TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Andri Kaaremäe 154819IABB

TARKVARASÜSTEEMI TESTIMISE TESTIHALDUSVAHENDITE ANALÜÜS ETTEVÕTTE FOB SOLUTIONS NÄITEL

Bakalaureusetöö

Juhendaja: Jaak Tepandi
Professor

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Andri Kaaremäe

31.12.2019

Annotatsioon

Bakalaureusetöö on kirjutatud teemal „Tarkvarasüsteemi testimise testihaldusvahendite analüüs ettevõtte FOB Solutions näitel.“ Tarkvara testimisprotsessi kõikide testjuhtumite ja sessioonide haldamine võib muutuda üsna aeganõudvaks ning keeruliseks. Lõputöö teema idee tekkis autoril töötades tarkvara testimisega tegelevas ettevõttes FOB Solutions.

Hetkel hallatakse suurt osa ettevõtte FOB Solutions testimisprotsessist Excel’i (*Microsoft Excel*) failides. Igapäevatöö efektiivsemaks ja kiiremaks muutmiseks vajab aga ettevõte testihaldusvahendit, mis toetaks nii testjuhtumite kui ka sessioonide haldamist.

Esmalt annab autor ülevaate tarkvarasüsteemi testimisest ning testimisprotsessi elutsüklist. Täpsemalt kirjeldatakse funktsionaalset testimist, regressioontestimist ning suitsutestimist. Töö praktilises osas valib autor välja neli testihaldusvahendit, mida analüüsitakse, katsetatakse reaalsete andmetega ning võrreldakse omavahel.

Lõpuks antakse soovitused kõige sobivama testihaldusvahendi kasutuselevõtuks ettevõttes FOB Solutions. Lisaks tutvustatakse ideid lõputöö edasiseks uurimiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 38 leheküljel, 6 peatükki, 13 joonist.

Abstract

Analysis of Test Management Tools for Software System Testing Based on FOB Solutions

The bachelor thesis is written about software system testing test management tools. Managing the software testing can get quite time consuming and difficult as there are a lot of test cases and test sessions to document. The subject emerged when the author was working in a small software testing company FOB Solutions.

Currently FOB Solutions is managing the testing process in different Excel files. In order to make the daily work more efficient and faster the company needs a test management tool that supports both test case and test session management.

Firstly, author gives an overview of the software system testing. Functional testing, regression testing and smoke testing are described in detail. In the next paragraph author describes the testing process life cycle. Based on the needs of FOB Solutions, previous research and earlier work experience in software testing the author puts together eleven software requirements in order to find out the most appropriate test management tool for software system testing.

In the practical part of the thesis four different test management tools are selected, tested out with real data, analysed and compared to each other. Following this, it becomes clear that three of the tested four tools meet all the requirements. After comparing the test management tools the author ranks them based on given weights.

Finally, recommendations for selecting the most suitable test management tool for FOB Solutions are given. Furthermore, ideas for further investigation of the thesis are introduced.

The thesis is in Estonian and contains 38 pages of text, 6 chapters, 13 figures.

Lühendite ja mõistete sõnastik

Android	mobiilsetel seadmetel levinud operatsioonisüsteem
CSV	*Coma Separated Values*, komaga eraldatud väärtused
Excel	Microsoft Excel, tabeltöötluse programm
exploratory testing	uuriv testimine on testimise tüüp, mille käigus järgmised testlood tulenevad eelmise tulemustest ning testija testimise ajal tekkivatest ideedest
iOS	ettevõtte Apple arendatud mobiilsete seadmete operatsioonisüsteem
Jira	maailmas enim kasutatud veebipõhine arendusprojektide haldamise tarkvara
PDF	*Portable Document Format,* teisaldatav dokumendi formaat
QA	*Quality Assurance,* kvaliteedi tagamine
regressioontestimine	testimise tüüp, mis on suunatud vigade avastamisele varasemalt testitud tarkvaras
sanity testing	meelerahu testimine on testimise tüüp, mis kontrollib tarkvaramooduleid sügavalt peale uuenduste elluviimist
smoke testing	suitsutestimine on testimise tüüp, mis hindab ja testib olulisi funktsioone
usability testing	kasutatavuse testimine on testimise tüüp, mis kontrollib, kas süsteem vastab esitatud nõuetele arusaadavuse, õpitavuse , kasutusmugavuse osas
Windows Phone	operatsioonisüsteemi Windows mobiilne versioon

Sisukord

Jooniste loetelu

1 Sissejuhatus

Tarkvara loomine on kiires arengus ning aina enam on saadud aru, et kvaliteetse toote või programmi tegemiseks on vaja seda enne klientidele tarnimist põhjalikult testida. Testimine kuulub ka programmeerijate peamiste tööülesannete hulka, kuid kvaliteetsema kontrolli aitavad tagada arendusmeeskonna välised erapooletud *QA* (*Quality assurance*) meeskonnad.

Tarkvara testimine on hädavajalik, sest esiteks aitab see pikas perspektiivis raha säästa. Vaatamata sellele üritatakse testimist sageli aja kokkuhoiu mõttes vältida või edasi lükata. Süsteemi vigade hilisem parandamine on aga mitmeid kordi kallim kui nende varane avastamine. Samuti võib tekkida püsiv kahju ettevõtte mainele. Teiseks aitab pidev testimine tagada turvalisuse. Klientide jaoks on oluline, et nende isiklikud andmed oleksid kindlalt kaitstud. Samuti tagab testimine, et tarkvara vastaks esmastele visioonidele. Tähtis on kontrollida nii funktsionaalsuse, disaini ja ühilduvuse vastavust algselt paika pandud kriteeriumitele ja nõuetele. Tarkvara edu aluseks on aga klientide rahulolu. Testimisprotsess aitab üles leida vead ning puudujäägid, mis muidu võivad kasutajakogemuse täielikult rikkuda [1].

Antud uurimistöö eesmärgiks on leida ettevõttele FOB Solutions sobiv testihaldusvahend, mis võimaldab tarkvarasüsteemi testimise testjuhtumite ja testsessioonide mugavamat haldamist. Hetkel tehakse seda suuremalt jaolt eraldiseisvates Excel'i failides. Eesmärkide saavutamiseks on esmalt defineeritud teoreetilised alused. Informatsioon on kogutud nii kirjalikest kui ka veebis olevatest allikatest. Seejärel annab autor ülevaate ning katsetab erinevaid turul olevaid testihaldusvahendeid, et vastavalt esitatud nõuetele leida kõige sobilikum.

Uurimistöö sisu algab peatükist 2, kus antakse ülevaade funktsionaalsest-, regressioon-, ja suitsutestimisest. Peatükis 3 on kirjeldatud kogu tarkvara testimisprotsessi elutsükkel, mille hulka kuuluvad testimisprotsessi ettevalmistus ja planeerimine, testimine ja testjuhtumite haldamine, testitulemuste kogumine ja hindamine. Lisaks on defineeritud probleemid ning paika pandud nõuded otsitavale tööriistale. Peatükis 4 katsetab autor

erinevaid testihaldusvahendeid ning teeb neist ülevaate. Sisu viimase osana võrdleb autor testitud tööriistu ning annab soovitused, milline neist võiks kõige paremini sobida just ettevõtte FOB Solutions töövõtete lihtsustamiseks.

2 Tarkvarasüsteemi testimine

Tarkvarasüsteemi testimine on testimise liik, mille käigus kontrollitakse, et integreeritud ning valmis programm vastaks eelnevalt paika pandud tingimustele ja nõuetele. Seda viiakse läbi musta kasti meetodil, mis tähendab, et testija ei pea varasemalt tundma programmi koodi. Tarkvarasüsteemi testimise hulka kuuluvad taastumise-, turbe-, kasutajaliidese, ühilduvuse- ja teised testid, mille kõigi ühiseks ülesandeks on kinnitada, et tarkvara osad on vigadeta integreeritud ja funktsionaalsed [2].

Tarkvarasüsteemi testimist kui testimise liiki on tihti valesti mõistetud. See on oma olemuselt kõige keerulisem testimise osa, mille eesmärgiks on demonstreerida, kuidas tarkvara tervikuna ei vasta ette kirjutatud nõuetele. Süsteemi testimist ei ole võimalik teostada kui varasemalt ei ole tarkvarale või tootele valmis kirjutatud mõõdetavaid eesmärke. Eesmärgid panevad paika kuidas ning kui hästi tarkvara peaks toimima, kuid need ei kirjelda eraldiseisvate osade funktsionaalsust. See muudab raskeks testjuhtumite loomise, sest puudub kindel metoodika, millest saaks lähtuda. Seetõttu nõuab tarkvarasüsteemi testimise testjuhtumite kirjutamine isegi rohkem loovust ja kogemust kui seda nõuab tarkvara või süsteemi enese disainimine [3].

Süsteemi testimine võib oma olemuselt olla nii funktsionaalne kui ka mittefunktsionaalne. Funktsionaalne testimismeetod põhineb musta kasti meetodil ning ainsaks juhiseks testjuhtumiste loomisel on tarkvarale esitatud nõuded. Funktsionaalse testimise korral ei pea tundma programmi koodi, sest oluline on vaid funktsioonide sisend- ja väljundväärtuste verifitseerimine [4]. Mittefunktsionaalne testimine seevastu kontrollib programmi jõudlust, usaldusväärsust, mastaapsust ning kõike, mis ei ole seotud funktsionaalsusega [5].

On äärmiselt tähtis, et tarkvarasüsteemi testimisega ei tegeleks tarkvara programmeerijad. Kõikidest testimise liikidest on see ainus, kus testimisprotsessis tuleks eelistada inimesi, kes pole seotud antud tarkvara arendusega. Süsteemi testimine nõuab objektiivset lähenemist ning testimisprotsessile peaks lähenema lõppkasutaja vaatepunktist. Kõige

parem lahendus on palgata väline kvaliteedikontrolli meeskond, kes ei ole tootest psühholoogiliselt mõjutatud [3].

Tarkvarasüsteemi testimise käigus testitakse kogu tarkvara, et tagada selle oodatud toimimine vastavalt nõuete dokumentidele. Regressioontestimise käigus tehakse kindlaks, et peale versiooniuuendusi või vigade parandusi on tarkvara jätkuvalt toimiv ning soovitud puudujäägid kõrvaldatud. Suitsutestimise (*smoke testing)* käigus kontrollitakse vaid tarkvara põhifunktsionaalsust ning stabiilsust [6]. Järgnevates peatükkides annab autor ülevaate funktsionaalsest-, regressioon- ja suitsutestimisest, toob välja nende erinevused ning vajalikkuse.

2.1 Funktsionaalne testimine

Funktsionaalne testimine on üks oluline osa tarkvara kvaliteedi tagamisest. Vigu otsitakse musta kasti meetodil. Toimivatele tarkvara funktsioonidele sisestatakse erinevaid väärtusi ning võrreldakse väljundeid tarkvara nõuetega. Nii on võimalik leida vead funktsionaalsuses. Kogu süsteem testitakse läbi väikeste osade kaupa. Funktsionaalse testimise tüübid on näiteks:

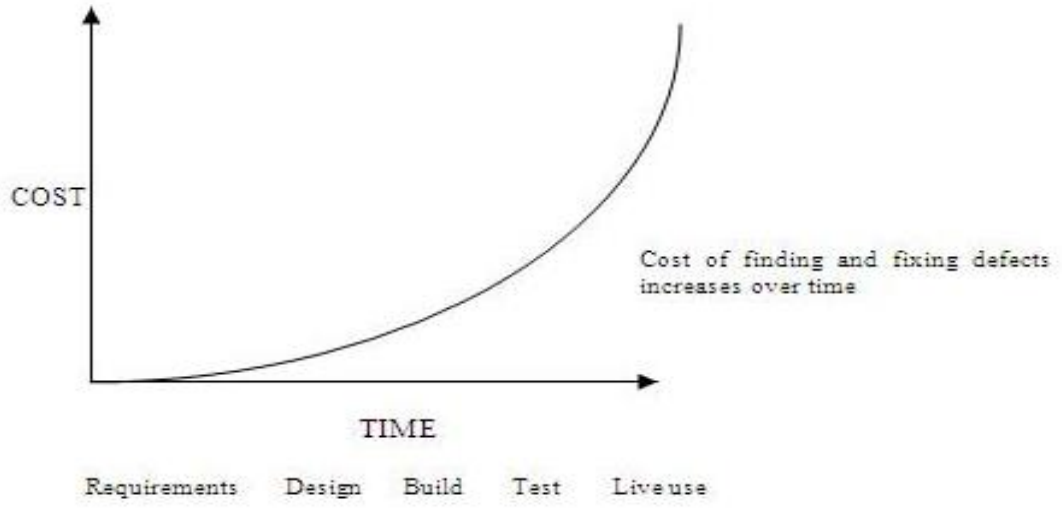
- Suitsutestimine (*Smoke testing).*
- Meelerahu testimine (*Sanity testing).*
- Regressioontestimine (*Regression testing).*
- Kasutatavuse testimine (*Usability tesing)* [7].

