Tartu Ülikool
Matemaatika-informaatikateaduskond
Arvutiteaduse instituut
Infotehnoloogia eriala

Meeli Pällin

Automaattestimisvahendite kasutus ning praktiline ülevaade Seleniumi näitel

Bakalaureusetöö (6 EAP)

Juhendaja: M.Sc Vambola Leping

Autor:..... „ ..... “ 2012
Juhendaja:..... „ ..... “ 2012

Lubada kaitsmisele
Professor ..... „ ..... “ 2012

Tartu 2012

Sisukord

Sissejuhatus

Antud bakalaureusetöö eesmärgiks on juhtida tähelepanu testimise olulisusele tarkvaraarenduses, keskendudes automaattestimisvahendite kasutamisele ja kasutamise võimalustele. Plaanis oli saada ülevaade Eestis asuvate tarkvarafirmade testimisharjumustest, rõhutatult on uuritud automaattestimise valdkonda ja kasutatavaid töövahendeid. Töös üritatakse anda ülevaade testimisest üldiselt – tuues välja erinevad testimise tehnikad ja meetodid, keskendudes seejärel põhjalikumalt erinevatele automaattestimise töövahenditele ja vaadeldes konkreetset töövahendit *Selenium* praktilisest küljest.

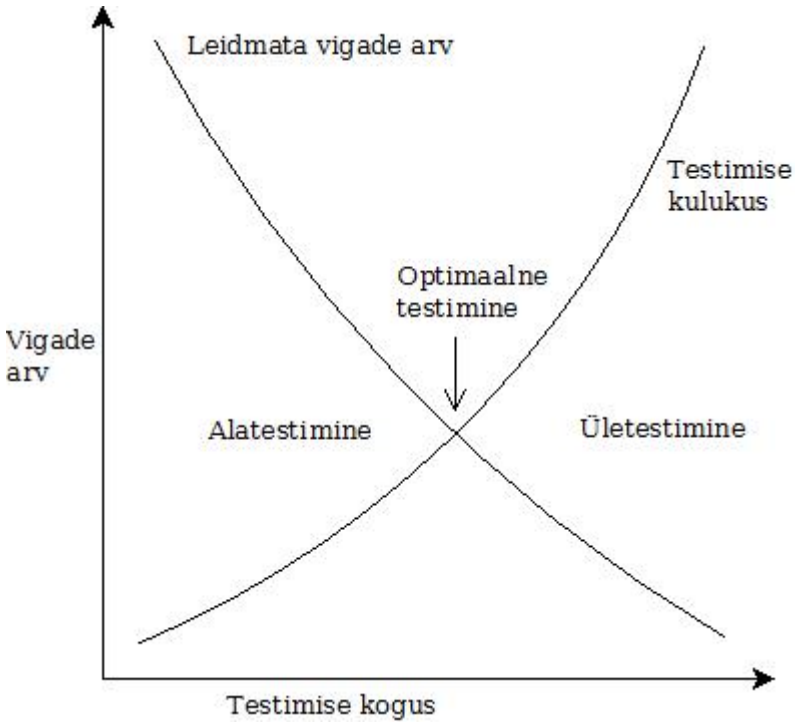
Töö teemat innustas valima antud valdkonna minimaalne osakaal õppeprogrammis ja huvi ning vajadus end nii testimise kui ka automatiseeritud testimise ja vastavate töövahendite osas harida. Samuti soov välja uurida, kuidas ja kui palju Eesti tarkvarafirmad testimisega tegelevad. Loodan, et küsitlusele vastamine pani ka tarkvarafirmad testimisele rohkem mõtlema.

Töö koosneb kolmest peatükist. Esiteks annan üldise ülevaade testimisest ning automaattestimisest rõhutades testimise olulisust ja vajalikkust. Kirjeldan enimkasutatavaid meetodeid, testliike ja töövahendeid. Seejärel kirjeldan ja analüüsin tarkvarafirmade hulgas läbiviidud küsitlust, mille eesmärgiks oli saada ülevaade ettevõtete testimisharjumustest ja kasutatavatest vahenditest. Ning lõpetuseks annan põhjalikuma ülevaate küsitluse tulemusena kõige populaarsemaks osutunud töövahendist *Selenium* ja viin sellega läbi mõned lihtsad praktilised näited.

1. Tarkvara testimine

1.1 Vajadus

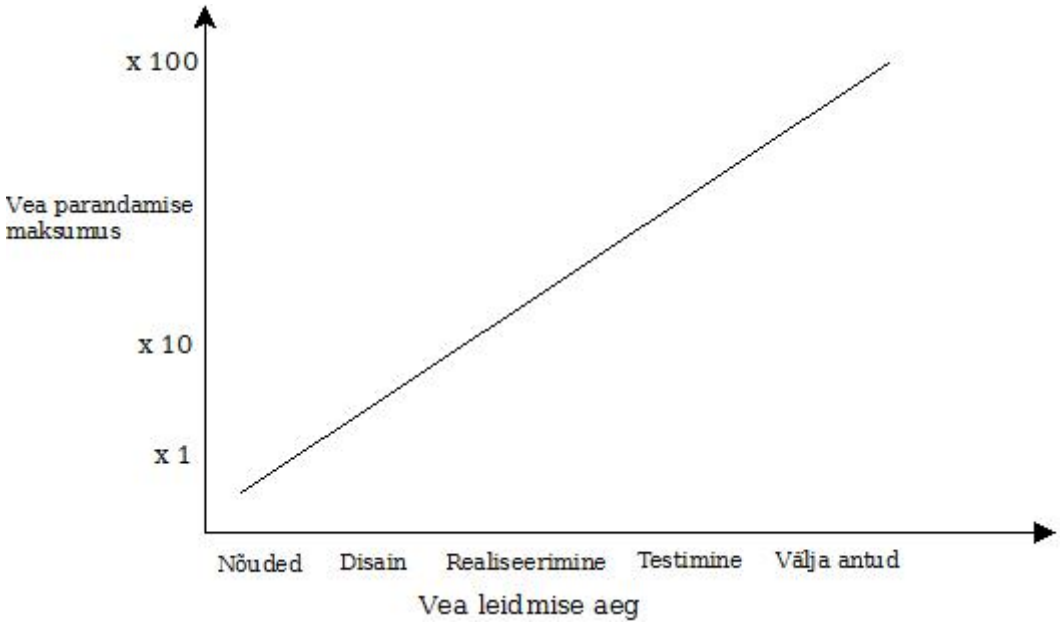
Tarkvara testimine on tarkvaraarenduse protsessis väga oluline etapp, mis annab objektiivse hinnangu tarkvarast – selle võimalustest, kvaliteedist ja jõudlusest. Tarkvaraarenduse protsessi eesmärgiks on pakkuda kliendile võimalikult kvaliteetset, töökindlat ja nõudmistele vastavat tulemust. Põhjalikult teostatud testimine katab tarkvaraarenduse kõik sammud. Kontroll selle üle, kas kõik nõutav on lahendatud ja töötab ilma vigadeta ka erandolukordades, juhuste kokkusattumise tulemusena ja suurtel koormustel, on määrava tähtsusega. Läbimõtlematusest ja kontrollimatusest tingitud vead halvendavad nii kasutajakogemust kui võivad tekitada ohtlikke olukordi sõltuvalt projekti vald-konnast ja kasutusest. Kuna testimine on siiski kulukas nii aja kui raha suhtes, on oluline testida mitte ülemäära palju, vaid optimaalsel tasemel. Vigade leidmise arvu, testimise kulukust ja õige testimise kestvuse leidmist illustreerib järgnev joonis. [1]



Joonis 1 – Optimaalne testimistase [1]

1.2 Ülevaade

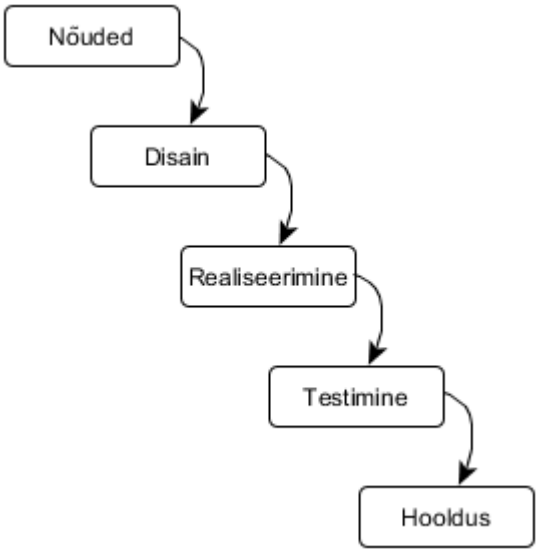
Testimine kipub tarkvaraprojektides jääma lõksu arenduse lõpptähtaja ja projekti valmimise tärmini vahele. Sageli võtab arendus rohkem aega, kui algselt planeeritud, ent valmimistähtaega edasi ei lükata – seetõttu jääb halva planeerimise korral testimiseks vähem aega kui vajalik ja testimise ning tarkvaratoote kvaliteet jäävad madalamaks. Testimisega tuleb arvestada ja alustada juba tarkvaraprojekti analüüsides. Testplaanide ja –juhtude koostamisel vaadatakse esitatud nõuded veelkord üle ja see võimaldab arutada läbi erinevad vaatenurgad ja võimalikud teisitimõistmised piisavalt vara, et muutuste tegemine ei oleks kulukas ja aeganõudev. Mida hilisemas etapis viga leitakse, seda kulukamaks parandus kujuneb. Nõuete analüüsi etapis ei maksa vea parandus peaaegu midagi, peale seda, kui tarkvaratoode on välja antud, võib sama vea parandamine osutuda kuni 100 korda kallimaks. Seost vea parandamise maksumuse ja vea leidmise aja vahel kirjeldab järgnev joonis.



