TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Automatiseeritud testide tasuvus EMT e-teeninduse näitel

Bakalaureusetöö

Üliõpilane: Sander Lepik
Üliõpilaskood: 123742IABB
Juhendaja: Dots. Gunnar Piho
Konsultant: Hannes Linno, MSc

Tallinn
2015

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.

---------- ----------

(*kuupäev*) (*allkiri*)

Annotatsioon

Käesoleva töö, „Automatiseeritud testide tasuvus EMT e-teeninduse näitel“, peamiseks eesmärgiks on analüüsida võrdlevalt manuaalselt ja automatiseeritult läbitavaid teste ning välja selgitada, kas automatiseerimise protsess on antud projekti raames olnud edukas.

Võrdlen oma töös manuaalset ja automatiseeritud testimist, toon välja mõlema voorused ja puudused ning kasutades välja töötatud automatiseeritud testide tasuvuse metoodikat, arvutan välja kas protsess oli edukas ka antud projekti raames.

Antud töö tulemustest selgus, et automatiseerimine osutus antud projektis edukaks. Tänu automatiseeritud testidele võidame süsteemi eeldatava eluea (1,5 aasta) jooksul testimisele kuluvas ajas 27%.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 37 leheküljel, 5 peatükki, 8 joonist, 1 tabel.

Abstract

The main aim of work, profitability of automated tests on the example of EMT self-service, was to comparably analyse manual and automated tests and to find out if test automation in the given project has been successful.

In the work, I will compare manual and automated testing, bring forth the virtues and drawbacks of both methodologies and by using cost-effectiveness methodology, I will calculate whether the process, in the given project, was successful.

The work showed that test automation in the current project was successful. By virtue of automated tests we save 27% of the time spent on testing during estimated life-span (1.5 years) of the system.

The thesis is in Estonian and contains 37 pages of text, 5 chapters, 8 figures, 1 table., etc.

Lühendite ja mõistete sõnastik

ISTQB	***International Software Testing Qualifications Board*** Tarkvaratestimise sertifikaat.
Regressioon-testimine	***Regression testing*** Tarkvaramuudatustest tulenev testimine, mille eesmärk on olla kindel, et muudatused süsteemi koodis pole teinud katki varem töötanud funktsionaalsust.
GUI/UI test	***Graphical user interface test*** Kasutajaliidese test.
Liferay	***Liferay Portal*** Arendusplatvorm.
Portlet	***Portlet*** Portlet on ühendatav kasutajaliidese komponent, mida hallatakse ja kuvatakse välja veebilehel.
Unit testimine	***Unit testing*** *Unit* testimine on tarkvaraarendus protsess, milles testitakse süsteemi kõige väiksemaid osasid – mida kutsutakse *unit*-iks.
Sprint	***Sprint*** Regulaarne ja teatud perioodi tagant korduv tööprotsess.
Investeeringu tasuvus	***ROI (Return on investment)*** Investeeringu tasuvus näitab projekti tulemuste efektiivsust.
Toodangu keskkond	***Live environment*** Keskkond, mida kasutavad kliendid.

Integratsiooni testimine	***Integration testing*** Süsteemi kõige väiksemate osade ehk *unit*-ite omavahelise suhtluse testimine. *Unit*-ite liidestuste kontrollimine.
Automatiseeritud testimine	***Automated software testing*** Automatiseeritud testimine on protsess, kus kasutatakse spetsiaalset tarkvara, et läbida enne süsteemi väljastamist toodangusse eeldefineeritud skriptide abil vajalikud testid.
Input/output testimine	***Input/output testing*** Sisend/väljund testimine. Sisestatakse teatud sisend ja kontrollitakse sisendist tulenevat väljundit.
Reliis	***Release*** Arenduse väljastamine toodangu keskkonda.

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

Kliendi jaoks on oluline, et temani jõudvad teenused oleks töötavad ja kvaliteetsed. Selleks, et arendatavasse tarkvarasse ei tekiks vigu, tuleb selle kvaliteeti süsteemis tehtud muudatuste tõttu testida – süsteem peab läbima teatud perioodi tagant regressioontestid. Antud töö raames tutvustan EMT e-teeninduse näitel testimist ning kirjeldan, kuidas manuaalselt läbitavad regressioontestid automatiseeriti - veel mõned kuud tagasi manuaalsel teel läbitav regressioontestimine asendus automatiseeritud testidega. Töö eesmärgiks on hinnata, kas antud automatiseerimise protsess osutus efektiivseks.

EMT e-teeninduse näol on tegemist süsteemiga, mida enam edasi ei arendata - võivad esineda vaid mõned pisimuudatused. Antud süsteemi vähene muudatuste arv tähendab, et regressioontestid on kuust-kuusse väga sarnased. Testide väike muutuvus andis võimaluse regressioon automatiseerida – hea võimalus muuta tavapärane aegavõttev manuaalne regressioon automatiseeritud testidega kiiresti läbitavaks.

1.1 Taust ja probleem

Uurin automatiseeritud testide tasuvust EMT e-teeninduse näitel. Hetkel on tegu süsteemiga, milles ei käi aktiivset arendamist, võivad tulla vaid mõned pisimuudatused. Minu eesmärk on uurida, e-teeninduse näitel, kas automatiseerimise protsess on olnud tulus ehk kas automatiseerimise protsess aitab võrreldes manuaalse testimisega ajas võita. Selle tööga saan vajalikud teadmised automatiseeritud testide tasuvusest, mida saan edaspidi rakendada oma igapäevases töös.

1.2 Ülesande püstitus

1. Selgitan, mis on tarkvara kvaliteet ning milleks on oluline, et tarkvara oleks kvaliteetne; kirjeldan testimist kui tarkvara kvaliteedi tagamise meetodit.

2. Selgitan manuaalsete ning automatiseeritud testide põhitõdesid ja võrdlen antud testimise meetodeid omavahel.

3. Esitan automatiseeritud testide tasuvuse arvutamiseks väljatöötatud metoodika ning rakendan seda EMT e-teeninduse näitel.

4. Saada teadmine, kas antud projekti raames osutus automatiseerimine aja vaates efektiivseks.

1.3 Metoodika

Seon eesmärgini jõudmiseks teooria testimise põhitõdedest ja olemusest ning oma kogemused automatiseeritud testide arendamisest ja regressioontestide läbimisest. Kirjeldan automatiseeritud testide tasuvuse arvutamiseks väljatöötatud metoodikat ning kasutades antud metoodikat selgitan välja, kas EMT e-teeninduse regressioontestide automatiseerimine osutus tulusaks. Juhul kui automatiseerimise protsess osutus edukaks, toon välja ka arvulise väärtuse, mis väljendab automatiseerimisega kaasnevat ajalist võitu antud projektis.

1.4 Ülevaade tööst

Töö esimene pool sisaldab ülevaadet tarkvara kvaliteedist ning selle tagamisest. Selgitan manuaalse ning automatiseeritud testimise põhitõdesid ja võrdlen neid omavahel – toon välja mõlema meetodi positiivsed ja negatiivsed küljed ning põhjendan neid. Töö teises pooles selgitan mis on EMT e-teenindus ning kirjeldan süsteemi tehnilist tausta. Esitan automatiseeritud testide tasuvuse arvutamise metoodika, millega abil selgitan välja ka tasuvuse antud projekti raames.

2. Tarkvara kvaliteet

Tänapäeval on igapäevaselt kasutuses süsteemid, mille komponendiks on tarkvara. Tarkvarast võib oleneda väga palju – meie töö, palk, või vahel isegi elu, seega on ülimalt oluline, et tarkvara töötaks veatult. Tarkvara töökindluse tagamiseks peab arendusprotsessi tulemuseks olema võimalikult kvaliteetne toode – toode, mis vastab talle esitatud nõuetele [4]. Selleks, et olla kvaliteedis kindel, vajab iga toode testimist.

Mis on testimine? Sellele võib vastata väga lihtsalt – kontrollimine kas kõik toimib nii, nagu peab [1]. Testimine on vajalik, kuna ükski inimene ei suuda oma tööd teha veatult, sealhulgas ka programmeerijad. Kuna eesmärk on välja anda töötavat ja kvaliteetset tarkvara, siis on väga vajalik tagada piisav kontroll, et tagada *live* keskkonna kvaliteet. Testimine peab olema pidev. Funktsionaalsusesse muudatusi tehes võib esineda ka süsteemi üldises toimimises muudatusi, mille tagajärjel võivad esineda vead. Sealt tuleneb vajadus kontrollida süsteemi kui terviku toimimist pidevalt iga teatud aja tagant.

2.1 Tarkvara arendusprotsessis esinevad vead

Vigade otsimisel tuleb arvesse võtta väga erinevaid faktoreid, millest võivad tuleneda tarkvaras esinevad vead. Alljärgnevas on allikale [20] tuginevalt loetletud enamesinevad vead ning näited, kuidas sellised vead tekivad.