Funktsionaalne testimine on kõige efektiivsem kui tarkvara nõuded koostatakse otse ärinõuete põhjal, mitte süsteemi nõudeid ega disaini dokumente järgides. Nii saab vajalikud andmed otse esialgsest allikast. Võib eristada viite protsessi. Esmalt alustatakse tarkvara ettenähtud funktsionaalsuse kindlaksmääramisega. Sama informatsiooni põhjal luuakse sisendandmed, tehakse kindlaks oodatavad väljundandmed. Läbitakse testjuhtumid ning võrreldakse oodatud ning tegelikke väljundandmeid [8].

2.2 Regressioontestimine

Tarkvara koodi uuendatakse ja muudetakse aja jooksul vaatamata sellele kui hästi ja kvaliteetselt see esialgselt kirjutatud on. Regressioontestimise eesmärk on tagada, et selliste versiooniuuenduste ja vigade paranduste käigus jääks kogu vana funktsionaalsus toimima ning et lisatud uuendused oleksid samuti funktsionaalsed ja vigadeta [9].

Vigade kulukus sõltub nende mõjust tarkvara funktsionaalsusele ning avastamise kiirusele. Mida varem probleemsed kohad leida, seda odavam on ka nende parandamine. Näiteks kui viga avastatakse juba tarkvara nõuete loomise faasis, siis on selle parandamine veel üsna odav. Joonis 1 näitab tarkvara vigade parandamise kulukust sõltuvalt vigade avastamise ajast [10].



Joonis 1. Vigade parandamise maksumus sõltuvalt vigade avastamise ajast

Regressioontestimine on tähtis, kuid kallis hoolduskulu, mis tagab, et koodimuudatused ei ole negatiivselt mõjutanud tarkvara toimimist. Regressioontestimine põhineb peamiselt olemasolevatel testjuhtumitel, mida läbides tagatakse programmi jätkuv funktsionaalsus. Kulukaks muudab selle aga asjaolu, et testjuhtumeid võib ajaga koguneda väga palju [11].

Regressioontestimisele võib läheneda kolme erinevat moodi. Esimeseks meetodiks on täielik regressioon, mille käigus testitakse läbi kõik olemasolevad testjuhtumid. See on küll kindlaim viis leida üles kõik vead, kuid samal ajal nõuab see väga palju aega ja ressurssi. Teine meetod on teatud hulga testjuhtumite valimine. Läbitakse testjuhtumid,

mis kõige tõenäolisemalt tagavad, et tehtud muudatused ei ole koodi tekitanud uusi vigu. Kolmas meetod on testjuhtumine prioritiseerimine. See kujutab endast testjuhtumite järjekorda seadmist vastavalt testimise eesmärkidele ja vajadustele [12].

Suure hulga testjuhtumite paremaks organiseerimiseks ning protsessi kiirendamiseks on välja töötatud testjuhtumite prioritiseerimise tehnikad. Põhinedes kogutud informatsioonile muudetud koodi ning testjuhtumite kohta aitavad need testimise aja- ja finantskulusid vähendada. Olenevalt testimise eesmärgist on testjuhtumite läbimist võimalik väga erinevalt järjestada:

- Testijad soovivad suurendada vigade leidmise tõenäosust.
- Testijad soovivad suurendada koodi katvust testidega, et kiiremini läbi töötada suurem osa tarkvarast.
- Testijad soovivad võimalikult kiiresti tagada koodi usaldusväärsuse ja töökindluse.
- Testijad soovivad esmalt leida kõige suurema riskiga vead.
- Testijad soovivad esmalt avastada kindla tarkvara osaga seotud vead [13].

Regressioontestimist on võimalik läbi viia ka uuriva testimise (*exploratory testing*) metoodikat kasutades. Sellisel juhul ei kasutata varasemalt kirja pandud testjuhtumeid, vaid proovitakse vigu leida oma teadmistele tuginedes, pidevalt õppides ning eksperimenteerides. Seetõttu sõltub efektiivsus testija oskustest ning kogemustest. Uuriva testimise puhul viiakse läbi kokkulepitud kestusega testsessioone, mille käigus keskendutakse segamatult ühe tarkvara funktsionaalsuse testimisele. Testsessiooni alguses pannakse paika selle eesmärk, kestus ning kasutatavad seadmed. Kogu protsessi jooksul kirjutatakse üles läbitud stsenaariumid ning vigade leidmise korral raporteeritakse need testsessiooni lõppedes. Sellise lähenemisega on võimalik leida üles vead, mida traditsioonilised testjuhtumid ei pruugi katta [14].

2.3 Suitsutestimine

Suitsutestimine on funktsionaalse testimise üks tüüp, mida tavaliselt viivad läbi arendusest eraldiseisvad QA meeskonnad. Selle eesmärgiks on kiiresti leida vigu ning

tõestada, et tarkvara uue versiooni põhjalikumalt edasi testimine ei ole enam mõttekas. Suitsutestid võivad olla automatiseeritud ning sellisel juhul saab samu testjuhtumeid kasutada iga versiooniuuenduse verifitseerimiseks. Testitakse tarkvara osasid, mis peaksid olema funktsionaalsed. Kui selle käigus leitakse mõni regressioon või tõsisem funktsionaalne viga, siis saadetakse tarkvara enne edasist testimist uuesti arendusmeeskonnale parandamiseks. Parandatud versiooni puhul kordub kogu protsess uuesti [15].

Järgnevad omadused iseloomustavad suitsutestimist:

- Ärikriitiliste funktsioonide tuvastamine, millele toode peab vastama.
- Tarkvara põhifunktsioonide kujundamine ja käivitamine.
- Tagamine, et testimise jätkamiseks on kõik eelnevad testjuhtumid edukalt läbitud.
- Kriitiliste vigade kiire avastamine eesmärgiga säästa aega ja ressursse.
- Suitsutestid võivad olla nii manuaalsed kui ka automatiseeritud [16].

Üldiselt viiakse suitsutestimist läbi manuaalselt, kuid see võib erinevates organisatsioonides paljuski erineda. Kohe kui uus tarkvara funktsionaalsus on valmis ning integreeritud olemasoleva versiooniga, siis viiakse testkeskkonnas läbi ka suitsutestimine. Läbitakse testjuhtumid, mis on loodud tarkvara uuendusi silmas pidades. See aitab kindlaks teha, kas kõik tähtsamad funktsioonid on jätkuvalt toimivad. Kui suitsutestimine on edukas, siis võib QA meeskond edasi liikuda põhjaliku funktsionaalse testimise juurde [17].

Suitsutestimine võib olla funktsionaalne- või ühiktestimine. Funktsionaalsed testid kontrollivad kogu tarkvara erinevate sisenditega. Need võivad sisaldada varasemalt valmis kirjutatud sisendite seeriat ning võimaluse korral isegi automatiseeritud mehhanismi hiire liikumise jäljendamiseks. Ühiktestid seevastu kontrollivad üksikuid funktsioone, meetodeid või objekte. Need võivad olla koodi sisse kirjutatud või asuda eraldiseisvas failis [18].

3 Testimisprotsessi elutsükkel

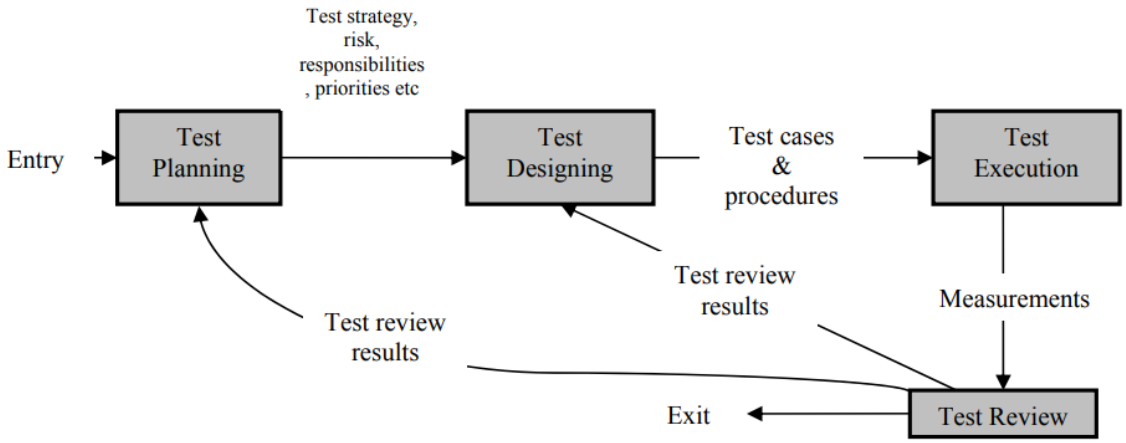
Tarkvara testimise elutsükkel on järjestatud etappidest koosnev testimisprotsess, mida järgides tagatakse kvaliteedieesmärkide täitmine. Igal elutsükli etapil on oma väljundid ja eesmärk, mis saavutatakse planeeritud ning süsteemse lähenemisega [19].

Testimisprotsess on tarkvara arenduse väga tähtis osa, mille käigus läbitakse erinevad etapid: projekti analüüs, testide planeerimine, testjuhtumite/testandmete/testkeskkonna ettevalmistamine, testimine, vigade logimine ja jälgimine, testimisprotsessi lõpetamine. On läbi viidud mitmeid uuringuid eesmärgiga optimeerida kogu testimisprotsess, et tagada tarkvara kvaliteet minimaalse ajakuluga. Pärast mitmete olemasolevate uurimistööde hindamist on jõutud järeldusele, et sama tarkvara testimiseks on võimalik kasutada väga erinevaid tehnikaid. Testimisprotsessis saab välja tuua kolm suuremat kategooriat, milleks on:

1. Ettevalmistus ja planeerimine, mille hulka kuulub ka testide disainimine.
2. Testimine ja testide haldamine.
3. Testitulemuste kogumine ja hindamine [6].

Neid on võimalik kohandada vastavalt ettevõttele või projekti nõuetele, kuid peamised kategooriad, millest lähtuda, jäävad siiski samaks.

Joonisel 2 on kujutatud testimisprotsessi elutsükkel [20].



Joonis 2. Testimisprotsessi elutsükkel

Selleks, et kogu testimisprotsess saaks olla edukas, peab see sisaldama nii madalama kui ka kõrgema tasemega testjuhtumeid. Madala tasemega testjuhtumid kontrollivad tarkvara koodi komponentide tasemel. Kõrgema tasemega testjuhtumid seevastu teevad kindlaks, kas tarkvara toimib vastavalt kirja pandud nõuetele. Testimisprotsessi elutsükkel on kui juhend testijatele ja kliendile. See muudab kogu testimisprotsessi arengu mõõdetavaks [20]. Järgnevalt kirjeldab autor kolme suuremat testimisprotsessi elutsükli kategooriat täpsemalt.

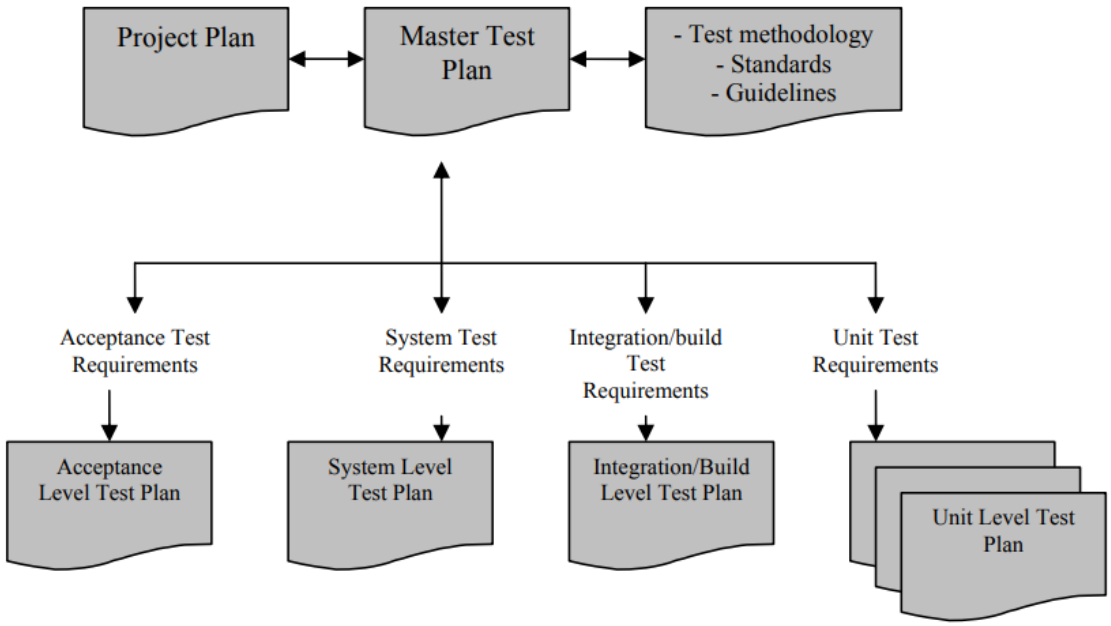
3.1 Testimisprotsessi ettevalmistus ja planeerimine

Esmaseks etapiks on projekti analüüs, mis on tarkvara testimise alustamiseks hädavajalik, sest see hõlmab nii funktsionaalsete kui ka mittefunktsionaalsete nõuete läbitöötamist. Nendeks võivad olla näiteks ärinõuded ja tehnilised nõuded. Analüüsi käigus pannakse paika tarkvara eeldatav käitumine, millele hiljem tuginetakse. Testijate meeskond defineerib testimismeetodid, mida hakatakse kasutama. Selgitatakse välja prioriteedid ning jaotatakse tööülesanded meeskonna siseselt. Kogu etapp toimib pidevas koostöös kliendiga. Samuti tehakse kindlaks, milliseid tarkvara nõudeid ei ole tulenevalt testkeskkonna piirangutele või muudele asjaoludele võimalik testida [6].

