TARTU ÜLIKOOL

MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Tarkvaratehnika eriala

Kaisa Paavo

Läbipaistvus testimise protsessis ja testimise lõpetamisel

Magistritöö (30 EAP)

Juhendajad: Helle Hein
Artur Assor

Autor: …… "……" mai 2012

Juhendaja: …… "……" mai 2012

Juhendaja: …… "……" mai 2012

Lubada kaitsmisele

Professor: …… "……" mai 2012

TARTU 2012

SISUKORD

1. Sissejuhatus

Tarkvaraarendus on keeruline protsess ja sellega on seotud palju riske. Riskid tarkvara arenduse protsessis on otseselt mõjutatud testimise protsessi efektiivsusest. Kõike ei ole kunagi võimalik testida ning kõiki võimalikke vigu üles leida on praktiliselt võimatu. Ühel hetkel tuleb vastu võtta otsus, millal süsteem on piisavalt töökindel, et testimine lõpetada. Kuna testimise protsessist on tihti keeruline head ülevaadet saada, siis testimise lõpetamisega ei kaasne kunagi täielikku kindlust. Testimise protsessi juhtimiseks ja jälgimiseks ei ole kirjeldatud üldiseid kindlaid reegleid, suurel määral sõltub see projekti iseloomust ning on testijate endi otsustada. Seetõttu ei ole suurt kindlust, et valitud meetmed või tehtavad tegevused on kõige efektiivsemad.

Käesoleva töö eesmärk on uurida, kuidas võimalikult adekvaatselt saada ülevaadet testimise protsessist ja mille abil otsustada, et rakendus on toodangukõlbulik ning testimise võib lõpetada. Kuna töö autor on ka ise ühes projektis testija, siis on eesmärgiks uurida erinevaid tarkvara projekte, et saada ülevaade, milliseid testimise protsessi jälgimise meetodeid enim kasutatakse ja mille alusel testimise lõpetamise otsus tehakse. Eesmärk on teha kindlaks, kas rakendatakse mingeid kindlaid testimise protsessi jälgimise mustreid, mis tagavad protsessist hea ülevaate ning mille abil saab lihtsamini testimise lõpetamise üle otsustada.

Töö esimene pool on teoreetiline, mis põhineb nii töö autori enda kogemustel, mida ta rakendab süsteeme testides kui ka erinevatest allikatest saadud teadmistel. Töö teoreetiline pool koosneb kolmest suuremast alamteemast: testimise protsess ja selle läbipaistvus, testimise lõpetamine ning testimise käigus tekkinud testimise dokumentatsioon. Esimene peatükk annab ülevaate tegevustest, mille abil testimise protsessi jälgida, et saavutada selle läbipaistvus ja ülevaatlikkus. Töö teine peatükk annab ülevaate testimise lõpetamisega seotud tegevustest ning vahenditest, mille abil oleks kergem testimise lõpetamise otsus vastu võtta ning ka meetoditest, mille abil saab testimise lõpetamist ehk toote valmisolekut ennustada. Kolmas peatükk kirjeldab missugust testimise dokumentatsiooni testimise protsessi jälgimisel ning testimise lõpetamisel koostatakse.

Töö teises osas viidi läbi kvalitatiivne uuring erinevate projektide testijate seas. Uurimuse eesmärgiks on võrrelda erinevates projektides rakendatavaid testimise protsessi jälgimise meetodeid ning uurida, mille alusel ning abil otsustatakse projektis testimine lõpetada. Töö käigus otsitakse ka hinnangut testimise protsessi läbipaistvuse mõjust arendatava tarkvara kvaliteedile.

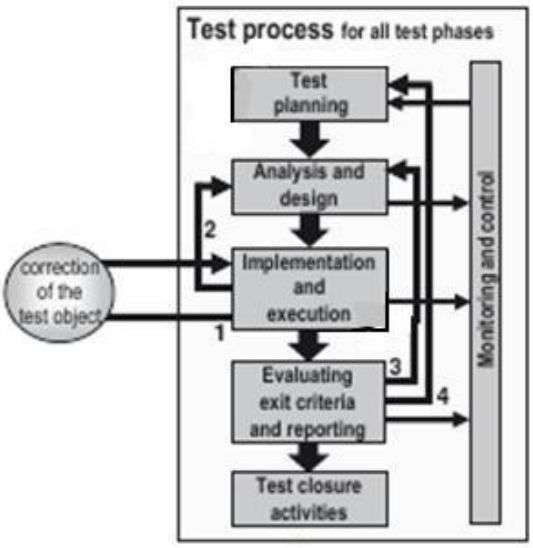
Tähtsamad mõisted, mida käesolevas töös kasutatakse:

- Testimise protsess - protsess, mis koosneb nii staatilistest kui dünaamilistest testimise tegevustest. Testimise protsess sisaldab endas testimise planeerimist, testimise teostamist, testimise hindamist, testimise lõpetamist ning testimise jälgimist.
- Testimise protsessi läbipaistvus –omadus, mis muudab testimise protsessi ülevaatlikuks ning lihtsasti jälgitavaks ja kontrollitavaks.
- Testimise lõpetamine – tegevus, mille käigus otsustatakse testimise lõpetamise ning toote toodangkõlbulikkuse üle.
- Testplaan – dokument, millega planeeritakse testimise protsessi.
- Testjuhtum (*test case*) – tegevuse kirjeldus, kus kirjeldatakse, kuidas test läbi viiakse ning selle alusel kontrollitakse, kas rakendus toimib ootuspäraselt.
- Viga - põhjustab ebakorrektset või ootamatut tulemust, erinedes spetsifikatsioonis kirjeldatust.
- Tööülesanne - vea, arenduse, täienduse või hoolduse ülesanne tööülesande haldustarkvaras.
- Spetsifikatsioon – nõuete dokument, milles on määratletud kirjeldus süsteemi väljatöötamiseks ja vastuvõtuks.
- Tööülesande ja konfiguratsioonihaldustarkvara – programm, tarkvarasüsteemi koodi, versioonide, dokumentatsiooni, vigade ning arendusülesannete halduseks.
- Iteratsioon - lühike etapp projektis, mille käigus luuakse uus funktsionaalsus või täiendatakse olemasolevat. Iteratsiooni lõpus valmib täisfunktsonaalne, töötav ja tarnimiseks valmis tulem.
- Verifitseerimine – protsess, mille käigus kontrollitakse, kas süsteem vastab spetsifikatsioonile.
- Järk (*build*) - projekti lähtekoodi järk, mida lõppkasutajal on võimalik käivitada.
- Artefakt – tarkvaraarenduse käigus loodav ja kasutatav tulem.

2. Tarkvara testimise protsess ja selle läbipaistvus

Tarkvara testimine on iga tegevus, mille eesmärgiks on hinnata programmi või süsteemi omadusi ja määrata kindlaks, et see vastab nõutud tulemustele ning ühtlasi avastada vigu. Tarkvara testimine on tarkvara käivitamine kontrollitud tingimustes, et verifitseerida, kas see töötab nii nagu oodatud, ning valideerida, et määratletud nõuded on need, mida kasutajad tegelikult tahtsid [1].

Standardne testimise protsess koosneb testimise planeerimisest, analüüsist ja disainist, testimise käivitamisest, testimise lõpetamise hindamisest ning testimise lõpetamise tegevustest (Joonis 1). Kuid näiteks *exploratory* testimise puhul nimetatud faasid ei toimu sellises kindlas järjekorras, vaid kordamööda. Enamikus tänapäeva tarkvaraarenduse projektides toimub testimine kogu arendusprotsessi jooksul ning testimise tegevusi ja protseduure rakendatakse korduvalt. Kõiki testimistegevusi tarkvaraprojektis nimetatakse üldiselt testimise protsessiks. Selleks, et testimisest oleks pidevalt hea ülevaade, peab testimisprotsessi koguaeg jälgima ning vajadusel seda kontrollima ja suunama.



Joonis 1. Testimise protsess kõikide testimisfaaside jaoks [2].

Testimise protsessi läbipaistvuse all on antud töös mõeldud pidevat ja ülevaatlikku ülevaadet testimise protsessist. Testimise protsessi läbipaistvus on oluline selle jaoks, et igal arendusprotsessi hetkel oleks olemas teadmine, mis seisus projekt on, kui palju on veel teha, mis on testitud, palju on veel testida, kui palju kriitilisi vigu on üleval, kui kaugel on testimise lõpetamine ja millal on võimalik tarkvara kliendile üle anda. Testimise protsessi läbipaistvust on mahukate arendusprojektide puhul sageli raske saavutada. Kui programmeerijate töö tulemust on üsna lihtne koodi näol mõõta, siis testijate töö puhul see nii lihtne ei ole.

Testimise protsessi juhtimiseks ja jälgimiseks ei ole kirjeldatud ühiseid kindlaid reegleid, suurel määral sõltub see projekti iseloomust - meetodid ja tegevused, mis võivad sobida ühele projektile, ei pruugi sobida teisele. Testimise korraldamine ja projekti seisu jälgimine on testimismeeskonna enda määrata ning otsustada. Järgnevates peatükkides kirjeldatakse testimise protsessi läbipaistvuse tagamiseks võimalikke tegevusi projektis.

2.1 Jooksva iteratsiooni arenduse verifitseerimine

Testimise protsessist paremaks arusaamiseks on oluline tunda ka arendusprotsessi spetsiifikat ja protsessi. Antud töö autori projektis ning üldse kaasaegsetes tarkvaraarendamis- ja testimisprotsessides kasutatakse järjest enam agiilset ja iteratiivset arendusprotsessi [3]. Agiilne arendusprotsess on lähenemine, kus süsteemi arendatakse evolutsiooniliselt lühikeste iteratsioonidega, kus olulisel kohal on kliendiga suhtlus, töötav rakendus ning muutustele vastuvõtlikkus [4]. Iteratiivse mudeli kohaselt koosneb kogu protsess mitmest järjestikusest tsüklist ehk iteratsioonist, mis kõik sisaldavad analüüsi, disaini, programmeerimist ja testimist [5].

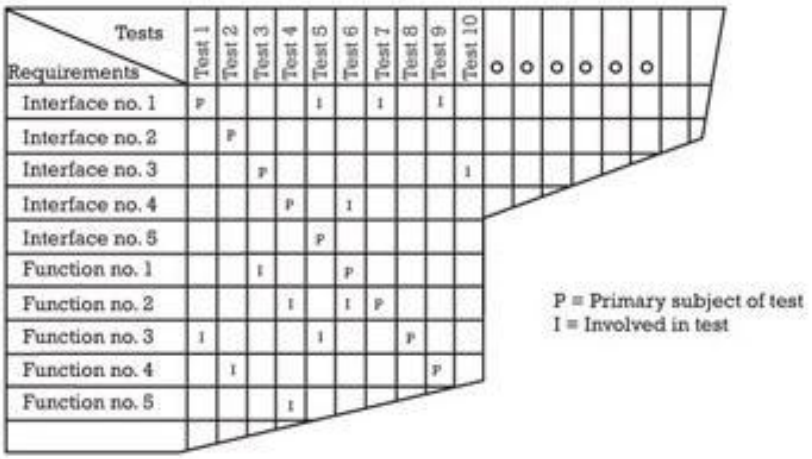
Verifitseerimine on protsess, mis hindab, kas tulemid rahuldavad esialgselt püstitatud tingimusi. Verifitseerimine näitab, kas tarkvaraarenduse protsessi vastava etapi tulemus vastab määratlusele. Antud juhul, kas jooksvas iteratsioonis tehtud arendused vastavad spetsifikatsioonile [6].

Enamikus tänapäeva projektides toimub tarkvara testimine arendusega paralleelselt, jooksvalt, mitte ainult arenduse lõppedes ning projekti lõpufaasis. Kui mingi osa funktsionaalsusest saab arendatud, siis alustatakse selle testimist, samal ajal kui arenduses on juba uus antud iteratsiooni planeeritud funktsionaalsus. Peamine töövahend, mida jooksva arenduse verifitseerimisel või testimisel ja tööülesannete haldamisel üldisemalt kasutatakse, on tööülesannete ja konfiguratsioonihaldustarkvara. Samuti on võimalik luua, kasutada erinevaid dokumente, tabeleid, mis annavad jooksva arenduse verifitseerimisest parema ülevaate.

2.1.1 Testide jälgitavuse maatriks

Korrektne testimine põhineb tarkvarale esitatud nõuetel. Üks viis, kuidas muuta nõuded mõõdetavaks ja testitavaks, on planeerida iga nõude jaoks test. Nõuete jälgitavuse maatriks jälgib nõudeid läbi disaini protsessi kuni arenduseni, mis neid rakendab. Nõuete jälgitavuse maatriksit kasutatakse, et valmistada ette testimise jälgitavuse maatriks *(test traceability matrice).*

Testimismeeskond kasutab jälgitavuse maatrikseid eelkõige sellepärast, et veenduda, et kõik nõuded oleks testidega kaetud. Testide jälgitavuse maatriks aitab näidata arendatud tarkvara vastavust nõuetele ehk verifitseerimist. See on tabel (Joonis 2), kus igal real on nõue ning veerus on test. Hästi kavandatud jälgitavuse maatriks võimaldab edaspidi ja tagurpidi jälgimist: näiteks saab jälgida, missuguse testi/testidega on antud nõue kaetud ning milliseid nõudeid antud test kontrollib. Igat nõuet jälgitakse vähemalt ühe testiga, et näidata selle korrektset rakendamist, kuigi mõne nõude kohta võib vaja minna ka mitut testi. Testi all võib mõelda nii testjuhtumit kui ka testimise või arenduse tööülesannet veahaldustarkvaras. Testimise jälgitavuse maatriksi abil on kiiresti näha, kui osad nõuded on testitud koondatult või kui mõnda nõuet pole veel üldse testitud. Samuti on jooniselt näha, et teised testid, millesse vastavad nõuded on kaasatud, on eraldi tähistatud. Sellisel viisil on võimalik näha seoseid erinevate nõuete vahel [7].



Joonis 2. Testimise jälgitavuse maatriks, kus P on peamine test, millega antud nõuet testitakse ning I on testid, kus vastav nõue on kaasatud [8].

Kuna tarkvara vananeb ning nõudeid muudetakse, siis antud maatriks annab vihjeid ootamatutele ja ebasoovitud tulemustele, kui nõudeid on muudetud või kustutatud. Ilma jälgitavuse maatriksita ei ole võimalik aru saada, milliseid teste või testilugusid peaks ümber tegema ning ei teki ka ülevaadet, kas kõik uued nõuded on testimisega kaetud. Antud maatriksit ei pea koostama terve rakenduse kohta. Piisab sellest, kui kaardistada maatriksil näiteks antud iteratsioonis loodud uued komponendid või funktsionaalsus. Antud meetodit sobib kasutada nii jooksva arenduse verifitseerimiseks (näitamaks, kas arendus vastab nõuetele) kui ka testimise progressist hea ülevaate saamiseks. Viimasest on täpsemalt kirjas töö järgmistes alapunktides.

2.1.2 Jälgitavuse maatriks tööülesannete haldustarkvaras

Jooksvast arendusest ja selle rakendamisest saab hea ülevaate ka näiteks tööülesannete ja konfiguratsioonihaldustarkvara abil. Antud töö autori projektis kasutatakse töövahendit nimega *Changelogic.* Töövahendisse on võimalik sisestada nii projekti nõudeid, tööülesandeid kui ka testjuhtumeid; on võimalik teha muudatusi, kus vastavad tööülesanded realiseeritakse ning on võimalik vaadata versioone, mis muudatustest tekivad. Programmis on olemas vaade (Joonis 3), kus on näha nõuete seosed spetsifikatsioonidega, tööülesannetega, muudatustega, testjuhtumitega ja versioonidega.

ID	Kood	Spetsifikatsioon	Tööülesanded	Muudatused	Testjuhtumid	Sooritused	Väljalasked
	RD1.14						OK
25	RD1.14.2	04. Ravi\Tehnilised Ülesanded\Tehniline ülesanne - RD1.14.2 TISS-lehe täitmine	5991	1487, 1534	11	OK (0) Neg (1) Kõik (1)	3.3.2.34 - toodang
26	RD1.14.3	04. Ravi\Tehnilised Ülesanded\Tehniline ülesanne - RD1.14.3 TISS-lehe vaatamine	5992	1415	12	OK (1) Neg (0) Kõik (1)	3.3.1.28 - toodang
27	RD1.14.1	Tehniline ülesanne - RD1.14.1 TISS-lehtede nimekiri	5990	1487, 1607	14	OK (0) Neg (2) Kõik (2)	3.3.3.3 - toodang
28	RD1.14.4	Tehniline ülesanne - RD1.14.4 TISS-lehe trükikuva	5993	1415, 1607	13	OK (1) Neg (0) Kõik (1)	3.3.3.3 - toodang

4 vastet, read 1-4

Joonis 3. Jälgitavuse maatriks konfiguratsiooni haldustarkvaras *Changelogic*.