- Viga süsteemi spetsifikatsioonis (rakenduse vajalike funktsionaalsuste ja mittefunktsionaalselt nõuete kirjeldamisel jäi vajalik nõue kirjeldamata)
- Valesti täidetud nõuded (kirjeldatud nõuetest saadi valesti aru ning programmeeriti valesti)
- Programmeerija tekitatud viga süsteemi disainimises (rakenduse arendaja tegi programmeerimisel vea)
- Süsteemi väärkasutamine (kasutaja nupuvajutuse peale hakkab leht laadima, aga laadimise ajal vajutab kasutaja veel 10 korda sama nuppu – tagajärjeks võib olla tehniline viga)
- Varasemate vigade parandamisel esinevad uued vead (vigase koodi parandamisel tekivad omakorda uued vead)
- Süsteemi keskkonnast tingitud vead (keskkond on ebastabiilne ja seetõttu süsteem ei toimi korrektselt)

Vigasid võib tekkida tarkvara arendusprotsessi kõikides faasides. Et täpsemalt kirjeldada tarkvaraarenduse erinevates faasides tehtud vigade mõju toote lõpptulemusele on alljärgnevas tabelis esitatud erinevad võimalikud stsenaariumid. Tabelis 1 on esitatud tarkvara arendusprotsessi etapid ja näidatud erinevad vigade tekke võimalused.

Tabel 1. Vigade mõju teatud arendusetappides

1.	Korrektsed nõuded	Korrektne disain	Programmeeritud vastavalt disainile	**Lõpp-produkt töötab, nagu on ette nähtud**	Vigadeta süsteem
2.	Korrektsed nõuded	Korrektne disain	**Programmeerimisel tehtud viga**	Lõpp-produktis esineb vigu	Süsteemi koodi tuleb parandada
3.	Korrektsed nõuded	**Valesti disainitud süsteem**	Programmeeritud vastavalt disainile	Lõpp-produktis esineb disainivigu	Süsteem tuleb ümber disainida
4.	**Nõuetes tehtud viga**	Disainitud vastavalt nõuetele	Programmeeritud vastavalt disainile	Lõpp-produkt on vale	Süsteemi nõuded tuleb ümber kirjutada

Kõige väiksemaid tagasiminekuid kogu protsessis põhjustab programmeerija poolt koodi kirjutamisel tehtud viga.

Süsteemi disainimisel tehtud viga aga osutub juba keerulisemaks ja kulukamaks. Vea parandamiseks peab süsteemi ümber disainima ja programmeerija peab ka oma koodi kohandama süsteemi disaini tehtud muutustega.

Kõige raskemini leitav ja ka kõige kulukam viga arendusprotsessis on nõuetes tehtud viga. Sellist viga ei pruugi avastada ei programmeerija ega ka testija, kuna vastavalt kirjeldatud nõuetele võib süsteem väga hästi töötada. Viga tuleb välja alles siis, kui toode jõuab tellijani, kes avastab, et soovis hoopis teist toodet. Sellisel juhul tuleb kogu arendusprotsess algusest läbi käia.

Selgub, et mida varasemas arendusprotsessi faasis viga esineb, seda kulukamaks see viga kokkuvõttes kujuneb. [19]

2.2 Regressioontestimine

Tarkvara muutes või parandades seal olevaid vigu me muudame olemasolevat süsteemi. Testimise eesmärk sellises olukorras oleks kontrollida, et tehtud muudatused poleks süsteemis muutnud midagi soovimatut. Sellisel eesmärgil testimist nimetatakse regressioontestimiseks. [1]

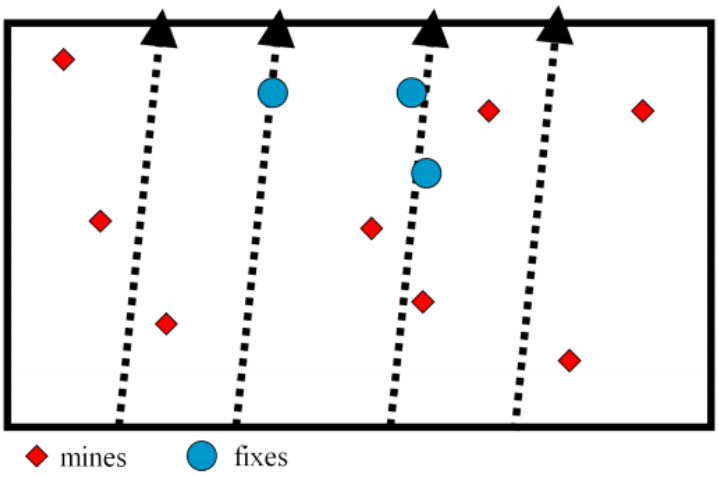
Regressioontestid võime kategoriseerida nende eesmärgi alusel alljärgnevalt [12]:

- Vigade regressioon – tehtud parandused on õigesti teostatud ning viga enam ei esine.
- Paranduste regressioon – tehtud parandused ei ole omakorda kahjustanud varasemalt tehtud parandusi.
- Funktsionaalne regressioon – uus kood ei ole kahjustanud varem töötanud funktsionaalsust.

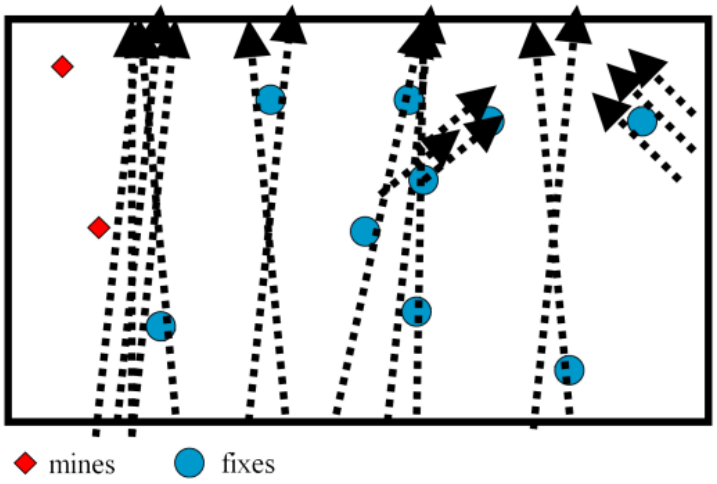
Regressioontestimine on vajalik, kuna tarkvara parandatakse ja uuendatakse pidevalt. Seega on peale muudatuste sisseviimist vaja kindel olla, et meie süsteem töötab selliselt, nagu ta peaks toimima. EMT e-teenindus vajab pidevat regressiooni, sest rakendused, millega ta liidestatud on, või millega ta andmeid vahetab, muutuvad pidevalt. Mida sagedamini regressioon läbitakse, seda kindlam saab olla süsteemi toimimises. Eesmärk võiks olla selline, et regressioon läbitaks igapäevaselt (tingimusel, et süsteemis iga päev ka midagi muudetakse), selline tegevusviis annaks pideva kindluse, et kõik tehtud arendused ja parandused töötavad ja ei ole varasemat töötavat funktsionaalsust kahjustanud. [12]

Regressioonis läbitavate kasutuslugude arv sõltub süsteemis teostatud muudatuste hulgast. Süsteem, mille arendus on peatatud, läbib iga kuu väga sarnased regressioontestid. Rakenduses, mis on aktiivselt arendamises, tekib pidevalt regressioontestimisel läbitavaid kasutuslugusid juurde, see tähendab, et regressioon kasvab pidevalt suuremaks. [2]

Regressioontestide koostamisel tuleks silmas pidada, et testid võiksid võimalikult palju varieeruda. Regressioonteste iga teatud perioodi tagant täpselt samade andmetega korrates leiame ainult teatud vead üles (vt joonis 1). Kasutades regressiooni läbiviimisel võimalikult erinevaid andmeid, suudaksime katta juba palju suurema ala – tehes samu teste, kuid kasutades pidevalt erinevaid andmeid testi tegemiseks, tekib suurem võimalus avastada vigu (vt joonis 2).



Joonis 1. Staatiliste andmetega testid [12]



Joonis 2. Dünaamiliste andmetega testid [12]

3. Manuaalne testimine *versus* automaattestimine

Iga arenduse läbinud tarkvara peab läbima testimise. Eesmärk on alati sama - väljastada toode kasutusse selliselt, et seal ei esine vigu. Paraku reaalsuses sellist olukorda siiski väga tihti ei juhtu. Ka parimaid manuaalse testimise meetodeid kasutades võivad tarkvarasse vead sisse jääda. Manuaalset testimist viib läbi inimene, kes istub arvuti taga ja üritab erinevaid kombinatsioone läbi proovides leida testitavas rakenduses vigu. Automaattestimine on aga meetod, milles on eelnevalt defineeritud skript, mille käivitamisel testib spetsiaalne tarkvara testitava rakenduse vastavust nõuetele. Kuid kumb kahest esitatud testimise viisist on efektiivsem ja millistes olukordades tuleks vastavaid meetodeid kasutada?