Joonis 2 – Seos vea leidmise aja ja vea parandamise kulu vahel [1]

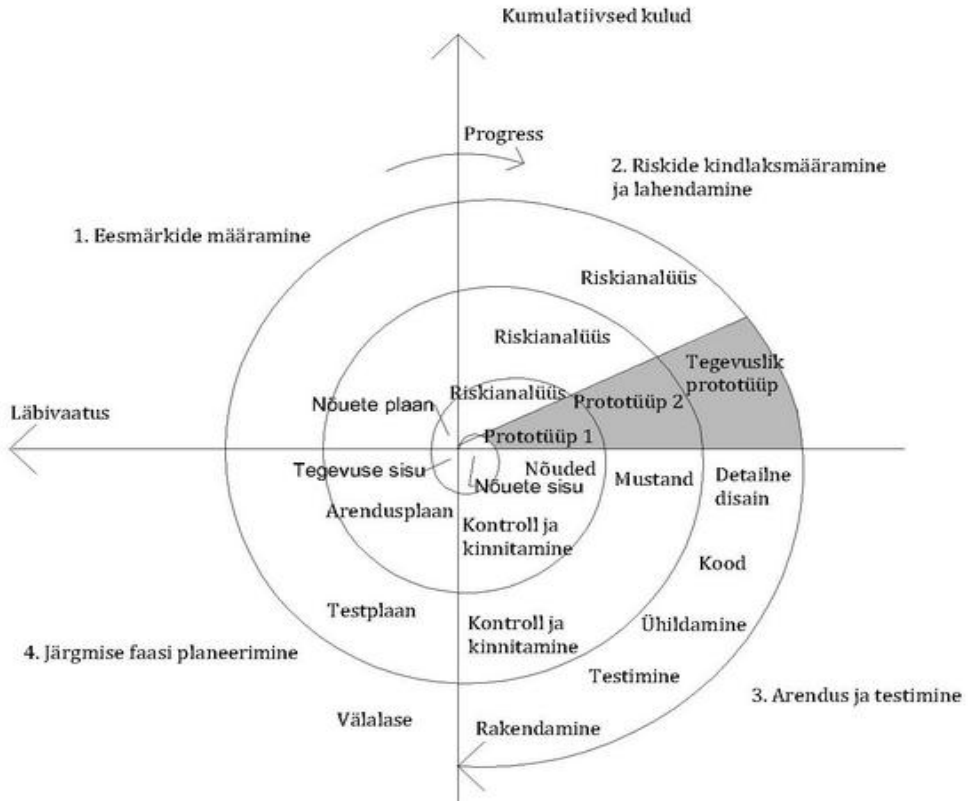
Tarkvaraarendus võib toimuda erinevate mudelite alusel. Testimisel on vaja arvestada, millise mudeli alusel arendusprotsess toimib, sest testimine võib olla nii üks eraldiseisev osa arendusprotsessist, kui ka rohkem seotud ülejäänud etappidega.

Koskmudel – tarkvaraarenduse etappe kujutatakse nii, et iga järgnev etapp on eelmisest allpool ning töö kulgeb ülevalt alla liikudes. Koskmudel on vana ja lihtne tarkvara-arenduse mudel, kus kõik sammud järgnevad üksteisele nii, et mitut tegevust pole võimalik ühel ajal sooritada, kogu ühe etapi töö sooritatakse korraga. Koskmudel sobib kasutamiseks väiksemates projektides, sest hilisem tagasipöördumine mõne eelneva etapi juurde on keeruline ja kulukas. [17]



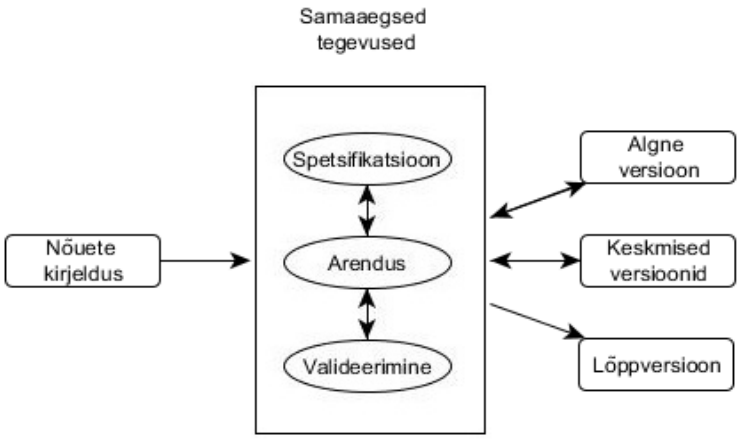
Joonis 3 – Koskmudel [21]

Spiraalmudel – tarkvaraarenduse mudel, mis üritab ühendada ülalt-alla ja alt-üles põhimõtetel toimivaid mudeleid. Arendusprotsessi kujutatakse spiraalina, kus iga kordus on mingi arendusfaas. Spiraalmudel koosneb neljast sektorist: 1) eesmärkide seadmine – siin määratakse käesoleva faasi eesmärgid ja võimalused, 2) riskide hindamine ja vähendamine – siin toimub riskianalüüs, 3) arendus – valitakse käesoleva faasi jaoks sobiv arendusmudel ning toimub arendamine, 4) järgmise etapi planeerimine – kogu projekt vaadatakse üle ja võetakse vastu järgmise faasi tegevusplaan. Spiraal-mudeliga arendus sobib hästi suurte ja keeruliste süsteemide jaoks. [12]



Joonis 4 – Spiraalne arendusmudel [5]

Evolutsiooniline mudel – arendusmudel, mille eesmärgiks on tihe koostöö kliendiga ning nõuete mõistmine. Evolutsioonilise arendusmudeli tulemusena valmiv tarkvara võib olla halvasti struktureeritud, sest süsteemi arendatakse osade kaupa. Mudel sobib väiksemate ja keskmiste süsteemide või suurte süsteemide osade jaoks. Evolutsioonilise mudeli puhul ei ole arendus eraldiseisev etapp, vaid on spetsifikatsiooni koostamise ja valideerimisega tihedalt seotud. Arendatav süsteem liigub väikeste valminud osade kaupa sageli kliendi ja arendusmeeskonna vahel. [12]



Joonis 3 - Evolutsiooniline arendusmudel [12]

1.3 Meetodid

Tarkvara testimise meetodid jagatakse üldjuhul valge *(white-box testing)* ja musta kasti *(black-box testing)* testideks. Valge kasti testimise puhul on testijal ligipääs andmestruktuuridele ja rakendatavale koodile. Valge kasti meetodiga testitakse põhiliselt rakendusliidest, kogu staatiline testimine viiakse läbi valge kasti meetodit kasutades. Musta kasti meetodiga testimisel ei oma testija ülevaadet tarkvara teostusest, ta saab sisestada andmed ja näeb ainult tulemust, aga ei oma teavet kuidas tulemus saadi. Musta kasti testimise puhul on vajalikud põhjalikud testjuhtumid, mida tarkvara kontrollimiseks läbi katsetada. Samuti vajab musta kasti testimine rohkem kogemusi ja head intuitsiooni. [1]

Testimist jagatakse lisaks musta ja valge kasti testimisele ka staatiliseks ja dünaamiliseks testimiseks. Staatiline testimine tähendab vaatlevat katsetamist nii, et programm ei tööta, näiteks nõuete analüüsi dokumendi kontroll. Dünaamilise testimise korral käivitatakse programmikood. [1]

1.4 Testiliigid

Arendusprotsessi eri etappidel on võimalik testida süsteemi erinevate nurkade alt. Mida rohkem erineval tasemel ja eesmärgil teste läbi viia, seda parema ülevaate süsteemi nõuetele vastavusest saab. Järgnevalt annan lühida ülevaate erinevatest testiliikidest. [2], [14], [19]

Ühiktestimine *(Unit testing)* – ühiktestimise käigus kontrollitakse tarkvara osade töötamist. Tavaliselt käivad ühiktestid konkreetse väiksema koodi osa, näiteks funktsiooni kohta ning kontrollivad, kas funktsioon töötab ettenähtult näiteks piirväärtuste korral.

Lõimumise testimine *(Integration testing)* – lõimumistestide korral kontrollitakse komponentide vaheliste liideste vastavust planeeritud disainile ja üritatakse leida vigu komponentide vahelisest andmevahetusest. Komponente võib liita nii ühekaupa kui ka suurema grupina korraga ja see võib põhjustada ebakõlasid süsteemi töös.

Süsteemitestid *(System testing)* – süsteemitestid leiavad aset projekti lõpus ja nende eesmärgiks on kinnitada süsteemi vastavust etteantud nõuetele.

Vastuvõtutestid *(Acceptance testing)* **–** vastuvõtu testimine võib tähendada kahte erinevat lähenemisviisi: 1) kasutaja vastuvõtu testimine – kliendipoolne testimine kliendi riistvara ja arenduskeskkonnaga. 2) proovitest, veendumaks uue tarkvara-komponendi sobivuses põhilisse testprotsessi. Proovitest viiakse tavaliselt läbi enne lõimumise testimist.

Regressioonitestimine *(Regression testing)* **–** regresioonitestimine viiakse läbi peale suuri muutusi koodis, et leida vigu, mis võivad olla tingitud uue ja vanade tarkvara osade kokkusobimatusest. Regresioonitestimise käigus viiakse harilikult läbi vanu teste, et teha kindlaks, kas varasemalt leitud vead tulevad taas esile.