Järgnevalt alustatakse planeerimise ja ettevalmistustega. Testimisprotsessi planeerimine on tarkvara eduka testimise eelduseks. Sellega alustamine märgib üleminekut ühelt tarkvara arenduse etapilt järgmisele. Pannakse paika eeldatav testide arv ja nende läbimise ajakulu. Samuti määratakse testimise lõpetamise eelduseks olevad kriteeriumid, tehakse kindlaks võimalikud riskid ning jaotatakse ressursse. Projekti nõuetest,

vajadustest, riskianalüüsist ning olemasolevatest oskustest lähtuvalt valitakse välja testimismeetodid, tehnikad ning tööriistad. Ettevalmistuse ja planeerimise etapi tulemiks on testplaani dokument, mis tegelikult koostatakse iga etapi lõppedes [20].

Testplaanide koostamine erinevates testimisprotsessi etappides sõltub projekti suurusest. Väiksemate projekti korral on võimalik, et piisab vaid ühest dokumendist. Kuna enamasti koostatakse testplaan igas testimisetapis, siis on soovitatakse luua üks peamine testplaan, kuhu sisse kogutakse projekti üldine informatsioon ning viited teistele täpsustavatele testplaanidele, milleks on näiteks projekti plaan, testjuhtumite standardid, testimise lõpetamise kriteeriumid, süsteemitestimise plaan, integratsioonitestimise plaan, ühiktestimise plaan ning teised testimisprotsessiga seotud dokumendid. Järgnevalt on joonisel 3 välja toodud testplaanide koostamisest [20].



Joonis 3. Testplaanide koostamine

Ettevalmistuse hulka kuulub ka testjuhtumite disainimise protsess, mis hõlmab endas mitmeid tähtsaid tegevusi: testjuhtumite koostamise tehnika valimine, testandmete ettevalmistamine, testimisprotseduuride väljatöötamine, testkeskkonna ülesseadmine ning sobivate tööriistade valimine. Üheks suureks veaks, mida tehakse, on suurema tähelepanu pööramine testjuhtumite läbimisele mitte nende disainimisele, sest hästi koostatud testjuhtumid on võimelised avastama vigu efektiivsemalt ning kiiremini [20].

Ettevalmistuse käigus alustavad testijad testjuhtumite koostamist. Iga tarkvarale esitatud nõude kohta kirjutatakse nii positiivne kui ka negatiivne stsenaarium, et tagada tarkvara

täielik kattuvus testidega. Tarkvara muudatuste korral hoiab see ka ära suure hulga testjuhtumite ümbertegemise vajaduse [6].

Testjuhtumite disainimise teevad raskeks ajalised ja finantsilised piirangud. Tähtis on luua võimalikult väike hulk testjuhtumeid, mis suudavad suurima tõenäosusega leida kõige rohkem vigu. Samuti peab jälgima, et testjuhtumite koostamine oleks testijate meeskonnas standardiseeritud ning kõigile arusaadav. Testplaane ning nendega seotud dokumente peab pidevalt uuendama ning ajakohastama, et säilitada testjuhtumite efektiivsus. Sellisel juhul on neid võimalik kasutada ka järgnevates sarnastes projektides. Kui järgneva testimise etapi ajal mõnda tarkvara funktsionaalsust uuendatakse või täiendatakse, siis peab see kajastuma ka testimisprotsessi kõigis vastavates dokumentides [20].

3.2 Testimine ja testjuhtumite haldamine

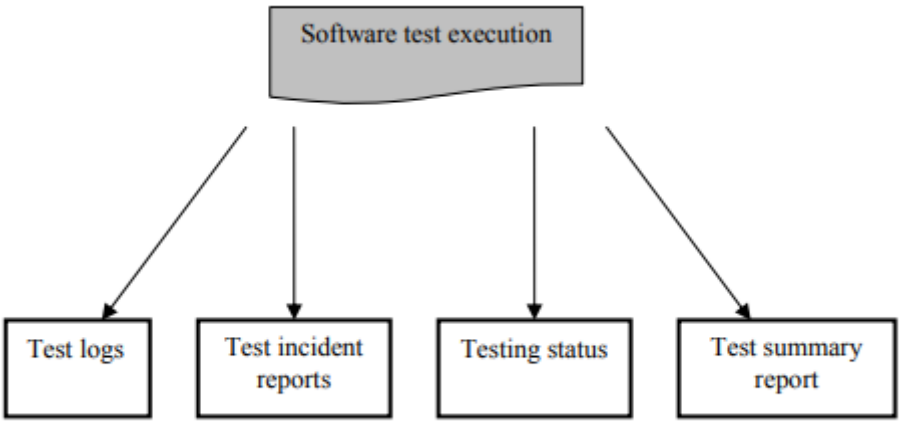
Järgmiseks etapiks ongi testimine. Tarkvara testimisele võib läheneda väga erinevalt. Üheks võimaluseks on seda teha uurivat testimist kasutades. Teiseks võimaluseks on läbida testjuhtumeid ning kui selgub, et oodatav ja reaalne tulemus ei ühti, siis on testija ülesandeks viga raporteerida ning see arendajatele edasi suunata [6]. Peamised tegevused käesolevas etapis on järgmised:

- Eelnevalt planeeritud testjuhtumite ja testsessioonide läbimine.
- Testjuhtumite olekute dokumenteerimine.
- Vigade leidmisel nende raporteerimine ning arendajatele suunamine.
- Vigade parandamisel testjuhtumite uuesti läbimine.
- Testide läbimine, kuni jõutakse oodatava tulemuseni [21].

Antud etapi tähtsaks osaks on ka testjuhtumite ja testsessioonide haldamine. Selleks kasutatakse spetsiaalseid testihaldusvahendeid, mis toetavad nii manuaal- kui ka automaatteste. Testihaldusvahendid on mõeldud testjuhtumite ja testsessioonide haldamiseks ning käivitamiseks. Testihaldusvahendite lõppeesmärgiks on kasutajale pakkuda testide tulemuste põhjal loodud täpseid ning informatiivseid graafikuid ja tabeleid. Need annavad parema ülevaate tarkvara testimisprotsessi hetkeseisust ning on

hea sisend aruannetesse. Lisaks on paljudel testihaldusvahenditel kas sisseehitatud või integreeritav vigade raporteerimise- ja jälgimise võimalus [22].

Testimise etapi väljunditeks on vahejuhtumite aruanded, testide logid, testide olekud ning testjuhtumeid kokkuvõtvad aruanded (joonis 4) [20].



Joonis 4. Testimise etapi väljundid

Testimise alustamisel on soovituslik alustada põhilise funktsionaalsuse kontrollimisest. Testitavuse hindamise kriteeriumid ning uuendustele suunatud suitsutestimise läbimine aitavad kindlaks teha, kas tarkvara on valmis edasiseks põhjalikumaks testimiseks. Testjuhtumite täitmisel avastatakse peaaegu alati uusi probleemseid stsenaariumeid, mis tuleks samuti dokumenteerida eesmärgiga luua põhjalik testjuhtumite kogu. Kui testjuhtum on edukas ehk leitakse uus viga tarkvaras, siis raporteeritakse see vastavalt kokkulepitud reeglitele. Igale veale määratakse tõsiduse aste, mida kajastatakse ka koostatavates aruannetes. Testimise edenemise jälgimiseks dokumenteeritakse pidevalt testimise olek. Järgnevalt on välja toodud nende aruannete sisu:

- Testitud funktsionaalsuse või koodi osade kirjeldus.
- Testjuhtumite ja testsessioonide koguarv.
- Läbitud testjuhtumite ja testsessioonide koguarv.
- Testjuhtumite kaal põhinedes kas ajakulule või koodi katvusele.
- Läbitud testjuhtumite protsent kindlate moodulite suhtes.
- Läbitud testjuhtumite protsent kogu tarkvara suhtes [20].

3.3 Testitulemuste kogumine ja hindamine

Viimaseks väga oluliseks etapiks on testimise lõpetamine. Vastavalt planeerimise etapis paika pandud lõpetamise kriteeriumitele otsustatakse, kas tarkvara on testimise edukalt läbinud, nõuetele vastav ning valmis klientidele avaldamiseks [6]. Testijate ülesandeks on koostada ka projekti lõpuraport, kus tuuakse välja testimise tulemused ning tehakse kokkuvõtted.

Testimisprotsessi juures on üks raskemaid küsimusi see, millal peaks testimise lõpetama. Ei ole ühtegi võimalust teha kindlaks, kas leitud viga oli viimane. Kui tegemist ei ole just äärmiselt väikese funktsionaalsusega tarkvaraga, siis on võimatu testida kõikvõimalikke stsenaariume ning seetõttu ei saa kunagi väita, et kõik vead on leitud. Seepärast pannakse testimisprotsessi planeerimise käigus paika testimise lõpetamise kriteeriumid. Kaks kõige enam levinud tingimust on järgmised:

- Lõpetada, kui testimisele planeeritud aeg on täis.
- Lõpetada, kui kõik testjuhtumid ebaõnnestuvad ehk rohkem vigu ei leita [23].

Antud etapi eesmärk on analüüsida kogutud andmeid ning nende põhjal hinnata ja anda tagasisidet eelnevate etappide kohta. Kogu testimisprotsessi tulemuste ülevaatamisel on võimalik analüüsida tarkvara usaldusväärsust, testide katvust ning leitud vigasid. Usaldusväärsuse analüüsi põhjal saab otsustada, kas lõpp-produkt vastab planeerimise etapis seatud eesmärkidele ja nõuetele. Kui kõik tingimused on täidetud, siis on tarkvara valmis klientidele avaldamiseks. Vastasel juhul peab arendusmeeskond veel parandusi tegema. Testide katvuse analüüsi saab kasutada kui alternatiivset kriteeriumit testimise lõpetamiseks. Leitud vigade analüüsi järgi on võimalik määrata kindlaks tarkvara probleemsemad kohad, mis omakorda aitab edaspidiseid vigu vältida [20].

Testimisprotsessi lõppedes koostatakse kliendile suunatud testimise lõpuraport. See sisaldab kokkuvõtet läbitud testjuhtumitest ning nende tulemustest. Lõpuraport on lühike, kuid informatiivne. See peab olema arusaadav ning loetav ka inimestele, kes pole antud valdkonnaga varem tihedalt kokku puutunud. Järgnevalt on välja toodud mõned teemad, mis võiksid olla lõpuraportis kajastatud.

- Ülevaade testimise eesmärkidest.

- Testimiseks valitud meetodid ning testimise tüübid.
- Tarkvara funktsionaalsus või osad, mida testiti.
- Tarkvara funktsionaalsus või osad, mida ei testitud.
- Testimiseks kasutatud seadmed ning operatsioonisüsteemid.
- Ülevaade, joonised, graafikud, statistika ning kokkuvõtted leitud vigadest.
- Kogu testimisprotsessi kokkuvõte ning tagasiside andmine kliendile [24].

4 Probleemide ja nõuete defineerimine

Selles peatükis toob autor välja tähtsamad probleemid, mis võivad tarkvarasüsteemi testimise elutsükli jooksul tekkida. Samuti antakse ülevaade ettevõttest FOB Solutions. Defineeritud probleemidest ning ettevõtte vajadustest ja eripäradest lähtudes koostatakse nõuded järgmises peatükis otsitavale testihaldusvahendile.

4.1 Probleemide defineerimine

Tarkvara testimisprotsessi elutsükli jooksul koguneb väga palju vajalikku informatsiooni, mis on kõik vaja kirjalikult talletada. Traditsioonilisel tarkvarasüsteemi testimisel kasutatakse testjuhtumeid, mida hoitakse eraldi dokumentides. Testimise käigus leitakse alati juurde uusi vigade avastamise stsenaariumeid, mis on vaja talletada. Kogu informatsioon peab olema arusaadavalt ning visuaalselt selgelt esitatud, et vältida segadusi ning ebavajalikku ajakulu. Siin tekib esimene probleem. Ettevõtetes, kus ei ole kasutusel mugavat testihaldussüsteemi, on sobivate testjuhtumine väljavalimine, grupeerimine ning vajadusel prioritiseerimine väga tülikas.