3.1 Manuaalne testimine

Manuaalne testimine on protsess, kus testija peab võtma endale lõppkasutaja rolli, testima rakenduse kõikvõimalikke funktsionaalsusi ja võimalusi ning saama kindluse, et see töötab selliselt, nagu ette on nähtud. [5]

Testimiseks nimetatakse ka seda, kui inimene, rakendusest midagi teadmata, proovib erinevaid pakutavaid võimalusi ja katsetab süsteemi. Ka selline testimine võib teatud mahus kasulik olla, kuna testija oleks sellisel juhul rakenduse kasutaja rollis, kes näeb esimest korda rakendust, aga peab samuti hakkama saama. Samuti võib taoline süsteemi kontrollimine kasuks tulla, kuna testija ei mõtle raamides. Tal pole ettekirjutatud testilugusid, mida läbida, ning seetõttu võib ta sattuda vigadele, mida teistviisi testides avastanud poleks.

Eelkirjeldatud testimise viis võib olla teatud mahus hea, kuid enamasti peab süsteemi toimimises kindel olemiseks ikkagi kasutama selgelt läbimõeldud testimise võtteid. Testimise meetodid võib laias laastus jagada kaheks: valge kasti meetod ja musta kasti meetod. [7]

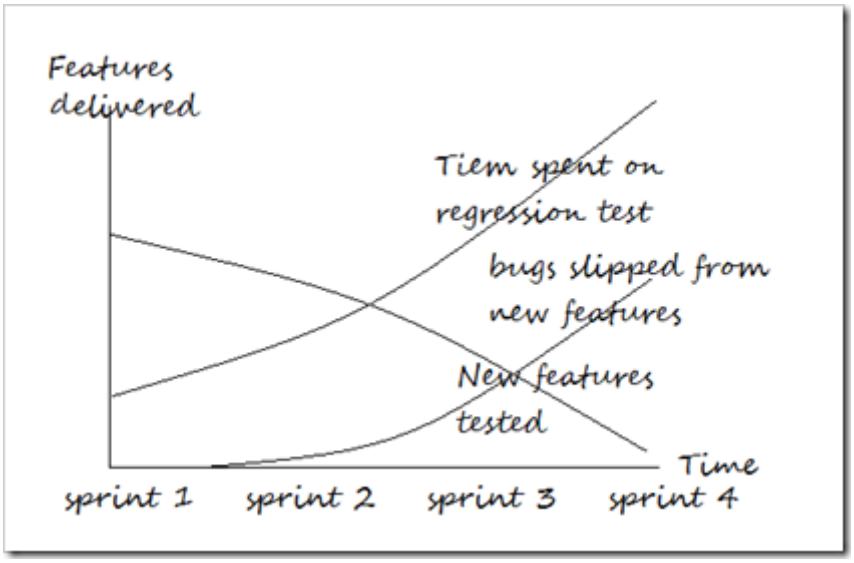
Musta kasti meetodi kasutamiseks peab vaatama programmi kui musta kasti. Testija eesmärk ei ole muretseda süsteemi sisemise käitumise ja struktuuri pärast, vaid keskenduda sellele, kas programm käitub vastavalt ettekirjutatud nõuetele. Sellisel lähenemisviisil tulenevad testandmed ainult rakenduse spetsifikatsioonist, testija ei pea olema teadlik rakenduse sisemistest struktuuridest ja arendatud koodi sisust. Musta kasti meetodit kutsutakse ka *input/output* testimiseks. Testija annab rakendusse sisendi ja kontrollib, pööramata tähelepanud

rakenduse sisemistele protsessidele, et rakendus tagastaks õiged väärtused [7]. Vastava olukorra näide – kasutaja muudab vormil oma kasutajaandmeid ja salvestab need. Peale isikuandmete muutmist peavad muudatused kajastuma ka andmete vormil.

Valge kasti meetod baseerub tugevalt ka rakenduse sisemistest struktuuridest arusaamisel. Testija peab aru saama rakenduse käitumisest ja kirjutama selle loogika põhjal valmis testlood. Siinkohal võib aga programmile kirjutatud spetsifikatsioon jääda tahaplaanile ning sellest tulenevalt ka mõned vead avastamata. Antud testimise meetodiga kontrollitakse just seda, kas kood käitub andmete töötlemisel õigesti – kas andmetega on tehtud kõik vajalikud tegevused, kas tingimuslaused on täidetud korrektselt jne. [7]

Manuaalne testimine on suhteliselt ajakulukas. Iga teatud perioodi tagant teostatakse regressioontestid, mis arenevas süsteemis kasvavad igakuiselt järjest mahukamaks. Regressioontestimise peale kulub seega järjest rohkem aega, kuna tuleb kindel olla kogu süsteemi korrektses toimimises. Regressioontestide tegemine, ehk kontrollimine kas hetkel juba kasutusel olev tarkvara endiselt töötab, ei tohiks aga võtta liiga palju aega, kuna vastasel juhul jääb uute arenduses olevate asjade testimise jaoks vähem aega. Uute arenduste mittetäielik testimine toob omakorda kaasa vigade suurenemise riski väljastavatel toodetel (vt joonis 3).

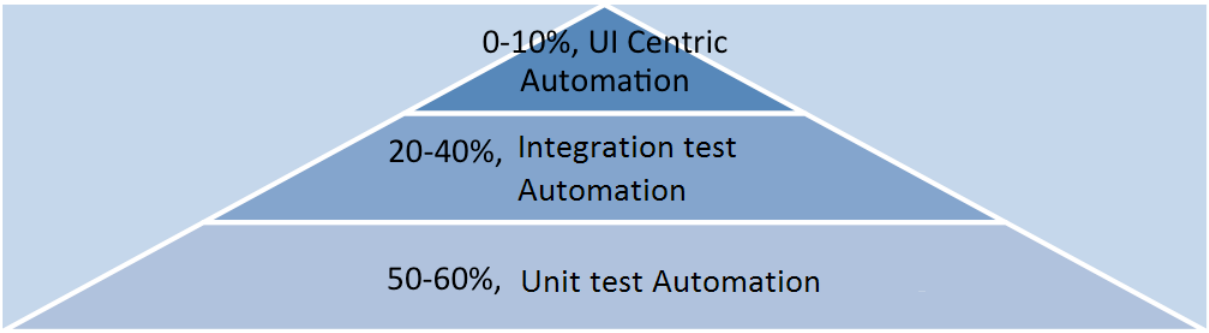
Et olukorda parandada ja kulutada vähem aega juba olemasoleva ja töötava funktsionaalsuse kontrollimiseks, oleks mõistlik teatud osa regressioonist automatiseerida. Selline tegevus annaks võimaluse teostada regressioone väiksema ajavahemiku tagant ning samas anda testijale rohkem aega tegeleda uute arenduses olevate funktsionaalsustega.



Joonis 3. Manuaalse testimise ajakulu ja vigade arv [8]

3.2 Automaattestimine

Automatiseeritud testimine on protsess, kus käivitatakse spetsiaalsel tarkvaral eeldefineeritud teste. Selline test koosneb järjestikustest tegevustest ning kontrollidest – automatiseeritud test peab kontrollima, kas testitav rakendus vastab oodatavatele tulemustele [10]. Regressioontestidele kuluva aja vähendamiseks ja manuaalse töö optimeerimiseks on kindlasti hea lahendus teste automatiseerida, kuid jääb küsimus, kui suures osas ja mis tasandil seda teha võiks? Seda, kui suures mahus tuleks testimist teatud tasandil teha, iseloomustab nn. ideaalne testimise kolmnurk, mis on esitatud joonisel 4.



Joonis 4. Ideaalne tarkvara automatiseerimise kolmnurk

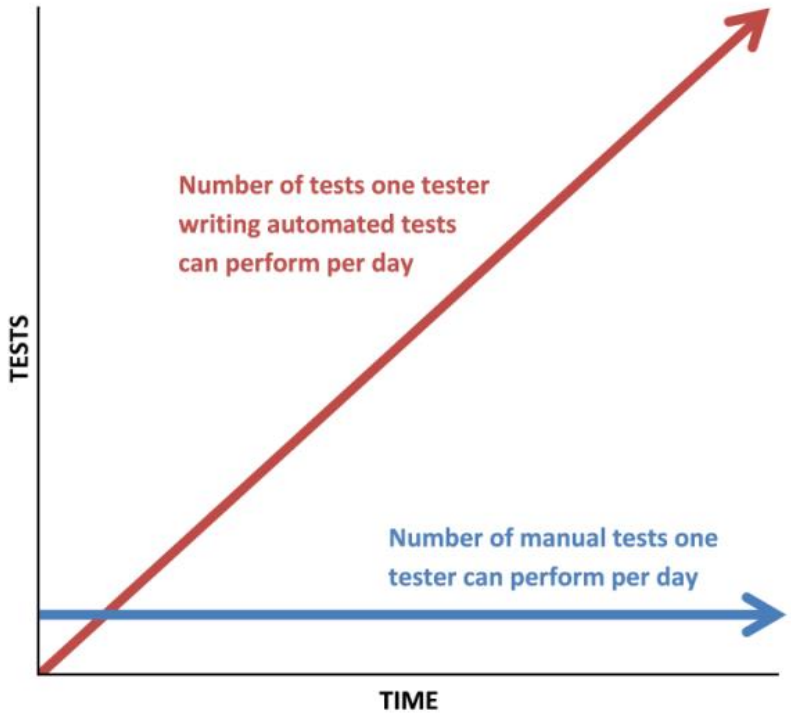
Joonis 4 näitab testimisprotsessi erinevaid tasandeid ning seda, kui suure osa kogu süsteemi testimisest võiks antud tasandil ära teha.