Kasutatavuse testimine *(Usability testing)* – annab ülevaate kasutajaliidese arusaadavusest, loogikast ja lihtsusest. Kasutatavuse testimine kuulub mitte-funktsionaalsete testide alla.

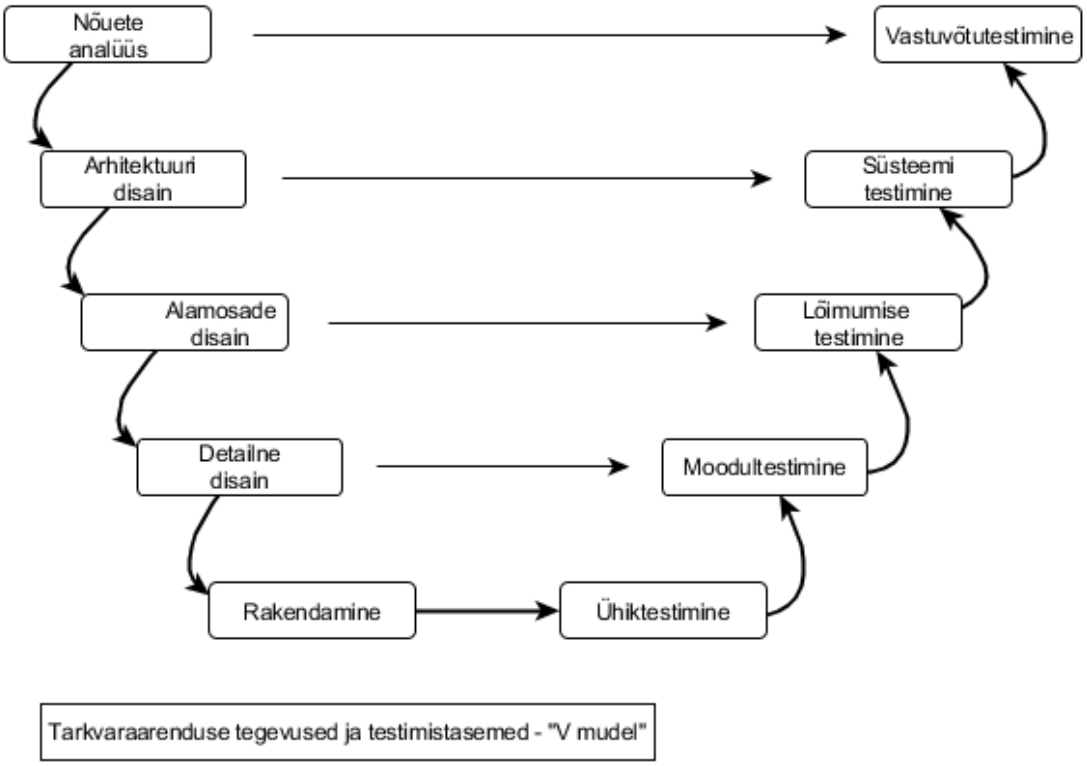
Jõudlus- ja koormustestimine *(Performance testing)* – jõudlustestimise eesmärgiks on välja selgitada, kui kiiresti suudab tarkvara tööd teha kindla koguse kasutajate või andmemahtudega. Selle abil leitakse süsteemi usaldusväärsus ja ressursside kasutamise jagunemine. Koormustestidega kontrollitakse, kas tarkvara suudab kindlal koormusel töötada püsivalt ja tõrgeteta.

Moodultestid *(Module testing)* – moodultestimise korral testitakse ühte alamsüsteemi kogu süsteemitervikust. Testimisel vaadeldakse peamiselt seda, kas mooduli funktsioonid töötavad korrektselt, kuid silmas tuleb pidada ka alamsüsteemi nõuetele vastavust. Moodultestimine tuleb viia läbi enne mooduli integreerimist kogu süsteemi külge.

Koodi ulatuse testimine *(Code coverage testing)* **–** koodi ulatuse testimine annab tagasisidet selle kohta, kui suur osa koodist on testidega kaetud. Näiteks võib luua testi, millega programmi kõiki avaldisi käivitatakse vähemalt üks kord.

Tarkvaraarenduse tegevuste ja testimistasemete V-mudelit peetakse ka koskmudeli edasiarenduseks. V-mudelis on arendus- ja testtegevused paigutatud sümmeetriliselt nii, et igale arendustegevusele vastab sobiv kontrollimisviis. Arendustegevused algavad

üldise nõuete analüüsiga, millele järgneb süsteemi üldise arhitektuuri ning alaosade disaini paikapanek, seejärel kavandatakse detailne disain ja realiseeritakse süsteem. Testtegevused liiguvad vastavalt arendustegevustele, alustades koodipõhiste ühik-testidega, millele järgneb moodul- ja lõimumise testimine. Lõpuks, enne viimast kliendipoolset vastuvõtutesti on kavas süsteemitest nõuetele vastavuse kontrollimiseks.



Joonis 4 – V-mudel [11]

1.5 Automaattestimine

Testide automatiseerimine võimaldab testimise viia läbi kiiremini ja odavamalt. Automaattestimine ei suuda asendada testijat, ent võimaldab testija töö muuta efektiivsemaks. Enamikes arendusmudelites toimub koodikirjutamise, testimise ja parandamise ring mitu korda ja sama asja on vaja testida uuesti. Suurte projektide puhul, kus testskripte on tuhandeid, oleks testide korduv läbimine käsitsi töötades ajaliselt peaaegu võimatu. Siin aitabki testide automatiseerimine, mis võimaldab valmiskoostatud testskripte korduvalt käivitada. [1]

Automaattestimise plussideks võib lisaks aja säästmisele pidada ka efektiivsust ja täpsust. See, et teste jooksutab automaatne süsteem, annab testijale võimaluse samal ajal koostada näiteks uut testi. Automatiseeritud testtööriist ei väsi, teste saab jooksutada ükskõik mis ajal. Samuti väheneb risk, et mõni viga jääb märkamata väsimusest tingitud inimliku eksimuse tõttu, kuna automatiseeritud testid töötavad igal käivitamiskorral identselt. [1]

Automatiseeritud testimisel on ka miinuseid – testskriptid tuleb koostada väga täpselt ja tähelepanelikult. Juhul, kui testskriptis tehakse viga, jooksutatakse skripti korduvalt vigaselt. Testskript peab olema kergelt muudetav – projekti jooksul võib süsteemi lisanduda uusi elemente, võivad muutuda nimed ja parameetrid. Automaattestid kontrollivad ainult seda, mida neil kästakse kontrollida ja ainult automaattestidele lootes võivad märkamata jääda kahtlased kohad süsteemis, mida skripti koostades ei oska kahtlustada, ent mis kogenud testijale silma jääksid. [1]

1.6 Töövahendid

Tarkvara testimiseks on mitmeid erinevaid tehnikaid ja töövahendeid, mida kombineerides on võimalik kohandada igale projektile sobivad testimisvõimalused, mis arvestavad ka testija personaalsete eelistuste ja harjumustega. Testimise automatiseerimiseks on palju erinevate eesmärkide ja funktsioonidega tööriistu, vahendite paljusus võib valiku keeruliseks teha, ent samas tähendab see suuremat paindlikkust, et leida just konkreetsele projektile sobiv lahendus. Leidub nii brauseri lisana toimivaid kui ka arenduskeskkonnapõhiseid töövahendeid. Sõltuvalt platvormist, rakenduse keelest, suurusest ja eesmärkidest on kindlasti võimalik leida automaattestimisvahend, mis testijat abistab. Järgnevalt iseloomustan lühidalt automaattestimisvahendeid, mida on nimetatud seoses tarkvarafirmade hulgas läbiviidud küsitlusega.

Selenium – tasuta avatud lähtekoodiga tööriistade kogum peamiselt veebirakenduste testimiseks. Koosneb järgnevatest osadest - *Selenium IDE, Selenium Remote Control, Selenium WebDriver* ja *Selenium Grid*. *Seleniumi* võimalusi on pikemalt kirjeldatud praktilise ülevaate peatükis. [8]

Mercury – veebipõhine testijuhtimise töövahend, mis võimaldab organiseerida ja hoida ülevaadet nõuetest, testplaanist, testlugudest, käivitatud testide ajaloost ja leitud vigadest. Töövahend võimaldab luua testplaane ja –skripte. [18]

HTTP Test Tool – avatud lähtekoodiga töövahend *HTTP* protokollil baseeruvate veebirakenduste, -serverite ja -brauserite testimiseks. Töövahend suudab käituda nii kliendi kui serverina, andmeid saata ja vastu võtta, jooksutada testskripti. [3]

LabVIEW – töövahend, mis võimaldab testida koodi katmist, viia läbi regressioonitestimist ja ühikteste, samuti genereerida erinevas formaadis raporteid. *LabVIEW* on *Windowsi* ja reaalajaliste operatsioonisüsteemide toega. [9]

Visual Studio Test Professional – testtööriistade kogum, mis võimaldab testida veebirakendusi otse *Visual Studio* keskkonnast. [15]

WATIR – *„Web Application Testing in Ruby“*, *WATIR* on tasuta avatud lähtekoodiga veebipõhine veebirakenduste automatiseerimise kogumik *(libary).* Mis töötab erinevatel platvormidel ja erinevate brauseritega. [10]

DBUnit – töövahend andmebaasiga seotud projektide ühiktestimiseks, võimaldab andmebaasi andmeid eksportida ja importida *XML* andmekoguks. [16]