Viimasel ajal kasutatakse regressioontestimise läbiviimiseks sageli ka uurivat testimist. See vähendab testimisele kuluvat aega ning võimaldab üles leida vead, mis traditsiooniliste testjuhtumitega võivad jääda avastamata. Uuriv testimine nõuab aga teistsugust lähenemist, sest enam ei kirjutata testjuhtumeid. Selle asemel peab olema testkeskkond, mis võimaldab hallata testsessioone ning vajadusel neid vigade raporteerimise keskkonnaga siduda. Eraldiseisvates Exceli failides testsessioonide haldamine on väga aeganõudev ning raskesti organiseeritav.

Testide tulemuste põhjal luuakse kliendile suunatud vahearuanded ning lõpuaruanne. Aruanded peavad olema kliendi jaoks informatiivsed, kuid samal ajal huvitavad lugeda. Seda aitavad saavutada testide tulemuste põhjal koostatud statistika, graafikud ning joonised. Modernsed testihaldusvahendid automatiseerivad kogu visualiseerimise ning andmete töötlemise.

Tarkvaraüsteemi testimise elutsüklist saab välja tuua kaks selget probleemi, millele on võimalik lahendus leida sobiva testihaldusvahendi kasutuselevõtuga.

1. Testjuhtumite ja testsessioonide haldamine.
2. Testimise aruannete jaoks vajalike informatiivsete graafikute, jooniste ning statistika koostamine.

4.2 FOB Solutions ülevaade

FOB Solutions OÜ on 2013 aastal asutatud tarkvaraarenduse ja kvaliteedikontrolli põhiseid teenuseid ja lahendusi pakkuv ettevõte. Eelkõige keskendutakse mobiilsetele platvormidele (*Android, iOS, Windows Phone*), kuid tegeletakse ka veebipõhiste lahendustega. Enamus tarkvara-alastest teenustest eksporditakse Lääne-Euroopasse. Väiksemas mahus panustatakse ka USA ja Eesti turgudel. Ettevõttes töötab ligi 40 inimest, kes omakorda jagunevad meeskondadesse.

4.3 Nõuded otsitavale tööriistale

Turul on suur valik erinevaid tööriistu, mis on mõeldud kogu testimisprotsessi haldamiseks. Selle uurimistöö eesmärgiks on leida testihaldusvahend, mis oleks sobilik just tarkvarasüsteemi testimiseks. Vastavalt sellele on otsitavale testihaldusvahendile esitatud ka olulised nõuded.

Sobivat tööriista otsides on esmatähtis, et korraga oleks täidetud kolm suuremat tingimust:

1. Testjuhtumite sorteerimise, kategoriseerimise ja prioritiseerimise võimalus.
2. Uuriva testimise testsessioonide haldamise võimalus.
3. Automaatne graafikute, jooniste ja statistika koostamine testihaldusvahendi poolt.

Otsitav testihaldusvahend peab võimaldama testjuhtumite prioritiseerimist, grupeerimist ning haldamist. Testide muudatuste kohta peaks tööriist koguma ja talletama informatsiooni. Lisandub väärtust kui on näha, kes ja millal on muudatusi teinud.

Tööriist peab sisaldama testsessioonide haldamise võimalust uuriva testimise läbiviimiseks. Minimaalselt peaks sessioonides olema võimalik määrata uuritav funktsionaalsus, tehtud toimingud, kasutatud seadmed, tulemused ning kasuks tuleb integratsioon veahaldusvahendiga Jira. Testsessioonid peavad olema kergesti ligipääsetavad, grupeeritavad ning säilima peaks ka muudatuste ajalugu.

Samuti peaks testihaldusvahend testimise tulemuste ja katvuse põhjal suutma koostada nii testijatele kui ka kliendile huvipakkuvaid graafikuid ning informatiivset statistikat. Kasuks tuleb, kui tööriist oskab kogutud statistika põhjal pakkuda lisainformatsiooni vigade leidmise efektiivsuse ning erinevate koodi moodulite usaldusväärsuse kohta. Sellise informatsiooni põhjal on hilisemal testimisel tõhusamate testide ning rohkem tähelepanu vajavate tarkvara osade välja selgitamine mitmeid kordi kiirem ning lihtsam.

Nõuete kirjutamisel lähtub autor tingimusest, et testihaldusvahendit otsitakse väiksemale ettevõttele FOB Solutions, mis tegeleb tarkvara manuaalse ja automaatse tarkvarasüsteemi testimisega ning kus töötab orienteeruvalt 40 inimest. Projektid, mida testitakse, võivad olla nii veebilehed kui ka mobiilirakendused.

Lisaks kolmele suuremale esitatud tingimusele võtab autor sobiva tööriista otsimisel arvesse ka mitmeid teisi faktoreid. Järgnevalt on välja toodud kõik hindamisel arvesse võetud tähtsamad kriteeriumid, mis koostati ettevõtte vajadusi arvesse võttes:

1. Testjuhtumite haldamine – testjuhtumite prioritiseerimine, grupeerimine, muudatuste tegemine.
2. Uuriva testimise haldamine – testsessioonide haldamise võimalus.
3. Aruanded ja meetrika – testide tulemuste statistika töötlemine ning visualiseerimine.
4. Projektide haldamine – võimalus luua mitu erinevat projekti.
5. Tarkvara versioon – millise tarkvara versiooni vastu testitakse.
6. Seadmete haldamine – milliseid seadmeid, platvorme ja operatsioonisüsteeme testimisel kasutatakse.
7. Lihtne õppida – tööriista tundmaõppimise juhendite ja videode olemasolu.

8. Muudatuste jälgimine – testide ning dokumentide muudatuste ajaloo säilitamine.
9. Tööriistasisene suhtlemise võimalus – kasutajate vahelise vestluse pidamise võimalus näiteks testjuhtumite raames.
10. Hind – kasutuselevõtu kogumaksumus peab jääma projekti arvestades kasumlikuks.
11. Integratsioon – minimaalselt võimalik integratsioon veahaldusvahendiga Jira.

5 Turul olevate testihaldusvahendite analüüs ja hindamine

Viimaste aastatega on turule lisandunud väga palju erinevaid testihalduseks mõeldud tööriistu. Valiku teeb veel raskemaks asjaolu, et mõned teenusepakkujad on oma tooted jaganud mitmeks erinevaks versiooniks. Testihaldusvahendit valides ei tohiks keskenduda vaid pakutavatele võimalustele ning lisadele, sest nii võivad tagaplaanile jääda tegelikud vajadused. Esmalt tuleb selgeks teha testitava projekti nõuded ning tingimused ning vastavalt paika pandud kriteeriumitele tuleks hakata sobivat tööriista otsima.

Selles peatükis katsetab ning tutvustab autor nelja testihaldusvahendit, mis on valitud vastavalt eelnevalt paika pandud nõuetele. Võrdlusesse võetud tööriistad on kõrgelt tunnustatud ning laialdaselt kasutusel. Nende kirjeldamisel toetub autor testimise valdkonnas töötamisel saadud kogemustele, koduleheküljel pakutavale informatsioonile ning reaalsetele katsetustele. Iga testitud tööriista kohta toob autor välja plussid ja miinused. Saadud tulemuste põhjal tehakse järeldused ning valitakse välja tarkvarasüsteemi testimise jaoks sobivaim testihaldusvahend, mida oleks võimalik reaalselt juurutada ka ettevõtte FOB Solutions testimisprotsessis.

5.1 TestPad

Esimeseks testihaldusvahendiks on uurimistöös valitud TestPad, mida saab kasutada nii testjuhtumite manuaalseks testimiseks kui ka uurivaks testimiseks. TestPad kombineerib kontrollnimekirjade lihtsuse ja arvutustabelite funktsionaalsuse. Lisaks testjuhtumitele ja sessioonidele on võimalik hallata ka kõikvõimalikke nõudeid, dokumente, lisaandmeid. Puudub testihaldusvahendi sisene vigade raporteerimise võimalus. Tööriista kirjeldamisel toetub autor reaalsetele katsetustele ning koduleheküljel olevale informatsioonile [25].

1. **Testjuhtumite haldamine –** TestPad sarnaneb oma ülesehituselt märkmiku kontrollnimekirjadele. Kasutajal on väga suur vabadus kõike oma soovi järgi paigutada ja grupeerida. Testide grupeerimine, prioritiseerimine ja muudatuste

tegemine on lihtne ja võimalusi on palju. Näiteks on võimalik sorteerida nii filtrite, siltide, tarkvara versioonide kui ka ise loodud tingimuste alusel. Igas testjuhtumis saab täpsemalt kirjeldada vajalikud läbimise sammud ning lisada selgitavaid pilte või faile. Tööriist toetab nii manuaal- kui ka automaattestimist.

2. **Uuriva testimise haldamine –** Tööriist põhineb puu struktuuril, mille kasutaja saab üles ehitada endale sobivalt. Vastavalt sellele kuidas kasutaja andmeid sisestab on võimalik läbi viia nii testjuhtumitel põhinevat testimist kui ka uurivat testimist. Uuriva testimise jaoks saab luua testsessioonide jaoks eraldi failid, kus on võimalik kirjeldada testitud funktsionaalsust, läbitud samme ning tulemusi. Kogu süsteem on väga hästi kohandatav.

3. **Aruanded ja meetrika –** TestPad'i kõige nõrgemaks küljeks on statistika visualiseerimine. Statistika põhineb sellel, kui palju on teste õnnestunud, ebaõnnestunud, blokeeritud või pooleli. Kuvatakse arvuline ja protsentuaalne ülevaade testide koguarvust versioonide ja läbimiste raames. Puudub aga võimalus saada ülevaadet erinevate filtrite ning funktsionaalsuste põhjal. Seevastu saab kogu statistika *CSV* (*comma separated values*) formaadis tööriistast eksportida ning mujal kasutada.

4. **Projektide haldamine –** On võimalus luua mitmeid erinevaid projekte ning projektide siseseid kaustasid ja skripte.

5. **Tarkvara versioon –** Versioonihaldus on olemas ning tarkvara versiooni määramine on lihtne ning kiire. Samuti on salvestatakse testimisandmed varasemate versioonide kohta.

6. **Seadmete haldamine –** Tööriistas puudub sisse-ehitatud seadmete haldamise võimalus. Siiski saab seadmeid, platvorme ja operatsioonisüsteeme ise määrata nii testjuhtumeid kui ka testsessioone läbides. Samuti saab seadme andmed määrata testjuhtumite verifitseerimise ajal kommentaarides.

7. **Lihtne õppida –** Tööriista kasutusele võtmine on väga lihtne ning sellega saavad hakkama ka inimesed, kes pole sarnaste programmidega varem kokku puutunud. Esmakordselt sisse logides on olemas testprojekt, kus näidatakse tähtsamaid funktsioone ning nende funktsionaalsust. TestPad on mõeldud peamiselt

klaviatuuriga kasutamiseks, mis nõuab esialgu küll klahvikombinatsioonide õppimist, kuid edasine töö muutub väga kiireks ja efektiivseks. Kogu veebiliides on sobilik ka mobiilsetel seadmetel kasutamiseks.

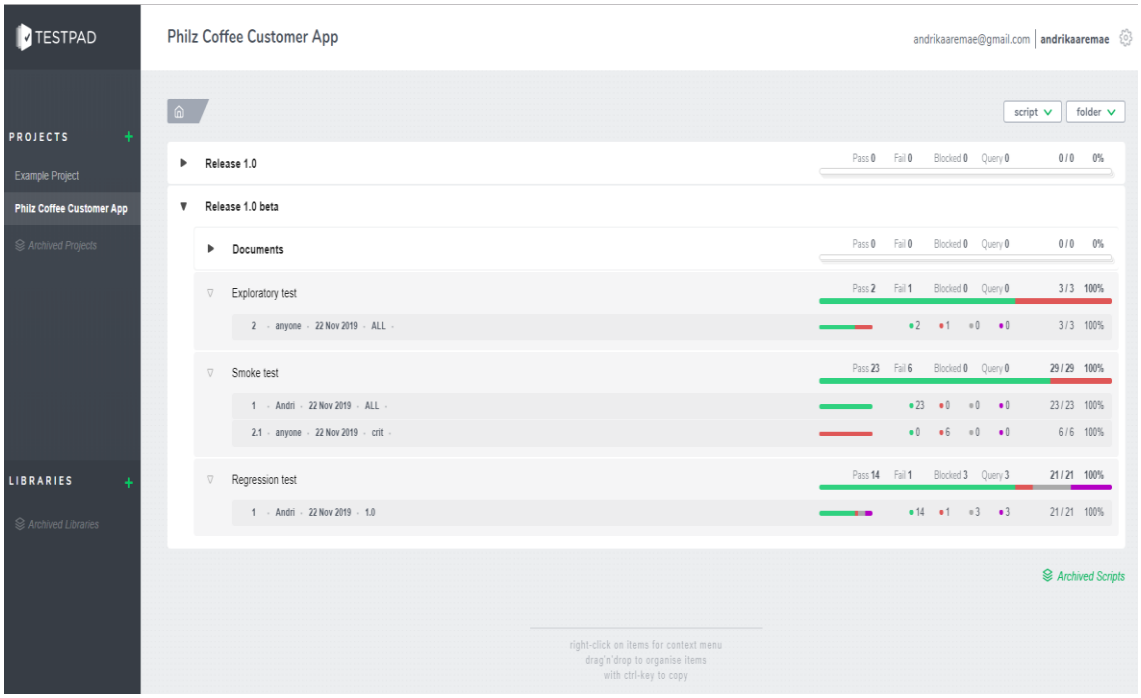
8. **Muudatuste jälgimine –** Muudatuste ajalugu säilitatakse ainult testjuhtumite ja testsessioonide läbimiste kohta. Samuti säilib kommentaarides vestluste ajalugu.