Jälgitavuse maatriks võimaldab vaadata projekti nõudeid, mis tööülesanded antud nõuete kohta on, missuguses muudatuses antud nõue või tööülesanne on realiseeritud, missugused testjuhtumid on seotud antud nõudega; kas testjuhtumid on õnnestunud või ebaõnnestunud ning lõpuks, millises versioonis mingi kindel nõue on realiseeritud.

2.2 Rakenduse kvaliteedi hetkeseisu jälgimine

Rakenduse kvaliteedi hetkeseisust saab ülevaate nii tööülesande- ja konfiguratsioonihaldustarkvaras kajastuva tööülesannete seisu järgi kui ka staatiliste koodi analüüsi vahenditega, mis annavad ülevaate arendatud koodi kvaliteedist. Rakenduse kvaliteedi seisu jälgimine peaks olema testijate igapäevane tegevus ning soovitavalt peaks iga hetk oskama öelda testitava rakenduse kvaliteedi hetkeseisu: näiteks, mis seisus on kood ja rakendus ning kui kaugel ollakse käesoleva tsükli arendusega.

2.2.1 Tööülesannete seis ning vigade arv

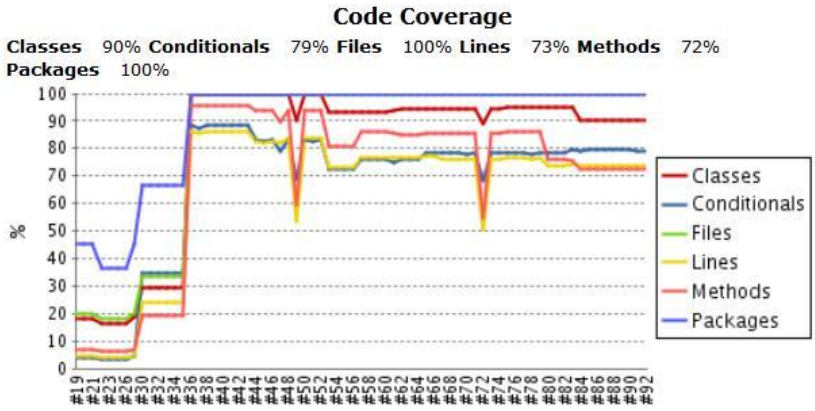
Kõik arendusega seotud tööülesanded ja leitud vead raporteeritakse tööülesannete- ja konfiguratsioonihaldustarkvaras. Tööülesannete- ja konfiguratsioonihaldustarkvara abiga saab rakenduse hetkeseisust hea ülevaate. Rakenduse kvaliteedi hetkeseisu on kõige lihtsam hinnata teostatud või teostamata tööülesannete ning leitud vigade arvu järgi. Näiteks, lugedes kokku tööülesandeid: kui palju funktsionaalsusest on juba arendatud ning kui palju on veel arenduse ootel, kui palju on testitud ja suletud, kui palju on testimise ootel või kui paljude järelkontroll ebaõnnestunud, kui palju on vigu raporteeritud ja parandatud, kui palju on kõrge prioriteediga vigu üleval jne. Nimetatud tööülesannete jälgimine annab hea ülevaate, kui kaugel projektiga ollakse ning missugune on projekti kvaliteedi hetkeseis.

Kuid tööülesandeid ja nendega tegelemist tuleb jälgida kogu arenduse jooksul, veendumaks, et ükski tööülesanne ei jääks mõne teise taha ootama, et toimuks pidev protsess ning et koguaeg oleks ülevaade jooksva iteratsiooni seisust.

2.2.2 Automaatsed kvaliteedikontrolli vahendid

Dünaamiline testimine ei pea olema projektis ainuke kvaliteedi tagamise tegevus. Staatilised koodianalüsaatorid annavad samuti hea ülevaate projekti kvaliteedi hetkeseisust. Staatilise analüüsi käigus uuritakse programmi lähtekoodi, eesmärgiga leida potentsiaalseid vigu programmis. Erinevalt dünaamilistest testimise meetoditest, staatilise testimise käigus programmi ei käitata. Staatilised analüsaatorid leiavad selliseid probleeme nagu näiteks mälu-vead, ebamäärased käitumised ja muud. Staatiliste analüsaatorite kasutamise eeliseks on, et nende rakendamine nõuab vähe vaeva ja aega ning nende abil avastatakse teatud vead palju varem kui muude testimise lähenemistega. See annab võimaluse vähendada projekti kulusid, kuna hilisemas etapis on vigadega tegelemine kallim. Muidugi ei ole ka staatiline analüüs kõikvõimas, sest analüsaatorid ei tea täpselt, millised väärtused on koodi käivitamisel võimalikud, seega nad peavad tegema teatud eeldusi, mis omakorda tähendab, et osa vigu jääb avastamata[9],[10],[11].

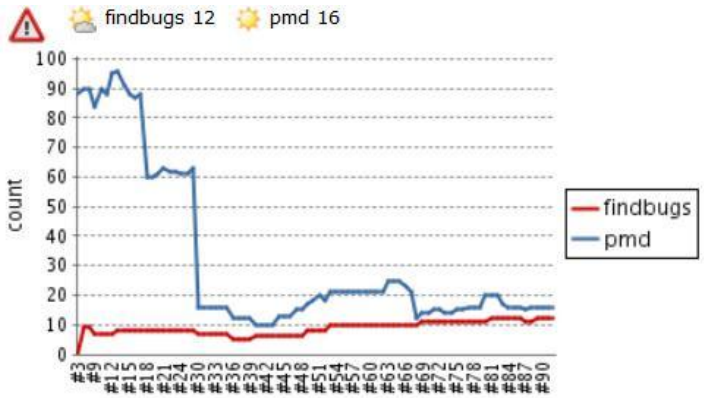
Antud töö autori projektis kasutatakse näiteks rakendust nimega *Jenkins* koos töövahenditega nagu *Findbugs* ning *PMD,* mis teostavad lähtekoodi staatilist analüüsi. *Jenkins* on avatud lähtekoodiga pideva integratsiooni server (*continuous integration server*), mis võimaldab arendusmeeskonnal lihtsasti jälgida muutusi ja vigu koodis: näiteks, kui järk (*build)* ebaõnnestub, kui kood ei kompileeru või mõned vajalikud ressursid puuduvad, rakendus ei paigaldu, koodi kaetavus testidega (*coverage*) on alla teatud piiri, *PMD* ja *Findbugs* teated on ületanud teatud piiri või kui koodis on liiga palju „TODO“ või „FIXME“ märgendeid [10],[11]. *Jenkins*'i reeglid on seadistatavad vastavalt projekti vajadustele ja koodikvaliteedi nõuetele. Kui projekti seis *Jenkins*'is muutub ebastabiilseks või üldse järgu tegemine ebaõnnestub, siis testija peaks reageerima juhtides sellele arendajate tähelepanu. Antud töövahend võimaldab järjepidevalt vaadata ka koodi/arenduse kvaliteedi kohta erinevaid trendide graafikuid, mis võiksid huvi ja ülevaadet pakkuda ka testijatele. Näiteks saab vaadata graafikuid selle kohta, kui palju teatud projekti või mooduli koodist on testidega kaetud erinevate järkude jooksul. (Joonis 4).



Joonis 4. Mooduli või projekti koodi kaetavuse graafik järkude lõikes [11].

Jooniselt 4 on näha, et paketid (*packages*) on ainsana kaetud 100%-liselt, tingimuslaused (*conditionals*) nagu *if-else* on testidega kaetud 79% jne. Samuti on näha, et arenduse alguses arendajad testide kirjutamisele suurt rõhku ei pane, vaid neid hakatakse kirjutama natuke hiljem.

Veel näitab *Jenkins*, kui palju vigu on avastatud analüsaatorite *findbugs* ja *PMD* poolt. Jooniselt 5 on näha, et arenduse alguses leiab PMD utiliit koodist rohkem probleeme ning mida järk edasi, seda enam leitud probleeme lahendatakse ning järjest vähem avastatakse uusi.



Joonis 5. *Findbugs* ja *PMD* poolt avastatud vead koodis erinevate järkude jooksul [11].

Koodi-analüsaatorite tulemustega tasub samuti arvestada rakenduse hetkeseisu üle otsustamisel, sest see annab ülevaate koodi hetkeseisu kvaliteedist, mis on samuti oluline rakenduse kvaliteedi faktor.

2.3 Testimise protsessi progressi jälgimine

Peale testimisplaani koostamist iteratsiooni alguses tuleb selles kajastatud tegevusi ja ajapiiranguid pidevalt jälgida ja võrrelda reaalsusega, kontrollimaks kuidas testimine tegelikult kulgeb. Testimise protsessi progressi jälgimisel on järgmised eesmärgid:

- Anda testimismeeskonnale ning projektijuhile tagasisidet, kuidas testimise kulgeb, võimaldades vajadusel suunata ja parandada testimise protsessi.
- Pakkuda projektimeeskonnale testimistulemuste nähtavust ja läbipaistvust.
- Mõõta testimise staatust ja testide katvust otsustamaks testimise lõpetamise üle.
- Koguda andmeid, hindamaks tulevasi testimise tegevusi.

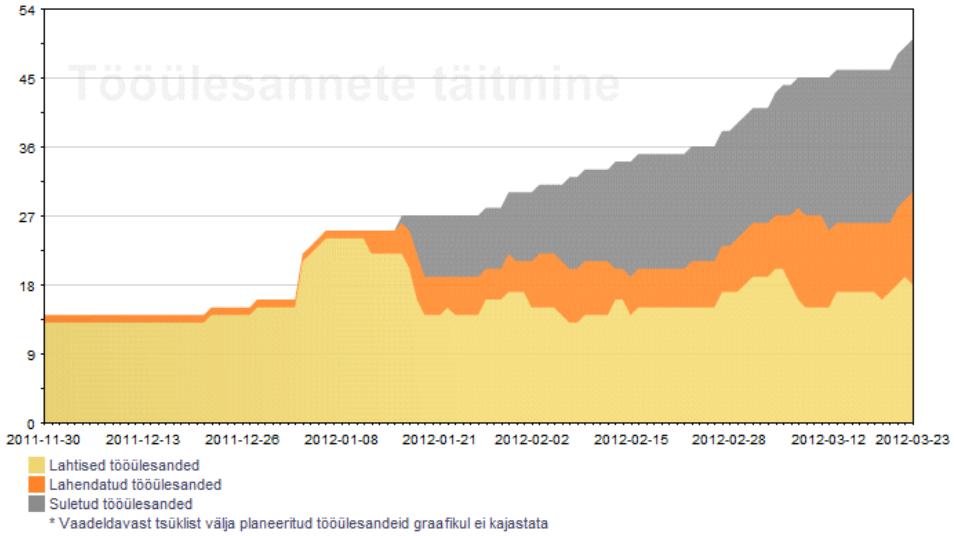
Kui klient tahab näha ülevaadet testimise tulemustest alles siis, kui testimine antud iteratsioonis on lõpetatud ehk vahetult enne tarnet, siis projektijuht vajab tavaliselt ülevaadet testimise protsessist pidevalt. Testijad ise peavad omama ülevaadet progressist väga regulaarselt, vähemalt igapäevaselt. Suures osas on testimise või kogu arenduse protsessi progressi jälgimisel abiks konfiguratsioonihaldustarkvara ja selle võimalused.

2.3.1 Protsessi graafikud

Infot, mida testimise protsessis monitoorida, saab koguda nii käsitsi kui ka automaatselt. Väikeste projektide puhul saab näiteks testija käsitsi lugeda kokku tööülesandeid või testjuhtumeid iga päeva lõpus. Suurte meeskondade, keeruliste projektide ning pikaajalise testimise puhul tõhusa ning järjepideva andmete kogumise puhul tulevad abiks automaatsed vahendid. Näiteks veahaldustarkvara abiga on võimalik koguda andmeid automaatse väljundina, kas lihtsalt vormistatud aruande kujul või isegi genereerituna graafikuks [12].

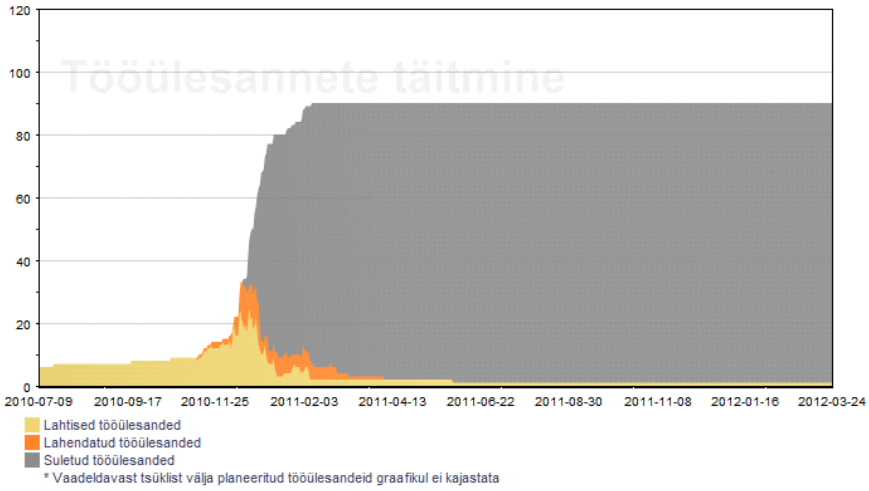
Tööülesannete haldussüsteem on võimas vahend jälgimaks toote kvaliteedi hetkeseisu ning progressi vastavust ajagraafikule. Juba eelnevalt nimetatud tööüleannete- ja

konfiguratsioonihaldustarkvara *Changelogic* võimaldab vaadata projekti protsessi graafikuid. Iteratsiooni keskel näeb tavaliselt protsessi graafik välja selline nagu Joonisel 6.



Joonis 6. Protsessi graafik programmis *Changelogic* iteratsiooni keskel.

Jooniselt on näha, et iteratsiooni alguses on antud iteratsiooni tööülesannete arv väiksem ning aegamööda tööülesannete arv kasvab. See on sellepärast nii, et tavaliselt kogu analüüsi ei tehta korraga, vaid kui osa on tehtud, siis arendusele minevad tööülesanded sisestatakse tööülesannete- ja konfiguratsioonihaldustarkvarasse ja arendajad saavad nende ülesannetega juba tegelema hakata. Iteratsiooni arenedes tehtud analüüsi osa kasvab ning tekib ka tööülesandeid juurde. Alguses on kõik tööülesanded lahtises olekus (kollane ala), kuid mida aeg edasi, seda rohkem ülesandeid saab arendajate poolt lahendatud (oranž ala) ning testijate poolt ka suletud (hall ala). Ideaalses olukorras iteratsiooni lõpuks peaksid kõik antud iteratsiooni tööülesanded olema suletud, st graafikul kollased ja oranžid alad peaksid samuti olema hallid. Protsessi graafik iteratsiooni lõpul näeb üldjoontes välja selline nagu Joonisel 7.



Joonis 7. Protsessi graafik programmis *Changelogic* iteratsiooni lõpul.

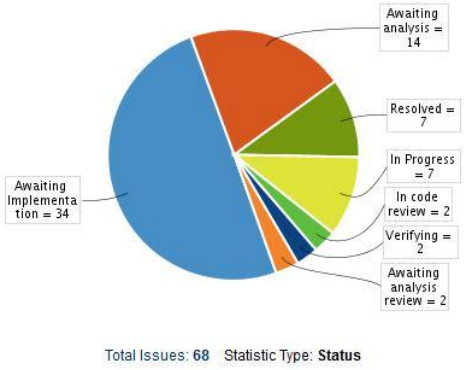
Antud jooniselt on näha, et alates kuupäevast 03.02.11 enam iteratsiooni tööülesandeid juurde ei lisandunud. Samuti on näha, millal viimased arendajate poolt lahendatud tööülesanded testijate poolt suleti. Antud jooniselt on samuti näha, et ühte kuni kahte tööülesannet ei ole kuni iteratsiooni lõpuni lahendatud ja need on siiani lahtises olekus. Ilmselt on jäänud antud tööülesanded järgmisse tsüklisse edasi suunamata või on tegu lihtsalt mõne tööülesande duplikaadiga, mis on jäänud sulgemata.

2.3.2 Meetrikad/mõõtmised

Kõigepealt tuleks jälgida kõige lihtsamaid meetrikaid. Näiteks, kõik testijad peaksid teadma tsüklis tehtavate tööülesannete arvu, nende hetkeseisu ning testimise aega ja kuupäeva. See on põhiline informatsioon, mida iga testija mingil viisil jälgib [13]. Põhilised meetrikad, millega testimise progressi mõõta:

- Kui palju tööülesannetest on arendatud, kui palju on veel arendamata, kui palju hetkel arendajate käes arendamisel.
- Kui palju tööülesandeid on testitud, kui palju veel testimise ootel.
- Kui palju tööülesandeid on analüütikute käes ja vajavad veel analüüsi.
- Kui palju tööülesandeid on suletud (st järelkontroll korras), kui palju tagasi lükatud.
- Kui palju vigu on leitud ja ootel.
- Kui palju vigu on parandatud.
- Iteratsiooni kuupäevad [12],[14].