Firebug – *Firefoxi* lisa, mis võimaldab muuta ja jälgida *CSS, HTML* ja *JavaScripti* koodi ja veateateid kõikidel veebilehtedel, samuti jälgida võrgu aktiivsust, lehe jõudlust ja kontrollida rakenduse turvalisust. [16]

JMeter – *Apache* poolt loodud *Java* töölauarakendus jõudluse testimiseks. Kui algselt loodi *JMeter* ainult veebirakenduste testimiseks, siis nüüd on sellele lisandunud ka teisi testfunktsioone. *JMeter* abil saab simulatsioonina katsetada näiteks serveri käitumist suure koormuse korral, samuti koostada jõudluse kohta graafilist analüüsi. [4]

PHPUnit – töövahend *PHP* jaoks ühiktestide loomiseks. *PHPUnit* töötab erinevatel platvormidel. [16]

NeoLoad – koormustestimise töövahend mobiili- ja veebirakendustele. Võimaldab disainida keerulisi ja reaalelulisi stsenaariume. Toetab erinevaid platvorme ja tehnoloogiaid (*AJAX, FLEX, Java Serialization*). [16]

SoapUI – tasuta vabavaraline töölauarakendus veebirakenduste (*java, .net*) testimiseks, jälgimiseks ja arendamiseks. Võimaldab testida jõudlust ja koormustaluvust. [16]

TestMaker – töövahend veebirakenduste jõudluse ning funktsionaalse testimise jaoks. Töötab kõikidel platvormidel, võimaldab integreerida teisi testvahendeid (*Selenium, SoapUI, HTMLUnit*) ja käivitada dünaamilistes skriptimiskeeltes (*Java, Jython, HRuby*) kirjutatud ühikteste. [16]

W3C – süntaksi kontrollimise töövahend, mis kontrollib veebilehtede ja dokumentide märgistuskeele kehtivust ja vastavust. [20]

2. Uurimus tarkvarafirmade hulgas

2.1 Taust

Eesti tarkvarafirmade testimisharjumuste väljaselgitamiseks viidi vahemikul 13.02.2012 - 16.03.2012 läbi veebiküsitlus justask.ee keskkonnas. Küsitluse blankett on lisatud töö lõppu. Uuringus paluti osaleda 40-l erineva suurusega firmal, kelle töövaldkonnaks on tarkvara arendus. Uuringu tulemusena saadi tagasisidet 22-lt firmalt, kelle vastustel järgnev analüüs põhineb. Küsitluses uuriti testimise mahtu tarkvaraprojektis, testimismeetodite- ning vahendite kasutamist ning automaattestimisvahendite rakendamist. Uuringu eesmärgiks on võrrelda testimisvahendite kasutamise mahu, erinevate testimeetodite ja –vahendite kasutamise erinevust eri suurusega ettevõtetes. Ettevõtted on töötajate arvu põhiselt jaotatud kolme suurusgruppi – kuni 10 töötajat, 11 kuni 30 töötajat ning rohkem kui 30 töötajat. Samuti on eesmärgiks saada infot, mille jaoks tarkvarafirmad automaattestimist kasutavad.

2.2 Analüüs

Uuringus osalenud 22-st ettevõttest 4 on väikese suurusega, ehk kuni 10 töötajaga, 5 ettevõtet keskmise suurusega ehk 11 kuni 30 töötajaga ning ülejäänud 13 suured, ehk enam kui 30 töötajaga.

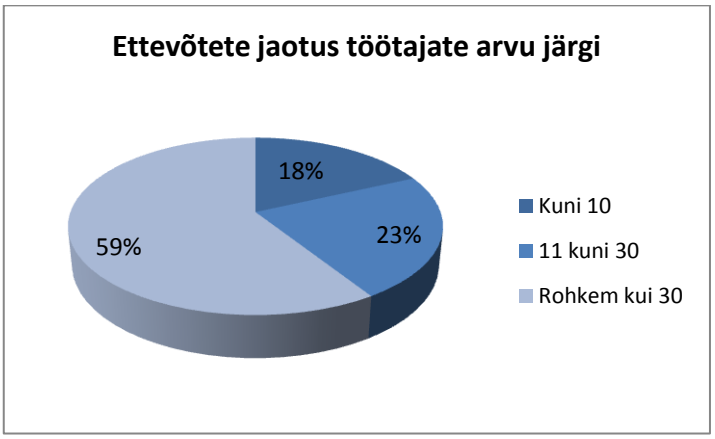


Diagramm 1 – Ettevõtete jaotus töötajate arvu järgi

12 ettevõtte puhul on projektimeeskonna suuruseks tavaliselt kuni 5 inimest ja 10 ettevõtte puhul 6 kuni 10 inimest. Neljal juhul 22-st testib koodi selle kirjutaja, 7-e ettevõtte puhul ei ole eraldi testijat, ent koodi testib siiski keegi teine peale selle kirjutaja ning 11-s ettevõttes on projektimeeskonnas eraldi testija. Väikeste ning keskmise suurusega ettevõtete puhul ei ilmne märgatavat erinevust eraldi testija olemasolu või puudumise suhtes. Suurte ettevõtete puhul on aga eraldi testija olemas 9-l juhul 13-st, seejuures ise testitakse enda koodi ainult ühes suures ettevõttes.

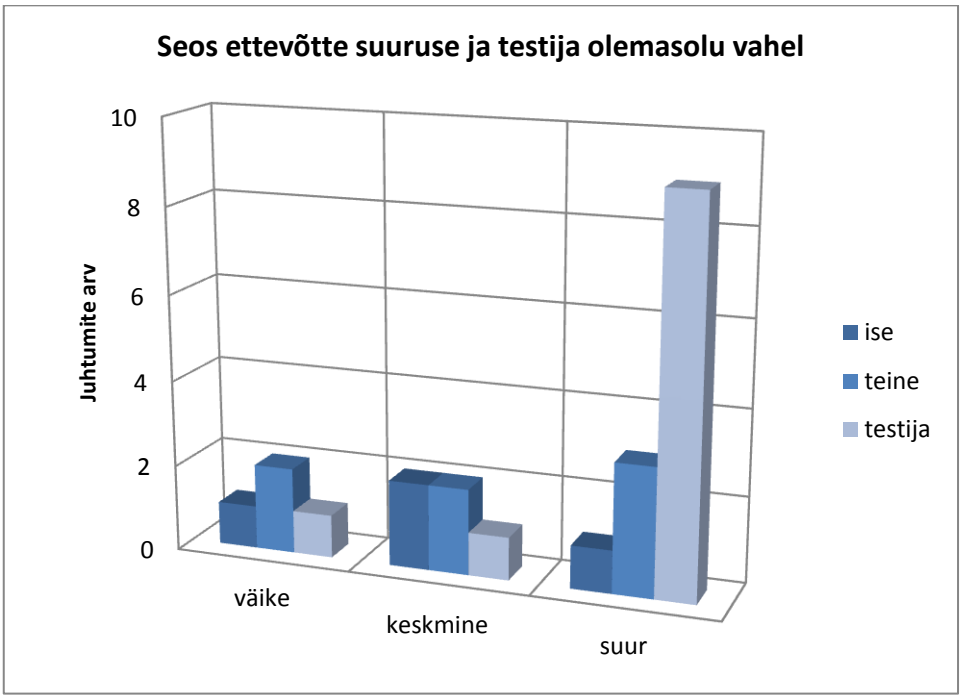


Diagramm 2 – Seos ettevõtte suuruse ja testija olemasolu vahel

Testimise mahtu tarkvaraprojektis hinnati vahemikes kuni 15%, 16 kuni 30% ning rohkem kui 30%. Selgus, et kõige rohkem testitakse vahemikus 16 kuni 30% tarkvaraprojekti mahust – 12-l juhul 22-st. Rohkem kui 30% tarkvaraprojekti mahust kulutatakse testimisele ainult 2 ettevõttes, kus mõlemas tegeleb testprotsessiga eraldi testija. Antud suurust võrreldakse eraldi testija olemasoluga. Võime öelda, et kui testib keegi teine peale koodi kirjutaja, siis on testimisele määratud töömaht suurem. See on ka loogiline tulemus, sest omakirjutatud koodi on keeruline värske pilguga ning erineva nurga alt vaadata. Samuti, kui testimiseks ettenähtud maht moodustab tarkvaraprojekti tervikust olulise osa, on ettevõttel mõttekas palgata eraldi testija, kellel on vastavad kogemused või võimalused ja motivatsioon end harida just testimise valdkonnas, ilma enda tööalaseid huve killustamata.