9. **Tööriistasisene suhtlemise võimalus –** TestPad võimaldab testidesse kirjutada lisamärkusi ning testjuhtumite ja testsessioonide läbimisel jätta kommentaare. Kommentaaridele vastamine ja otsene suhtlus kasutajate vahel pole võimalik.

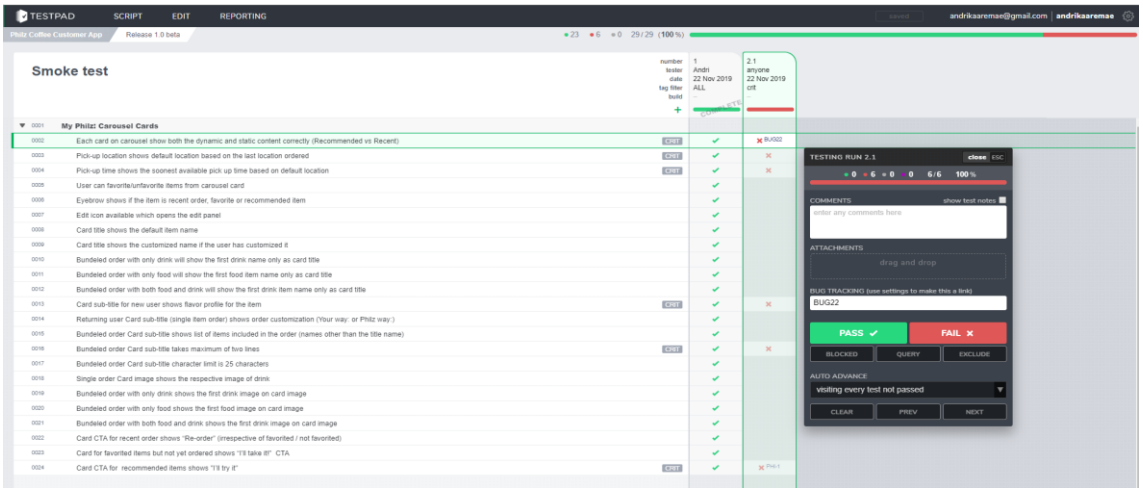
10. **Hind –** Käesolev testihaldusvahend on oma hinnatasemelt väga konkurentsivõimeline ning kuulub odavamasse hinnaklassi. Ostmine on võimalik nii paketina kui ka kasutajate arvu põhjal. Pakett kuni 25 liikmega maksab 249 dollarit kuus. Kasutajate arvu põhjal on hinnaks 15 dollarit kuus ühe inimese kohta.

11. **Integratsioon –** Tööriista saab integreerida sellise programmidega nagu Jira, GitHub ja PivotalTracker. Testjuhtumeid või testsessioone läbides on vea leidmisel võimalik see kohe üles märkida ja mugavalt valitud programmidesse raporteerida.

Joonistel 5 ja 6 on esitatud kuvatõmmised testihaldusvahendist TestPad.



Joonis 5. Tööriista TestPad avavaade



Joonis 6. Tööriista TestPad testide läbimise vaade

5.2 TestMonitor

TestMonitor on tasuline testihaldusvahend, mis on suunatud väikese ja keskmise suurusega ettevõtetele. Selle abil on võimalik hallata nii tarkvara nõudeid, riske, testjuhtumeid, testide tulemusi ning ka raporteeritud vigu. Tööriista kirjeldamisel toetub autor reaalsetele katsetustele ning koduleheküljel olevale informatsioonile [26].

1. **Testjuhtumite haldamine** – TestMonitor pakub võimalust siduda testjuhtumid tarkvarale esitatud nõuetega ning riskidega. Samuti on olemas võimalus

testjuhtumid vastavalt vajadusele mugavalt grupeerida. Igas testjuhtumis saab täpsemalt kirjeldada vajalikud läbimise sammud, ajakulu ning lisada selgitavaid pilte või faile. Testjuhtumite detailvaates on näha läbimiste ajalugu ja tulemused, kuid ei näidata tehtud muudatusi andmetes. Häirivaks on asjaolu, et iga testjuhtumi läbikukkumise korral sunnib tööriist sisestama selgitavaid pilte või faile. Tööriistal puudub võimalus testjuhtumeid tulemuste põhjal prioritiseerida.

2. **Uuriva testimise haldamine** – Puudub võimalus testsessioonide haldamiseks.

3. **Aruanded ja meetrika –** Meetrika ja statistikaga saab tööriist väga hästi hakkama. Kogu pakutav informatsioon on visuaalselt selge ning sobib hästi aruannetes kasutamiseks. Andmete põhjal koostatakse näiteks tarkvara nõuete kokkuvõtteid vastavalt nendega seotud testide tulemustele ja kategooriatele. Lisaks on veel riskide, testjuhtumite, testihalduse, testide tulemuste ning raporteeritud vigade kohta väga erinevaid graafikuid ja kokkuvõtteid. Eraldiseisvad dokumentide koostamise ja haldamise võimalust tööriist siiski ei paku.

4. **Projektide haldamine –** Tööriist lubab kasutajatel luua mitu eraldiseisvat projekti ning nende vahel navigeerimine on lihtne ja kiire.

5. **Tarkvara versioon –** Versioonihaldus on olemas ning testjuhtumeid saab lisada erinevatesse versioonidesse ning neid hiljem sorteerida. Samuti on võimalik kogu statistika ja meetrika koguda erinevate versioonide näitel.

6. **Seadmete haldamine –** Tööriist ei oma sisseehitatud seadmete ja operatsioonisüsteemide haldamise võimalust, kuid vajadusel on võimalik kogu info kirjutada kommentaaridena. Seda infot aga ei võeta arvesse statistika ning meetrika koostamisel.

7. **Lihtne õppida –** TestMonitor'i esmakordsel kasutamisel kuvatakse lühike õpetus, mis teeb kasutajale selgeks põhifunktsionaalsuse ning suunab vajadusel soovitud lehekülgedele. Samuti on kohe olemas testprojekt, mis kasutab ära kõik tööriista võimalused ning aitab funktsionaalsust kiiremini selgeks saada. Lisaks on tööriista all nurgas alati olemas vestlusroboti kasutamise võimalus. Kogu

kasutajaliides on arusaadav ning loogilise paigutusega. Sobib kasutamiseks ka mobiilsetel seadmetel.

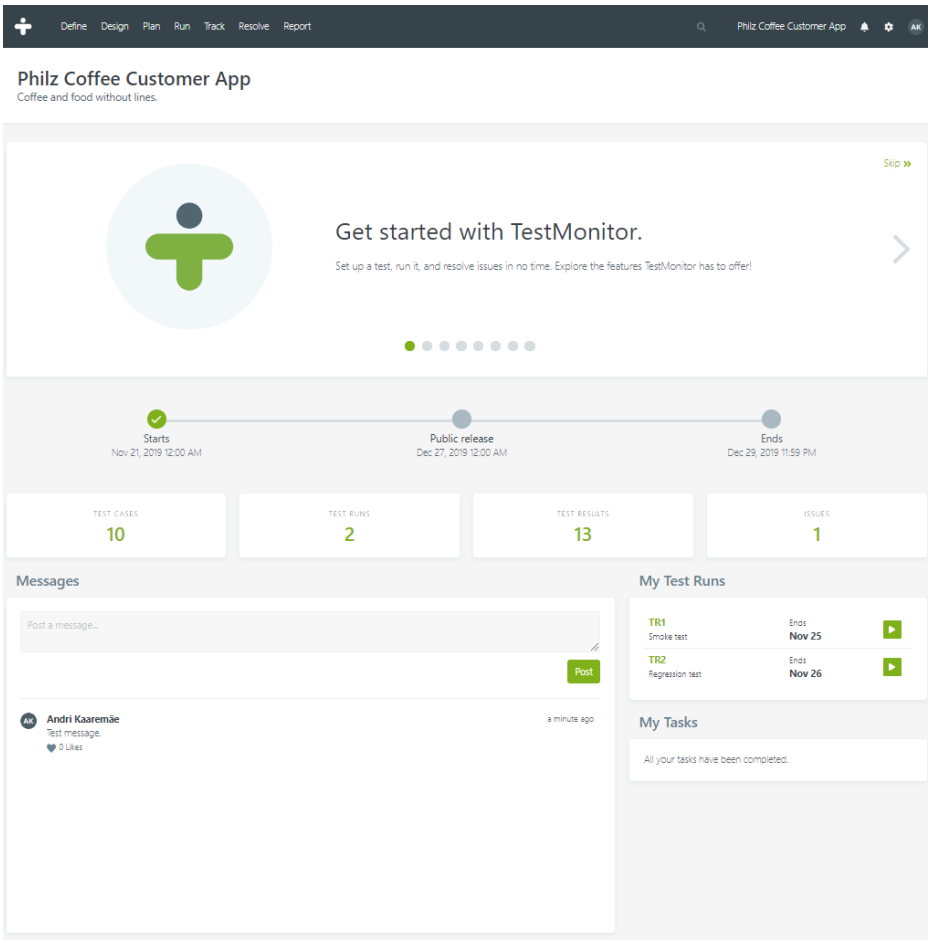
8. **Muudatuste jälgimine –** Ajaloo jälgimine on võimalik raporteeritud vigade, testjuhtumite läbimise, testjuhtumite tulemuste, nõuete muudatuste ja riskide muudatuste kohta. Samuti säilib kommentaarides vestluste ajalugu.

9. **Tööriistasisene suhtlemise võimalus –** Kommenteerimise võimalus on läbitud testjuhtumite ning raporteeritud vigade juures. Lisatud kommentaarid on hästi nähtavad. Üldiselt on tööriista sisene suhtlus siiski minimaalne ning ei sobi intensiivsemaks kasutamiseks.

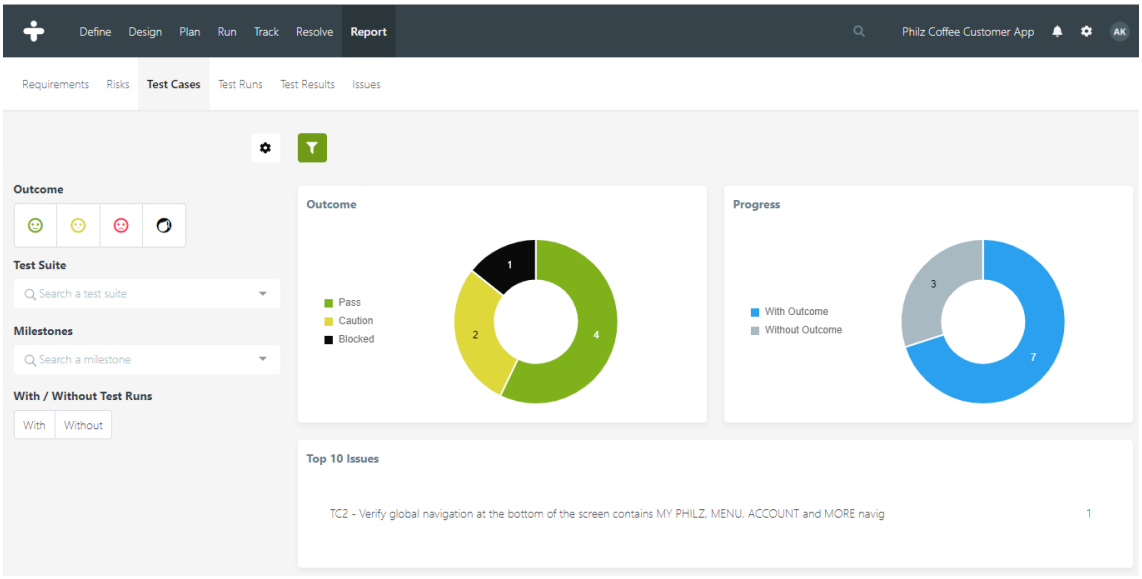
10. **Hind –** Käesolev testihaldusvahend on oma hinnatasemelt väga konkurentsivõimeline ning kuulub odavamasse hinnaklassi. Pakutakse kahte erinevat tellimuse võimalust. Aastase paketi korral on hind ühe kasutaja kohta 19 dollarit kuus, igakuise paketi korral 29 dollarit kuus.

11. **Integratsioon –** Tööriista saab integreerida sellise programmidega nagu Jira, Slack, Microsoft Azure DevOps, TOPdesk ja Asana. Samuti on peagi tulemas integratsioon GitHub'iga.

Joonistel 7 ja 8 on esitatud kuvatõmmised testihaldusvahendist TestMonitor.



