas/kontrollitud keskkonnas

Üks komplekssetest metoodikatest, 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 korrektsust 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, korrektsuse ja arusaadavuse paranemine. Hinnangud töökindluse ning vigade arvu kohta.

Kasutajad (näited): suured tarkvara- ja telekommunikatsioonifirmad, õhujõud, maismaa väed.

Suhe teiste meetoditega: kasutatakse testimist juhuslike andmetega, statistilist analüüsi, koodi tõestamist. Tarkvaraarendus puhtas/kontrollitud keskkonnas ja tarkvaraarenduse küpsusmudelid täiendavad üksteist.

Vahendid: Toetavad süsteemid ja keskkonnad on olemas.

5.3.6. 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. Metoodiliselt testitud ja kontrollitud (lisaks eelmistele tuleb nt rahuldada spetsifikatsiooni adekvaatsuse/katvuse kriteeriumid ning testida disaini)
4. Metoodiliselt 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/.

5.3.7. 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

Küsimusi ja materjale

- Saavutatav töökindluse tase seniste meetoditega, selle olulise suurendamise võimalusi
- Teha kokkuvõte meetoditest: idee, eeltingimused, eelised, puudused, tulemused, suhe teistesse, hinnang, vahendid
- Arenduse dubleerimine /N-versiooniline programmeerimine
- 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
- Töökindluse parandamise organisatsioonilised ja tehnilised meetodid
- Võrdlus: töökindluse saavutamine riist- ja tarkvara puhul

Materjale iseseisvaks tööks (näited)

- N-version programming, http://www.cse.cuhk.edu.hk/~lyu/book/sft/pdf/chap2.pdf
- Fault tree analysis https://en.wikipedia.org/wiki/Fault_tree_analysis
- Program verification https://en.wikipedia.org/wiki/Formal_verification#Software
- Program proof example: http://homepage.cs.uiowa.edu/~fleck/181content/seqproof.pdf, http://homepage.cs.uiowa.edu/~fleck/181content/assignax.pdf, http://homepage.cs.uiowa.edu/~fleck/181content/assignax.pdf
- Cleanroom Software Engineering ftp://ftp.sei.cmu.edu/pub/documents/96.reports/pdf/tr022.96.pdf
- *ISO/IEC 15408. Evaluation Criteria for IT security* http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

5.4. 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.

5.4.1. 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 arendaja vahel järgmine suhtlus:

- tellija annab tellimuse
- programmeerija loob tarkvara, teeb esialgse testimise ja 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 eespool 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 kallis - 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, programmeerijate ja testijate 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?)
- väga palju erinevaid kasutajaid, erinevad riistvara- ja tarkvaraplatvormid, ajakriitiline arendus – koormus- ja porditavuse testimise suur osakaal, etapiviisiline arendus ja testimine
- ja nii edasi

Need tegurid esinevad tihti koos (püüdke tuua näiteid). Mõnel juhul piisab eelmises jaotises toodud korraldusskeemide detailiseerimisest. Allpool vaadeldakse keerukamaid skeeme.

5.4.2. 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 (see järjestus on ligikaudne ja võib konkreetsetes projektides olla erinev):

- arendaja poolne esmane testimine arenduse käigus, on mitmete arendusmetoodikate põhikomponent
- suitsutestimine (väga kiire esmane testimine)
- riskipõhine testimine (katsetab kõige kriitilisemaid omadusi)
- läbivaatused (avastavad vigu vara)
- testimine kasutaja andmetega (peavad kindlasti töötama), selleks võib kasutada ka normaalse töö ekvivalentsiklasside teste
- uuriv testimine
- vea otsing (kui testija on ekspert)
- testimise automatiseerimine (mõne arendusmeeetodi puhul kuulub arendusprotsessi)
- mittefunktsionaalsete nõuete testimine (erineva raskusega meetodid)
- piirjuhud (vigade kuhjumise kohad)
- ekvivalentsiklassid (sealhulgas veaolukorrad)
- välise osapoole poolt läbi viidud programmipõhine testimine

- testimine juhuslike andmetega
- lisatud vead
- tõestamine

Järgnevas tabelis on soovitused kontrolli meetodi valikuks, lähtudes rakenduse töökindluse ja kriitilisuse nõuetest. Üldine loogika põhineb meetodite efektiivsuse ja maksumuse hinnangul; eriti nõrgemate nõuete 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**	**Soovitatavad 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, riskipõhine testimine, piirjuhud, testimise automatiseerimine
Keskmine	Märgatav rahaline kahju, ebamugavused	+ üldised kvaliteedihalduse ja töökorralduse meetodid, kvaliteedi ja protsessihalduse standardite kasutamine, mittefunktsionaalsete nõuete testimine, küsimustikud, funktsionaalne testimine, haruadekvaatsus, andmepõhine testimine
Kõrge	Suured rahalised kahjud, firma võimalik häving, vigastused, keskkonnareostus	+ muud programmipõhised meetodid, testimine juhuslike andmetega, kvaliteedinäitajad, veapuu analüüs
Väga kõrge	Inimelud, paljusid mõjutav finantskahju, suur keskkonnareostus	+ tõestamine, dubleerimine, *Cleanroom software engineering*, muud meetodid

5.4.3. Arendusprotsessi integreeritud kontroll

Otstarbekas on alustada kontrolli ja testimisega võimalikult varakult - mida varem vead leitakse, seda odavam on neid parandada. Seepärast on hinnaefektiivne projekteerida iga arendusetapi käigus ka testid. Lihtsustatult võib öelda, et arendusetappidele vastavad eri tüüpi testid. Seda saab teha igasuguste tarkvara elutsükli mudelite korral, nt agiilses arenduses, V-mudeli ning kosemudeli korral jne. Üks levinum skeem on järgmine:

- süsteemi üldprojektile (sealhulgas tarkvara) vastab süsteemitestimine
- tarkvara nõuete spetsifikatsioonile vastab valideerimistestimine

- tarkvara realisatsiooniprojektile vastab integratsioonitestimine
- tarkvara kodeerimisele vastab mooduli testimine (üksuste testimine)

Tegemist on nelja erineva testimisetapiga, mis vastavad neljale süsteemiarenduse olulisele tegevusele. Igal arendusetapil luuakse oma testimise metoodika või testiprojekt (see võib olla 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.

Vaatame testimise etappe lähemalt, alustades madalama taseme testidest.

Mooduli testimine

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

Integratsioonitestimine

Moodulid tuleb integreerida 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 reaalset käitumist imiteerivaid objekte (*mock objects*), milles on realiseeritud vaid teatud komponendi omadused – näiteks kasutajaliides.

Alt üles integreerimise puhul asendatakse osa ülemise taseme väljakutsuvaid mooduleid nn draiveritega. Eeliseks on see, et korraga saab töötada palju inimesi (rühmi), 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

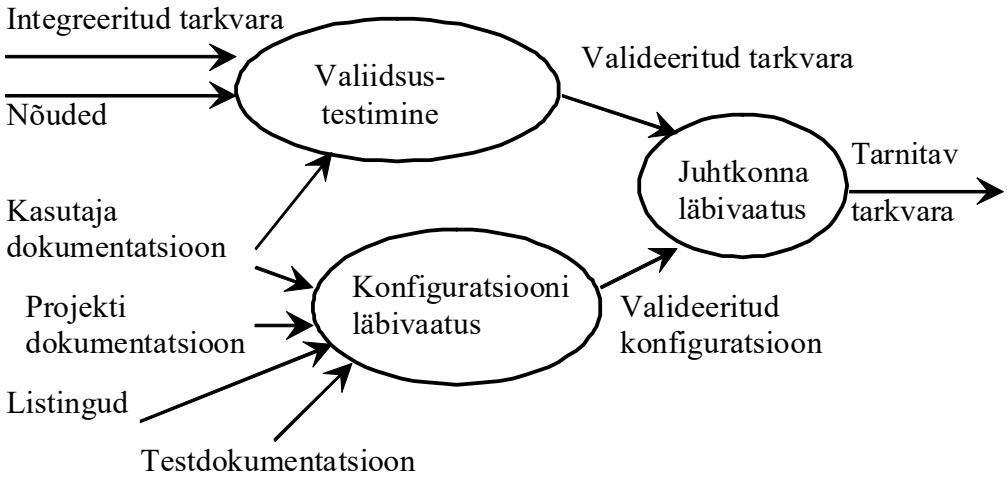
Ülalt alla integreerimistestide puhul kirjutatakse väljakutsutavate moodulite asemel nn lühised (asendavad programmid, *mock objects, stubs*; näiteid: prindi mooduli nimi; prindi üleantav parameeter; tagasta vastusparameeter; otsi tabelist vastus ja tagasta väljund vastavalt sisendile). Testimise käik: 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 - vastavalt projektile?" (verifitseerimistest, verifitseerimine). Siiski võib juba projekt 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 mingi süsteemi koosseisu. Tarkvara jaoks spetsifitseeritud nõuded 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; kauba tarne korrektsus ja aeg).

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 raskem (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 suure laeva või lennuki avariisüsteemide teatud omadusi. 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.

5.4.4. 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 võimalikud artefaktid (tegelik vajadus sõltub olukorrast):

- *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*

5.4.5. 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:

1. *Find out what you have to do.*
2. *Write a unit test for the desired new capability. Pick the smallest increment of new capability you can think of.*
3. *Run the unit test. If it succeeds, you're done. Go to step 1, or if you are completely finished, go home.*
4. *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.*

5.4.6. Testimisprotsessi standardid ja täiustamine (TPI metoodika)

Eelpooltoodud metoodikad (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 käsitletavad metoodikad 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 metoodika, mis on avaldatud raamatutes (Koomen and Pol, 1999; Sogeti, 2009). Raamatud annavad praktilisi juhendeid 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 võtmealadeks (TPI - 20, TPI NEXT - 16 võtmeala). Võtmealad on näiteks testimise strateegia, testide spetsifitseerimise meetodid, testimise vahendid, näitajad ja nii edasi. Et jälgida organisatsiooni küpsuse taset, on iga võtmealaga seostatud selle tasemed ja kontrollküsimused. Mudel sisaldab ka soovitusi ja instruktsioone 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.