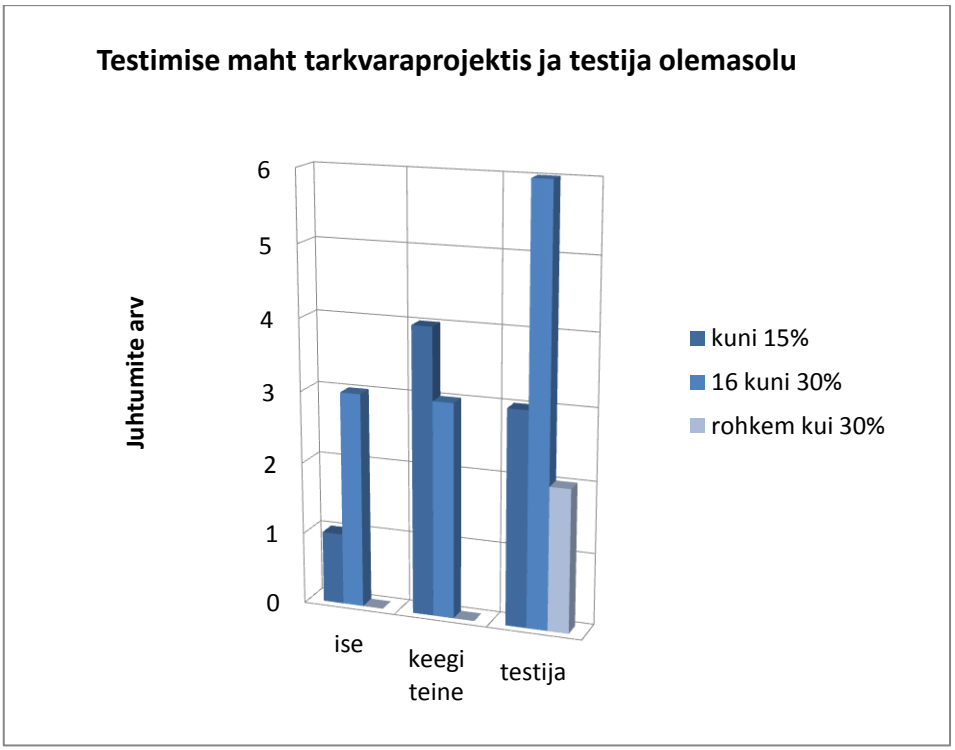


Diagramm 3 – Testimise maht tarkvaraprojektis ja testija olemasolu

Testimismeetodite puhul kasutavad 13 ettevõtet mõlemat, nii musta kui ka valge kasti testimist. 5 ettevõtet kasutavad ülekaalukamalt musta kasti meetodit ja 4 valge kasti meetodit. Musta kasti meetodiga testijad kulutavad testimisele projektimahust ühel juhul rohkem kui 30% ja ühel juhul vähem kui 15%, kolmel juhul aga 16 kuni 30%. Valge kasti meetodil testijad aga kahel juhul kuni 15% ning kahel juhul 16 kuni 30%. Märgatavat seost testimise mahu ja testimismeetodi vahel ei ilmnenud.

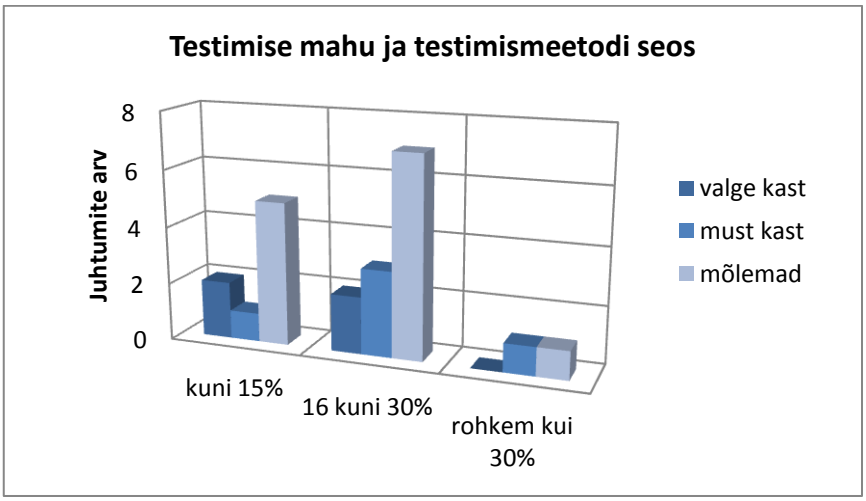


Diagramm 4 – Testimise mahu ja testimismeetodi seos

Valge kasti meetodit eelistavad üks väike, üks suur ja kaks keskmise suurusega ettevõtet. Musta kasti meetodit eelistavad eranditult suured ettevõtted, millest 3-l on eraldi testija ning 2-l testib keegi teine peale programmeerija. Ise enda koodi testides kasutatakse kahel juhul valge kasti meetodit ja kahel juhul mõlemat meetodit, valge kasti meetodit aga ise koodi testides ei kasutata. Seevastu eraldi testijate puhul on eelistatuim just musta kasti testimine. See on testimismeetodite olemust arvestades loogiline tulemus, sest musta kasti tehnika eeldab intuitiivsust ja kogemust, mida omab spetsialiseerunud testija ning koodi kirjutajal endal on lihtsam teha koodipõhiseid teste.

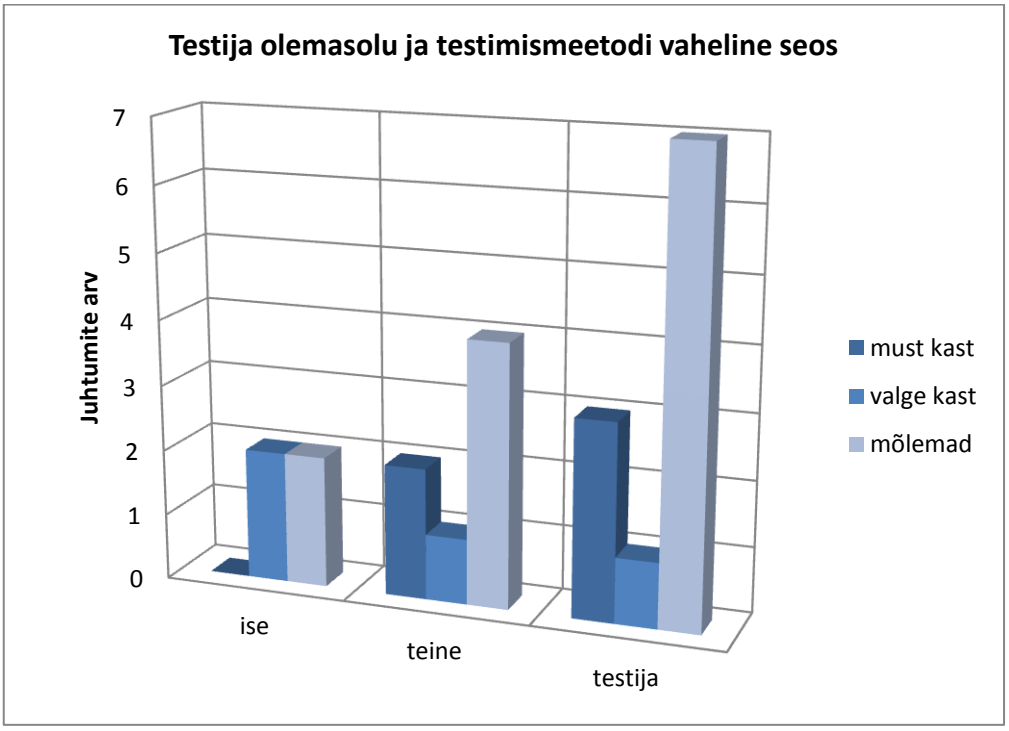


Diagramm 5 – Testija olemasolu ja testimismeetodi vaheline seos

Kasutatavatest testimisvahenditest leiab ülekaalukalt enim kasutust *Selenium*, mida kasutab 11 ettevõtet. Vähem, ent siiski vähemalt kolme ettevõtte poolt kasutatakse ka vahendeid *Watir* ja *Jmeter*. Kahe ettevõtte poolt kasutatakse ka vahendeid *SoapUI* ning *Visual Studio Test Professional*. Töövahendit *LabVIEW* vaadeldud ettevõtetes ei kasutata ning kõiki ülejäänud testimisvahendeid kasutab valikus olevatest ettevõtetest üks.

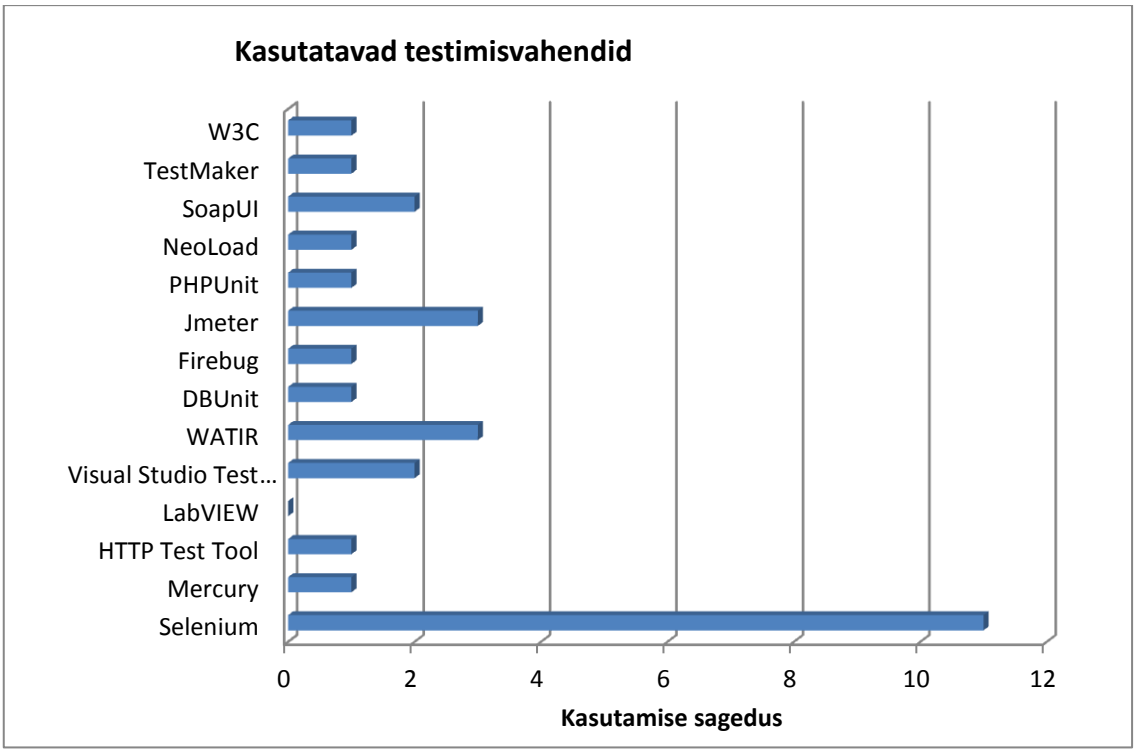


Diagramm 6 – Kasutatavad testimisvahendid

Automaatteste kasutavad 22-st vastanud ettevõttest 14, neist 2 viivad automaattestimist kasutades läbi rohkem kui 60% testimise mahust, 1 ettevõte kasutab automaattestimist 31 kuni 60% mahus ja ülejäänud 11 kasutavad automaattestimist kuni 30% ulatuses projekti testimise üldmahust. Automaattestimise kasutamine ükskõik millises mahus langeb märgatavalt juhul kui testimise maht tarkvaraprojektis on suurem kui 30%. Juhtumitel, kus testimise osakaal tarkvaraprojekti mahust on kuni 15% või 16 kuni 30% väga suurt erinevust ei ilmne – automaattestimist kasutatakse mõlemal juhul kõige rohkem kuni 30% ulatuses üldisest testimismahust.