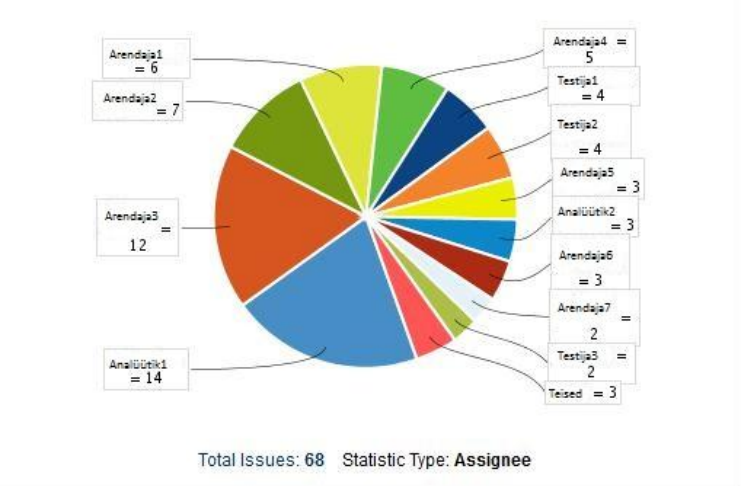
Mõned meetrikate kogumised tehakse ära tööülesannete haldustarkvara poolt. Näiteks töövahend *Jira* loeb ise kokku kõik tööülesanded ja paigutab need graafikusse, kus on näha, mis staatuses mingi tööülesanne parasjagu on.



Joonis 8. Tööülesanded jagatud staatuse järgi programmis *Jira* [15].

Jooniselt 8 on näha, et pool kogu käimasoleva tsükli tööülesannetest on veel arenduse ootel, ligi viiendik tööülesannetest on analüüsi ootel, ligi 10% tööülesannetest on lahendatud ja veel 10% on hetkel arenduses jne.

Teine võimalus on lugeda tööülesandeid meeskonnaliikmete lõikes (Joonis 9). Jooniselt 9 on näha, kellel meeskonnast on kõige rohkem tööülesandeid.



Joonis 9. Tööülesanded jagatud omaniku järgi programmis *Jira* [15].

Tavaliselt mõõtmised tehakse algandmete pealt, nagu tööaeg, vigade raportid, tööülesanded. Otsesed mõõtmised on need, mis saadakse otse algandmete pealt, näiteks lugedes kokku tööülesandeid, mis on suletud või teostusel. Enamusel otseste mõõtmiste tulemustel pole aga mingit tähendust, kui neid ei paigutata millegagi seosesse. Näiteks, lihtsalt vigade arv 50 ei ütle mitte midagi testimise protsessi või arendatava süsteemi kohta. Kaudsed mõõtmised on mõõtmised, mida saab arvutada otseste mõõtmiste pealt, muutes antud andmed palju kasulikumaks. Kaudsete mõõtmiste abil saab testimise protsessi progressi jälgida mitmel erineval tasandil (nt testija, mooduli või projekti kohta). Meetrikate valik peab põhinema seatud eesmärkidel ning teistel küsimustel, millele vastuseid soovitakse. Näiteks, kui kaugel ollakse konkreetse ülesande täitmisega plaani ja ootuste suhtes. Testimise meetrikad ja nende abil saadud tulemused on olulised ka testimise lõpetamise otsuse juures. Meetrikaid, mida on võimalik kasutada testimise lõpetamise juures, kirjeldatakse antud töö järgmises peatükis.

Monitoorides testimise progressi, saadakse infot, kuidas testimise protsess kulgeb. Mõnikord, kui tulemused ei ole sellised nagu oodatud, tuleb kasutusele võtta testimise protsessi kontrollimise meetmed *(test control)*. Testimise kontrollimine tähendab igat suunamist või parandusmeetmeid, mida tehakse kogutud meetrikate tulemuste alusel. Testimise kontrollimise tegevusteks on: otsuste tegemine testimise protsessi monitooringu tulemuste

alusel, tööülesannete prioriteetide muutmine mingi kindla riski olemasolul (nt kui graafikust ei suudeta kinni pidada). Kui on näha, et plaanitud iteratsiooniga ei saada ettenähtud ajagraafiku raames valmis, siis on peamiselt 3 varianti: lükata tarne edasi, suurendada/kaasata rohkem ressursse, vähendada skoopi, lükates osa töid edasi järgmisse iteratsiooni. Antud otsuseid teevad projektijuhid ning tihti just testijatelt või juhtivtestijatelt saadud info alusel. Mõnikord skoobi vähendamine on kõige kasulikum otsus – näiteks, kui on näha, et mingi kindla vea parandamine võib olla kriitilise mõjuga, ent viga ise enesest ei ole väga kriitiline, siis sellise vea paranduse võib otsustada lükata järgmisse iteratsiooni [2],[16].

Vigasid, mis veahaldustarkvaras antud tsüklis veel lahendamata on, jälgitakse igal nädalal. Eesmärgiks on leida näiteks probleeme kommunikatsioonis, vigade klasterdumist, mis viitab kehvale/nõrgale koodi alale või näiteks üksikuid vigu, mis lihtsalt ei lähe ära, vaatamata sellele, kui mitu korda neid parandatud on [16]. Testimise protsessi progressi kohta võib luua ka dokumendi, kus on kirjas näiteks antud nädala saavutused ning probleemid, takistused. Antud dokumenti kirjeldatakse käesoleva töö viimases peatükis.

2.4 Regressioontestimine

Paljud uurimused näitavad, et umbes 50% kõikidest muudatustest, mis süsteemis tehakse, tekitavad süsteemi uusi vigu [7]. See näitab, et paranduste või muudatustega süsteemis on seotud märgatav risk. Üks osa vigadest on vead muudatuses, mida parasjagu tehti, nt koodi kirjutamise vead, muudatuse disaini vead jms, teine osa vigadest aga tekib ootamatutest koostoimetest allsüsteemidega. Regressioontestimine on just selliste vigade avastamiseks. Regressioonteste tehakse eesmärgiga, et näha, kas muudatused ühes süsteemi otsas on katki teinud muid süsteemi osi. Näiteks, kas muudatused, mis tehti vigade paranduseks või täiendusteks, tekitavad vigu kusagil mujal süsteemi osas, kus jooksvas iteratsioonis muudatusi tegelikult ei tehtud. Regressioontestimine kujutab endast kogu rakenduse või selle kõige olulisemate protsesside läbi testimist, nii käsitsi testimise kui ka automaattestide abil.

Erinevad allikad annavad regressioontestimise rakendamise ajast arendustsüklis erinevat infot. Arendustsüklis tehakse regressioonteste tavaliselt arenduse lõpuosas, siis kui arendused on tehtud ja enne kui hakatakse versiooni kasutajale üle andma. Regressioontestid on tavaliselt üleandmistestide ja kasutajatestide alamosa [7]. Allika [17] kohaselt on aga regressioonitestimise kontseptsioon muutumas - kui varem oli regressioontestimine viimaseks väravaks enne toodangusse minekut, siis nüüd käsitletakse seda kui pidevat tegevust terve iteratsiooni käigus [17].

Praktika näitab, et suuremal määral tegeldakse regressioontestimisega siiski pigem iteratsiooni lõpuosas, kuid on oluline planeerida piisavalt aega ka regressioontestimise käigus avastatud vigade parandamiseks ning üle testimiseks. Regressioontestimise tulemusena valminud tabeli kujul dokumenti arvestatakse tihti kliendi tarne raporti osana. Antud dokumenti on võimalik kasutada rakenduse kaardistamiseks ning projektist ja selle testimise tulemustest parema ülevaate saamiseks.

Kuna regressioontestimine on enamikes projektides väga ressursimahukas tegevus - erinevad uuringud näitavad, et koguni 80% testimise kuludest kulub regressioonitestimisele [17] - siis on oluline regressioontestimise ressursimahukust vähendada. See on võimalik erinevate regressioontestimise lähenemistega.

2.4.1 Kogemustel põhinev lähenemine

Kogemustel põhinev regressioontestimine on enamlevinud regressioontestimise viis. Tavaline lähenemine regressioonitestimises on, et kasutatakse rakenduse protsesside või testjuhtumite hulka, mis on paika pandud testijate poolt. Iga protsessi jaoks määratakse protsessi olulisus, testjuhtum ja automaattest, juhul kui see on olemas. Regressioontestimise jooksul määratakse ka testide tulemus, mis hakkab näitama protsessi seisu. Selleks koostatakse ja täiendatakse jooksvalt rakenduse protsesside või testjuhtumite nimekirja kogu projekti eluea jooksul. Joonisel 10 on esitatud näide regressioontestimise protsesside nimekirjast.

Protsess	Prioriteet	Automaattest	Testjuhtum	Tulemus
Asutuse otsimine	1	ee.project.asutus.otsing.test	Testjuhtum 100	OK
Asutuse lisamine	1	ee.project.asutus.lisamine.test	Testjuhtum 101	Not OK
Asutuse kustutamine	1	ee.project.asutus.kustutamine.test	Testjuhtum 102	Not OK
Asutuse muutmine	1	ee.project.asutus.muutmine.test	Testjuhtum 104	Not OK
Kindlustuse lisamine	1	ee.project.kindlustus.lisamine.test	Testjuhtum 104	OK

Joonis 10. Regressioontestimise tabel.

Kogemusel põhinev regressioontestimise protseduur põhineb tavaliselt individuaalsel kogemusel ja otsusel, mistõttu ei ole see piisavalt läbipaistev, et võimaldada järjepidevalt hinnata regressioontesti ulatust ja kvaliteeti. Selline lähenemine muudab regressioonitestimise väga suurel määral sõltuvaks inimestest, kellel on antud süsteemi ja testjuhtumitega kogemusi. Testija peab teadma, millised testjuhtumid on olnud viimati kõige vearohkemad/altimad. Veel on probleemiks see, et ka kogenud testija võib valida ebaefektiivse testide kogumi. Enamasti testid valitakse rutiinsel viisil, valides igaks regressioontestiks sama testide hulga ning kuna see põhineb ainult testija otsusel, siis ei ole mingit tõendit, et see on kõige efektiivsem testide hulk [17].

Iga uue täienduse kohta süsteemis tehakse mitu uut testjuhtumit. Kuna täienduste arv projektis muudkui kasvab, siis järjest kasvav testjuhtumite hulk on liiga suur, et regressioontestimise käigus testida läbi kõik võimalikud testjuhtumid. Igas iteratsioonis ei olegi vaja täielikku regressioontesti, piisab ka osalisest, kuid testkomplekti tuleb osata valida [4]. Regressioontestimise komplekti valitakse tihti näiteks koostöös projekti arhitektiga, kellel on

tehnilisemad teadmised ning kes üle vaadates oskab teostatud tööde nimekirja paremini hinnata, missugused tööd võivad missuguseid süsteemi teisi osi mõjutada.

2.4.2 Ajalool põhinev lähenemine

Regressioonitestimise protsesside või testjuhtumite komplekti valik vajab süstemaatilist ja läbipaistvast strateegiat, sest see aitab kaasa ressursside tõhusamale kasutamisele. Artikli [17] raames viidi läbi juhtumiuuring, kus kasutati hoopis ajalool põhinevat lähenemist regressioon testimisele, eesmärgiga parandada regressioontestimise testide valiku läbipaistvust ja prioriteete. Regressioontestimise uurimus eristab kolme klassi tehnikaid: minimeerimine, valik (selekteerimine) ning prioriteetide määramine. Minimeerimise eesmärk on vähendada testide arvu eemaldades üleliigsed ja vananenud testjuhtumid, selekteerimise eesmärk on identifitseerida testide hulk, mis on piisav ning prioriteetide määramisega hinnatakse testjuhtumeid, mis suurema tõenäosusega leiavad vigu.

Uuringu hüpoteesiks oli, et poolautomaatne testide valimise vahend, mis põhineb ajalool põhineval testide valikul, pakub eelnevalt nimetatud probleemidele lahendust. Tehnika, mille pakkus välja *Fazlalizadeh* ning mida nimetatakse *Faz* lähenemiseks, põhineb kolmel faktoril: ajalooline tõhusus vigade avastamisel, iga testjuhtumi teostamise ajalugu regressioontestimisel ning testjuhtumi määratud prioriteet. Testjuhtumite prioriteedid arvutatakse järgmise valemi alusel:

$$PR_k = \frac{\alpha * f_{ck}}{e_{ck}} + \beta * PR_{k-1} + \Upsilon * h_k,$$
$$0 \leq \alpha, \beta, \Upsilon \leq 1, k \geq 1$$

kus:

Ajalooline tõhusus (f_{ck}/e_{ck})*:* f_{ck} on number, mitu korda testjuhtum on läbikukkunud ning e_{ck} on number testjuhtumi käitamisest k -nda testsessiooni ajal.

Teostamise ajalugu (h_k)*:* iga kord, kui testjuhtumit ei käitata või teostata, selle teostamise ajaloo arv suureneb ühe võrra. Kui testjuhtum teostatakse, siis number saab nulliks ja nii protsess kordub.

Eelnev prioriteet (PR_{k-1}): on testjuhtumi viimane prioriteet. *Kolm osakaalu parameetrit* ($\alpha, \beta\ ja\ \Upsilon$): parameetreid $\alpha, \beta\ ja\ \Upsilon$ saab muuta, et tasakaalustada tegurite mõjusid. Nende väärtused kuuluvad vahemikku nullist üheni.

Esialgseks testjuhtumi prioriteediks PR_0 võib olla määratud näiteks prioriteet, mis igale testjuhtumile on määratud selle loomisel. Testjuhtumitele antakse järgmine PR_0 väärtus:

$$PR_0 = \begin{cases} 0.4 & kui\ prioriteet\ on\ 1 \\ 0.2 & kui\ prioriteet\ on\ 2 \\ 0.1 & kui\ prioriteet\ on\ 3 \end{cases}$$

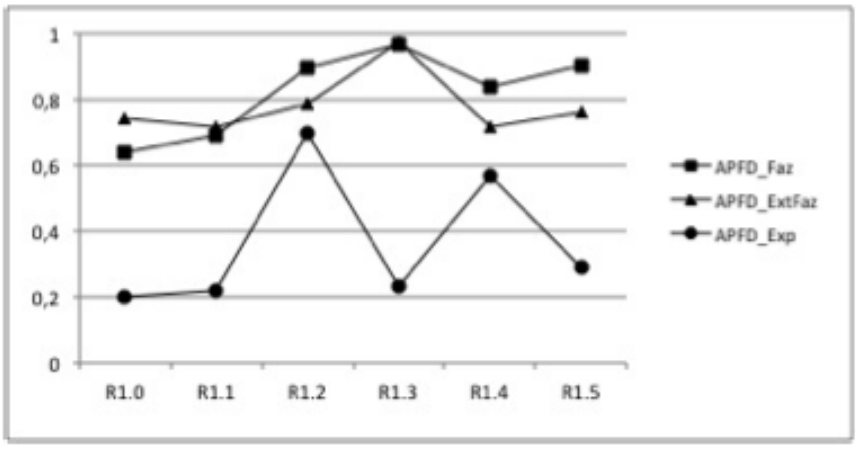
Püüdes lisada teisi soovitud tegureid antud valemile, sai loodud laiendatud versioon antud lähenemisele. Faz lähenemise laiendatud versioon on nimetatud $ExtFaz$ lähenemiseks. Selle kohaselt lisati veel kaks tegurit testjuhtumite prioritiseerimisele ja valikule: staatiline prioriteet ning testjuhtumi vanus. Laiendatud valem on järgmine:

$$CalculatedPriority_k = PR_k + N_k + PR_0,$$

kus:

$$N_k = \begin{cases} 0.4\ \ kui\ hetkekuupäev - loomisekuupäev < 3\ kuud \\ 0\ kui\ hetkekuupäev - loomisekuupäev\ > 3\ kuud \end{cases}$$

Nii Faz kui $ExtFaz$ valemi rakendamise jaoks loodi töövahend, kus kasutajaliideses on võimalik määrata, millist valemit kasutada, määrata testjuhtumite arv ning millist tüüpi testjuhtumid välja jätta. Seejärel võrreldi kõiki kolme strateegiat - praegust kogemustel põhinevat (Exp), Fazlalizadeh'i väljapakutud strateegiat (Faz) ning viimase laiendatud strateegiat ($ExtFaz$). Andmed koguti kuue järjestikuse tarkvara versiooni regressioonitestimise sessiooni pealt ning analüüsiti testide käitamise järjekorra ning selekteerimise mõju regressioontestimisele loodud vahendi abiga. Graafik tulemustest on näha Joonisel 11.



Joonis 11. Keskmine vigade avastamise protsent erinevate meetodite abiga [17].