Ettevõte, kes soovib parendada oma tooteid ja teenuseid, peaks kaaluma, kas TPI mudel eraldi on piisav seatud sihtide saavutamiseks. Mudel keskendub testimisele, mis on üks paljudest olulistest protsessidest (või osa protsessist). Näiteks esitab rahvusvaheline standard EVS-ISO/IEC 12207 kokku nelikümmend kolm süsteemikonteksti ja tarkvaraspetsiifilist protsessi. Kindlasti saab TPI mudelit kasutada kasuliku täiendusena ja täpsustusena sellele ja teistele laiaulatuslikele standarditele.

On kättesaadavad nende metoodikate järgmised versioonid (TPI®NEXT AND TMAP®NEXT). TMAP NEXT on metoodika testimise korraldamiseks ning TPI NEXT - raamistik testimise protsessi edasiarendamiseks.

5.4.7. 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, Scrum, testipõhine arendus, Cleanroom development jne) hädavajalik komponent.

Testimise automatiseerimise vahendi üks näide on JUnit (http://junit.org) koos arvukate xUnit modifikatsioonidega.

Kontrolli automatiseerimiseks on mitmeid võimalusi ja tarkvaravahendeid:

- Ühiktestimise vahendid, nt xUnit raamistikud erinevate keelte jaoks
- 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
- testide genereerimise automatiseerimine: näiteks, on vahendeid, mis genereerivad automaatselt lause- või haruadekvaatsed testid, samuti vahendeid, mis genereerivad teste mitmesuguste funktsionaalsete kirjelduste põhjal
- mitmesugused vahendid testimise lihtsustamiseks: näiteks tarkvara, mis lihtsustab draiverite (*mock objects*) või lühiste (*stub*) loomist integratsioonitestimisel, juhuslike sisendite või vigade generaatorid ja nii edasi
- vahendid testimise kvaliteedi hindamiseks, näiteks kontrollimaks, millised programmi komponendid on testidega läbimata
- staatilise analüüsi vahendid, näiteks programmi näitajate hindamise tarkvara
- vigade, paranduste, intsidentide, probleemide haldamise abivahendid
- vahendid 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 lisaressursse järgnevaks:

- 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, võib sellise tegevuse hinnaefektiivsus olla madal)
- 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 välja arendatud - 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 - nad võivad kaasa tuua palju lisatööd, eriti muutuva tarkvara või selle keskkonna puhul.

Automatiseeritud vahendite näitena on kursuse praktikumides vaadatud mitmeid süsteeme (vt kursuse materjalid veebis).

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, käsitletavad 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 25010 kvaliteediatribuutide skeemi. Lihtsamate saitide puhul testitakse tüüpiliselt funktsionaalsust, töökindlust, efektiivsust ja kasutatavust, sealhulgas järgmisi omadusi:

- kasutatavus, eriti kasutusmugavus
- funktsionaalsus, 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 - komplekssed laadimis- ja koormustestid, nagu näiteks testimine teatud arvu samaaegsete kasutajatega
- koostalitlusvõime – näiteks töötamine erinevate brauseritega

Veebisaitide või veebiliideste testimiseks on palju vahendeid, 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 arenduskeskkonda sisseehitatud testimisvahendid võivad anda häid tulemusi. Sisseehitatud vahend on käepärasem, kui sait on ka loodud sama vahendiga. Võib siiski osutuda, 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).

Testimisvahend	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
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
Selenium	Üld-otstarbe-line	https://www.selenium.dev/	Veebirakenduste testimisvahend
Apache JMeter	Koormus-testimine	http://jakarta.apache.org/jmeter/	Palju komponente

5.4.8. Pidev testimine ja arendus

Testimise ja laiemalt kvaliteedihalduse ülesanded muutuvad keerukamaks kaasaegsete teenus-orienteeritud arhitektuuride puhul, kus süsteem on pidevas muutumises, võivad olla kehtestatud teenustaseme lepped ja nõutav töökindluse tase võib olla väga kõrge.

DevOps on selliste süsteemide puhul kasutatav metoodikate, praktikate ja tööriistade karkass. Ta soodustab tarkvarasüsteemi arendajate, haldajate ja kasutajate pidevat koostööd ja kommunikatsiooni ning võimaldab oluliselt kiirendada muudatuste tegemist ja uute reliiside paigaldust (https://en.wikipedia.org/wiki/DevOps).

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. Sellisel juhul on pakutud põhimõtet „testitakse kõike, kõik testivad, testitakse pidevalt", mis nõuab testimise automatiseerimist ja kõikide osapoolte poolt teostatavate testide pidevat täitmist ja haldamist. Selline arenduslaad ei pruugi olla sobiv kõrgete töökindlusnõuete puhul.

Küsimusi ja materjale

Kontrollküsimusi ja ülesandeid

- Iseloomustage tarkvara kontrolli korralduse lihtsamaid skeeme. Milline neist on parim? Millal tekib vajadus keerukamaks korralduseks?
- Kontrolli meetodite efektiivsuse võrdlus. Millises järjekorras kasutada?
- Iseloomustage kontrolli korralduse loogikat: olukord ja testitavad objektid, meetodid, tegevused (valdkonnad)
- Arendusprotsessi integreeritud kontroll erinevate tarkvara elutsükli protsesside puhul
- Testimise etapid: ühiku-/mooduli-, integratsiooni-, valideerimise- ja süsteemitestid
- Mida teha, kui testida ei saa?
- Võrdlus: lihtsad skeemid, V-mudel, RUP ja testimine, agiilne testimine, XP, TPI mudel
- Tooge näiteid erinevatest missioonidest sama arenduse erinevatel iteratsioonidel.
- Millised rollid võiksid vastutada milliste testimise tulemite eest?
- TPI või TPI NEXT metoodika 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? Oskus kasutada vähemalt kahte testimise automatiseerimise vahendit
- Veebi testimise kriteeriumid, vahendite liigitus, vahendite näited
- Kuidas areneb testimine / testimise automatiseerimine uutes hajus- / teenus-orienteeritud arhitektuurides?
- Kokkuvõte meetoditest: idee, eeltingimused, eelised, puudused, tulemused, suhe teistesse, hinnang, vahendid

Materjale iseseisvaks tööks (näited)

- Ian Sommerville. Software Engineering. Tenth Edition. Addison-Wesley, Chapter 8
- Daniel Galin, Software Quality assurance from theory to implementation, Pearson - Addison-Wesley. Chapter 10.
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE. Chapter 4 Sections 5,6.
- Certified Tester Foundation Level Syllabus, ISTQB. Chapter 5 and 6.
- Test Process Improvement - https://www.ict-books.com/topics/testing
- Bach J. Test Automation Snake Oil, v2.1. http://www.satisfice.com/articles/test_automation_snake_oil.pdf

6. 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" võib olla 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 tarkvaraprotsesside kvaliteeti?
- Kuidas kvaliteeti mõõta?
- Kuidas kasutada standardites olevat teadmist?
- Kuidas saada sõltumatut toetust ja hinnangut tarkvara arendusele?

6.1. Kvaliteedihaldus

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

6.1.1. 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. Kvaliteeti luuakse inimeste jaoks ja nende koostöös. Seega võib öelda, et kvaliteet on toote/teenuse, nõuete, protsesside ja osapoolte vaheline suhe; siia lisanduvad ressursid, vahendid jne.

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.

Toode/teenus: tarkvara arenduse tulem hõlmab mitmesuguseid komponente, mis kõik võivad olla kvaliteedihalduse objektid. Neid vaadeldi eespool, käesoleva materjali jaotises 4.

Nõudeid ja vajadusi 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 jaotises 3. Nõuded võivad olla orienteeritud kõikidele süsteemiarenduse tulemitele, arendusprotsessile, muudele protsessidele jne.

Protsessid: infotööga seotud tegevusi on parema haldamise huvides mõistlik jaotada (alam)protsessideks ja etappideks. Paljudes, sealhulgas agiilsetes arendusprotsessides eristatakse selliseid tegevusi nagu spetsifitseerimine, analüüs, disain, realiseerimine jne. Etappide sisu, struktureerimine (nt iteratiivne, lineaarne, V-kujuline vms) ja nimetused võivad mudeliti erineda. Antud teema jaoks on oluline, et sellised etapid on olemas ja et igale etapile peaksid vastama kindlad tegevused. Jaotises 2 kirjeldati mõningaid tarkvara protsesse 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 (nt kvaliteedipoliitika kujundamine). Toome järgnevalt mõned kvaliteedihalduse mõisted.

Kvaliteedipoliitika - organisatsiooni tippjuhtkonna ametlikult esitatud kvaliteedialased üldeesmärgid ja juhtnöörid.

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.

6.1.2. Miks levib ebakvaliteetne (laiatarbe)tarkvara?

Võib küsida, et kui kvaliteedil ja kvaliteetsel tootel on eelised, miks siis on nii palju ebakvaliteetset (näiteks ebaturvalist) 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 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 (Metcalfe seadus – võrgu väärtus on võrdeline 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 raskem kvaliteeti efektiivselt tootesse sisse viia.
- Asümmeetrilise informatsiooni tingimustes ei pruugi turg olla efektiivne (http://en.wikipedia.org/wiki/The_Market_for_Lemons). Tarkvara puhul - kuna

kasutajatel võib olla 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 võib olla 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.