Kõige suurema osa automatiseeritud testidest tuleks teha nn. *Unit*- teste. *Unit* testide tegemine on tarkvaraarenduse protsessi osa, milles testitakse kõige väiksemaid rakenduse osasid [9]. *Unit*- teste kirjutavad tavaliselt arendajad ning kuigi testide kirjutamine võib algselt tunduda tüütu ja aeganõudev, siis nende tulu tuleb esile aja jooksul. *Unit* testid annavad kõige madalamal tasemel kindluse, et rakenduse kõik osad eraldiseisvalt töötavad. Selliste testide pluss on ka see, et testile kuluv aeg on väga väike ja saame kiiresti tagasisidet, kas süsteemi komponentidega on kõik korras.

Eraldiseisvate osade toimimine ei anna siiski kindlust, et rakendus tervikuna seega vigadeta töötab. On vaja testida ka rakenduse kõige väiksemate osade omavahelist suhtlemist. Sellist meetodit nimetatakse integratsioon-testimiseks [11]. Integratsioontestid baseeruvad *unit*-testidel, täpsemalt integratsioontestid tähendavad *unit*- testide omavahelist ühendamist. Integratsioonitestid aitavad välja tuua rakenduse osade liidestamisel tekkinud vead. Integratsioonitestide peamised puudused võrreldes *unit*- testidega on vigade raske väljaselgitamine (kuna eraldiseisvalt väiksed osad töötasid, aga peale osade ühendamist ei tööta, siis vea leidmine, miks rakendus enam ei tööta, võib olla aeganõudev ning tülikas) ning kulukas testide haldamine.

GUI testide ehk graafilise kasutajaliidese testide eesmärk on testida kasutajaliidest ning aru saada, kas rakenduse funktsionaalsus on korrektne. Kasutajaliidese testid enamasti täidavad eeldefineeritud tegevusi ning kontrollivad, et väljund vastaks eeldatavale tulemusele. Automatiseeritud *GUI* testid on võrreldes manuaalse testimisega tõhusamad, usaldusväärsemad ja mitte nii kulukad [13]. Testide automatiseerimine võib küll esialgu ajakulukas olla, kuid selle eelised ilmnevad teatud kasutusperioodi pärast.

Automatiseeritud testide üks väga suur boonus on välja toodud alljärgnevas (vt joonis 5). Manuaalselt teste tehes on inimesel selge piir ees, kui palju testilugusid jõuab ühe tööpäeva jooksul läbida. Automatiseeritud testidel aga sellist piiri ei ole – ööpäevas läbitavate testide arv on piisavalt suur, et see takistuseks ei tule.



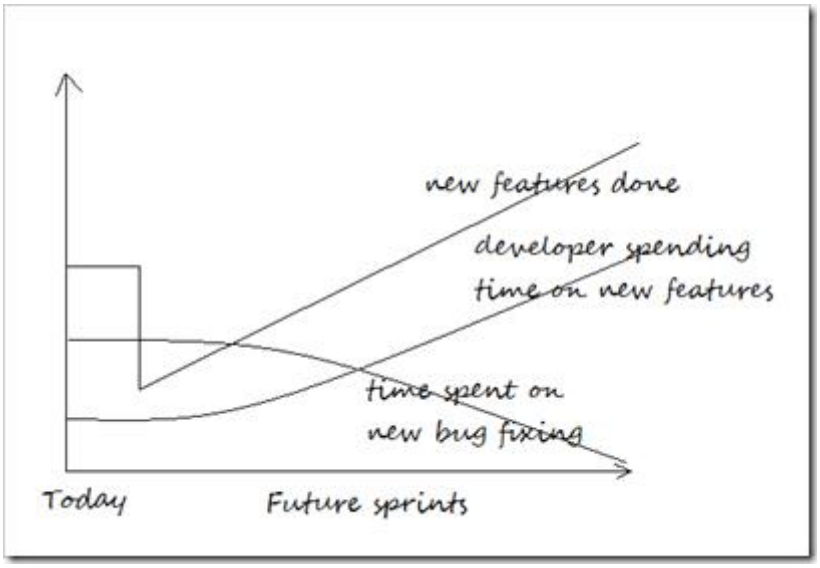
Joonis 5. Testide läbimise hulk x perioodi jooksul [14]

3.3 Automatiseeritud regressioon

Argumente, miks regressioonteste tasub automatiseerida leidub päris palju. Manuaalse regressiooni läbimiseks kuluv aeg on tavapäraselt päris suur, mis aga selle juures eriti halb on – testija ei saa tegeleda uue arenduses oleva funktsionaalsusega, vaid peab tegelema asjadega, mis juba peaksid töötama. Selline tegevus aga vähendab arenduse kiirust. Automatiseeritud testide arendamine võib olla ajakulukas, kuid pikemas perspektiivis osutub automatiseeritud testide olemasolu kindlasti tulusaks. Automaatse regressiooni kasuks räägib asjaolu, et regressioonile kuluv aeg väheneks märgatavalt, mis omakorda tähendaks, et regressioone saaks teha väiksema ajavahemiku tagant. Seeläbi saame olla kindlamad süsteemi vigadeta toimimises. Automatiseeritud testid annavad meile võimaluse neid oma soovi järgi teatud ajavahemiku tagant käima lasta ja saame kiiresti tagasiside, mis seisus testitav süsteem on. See on suur eelis manuaalse testimise ees, kuna mahukat süsteemi ei jõua testija regressiooni sees kindlasti mitu korda läbi testida, kuid koodis võib sageli toimuda veel peale regressiooni algust muudatusi. Automatiseeritud testidega tekib programmeerijatel kindlam tunne arendada, kuna tekkinud vead on kohe automatiseeritud testide genereeritud testraportitest näha.

Automatiseeritud testidel on ka negatiivseid aspekte. Esiteks testide arendamine on küllaltki aeganõudev ning valmis testide haldamine on samuti kulukas. Arenduses oleva süsteemi automatiseeritud *GUI* teste peab muutma suhteliselt tihti, kuna ka kõige väiksemad muudatused kasutajaliideses tähendavad muudatusi ka testi koodis [3]. Kokkuvõttes võib öelda, et regressiooni automatiseerimine tõstab arenduse kiirust, tagab kiire ja täpse tagasiside süsteemi hetkeolukorrast ning mis peamine, annab testijale vabaduse tegeleda uute funktsionaalsuste testimisega.

Väga hästi ilmestab automatiseerimist alljärgnevas olev joonis 6. Võttes vastu regressiooni automatiseerimise otsuse, võtab see alguses testija ajast päris suure osa ja teatud perioodiks ei ole uus funktsionaalsus prioriteet number üks. Kui aga automatiseeritud testid saavad kirjutatud, muutub graafik oluliselt – nii testija kui arendaja saavad rohkem vabadust testida ning arendada uut funktsionaalsust ning vana funktsionaalsuse toimimise tagavad automatiseeritud testid.



Joonis 6. Automatiseeritud testide arendamise mõju [8]

4. Automatiseeritud testide koostamine ja nende tasuvus EMT e-teeninduse näitel

4.1 Testitava süsteemi ülevaade

Käesolevas peatükis selgitan EMT e-teeninduse rakenduse olemust, tehnilist tausta ja testimist ning selgitan miks valisin GUI testimise metoodika ning miks see on antud süsteemi puhul otstarbekas.

EMT iseteenindus on kõigile firma klientidele suunatud veebirakendus (vt www.emt.ee/eteenindus). Iseteenindus võimaldab kliendil EMT pakutavaid teenuseid Interneti kaudu tellida ning saada infot oma mobiilsideteenuste kohta. Protsessidena on võimalik iseteeninduses teha näiteks järgnevaid tegevusi: mobiilsidega liitumine, paketivahetus, ümbervormistamine, kõne-, interneti- ja teenuste eristuse tellimine ja palju muud.

EMT iseteenindus on *Liferay* rakendus, mis on jaotatud erinevate *portletite* vahel. Rakendusel on äriloogika kiht, mis on ühenduses andmebaasiga. Antud süsteemile ei olnud algselt kirjutatud ühtegi testi peale GUI testide. Kogu süsteemile tagantjärgi *unit*- testide kirjutamine pole efektiivne ja ei tasu ennast ära. Integratsiooni teste on samuti tagantjärgi raske teha, kuna süsteemi liidestused on kasvanud ajaga nii keerukaks, et hetkel tagantjärgi testide kirjutamine oleks tehnoloogiliselt mitteotstarbekas. Seega hetkeolukorras on kõige otstarbekam katta süsteem UI testidega.

UI testide tegemine on sageli raskendatud, kuna väiksemgi disainimuudatus võib testid nö „ära lõhkuda“. Antud olukorras, kus süsteem ei ole igapäevases arenduses ning muudatusi tuleb ette väga vähe, siis UI testidega süsteemi katmine on mõistlik. Vähene muutumine peaks tagama selle, et testid on stabiilsed ning kajastaksid rakenduse olukorda hästi. Kuna antud rakendusel pole ei *unit*- ega integratsiooni teste, siis peame kuidagi tagama, et süsteem ka taustal õigesti käituks. Selle lahenduseks valideerime UI testi tehtud tegevused andmebaasis: lisaks graafilises kasutajaliideses väljundi kontrollimisele vaatame, et andmed oleksid andmebaasi korrektselt maha toodud (vt Lisa 1. GUI Test EMT e-teeninduse näitel).