R1.* - on tarkvara erinevad versioonid

APFD (*Average of the Percentage of Faults Detected*) - keskmine vigade avastamise protsent

Jooniselt 11 on näha, et ajalool põhinevad tehnikad käituvad paremini kõigi kuue regressioontesti sessiooni puhul. Ka uuringus osalenud testijad jäid üldiselt ajalool põhineva regressioontestimise ideedega rahule. Siiski on automaatselt valitud testkomplektil ka puudujääke. Näiteks võib sellisel viisil valitud komplekt sisaldada ka ebavajalikke teste. Põhjuseks võib olla näiteks see, kui testjuhtumite prioriteeti veahaldustarkvaras ei uuendata, kui see tegelikult muutub. Seega ka automaatselt valitud testkomplekt tuleks testijate poolt käsitsi üle vaadata, et leida puudujäägid. Samuti on antud meetodi kasutamine väikeste ja keskmiste projektide puhul liiga ajamahukas, mistõttu ei tasu end ära.

3. Testimise lõpetamine

Vigane tarkvara võib kaasa tuua suure ajakaotuse, eelarve kahjumi, turgude kaotamise ning võib nõuda isegi inimelusid. Edukalt arendatud tarkvaratoode või rakendus võib tuua märkimisväärset kasumit. Seetõttu pinge tarnida kõrge kvaliteediga tarkvararakendusi nõuab tarkvara arendusprotsessist paremat arusaamist ning parandatud/täiendatud tarkvara töökindluse mudelit [18].

Testimine on peamine vahend, et avastada puudused tarkvaratootes ning sellega parandada tarkvaratoodete usaldusväärsust ja töökindlust. Kuid usaldusväärsuse suurendamisega suureneb plahvatuslikult ka kulude kasv. Testimiseks on väga harva piisavalt aega ja raha, et saavutada testimise kvaliteeti, mida soovitakse. See võtab tohutu hulga ressursse, et testida ammendavalt, mistõttu pole see seda väärt. Seetõttu on väga oluline aru saada ning kindlaks määrata, millal testimine lõpetada. Kui tarkvara tarnitakse liiga vara, siis on oht, et toote kvaliteet pole piisavalt hea ning klient ei ole sellega rahul. See tõstab tarkvara kulusid, kuna rakenduse vigu peab parandama kasutusfaasis, mis on kordades kallim kui parandamine testimisfaasis. Samas, kui tarkvara testitakse liiga kaua ja tarnitakse hilja, siis on väga suur risk, et ületatakse aja ja eelarvepiiranguid ning on oht saada kliendilt trahvi hiljaks jäänud tarne eest. Testimise lõpetamise ja tarneks valmisoleku otsus põhineb tarkvara töökindluse, usaldusväärsuse ja kulude hindamisel [18],[19].

Järgnevalt kirjeldatakse põhilisi testimise lõpetamisega seotud tegevusi ja kriteeriumeid ning meetrikate abil kogutud analüüsitavaid andmeid. Samuti on võimalik testimise lõpetamist ennustada erinevate lähenemistega.

3.1 Testimise lõpetamine projektis

Ideaalselt peaks testimise maht sõltuma tarkvarale esitatud nõuetest. Testitakse seni, kuni need on rahuldatud [20]. Praktiliselt tehakse nii vaid tõeliselt kriitiliste rakenduste korral. Põhjusi on palju, näiteks nõudeid ja nende rahuldatust on raske hinnata; töö tuleb kiiresti üle anda; on olemas eelnev kogemus ja see määrab testimise mahu; rakendus ei ole kriitiline (kui midagi juhtub, keegi eriliselt ei kannata) jne [20]. Seega igas projektis testimise lõpetamise tegevused võivad mingil määral varieeruda. See oleneb testijate enda kogemusest ja harjumustest: sellest, mida nõuab projektijuht või klient, üldisest testimispoliitikast ning projekti tüübist ning olemusest. Kuid on olemas üldised heaks kiidetud tegevused, mida ühe iteratsiooni lõppedes oleks kasulik teha.

3.1.1 Testimise lõpetamise tegevused

Testimise lõpetamine koosneb tavaliselt tegevuste ja tulemuste analüüsist ning tulemuste raporteerimisest. Testimise lõpetamise tegevuste eesmärk on koguda kogemused ning arhiveerida testimise varad tulevaseks kasutamiseks. Testimise lõpetamise tegevuste sisendiks on antud tsükli või iteratsiooni testplaan ning testimise varad, sealhulgas testimiskeskkonnad. Testimise lõpetamise protseduur sisaldab lõplikku tarnete ning tööülesannete kontrolli, testimise varade hoiustamist ning eelnevatele tegevustele tagasivaatamist. Tulemuseks on testimise raport ning nimekiri antud iteratsioonis teostatud tööülesannetest.

Testimise lõpetamise peamised tegevused:

1) Kontrollida, kas kõik testimise lõpetamise kriteeriumid on täidetud

Selle etapi raames vaadatakse üle kõik tööülesanded, kõik prioriteetsed ja olulised ülesanded peavad olema kinni pandud või järgmisse iteratsiooni edasi lükatud. Juhul, kui mõnda ülesannet ei ole võimalik teostada või kontrollida, siis tuleb sellest klienti kiiresti teavitada. Testimise lõpetamise kriteeriumidest saab pikemalt lugeda järgmises alampeatükis.

2) Artefaktide arhiveerimine

Testimise käigus loodud ning tekkinud varad ning artefaktid on meeskonnale väga kasulik info ning sellega tuleks hoolikalt ümber käia. Loodud testimise vara tuleks hoida; seda eesmärgiga, et testimine oleks tulevikus lihtsam ning ökonoomsem. Seega, selles etapis kõik dokumendid, testjuhtumid ja automaattestid, mida antud tsüklis loodi, kogutakse kokku ja arhiveeritakse. Kui testimise varasid ei hoita, on tegemist raha ja aja raiskamisega [2].

3) Tagasivaatav koosolek

Antud koosolekul analüüsitakse eelmisel etapil saadud tulemusi ja tehakse vastavad järeldused. Analüüsi eesmärk on saadud kogemuste põhjal teha vajalikud parandused järgnevasse protsessi nii vara kui võimalik, et vältida samasuguste vigade kordamist. Koosolekul raporteeritakse kogemustest ning mõõtmistest, mis antud testimise tsükli käigus omandati või tehti. On oluline, et testijad lõpetaksid testimise korralikult, koostades raporti. Testimise lõpetamise raportist saab lugeda töö viimases peatükis.

Kogu protsessi tegevuste parandamise huvides on oluline, et ka kõrgem juhtkond on kaasatud, kes küsib ning aktiivselt ka kasutab testimise raportit. Vastasel juhul antud koosolekut ei peeta ning inimesed süvenevad kiiresti uude testimise tsüklisse ning unustavad eelmise.

4) Testimise lõpetamise meetrikate kogumine

Meetrikad, mis on seotud testimise lõpetamisega, võivad sisaldada järgmist infot: kui palju tööülesandeid alustati teatud aja jooksul, tööülesannete täitmise protsent aja jooksul, kui palju tööülesanded teostati teatud aja jooksul ning kui palju aega kulus iga tööülesande peale. Neid meetrikaid tuleb võrrelda testimise lõpetamise esialgsete hinnangute ning graafikuga [2]. Testimise lõpetamisega seotud meetrikatest on kirjas järgmises testimise lõpetamise alampeatükis.

3.1.2 Testimise lõpetamise tingimused/kriteeriumid

Testimise lõpetamise kriteerium (*exit criteria)* põhineb riskide analüüsil - mida kõrgem risk, seda rangemad on testimise lõpetamise kriteeriumid; mida madalam risk, seda vähem nõudlikud ning vähem konkreetsemad on testimise lõpetamise kriteeriumid. On oluline paika panna, millised testimise lõpetamise kriteeriumid peavad täidetud olema enne testimise lõpetamist.

Testimise lõpetamise kriteeriumiteks võivad olla:

- Iteratsiooni testplaanis hinnatud riskid on piisavalt maandatud.
- Kõik iteratsiooni tööülesanded on suletud või planeeritud järgmisse tsüklisse.
- Projektijuht või juhtivtestija otsustab, et rakendus on valmis tarnimiseks.
- Määratud kaetavus on saavutatud või töökindlus on jõudnud määratud künniseni.
- Tarkvara töökindluse kasv ei õigusta testimise kulusid.
- Ühtegi teadaolevat kriitilist viga pole üleval [2].
- Testimise aeg on otsa saanud.

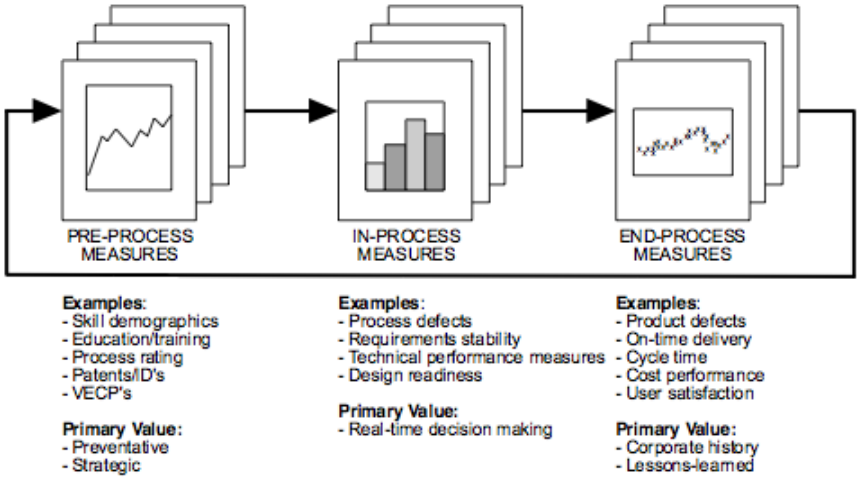
Viimane punkt ei ole ametlik testimise lõpetamise kriteerium ja seda ei tuleks kasutada, kuid tegelikus elus on see nii mõnigi kord testimise lõpetamise üheks tingimuseks.

Kui testimise lõpetamise kriteeriumid ei ole täidetud, siis ei saa ka testimist veel lõpetada. Kui kriteeriumid ei ole täidetud, siis tuleb tegevusi korrata, mis tagaksid, et kriteeriumid oleksid täidetud.

3.2 Meetrikad

Nii nagu tarkvara arenduses mõõdetakse üldiseid edusamme funktsionaalsete ja ettevõtte eesmärkide seisukohalt, nii ka tarkvara testimist tuleb mõõta, et saada teada selle progressist ning osata teha teadlikke otsuseid [13].

Meetrikaid defineeritakse kui "Mõõtmise standardid" ning neid on IT tööstuses kasutatud tulemuslikkuse ja efektiivsuse mõõtmiseks juba pikka aega [13]. Meetrikad on projekti tegevuste, ressursside ning tulemuste andmete mõõtmised. Testimise meetrikaid saab koguda terve testimisprotsessi ajal: nii testimist planeerides, testimist teostades, kui ka testimist lõpetades. Jooniselt 12 on näha ühe programmi mõõtmised ning mõõtmiste kasu kolmes erinevas protsessi etapis.



Joonis 12. Näide kosmosesüstiku programmi elutsükli mõõtmistest ning mõõtmiste kasust [21].

Antud peatükis keskendutakse testimise protsessi lõpetamisega seotud mõõtmistele. Kui testimismeetrikad on testimisprotsessi jooksul kogutud ja pidevalt uuendatud ning raporteeritud, siis on testimise lõpetamisel antud informatsiooni põhjal võimalik teha trendide analüüsi, mis on kasulik uurimaks protsesside muutumist erinevates projektides. Meetrikaid kasutades saab hinnata projekte, mõõta projektide progressi ja tulemuslikkust, mõõta

lõpptoote omadusi. Ilma meetrikateta ei pruugita teada, kas projekti seis paraneb või halveneb, kas projekt õnnestub või ebaõnnestub [13].

3.2.1 Meetrikate tüübid ja kasutamine

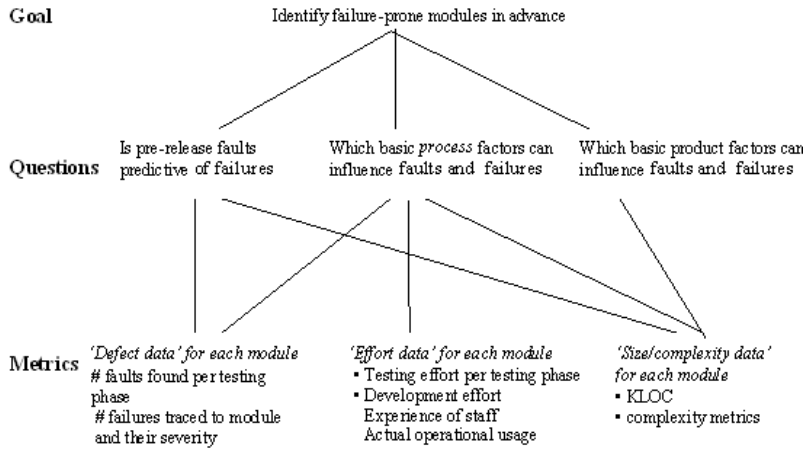
Testimise planeerimisel määratakse paika kindlad ja mõõdetavad eesmärgid. Ent ainult sellest ei piisa, et eesmärgid oleksid mõõdetavad. Tuleb koguda ka fakte, mis ütleksid, kas eesmärgid on saavutatud. Andmete kogumisega seoses eristatakse kolme mõistet:
Meetrika - defineerib, mida mõõta, sisaldades andmetüüpi, ulatust ja ühikut.
Mõõtmise meetod - kirjeldus, kuidas andmed saadakse.
Mõõtmised - tegelikud väärtused, mida meetrikate jaoks kogutakse [2].

On olemas erinevat tüüpi meetrikaid:
1) projekti meetrikad;
2) protsessi meetrikad;
3) toote meetrikad.

Projekti meetrikad on projekti seisundi hindamiseks; näiteks, planeeritud vs tegelike ajagraafikute kuupäevad. Protsessi meetrikatega mõõdetakse protsessi efektiivsust; näiteks, produktiivsus või vigade eemaldamise efektiivsus. Toote meetrikad on rakenduse suuruse, keerukuse ja kvaliteedi kohta. Antud töö seisukohast on kõige olulisemad toote meetrikad, mis on seotud kvaliteediga; näiteks iteratsiooni lõpus arvutatud raporteeritud vigade tihedus, vigade koguarv, mis raporteeritakse kliendi poolt toodangusse paigaldatud rakenduse kohta [22].
Meetrikate põhilised kasutusalad tarkvaraarendusprojekti juures on: näidata projektijuhile, kui kaugel või valmis on projekt; pakkuda informatsiooni, millele otsused rajada; pakkuda hinnangute aluseid tulevastele projektidele; pakkuda juhtkonnale informatsiooni valminud või lõpetatud toote kvaliteedi ja töökindluse kohta.

Meetrikate kasutamine on planeeritud tegevus täitmaks mingit kindlat eesmärki. Kui eesmärk on määratud, siis on järgmiseks kasulik formuleerida mõned küsimused, mis aitavad selgitada

seatud eesmärki. Küsimused omakorda annavad vihjeid meetrikatele, mis aitavad neile vastata. Näidet antud lähenemisest, mida nimetatakse eesmärk-küsimus-meetrika lähenemiseks (*Goal-Question-Metric approach*) on näha Joonisel 13.



Joonis 13. Näide eesmärk-küsimus-meetrika lähenemisest [23].

GQM lähenemine koosneb neljast faasist: kõigepealt valitakse projekt, mida hakatakse mõõtma, seejärel defineeritakse mõõtmise programm (sõnastatakse eesmärgid, küsimused, meetrikad ning hüpoteesid) , peale mida hakkab toimuma andmete kogumine ning lõpuks toimub tõlgenduste tegemine [23].

Testimise lõpetamise seisukohast on olulised järgmised kaudsed/arvutatud meetrikad:

- kui suur protsent tööülesannetest on teostatud;
- kui suur protsent tööülesannetest on testitud;
- kui suur protsent tööülesannetest on õnnestunud;
- kui suur protsent vigadest on parandatud;
- üleüldine ebaõnnestumise määr;
- vigade avastamise määr;
- vigade parandamise kulu.

Meetrikaid saab kasutada ka kui kvaliteedikriteeriume [20]. Kui seatud kriteeriumid on täidetud, siis saab otsustada testimise lõpetamise üle.

3.2.2 Meetrikate tasemed

Testimise meetrikaid on erinevate tasemetega, mistõttu kerkib küsimus, milliseid kasutada. Luues efektiivset meetrikate programmi, siis suurim väljakutse ei ole seotud mitte valemite, statistika ning keerulise analüüsiga, vaid hoopis projekti jaoks väärtuslike meetrikate määramisega. Mida lihtsamad on meetrikad, mida kasutatakse, seda parem. Seda seetõttu, et väga keeruliste algoritmide kasutamine ei pruugi olla ressursside ning aja suhtes mõistlik. Jõupingutused, mida tehakse informatsiooni saamiseks, ei pruugi olla mõistlikud protsessi täiustamisest saadud kasuga võrreldes[13].