6.1.3. Kvaliteedihalduse mitteformaalseid meetodeid

USA, Euroopa ning Jaapani arusaamad kvaliteedist on olnud mõnevõrra erinevad. Viimastel aastakümnetel on need ühtlustunud. USA ja Euroopa kvaliteedihaldus on minevikus olnud rohkem orienteeritud tulemusele - oluline on lõpptulemuse, vähem protsessi kvaliteet. Paraku ei taha see põhimõte keerukate toodete puhul hästi toimida, nagu ülal põhjendatud. Jaapanis on kvaliteedihaldus olnud 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*).

Steve Jobs ütleb ühes oma kvaliteedi teemalises intervjuus[5]: „*The things that we've learned most from Dr Juran are to look at everything as a repetitive process, to instrument that process and find out how it is running, and then start to take it apart and put it back again in ways that dramatically improve its effectiveness ... a very straightforward way, no magic…*". See võib olla väga hea soovitus nii tööalases kui ka igapäevases kvaliteedihalduses.

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.

[5] http://www.youtube.com/watch?v=XbkMcvnNq3g

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.

6.1.4. Kvaliteedi hindamise süsteeme ja auhindu

Kvaliteedi hindamiseks on loodud mitmesuguseid süsteeme. Selliste süsteemide alla võib paigutada ka kvaliteedi konkursid ja auhinnad, näiteks Euroopas EFQM (*EFQM Excellence Award*), Eesti Kvaliteediühingu tunnustused (https://www.eaq.ee/tunnustused/) ja USA-s Malcolm Baldrige auhinna (*Malcolm Baldrige National Quality Award*).

Malcolm Baldridge auhinna määramisel arvestatakse tulemuslikkust viies põhivaldkonnas:

1) Toote ja protsessi tulemused
2) Kliendi tulemused
3) Tööjõu tulemused
4) Juhtimise ja valitsemise tulemused
5) Finants - ja turutulemused

6.1.5. ISO 9000 seeria standardid

Sertifitseerimine ISO 9001 (hetkel enamasti ISO 9001:2015) alusel on Eesti ettevõtete, sealhulgas IT ettevõtete poolt kõige rohkem kasutatav (www.eaq.ee). Lisaks ettevõtte töö korrastamisele ja efektiivsemaks muutmisele võib see anda konkurentsieelise rahvusvahelisel turul.

Senised kogemused näitavad, et ISO 9001 kasutamine ja selle põhjal sertifitseerimine on tulemuslik, kui põhirõhk on töö tulemuslikkuse ja kvaliteedi parandamisel, mitte vaid sertifikaadil. See tähendab, et tuleb kasutada (äri)loogikat, üldisi kvaliteedijuhtimise põhimõtteid ja tervet mõistust, arvestades ka ISO 9001 nõudeid - mitte vastupidi. Juhtkond ja kogu

organisatsioon peab olema kaasa haaratud. Süsteem peab tooma reaalset kasu, tuleb mõõta tulemusi ja hinnata tasuvust. Vastupidi - kui esmane eesmärk on sertifikaat, võib töö tulemuslikkus kohati väheneda, sest on oht tuua sisse ebavajalikku bürokraatiat.

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: ajalugu ja hetkeseis

- Esimene versioon 1987; uuendatud 1994 (u. 20 standardit); uuendatud 2000 ("ISO 9000: 2000 standardid"); vahepeal uued versioonid
- ISO 9000:2015, *Quality management systems. Fundamentals and Vocabulary*
- ISO 9001:2015, *Quality management systems. Requirements*
- ISO 9004:2018, *Quality management — Quality of an organization — Guidance to achieve sustained success*

Lisaks on mitmesuguseid rakendusmaterjale, muuhulgas standard *ISO/IEC/IEEE 90003:2018 | Software engineering — Guidelines for the application of ISO 9001:2015 to computer software* sisaldab suuniseid ISO 9001:2008 kohaldamiseks tarkvara väljatöötamisele, tarnimisele, installeerimisele ja hooldusele.

ISO 90003 sisu ülevaade:

1-3. Ulatus, viited, terminid
4. Organisatsiooni kontekst
5. Juhtkond
6. Planeerimine
7. Tugi
8. Käitus *(Operation)*
9. Soorituse hindamine
10. Parendamine

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.

ISO 9001: 2015 ei kohusta sisse seadma ühesuguse struktuuriga kvaliteedihalduse süsteeme, kavandama kvaliteedihalduse dokumentatsiooni vastavuses selle standardiga või kasutama organisatsioonis selle standardi spetsiifilist terminoloogiat.

Kokkuvõttes, ISO 9001 rakendamisel tuleks lähtuda ärivajadustest, kvaliteedijuhtimise põhimõtetest ja oma valdkonna spetsiifikast, muuhulgas arvestades järgmises jaotises toodud esimesi kvaliteedijuhtimise tegevusi ettevõttes.

6.1.6. Kvaliteedihalduse protsessid - esimesed sammud ettevõttes

Kvaliteedihaldus on enamasti suunatud organisatsiooni laiemate eesmärkide saavutamisele. Kui kvaliteedinõudeid ei ole väga arusaadavalt püstitatud, võib konkreetse töö tegijal olla vähe otsest 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 pole ettevõttes kvaliteediküsimustega tegeldud, siis ei ole otstarbekas alata kogu ettevõtet hõlmava kvaliteedihaldusega. Pigem tasub valida kõige kriitilisem töölõik, kus kvaliteedihaldus annab suurima efekti. Kui valitud meetod õigustab ennast selles lõigus, võib üle minna järgmistele kriitilistele valdkondadele. Vajadusel korrigeeritakse meetodit või kvaliteedipoliitikat.

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. Tutvustage eesmärke ja poliitikaid kõigile töötajatele
4. Valige kvaliteedihalduse meetod
5. Valige ettevõtte kõige kriitilisem töölõik
6. Parandage seda töölõiku vastavalt valitud metoodikale, nii et tulemused oleksid näha
7. Kui põhimõtted ja metoodika on ennast õigustanud, minge tagasi sammule 5
8. Kui metoodika või kvaliteedipoliitika vajavad ümbervaatamist, minge tagasi sammudele 4, 3, 2 või 1

Küsimusi ja materjale

- Kvaliteet, tarkvara, nõuete liigitusi, tarkvara elutsükkel, elutsükli liigid
- Kvaliteedihaldus, -poliitika, -süsteem, -tõendus

- Miks levib ebakvaliteetne tarkvara?
- Kvaliteet ja organisatsiooni juhtimine
- Kvaliteedihalduse ja arendusmeetodite seos
- Kvaliteedihalduse integreerimine tarkvara elutsüklisse
- Kvaliteedihalduse käsitlusi, teese, süsteeme ja auhindu
- ISO 9000 seeria, ISO 90003. Rakendamine ja sertifitseerimine
- Verifitseerimine ja valideerimine, protsessi ja toote kvaliteedihaldus, arendus- ja kvaliteediohje protsessid - võrdlus ja standardid
- Mida peaks süsteemi arendaja (tellija, kasutaja, hooldaja, tarnija, firmajuht) teadma kvaliteedist ja standarditest (vt ka kursuse sihtrühmade ülevaade ning lugemissoovitusi materjali algusest)? Millest alustada tarkvaraprotsessi kvaliteedihalduse kavandamist?

Materjale iseseisvaks tööks (näited)

- Daniel Galin, Software Quality assurance from theory to implementation, Pearson - Addison-Wesley. Chapters 22, 23.2, 25.
- Ian Sommerville. Software Engineering. Tenth Edition. Addison-Wesley, Ch 24.
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE. Chapter 10.
- Stefan Wagner. Software Product Quality Control. Springer, www.it-ebooks.info.
- Certified Tester Foundation Level Syllabus, ISTQB. Chapter 1.1.4.
- http://www.efqm.org/sites/default/files/overview_efqm_2013_v1.1.pdf
- https://www.iso.org/obp/ui/#iso:std:iso:9001:ed-5:v1:en

6.2. Kvaliteedi mõõtmine: näitajad

Tarkvara ja sellega seotud protsesse on vaja mõõta või hinnata mitmetel põhjustel - näiteks selleks, et anda eelhinnang tehtava töö mahule ja maksumusele, et teha korrektset pakkumist, et otsustada toote vastuvõtmise üle, et osata arendajate või hooldajate tööd rahaliselt väärtustada jne.

Selliseks mõõtmiseks saab kasutad tarkvara näitajaid - arvutatavaid või hinnatavaid suurusi, mis iseloomustavad olulisi omadusi ja võivad olla seotud ülesande, projekti, dokumendi või ettevõttega.

Tarkvara näitajate kasutamine on efektiivne vahend arendus- ja muude protsesside haldamiseks, kuid sellega peavad kaasnema teised kvaliteedihalduse tegevused ning võrdlevad hinnangud teiste arendajatega.

6.2.1. Ülevaade tarkvara näitajatest

Tarkvara näitajad hindavad / iseloomustavad programmi (mahtu, keerukust), hindavad kvaliteeti, toetavad arendusprotsessi (prognoosivad töö mahtu, ridade arvu) jne. Võimalikke näitajaid on palju, ülevaate saamiseks võib neid klassifitseerida:

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

Toome selles ja järgmises punktis mõned näited.

Kuna teenustaseme leppeid tuleb jälgida ja täita, on nendes sisalduvad parameetrid loomulikud ekspluatatsiooni protsessi näitajad, näiteks: lubatud summaarse tööseisaku aeg etteantud ajavahemikus, kasutaja pöördumise lahendamise aeg, teenuse seadistamise aeg uuele kasutajale, teenuse taastamise aeg peale suurt tõrget, andmete taastamise aeg jne.