Kuna automaattestimine võimaldab testimisele kulutatavat aega ja tööd vähendada, siis tulemus, et suurte testimismahtude puhul on automaattestimise kasutus minimaalne, on loogiline.

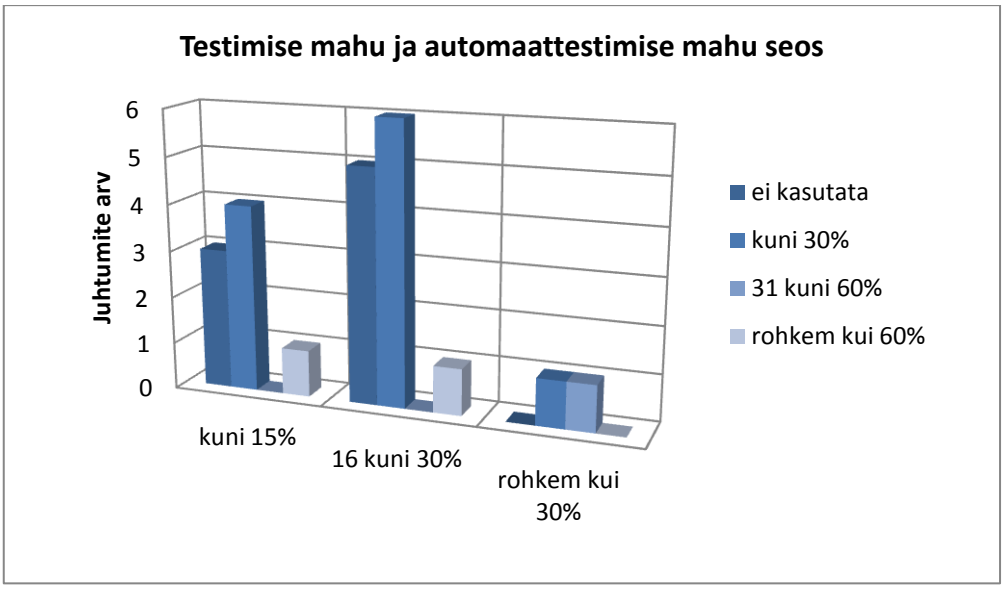


Diagramm 7 – Testimise mahu ja automaattestimise mahu seos

22-st ettevõttest kaheksa ei kasuta automaattestimist. Kasutust leiab automattestimine enim jõudlus- ja koormustestimise jaoks 13-l juhul 22-st, populaarseks võib pidada ka rakendusliidese ning süsteemi nõuetele vastavuse kontrolli, vähim kasutust leiab see kasutatavuse testimisel, vaadeldud andmete korral ainult 2-l juhul.

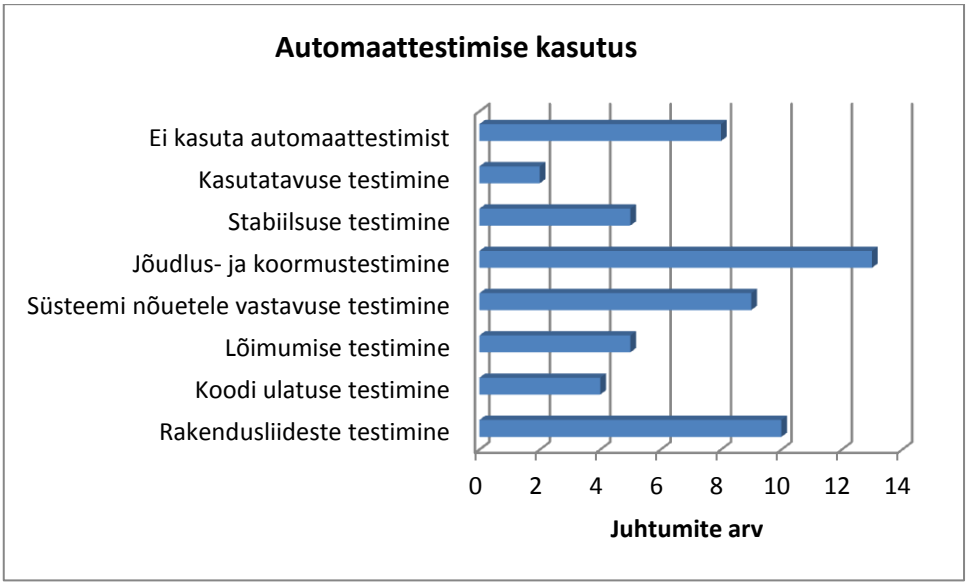


Diagramm 8 – Automaattestimise kasutus

2.3 Järeldused

Vaadeldud 22 ettevõtte tulemused andsid pinnapealse ülevaate Eesti tarkvarafirmade testimisharjumustest. Meeldiv üllatus oli fakt, et ainult neljas ettevõttes testib koodi selle kirjutaja ning 18 ettevõtte puhul teeb seda kui mitte eraldi testija, siis vähemalt keegi teine. Samuti on positiivne, et 12 ettevõtet kulutavad testimisele 16 kuni 30% tarkvaraprojekti mahust ja 2 ettevõtet isegi rohkem kui 30%. Testimismeetodite ja kasutatavate töövahendite valik sõltub kindlasti ka ettevõtte täpsemast tegevussuunast, kuid neid andmeid antud küsitluse raames ei kogutud ega uuritud.

Ilmnes, et musta kasti testimist eelistatakse eraldi testijate puhul ja valge kasti testimist eelistavad oma koodi ise testijad. Samuti on näha seos, et suuremate testimismahtude korral on ettevõttes tõenäolisemalt ka eraldi testija. Rohkem kui pooled – 14 ettevõtet 22-st kasutavad oma tegevuses ka automaattestimist, neist enamik, ehk 11 vahemikus kuni 30%, üks vahemikus 31 kuni 60% ja kaks isegi enam kui 60%. Ülekaalukalt populaarseimaks automaattestimisvahendiks valiti *Selenium*, mida kasutab 50% küsitluses osalenud ettevõtetest. Automaattestimisvahendid on enim kasutusel jõudlus- ja koormustestimise, rakendusliidese ja süsteemi nõuetele vastavuse kontrollimiseks.

3. Populaarseim automaattestimise töövahend Eesti ettevõtetes

3.1 Selenium

Automaattestimisest praktilisema ülevaate saamiseks on katsetasin küsitluse tulemusena kõige populaarsemat automaattestimisvahendit *Selenium*. [6], [8]

Seleniumi kasutusalaks on peamiselt veebirakenduste automaattestimine. *Selenium* võimaldab lindistamise/taasesitamise võimalust, mis lubab *Firefoxi* lisana implementeeritud *Selenium IDE*'t kasutades koostada teste ilma testskripti keelt valdamata. Lisaks brauseripõhisele testikoostamisele kuulub *Seleniumi* alla ka spetsiifiline testkeel *Selenese*, mille abil on võimalik koostada skripte enamlevinud programmeerimiskeeltes, näiteks *Java, C#, Perl, PHP, Python, Ruby* ning valmis skripte seejärel erinevate veebibrauserite vastu jooksutada. *Selenium* koosneb erinevatest komponentidest.

Selenium IDE – täielik integreeritud arenduskeskkond *Seleniumi* testide jaoks, mis on rakendatud *Firefoxi* laiendusena *(extension). Selenium IDE* võimaldab salvestada kasutajapoolsed tekstisisestused, nupuvajutused ja linkide avamised. Lisaks kasutajapoolsetele tegevustele saab testjuhtumisse lisada ka laiendusepoolsed käsud, näiteks teksti või elemendi salvestamise.

Selenium Core – raamistiku tuum, mis integreerib testid veebilehitsejasse kasutades *JavaScripti* ja *HTMLi*.

Selenium Remote Control – server, mis vahendab teistes keeltes kirjutatud *Seleniumi* teste erinevate brauseriga. Klientdraiverid on olemas enamlevinud programmeerimis-keeltele (*Java, Python, Perl, PHP, Ruby, .NET*).

Selenium WebDriver – Selenium Remote Control järeltulija, mis võimaldab *Selenese* või klient API käske vastu võtta ja saadab need brauserile.

Selenium Grid – server, mis võimaldab teste käivitada paralleelselt või erinevates masinates ning erinevate brauseri versioonide ja seadistustega.