Jerry Weinberg jagab mõõtmised kolme suure kategooriasse: esimest, teist ning kolmandat järku mõõtmised. Esimest järku mõõtmised on need, mida on vaja alustamiseks; seda tüüpi mõõtmised kipuvad olema kvalitatiivsed, kiired ja odavad ning need tavaliselt ei vaja erilisi mehhanisme. Esimest järku mõõtmised on märkamatud ja ei vaja palju vaeva, pakuvad palju olulist informatsiooni, mis võivad juhtida uue informatsioonini või kohesele tegutsemisele. Need mõõtmised vastavad küsimustele: „Mis on toimumas?“, „Mida peaks nüüd ette võtma“. Antud järku mõõtmised põhinevad kogemusel ja tunnetusel. Esimest järku mõõtmistelt saab edasi liikuda teist järku mõõtmistele, mis pakuvad suuremat täpsust. Teist järku mõõtmised vastavad küsimusele „Mis tegelikult toimub?“, „Kuidas miski muutub?“. Need mõõtmised kalduvad olema rohkem kvantitatiivsed ning täpsemad. Kolmandat järku mõõtmised on kõige täpsemad ning väga kvantitatiivsed, nende abil avastatakse loodusseadusi, otsides vastusi küsimustele
„Mis alati juhtub?“ [24].

Tarkvaraarenduses püütakse tihti teha kolmandat järku mõõtmisi, kuigi esimest ja teist järku mõõtmised oleksid piisavad. Kõik tarkvaraprojektid hõlmavad inimkonteksti – toimub suhtlus erinevate klientide ning meeskonnaliikmete vahel, suhtlus erinevate ülesannete ja probleemide lahendamisel. Sellistes keskkondades kolmandat järku mõõtmised ei ole saavutatavad, see on kallis ning pigem segav.

Kui vaadata projekti lõpus mõõtmistest saadud tulemustele peale, siis tasuks meeles pidada, et ainult numbrite kokkulugemisest on vähe kasu. Tuleb minna süvitsi, prioritiseerida, õppida tundma, mis on numbrite taga. Tulemused või järeldused meetrikatest ei pea olema tingimata ainult numbrilised, sest number ei kirjelda nii palju kui sõna. Sõnad võivad olla küll mitmetähenduslikud ja ähmased, kuid numbrid ilma selgitusteta ja täpse teadmiseta, mis nende taga peitub, võivad olla veel hullemad. Arvamused on küll subjektiivsed, kuid kui liige on tiimis ja arendusprotsessis tõeliselt sees, siis on see subjektiivsus suure tõenäosusega objektiivsem kui suvaline number. Küsida tuleks erinevate meeskonnaliikmete arvamust ja panna pilt toote valmisolekust üheskoos kokku.

3.3 Riskid

Kuna testimine on tarkvara arenduses viimane tegevus enne toodangusse minekut, siis selles etapis on tavaliselt kõige vähem aega. Kui aeg hakkab otsa saama ja arendus toodangusse hilinema, siis on ahvatlev kärpida testimise aega, et püsida ajagraafikus. Kuid kui vähendada testimisele kuluvat aega, siis tuleb arvestada teiste tagajärgedega, sest kõik peab olema tasakaalus. Testimise aeg ja kulutused, et suurendada kvaliteeti, peavad olema tasakaalus riskidega ehk kuludega, mis kaasnevad tähtaegadest üle minnes või toodangusse mineva vigase süsteemiga.

On olemas ütlus: “Testida tuleb nii, et millal iganes testimine tuleb lõpetada, siis sa oled teinud oma parimad võimalikud testid”. Parim võimalik test sõltub riskist, mis on võetud, kui tarnitavasse tootesse on jäetud sisse vead. Tuleb leppida tõsiasjaga, et kõike ei ole võimalik testida. Testimine on kui valimi kontroll [2] ning sellega seoses on alati võetud risk, et vead võivad olla seal, kust me parasjagu ei testi.

Risk on defineeritud kui: “Võimalus, et soovimatu negatiivne juhtumi tagajärg realiseerub”. Teine definitsioon: “Probleem, mis ei ole veel materialiseerunud ja võimalik, et seda ka ei juhtu.” Seetõttu riskiga on seotud kaks aspekti - mõju ja tõenäosus [2].

$$Riskiaste = mõju * tõenäosus$$

Siit on selgelt näha, et kui riski tõenäosus on null või kui selle mõju ei mõjuta meid, siis pole ka riski. Kuid kõik riskid, millel on mõju ning tõenäosus suurem kui null, on riskid, millega tuleb tegeleda.

Riske on mitut tüüpi: äri-, protsessi-, projekti- ning tootepõhised riskid. Äripõhised riskid on need, mis ähvardavad tervet ettevõtet või organisatsiooni. Protsessipõhised riskid on seotud protsessidega ja sellega, missugusel viisil tööd tehakse. Riskid, mis puudutavad projekti ning toodet on testijate põhiline mure [2]. Antud peatüki raames keskendutaksegi enim projekti ning tootega seotud riskidele.

3.3.1 Projekti- ja tooteriskid

Projektirisk on seotud projektiga ning projekti eduka lõpetamisega. Projektiriskid on need, mis ohustavad projekti progressi ning edukat plaani järgi lõpetamist. Antud riskid on peamiselt projektijuhi ja kõrgema juhtkonna mure. Projektiga seotud riskideks võivad olla:

- inimesed, kes tegelevad projektiga;
- aeg;
- raha;
- arendus- ja testkeskkond, kaasa arvatud töövahendid/programmid;
- välisliidesed:
- kliendi/tarnija suhted;
- ebaadekvaatsed ajahinnangud [2].

Näiteid projektiga seotud riskidest reaalelust: vajalikku analüütikut ei ole kohal, kui nõuete analüüs peaks algama; testijad ei ole piisavalt pädevad, seega testimisele kulub rohkem aega kui eeldatud; mõne mooduli või funktsionaalsuse keerukust on alahinnatud.

Tooteriskid on riskid, mis on seotud lõpliku tarnitava tootega. Tooteriske nimetatakse ka kvaliteediriskideks, sest need peamiselt mõjutavad toote kvaliteeti. Need riskid on seotud

vigadega, mis tarnitavas tootes esinevad. Tooteriskid on otseselt seotud testija tööga, sest testimine aitab antud riske leevendada. Tootega seotud riskid pärinevad:

- funktsionaalsetest ja mittefunktsionaalsetest nõuetest;
- puuduvatest nõuetest;
- valesti mõistetud nõuetest;
- mitmeti mõistetavatest nõuetest;
- väga keerulisest äriloogikast;
- madalast testitavusest [2].

Näiteid reaalelust: soodustuse arvutamise algoritm on valesti realiseeritud, mistõttu klient võib kaotada palju raha; on võimalik välja printida konfidentsiaalsete andmetega aruannet, põhjuseks viga aruandluse rakenduses; väike, kuid oluline funktsionaalsus on nõuete analüüsi protsessis
tähelepanuta jäänud ning on seetõttu realiseerimata.

Projekti- ja tooteriskid võivad üksteist mõjutada ning olla üksteise põhjustajaks - projektirisk võib olla tooteriski põhjuseks ning tooterisk võib olla projektiriski põhjuseks. Näiteks, kui projektirisk avaldub kujul, et ajast on puudu ning kärbitakse komponentide testimise aega, siis see võib põhjustada tooteriski, sest toote testimata komponentidesse või ebapiisavalt testitud komponentidesse võivad selletõttu sisse jääda vead [2].

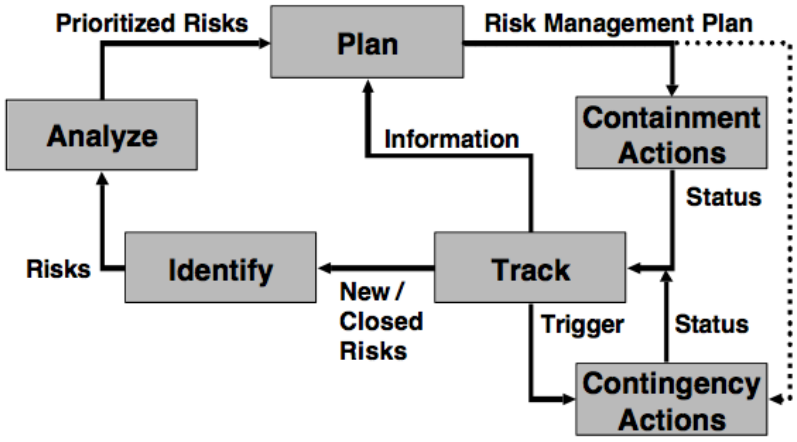
3.3.2 Riskide haldus

Hea kvaliteediga tarkvara loomisega aja- ning kulueelarves on seotud palju riske. Tarkvara riskihalduse vajalikkust illustreerib hästi Gilb'i ütlus "Kui sa aktiivselt ei ründa riske, siis nad ründavad sind" [25]. Riskide halduse tulemusi saab kasutada, et prioritiseerida testimist. Piirkonnad, moodulid või teatud funktsionaalsus, mis on kõrgema riskiga, tuleks testida esimesena ning selle testimisele tuleks eraldada rohkem aega ning muid ressursse. Lõpuks rakenduse riskihaldus võimaldab hinnata tehtud testimist. Näiteks, kui testimise jaotus on seotud riskidega, siis peaks olema võimalik iga hetk teada maandatuid ning

allesolevaid riske.

Testimise abil saab, nagu eelnevalt mainitud, leevendada tootega seotud riske. Tõenäosust tarnida kliendile toodet, milles on veel vigu, saab vähendada testimisega, mis leiab vead ning sellele järgneva vigadeparandamisega.

Riskide analüüs koosneb mitmest etapist (Joonis 14): riskide määratlemine, riskide analüüs, riskide leevendamine, riskide järelmeetmed.



Joonis 14. Riskide halduse protsess [25].

Riskide määratlemisega leitakse, mis võib juhtuda, mis ohustaks tarkvara arendusprotsessi, projekti ennast või näiteks kliendi rahulolu tarnitava tootega. Riskide analüüsi käigus määratletud riskid hinnatakse ning järjestatakse üksteise suhtes. See tähendab, et riskidele määratakse prioriteedid nende esinemise tõenäosuse (kui tõenäoliselt risk võib juhtuda) ning mõju (kui oluliselt antud risk võib mõjutada) järgi. Selle põhjal saab näha, milline risk on suurim. Kui riskid on määratletud ning hinnatud, siis tuleks need üles kirjutada ka riskide halduse plaani. Riskide halduse plaan peaks sisaldama ka riskide leevendamise meetmeid. Riskide leevendamisel planeeritakse tegevusi, mis võiksid riskide esinemise tõenäosust ning mõju vähendada. Näiteks tootega seotud riskide puhul rakendatakse testimise tegevusi, mis vähendavad tõenäosust, et tootes on vigu, kui see tarnitakse - mida rohkem vigu saab tootest testimise tulemusel eemaldatud, seda rohkem riski tõenäosus langeb. Riskiplaanis on

määratletud ka erakorralised tegevused, mida rakendatakse, kui risk peaks realiseeruma. Riskide määratlemine, analüüs ning leevendamine ei peaks olema ühekordsed tegevused. On oluline jälgida riske ning riskide leevendamise tegevusi. Jälgimise tulemusel on teada, millal ja kas teadaolevad riskid saavad maandatud ning kas uusi riske on tekkinud [2], [25].

Riski mõju saab mõõta näiteks tegeliku kuluga, mis tekib, kui rike esineb. Antud kulu saab mõõta rahas - olgugi, et on tihti keeruline hinnata, mis riski realiseerudes selle rahaline väärtus võiks olla [2]. Reaalses elus kõiki teadaolevaid riske ei suudetagi maandada enne toote tarnimist. On üsna tavaline olukord, et näiteks toode tarnitakse koos teadaolevate riskide arvesse võtmisega. Mõnikord on isegi odavam riskide maandamiseks mitte midagi teha - kui kasu ootamisest, kuidas asjad kulgevad, on suurem, kui millegi tegemise kulu. Näiteks, ei ole mõistlik osta millelegi 200 eurost kindlustust, kui ese ise on väärt kõigest 50 eurot. Kui nimetatud 50 eurot varastatakse, siis saab osta uue ning ikka on kulunud vähem raha kui 200 eurot. Seega, juhtivtestija ja projektijuht peaksid enne tarnet teadaolevatele riskidele peale vaatama ja hindama, kas riskid on piisavalt väikesed ega takista tarnet või tuleb tähtaega, ressursse või töid ümber planeerida.

3.4 Analüüs ja ennustused

Eelnevates peatükkides kirjeldati, kuidas ja mis andmeid arendustsükli jooksul monitoorima ning koguma peaks. Kogutud andmete analüüsimisel saab arendusprotsessi oluliselt parandada või kasutada andmeid tulevasteks hinnanguteks või ennustusteks. Antud peatükis kirjeldatakse arendustsükli kvaliteedi parandamist vigade analüüsi ja ennetamise ning erinevate hinnangute ja ennustamiste mudelite näol.

Vigade analüüs rakendub kõikidele vigadele ning on mõeldud parandamaks praeguseid puudusi ning eemaldamaks vigu tulevikus. Vigade analüüs on peamine viis, kuidas vähendada vigu.

On kasulik analüüsida vigu süsteemis, mida parasjagu arendatakse, kuid pikaajalisemate trendide analüüsimist tuleks samuti teha, et saada teada tarkvara arendamise protsessi

nõrgematest kohtadest. Kui vigade andmete ajalugu on kogutud, siis see võib anda viiteid, missugused muutused tarkvara arenduse protsessile võivad olla edukad [26].

Tarkvaraprojekti arendustsükli puhul saab ennustada erinevaid asju. Projekti alguses on näiteks võimalik kindlaid meetodeid kasutades (FP - *Function points*, LOC - *Lines of Code*) ennustada kui keeruliseks ja ajamahukaks loodav projekt võib osutuda. Kuid projekti jooksul kogutud andmete põhjal on võimalik ennustada või püüda hinnata, millal käimasolev projekt valmis saab, kui palju vigu see võib sisaldada, kui palju vigu olemasolevas projektis veel olla võib jne. Ennustamise tehnika valitakse selle järgi, mida parasjagu ennustada tahetakse. Ennustused ei lõpe aga lihtsalt hinnangu või ennustusega. Kui testimine on alanud, tuleb tihedalt jälgida, kuidas tegelikkus vastab tehtud hinnangutele [2]. Kui kõrvalekalded tegelikkuse ja hinnangute vahel lähevad liiga suureks, siis tuleb uue informatsiooni põhjal, mida monitoorides kogutakse, teha uued hinnangud/ennustused. See on tsükliline protsess. Ainult projekti lõppedes, kui kõik testimise tegevused on lõpetatud, siis saab lõpetada ka monitoorimise ja hinnangute ning ennustuste tegemised [2].

3.4.1 Vigade analüüs

Tarkvara loomise käigus avastatakse palju vigu ning need esinevad terve arendustsükli jooksul. Vigade ennetamine muutub oluliseks osaks siis, kui tahetakse parandada tarkvara protsessi kvaliteeti. Vigade ennetamise eesmärk on tuvastada peamised ja levinumad vigade põhjused ning muuta protsesse, et vältida samasuguste vigade edasist kordumist. Tihti vigade ennetamine jääb aga tahaplaanile ja sellega ei tegelda. Kuid kui projektis juba vigade andmeid kogutakse, siis on nende andmete pealt võimalik lihtsamini teha ka vigade analüüsi ja rakendada meetmeid vigade ennetamiseks. Mõõtmiste tegemise kulu on suhteliselt madal, kuid kasu mis saadakse üldisest kulu kokkuhoiust on palju suurem võrreldes kuluga, mis kulub vea parandamiseks hilisemas staadiumis. Kui teadmine vigu põhjustavatest meetoditest ja protsessidest on olemas, siis on
võimalik rakendada vigade ennetamist [26].

Vigade ennetamise protsess algab sellega, et kõigepealt tuvastatakse vead projektis. Seda tehakse eelnevalt planeeritud testimistegevuste abil, mis on spetsiaalselt mõeldud selleks, et leida vigu. Näiteks vead, mis leiti mooduli või süsteemi testimisel või isegi koodiülevaatusel. Kui vead on tuvastatud, siis need klassifitseeritakse. Vigade klassifitseerimine väikeste või keskmiste projektide puhul võib toimuda üldisemalt, kuid kriitiliste projektide puhul peab klassifitseerimine toimuma põhjalikumalt ning sügavamalt. Üldiselt toimub vigade klassifitseerimine nende leidmise koha järgi, näiteks vead, mis leiti nõuetest, disainist, funktsionaalsel testimisel jne. Joonisel 15 on näide vigade klassifitseerimisest ühe juhtumiuuringu näitel, kus osales 5 projekti.