Üks ideelt selgem ja mitmeti kasutatav näitaja on koodi ridade arv, kuid selle näiliselt lihtsa näitajaga pole alati kõik lihtne. Kõigepealt, tarkvara funktsionaalsus ja kvaliteet on olulisemad kui koodi ridade arv. Edasi, uurimused näitavad, et erinevad inimesed võivad sama programmi ridade arvu hinnata kuni suurusjärgu võrra erinevaks. Muuhulgas võib hoolduse puhul negatiivne koodiridade arv päevas olla hea näitaja, kui liiasus väheneb. Seega saab koodi ridade arvu (nagu ka teisi näitajaid) kasutada vaid kindlate reeglite korral, teatud olukordades ning võrdlusandmete olemasolul.

Näiteks pakutakse COCOMO mudeli puhul (Boehm, B. ja teised. Software Cost Estimation with COCOMO II. Prentice-Hall, 2000) järgmised koodi ridade arvu hindamise põhimõtted.

- Arvestatakse vaid lõplikku koodi, mis antakse üle tarkvaratoote osana - vahepealseid (prototüüp)versioone, testimistarkvara jms ei loeta.
- Arvestatakse vaid koodi, mis on loodud arendustiimi poolt - ei loeta koodi, mida toodavad arenduskeskkonnad, programmigeneraatorid vms abivahendid.
- Arvestatakse loogilisi koodiridu (lauseid - määratlus sõltub programmeerimiskeelest; üks loogiline rida võib paikneda mitmel füüsilisel real või vastupidi).
- Deklaratsioonid loetakse ridade arvu sisse, kommentaarid mitte.

Koodi ridade arv võib muuhulgas anda üldise pildi arenduse mahu suurusjärgust, seda on kasutatud ka arendaja tootlikkuse ja vigade tiheduse hindamisel (sobib vaid teatud tingimustel, vt ülal).

Esimesed näitajad ilmusid seitsmekümnendate aastate algul. Tuntumad olid McCabe programmi keerukuse mõõt (1976) ja Halsteadi tarkvara teadus (1977).

Toome allpool veel kvaliteedinäitajate 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 = (Funktsioonide arv tehnilises ülesandes) / (Funktsioonide arv spetsifikatsioonis).
- Keskmine tõrgetevaheline aeg = (Funktsioneerimise kestvus) / (Tõrgete arv)
- Probleemi lahendamise keskmine aeg = (Probleemide lahendamise summaarne aeg) /(Probleemide arv)

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

Allpool on toodud valik näitajate teemalistest korduma kippuvatest küsimustest koos esialgsete vastustega:

- "Mind ei huvita tehnilise ülesande ja spetsifikatsiooni vastavus, mind huvitab, kas see tarkvara teeb, mida vaja!" Lahendus - näitajaid tuleks kasutada seal, kus nende põhjal võetakse vastu otsuseid ja nende osapoolte puhul, kes neist on huvitatud
- "Ütleme kliendile, et tehnilise ülesande ja spetsifikatsiooni vastavus on 1.3333… See arv ei ütle talle mitte midagi." Lahendus - näitajate/projektide võrdlusandmete andmebaasid
- "Suurus, mida saab mõõta, pole huvitav; suurust, mis on huvitav, ei saa mõõta." Lahendus - standardida näitajate väärtustamise metoodikad, kasutada võrdlusandmeid, loobuda üldsõnalistest väidetest
- "Millised probleemid võivad kaasneda näitajate sisseviimisega?" Näitaja peab olema sisukas ja oluline, tema sisseviimise võimalikud kulud ja tulud peavad olema hinnatud

6.2.2. Tarkvara arendusmaksumuse prognoos

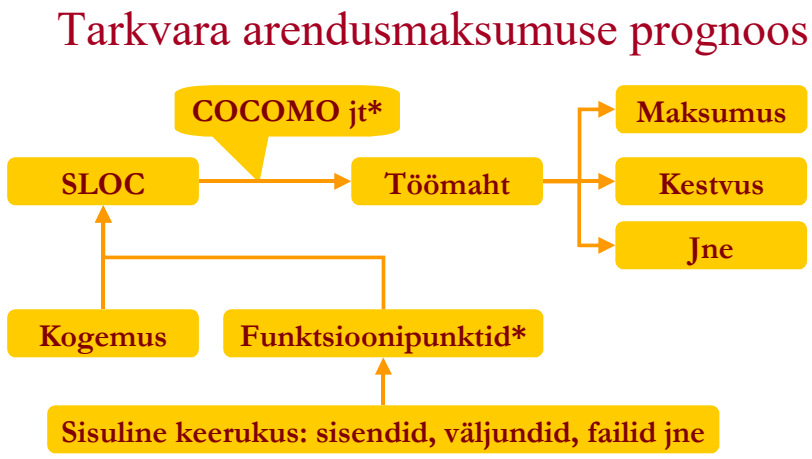
Näitajate rakendamise näitena vaatame tarkvara arendusmaksumuse prognoosi. See on maailmas tunnustatud kui keerukas ülesanne. Ideaalis, selle ülesande sisendiks on tarkvara spetsifikatsioon ning väljundiks - arenduse maksumus või arenduseks vajalike inimkuude (-päevade, -aastate) arv. Kahjuks pole head spetsifikatsiooni alati olemas, ka on lisaks funktsionaalsusele palju muid parameetreid, 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 - eelnev kogemus, mitmesuguste parameetritega projektide tegelikud maksumused vms.

Eelneval kogemusel põhinevad mitmed meetodid arenduse mahu hindamiseks arendustiimi poolt (*Planning poker, Delphi)*. Nende puhul hindavad osalejad arenduse mahtu alguses anonüümselt, et vältida grupi mõju üksikule arendajale. Saadud hinnanguid arutatakse ning püütakse leida konsensuslik mahu hinnang.

On ka loodud meetodeid ja tarkvara, mis leiavad prognoositava maksumuse, tuginedes eelnevale kogemusele. Siin saab kasutada tarkvara mahu hinnangut ja anda 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 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 SystemStar (http://www.softstarsystems.com/) demoversioon.



Tarkvara maksumuse hinnang ilma ettevõtte sisese või välise võrdlusbaasita on vähe realistlik. Hindamiseks võib muuhulgas 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.

Lisaks prognoosile on vaja maksumust ka hallata. Siin võivad kasulikud olla mitmesugused meetodid, mida pakuvad https://en.wikipedia.org/wiki/Earned_value_management, PMBOK ja teised.

Tarkvara arendusmaksumuse prognoosi kokkuvõtteks:

- Eelneval tiimi kogemusel põhinevad mitmed meetodid arenduse mahu hindamiseks arendustiimi poolt, kus mahtu hinnatakse alguses anonüümselt ning saadud hinnangute baasil püütakse leida konsensuslik mahu hinnang.

- Tarkvara maksumuse prognoosi mudelipõhised 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 igal juhul kasulik arvestada lisaks lokaalseid, ettevõtte oma kogemusi ja andmeid

Küsimusi ja materjale

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

Materjale iseseisvaks tööks (näited)

- Daniel Galin, Software Quality assurance from theory to implementation, Pearson - Addison-Wesley. Chapter 21.
- Ian Sommerville. Software Engineering. Tenth Edition. Addison-Wesley, Ch 24.
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE. Chapter 4 Section 4.
- Certified Tester Foundation Level Syllabus, ISTQB. Chapter 5.2.5, 5.3.1.
- Boehm, B. et al. Software Cost Estimation with COCOMO II. Prentice-Hall, 2000
- David Garmus, David Herron. Function Point Analysis: Measurement Practices for Successful Software Projects. Addison-Wesley Information Technology Series, 2000

6.3. 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 flööni vooluvõrku. Standardid püstitavad nõuded suurtele toote või rakenduse klassidele. Anname allpool standardi mõiste ja ülevaate tarkvara puudutavatest standarditest.

6.3.1. Standardi mõiste

Standard on konsensuse 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, ettevõte, 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 aktsepteeriksid, 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 ettevõtteilt kindlalt töötavaid protseduure üle kanda)
- Kvaliteedihaldus
- Standardid võimaldavad reguleerida tootjate omavahelist konkurentsi

Puudused:

- Lisakulu, -aeg, -bürokraatia
- 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 soovitused
- 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 250 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 vähe mõtet tema peale aega kulutada.

6.3.2. Standardimine Eestis

Eestis on üle võetud rida IT standardeid, mis sisaldavad hinnalist rahvusvahelist kogemust. Paljud olulised standardid on tõlgitud. Lisaks juba vaadeldud tarkvara elutsükli protsesside ning tarkvara kvaliteedimudeli standarditele on tõlgitud infoturbe alaseid standardeid, dokumentatsiooni koostamise ja halduse suunised jne. Samuti on koostatud infotehnoloogia reeglid eesti keele ja kultuuri keskkonnas, infotehnoloogia sõnastik ja muid originaalseid materjale.