Selenium IDE	*Selenium Remote Control*	*Selenium WebDriver*
Töötab ainult *Mozilla Firefoxiga*	Töötab peaaegu kõikide brauseritega, välja arvatud uusimad *Internet Exploreri* ja *Firefoxi* versioonid	Töötab peaaegu kõikide uusimate brauseritega
Lindistamise ja taasesitamise tööriist	Lindistamist ja taasesitamist ei ole	Lindistamist ja taasesitamist ei ole
Ei vaja serverit	Vajab serverit	Ei vaja serverit
Tuum on *JavaScripti* baasil	Tuum on *JavaScripti* baasil	Suhtleb brauseriga
Väga lihtne kasutada	Väike ja lihtne API	Keerukas ja suur API
Pole objekt-orienteeritud	Vähesel määral objekt-orienteeritud API	Puhtalt objekt-orienteeritud API
Ei võimalda testida *Iphone* ega *Androidi* rakendusi	Ei võimalda testida *Iphone* ega *Androidi* rakendusi	Võimaldab testida *Iphone* ega *Androidi* rakendusi

Tabel 1 – *Seleniumi* komponentide võrdlus [7]

3.2 Praktiline tutvustus

Praktilisest kasutusest kerge ülevaate saamiseks katsetasin kõige lihtsamat viisi *Seleniumi* kasutamiseks – *Firefoxi* laiendusena töötavat *Selenium IDE't*. Testin lühidalt kasutajapoolsete tegevuste ning laiendusepoolsete käskude salvestamist ja taasesitamist Tartu Ülikooli koduleheküljel *www.ut.ee*.

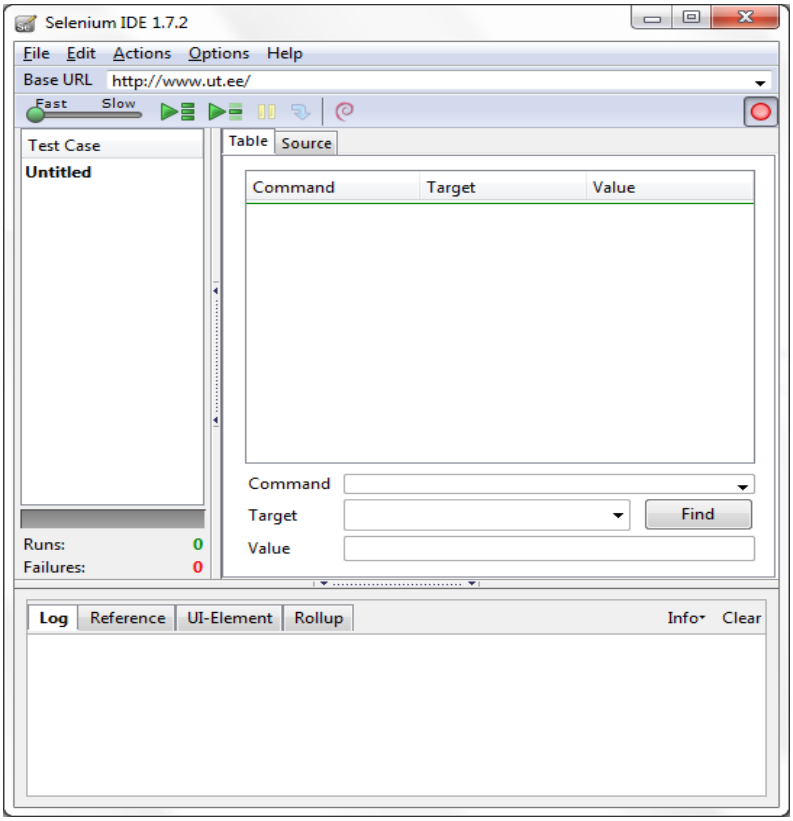
3.2.1 Selenium IDE

Selenium IDE on tasuta ja selle installeerimine on väga lihtne. Kõik *Seleniumi* osad on kättesaadavad *Seleniumi* kodulehelt *http://seleniumhq.org*. *Selenium IDE* avaneb uue brauseri aknana *Firefoxi* menüüst tööriistad *(tools) – Selenium IDE*.

Selenium IDE aken on loogiline, minimalistlik ja selge ülesehitusega. Kasutajale pakutakse võimalust salvestada nii üksikut testskripti *(test case)* kui ka testskriptide komplekti *(test suite)*. Mõlemat saab ka erinevate keelte formaatidesse eksportida. Kasutajal on nupud salvestamise alustamiseks/lõpetamiseks ning testskripti või testide komplekti taaskäivitamiseks.

Testskript koosneb käskudest, mis omakorda koosnevad kolmest osast: käsk *(command)*, siht *(target)* ja väärtus *(value)*. Väärtuse veerg võib jääda tühjaks, ent igal

käsul on alati olemas siht, selleks võib olla link, tee, tabel või konkreetne tekst. Käske ja sihte näeme täpsemalt järgnevates näidetes. Testskripti sisse saab lisaks käskudele lisada ka kommentaare.



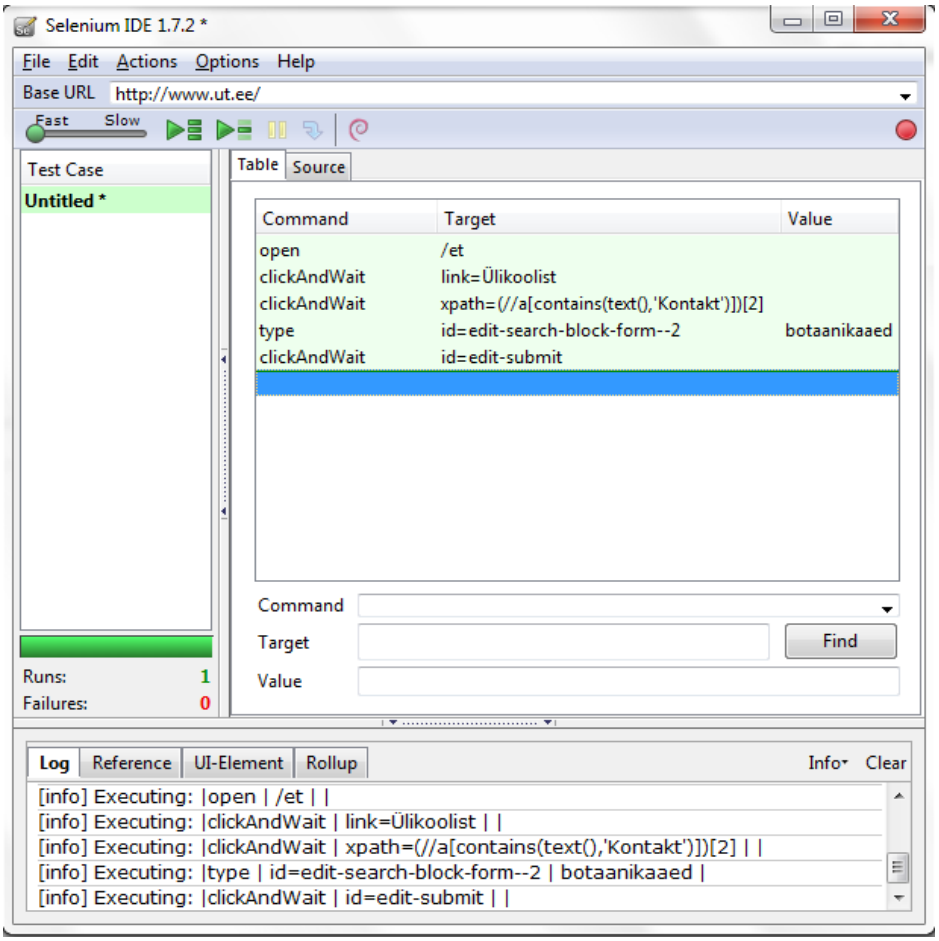
Joonis 5 – *Selenium IDE* aken

3.2.2 Kasutaja tegevuste salvestamine ja taasesitamine

Alustades lindistamist veebilehelt *google.ee* avan ülikooli kodulehe aadressilt *www.ut.ee*, avan järgnevad alamenüüd: ülikoolist, kontakt, seejärel sisestan üleval paremas nurgas olevasse otsingukasti sõna „botaanikaaed“ ja vajutan otsingunupule, peale lehe avamist lõpetan lindistamise. Salvestatud tegevusteks/käskudeks olid linkidel klikkimised, avamine ja tippimine. Salvestatud testskripti uuesti käivitades läbib brauser ilma kasutajapoolse sekkumiseta probleemideta kõik salvestatud käsud, kaasaarvatud sisestatud sõna järgi otsimise.