Life cycle phases	Activity	Defect Type	No of Defects
Requirements	Review	REQ	74
Design	Review	DSN	58
Code	Testing (Function/unit)	LOG	420
GUI	Review	GUI	55
Documentation	Review	TYP	30

Joonis 15. Vigade klassifitseerimine 5 sarnase projekti põhjal [26].

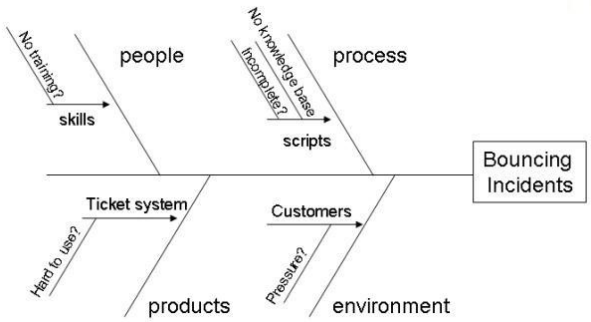
Kui vead on klassifitseeritud, siis neid analüüsitakse, et leida mustreid. Jooniselt 16 on näha juhtumiuuringu 5 projekti vigade tüübid ja arvud eraldi välja toodult. On näha, et 70-80% vigadest on klassifitseeritud kui koodi vead.

Proj No	REQ	Design	LOG	GUI	Doc	Total
Proj 1	20	17	120	9	6	172
Proj 2	15	11	75	10	8	119
Proj 3	12	14	58	13	7	104
Proj 4	12	8	60	15	2	97
Proj 5	15	8	107	8	7	145
Total	74	58	420	55	30	637

Joonis 16. Täheldatud vigade muster üle 5 projekti [26].

Seejärel rakendatakse vigade algpõhjuste analüüsi (*root causes analysis)*, mille eesmärk on lisaks vigade algpõhjuse kindlaks määramisele ka selliste tegevuste rakendamine, et vigade allikad saaksid eemaldatud, eesmärgiga vähendada sarnaste vigade esinemiskordi järgnevates projektides või projekti iteratsioonides. Vigade algpõhjuste analüüsi juures peaksid olema ka spetsialistid, kes saavad aru ka vigade põhjustest ning oskavad neid eemaldada. Vahend, mis

antud analüüsi puhul abiks võiks olla, on põhjuse ja tagajärje diagramm, mida tuntakse kui kalaluu (*fishbone)* diagrammi nime all (Joonis 17).



Joonis 17. Näide *Fishbone* diagrammist [27].

Vigade analüüsi korratakse seejärel veelkord, et uuesti leida uued enim esinenud vead ja need eemaldada, sel viisil järk-järgult kvaliteeti parandades. Sarnast vigade põhjuste analüüsi saab ja on kindlasti kasulik rakendada ka nende vigade puhul, mida arendustsükli jooksul avastada ei suudetud ning mis leiti alles siis, kui rakenduse oli juba toodangus. Seda eesmärgiga, et sarnased vead enam rohkem ei avalduks ning vähendada kliendi poolt leitud vigu, vähendada kulusid ning parandada toote kvaliteeti ja mainet [28].

3.4.2. Erinevad ennustamise mudelid

Tarkvara arendusprotsessi jooksul kogutud andmete põhjal saab teha erinevaid olulisi ennustusi.

Enamasti on sellised ennustused üsna keerulised ja ajamahukad, mistõttu kasutavad selliseid ennustamise mudeleid siiski vaid väga suured ettevõtted nagu näiteks Microsoft, HP, NASA jne [19]. Väikeste tarkvara rakenduste loomisel pole see ajaliselt otstarbekas. Antud peatükis on toodud lühike ülevaade, mida veel on võimalik arendusprotsessi jooksul kogutud andmete põhjal hinnata ja ennustada.

Näiteks on võimalik ennustada süsteemis allesolevate vigade arvu, et selle järgi otsustada, millal rakendus on toodangukõlbulik. Arendatavas süsteemis allesolevate vigade arv on üks peamisi tegureid, mis võimaldab otsustada, kas tarkvara on tarnimiseks valmis või mitte.

Võrreldes ennustatud vigade arvu testimise jooksul leitud vigade arvuga, saab projektijuht otsustada, kas toode on tarnimiseks valmis. Läbi on viidud ka uuringuid ning loodud mudelid, mille abil on võimalik ennustada vigade arvu süsteemis. Antud mudelid kasutavad ennustuste tegemisel kogutud toote meetrikate andmeid ning eelnevate testimiste jooksul kogutud vigade andmeid, mille abil hinnatakse tarkvara valmidust süsteemi alles jäänud vigade arvu näol [19].

Kui süsteemi alles jäänud vigade arvu ennustamine on kasulik, siis ennustuste vähesus vigade parandamisele kuluva aja kohta teeb ikkagi testimise ajagraafikute ennustamise ja testimise ressursside planeerimise raskeks. Sellepärast, et vigade parandamise aeg ei sõltu ainult vigade arvust, vaid ka nende keerukusest ning parandaja oskustest ja töökoormusest. Näiteks mitu lihtsalt viga võivad vajada parandamiseks palju vähem aega kui üks keeruline viga. Samuti vigu, mis on leitud hilisemas etapis, on keerulisem parandada, kui varasemas etapis. Seega veel üks ennustamise võimalus on ennustada vigade parandamiseks kuluvat aega. Võime ennustada vigade parandamise aega on kasulik testplaanide ja ajaplaanide loomiseks ning vältimaks projektide ajagraafikute ületamist. Vigade parandamise aja ennustamine põhineb vigade andmete analüüsil ning tegeleb juba kogutud andmetega, mitte andmete kogumise endaga. Eesmärk ei ole uurida tarkvara meetrikaid, vaid rakendada neid, mis on testimise protsessi jaoks tähendusrikkad. Vigade parandamise aja leidmiseks on läbi viidud samuti juhtumiuuringuid. Ühel juhul kasutati antud ennustust ühe suure meditsiini-infosüsteemi puhul, kus vigade raportid koguti meditsiini-infosüsteemi kolme tarne pealt [29]. Uurimuse huvi oli ennustada aega vea raporteerimise kuupäevast kuni selleni, millal viga viimati parandati. Uuringus kasutatud vigade andmed sisaldasid: vea numbrit, versiooni numbrit, mis faasis viga leiti (nt kas arendusest või toodangust), komponendi numbrit, raporteerimise kuupäeva, teostamisele määramise kuupäeva, parandamise kuupäev, vea sulgemise kuupäeva. Tulemuste saamiseks kasutati andmeanalüüsi, sealhulgas ka statistilisi meetodeid ning masinõpet. Andmekaeve meetoditest kasutati: Naive-Bayes klassifikaatorit (*Naive-Bayes Classifier)*, otsustuspuud (*Decision Tree Learning*) ning neurovõrkude lähenemist (*Neural Network approach*). Antud juhtumiuuringu tulemuseks oli, et keskmine vigade parandamise aeg on antud süsteemis 90 päeva ning kõige suurema täpsuse andis otsustuspuu lähenemine - 93.5% [29].

Arvutamaks välja, millal on kõige mõistlikum testimine lõpetada, on arvutatud töökindluse-kulu mudeleid (*Reliability models*). Töökindluse mudelid määravad kindlaks, millal töökindluse kasv õigustab eeldatavat kogu kulu/maksumust. Antud mudelite kasutamise käigus kaalutakse erinevaid tarkvara kuluallikaid, nagu testimise kulud, leitud vigade eemaldamise kulu ning riskikulu, mis on tingitud tarkvara rikkest [18]. On olemas erinevaid töökindluse kulu mudeleid ning ei ole olemas ühte kindlat mudelit, mis sobiks kõikides olukordades. Üks mudel võib sobida teatud tarkvara puhul, kuid ei pruugi kõiksugu probleemide korral [30].

4. Testimise dokumentatsioon

Antud peatükk annab ülevaate, missugust testimisalast dokumentatsiooni eelnevates peatükkides kirjeldatud etappide jooksul tekib - ehk dokumentatsioon, mida luuakse testimise protsessi jälgimise käigus ning testimise lõpetamisel.

Dokumendid, mida luuakse testimisprotsessi alguses või enne seda, on: nõuete dokument, nõuete spetsifikatsioonid, testimise strateegia, testplaan (võib koosneda omakorda peatestplaanist, iteratsiooni testplaanist jne). Nimetatud dokumentidel aga antud töö raames pikemalt ei peatuta, sest need on testimise protsessi sisendiks. Antud töö raames kirjeldatakse ainult testimise protsessi käigus ning lõpetamisel loodavat dokumentatsiooni. Dokumentatsiooni koostamine teeb testimise protsessi nii arendusmeeskonna kui kliendi jaoks läbipaistvamaks.

4.1 Projekti staatuse raport

Projekti staatuse raportit koostab testimismeeskond ning seda saab koostada nii päeva, nädala, kuu, kui kogu projekti kohta [31]. Näiteks nädalane staatuse raport on oluline, et jälgida olulisi projekti küsimusi ja probleeme, projekti saavutusi, pooleliolevaid töid. Iga nimetatud ajavahemiku kohta koostatud staatuse raport peaks olema suunatud testimise eesmärkidele ja ulatusele, mis antud ajavahemikku planeeritud. On kasulik antud vahemiku testimise eesmärgid ning ulatus panna kirja enne antud etapi algust ning seejärel kirjeldada saavutused ja täidetud eesmärgid käesoleva raporti perioodi jooksul. Antud raporti põhjal võib valmistada ette tulevased tegevused ning moodustada näiteks järgmise nädala tööde nimekirja. Iga teadaolev risk, mis võib otseselt testimist mõjutada, tuleb antud raportis detailiseerida, eriti need mille tõttu edasine testimine on häiritud või takistatud [31].
Näiteks nädalane staatuse raport koosneb järgmistest osadest:

- Kokkuvõte - kirjeldatakse lühikese lõiguna testimise progressi eelmise nädala jooksul ning nimetatakse suuremad murekohad.

- Olulised vead - kirjeldatakse nimekiri vigadest, mis mõjutavad või segavad testimist kõige rohkem. Need on vead, millele tuleks kohe tähelepanu pöörata, et testimine ei takerduks.
- Probleemid - kirjeldatakse probleeme, mis takistavad testimismeeskonda õigeaegsel tarnimisel, probleemide kirjeldusi ning pakutud lahendusi. Samuti tuuakse välja probleemid, millest juhtkond peaks teadlik olema.
- Järgmise nädala prioriteedid - kirjeldatakse lahtised tulemused, mis on näiteks eelmise nädala tulemused, mis peaksid tehtud saama niipea kui võimalik ning uued tööülesanded, mis antud nädalal teha ei jõua, kuid peaks tegema järgmisel nädala.
- Progress - kirjeldatakse, kas testimisega ollakse ajagraafiku suhtes järje peal. Näiteks, kui jälgitakse testjuhtumite läbi testimist, siis võib võrrelda läbi testitud juhtude arvu kogu plaanitud juhtude arvuga [32].

4.2 Testimise koondaruanne

Testimise koondaruanne on dokument, mis luuakse testimise lõpetades juhtivtestija või kogu testimismeeskonna poolt ühiselt. Antud dokumendi eesmärk on kokku võtta tulemused ja teha nende põhjal hinnanguid, analüüse. Testimise koondaruannet peaks koostama iga testimise tsükli lõpus ning kogu projekti testimise lõpus. IEEE829 standardi järgi testimise koondaruanne koosneb järgmistest sisulistest osadest:

- Erinevused - raporteeritakse kõrvalekalded testimise strateegiast, testplaanist ja muudest plaanidest.
- Hinnang - antakse hinnang testimise kvaliteedi kohta, samuti ülevaade teadaolevatest riskidest või vigadest, mida rakendusse sisse jäid. Hinnang põhineb testimistulemuste meetrikatel ning testimise lõpetamise kriteeriumidel.
- Tegevuste kokkuvõte - kokkuvõte peamistest testimistegevustest ning sündmustest. Sisaldab ka andmeid ressursside kasutamise kohta, näiteks kui palju aega oli millegi jaoks plaanitud kulutada ning palju tegelikult kulus.
- Tulemuste kokkuvõte - antakse ülevaade testimise tegevuste tulemustest ning vigadest, mis said parandatud ning mis juurde tekkisid.

- Kinnitamine - dokumendi kinnitajate nimekiri [12].

4.3 Tehtud tööde nimekiri

Antud iteratsioonis koostatud tööde nimekiri võib olla testimise koondaruande osa, kuid võib olla ka eraldiseisev dokument. Antud dokumendis on kirjas kõik antud iteratsioonis tehtud tööülesanded. Kirjas on nii eelmise verisooni kui ka uue versiooni numbrid, ning antud vahemikus tehtud uute arenduste tööülesanded, parandatud vigade tööülesanded ning tehtud hooldustööde tööülesanded. Tööülesanded on soovitav järjestada vastavalt nende tüübile, kas ülesanne on seotud arenduse, vea või hooldusega. Antud dokument on oluline nii projekti meeskonnale endale, kuid eelkõige kliendile. Selle dokumendi järgi saab klient ülevaate, mis tööd täpselt antud tsüklis tehtud said.

4.4 Regressioontestimise tabel

Regressioontestimise tabel, mida luuakse regressioontestimise käigus, on oluline dokument saamaks ülevaadet teostatud testimisest. Mõnikord pannakse antud dokument ka tarnega kaasa, näitamaks, et kõik olulisemad kasutuslood on testide üle käidud. Regressioontestimise tabelist
saab täpsemalt lugeda töö esimese peatükis.

4.5 Testide jälgitavuse maatriks

Kui tarkvara nõuete ja nende testide jaoks loodi testimisprotsessi käigus testide jälgitavuse maatriks, siis ka see on täiesti arvestatav testimise dokumentatsioon. Antud tabelit võib samuti tarnega kaasa panna, kui klient soovib täpsemalt näha nõuete kaetavust testidega. Antud maatriksist ja selle koostamisest saab täpsemalt lugeda töö esimeses peatükis.

5. Uurimuse kirjeldus

Antud töö uurimus kasutab kvalitatiivset meetodit. Kvalitatiivne uurimustöö kasutab intervjuu vormi, lõpetamata vastustega küsimusi indiviidi või grupi hoiakute, arvamuste, käitumiste uurimiseks ja mõistmiseks. Kvalitatiivne uuring otsib vastusi küsimustele "Miks", "Kuidas", "Millisel viisil" [33]. Antud töö puhul otsitakse vastuseid küsimustele : "Millisel viisil tarkvara testimise protsessi läbipaistvus erineb erinevates projektides ning kas ja kuidas see mõjutab
loodava tarkvara kvaliteeti?".
Kvalitatiivse uuringu tugevusteks on sügavus ja detailid, mida ei ole võimalik saada standardiseeritud küsimustikega. Nõrkusteks aga on see, et uuring hõlmab tavaliselt väiksemat arvu inimesi, andmeid on raskem koguda ning keerulisem teha süstemaatilisi võrdlusi, mistõttu on tulemus hõlpsamini üldistav [33].

5.1 Küsitluse ettevalmistus

Uurimuse läbiviimiseks viidi läbi küsitlus, mille koostamisel kasutati intervjuu vormi ning elektroonilist küsitluse koostamise vahendit eFormular [34]. Kuna kõikide vastanutega ei õnnestunud sobivaid kokkusaamise aegu leida, et esitada küsimusi intervjuu vormis, siis koostati ka elektroonne küsitlus. Elektroonsel kujul küsitlus on kättesaadav aadressil http://www.eformular.com/kaisapaavo/kysimustik.html, kuid küsimused on näha ka antud töös lisana (Lisa1). Uurimuses osalesid erinevate tarkvaraprojektide juhtiv testijad, kes vastasid 16-nele küsimusele. Osalejaid otsiti erinevatest firmadest ning firmade erinevatest projektidest internetis sotsiaalsuhtlusvõrgustike ning tuttavate vahendusel. Eesmärk oli küsitleda võimalikult erinevate projektide testijaid, et näha, kas projekti tüüp ja olemus mõjutab testimise protsessi korraldamist. Küsimustik koosneb kolmest peamisest osast:

1. osa - projekti tüüp;
2. osa - testimise läbipaistvus;
3. osa - projekti edu.

Küsimustiku esimeses osas uuritakse, mis valdkonna projektiga on tegemist, kas rakendus on avalik või kinnine, kui palju kasutajaid on antud rakendusel ning kui keeruline on antud

rakenduse äriloogika. Samuti uuritakse, kui palju testijaid, arendajaid, analüütikuid antud rakenduse juures töötab, kui kaua kestab keskmiselt üks arendustsükkel ning kaua üldse on kestnud ja kestab antud projekt ning mis arendusmetoodikat antud projektis kasutatakse. Antud küsimuste eesmärgiks on välja selgitada, kas järgnevate küsimuste vastused sõltuvad kuidagi projekti tüübist ja kui sõltuvad, siis leida seaduspärasusi.