Joonis 7. Tööriista TestMonitor avavaade



Joonis 8. Tööriista TestMonitor statistika ja meetrika

5.3 PractiTest

PractiTest on tasuline testihaldusvahend, mis on mõeldud professionaalsel tasemel testimisprotsessi haldamiseks See pakub väga palju erinevaid võimalusi tarkvara nõuete, testide, raporteeritud vigade ning tulemuste sorteerimiseks ja filtreerimiseks. Tööriista kirjeldamisel toetub autor reaalsetele katsetustele ning koduleheküljel olevale informatsioonile [27].

1. **Testjuhtumite haldamine –** Antud testihaldusvahendis saab hallata ja omavahel siduda tarkvara nõuded ning testjuhtumid. Tarkvara nõuded saab mugavalt kas näiteks Jira või Excel'i kaudu ka importida. Traditsioonilist testjuhtumit koostades saab paika panna läbimiseks vajalikud sammud, eeltingimused, oodatava tulemuse, filtrid, kommentaarid ning ise loodud tingimused. Samuti on võimalik lisada informatiivseid pilte ja faile. Testjuhtumeid saab laialdaselt seadistada ja grupeerida. See muudab tööriista sobilikuks väga erinevatele kasutajatele, sest igaüks saab luua enda jaoks sobivad lahendused. Testjuhtumite läbimine on mugav ning kiire. Lisaks tulemustele saab teada ka igale testile kuluva aja, mille põhjal saab edasist testimist planeerida. Vigade leidmise korral on need võimalik kohe tööriista siseselt ka raporteerida või juba olemasoleva raporteeritud veaga siduda.

2. **Uuriva testimise haldamine** – Tööriist võimaldab ka uurivat testimist. Selleks on teste luues võimalik valida kas tegemist on traditsioonilise testjuhtumiga või testsessiooniga. Testsessiooni koostades saab anda infot selle kohta, millele peaks töö käigus keskenduma. Samuti saab ise luua uusi tekstilahtreid, et vajalikku infot edastada. On võimalik lisada selgitavaid pilte ning faile. Testsessiooni läbides saab üles märkida tekkinud küsimusi ning mõtteid. Samuti mõõdetakse iga testsessiooni ajakulu. Vigade leidmisel on need võimalik kohe raporteerida või juba olemasoleva raporteeritud veaga siduda.

3. **Aruanded ja meetrika –** Tööriist pakub võimalust kogutud andmete ja tulemuste põhjal luua erinevaid graafikuid ning tabeleid. Visuaalseid kokkuvõtteid saab teha üksikute testide tulemusi, testide kogumike tulemusi, tarkvara versioone, tarkvara nõudeid ning palju muud kasutades. Otse testihaldusvahendis graafikuid ja tabeleid siiski ei looda. Kasutaja poolt valitud tingimuste põhjal luuakse failid,

mida on võimalik eksportida kas PDF (*portable document format*) või Excel'i dokumendina. Kuna valikuid ja sorteerimise võimalusi on väga palju, siis nõuab meetrika koostamine rohkem teadmisi tööriista toimimisest.

4. **Projektide haldamine –** On võimalik luua mitu erinevat projekti ning nende vahel navigeerimine on lihtne ning kiire.

5. **Tarkvara versioon –** Tööriist pakub versioonihalduse võimalust. Testjuhtumeid ja testsessioone saab versioonide põhjal grupeerida ja sorteerida. Samuti antakse visuaalne ülevaade erinevate versioonide läbimise tulemuste kohta.

6. **Seadmete haldamine –** Sisseehitatud seadmete ja operatsioonisüsteemide haldamise võimalust ei pakuta. Siiski on igas testjuhtumis ja testsessioonis võimalik ise vajalikud andmed üles märkida. Tööriist ei kasuta neid andmeid aga meetrika koostamisel.

7. **Lihtne õppida –** PractiTest on suunatud suurtele ettevõtetele ning pakub väga laialdast isikupärastamise ja konfigureerimise võimalust. Seetõttu nõuab tööriista esmane tundmaõppimine tavapärasest rohkem aega. Positiivse küljena on koduleheküljel olemas mitmeid õpetusvideosid ja artikleid. Esmakordsel testihaldusvahendi kasutamisel ei pakuta näidisprojekti, kuid on olemas interaktiivne õpetus, mis näitab ära olulisema funktsionaalsuse. Samuti on veebilehe all nurgas alati olemas võimalus vestlusakna kaudu ettevõtte PractiTest töötajatelt reaalajas abi küsida. Kogu tööriista põhilise funktsionaalsuse tundmaõppimine käib kiiresti, kuid erinevate filtrite, integratsioonide, meetrika ja lisavõimaluste selgeks tegemine nõuab rohkem aega.

8. **Muudatuste jälgimine –** Tööriist salvestab muudatused kogu projekti raames. Näiteks on võimalik näha kõikide testjuhtumite ja testsessioonide loomise ja muudatuste aegasid. Samuti näidatakse testide viimase läbimise aega, tulemusi ning ajakulu.

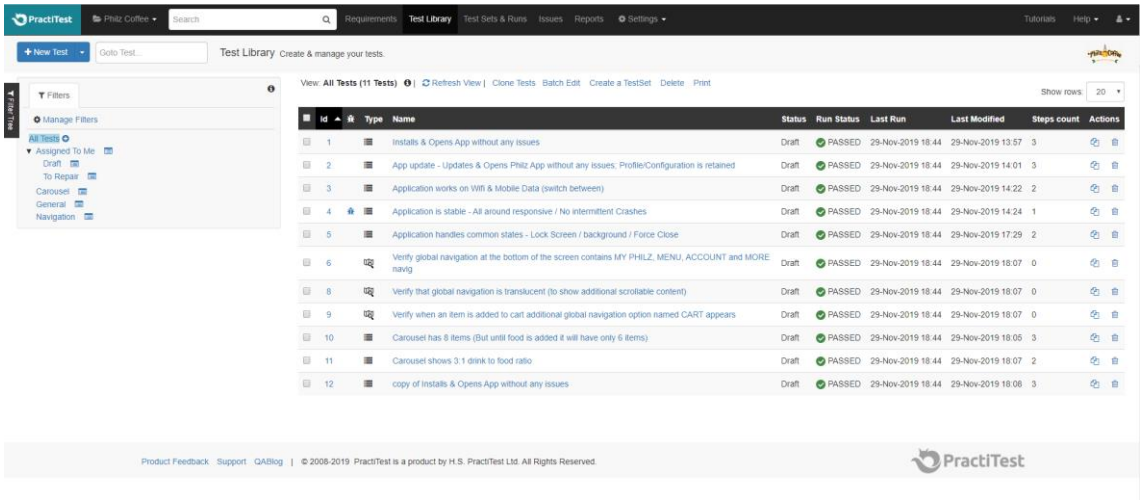
9. **Tööriistasisene suhtlemise võimalus –** Suhtlus on võimalik vaid testjuhtumite ja raporteeritud vigade kommentaarides. Tööriist ei paku kasutajate vahelist otsest suhtluse võimalust ning seetõttu ei sobi see ka intensiivsemaks aruteluks. Suurem

rõhk on pandud testimisprotsessi muudatuste jälgimisele ja testimise jaoks vajaliku info jagamisele.

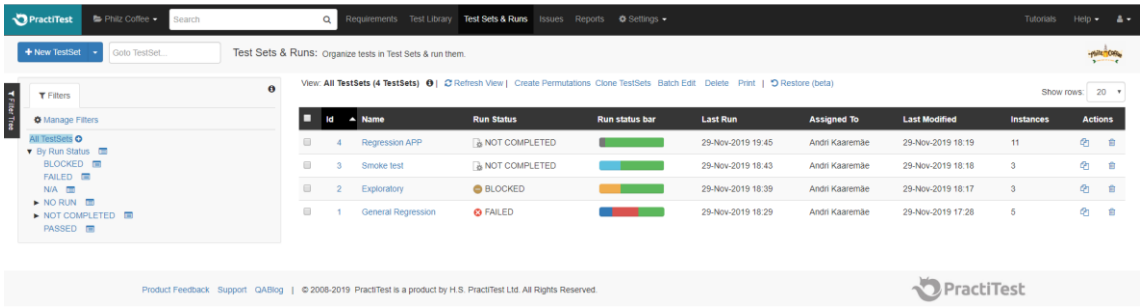
10. **Hind** – PractiTest on oma hinnatasemelt natukene kallim kui eelnevad kaks testihaldusvahendit. Seevastu pakutakse aga rohkem funktsionaalsust ning isikupärastamise võimalust. Olemas on nii uuriva testimise kui ka meetrika ja statistika kasutamise võimalus. Ühe inimese kohta on kuutasu 39 dollarit.

11. **Integratsioon** – Tööriista on võimalik väga mugavalt integreerida veahaldusvahendiga Jira. Testimise käigus leitud probleemid saab vaid mõne sammuga raporteerida ka sealses keskkonnas. Samuti on võimalik erinevatest failidest tarkvara nõudeid ja testsessioone importida. Integreerimise võimalus on olemas väga paljude erinevate tööriistadega, milleks on näiteks EggPlant, Pivotal Tracker, RedMine, Bugzilla, GitHub ja Slack.

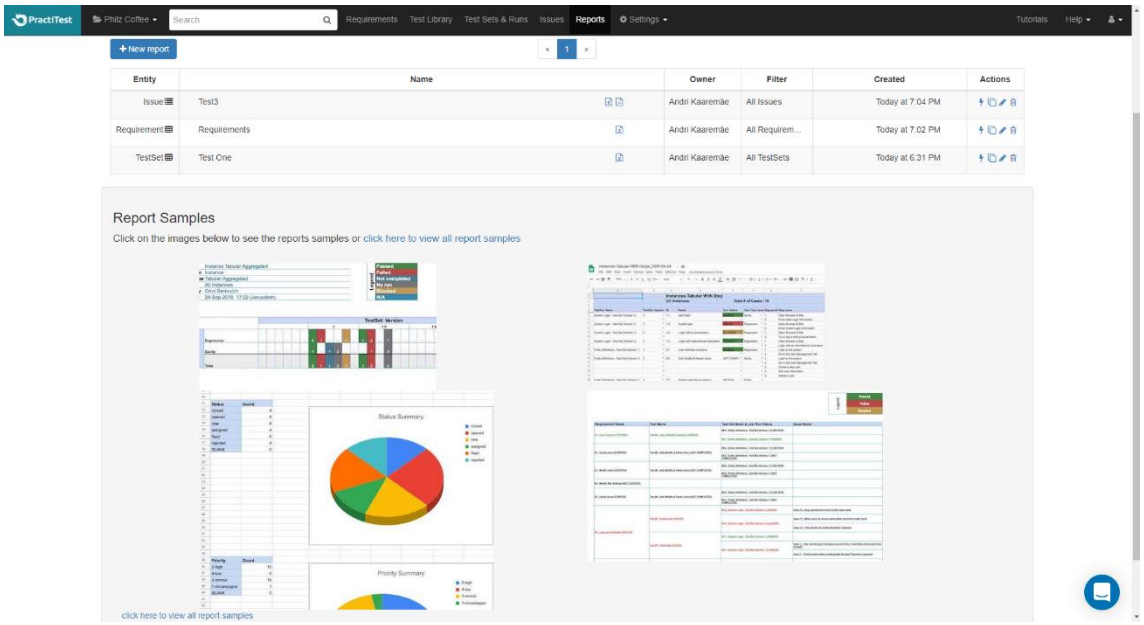
Joonistel 9, 10 ja 11 on esitatud kuvatõmmised testihaldusvahendist PractiTest.



Joonis 9. Tööriista PractiTest testide üldine vaade



Joonis 10. Tööriista PractiTest testide läbimiste kokkuvõte



Joonis 11. Tööriista PractiTest statistika ja meetrika vaade

5.4 QTest Manager

QTest Manager võimaldab mugavalt hallata, grupeerida ja sorteerida testjuhtumeid, tarkvara nõudeid, raporteeritud vigasid, statistikat ning meetrikat. Tööriista kirjeldamisel toetub autor reaalsetele katsetustele ning koduleheküljel olevale informatsioonile [28].

1. **Testjuhtumite haldamine** – QTest Manager on oma kasutajaliidese ja funktsionaalsuse poolest väga sarnane testihaldusvahendiga PractiTest. Testjuhtumid saab mugavalt siduda tarkvarale esitatud nõuetega. Tarkvara nõuded on võimalik Excel'i kaudu importida ning vajadusel ka eksportida. Tarkvara nõuete sisestamisel saab lasta tööriistal nende põhjal automaatselt koostada testjuhtumid. Testjuhtumeid saab lihtsasti kaustadesse grupeerida ning nende otsimisel on võimalik kasutada palju erinevaid filtreid. Testjuhtumeid luues saab lisada kirjelduse, läbimiseks vajalikud sammud, testimise tüübi, seotud tarkvara nõuded, pildid, failid ning kommentaarid. Muudatuste tegemisel salvestatakse ka nende ajalugu. Testjuhtumite läbimist on võimalik grupeerida lähtudes tarkvara versioonist. Testimisprotsessis saab vigade leidmisel need kohe tööriista siseselt või mõne integratsiooni kaudu ka raporteerida. Raporteeritud vigade haldamise lihtsustamiseks pakub ettevõte lisaks veel tööriista qTest Insights.