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

- tarkvaraga seotud tööprotsesside kujundamiseks, juhtimiseks ja korrastamiseks (ISO/IEC 12207)
- kvaliteedihalduseks (ISO 9000 seeria, ISO/IEC 25000 seeria)
- infoturbe haldamiseks (ISO/IEC 27000 seeria)
- teenuste loomiseks ja haldamiseks (ISO/IEC 20000 seeria)
- eestikeelse tarkvara loomisel (EVS 8)
- IT terminoloogia korrastamiseks (ISO/IEC 2382 seeria)
- dokumenteerimiseks (ISO/IEC 26511 jt)

Standardimise korraldus Eestis

- Eesti standardiorganisatsioon (EVS)
- Erialastandardite arendamiseks on sõlmitud koostöölepingud erialaliitude ja asutustega
- Tehnilised komiteed (TK, üle 50), sh EVS/TK 4 Infotehnoloogia
- Üle 25000 Eesti standardi kõigis valdkonades, neist tõlgitud üle 1500, originaalseid alla kolmesaja
- EVS/TK4 - IT standardimise tehniline komitee, asutatud 29.10.97, on liige mitmetes ISO allkomiteedes

6.3.3. Ülevaade tarkvara standarditest

Tarkvara standardid soodustavad kvaliteedi parandamist, tarkvara koostalitlust, efektiivsust ja mõõtmisi. Tarkvara standardite liigitusi:

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

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

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 standardimiskehamid on loonud mitmeid standardeid:

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

Tarkvara elemendid (ANSI/IEEE Std 1028):

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

Standardite kasutamisvõimalusi

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

6.3.4. 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, seepärast erinevad ka protsessid, nende tegevused ja tegevustega kaasnevad tulemid. Ei saa näiteks võrrelda dokumentatsiooni, mida koostatakse väikeettevõtte isikliku klientide andmebaasi programmeerimisel ja (…loodetavasti…) lennuki juhtimissüsteemide 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
- ISO 15489. Dokumendihaldus (standardite seeria)
- ISO 5127. Informatsioon ja dokumentatsioon. Põhialused ja sõnastik

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

Küsimusi ja materjale

- 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
- Dokumenteerimist käsitlevad standardid
- Võrdlus: protsessihalduse raamistikud ja arendusmetoodikad
- ISO/IEC 25000 seeria; EVS-ISO/IEC 12207; CMMI; RUP; XP; ISO/IEC 20000 seeria ja ITIL; ISO/IEC 27000 seeria/ IT Grundschutzhandbuch / ISKE; ISO 9000 seeria ja ISO 90003 – sisu ja omavaheline võrdlus (kui saab võrrelda)

Materjale iseseisvaks tööks (näited)

- ISO rahvusvahelised standardid, http://www.iso.org/
- EVS tööpõhimõtted ja Eesti standardid, https://www.evs.ee/

6.4. Infosüsteemi audit

Nagu iga keerulise asjaga, on ka tarkvaraga (laiemalt, infosüsteemidega) aeg-ajalt probleeme. Üks võimalus probleemide ennetamiseks 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 metoodikast COBIT.

6.4.1. 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 infolekkega. Kas saab selgitada põhjuseid ja hoida ära sellised juhtumid tulevikus? Juhtkonnale tundub, et infotehnoloogia ressursse ei kasutata hästi. Mida teha? Oluline projekt venib. Mida ette võtta?

Näiteid auditi eesmärkide kohta:

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

Auditeeritakse kõiki infosüsteemidega seotud objekte, tegevusi, protsesse ja valdkondi, sealhulgas planeerimist, organisatsiooni, dokumentatsiooni, hanget, projekti, projekti juhtimist, arendust, metoodikaid, kasutamist, hooldust, mõõtmist, aruandlust, jälgimist (sisemist 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. Audiitor võib kuuluda organisatsiooni koosseisu või olla väljaspool seda.

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 metoodikat.

Eetikareeglid ütlevad muuhulgas, et audiitor peab

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

6.4.2. 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 metoodika, 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.

Spetsiifiliste auditite (nt ISKE või Eesti infoturbe standardi audit) korraldus võib olla määratud eraldi juhenditega.

6.4.3. Organisatsioonid ja sertifitseerimine

Maailmas ühendab infosüsteemide audiitoreid eelkõige ISACA (praegu on see akronüüm, kunagi oli kasutusel nimetus Infosüsteemide Auditi ja Kontrolli Assotsiatsioon, *Information Systems Audit and Control Association*). Assotsiatsioonil on üle 165,000 liikme ning 200 haruühingu enam kui 80-s riigis, sealhulgas Eestis (Eesti Infosüsteemide Audiitorite Ühing, EISAÜ). ISACA arendab COBIT raamistikku, 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 töölõike on audiitorite sertifitseerimine. Sertifitseeritud infosüsteemide audiitor (CISA) peab sooritama vastava eksami, tõendama pidevat koolitust, jälgima audiitori eetikanorme ja standardeid, omama töökogemust, maksma aastamaksu.

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 etteantud arvule küsimustele. CISA sertifikaat eeldab lisaks eksami sooritamisele eetikanormide ja standardite tunnustamist, töökogemust, pidevkoolitust ja aastamakset. Maailmas on CISA sertifikaadi saanud üle 140,000 inimese. Lisaks CISAle annab ISACA välja ka muid sertifikaate: Certified Information Security Manager (CISM), Certified in the Governance of Enterprise IT (CGEIT), Certified in Risk and Information Systems Control (CRISC) jm.

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

Eesti infosüsteemide audiitorühingusse kuuluvad alast huvitatud isikud. Ühingu tegevusvaldkonnad on IS auditi alane koostöö ja teavitamine, koolitus, avalikkuse informeerimine, koostöö teiste organisatsioonidega (ISACA), auditi standardite ülevõtmine või koostamine, infosüsteemide audiitorkontrolli eeskirjade koostamine ja arendamine.

6.4.4. COBIT

ISACA arendab ja haldab COBIT raamistikku (COBIT on praegu akronüüm, algne tähendus on muutunud), millest on ilmunud mitu versiooni. Algselt andis COBIT auditi korralduse üldise metoodika. Viimastel aastatel on COBIT arenenud pigem üldise protsessiraamistiku suunas. Viimane versioon, COBIT 2019, on äriraamistik ettevõtte IT valitsemiseks ja juhtimiseks.

COBIT 2019 põhimõtted:

- Huvipoolte vajaduste rahuldamine
- Kõigi ettevõtte funktsioonide ja protsesside katmine
- Ühtse integreeritud raamistiku kasutamine
- Erinevate võimaldajate kirjeldamine

- Valitsemise ja juhtimise eristamine

Küsimusi ja materjale

- IT audit, selle eesmärgid, auditeeritavad objektid, maht, audiitor, riskid
- Planeerimine, korraldus
- Organisatsioonid, sertifitseerimine
- ISACA, COBIT, CISA

Materjale iseseisvaks tööks (näited)

ISACA, http://www.isaca.org

COBIT, http://www.isaca.org/cobit

CISA, http://www.isaca.org/cisa

EISAÜ, http://eisay.ee/

6.5. Millest alustada?

Eelmistes jaotistes on toodud erinevaid lähenemisi (tarkvara) kvaliteedihaldusele: lihtsate asutuse poolt välja töötatud põhimõtete rakendamine, mitteformaalsed kvaliteedihalduse meetodid, kvaliteedi auhinnad, ISO 9000 seeria (sealhulgas tarkvarale orienteeritud ISO 90003). Tarkvara ja sellega seotud protsesside kvaliteedihalduses võivad olla kasulikud agiilsed metoodikad, kvaliteedi näitajad, ISO/IEC 20000 seeria ja/või ITIL, ISO/IEC 27000 seeria ja/või ISKE, EVS-ISO/IEC 12207, COBIT, CMMI, TCO (Total Cost of Ownership). Üldisematest kvaliteedijuhtimise raamistikkudest tulevad veel kõne alla näiteks Six Sigma ja tasakaalus tulemuskaart.

Arvestada tasub kindlasti tarkvarafirmade metoodikaid (enamikul suurematest firmadest on oma arendus- või vähemalt projektijuhtimise metoodika) ning võimalust luua ise oma protsessimudel.

Laias laastus võib kõiki neid raamistikke jagada infotehnoloogiale orienteerituteks ja üldisteks, viimaste hulka kuuluvad eelmainitutest ISO 9000 seeria, Six Sigma, tasakaalus tulemuskaart. Samuti eristub laiapõhjaliste (paljudele elutsükli protsessidele orienteeritud) raamistikkude rühm, sh ISO/IEC 12207, ISO 90003, CMMI.

Mitmed rahvusvahelised standardid (nt EVS-ISO/IEC 12207, EVS-EN ISO 9000, samuti olulisemad ISO/IEC 20000 ning ISO/IEC 27000 seeria standardid) on vastu võetud eesti standarditena, mis hõlbustab nende kasutamist.

Kui võrrelda standardeid ISO/IEC 12207 ja ISO 9000 seeriat, siis esimesel puudub sertifitseerimine, teisel on olemas rahvusvaheline tunnustamise metoodika ja infrastruktuur. Esimene on protsesside juhtimisele orienteeritud, näiteks arendus 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 ISKE puhul - õigusaktide nõuetest; ISO 27000, ISO 9000, CMMI, COBIT rakendamise puhul – klientidelt või partneritelt; ISO 9000, kvaliteediauhinna, Six Sigma puhul – emafirmalt või juhtkonnalt; COBIT puhul – kõigilt eelpoolmainitutelt, 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 võib vaadata kui rahvusvahelise kogemuse süstematiseeritud esitust, enamaltjaolt nad ei vastandu, neid võib kasutada (ja kasutatakse) koos.

Küsimusi ja materjale

- Millest alustada tarkvaraprotsessi kvaliteedihalduse kavandamist?
- Millised on tarkvara elutsükli protsessid, kuidas on kvaliteedihaldus nende protsessidega seotud?
- Võrdlus: protsessihalduse raamistikud ja arendusmetoodikad
- Ülevaade tarkvara elutsükli mudelitest
- Põhimõisted, eesmärgid, ülevaade, millal ja kuidas kasutada: ISO 9000 seeria, ISO 90003, EVS-ISO/IEC 12207, CMMI, ISO/IEC 20000 seeria ja ITIL, ISO/IEC 27000 seeria ja ISKE, TDD, XP, Scrum, RUP.