Küsimustiku teises osas uuritakse projekti testimise protsessi ja selle läbipaistvust - kuidas, mille abil hinnatakse rakenduse hetkeseisu, mille abil hinnatakse süsteemi kaetavust testidega, millal tehakse regressioonitestimist ja kuidas valitakse regressioontestimise jaoks testjuhtumid ning mille alusel otsustatakse testimine lõpetada. Veel uuritakse, missugust testimise alast dokumentatsiooni arendustsükli jooksul koostatakse ning missugust neist tarnega kaasa pannakse. Antud küsimuste eesmärk on näha erinevaid testimise protsessi ja selle läbipaistvusega seotud tegevusi erinevates projektides.

Küsimustiku kolmandas osas uuritakse projekti edu kohta - kas klient on üldiselt tarnetega rahul ning kas projekt on finantsiliselt edukas, graafikus ja eelarves. Antud küsimuste abil otsitakse hinnangut testimise protsessi läbipaistvuse mõjust arendatava tarkvara kvaliteedile.

5.2 Tulemused

Uurimuses osales 10 testijat. Uurimuses osalejad olid kõik erinevatest projektidest, kuid mõned samas ettevõttest. Üldiselt jagunesid ettevõtted, kust vastajaid otsiti, kolme kategooriasse: ettevõtted, kus teostatakse palju erinevaid projekte, kus iga projekti jaoks on erinevad meeskonnad ning kus analüüs, testimine ja arendus tehakse kõik ühe firmasiseselt; ettevõtted, mis arendavad ühte kindlat toodet (analüüs, testimine, arendus tehakse samuti firmasiseselt); ettevõtted, mis pakuvad ainult testimisteenust. Kuid kuna küsitlus oli anonüümne, ei küsitud otseselt ettevõtte ega projekti kohta, siis vastused olenemata ettevõtte tüüpidest, on segamini. Kõikide vastajate originaalkujul vastused salvestati Exceli faili (Lisa 2).

Andmeanalüüsiga antud töö raames ei tegeleta, kuna vastajaid ei olnud nii palju ja tugevaid adekvaatseid seoseid seega ei teki. Pigem analüüsitakse vastuseid teksti kujul.

5.2.1 Ülevaatlikud andmed vastanute projektide kohta

Kõigepealt on toodud välja ülevaatlikud andmed vastanute projektide tüüpide kohta. Näiteks rakenduse kasutajaskonna arvu kohta selgus, et uuringus osalenud projektidel on pigem suurem kasutajaskond (Diagramm 1). Nimelt 40% vastanutest vastas, et rakenduse kogu kasutajaskond on üle 10 000 inimese ning 20%-ndil on kasutajaid 1000 – 10 000 inimest. Ülejäänud 40%-ndil vastanutest jääb rakenduse kogu kasutajaskond alla 1000 inimese.

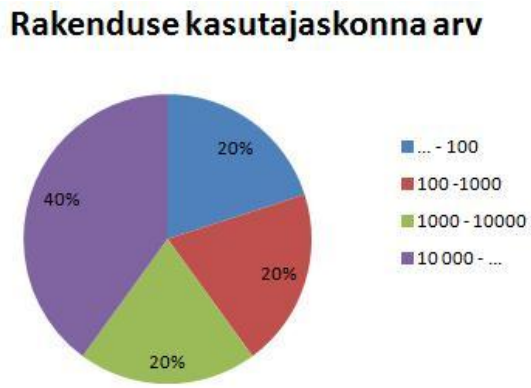


Diagramm 1. Rakenduse kasutajaskonna arv

Uuringule vastanute meeskondade kohta selgus, et enamus projektides testijate osakaal meeskonnas on pigem hea (Diagramm 2). Kuus vastajat vastasid, et nende projektides testijaid on kogu meeskonnast ligikaudu 20-30%, mis on hea suhe. Kolm vastajat vastasid, et testijaid on 10-20% ning üks vastas, et alla 10%, mis tähendab, et testijate osakaal meeskonnas on päris väike, millest võib järeldada, et testijal on pigem palju tööd.

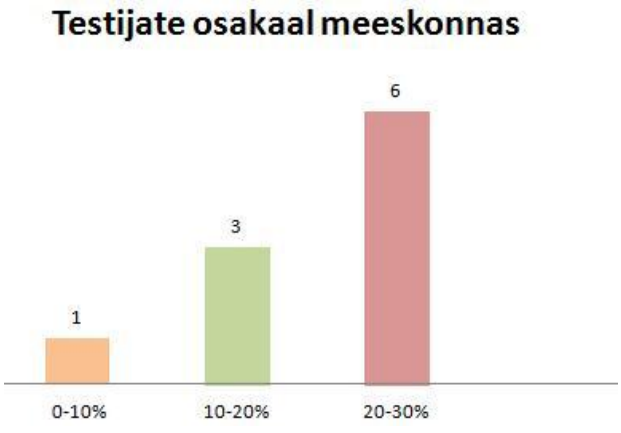


Diagramm 2. Testijate osakaal meeskonnas

Rakenduse avalikkuse ja suletuse kohta selgus, et enamusel, 40%-ndil uuringus osalejatel on suletud kasutajaskonnale mõeldud rakendused (Diagramm 3). Ülejäänud vastused jagunesid võrdselt, 30%-ndil on avalikud rakendustest ning teisel 30%-ndil vastanutest on osalt tegu avalike osalt suletud rakendustega.

Diagramm 3. Rakenduste avalikkus

Enamus uuringule vastanud hindasid oma süsteemi äriloogika keerukust pigem kõrgeks (Diagramm 4). Viie palli süsteemis 7 vastanuist hindasid enda arendatava süsteemi äriloogika nelja või viie palli väärilisekс. Kaks vastasid, et nende rakenduse äriloogika on kolm palli ning üks vastas, et kaks palli. Ühe palli väärilisekс ei hinnanud oma rakendust keegi.

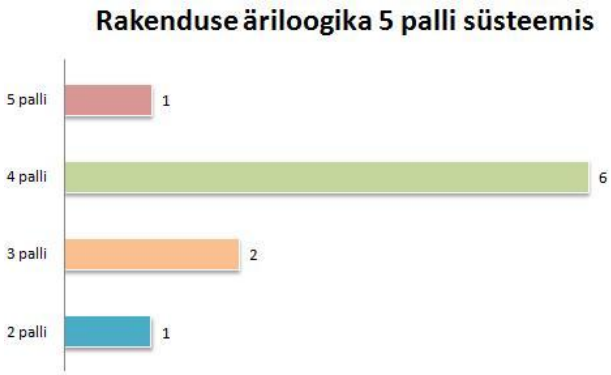


Diagramm 4. Vastanud projektide äriloogika

Uuringule vastanute projektidest neli on kestnud kuni kaks aastat, neli on kestnud 2-5 aastat ning ainult ühe arendamine on kestnud juba üle 5 aasta. Vastanuist üks testija ei osanud öelda, kui kaua nende projekti on arendatud (Diagramm 5).

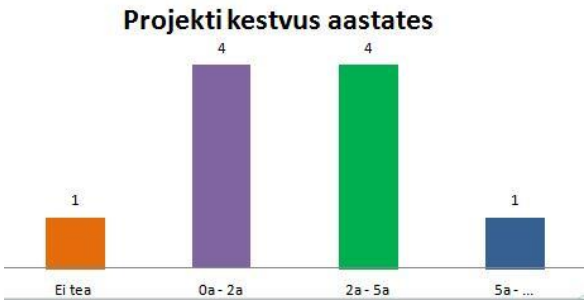


Diagramm 5. Vastanud testijate projektide kestvus

5.2.2 Testimisprotsessi kirjeldavad vastused

Arendatava rakenduse hetkeseisu hindamise kohta tõid vastanud testijad välja enim teadaolevate vigade hulga, seda koguni 8 vastanuist. Vigade puhul jälgitakse nende prioriteeti ning staatust. Samuti jälgitakse ka teiste antud tsükli tööülesannete seisu ning hinnatakse valminud funktsionaalsuse mahtu ning vastavust spetsifikatsioonile. Rakenduse hetkeseisu hindamisel lähtutakse veel ka hetkeseisu vastavusest testimisplaani peaplaanist – kaks vastajat, regressioontestimise või automaattestidega kaetud või katmata protsesside hulgast – kaks vastajat. Testitava rakenduse hetkeseisu hindamise aluseks mainiti ka kõhu- või sisetunnet – kaks vastajat.

Arendatava süsteemi kaetust testidega hinnatakse enamasti prioriteetide järgi sorteeritud protsesside nimekirja ning regressioontestimise tulemuste abil, mis samuti sisaldab endas prioriteetide järgi sorteeritud protsesside nimekirja – selle tõid välja neli vastanuist. Nimetati ka lihtsalt testijate märkmeid testidega kaetud funktsionaalsuse kohta ning üleüldist testijate kursisolekut testitud komponentide ja funktsioonide seisukorra kohta – kaks vastajat. Üks vastaja vastas, et kuna automaatteste ei kasutata, siis süsteemi kaetavust testidega üldse ei hinnatagi. Tundus, et antud küsimuse juures tekitasid pisut segadust definitsioonid „katvus“ ning „kaetavus“. Antud juhul oli küsimuses mõeldud testide kaetust, ehk kuidas, mille abil hinnatakse süsteemi testitud ja testimata osasid, mitte koodi kaetavust ühiktestidega. Kahe vastaja vastustest ei lugenud midagi antud küsimusele vastuseväärset välja.

Regressioontestimist rakendatakse 10-st vastanud projektist 8-s, ühes projektis tegeleb regressioontestimisega klient. Kõik vastanutest ei maininud, millal arendustsükli jooksul regressioontestimisega tegeletakse, kuid kolm vastajat siiski mainisid, et sellega tegeletakse jooksvalt ning kolm, et sellega tegeletakse pigem tsükli lõpuosas, näiteks stabiliseerimise faasis. Regressioontestimise käigus, mis toimub nii manuaalselt kui automaatselt, läbitavad protsessid valitakse olulisuse ehk prioriteetide järgi, seda kirjutas 5 vastanuist. Ühes projektis käiakse regressioontestimise käigus läbi kõik protsessid ning ühe projekti puhul kasutati protsesside valimisel heuristilist strateegiat (*Heuristic Strategy*). Protsesside valimisel on abiks ka varasem kogemus ning mõnikord ka arendajate hinnang. Kaks vastajatest kirjutasid, et suurt kindlust testkomplekti valimisel alati siiski ei ole.

Testimise lõpetamise aluseks on suures osas see, kui testijatel on plaanitud testid tehtud, tsüklisse planeeritud tööülesanded testitud ning (kriitilisi) vigu pole enam üleval, see kajastus 7 vastaja vastustest. Kolm vastanuist nimetas testimise lõpetamise otsuse aluseks ka testimisele määratud aja otsa saamist. Ühe vastanud projekti puhul otsustab testimise lõpetamise üle projektijuht.
Mainitud oli ka sisetunne.

Testimise lõpetamisega seoses koostavad tsükli lõpus suurem osa projektidest, koguni 7 vastanuist, kliendile tehtud tööde nimekirja (*release notes),* mis sisaldab ühtlasi ka teadaolevate vigade nimekirja, kui neid tarnimise hetkel on. Mõnel juhul koostatakse ka nimekiri teostatud testimistegevustest ning uutest testjuhtumistest. Samuti tekib testimise lõpetamisega seoses dokument regressioon testimise tulemustest, kui seda testimisprotsessi jooksul tehakse. Kolm vastanuist ütlesid, et testimisalast dokumentatsiooni ei koostatagi. Kaks vastanuist nimetasid, et sõnalise dokumentatsiooni asemel kasutatakse kohati mudeleid, pilte, tabeleid, heuristikuid. Veel mainiti, et tarne käigus pannakse kliendile kaasa treeningmaterjal, paigalduse juhend või litsentsileping ja kasutustingimused (*EULA/ToS*).

5.3.3 Projekti eduga seotud vastuste analüüs

Vastustest, mis olid seotud projekti eduga, selgus, et 10-st vastanuist 6-l juhul on klient tarnete kvaliteediga üldiselt rahul. Üks vastanud testijatest ei osanud öelda, kas klient on tarnetega rahul või mitte ning üks testija vastas, et nende klienti häirib vigade olemasolu toodangus, mistõttu klient ei ole tarne kvaliteediga väga rahul. Küsimus kliendi rahulolu kohta tarnega tekitas kahe vastaja seas mitmeti mõistetavust. Esitatud küsimusega oli mõeldud milliseks klient hindab tarne kvaliteeti, mitte millisel viisil klient hindab tarne kvaliteeti.
Seaduspärasusi, missugustel juhtudel klient tarnega rahul on või ei ole, selgelt välja ei joonistunud. Näiteks, kõikidel juhtudel, kui vastati, et klient on tarnega rahul, jälgiti testimisprotsessi jooksul testidega katvust ning hinnati rakenduse kvaliteedi hetkeseisu teostatud tööülesannete ja olemasolevate vigade ning olulisemate protsesside läbimise näol. Kuid sama tehti ka selles projektis, kus klient tarnetega rahul ei olnud. Veel võis märgata, et

tarnetega rahulolevate klientidega projektides teostati regressioontestimist 6-st projektist 5-s ning projektis, kus klient ei olnud tarne kvaliteediga rahul regressioonitestimist ei teostatud. Samas ei saa nii väikeste andmemahtude pealt otseseid järeldusi teha.

Uuringus osalenud 10-st projektist 7 on projektide testijate teadmistekohaselt finantsiliselt edukad. Kaks testijat ei osanud öelda enda projekti finantsilist seisu ning üks testija vastas, et tema projekt ei ole väga kasumlik. Tegelikult ei saa vastuseid projektide finantsilise edu kohta täieliku tõena võtta, kuna tihti testijatel ei ole täielikku teadmist projekti edu kohta, mõnikord pole seda igal hetkel isegi projektijuhil.

Vastanud testijatest pooled vastasid, et nende projekt on üldjoontes alati graafikus ning teised pool vastasid, et nende projekt pigem ei püsi planeeritud ajagraafikus. Tulemustest järeldades ei saa öelda projektid, mis ei püsi graafikus, ei ole finantsiliselt edukad. Näiteks, kolmel juhul, kui vastati, et projekt ei pea alati kinni ajagraafikust, vastati, et projekti on siiski kasumlik. Ainult ühel juhul, kui vastati, et projekt ei pea kinni graafikust, vastati ka, et see ei ole kasumlik projekt. Antud vastuse puhul on positiivne see, et kõik vastanud teadsid projekti ajagraafikust kinnipidamise või mitte kinnipidamise kohta ning see on testija töös oluline teadmine.

6. Üldine tulemuste analüüs

Uuringu võib nimetada õnnestunuks selles mõttes, et käesoleva töö autor sai põhjaliku ülevaate 10-s erinevas tarkvara projektis rakendatavatest testimise protsessi jälgimise ning testimise lõpetamisega seotud tegevustest. Olgugi, et tehtud uuringu põhjal ei tulnud välja päris kõiki oodatud tulemusi. Nimelt oodati selgemaid vastuste erinevusi erinevat tüüpi projektide puhul. Näiteks, et avalike ning väga suure kasutajakonnaga süsteemide puhul testimise protsessid on rangemad või rakendatakse tegevusi, mida väiksemate projektide puhul ei rakendata. Selliseid seoseid antud uuringu põhjal aga välja ei tulnud. Samuti ei saa selgunud vastuste põhjal öelda, et testimise protsessi jooksul teostatavad tegevused erineksid oluliselt näiteks lühikest aega kestnud ning mitmeid aastaid kestnud projektide vahel, suure või väikse kasutajakonnaga projektide vahel, ühe või mitme testijaga projektide vahel, keerukama ja vähem keerukamate rakendustega projektide vahel jne. Võimalik, et selliseid seaduspärasusi ei joonistunud välja, kuna uuringus osalejate arv ei olnud niivõrd suur. Samuti muutis selgete joonte kujunemist keerulisemaks vastajate väga pikad kirjeldused, kust olulisema info välja sorteerimisega tekkis kohati raskusi. Selles suhtes oleks aidanud ehk kvantitatiivne uuring, kus osalejaid on rohkem ning vastajatele antud ette vastused, milles seast valida. See polnud aga päris antud töö eesmärk.

Küll aga langevad uuringu tulemused suures osas kokku töö teoreetilise osaga ning antud töö autori enda kogemustega. Kõiki uuringu vastustes kirjeldatud testimise jälgimise tegevusi on kirjeldatud ka töö esimeses osas. Kõiki töö teoreetilises osas kirjeldatud tegevusi ja meetmeid aga vastustes ei mainitud - ent see oligi pigem oodatud tulemus. Teoreetilises osas kirjeldati ka keerukamaid ja ajamahukamaid testimisega seotud tegevusi või meetodeid (näiteks ajaloo-põhine regressioontestimise komplekti valik, ennustamise mudelis jms). Niivõrd suurte projektide testijaid, kus antud tegevusi rakendada võiks, aga küsitleda ei õnnestunud.