2. **Uuriva testimise haldamine** – QTest Manager on oma olemuselt suunatud traditsioonilisele testjuhtumitega testimisele. Vaatamata selle on võimalik testjuhtumite loomise kasutajaliidest ära kasutada testsessioonide loomiseks. Testsessiooni loomisel saab lisada kirjelduse, fookuse, eeltingimused, tarkvara nõuded, pildid, failid. Erinevus testjuhtumitest tuleb välja testimisprotsessis. Paika pandud sammude asemel on võimalik märkmete vaates lisada leitud probleemid ning tekkinud küsimused. Kõik testsessioonide tulemused ning märkmed salvestatakse ning nende põhjal luuakse ka meetrika. Hilisem testsessioonide ülevaade on aga tülikas. QTest pakub uuriva testimise jaoks juurde ostetavat veebirakendust qTest Explorer. QTest Explorer võimaldab salvestada ekraanipilti ning samal ajal teha märkmeid ning joonistusi. Testsessiooni lõppedes on võimalik lisatud andmete põhjal kohe ka vead raporteerida. Antud uurimistöö käigus testis autor siiski ainult tööriista qTest Manager, kuna qTest Explorer ei võimalda mugavat mobiilsetel seadmetel testimist.

3. **Aruanded ja meetrika –** Aruannete ja meetrika koostamine on võimalik testide haldamise, tarkvara nõuete, raporteeritud vigade ning testide tulemuste kohta. Tööriist koostab väga erinevaid ning informatiivseid jooniseid, mis on arusaadavad ka valdkonnaga vähem kokkupuutuvatele inimestele. Jooniste, graafikute ja tabelite koostamisel on võimalik ise rakendada mitmeid filtreid nagu näiteks testide tulemused ja ajaperiood, millal teste läbiti.

4. **Projektide haldamine –** Tööriist lubab kasutajatel luua mitu eraldiseisvat projekti ning nende vahel navigeerimine on lihtne ja kiire.

5. **Tarkvara versioon –** Tööriist võimaldab mugavat versioonihaldust. Testid saab koondada ning läbida versioonide põhiselt. Samuti arvestatakse statistika ja meetrika koostamisel tarkvara versioonidega.

6. **Seadmete haldamine –** Tööriist ei oma sisseehitatud seadmete ja operatsioonisüsteemide haldamise võimalust, kuid vajadusel on võimalik kogu info kirjutada kommentaaridena. Seda infot aga ei võeta arvesse statistika ning meetrika koostamisel.

7. **Lihtne õppida –** QTest Manager on oma kasutajaliidese poolest väga sarnane tööriistaga PractiTest. Vaatamata sellele nõuab testihaldusvahendi

tundmaõppimine aega, sest pakutakse väga mitmekülgseid sorteerimise ja filtreerimise võimalusi. Olemas on ka testprojekt, kus on ülevaatlikult näidatud, kuidas versioone, kaustasid ning faile struktureerida. Koduleheküljel pakutakse mitmeid tutvustavaid ning ülevaatlikke videosid. Kui kasutajaliides ja navigeerimine on selge, siis edasine töötamine on kiire ning mugav.

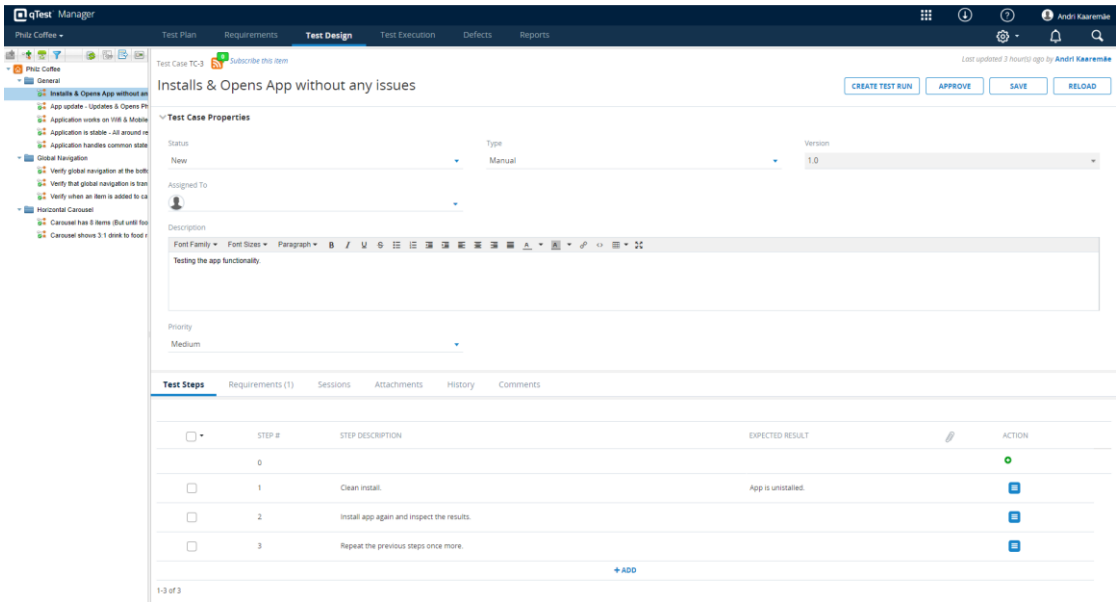
8. **Muudatuste jälgimine** – Ajalugu säilitatakse tarkvara nõuete muudatuste, testide muudatuste, testide läbimiste ning raporteeritud vigade kohta. Samuti säilib kommentaaride ajalugu.

9. **Tööriistasisene suhtlemise võimalus** – Tööriista siseselt on võimalik sisestada kommentaare ning märkmeid tarkvara nõuete, testide, raporteeritud vigade ning testide tulemuste kohta. Kasutajate vaheline otsene suhtluse võimalus puudub.

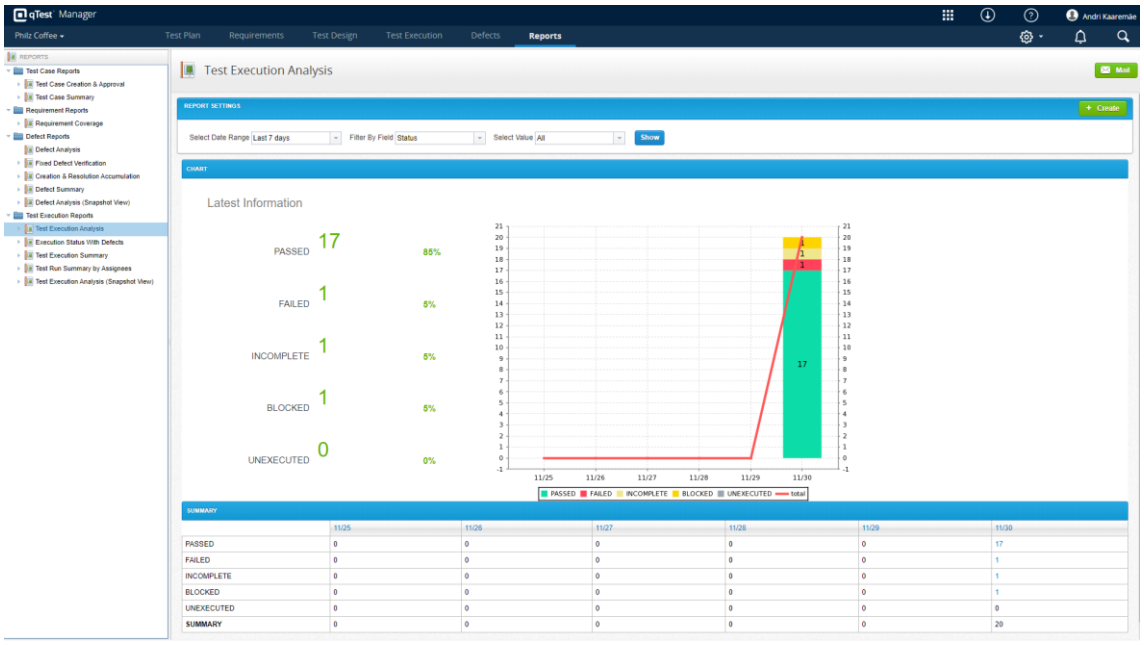
10. **Hind** – Koduleheküljel puudub informatsioon hindade kohta. Pakkumise saamiseks tuleb ühendust võtta klienditeenindusega.

11. **Integratsioon** – Tööriista saab integreerida sellise programmidega nagu Jira, Bamboo, BDD for Jira, Katalon, Cucumber, Selenium ja paljud teised. Automaatne vigade raporteerimine keskkonda Jira on võimalik kohe testide läbimisel.

Joonistel 12 ja 13 on esitatud kuvatõmmised testihaldusvahendist PractiTest.



Joonis 12. Tööriista qTest Manager testjuhtumi detailvaade



Joonis 13. Tööriista qTest Manager statistika ja meetrika vaade

5.5 Testihaldusvahendite võrdlus

Kõik valitud tööriistad said suure osa neile esitatud nõuete täitmisega hakkama, kuid esines ka mõningaid puudujääke. Järgnevalt võrdleb autor katsetatud testihaldusvahendite vastavust igale esitatud nõudele. Nende kohta koguti infot nii kodulehekülgedelt kui ka reaalsete katsetustega. Autoril oli võimalik kasutada tööriistade tasuta prooviperioode, et kogu funktsionaalsust täpsemalt uurida.

1. **Testjuhtumite haldamine –** Testjuhtumite haldamisega sai kõige paremini hakkama PractiTest. Neid sai väga erinevalt sorteerida ning filtreerida. Eeliseks teiste tööriistade ees oli testjuhtumite läbimise kasutajaliides, mis võimaldas näha ajakulu, sisestada kommentaare ja tulemusi iga eraldiseisva sammu jaoks ning vajadusel lisada pilte või raporteerida vigu. Testjuhtumite ja tarkvara nõuete omavahelised seosed olid arusaadavad ning kergesti leitavad. Paremuselt teiseks osutus tööriist qTest Manager, mis on visuaalse ja funktsionaalse poole pealt oli väga sarnane. TestMonitor ja TestPad said samuti nõude täitmisega hästi hakkama, kuid neil tööriistadel oli märgatavalt vähem erinevat funktsionaalsust. Kõige paremini said hakkama:

    1) PractiTest

2) QTest Manager

2. **Uuriva testimise haldamine –** Jällegi sai nõude täitmisega kõige paremini hakkama PractiTest. Tööriist omab sisse-ehitatud testsessioonide haldamise võimalust ning samuti on selleks olemas kogu vajalik funktsionaalsus. Samuti sai testsessioonide haldamisega hästi hakkama TestPad, mis oma puustruktuuriga võimaldab kasutajatel luua ise testsessioonide haldamiseks vajalik süsteem. qTest Manager ei omanud küll sisse-ehitatud testsessioonide haldamise võimalust, kuid vaatamata sellele oli võimalik testjuhtumite kasutajaliidest kasutada ära ka uuriva testimise jaoks. TestMonitor testsessioonide haldamist ei võimaldanud. Kõige paremini said hakkama:

    1) PractiTest

    2) TestPad

3. **Aruanded ja meetrika –** Vaieldamatult kõige paremini sai meetrika ja aruannete koostamisega hakkama qTest Manager, mis võimaldas luua graafikuid nii testide, tarkvara nõuete, raporteeritud vigade kui ka testide tulemuste kohta. Kogu meetrika oli vastavalt seatud tingimustele kiiresti tööriistas genereeritav ning visuaalselt arusaadav ning huvipakkuv. Graafikute ja statistika koostamise lihtsusega jäi silma ka TestMonitor, mis kuvas samuti kogu info tööriista siseselt. TestMonitor koostas meetrikat lisaks veel ka riskide kohta. PractiTest sai nõude täitmisega hästi hakkama, kuid nõudis rohkem õppimist ning filtreerimisvõimaluste tundmaõppimist. Samuti pakuti rohkem tabeleid ja arvulist statistikat kui jooniseid ja graafikuid. Kõike nõrgemaks osutus TestPad, mis andis vaid mõningase ülevaate testide läbimise kohta. Kõige paremini said hakkama:

    1) QTest Manager

    2) TestMonitor

4. **Projektide haldamine –** Kõik testitud tööriistad võimaldasid ka prooviversioonis luua mitu erinevat projekti. Nende vahel liikumine oli kiire ning arusaadav. Siinkohal on raske ühte tööriista teisele eelistada.