Materjale iseseisvaks tööks (näited)

- Ian Sommerville. Software Engineering. Tenth Edition. Addison-Wesley, Ch 24, 25
- ISO/IEC 12207:2017, https://www.evs.ee/et/iso-iec-ieee-12207-2017
- CMMI, http://cmmiinstitute.com/ , http://www.cmmifaq.info/
- CMMI-DEV, http://resources.sei.cmu.edu/asset_files/TechnicalReport/2010_005_001_15287.pdf
- Test Maturity Model integration (the TMMi Model), https://www.tmmi.org/
- TPI Next, http://www.tmap.net/tpi-downloads
- Daniel Galin, Software Quality assurance from theory to implementation, Pearson - Addison-Wesley. Chapter 23, 24
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE. Chapter 8, Section 3
- ISKE materjalid https://www.ria.ee/ee/iske.html, https://www.ria.ee/en/iske-en.html
- ISO 27000 standardipere http://www.iso.org/iso/iso27001
- IT-Grundschutz https://www.bsi.bund.de/EN/Topics/ITGrundschutz/itgrundschutz_node.html

7. Ülevaateid standarditest

Järgnevates jaotistes antakse lühiülevaade veel mõnedest standarditest, mis võivad olla kasulikud tarkvara arenduses ning mida saab kasutada ka iseseisvates ja ainetöödes. Standardi kokkuvõtteid ja uute variantide projekte võib leida veebist

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

Standardist ISO/IEC 12207 on olemas mitu järjestikust versiooni. Viimane neist on ISO/IEC/IEEE 12207:2017 "Süsteemi- ja tarkvaratehnika. Tarkvara elutsükli protsessid". Hetkel olemasolev eestikeelne versioon on EVS ISO/IEC 12207:2009 – varasema ingliskeelse versiooni tõlge. Viimane ühendab süsteemi- ja tarkvara elutsükli protsessid ning sisaldab 43 protsessi - oluliselt rohkem kui eelnevad versioonid.

Standard EVS-ISO/IEC 12207 võib olla aluseks ettevõtte tarkvaraprotsesside haldusele. Selle evitamise hõlbustamiseks saab kasutada juhendit *ISO/IEC/IEEE 24748-3 Systems and software engineering — Life cycle management — Part 3: Guidelines for the application of ISO/IEC/IEEE 12207 (software life cycle processes)*.

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.

ISO/IEC/IEEE 12207:2017 sisaldab leppeprotsesse (2 protsessi), organisatsioonilisi protsesse (6), tehnilise halduse protsesse (8), tehnilisi protsesse (14).

ISO /IEC 12207 evitamine sisaldab soovitatavalt järgmisi samme.

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

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

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

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

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

7.2. Küpsus- ja suutvusmudelid

Kuna tarkvaraga seotud võimalikke protsesse ja tegevusi on palju, on loomulik küsida, millisele tasemele ollakse ise jõudnud või kuhu on jõudnud partnerid, nt tarnijad. Tasememudelid võimaldavad hinnata olemasolevat taset ning püsitada sihttaset ning prioriseerida tegevusi, mis on vajalikud sihttasemele jõudmiseks.

Enim kasutatav, ajalooliselt üks esimesi ja veebis kättesaadav on Carnegie Mellon ülikooli Software Engineering Institute poolt välja töötatud *Capability Maturity Model Integration* ning selle mitmed modifikatsioonid. Sisu:

- sisaldab mitut mudelit (hange, arendus, teenused)
- 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 hanke, arenduse, hoolduse ja muude protsesside head tavad
- enesehindamine, võrdlus teistega
- 4 suutvustaset, kasutatakse eelkõige protsesside hindamiseks: Level 0 (incomplete), 1 (performed), 2 (managed), 3 (defined)
- 5 küpsustaset, kasutatakse eelkõige organisatsiooni või protsessigruppide hindamiseks: Level 1 (initial), 2 (managed), 3 (defined), 4 (quantitatively managed) and 5 (optimizing)
- tasemevaade (individuaalsete protsesside hindamiseks) ja etappide vaade (organisatsiooni või protsessigruppide hindamiseks)
- 22 protsessivaldkonda
- protsessivaldkonnad ütlevad, mida tehakse
- suutvuse tasemed ütlevad, kui palju tehakse
- küpsustasemed ütlevad, kui hästi seda tehakse

Tugevused / nõrkused:

- Võib valida sobiva mudeli
- Detailne
- Spetsiaalselt tarkvara arendavatele organisatsioonidele
- Rõhk on pideval arengul, mitte vaid taseme hindamisel ja tunnustamisel
- Võib kasutada nii sertifitseerimise, täiendamise kui ka enesehindamise vahendina
- Veebis tasuta kättesaadav
- Oskamatul rakendamisel võib tekitada asjatut bürokraatiat

Ka mitmed muud raamistikud, nt ISO/IEC 33000 standardite seeria ja testimisprotsesside raamistikud (Test Maturity Model integration ja TPI Next) kasutavad küpsustasemeid.

7.3. ISO/IEC 25000 standardite seeria

ISO/IEC 25010 - tootekvaliteet

Üks levinud tarkvara kvaliteedinäitajate skeem on esitatud rahvusvahelises ISO/IEC 25000 standardiseerias. Selle seeria üks olulisemaid osi on kvaliteedimudelite standard ISO/IEC 25010 *Software engineering: Software product Quality Requirements and Evaluation (SQuaRE) — Quality model,* mille eestikeelne tõlge on standard EVS-ISO/IEC 25010:2011.

Standardi ISO/IEC 25010 tootekvaliteedi mudel on toodud järgnevas tabelis (vt ka ISO terminite andmebaas https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en).

Süsteemi / tarkvaratoote kvaliteet (EVS-ISO/IEC 25010:2011 põhjal)

Funktsionaalne sobivus	Soorituse tõhusus	Töökindlus	Ühilduvus	Kasutatavus	Turvalisus	Hooldatavus	Porditavus
Funktsionaalne täielikkus Funktsionaalne õigsus Funktsionaalne kohasus	Ajaline käitumine Ressursikasutus Suutvus	Küpsus Tõrketaluvus Taastuvus Töökindluse vastavus	Koosolu-võime Koostalitlus-võime	Kohasuse äratuntavus Õpitavus Käsitsetavus Eksituskindlus Kasutajaliidese esteetika Hõlpsus	Konfidentsiaalsus Terviklus Salgamatus Jälitatavus Autentsus	Modulaarsus Taaskasutatavus Analüüsitavus Modifitseeritavus Testitavus	Sobitatavus Installeeritavus Asendatavus

Toodud kvaliteediomaduste (alla joonitud) ja alamkarakteristikute täpsemad määratlused on järgmised:

funktsionaalne sobivus (*functional suitability*) - määr, milleni toode või süsteem pakub funktsioone, mis kasutamisel ettemääratud tingimustes rahuldavad sõnastatud ja eeldatavaid vajadusi.

funktsionaalne täielikkus (*functional completeness*) - määr, milleni funktsioonistik katab kõiki ettemääratud ülesandeid ja kasutaja eesmärke

funktsionaalne õigsus (*functional correctness*) - määr, milleni toode või süsteem annab õigeid ja vajaliku täpsusega tulemusi

funktsionaalne kohasus (*functional appropriateness*) - määr, milleni funktsioonid soodustavad ettemääratud ülesannete täitmist ja eesmärkide saavutamist

soorituse tõhusus (*performance efficiency*) - sooritus ettemääratud tingimustes kasutatava ressursikoguse suhtes

ajaline käitumine (*time behaviour*) - määr, milleni toote või süsteemi reaktsiooni- ja töötlusajad ning läbilaskevõime oma funktsioonide täitmisel nõuetele vastavad

ressursikasutus (*resource utilization*) - määr, milleni ressursside kogused ja liigid nõuetele vastavad, mida toode või süsteem kasutab oma ülesannete täitmisel

suutvus (*capacity*) - määr, milleni toote või süsteemi parameetri ülarajad nõuetele vastavad

ühilduvus (*compatibility*) - määr, milleni toode, süsteem või komponent saab vahetada teavet teiste toodete, süsteemide või komponentidega ja/või täita oma nõutavaid ülesandeid, kasutades nendega ühist riistvara- või tarkvarakeskkonda

koosoluvõime (*co-existence*) - määr, milleni toode võib tõhusalt oma ülesandeid täita, kasutades teiste toodetega ühist keskkonda ja ühiseid ressursse, avaldamata kahjulikku toimet ühelegi teisele tootele

koostalitlusvõime (*interoperability*) - määr, milleni kaks või suurem arv süsteeme, tooteid või komponente saab teavet vahetada ja vahetatud teavet kasutada

kasutatavus (*usability*) - määr, milleni ettemääratud kasutajad saavad toodet või süsteemi ettemääratud kasutuskontekstis toimivalt, tõhusalt ja rahuldusega ettemääratud sihtide saavutamiseks kasutada

kohasuse mõistetavus (*appropriateness recognizability*) - määr, milleni kasutajad suudavad aru saada, kas toode või süsteem on nende vajadusteks kohane