Üldiselt võib saadud tulemuste alusel öelda, et antud töös on kajastatud tarkvara projektis teostatava testimise jälgimise ja testimise lõpetamisega seotud tegevused. Uuritud väikestes ja keskmistes tarkvara projektides enamikke töö sisus toodud tegevustest ja meetmetest on kasutusel. Kahjuks ei saa antud tegevuste rakendamise või mitte rakendamise kohta teha

otseseid järeldusi projekti edu kohta - kliendi rahulolu tarnitava rakenduse kvaliteedi, finantsedu või graafikus püsimise kohta. Projekti edu mõjutavad suures osas ka klient ning projekti alguses tehtud mahu ja ajahinnanguid, mis ajavad nihkesse ka testimise hinnangud. Antud aspektidele aga käesolevas uuringus tähelepanu ei pööratud. Seega ei saa öelda, et on olemas kindlad testimise tegevused, mis alati tagavad projekti edu või vastupidi. Mõned projektid, kus arenduse ja testimise seisukohast kõik tehakse korrektselt võivad olla rahaliselt mitte edukad, ning vastupidi – projektid, kus arenduse ja testimise korraldamisega võib olla suuri probleeme, võivad siiski olenemata sellest olla finantsiliselt edukad.

7. Kokkuvõte

Kõike ei ole kunagi võimalik testida ning kõiki võimalikke vigu üles leida on praktiliselt võimatu. Seetõttu tarkvaraprojekti testimise protsessist selge ülevaate saamine on testimise lõpetamise otsuse seisukohast väga oluline. On oluline saada ülevaatlikkus ja teadmised, millele otsused rajada. Antud töösse on kogutud kirjeldused, kuidas saavutada läbipaistvus tarkvara testimise protsessis ja selle lõpetamisel.

Töö esimeses osas anti ülevaade autori enda kogemustel põhinevatest teadmistest ning erinevatest allikatest pärit tarkvara protsessi läbipaistvamaks muutvatest meetmetest ning testimise lõpetamisega seotud tegevustest ja asjaoludest. Töö esimeses peatükis kirjeldati tegevusi, mille abil testimise protsessi jälgida, et saavutada selle läbipaistvus ja ülevaatlikkus. Töö teises peatükis kirjeldati testimise lõpetamisega seotud tegevusi ning meetodeid, mille abil saab toote valmisolekut ennustada. Töö kolmandas peatükis kirjeldati, missugust testimise dokumentatsiooni testimise protsessi jälgimisel ning testimise lõpetamisel peamiselt koostatakse.

Töö teises osas kirjeldati läbiviidud kvalitatiivset uuringut erinevate projektide testijate seas. Esmalt kirjeldati, kuidas uuring ettevalmistati, milliseid küsimusi ja mis põhjusel küsiti ning kuidas uuring läbi viidi. Uurimuse käigus võrreldi erinevates tarkvaraprojektides rakendatavaid testimise protsessi jälgimise meetodeid ning uuriti, mille alusel ning abil otsustatakse projektis testimine lõpetada. Uuringu käigus otsiti ka hinnangut testimise protsessi läbipaistvuse mõjust arendatava tarkvara kvaliteedile. Uuringu tulemused kirjeldatakse kolmes osas: ülevaatlikud andmed vastanute projektide tüüpide kohta; testimise protsessi kirjeldavad vastused; projekti eduga seotud vastuste analüüs.

Töö käigus jõuti järeldusele, et suur osa töö teoreetilises osas kirjeldatust kajastub ka uuringus osalejate vastustes. Uuringus osalenud väikeste ja keskmiste suurustega tarkvara projektides enamik töö sisus toodud testimise protsessi jälgimise ja lõpetamisega seotud tegevustest ja meetmetest on kasutusel. Küll ei ole võimalik aga teha testimise tegevuste rakendamise kohta otseseid järeldusi projekti edu kohta. Projekti edukuse juures on suur roll näiteks ka kliendil, keda aga antud uuringusse ei kaasatud. Seega saadud tulemuste alusel ei saa öelda, et on olemas kindlad testimise tegevused, mis alati tagavad projekti edu või vastupidi.

Transparency in the process of software testing and exit criteria

Master thesis (30 EAP)

Kaisa Paavo

Summary

According to the wellknown principle of software testing, it is practically impossible to test everything and to find all the possible errors. Therefore, it is important to get a good clear overview of the testing process, in order to know when to complete the testing and when to release developed sofware or product. This thesis provides detailed descriptions of software testing, more precicely how to achieve transparency in software testing process and its completion or exit criteria.

The first part of the thesis gives an overview of the activities and methods related to making the software testing process more transparent and related to testing exit criteria. The theoretical part is based on author's own experience and knowledge from a variety of sources. The first chapter describes how to monitor the testing process in order to achieve the transparency and simplicity. The second chapter describes the activities related to exiting the testing and the methods that can be used to predict the readiness of the product. The third chapter describes what kind of testing documentation is being generated during the testing process monitoring and completion of testing.

The second part of the thesis describes the qualitative research that was carried out among different software project's testers. At first, it is described how a study was being prepared, which questions were asked and why. Different software projects testing process monitoring activities and methods where compared and it was studied which kind of testing completion decisions were used while exiting the testing. The study looked for an assessment of the impact of transparency in the process of testing to the quality of developed software. The results of the research are described in three parts: comprehensive data about the types of respondents projects, the answers to describe the testing process and the analysis of the responses related to project success.

In conclusion it can be said that much of the work described in the theoretical part of this thesis, is also reflected in the research participants' responses. Most of the things mentioned in the thesis theoretical part are being used or done in these small and medium-sized software projects, that were participating in the research. However, it is not possible to make direct conclusions on the implementation of testing activities on the project's success. Success of the project also greatly depends from the client, but that was not enrolled in the research. In the results, it cannot be said that there is specific kind of testing activities, which always ensure project success, or vice versa.

Kasutatud kirjandus

1. Testing definition. [Veebis 03.05.2012]. http://bazman.tripod.com/what_testing.html

2. A. M. Jonasse "Guide to advanced software testing", 2008. ". [Veebis 10.04.2012]. http://site.ebrary.com.ezproxy.utlib.ee/lib/tartu/docDetail.action?docID=10312943&p00=guide%20advanced%20software%20testing

3. C. Keith, M. Cohn „How to fail with agile", „Better Software", juuli/august 2008.

4. T. Nargla "Testija roll agiilses arendusprotsessis". [Veebis 03.05.2012]. https://docs.google.com/viewer?a=v&q=cache:pS1AA7L3w9YJ:cs.ttu.ee/tiki-download_wiki_attachment.php?attId%3D157+agiilne+arendus+protsess&hl=en&pid=bl&srcid=ADGEESjQliuJkAcHb9EsKKlJqkr8aqjeVOSbwDgDumSWZKe98NMGCdTGhkSBCA1u6Zuk8OUriyRplxQqa0-uWtfKSOKCg2CLdQRhKEGuQ5z-qbgAiiEvViUuAyb3hwRJ1gQk13a5WBg1&sig=AHIEtbS4CcPwXdBgHmfd6KSUk3rrpnZ_Jg

5.Tarkvaraarendus. [Veebis 03.05.2012]. http://et.wikipedia.org/wiki/Tarkvaraarendus

6. Verification and validation . [Veebis 03.05.2012]. http://en.wikipedia.org/wiki/Verification_and_validation_(software)

7. John W. Horch "Practical guide to software quality management", Second edition, 2003.

8. Traceability matrix. [Veebis 08.05.2012]. http://en.wikipedia.org/wiki/Traceability_matrix

9. Static Analysis. [Veebis 11.05.2012]. http://www.vigilantsw.com/resources/white-papers/why-static-analysis/

10. U. Hafner, „Static Code Analysis Plug-ins", 2010. [Veebis 11.05.2012]. https://wiki.jenkins-ci.org/display/JENKINS/Static+Code+Analysis+Plug-ins

11. Jenkins. [Veebis 11.05.2012]. https://wiki.jenkins-ci.org/display/JENKINS/

12. B.Hambling, P. Morgan, A. Samaroo, G. Thompson, P. Williams, "Software testing: An ISTQB-ISEB Foundation Guide", Second edition, 2010. [Veebis 08.05.2012]. http://books.google.ee/books?id=sFj3SsYYRJEC&pg=PA150&lpg=PA150&dq=software+testing+progress+monitoring&source=bl&ots=z9fap0PdNW&sig=L5O-7rLEyMEKcADqtWs60I9AiyU&hl=en&sa=X&ei=zAxpT5a9NNKg8gPbg5DWCQ&redir_esc=y#v=onepage&q=software%20testing%20progress%20monitoring&f=false

13. L. Lazic, N. Mastoralis “Cost effective software test metrics”. [Veebis 14.05.2012]. https://docs.google.com/viewer?a=v&q=cache:VwucerD5CgcJ:www.jameslewiscoleman.info/jlc_stuff/project_research/CostEffectiveSoftwareTestMetrics_2008.pdf+Cost+effective+software+test+metrics&hl=en&pid=bl&srcid=ADGEESgVYpLNtNiw7SXtLTsksemMDqqT0SW4AH2aV7FibyiYn_zsi7w23iHhcgqGLQRSUbtz1jG7CJ1Yb0-fnYq8YN3r1i1xQNugPJo9Cd5V6FTxcEpqbAuYj-ICTJ_Uc7LpTqugdCYq&sig=AHIEtbQxo9ZzWchN39EP8eYjcap3n0UVYg
14. Test progress Monitoring, Reporting & Control. [Veebis 08.05.2012]. http://www.softwaretestingtimes.com/2010/08/test-progress-monitoring-reporting.html
15. JIRA. [Veebis 11.05.2012]. http://www.atlassian.com/software/jira/overview
16. J. Kaner, J. Falk, H.Q. Nguyen “Testing Computer Software”, Second edition, 1999 .
17. E. Engström, P. Runeson, A. Ljung, “Improving Regression Testing Transparency and Efficiency with History-Based Prioritization - an Industrial Case Study”, 2011.
18. H.Pham, X. Zhang, ”Software release policies with gain in reliability justifying the costs”, 1999.
19. T. S. Quah “Estimating software readiness using predictive models”, 2009 .
20. J. Tepandi „Tarkvara kvaliteet ja standardid“. [Veebis 09.05.2012]. http://www.lap.ttu.ee/erki/failid/konspekt/tarkvara_kvaliteet_ja_standardid_idx5721/idx5721_konspekt.pdf
21. Software estimation, measurement, and metrics. [Veebis 11.05.2012]. http://www.stsc.hill.af.mil/resources/tech_docs/gsam3/chap13.pdf
22. Software process and product metrics. [Veebis 11.05.2012]. https://docs.google.com/viewer?a=v&q=cache:KKXsLww_wSAJ:groups.engin.umd.umich.edu/CIS/course.des/cis375/ppt/lec7.ppt+product+and+process+metrics&hl=en&pid=bl&srcid=ADGEESgE7zBRxwV0AK-OUlUfeU-BwDj1pAEgfVrMnBlUDBV7ISbc0Bxd5-EO4xinc_OxVSEk90hW7LrwiZ5u_pzGNgg4BwPUDbAaM_QGkm1hplbVb2MQOnHWVSf3FsxqqnBp24U6Sd4i&sig=AHIEtbSH1dVc9xeITLB68OEW8rMJBwsD_Q
23. Software Metrics Programs. [Veebis 11.05.2012]. http://www.eecs.qmul.ac.uk/~norman/papers/Metrics_Prog_Article/metrics_programs.htm

24. M. Bolton „Three kind of measurements and two ways to use them“, “Better Software“, 2009. [Veebis 11.05.2012].
http://www.developsense.com/articles/2009-07-ThreeKindsOfMeasurement.pdf
25. L. Westfall, “Software risk management”. [Veebis 11.05.2012].
http://westfallteam.com/Papers/risk_management_paper.pdf
26. S. Kumaresh, R. Baskaran, „Defect Analysis and Prevention for Software Process Quality Impovement“, 2010. [Veebis 11.05.2012].
https://docs.google.com/viewer?a=v&q=cache:1y-3nOEwSdEJ:www.ijcaonline.org/volume8/number7/pxc3871759.pdf+defect+analysis&hl=en&pid=bl&srcid=ADGEESh20khShrelDSZkCCPUHZDxacRfxbOtTw9yspTwnyqGDiNrxn3H7zSUmwf4mfzrUFd5e95KkJxt55pADGvUHvkSzcqwSj1XHm4BEKzl-9JGERxAzPCcFDSZUwNuaHfBvVnxSkLd&sig=AHIEtbRuOjkoea1SQDTJkunYh7BVcmkYsw
27. *Fishbone* diagramm. [Veebis 13.05.2012].
http://www.itsmsolutions.com/newsletters/DITYvol5iss42.htm
28. M. A. Vandermark „Defect Escape Analysis: Test Process Improvement“, 2003. [Veebis 11.05.2012]. https://docs.google.com/viewer?a=v&q=cache:jb4k-fJtz1kJ:techwell.com/sites/default/files/articles/XDD6919filelistfilename1_0.pdf+Defect+Escape+Analysis:+Test+Process+Improvement&hl=en&pid=bl&srcid=ADGEESh7BLlKM9TPqe6bU-vWRgkF7n1F5eCIytmDScBJFBfx7o1sN8IPs5_FQOY4J1EzTT6-P-Ie1CCA8OiYgQUjC21Cs9K1eBwrV2u8FDc41M23Ulm1PA1HbI7ijShQPup4mvxAW8fg&sig=AHIEtbRGY7VzFPCAJuQ-9eoTo29Mqj94BA
29. R. Hewett, “Mining software defect data to support software testing management”, 2009.
30. J.Pan „Software Reliability“, 1999. [Veebis 11.05.2012].
http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/#models
31. D. W. Johnson „Software Testing Deliverables: From Test Plans to Status Reports“, 2009. [Veebis 11.05.2012]. http://www.testingexcellence.com/software-testing-deliverables-from-test-plans-to-status-reports/
32. W. Echlin „Reporting on Software Test Status“, 2009. [Veebis 02.05.2012].
http://www.softwaretesting.net/blog/2009/03/29/reporting-on-software-test-status//

33. „Kvalitatiivse uurimustöö metodoloogia tarbijakaitsesüsteemidele“, 2010. [Veebis 07.04.2012]. https://docs.google.com/viewer?a=v&q=cache:E-9l10xaHq4J:www.tartutarbija.ee/koduleht/images/stories/Grundtvig/grundtvig_pikoostprojekti_ksiraamat.pdf+kvalitatiivne+uuring&hl=en&pid=bl&srcid=ADGEESg_xwxwg4ONbPW-a3iX82AKVWp1RN5kFw3OfcbNsBjV8LJzFcO16GzIKvKE-oguQ5T7oJQKFRajQOCpmNCoGebtg2sL4UiSDiNxxroR0xgfpOwQAQN_l_S-_xMCuIYe4H5289qL&sig=AHIEtbQUKz1oQz3nPU7z1uvpty_urVGtIg
34. eFormular. [Veebis 07.04.2012]. http://www.eformular.com/avaleht

Lisad

Lisa 1. Küsimustik erinevate projektide testijatele

I osa: Projekti tüüp

1. Mis on teie rakenduse valdkond?
2. Kuidas hindade rakenduse äriloogikat 1-5 skaalal?
3. Palju on rakendusel keskmiselt kasutajaid?
4. Kas rakendus on avalik?
5. Kui palju on teie projektis testijaid / arendajaid / analüütikuid?
6. Kui kaua on kestnud projekti arendamine? Kui kaua veel kestab?
7. Kui pikk on/kui kaua kestab keskmiselt üks arendustsükkel?
8. Mis arendusmeetodit projektis kasutatakse? (waterfall, incremental, spiral, rapid, prototyping, agile)

II osa: Testimise läbipaistvus

9. Kirjeldage oma projekti testimise protsessi? Kuidas üks tööülesanne liigub analüüsist kuni kinnipanekuni?
10. Kuidas / mille abil hindade rakenduse hetkeseisu?
11. Kuidas / mille abil hindade süsteemi kaetust testidega?
12. Millal testimisprotsessi jooksul teete regressiooni testimist? Kuidas valite regressioon testimise jaoks testjuhtumid, mida läbite? Kuidas hindate, kui palju aega iga testjuhtum lisab tervele regressioon testimisele? Kas tunnete end kindlalt, et korrektne / täielik hulk teste sai regressioon testimise jaoks iga kord valitud?
13. Mille alusel otsustate testimise lõpetada?
14. Missugust testimise alast dokumentatsiooni arendustsükli jooksul koostate?
15. Millist dokumentatsiooni tarnega kaasa pannakse?

III osa: Projekti edu

16. Kuidas klient hindab tarne kvaliteeti?
17. Milline on projekti finantsedu? Kas projekt on graafikus ja eelarves?

Lisa 2. Küsimustiku vastajate vastused

Kaasasoleval CD-l „Magistritöö_Paavo_lisa2“ fail nimega Küsimustiku_vastused.xls