4.2 Tasuvuse uurimise metoodika

Enne, kui hakata tarkvara testimist automatiseerima, peaks kindlasti analüüsima väga mitmeid erinevaid faktoreid. Automatiseerimine muudab oluliselt testimise ülesehitust alates testi disainimisest kuni testi kirjutamise ja käivitamiseni. Automatiseerimine avaldab organisatsioonis päris suurt mõju: muutuvad testijate ülesanded, lähenemisviis testidele ja isegi tooteomadused. Automatiseerimise kasu ja võimaluste kohta levib väga palju erinevaid müüte. On väga oluline, et enne kui otsustada automatiseerimise kasuks ja olla nõus kõigi nende muutusega, mida see endaga kaasa toob, peab olema arusaam ja mõistmine kõigist automatiseerimisega kaasnevatest kasudest ja kuludest. See omakorda annab võimaluse automatiseerimisest maksimaalselt tulu saada. [16]

Testide automatiseerimisega kaasnevate muutuste hulka kuuluvad näiteks koolitused testijatele automatiseeritud testide tarkvara ning koostamise kohta. Automaattestide koostamine ja arendamine on väga erinev manuaalsest testimisest, seega olulisel määral muutuvad testijal tööks vajaminevad oskused ja testimisele lähenemine. Neid muutuvaid faktoreid peab kindlasti tõsiselt kaaluma ning töötajatega läbi arutama, kuna sellest võib sõltuda, kas automatiseerimine õnnestub ja osutub kasulikuks või mitte. Võib juhtuda, et mõnel testimisega pikemat aega tegelenud inimesel ei olegi motivatsiooni ja tahtmist uue lähenemisviisi peale üle minna ja teste automatiseerima hakata. Sellisel juhul ei tohiks automatiseerimist kindlasti peale suruma hakata, kuna kaasnevad tulemused ei oleks ilmselt ootustele vastavad. [16]

4.2.1 Automatiseerimise tasuvus juhatuse vaatest

Automatiseeritud testide edu peaks tulenema juhtide realistlikest ootustest ja selgest arusaamisest, kuidas läbi testide automatiseerimise muuta tarkvara kvaliteeti paremaks ja loodetavasti hoida kokku ka testimisele kuluvat aega.

On mitmeid valdkondi, millele juhtkond peaks seadma oma ootused: immateriaalsetele kuludele ja tuludele, manuaalse testimise ja automaatse testimise erinevustele, automatiseerimisega kaasnevatele muudatustele ja mõjudele organisatsioonis.

Immateriaalseid kulusid on raske hinnata ning isegi kui neile saab rahalist väärtust omistada, siis see väärtus võib olulisel määral varieeruda. Samuti on raskusi automatiseerimise tagajärjel esinevate muutuste mõõtmisel. Immateriaalsed faktorid võivad olla nii positiivsed kui

negatiivsed, kuid enamasti sisaldavad nad nii üht kui teist, sõltuvalt olukorrast ja sellest, kui hästi suudetakse automatiseerimist rakendada. Tulenevalt raskusest nende faktorite, jäetakse nad tavaliselt tasuvuse arvutamisest välja.

Mõned immateriaalsed faktorid [16]:

- *Hands-off* testimine. Testija töötasu on kergesti mõõdetav, aga arvuti poolt tulenev lisaväärtus on raskesti mõõdetav (automatiseeritud testid võivad käia ka ajal, mil inimesi pole tööl).

- Töötajate professionaalsuse tõus. Automatiseerimisel on tavaliselt motiveeriv ja töötajate tootlikkust tõstev mõju. See tuleneb uuest distsipliinist ja ülesannetest, mida automatiseerimine endaga kaasa toob.

- Edasiarenenud testid. See tuleneb täiendavatest võimalustest, mida automatiseerimine pakub.

- Töötajate tootlikkuse langemine. Testijatel võib tulla uute funktsionaalsuste testimises väike paus, mis tuleneb automatiseerimisega seotud tarkvara paigaldamisest ning testide automatiseerimisest.

- Kõik tiimi liikmed ei pruugi nõus olla automatiseerimisega kaasnevate muudatustega. Leidub inimesi, kes on nõus jätkama vaid manuaalse testimisega ja halvemal juhul võib ette tulla muutusest tulenev töötajate lahkumine organisatsioonist.

- Muutused testi kvaliteedis. Kvaliteedis võib esineda nii tõus kui langus.

- Läbitud testitsüklite arv enne reliisi. Automatiseerimine annab võimaluse saada kiire tagasiside süsteemi kohta ja seda lühikese ajavahemiku tagant. Testtsüklite rohkus võib kaasa tuua tõusu nii tootlikkuses kui kvaliteedis, kui samas ka põhjustada laiskust, testijate tähelepanu hajumist ning tarkvara kvaliteedi langust.

- Süsteemi kaetus testidega. Kattuvus võib nii tõusta kui kahaneda, sõltuvalt tehtud manuaalsete testide kvaliteedist, automatiseerimise vahenditest ja ka automatiseeritud testidest

    - Teatud sorti testimist saab sooritada vaid läbi automatiseerimise.

- Kattuvuse muutuse väärtust on raske rahaliselt hinnata.
- Hästi läbi viidud käsitsi testimine võib vahel katta palju huvitavaid testilugusid, mida automatiseeritud testid poleks katnud.
- Käsitsi testimine võib täiendada automatiseerimist olukordades, kui testi automatiseerimine pole otstarbekas.

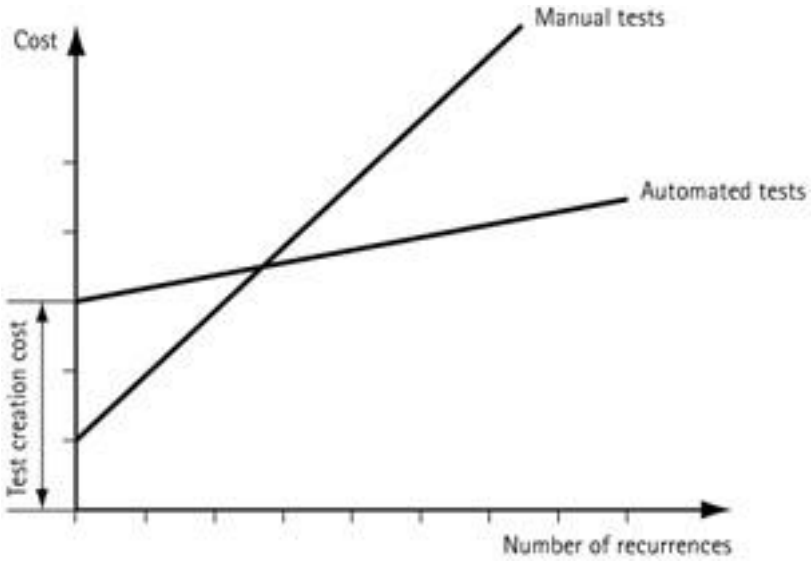
Juhtkonna ootused automatiseerimisega seoses lähtuvad sageli meediast, konverentsidelt või raamatutest pärit ülistavast infost automatiseeritud testide kohta. Teatud informatsioon võib olla paikapidav ja rakendatav, kuid automatiseerimise otsus peab olema langetatud analüüsides konkreetset projekti. Üldsusele kajastatav info automaattestimise kasulikkusest on pahatihti liialdatud ning põhineb vaid edulugudel. Testide automatiseerimine ei lahenda kindlasti kõiki testimisega seotud probleeme. Juhtkonna poolt halvasti seatud ootused võivad võimaliku edulooga automatiseerimise pöörata hoopis läbikukkumiseks. [16]

Automatiseerimise kulude osas peaks arvestama ka alljärgnevat [16]:

- Suur osa automatiseerimisega kaasnevast kasust tuleneb suuremast töömahust, mida testi planeerimise etapis analüüsile rakendatakse. Need kasutegurid on tegelikult automatiseerimisest sõltumatud ning oleksid manuaalsegi testimise puhul samad.
- Automatiseerimine toob alguses kaasa tulemuslikkuse languse ja suurema osa aja kulutamise testide automatiseerimisele.
- Automatiseeritud testide kirjutamine on palju raskem kui manuaalselt testide läbiviimine. Õnnestunud testide kirjutamine eeldab väga olulisel määral vastavaid teadmisi ja kogemusi. Tõenäoselt ei ole kõik testimise alal töötavad inimesed suutelised või huvitatud automatiseerimist ära õppima.
- Testide automatiseerimise sisseviimine organisatsioonis toob sageli kaasa töötajate professionaalsuse kasvu. See omakorda viib kvaliteetsema testimiseni, testijate paremate suheteni arendusosakonnaga ja laienemiseni arenenumasse tarkvara testimisse.
- Automatiseeritud testid vajavad sagedast hooldamist. Rakendusse kasutajaliidesesse tehtavate muudatustega kaasnevad peaaegu kindlasti muudatused GUI testides. Vastasel juhul testid ebaõnnestuvad.