õpitavus (*learnability*) - määr, milleni kindlad kasutajad saavad toodet või süsteemi ettemääratud sihtide saavutamiseks õppida, et kasutada toodet või süsteemi ettemääratud kasutuskeskkonnas toimivalt, tõhusalt, riskitult ja rahuldusega

käsitsetavus (*operability*) - määr, milleni tootel või süsteemil on atribuute, mis hõlbustavad selle käsitsemist ja juhtimist

eksitusekindlus (*user error protection*) - määr, milleni süsteem kaitseb kasutajat vigade tegemise eest

kasutajaliidese esteetika (*user interface aesthetics*) - määr, milleni kasutajaliides võimaldab kasutajale mõnusat ja rahuldavat interaktsiooni

hõlpsus (*accessibility*) - määr, milleni toodet või süsteemi saavad ettemääratud kasutuskontekstis kasutada väga mitmesuguste erijoonte ja võimetega inimesed ettemääratud sihi saavutamiseks

töökindlus (*reliability*) - määr, milleni süsteem, toode või komponent ettemääratud tingimustel ja ettemääratud ajavahemikus täidab kindlaid ülesandeid

küpsus (*maturity*) - määr, milleni süsteem, toode või komponent vastab normaalse töö korral vajadustele töökindluse alal

käideldavus (*availability*) - määr, milleni süsteem, toode või komponent on töövõimeline ja kättesaadav, kui seda on vaja kasutada

tõrketaluvus (*fault tolerance*) - määr, milleni süsteem, toode või komponent töötab kavatsetud viisil, hoolimata riist- või tarkvara tõrgetest

taastuvus (*recoverability*) - määr, milleni toode või süsteem katkestuse või tõrke korral suudab taastada sellest otseselt mõjutatud andmed ning süsteemi soovitud oleku ennistada

turvalisus (*security*) - määr, milleni toode või süsteem kaitseb teavet ja andmeid nii, et inimestel või teistel toodetel ja süsteemidel on nende tüüpidele ja volitustasemetele vastav andmetele juurdepääsu määr

konfidentsiaalsus (*confidentiality*) - määr, milleni toode või süsteem tagab, et andmed on kättesaadavad ainult selleks volitatuile

terviklus (*integrity*) - määr, milleni süsteem, toode või komponent väldib volitamatut juurdepääsu programmidele või andmetele või nende muutmisele

salgamatus (*non-repudiation*) - määr, milleni saab tõendada toimingute või sündmuste toimumist, nii et hiljem ei saa neid toiminguid või sündmusi eitada

jälitatavus (*accountability*) - määr, milleni saab mingi olemi toiminguid üheselt selle olemini jälitada

autentsus (*authenticity*) - määr, milleni saab tõendada, et mingi subjekti või ressursi identiteet ühtib väidetava identiteediga

hooldatavus (*maintainability*) - toimivuse ja tõhususe määr, millega ettemääratud hooldajad saavad toodet või süsteemi muuta

modulaarsus (*modularity*) - määr, milleni süsteem või arvutiprogramm on koostatud diskreetsetest komponentidest nii, et ühe komponendi muudatusel on minimaalne mõju teistele komponentidele

taaskasutatavus (*reusability*) - määr, milleni mingit vara saab kasutada mitmes süsteemis või muude varade loomiseks

analüüsitavus (*analysability*) - toimivuse ja tõhususe määr, millega on võimalik hinnata toimet, mida tootele või süsteemile kavatsetav ühe või mitme osa muudatus avaldab, või diagnoosida toote defekte või tõrgete põhjuseid või selgitada välja, milliseid osi tuleb muuta

modifitseeritavus (*modifiability*) - määr, milleni saab toodet või süsteemi toimivalt ja tõhusalt modifitseerida, tekitamata sellega defekte ja madaldamata senist tootekvaliteeti

testitavus (*testability*) - toimivuse ja tõhususe määr, millega saab süsteemile, tootele või komponendile määrata testimiskriteeriume ning sooritada teste nende kriteeriumite rahuldatuse otsustamiseks

porditavus (*portability*) - toimivuse ja tõhususe määr, millega saab süsteemi, toote või komponendi ühest riistvara-, tarkvara- või muust käitus- või kasutuskeskkonnast teise üle viia

sobitatavus (*adaptability*) - määr, milleni saab toodet või süsteemi toimivalt ja tõhusalt teistsuguste või tulevaste riistvara-, tarkvara- või muude käitus- või kasutuskeskkondadega sobitada

installeeritavus (*installability*) - toimivuse ja tõhususe määr, millega saab toodet või süsteemi edukalt installeerida ja/või desinstalleerida mingis kindlas keskkonnas

asendatavus (*replaceability*) - määr, milleni toode saab asendada teist ettemääratud samaotstarbelist tarkvaratoodet samas keskkonnas

ISO/IEC 25010 - kasutuskvaliteet

Standardi ISO/IEC 25010 kasutuskvaliteedi karakteristikud (alla joonitud) ja nende alamkarakteristikud on järgmised:

toimivus (effectiveness) - täpsus ja täielikkus, millega kasutajad saavutavad ettemääratud sihte

tõhusus (efficiency) - kulutatud ressursid suhtes täpsuse ja täielikkusega, millega kasutajad saavutavad ettemääratud sihte

rahuldus (satisfaction) - määr, milleni rahuldatakse kasutaja vajadused toote või süsteemi kasutamisel mingis ettemääratud kasutuskontekstis

kasulikkus (usefulness) - määr, milleni kasutaja on rahul enda tajutava pragmaatiliste sihtide saavutamisega, sealhulgas kasutamise

usaldus (trust) - määr, milleni kasutaja või muu riskiosaline usub, et toode või süsteem hakkab käituma nii nagu on kavatsetud

mõnu (pleasure) - määr, milleni kasutaja naudib oma isiklike vajaduste rahuldamist

mugavus (comfort) - määr, milleni kasutaja on rahul füüsilise mugavusega

riskitus (freedom from risk) - määr, milleni toode või süsteem vähendab potentsiaalset riski majanduslikule seisundile, inimelule, tervisele või keskkonnale

majandusliku riski vähenemine (economic risk mitigation) - määr, milleni toode või süsteem vähendab kavatsetud kasutuskontekstides potentsiaalset riski rahalisele seisundile, tõhusale tööle, ärilisele omandile, mainele või muudele ressurssidele

tervise- ja ohutusriski vähenemine (health and safety risk mitigation) - määr, milleni toode või süsteem vähendab kavandatud kasutuskontekstides potentsiaalset riski inimestele

keskkonnariski vähenemine (environmental risk mitigation) - määr, milleni toode või süsteem vähendab kavatsetud kasutuskontekstides potentsiaalset riski omandile või keskkonnale

kontekstikate (context coverage) - määr, milleni toodet või süsteemi saab toimivalt, tõhusalt, riskitult ja rahuldusega kasutada nii ettemääratud kasutuskontekstides kui ka muudes kontekstides väljaspool neid, mis on algselt otseselt piiritletud

konteksti täielikkus (context completeness) - määr, milleni toodet või süsteemi saab toimivalt, tõhusalt, riskitult ja rahuldusega kasutada kõigis ettemääratud kasutuskontekstides

paindlikkus (flexibility) - määr, milleni toodet või süsteemi saab toimivalt, tõhusalt, riskitult ja rahuldusega kasutada kontekstides, mida pole nõuetes algselt kindlaks määratud

7.4. ISO/IEC 27000 standardipere, ISKE ja IT etalonturbe käsiraamat

Infoturve ei ole kaugeltki vaid IT teema, vaid hõlmab inimesi, organisatsiooni, hooneid, tööprotsesse, IT vahendeid jne. Infoturvet on soovitatav hallata, luues infoturbe halduse süsteemi (ISMS, *Information Security Management System*) - poliitikad, protseduurid, juhised ning nendega seotud ressursid ja tegevused, mille abil organisatsion kaitseb oma infovarasid.

ISO/IEC 27000 standardipere aitab organisatsioone ISMSi teostamisel ja käigushoiul ning sisaldab üle 15 olemasoleva või kavandatava standardi. Siia kuuluvad muuhulgas järgmised kasulikud standardid, mille kohta lähemat infot saab näiteks aadressilt http://www.iso.org/iso/iso27001.

- ISO/IEC 27000:2018. Infoturbe halduse süsteemid. Ülevaade ja sõnavara
- ISO/IEC 27001:2013. Infoturbe halduse süsteemid. Nõuded
- ISO/IEC 27002:2013. Infoturbe halduse tegevusjuhis

- ISO/IEC 27005:2018. Infoturvariski haldus

Riigi ja kohaliku omavalitsuse andmekogude kaitseks tuleb rakendada infosüsteemide kolmeastmelise etalonturbe süsteemi (ISKE). ISKE põhineb infovarade kaardistamisel, nende turbeastmete määramisel, tüüpmoodulite kasutamisel, turvameetmete valikul vastavalt tüüpmoodulitele ning turbeastmele ning turvameetmete rakendamisel.

ISKE rakendamine sisaldab järgmisi tegevusi:

1. Infovarade inventuur
2. Andmekogude kaardistamine ja turvaklasside määramine
3. Muude infovarade turvaklasside määramine
4. Turvaklassiga infovarade turbeastme määramine
5. Tsoonide vajaduse analüüs, asutuse tsoneerimine vajadusel
6. Tüüpmoodulite spetsifitseerimine
7. Turvameetmete loetelu koostamine
8. Turvameetmete rakendamise plaani koostamine
9. Turvameetmete rakendamine
10. Tegeliku turvaolukorra kontroll, vajadusel täiendavate meetmete rakendamine
11. Konfiguratsiooni- ja muudatustehalduse sisseviimine

ISKE põhineb Saksamaa Infoturbeameti (Bundesamt für Sicherheit in der Informationstechnik, BSI) poolt publitseeritaval IT etalonturbe 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 etalonturbe käsiraamatule on kättesaadavad aadressil https://www.ria.ee/ee/iske.html.

ISKE materjale enam ei täiendata, selle asendab lähiaastatel uus Eesti infoturbe standard.

7.5. Teenusehaldus: ISO/IEC 20000 standardite seeria ja ITIL

ISO/IEC 20000 on teenusehaldust käsitlevate standardite pere. Selle esimeses osas (ISO/IEC 20000-1) püstitatakse teenusehalduse süsteemi nõuded. Need sisaldavad teenuse nõuetele vastavate ning nii kliendile kui ka teenuseosutajale väärtust pakkuvate teenuste projekteerimist, üleminekut, tarnimist ja täiustamist. Standardi viimane versioon on ISO/IEC 20000-1:2018.