5. **Tarkvara versioon –** Versioonihaldus on olemas kõikides testitud tööriistades. PractiTest ja qTest Manager pakuvad aga teistest paremat ülevaadet erinevates versioonides olevate testide ning nende tulemuste kohta. PractiTest võimaldab andmeid vastavalt tarkvara versioonidele ning testija soovidele mugavaimalt sorteerida. Kõige paremini said hakkama:

    1) Practitest

    2) QTest Manager

6. **Seadmete haldamine –** Mitte ükski uuritud tööriistadest ei omanud sisse-ehitatud seadmete ja operatsioonisüsteemide haldamise võimalust. Kogu info sai lisada kommentaaride või märkustena, kuid seda ei võetud kasutusele meetrika ja statistika koostamisel. Kõige paremini oli seadmete haldamine võimalik tööriistas TestPad, kus testimisel sisestatud andmed olid hilisemal vaatlusel lihtsasti leitavad. Samuti sai sellega hästi hakkama TestMonitor, kus oli samuti võimalik iga testi läbimisel kirjutada kommentaarina vajalik lisainfo. PractiTest ja qTest Manager omasid küll sama funktsionaalsust, kuid sisestatud seadmete andmed olid hiljem raskemini leitavad. Kõige paremini said hakkama:

    1) TestPad

    2) TestMonitor

7. **Lihtne õppida –** TestPad on valikus olnud tööriistadest kõige lihtsama ning kiiremini õpitava kasutajaliidesega. See põhineb tavalisel puustruktuuril, millest saavad aru ka inimesed, kes igapäevaselt antud valdkonnaga kokku ei puutu. Kasutajasõbralikuks osutus ka TestMonitor, milles orienteerumine oli kiire ning arusaadav. Seevastu PractiTest ning qTest Manager olid suunatud rohkem professionaalsematele kasutajatele. Olid olemas küll õpetused ning näidisprojektid, kuid väga suur hulk erinevaid filtreerimise ja sorteerimise võimalusi muutis tööriista õppimise aeganõudvaks. Kõige paremini said hakkama:

    1) TestPad

    2) TestMonitor

8. **Muudatuste jälgimine –** Muudatuste ajalugu salvestati kõikides valikus olnud tööriistades. Kõige ülevaatlikumaks ja kasulikumaks osutusid PractiTest ning TestMonitor. Need tööriistad salvestasid vajaliku info kõige detailsemalt. Samuti oli andmete hilisem otsimine mugav ja kiire. Kõige paremini said hakkama:

    1) Practitest

    2) TestMonitor

9. **Tööriistasisene suhtlemise võimalus –** Mitte ükski katsetatud testihaldusvahenditest ei võimaldanud teiste kasutajatega otse suhelda. Kommentaaride ja märkmetega sai kõige paremini hakkama qTest Manager, mis võimaldas suhtlust nii tarkvara nõuete, testide ning testide tulemuste juures. TestMonitor võimaldas kommentaare ainult testjuhtumite läbimiste juures ning Practitest ainult testjuhtumite juures. Kõik kolm tööriista pakkusid ka võimalust kommenteerida raporteeritud vigasid. Minimaalseimat suhtlemise võimalust pakkus TestPad. Kõige paremini said hakkama:

    1) QTest Manager

    2) TestMonitor

10. **Hind –** TestPad, TestMonitor ning Practitest on kõik sarnases hinnaklassis pakutavad testihaldusvahendid. Seetõttu ei ole hind valikul kõige tähtsam kriteerium. QTest Manageri kohta ei õnnestunud autoril hinda välja selgitada.

11. **Integratsioon –** Kõik katsetatud tööriistad pakuvad suurel hulgal erinevaid integratsioonide võimalusi. Näiteks võimaldasid kõik testihaldusvahendid leitud vead mugavalt raporteerida veahalduskeskkonda Jira.

5.6 Soovitused tööriista kasutuselevõtuks ettevõttes FOB Solutions

Turul on väga palju erinevaid testihaldusvahendeid, mis on mõeldud kogu testimisprotsessi haldamiseks. Viimaste aastatega on nende arv märgatavalt kasvanud. Ettevõtte FOB Solutions vajadusi järgides tegi autor kindlaks tööriistale esitatud nõuded. Neid järgides oli võimalik testihaldusvahendeid omavahel võrrelda ning selgitada välja just nende vajaduste täitmiseks sobivaim.

Tööriistale esitatud nõudeid kogunes 11, millest kolm olulisemat olid testjuhtumite haldamine, uuriva testimise haldamine ning aruannete ja meetrika koostamine. Katsetamise käigus selgus, et TestMonitor uurivat testimist ei toeta. Samuti oli tähtis, et tööriistades oleks võimalik hallata nii mobiilirakenduste kui ka veebilehtede põhjal koostatud teste. Seda võimalust toetasid kõik uuritud testihaldusvahendid.

Sobivaima testihaldusvahendi väljaselgitamiseks on kõikidele esitatud nõuetele antud kaalud. Kaalude summeerimine võimaldab tööriistad seada paremusjärjestusse. Peatükis 4.5 tõi autor tööriistadele esitatud nõuete juures välja kaks sobivaimat. Esimese koha kaaluks on 3 punkti, teise koha kaaluks 2 punkti, ning kolmanda ja neljanda koha kaaludeks vastavalt 1 punkt. Kuna kolm esimest nõuet on ettevõtte FOB Solutions vajadusi silmad pidades eriti tähtsad, siis nende kaalud kahekordistatakse. Nõudeid, mille täitmisel olid tööriistad võrdväärsed, summeerimisel ei arvestata. Järgnevalt on välja toodud iga katsetatud testihaldusvahendi summeeritud kaalud:

1. Practitest: 6 + 6 + 2 + 3 + 1 + 1 + 3 + 1 = 23
2. QTest Manager: 4 + 2 + 6 + 2 + 1 + 1 + 1 + 3 = 20
3. TestPad: 2 + 4 + 2 + 1 + 3 + 3 + 1 + 1 = 17
4. TestMonitor: 2 + 2 + 4 + 1 + 2 + 2 + 2 + 2 = 17

Vastavalt tööriistadele esitatud nõuete kaaludele selgub, et ettevõttele FOB Solutions sobivaimaks testihaldusvahendiks on PractiTest. PractiTest nõuab kasutajatelt küll natuke rohkem õppimisaega, kuid see-eest võimaldab see väga heal tasemel testjuhtumite haldamist. Ainsana katsetatud tööriistadest pakub PractiTest ka sisse-ehitatud testsessioonide läbiviimise võimalust. Lisaks on heal tasemel nii statistika kui ka jooniste loomine.

QTest Manager ning TestPad said samuti kõikide nõuete täitmisega hakkama ning sobiksid alternatiivina tööriistale PractiTest. Suurimaks puuduseks on sisse-ehitatud testsessioonide haldamise puudumine, kuid tööriistad võimaldavad neid läbi viia muid lahendusi kasutades. Mõningate nõuete täitmisega, said antud tööriistad isegi paremini hakkama. QTest Manager'i kõige tugevamaks küljeks võib pidada erineva statistika ning meetrika töötlemist ja visualiseerimist. TestPad paistis silma oma lihtsusega ning võimalusega kõike ettevõtte vajaduste järgi organiseerida ja struktureerida.

Autor ei soovita kasutusele võtta testihaldusvahendit TestMonitor. Tööriist on küll kiiresti õpitav, rohke funktsionaalsusega ning visuaalselt ilus, kuid määravaks asjaoluks on testsessioonide toe puudumine.

6 Kokkuvõte

Antud bakalaureusetöö eesmärgiks oli analüüsida, katsetada ning omavahel võrrelda erinevaid tarkvarasüsteemi testimise testihaldusvahendeid eesmärgiga leida tööriist, mis lihtsustaks ettevõtte FOB Solutions tööprotsesse.

Töö esimeses pooles andis autor ülevaate tarkvarasüsteemi testimisest. Täpsemalt kirjeldati funktsionaalset testimist, regressioontestimist ja suitsutestimist. Samuti kirjeldati testimisprotsessi elutsüklit. Kogutud teadmiste ning varasemate kogemuste põhjal pani autor kirja 11 nõuet, millest lähtudes valiti välja neli testihaldusvahendit.

Põhjalik analüüs tehti nelja erineva tööriista kohta, milleks olid TestPad, TestMonitor, PractiTest ning qTest Manager. Autor tõi välja kõikide testihaldusvahendite tugevad ja nõrgad küljed. Katsetatud testihaldusvahenditest vastas kõikidele nõuetele kolm. Ettevõtte FOB Solutions vajadusi arvesse võttes ei sobinud ainsana tööriist TestMonitor, kuna sellel puudus testsessioonide haldamise võimalus. Esitatud tingimused täitsid edukalt nii Practitest, qTest Manager kui ka TestPad. Testihaldusvahendeid omavahel võrreldes selgus, et kõige paremini vastas nõuetele PractiTest.

Bakalaureusetöö käigus õnnestus leida kolm sobilikku testihaldusvahendit tarkvarasüsteemi testimise lihtsustamiseks ettevõttes FOB Solutions. Autor soovitab ettevõttel antud tööriistasid reaalsete projektide testimisel katsetada ning sobivusel ka kasutusele võtta.

Samuti on võimalik käesolevat lõputööd edasi arendada. Ettevõtte FOB Solutions testimisprotsessi elutsükli lihtsustamiseks on võimalik uurida ning võrrelda erinevaid dokumentide halduseks mõeldud tööriistasid, mis oleksid integreeritavad välja valitud testihaldusvahendiga.

Kasutatud kirjandus

[1] TestDevLab blog. *4 Reasons Why Software Testing is Important*. https://www.testdevlab.com/blog/2018/07/importance-of-software-testing/ (10.30.2019)

[2] A. A. Sawant, P. H. Bari, P. M. Chawan. *Software Testing Techniques and Strategies*. 2012. https://www.ijera.com/papers/Vol2_issue3/FQ23980986.pdf (31.10.2019)

[3] G. J. Myers, C. Sandler and T. Badgett. *The Art of Software Testing*. Hoboken: John Wiley & Sons, 2012.

[4] P. C. Jorgensen. *Software Testing: A Craftman's Approach*. Third Edition. New York: Taylor & Francis Group, 2008.

[5] Guru99. *What is Functional Testing? Types & Examples.* https://www.guru99.com/functional-testing.html (31.10.2019)

[6] I. Hooda ja R. S. Chhillar. *Software Test Process, Testing Types and Techniques.* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.695.1299&rep=rep1&type=pdf (31.10.2019)

[7] Wikipedia.org. *Functional Testing*. https://en.wikipedia.org/wiki/Functional_testing (31.10.31)

[8] Software Testing Fundamentals. *Functional Testing*. http://softwaretestingfundamentals.com/functional-testing/ (10.31.2019)

[9] H. Agrawal, J. R. Horgan, E. W. Krauser ja S. A. London. *Incremental Regression Testing*. http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.23.4773&rep=rep1&type=pdf (11.11.2019)

[10] Try QA. *What is the cost of defects in software testing?* http://tryqa.com/what-is-the-cost-of-defects-in-software-testing/ (09.12.2019)

[11] G. Rothermel ja M. J. Harrold. *Analyzing regression test selection techniques*. http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.224.870&rep=rep1&type=pdf (07.11.2019)

[12] Katalon Blog. *What is Regression Testing? Definition, Tools & How to Get Started*. https://www.katalon.com/resources-center/blog/regression-testing/ (11.11.2019)

[13] G. Rothermel, R. H. Untch, C. Chu ja M. J. Harrold. *Prioritizing Test Cases For Regression Testing* http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.130.7069&rep=rep1&type=pdf (2019.11.07)

[14] J. Itkonen ja K. Rautiainen. *Exploratory testing: A multiple case study* https://ieeexplore.ieee.org/abstract/document/1541817 (22.11.2019)

[15] C. Kaner, J. Bach ja B. Pettichord. *Lessons Learned in Software Testing: A Context-Driven Approach*. New York: Wiley, 2002.

[16] TutorialsPoint. *Smoke Testing* https://www.tutorialspoint.com/software_testing_dictionary/smoke_testing.htm (18.11.2019)

[17] P. Itchapurapu. *What is Smoke Testing? How to do with EXAMPLES*. https://www.guru99.com/smoke-testing.html (18.11.2019)

[18] Wikipedia.org. *Smoke testing (software)*. [Võrgumaterjal]. https://en.wikipedia.org/wiki/Smoke_testing_(software) (18.11.2019)

[19] Software Testing Help. *What Is Software Testing Life Cycle (STLC)?* https://www.softwaretestinghelp.com/what-is-software-testing-life-cycle-stlc/ (18.11.2019)

[20] W. Afzal. *Metrics in Software Test Planning and Test*. http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.102.4321&rep=rep1&type=pdf (18.11.2019)

[21] ProfessionalQA. *Test Execution Cycle*. http://www.professionalqa.com/test-execution-cycle (19.11.2019)

[22] Wikpedia.org. *Test management* https://en.wikipedia.org/wiki/Test_management (19.11.2019)

[23] G. J. Myers. *The Art of Software Testing, Second Edition*. New Jersey: John Wiley & Sons, 2004.

[24] P. Rani. *What should be Included in a Test Summary Report?* TestLodge, https://blog.testlodge.com/test-summary-report/ (19.11.2019)

[25] TestPad. *It's like testing... only faster*. https://ontestpad.com/ (22.11.2019)

[26] TestMonitor. *Simple Online Test Management for Every Organization*. https://www.practitest.com/ (21.11.2019)

[27] PractiTest. *PractiTest Test Management Tool*. https://www.practitest.com/ (29.11.2019)

[28] Tricentis qTest. *The qTest Platform*. https://www.qasymphony.com/ (30.11.2019)