- Automatiseerimisest tulenev tulu ei pruugi sageli ilmneda sama projekti raames, vaid alles järgmise või ülejärgmise projekti raames, mis kasutab samu automatiseeritud teste. See juhtub sageli paar aastat peale testide arendamist.

Nagu eelnevas kirjeldatud automatiseerimise kuludest võis aru saada, tasub automatiseerimine ennast ära pikemas perspektiivis. Alguses on kulud testide automatiseerimisel kindlasti suuremad, kui manuaalsel testimisel. Aga automatiseeritud testide rakendamise kulud on olulisel määral väiksemad, kui igakuiselt manuaalse testimise peale kuluv summa. Eelnevat automatiseeritud ja manuaalselt läbitavate testide kulude käsitlust iseloomustab joonis 7.



Joonis 7. Automaatestimise ja manuaaltestimise kulu võrdlus [17]

4.2.2 ROI faktorid

ROI (*return on investment*) tähendab eesti keeles investeeringutasuvust. Investeeringutasuvus arvutatakse tavaliselt saadud kasu jagamisel investeeringutega, mis antud projekti tehti [21]. Väga sageli võetakse automatiseerimine ette peale mõnda aega manuaalset testimist. Sellisel juhul saab automatiseerimisest saadavat tulu võrrelda manuaalse testimise peale kulutatava summa vähenemisega. [18]

Finantskulud seoses testide automatiseerimisega võib jagada kaheks: püsikulud ja muutuvkulud. Muutuvkulud sõltuvad testide hulgast, sellest mitu testtsüklit automatiseeritud

testid läbivad jpm. Automatiseerimise rakendamisel muutuvkulud kas suurenevad või vähenevad sõltuvalt sellest kui palju teste juurde arendatakse või kui palju teste jooksutatakse. Püsikulud on teatud perioodi kulud, mis on kindla väärtusega: automatiseerimise tarkvara hind, testkeskkonna tasu jpm. [18]

Näiteid püsikuludest:

- Tarkvara rakenduse testimiseks
- Automatiseerimise tarkvara
- Litsentsitasud
- Testkeskkonna loomine
- Testkeskkonna hooldamine
- Automatiseerimise tarkvara koolitused

Näiteid muutuvkuludest:

- Automatiseeritud testide testlugude koostamine
- Testide kirjutamine
- Testide hooldamine
- Testide käivitamine
- Testitulemuste analüüs
- Vigade raporteerimine
- Testitulemuste raporteerimine
- *After-hours* testimine süsteemi poolt

4.2.3 ROI arvutamine

Eelnevas oli käsitletud investeeringutasuvuse arvutamisel arvestatavaid faktoreid, järgnevas aga on tutvustatud investeeringutasuvuse arvestamise metoodikat. Selle meetodiga tasuvuse arvutamisel arvestatakse pigem võitu ajas mitte niivõrd rahas. Selle meetodi rakendamisel ei ole vaja teada projekti rahalist infot, sealhulgas testija tunnitasu jms. Arvutusi saab teha alles peale automatiseerimist, kui on teada, kui palju erinevate automatiseerimisega seotud tegevuste peale aega kulus. ROI arvutamiseks vaadeldakse automatiseerimisega kaasnevat ajakulu, automatiseerimisele eelneval ajal manuaalsele testimisele kulunud aega, kogu antud rakenduse testilugude arvu ja ROI perioodi (näiteks eeldatav süsteemi eluiga). [18]

Kasu all mõeldakse:

manuaalselt testide läbimiseks kuluv aeg * rakenduse kõikide testilugude arv * rakenduse eeldatav eluiga / 8

$$ROI = \frac{Gain - Investment}{Investment}$$

Investment = Automated Test Script Development Time (a)
+ Automated Test Script Execution Time (b)
+ Automated Test Analysis Time (c)
+ Automated Test Maintenance Time (d)
+ Manual Test Case Execution Time (e)

Gain = Manual Test Execution Time or Analysis Time
Multiplied by Total Number of Test Cases (Automated & Manual)
Multiplied by Period of ROI
Divided by 8

Joonis 8. Investeeringutasuvuse arvutamine [15]

Joonisel 8 on esitatud valem investeeringutasuvuse arvutamiseks. Et saada valemis sisalduvate suuruste väärtused tuleb kõigepealt arvutada välja alljärgnevas punktidena esitatud tegurid. Lisaks tuleb arvestada, et investeeringutasuvuse arvutustel arvestatakse ROI perioodiks nädalate arvu, milleks arvatakse olevat süsteemi eeldatav eluiga. Järgnevates arvutustes on igas arvutuses jagamistehe. Järgnevates arvutustes tähistab kaheksaga jagamine testija tööpäevas olevat kaheksat tundi, kaheksateistkümnega jagamine aga automatiseeritud testide oletatavat tööaega ööpäevas – 18 tundi (eeldan, et automatiseeritud testid ei saa erinevate takistuste tõttu ööpäevas 24 tundi järjest töötada).

- Automatiseeritud testide arendamiseks kulunud aeg = keskmine testi automatiseerimiseks kulunud aeg tundides * automatiseeritud testilugude arv / 8
- Automatiseeritud testide testilugude läbimiseks kuluv aeg = keskmine automatiseeritud testi läbimiseks kuluv aeg * automatiseeritud testilugude arv * ROI periood / 18
- Automatiseeritud testide analüüsiks kuluv aeg = testide analüüsiks kulunud aeg * ROI periood / 8

- Automatiseeritud testide hooldamiseks kuluv aeg = hoolduseks kuluv aeg * ROI periood / 8

- Manuaalselt läbitavateks testideks kuluv aeg = (keskmine manuaalse testi testimiseks kuluv aeg * manuaalselt läbitavate testilugude arv * ROI periood) / 8

Kui kõik tegurid on eraldi välja arvutatud, tuleb nad panna valemisse (joonis 7) ja seejärel saab antud projekti raames automatiseeritud testide tasuvuse välja arvutada. [18]

4.3 Metoodika rakendamine EMT e-teeninduses

Ka EMT e-teeninduse automatiseerimise projekti korral kasutan eelkirjeldatud ROI meetodit arvutamaks, kas antud projekti raames on automatiseerimise protsess olnud tulus.

Hetkel olen EMT e-teeninduse automatiseerimise raames testiks realiseerinud 25 testilugu. See katab suurema osa e-teenindusest, kuid siiski mitte kõike. On testilugusid, mille puhul olen otsustanud ka edaspidi jätkata manuaalset testimist, kuna testiloo automatiseerimine ei ole otstarbekas. Otstarbekuse hindasin automatiseerimiseks kuluva aja võrdlemisel manuaalsele regressioonile kuluva ajaga. Hetkel on veel ka selliseid testilugusid, mille realiseerimiseni automatiseeritud testiks pole ma veel jõudnud. Neid testilugusid on hinnanguliselt 5. Automatiseerimise tasuvuse arvutamisel kaasan arvutusse ka 5 automatiseerimata testi – arvestan nende testide manuaalseks läbimiseks kuluva aja automatiseerimise protsessile kuluvale ajale juurde.

Eelnevas punktis (4.2.3) kirjeldatud arvutus sobib minu uurimusse hästi, kuna eesmärgiks ei olnud hinnata mitte rahalist tasuvust, vaid selgitada, mis perioodi jooksul tasub automatiseeritud testide tegemine ennast ajaliselt ära.

Oma projekti tasuvuse arvutamiseks muudan pisut ROI arvutamise valemit. EMT e-teeninduse puhul ei oma tähtsust aeg, millega automatiseeritud test testilood läbib – otsustasin selle teguri seega valemist välja jätta, kuna see aeg on sõltumatu testija kulutatavast ajast. Teine minu poolt valemisse tehtav muudatus: arvestan arendamiseks ja hoolduseks kuluva aja koos. Selline muudatus tuleneb sellest, et antud projekti raames pole tööaja raporteerimisel eraldatud arendamise ja hooldamise etappe, kõik on fikseeritud ühtselt automatiseeritud testidele kuluva ajana. Kokkuvõttes viimane muudatus arvutuse tulemust aga ei muuda.

Järgnevas esitan ROI arvutused ning selgitan EMT e-teenindusest tulevaid väärtuseid:

- EMT e-teeninduse eeldatav kasutusperiood on 1,5 aastat. ROI arvutuses on vajalik aastate arv teisendada nädalateks. Kuna ühes aastas on ligikaudu 52 nädalat, siis arvestan eeldatavaks rakenduse kasutusperioodiks 78 nädalat.

  ROI periood: 78 nädalat

- Arvestan, et keskmiselt kulub nädalas automatiseeritud testide analüüsile 2 tundi.