Teises osas (ISO/IEC 20000-2) antakse juhiseid teenusehalduse süsteemide (*Service Management System,* SMS) rakendamiseks standardi ISO/IEC 20000-1 nõuete põhjal. Teine osa ei lisa uusi nõudeid standardis ISO/IEC 20000-1 sätestatud nõuetele. Mõlemad osad on sõltumatud konkreetsetest parima praktika raamistikest ja teenuseosutaja võib rakendada üldiselt aktsepteeritud juhiste ning oma meetodite kombinatsiooni.

Seeria järgmised osad annavad juhiseid SMSi käsitlusala määratlemise ja ISO/IEC 20000-1 kohaldatavuse kohta, esitavad protsesside etalonmudeli ning pakuvad standardi ISO/IEC 20000-1 näitliku evitamisplaani.

ISO/IEC 20000 kohta saab teavet näiteks aadressilt http://www.iso.org.

ITIL (IT Infrastructure Library) on IT teenuste haldamise parima praktika kogum, mida saab kasutada ISO/IEC 20000-1 nõuete rakendamiseks.

ITILi, sealhulgas selle protsesside kohta saab teavet näiteks aadressidelt https://www.axelos.com/best-practice-solutions/itil/what-is-itil, http://www.itsmf.ee/.

Üldine vaade teenustele on pakutud avalike teenuste korraldamise rohelises raamatus, https://www.mkm.ee/sites/default/files/avalike_teenuste_korraldamise_roheline_raamat.pdf.

7.6. Standarditest IEEE 829-2008 ja ISO/IEC/IEEE 29119-3

Standard IEEE 829-2008 annab struktuuri testimise dokumenteerimiseks ning kogemuse tööks rahvusvahelise standardiga, mis võib edasises kasuks tulla. See standard on asendatud standardiga ISO/IEC/IEEE 29119-3:2013, kuid võib sellegipoolest olla kasulik.

Standardi IEEE 829-2008 erinevaid komponente võib valikuliselt kasutada ka iseseisvates töödes. Eelkõige on seda standardit soovitav 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. Ingliskeelsed 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 lühidalt kajastama olukorda. 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 reaalset 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 25010 kvaliteedinäitajate 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 soovitav konsulteerida õppejõuga):

- Programmipõhine testimine: lause-, haru-, elementaartingimuste-, teeadekvaatsuse 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*,
- Näitajate 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, nt ettevalmistused, kooskõlastused, koolitus...

11. Nõuded testimise ümbrusele (Enviromental 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. Beeta-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 iseseletgitav), 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 detailsed kirjeldused on eraldi dokumendis. Iseseisvas töös tavaliselt sellist eraldi dokumenti pole, seepärast kirjeldatakse siin ülevaatlikult 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 vajadusel 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 (Enviromental 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 Transmittial 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)
Enviroment - 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)

8. Lühendid, sõnastik ja lisamaterjalid

8.1. 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

PDCA - Plan-Do-Check-Act, plaanimine-teostus-kontroll-tegutsemine metoodika

RUP - Rational Unified Process

TDD - Test Driven Development

TK - tehniline komitee

TTL - teenustaseme lepe

XP - Extreme Programming

8.2. Sõnastik

Toome olulisemate kasutatud terminite jaoks ingliskeelsed vasted ja sisuseletused, põhinedes rahvusvahelistele ja Eesti terminoloogiastandarditele. Erinevates standardites võivad ingliskeelsed mõisted olla 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).

Agiilne tarkvaraarendus (välearendus) - agile software development. Kompleks põhimõtteid, metoodikaid, tehnikaid ja raamistikke, mis seavad fookusse inimeste vahelise koostöö, töötava tarkvara, nõuete muudetavuse vastavalt kliendi vajadustele ja paindliku olukordadega kohanemise.

Märkus: eestikeelne soovitatav termin on "välearendus" (https://akit.cyber.ee/term/2148-agile-development, http://www.eki.ee/dict/ametnik/index.cgi?F=M&Q=agiilne), kuid selles konspektis kasutame ka terminit "agiilne [tarkvara]arendus" ning seda järgmistel põhjustel: (1) tõlkevastena ei anna "väle" edasi "agile" sisu, seega tuleks seda võtta pigem terminina (ÕS: väle <2: -da> kiire, kärme, nobe, vilgas); samas tekib kohe uusi termineid, kus see lahendus hästi ei toimi, nt "agile methods", "agile development techniques" jne; (2) termin "agiilne" sobib eesti keelde, vrd nt "labiilne" - vastandina sellistele moodustistele nagu "kredentsiaal" vms; (3) eriala spetsialistide praktika ja arvukad juba olemas olevad tõlked (nt https://agilemanifesto.org/iso/et/manifesto.html).

Arhitektuur - architecture. Standardi ISO/IEC/IEEE 42010:2011 järgi on arhitektuur süsteemi aluskorraldus, mida kehastavad süsteemi komponendid, komponentide seosed üksteisega ja keskkonnaga ning süsteemi kavandamise ja arenduse põhimõtted (https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:42010:ed-1:v1:en).

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

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

Elutsükkel - life cycle. Süsteemi, toote, teenuse, projekti või muu tehisolemi areng ta algatamisest mahavõtuni (EVS-ISO/IEC 12207)

Elutsükli mudel - life cycle model. Elutsüklit puudutavate protsesside ja tegevuste raamstruktuur, mis võib olla korraldatud järkudeks ning mis ühtlasi toimib suhtluse ja mõistmise üldise etalonina (EVS-ISO/IEC 12207)

Funktsionaalne testimine - functional testing. Testimine, mis põhineb süsteemi või komponendi funktsionaalsuse analüüsil.

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. Testimine, verifitseerimine, valideerimine (kursuses kasutatav tähendus)

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 Softvare Reviews and Audits* mõisteid järgmiselt: juhtkonnapoolne hindamine (*management review*), tehniline hindamine (*technical review*), tarkvara inspektsioon (*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)

Protsess - process. Sisendolemeid väljundolemiteks muundav omavahel seotud või interakteerivate tegevuste kogum (ISO 9000:2005)

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)

Risk - risk. Määramatuse toime eesmärkidele (ISO/IEC 27005:2014)

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 elutsükkel - 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 projekteerimine - system design. Süsteemi riistvara ja tarkvara arhitektuuri, komponentide, moodulite, liideste ja andmete määratlemine, nii et süsteem vastaks spetsifitseeritud nõuetele (20.03.02)

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

Teenus - service. Vahend väärtuse pakkumiseks kliendile, hõlbustades kliendi soovitud tulemuste saavutamist. MÄRKUS 1 Teenus on üldiselt immateriaalne. MÄRKUS 2 Teenuse võib teenuseosutajale tarnida ka tarnija, sisemine grupp või tarnijana tegutsev klient (EVS-ISO/IEC 20000-1:2013).

Teenustaseme lepe (TTL) - service level agreement (SLA). Teenuseosutaja ja kliendi vaheline kirjalik kokkulepe, mis määratleb teenused ja teenuste sihttasemed. MÄRKUS 1 Teenustaseme lepe võib olla sõlmitud ka teenuseosutaja ja tarnija, sisemise grupi, või tarnijana tegutseva kliendi vahel. MÄRKUS 2 Teenustaseme lepe võib kuuluda lepingu või muud tüüpi dokumenteeritud kokkuleppe koosseisu (EVS-ISO/IEC 20000-1).

Tehis - artifact. Kunstlikult valmistatud asi, toode vms

Test, testjuht, testimine – test, test case, testing. Vt jaotis 5.1.1.

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)

Välearendus. Vt agiilne tarkvaraarendus

8.3. Lisamaterjalid

Põhilised lisamaterjalid, sh standardid, on viidatud tekstis vastavate teemade juures. Allpool on üldisemaid viiteid.

- Ian Sommerville. Software Engineering. Tenth Edition. Addison-Wesley
- Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE, http://www.computer.org/portal/web/swebok
- Certified Tester Foundation Level Syllabus. International Software Testing Qualifications Board, http://www.istqb.org/download.htm + veel materjale sellelt veebiaadressilt
- Daniel Galin. Software Quality assurance from theory to implementation. Pearson - Addison-Wesley
- ISO/IEC 25010 Software engineering: Software product Quality Requirements and Evaluation (SQuaRE) — Quality model
- ACM / IEEE kursuste soovitused, http://www.acm.org/education/curricula-recommendations
- Rahvusvahelised standardid, https://www.iso.org/home.html, https://www.evs.ee/et/
- COBIT. Governance, Control and Audit for Information and Related Technology. Information Systems Audit and Control Foundation, Rolling Meadows, USA, https://www.isaca.org
- Eesti infosüsteemide audiitorühing. Tallinn, https://www.eisay.ee/
- ISKE, https://www.ria.ee/et/kuberturvalisus/infosusteemide-turvameetmete-susteem-iske.html

- Eesti kvaliteediauhind - www.eaq.ee
- Andmekaitse ja infoturbe leksikon, https://akit.cyber.ee/
- EVS-ISO/IEC 2382 Infotehnoloogia. Sõnastik
- EVS 8. Infotehnoloogia reeglid eesti keele ja kultuuri keskkonnas
- Robert C. Martin. Clean Code. Prentice-Hall
- Perry W.. Effective Methods of Software Testing. John Wiley & Sons, Incorporated
- Glenford J. Myers. The Art of Software Testing. John Wiley & Sons
- Kirjandust ja viiteid võib otsida WWW-st märksõnade nagu *Testing* ja *Quality* alt, samuti vaadata erialaseid uudistegruppe
- Raamatud, 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