  Automatiseeritud testide analüüs: 2 * 78 / 8 = 19,5 (päeva)

  Automatiseeritud testide analüüs: 19,5 päeva

- Keskmiselt on kulunud testi automatiseerimise ja hoolduse peale 4,56 tundi. Kuna automatiseeritud on 32 testi, siis automatiseerimise ja hooldamise peale kuluva aja saan korrutades ühele testile keskmiselt kuluva aja testilugude arvuga. Kuna vastust on vaja päevades, siis jagan saadud tulemuse kaheksaga (töötundide arv päevas):

  4,56 * 32 / 8 = 18,24 (päeva)

  Automatiseeritud testide arendus ja haldus: 18,2 päeva

- Manuaalselt võtab ühe testiloo läbimine aega 9,4 minutit. Arvutuses on vaja saada antud väärtust tundides – 9,4 minutit = 0,16 tundi.

  Ühe testi läbimiseks kulub keskmiselt 0,16 tundi

- Automatiseerimata 5 testi läbimiseks kuluva aja saame keskmise manuaalselt testi läbimiseks kuluva aja korrutamisel 5-ga, seejärel jagamisel kaheksaga, et saada vastust päevades, ning vastuse korrutamisel 78-ga, et saada aega, mis kulub kogu ROI perioodi vältel manuaalsele testimisele.

  0,16 * 5 / 8 * 78 = 7,80 (päeva)

  5 manuaalselt läbitava testi jaoks kulub 7,8 päeva

ROI valem:

Manuaalse testimise kulu (*Gain*) = 0,16 * 37 * 78 / 8 = **57,7 (päeva)**

Investeering automatiseerimisse (*Investment*) = 19,5 + 18,2 + 7,8 = **45,5 (päeva)**

(Manuaalse testimise kulu – Investeering automatiseerimisse) / Investeering automatiseerimisse = (57,7 – 45,5) / 45,5 = 0,27

0,27 * 100% = **27%**

Saadud tulemusest võin järeldada, et automatiseerimine on olnud tulemuslik. Investeeringuna testide automatiseerimisse panustasime 45,5 päeva, ilma automatiseerimiseta oleksime kulutanud regressioonide läbimiseks 57,7 päeva. Automatiseerimine osutus efektiivseks ning tänu sellele võidame ajas 12,2 tundi (27%).

5. Kokkuvõte

Antud töö eesmärgiks oli analüüsida võrdlevalt manuaal- ja automatiseeritud testide olemust, selgitada nii automatiseerimisega kaasnevaid häid omadusi kui ka probleeme ning käsitleda testide automatiseerimise tasuvust.

Töö põhieesmärgiks oli selgitada, kas EMT e-teeninduse automatiseerimine osutus efektiivseks – kas testide automatiseerimisega kaasneb võrreldes manuaalse testide tegemisega ajavõit.

Rakendasin automatiseerimise tasuvuse metoodikat EMT e-teeninduse regressioontestide automatiseerimise projektile ning tulemustest selgus, et automatiseerimise protsess osutus efektiivseks. Rakendatud metoodika keskendub vaid ajalise efektiivsuse arvutamisele, jättes välja kõik rahalised väärtused.

Analüüsi tulemusena selgus, et ilma automatiseerimiseta oleksime e-teeninduse testimisele süsteemi eeldatava eluea (1,5 aasta) jooksul kulutanud ligikaudu 57 tööpäeva. Tänu automatiseerimisele kulub e-teeninduse testimiseks nüüd 45 tööpäeva. See tähendab, et ajaliselt võidame tänu automatiseerimisele 12 päeva ehk 27%.

1. Testide automatiseerimine osutus edukaks, kuna automatiseerimise otsus oli tõsiselt läbi mõeldud ja arvestatud nii võimalike positiivsete kui ka negatiivsete aspektidega.

2. Antud *GUI* testide automatiseerimise edu taga on ka kindlasti minimaalne keskkonna arendamine, mis tähendab, et teste ei pea olulisel määral muutma.

3. Automatiseerimise efektiivsus ei pruugi väljenduda kõikides projektides, kuid põhjalikult läbi mõeldud ja kaalutletud otsus teste automatiseerida toob sageli kvaliteetsema tarkvara ja lühemad testtsüklid.

Antud projekti tasuvuse arvutamisel on kindlasti ka edasiarendamise võimalusi. Hetkel arvutasin automatiseeritud testide tasuvuse ajaliselt, kuid tasuvust on võimalik arvutada ka lähtudes rahalisest väärtusest, võttes arvesse testija palga, litsentsitasud, koolituskulud jpm. Hetkel polnud automatiseerimise protsess veel ka täielikult lõpule viidud, seega oleks võimalus tasuvus arvutada ka peale täielikku automatiseerimist.

Summary

The aim of work, was to comparably analyse the essence of manual and automated testing, to explain the virtues and problems derived from utilization of automated tests and to question the profitability of automated tests.

The main focus of the work was to clarify whether the automation of EMT self-service was successful – if the automation helped to save time comparing to the time spent on manual testing.

The results of the utilization of the cost-effectiveness methodology on the regression tests of EMT self-service showed, that the automation process appeared to be effective. Applied methodology focuses on the efficiency of time by leaving out all the financial values.

The results of the analysis showed that without the automation, the time spent on testing the self service during its life-span would have been approximately 57 working days. By the virtue of automated tests the time spent on testing is now 45 working days. That means, that test automation helped to save 12 working days, which is 27% of time spent on testing.

1. Test automation proved to be successful, because the decision of test automation was seriously considered and both positive and negative aspects were taken into account.

2. The success behind automated GUI tests is the fact that the environment rarely changes. This means that there is almost no need to change tests dramatically.

3. The efficiency of automated tests would not come off with every project, but frequently, comprehensively considered decision of automating tests would bring more qualitative software and shorter test cycles.

There are certainly opportunities for further development of this work. Presently, I calculated the efficiency of tests considering time, but the efficiency could be also calculated with considering financial values – for example, considering tester's wage, licence fees, fees of special education and so forth. At the moment, test automation with current project was not completely finished, so there is opportunity to calculate the efficiency of test automation after the automation is completed.

6. Kasutatud kirjandus

1. Graham, D., Veenendaal, E., Evans, I., Black, R. (2011). *Foundations of Software Testing.*

2. Jeffrey, D. (2006). *Regression Testing*. [WWW] http://www.cs.ucr.edu/~gupta/teaching/620-06/regression.pdf (02.04.2015)

3. George, C. (2013). *To Automate or Not To Automate…Is that the Question?*[WWW] https://www.simple-talk.com/blogs/2013/02/22/to-automate-or-not-to-automateis-that-the-question/ (06.04.2015)

4. LAP. (2005). Tarkvara kvaliteet ja standardid. [WWW] http://www.lap.ttu.ee/erki/failid/konspekt/tarkvara_kvaliteet_ja_standardid_idx5721/idx5721_konspekt.pdf(02.04.2015)

5. Softwaretestingclass. (2014). *Automation Testing Vs Manual Testing*. [WWW] http://www.softwaretestingclass.com/automation-testing-vs-manual-testing/ (06.04.2015)

6. Idtus. (2015). *What Is Automated Software Testing?* [WWW] http://idtus.com/what-is-automated-software-testing/ (06.04.2015)

7. Myers, G. (2004). *The Art Of Software Testing, Second Edition*. Canada: John Wiley & Sons, Inc.

8. Zabir, O. (2011). *How to Convince Developers and Management to Use Automated Test instead of Manual Test.* [WWW] http://www.codeproject.com/Articles/78499/How-to-convince-developers-and-management-to-use-a (11.04.2015)

9. Rouse, M. (2014). *Automated Software Testing.* [WWW] http://searchsoftwarequality.techtarget.com/definition/automated-software-testing (11.04.2015)

10. Rouse, M. (2014). *Unit Testing Definition.* [WWW] http://searchsoftwarequality.techtarget.com/definition/unit-testing (11.04.2015)

11. Rouse, M. (2014). *Integration Testing Or Integration And Testing Definition.* [WWW] http://searchsoftwarequality.techtarget.com/definition/integration-testing (11.04.2015)

12. Anderson, S. (2011). *Regression Testing.* [WWW] *http://www.inf.ed.ac.uk/teaching/courses/st/2011-12/Resource-folder/11_regression.pdf* (11.04.2015)

13. Appperfect. (2010). *GUI Testing.* [WWW] http://www.appperfect.com/products/application-testing/app-test-gui-testing.html (16.04.2015)

14. Sphavens. (2012). *A Comparison Of Automated Testing And Manual Testing.* [WWW] http://sphavens.com/2012/08/a-comparison-of-automated-testing-and-manual-testing/ (16.04.2015)

15. Testing-whiz. (2011). *How to Calculate ROI for Test Automation.* [WWW] http://blog.testing-whiz.com/2011/12/how-to-calculate-roi-for-test.html (02.05.2015)

16. Hoffman, D. (2012). *Cost Benefits Analysis of Test Automation.* [WWW] http://www.softwarequalitymethods.com/papers/star99%20model%20paper.pdf (28.04.2015)

17. Frohlich, P. (2011). *Costs and Benefits of Automated Unit Tests*. [WWW] http://flylib.com/books/en/2.671.1.99/1/ (19.04.2015)

18. Johnson, D. (2013). *Test Automation ROI.* the United States: DiJohn Innovative Consulting, Inc. [WWW] http://www.automatedtestinginstitute.com/home/articleFiles/articleAndPapers/Test_Automation_ROI.pdf (15.04.2015)

19. Getautoma. (2011). *The costs of a bug.* [WWW] http://www.getautoma.com/blog/the-costs-of-a-bug (15.04.2015)

20. Helper, D. (2012). *Software development bugs: How to identify and prevent them.* [WWW] http://searchsoftwarequality.techtarget.com/tip/Software-development-bugs-How-to-identify-and-prevent-them (15.04.2015)

21. Entrepreneur. (2015). *Return on Investment (ROI).* [WWW] http://www.entrepreneur.com/encyclopedia/return-on-investment-roi (28.04.2015)

Lisa 1. GUI test EMT e-teeninduse näitel

```
*** Settings ***
Documentation     New Subscription, minuEMT package
Suite Setup       Open Browser    ${homepage}    ${test_suite_browser}    chrome
Suite Teardown    Close All Browsers
Test Setup        Test Case Setup
Test Teardown     Test Case Teardown
Test Timeout      15 minutes
Resource          /Shared_resources_EMT/common_resources.txt
Resource          ${EMT_shared_resource_path}/environment_${ENVIRONMENT}_resources.txt
Resource          ${EMT_itb_tc_resource_path}/02_010_resources.txt

*** Test Cases ***
minuEMT liitumine
    [Tags]    Ready    debug
    Find Customer
    Eliminate Party Debts    maac    ${MAAC1_ref}
    ITB login
    Kliki uus liitumine
    Vali erakliendi liitumine
    Täida andmed
    Vali paketi tüüp minuEMT
    Vali number
    Vali SIM kaarti tüüp
    Kinnita leping
    Allkirjastamine kulleriga
    Kohaletoimetamine
    Ostukorv
    Täida kohaletoimetamise andmed
    Check TBCIS Record

*** Keywords ***
Find Customer
    Connect To Database    ${DB_USERNAME}    ${DB_PASSWORD}    ${DB_DSN}
    ${query_results}=    Query From File    ${EMT_itb_tc_query_resource_path}/uus_liitumine.txt
    Disconnect From Database
    Set Suite Variable    ${MAAC1_ref}    ${query_results[0][0]}
    Set Suite Variable    ${susgREF1}    ${query_results[0][1]}
    Set Suite Variable    ${MAAC1}    ${query_results[0][2]}
    Set Suite Variable    ${maacPN1}    ${query_results[0][4]}
    Set Suite Variable    ${susgEN1}    ${query_results[0][3]}
    Set Suite Variable    ${susgPN1}    ${query_results[0][4]}
    Set Suite Variable    ${IK1}    ${query_results[0][7]}
    Set Suite Variable    ${susgNUM1}    ${query_results[0][8]}
    Set Suite Variable    ${SUSG1}    ${susgEN1} ${susgPN1}

Eliminate Party Debts
    [Arguments]    ${case}    ${value}
    Import Resource    /Shared_resources_EMT/dealgateVariable robot
    Connect To Database    ${DB_USERNAME}    ${DB_PASSWORD}    ${DB_DSN}
    Run Keyword If    '${case}'=='maac'    Insert Or Delete    ${eliminatePartyDebtsUsingMaac}
    Run Keyword If    '${case}'=='party'    Insert Or Delete    ${eliminatePartyDebtsUsingParty}
    Disconnect From Database

LogIn
    [Arguments]    ${CustomerNumber}    ${FirstName}    ${LastName}    ${MAAC_name}    ${MAAC_ref}
    Log    ============================================================
    Log    LogIn
    Log    ============================================================
    Wait Until Page Contains Element    ID=personalCode    ${TIMEOUT}
    Input Text    ID=personalCode    ${CustomerNumber}
    Sleep    2
    Click Button    ID=personalCodeLoginButton
    Sleep    5
    ${status}    ${value} =    Run Keyword And Ignore Error    Page Should Contain    Esindatava valimine
    Run Keyword If    '${status}'=='PASS'    Esindatava valimine
    ${status}    ${value}=    Run Keyword And Ignore Error    Click Element    //div[@class="summary-data choose-user

Kliki uus liitumine
    Wait Until Page Contains    Ülevaade
    Go To    ${homepage}/eteenindus/liitumine
    Sleep    5
    Wait Until Page Contains Element    //button[@type='button']    40
    Click Element    //button[@type='button']

Vali erakliendi liitumine
    Wait Until Page Contains Element    //input[@name='clientType' and @value='0']/following::h2    30
    Click Element    //input[@name='clientType' and @value='0']/following::h2
    Sleep    5
    # järgmised on teiste klienditüüpide checkboxid!!!
    #Run Keyword If    '${type}' == 'existingB2C'    Click Element    //input[@name='clientType' and @value='0']/followir
    #Run Keyword If    '${type}'== 'newB2C'    Click Element    //input[@name='clientType' and @value='1']/following::h2
    #Run Keyword If    '${type}' == 'newB2B'    Click Element    //input[@name='clientType' and @value='2']/following::h2
    Click Button    //div[@class='buttons clear']//button[@type='submit' and contains(text(), 'Jätka')]    #for Jätka b
    Sleep    5

Täida andmed
    Run Keyword And Continue On Failure    NEWMOB Verify Page Liitumine - Andmed    #TODO: Verify variables
    NEWMOB Fill In Page Liitumine - Andmed    FirstName1    LastName1
    Sleep    1
    Execute Javascript    var a=window.document.getElementById('_newcontract_WAR_processportlets_processForm').getEleme
    Sleep    1

Pakett
    Wait Until Page Contains Element    //div[@class='buttons clear']//button[@type='submit' and contains(text(), 'Jä
    Click Button    //div[@class='buttons clear']//button[@type='submit' and contains(text(), 'Jätka')]

Vali number
    Wait Until Page Contains    Kõik vabad numbrid    100
    Click Link    Liitumine olemasoleva numbriga
    Sleep    4
    Click Link    Kõik vabad numbrid
    ${newmob1}=    Get Text    //form[@id='_newcontract_WAR_processportlets_freeNumberForm']/table/tbody/tr[1]/td[1]/lab
    Set Suite Variable    ${newmob}    ${newmob1}
    Wait Until Page Contains Element    //form[@id='_newcontract_WAR_processportlets_freeNumberForm']/table/tbody/tr[
    Click Element    //form[@id='_newcontract_WAR_processportlets_freeNumberForm']/table/tbody/tr[1]/td[1]
    Wait Until Page Contains Element    //form[@id='_newcontract_WAR_processportlets_freeNumberForm']/p/button    35
    Sleep    2
    Click Element    //form[@id='_newcontract_WAR_processportlets_freeNumberForm']/p/button

    Click Element    //label[@for="_newcontract_WAR_processportlets_termsForm_agree" and @class='label-check']
    Execute Javascript    var a=window.document.getElementById('_newcontract_WAR_processportlets_confirmBtn'); a.click
    Sleep    1
    Execute Javascript    var a=window.document.getElementById('confirmButton'); a.click();

Allkirjastamine kulleriga
    Wait Until Page Contains Element    //label[@for="_newcontract_WAR_processportlets_parcel_sign" and @class='label
    Click Element    //label[@for="_newcontract_WAR_processportlets_parcel_sign" and @class='label-radio']
    Execute Javascript    var a=window.document.getElementById('_newcontract_WAR_processportlets_signingMethodButton').

Kohaletoimetamine
    Sleep    30
    Click Button    Jätka

Ostukorv
    Wait Until Page Contains Element    id=_shoppingcart_WAR_servicesportlets_deliverySaveButton    35
    Execute Javascript    var a=window.document.getElementById('_shoppingcart_WAR_servicesportlets_deliverySaveButton'
    Sleep    5
    Execute Javascript    var a=window.document.getElementById('confirmButton'); a.click();

Mine Ostukorvi
    Wait Until Page Contains Element    //a[@title="Ostukorv"]    20
    Click Element    //a[@title="Ostukorv"]

Check TBCIS Record
    ${res}=    TBCIS Get SUDE Record    ${newmob}    NEWMOB
    Should Not Be Empty    ${res}
    TBCIS ITB Poll Subscriber Details Status    ${newmob}    NEWMOB    COMPLETE    15

TBCIS Get SUDE Record
    [Arguments]    ${serv_num}    ${serv_type}
    Connect To Database    ${DB_USERNAME}    ${DB_PASSWORD}    ${DB_DSN}
    ${query_results}=    Query From String    select a.request_status, a.ref_num, a.rety_type_code, a.usre_ref_num,a.se
    Disconnect From Database
    Should Not Contain    ${query_results[0]}    ERROR
    [Return]    ${query_results}

SUB ITB Validate SQL SUDE
    [Arguments]    ${number}    ${type}
    Connect To Database    ${DB_USERNAME}    ${DB_PASSWORD}    ${DB_DSN}
    @{query_results}    Query From String    Select * From Subscriber_Details A Where Serv_Num = '${number}' And A.Rety
    Disconnect From Database
    Should Not Be Empty    @{query_results}[0]
    Set Suite Variable    ${sude_results}    @{query_results}[0]
    Set Suite Variable    ${sude_status}    ${sude_results[3]}
    [Return]    ${sude_status}
```