Kasutajale kuvatakse logi vastavalt soovile nii info, silumise, hoiatuste kui veateadete kaupa. Samuti on kasutajal võimalik saada infot käsu kohta kasutades akent viide

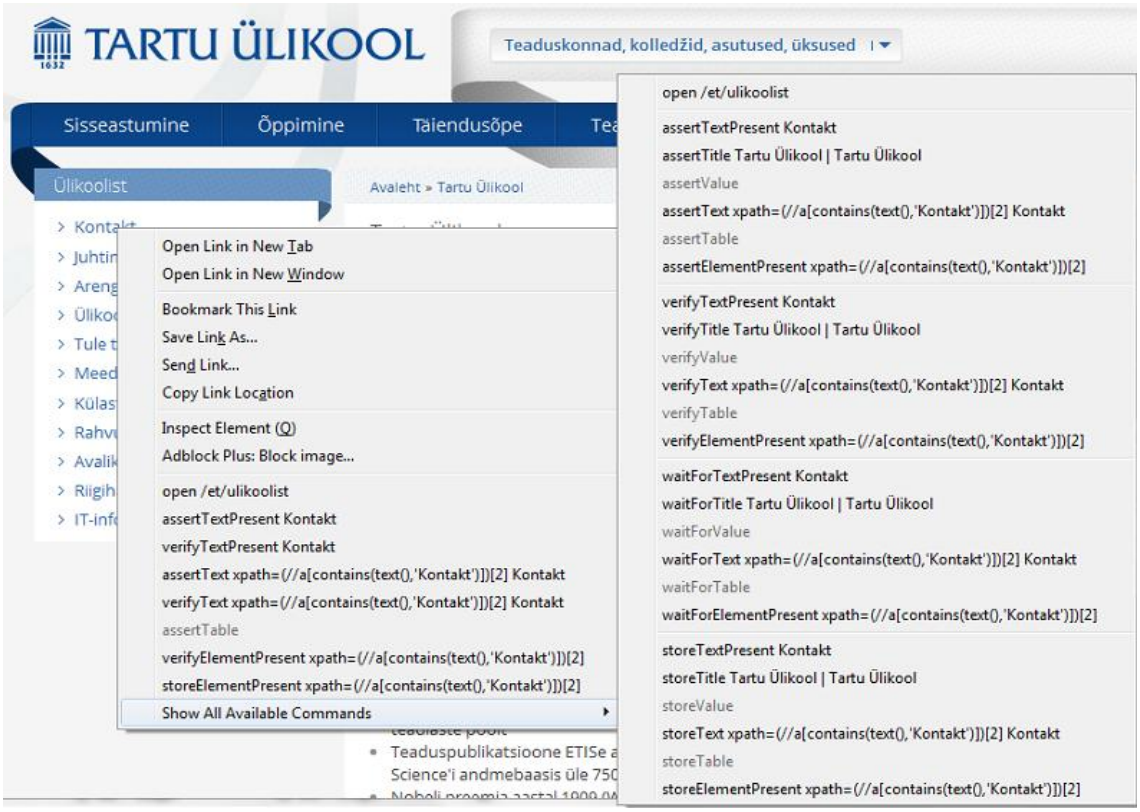
(reference). Valmis testskripti on võimalik muuta nii üksikut käsku eemaldades kui ka suvalisse kohta uut lisades.



Joonis 6 – Taaskäivitatud kasutaja tegevuste põhine testskript

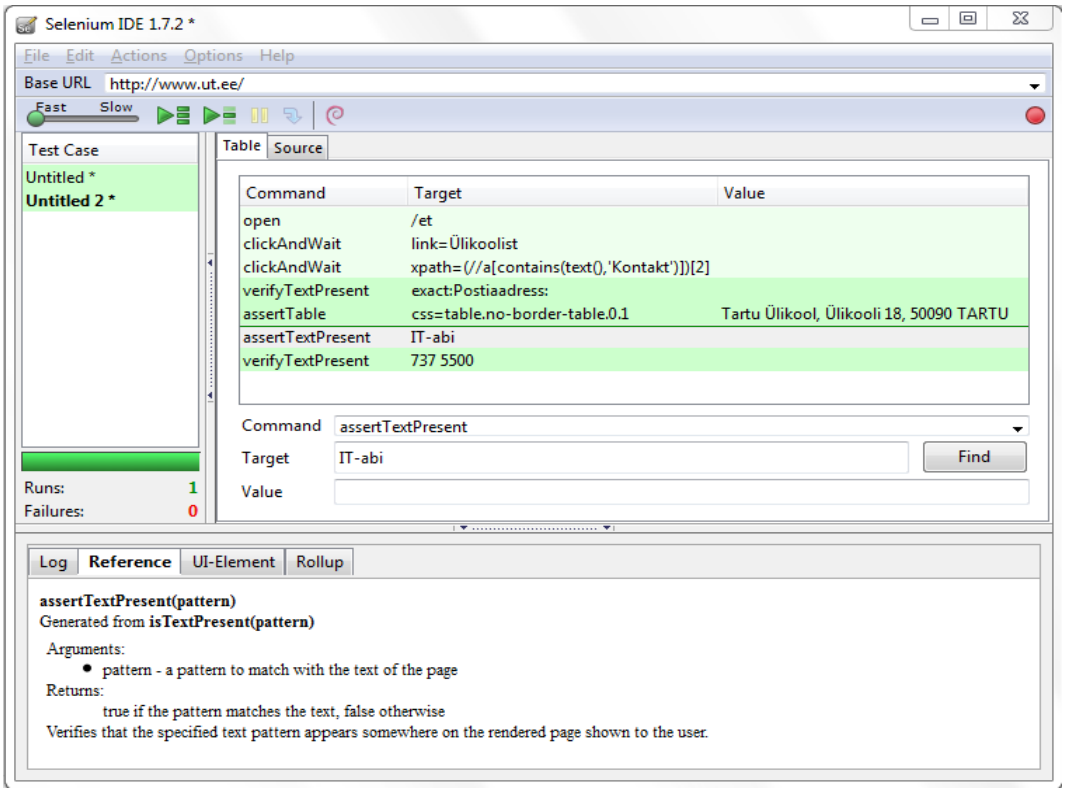
3.2.3 Laiendusepõhiste käskude salvestamine ja taasesitamine

Lisaks kasutaja tegevuse jälgimisele võimaldab *Selenium IDE* lisada testskriptile laiendusepõhiseid käske. Käsud jagunevad nelja suuremasse gruppi: kinnitada *(assert)*, kontrollida *(verify)*, salvestada *(store)*, oodata *(waitFor)*, samuti on olemas ka käsk avada *(open)*. Neljale põhikäsule lisanduvad valitavad täpsustused, kas sooritada toiming pealkirja, teksti olemasolu, väärtuse, lingi või elemendi kohta. Käsu valimiseks tuleb teha paremklõps soovitud elemendil ja liikuda valikus *Seleniumi* käskudeni. Erinevate elementide puhul lastakse kasutajal valida erinevate käsuvalikute hulgast.



Joonis 7 – *Selenium IDE* laiendusepõhised käsud

Alustasin lindistamist jällegi *google.ee* lehel olles ja avasin ülikooli kodulehe. Valisin menüüst ülikoolist ja kontaktid, seejärel asusin erinevate laiendusepõhiste käskudega kontrollima ülikooli postiaadressi ja IT-abi telefoninumbri väärtust ja olemasolu. Käsuga `verifyTextPresent exact:Postiaadress` kontrollin, kas valitud tekst leidub lehel. `AssertTable`, mille väärtuseks on ülikooli postiaadress, tagastab väärtusena valitud tabelilahtri sisu ja kontrollib selle väärtust. `AssertTextPresent IT-abi` kontrollib, et valitud tekst leiduks lehel.



Joonis 8 – Taaskäivitatud laiendusepõhiste käskude testskript

3.2.4 Valmistehtud skripti uurimine ja muutmine

Vaikimisi kuvatakse koostatud skripti tabelivaates, mida on näha ka eelnevates näidetes, ent *Selenium IDE* võimaldab skripti vaadata ja muuta ka koodi kujul (*source*). Lisaks laiendusepõhisele tegevusele on võimalik pisut *HTML* teadmisi omades vajalike käskudega testskript ise kasvõi nullist koostada, see siis *Selenium IDE´*ga avada ning jooksutada. Järgnevalt on lõik eelmises näites läbitud käskudest koodi kujul.

```
...
<body>
...
<tr>
	<td>clickAndWait</td>
	<td>xpath=(//a[contains(text(),'Kontakt')])[2]</td>
	<td></td>
</tr>
<tr>
	<td>verifyTextPresent</td>
	<td>Postiaadress</td>
	<td></td>
</tr>
...
```

Koodikujul skripti koostamine on peale mõne näitega tutvumist lihtne ja võimaldab testskripti koostamise kiiremaks ja põhjalikumaks muuta. Just koodipõhist kuju jälgides ilmneb *Seleniumi* võimaluste paljusus. Kuigi laiendusesiseselt kuvatakse kasutajale *HTML* kujul testskripti, siis üksiku skripti eksportimise võimalusi on lausa 17 eri formaadi (näiteks *PHPUnit, Perl, C# WebDriver, C# Remote Control, Junit 4 WebDriver*) jaoks, sealhulgas erinevate *Seleniumi* komponentide jaoks sobivatele kujudele.

3.3 Järeldused

Seleniumi populaarsuse põhjuseks võib pidada rohkeid komponente, mis annavad kasutajale palju võimalusi testida erinevates keeltes kirjutatut eri platvormide ja brauserite kaudu. *Selenium* pakub võimalusi väga erinevate raskusastmetega testide loomiseks ja käivitamiseks. Vaadeldud töövahend võimaldab läbi laienduse koostada nii väga lihtsaid kui ka edasijõudnutel suuremaid ja põhjalikumaid koodipõhiseid skripte, rääkimata praktilisest katsetusest välja jäänud komponentidest, mis võimaldavad tarkvara testimist professionaalsel tasemel. *Selenium IDE-ga* tutvumine ja selle kasutamine peaksid olema jõukohased igale tarkvaraarendusega kokkupuutuvale inimesele. Automaattestimisvahendite kasutamisele tuleks pöörata rohkem tähelepanu ja miks mitte neid õppeprogrammis ka üliõpilastele tutvustada.

Kokkuvõte

Bakalaureusetöö eesmärgiks oli rõhutada testimise olulisust tarkvaraprojektis, näidata erinevaid viise testimiseks ja tutvustada veidi lähemalt automaattestimise võimalusi. Andsin ülevaate erinevatest tarkvaraarenduse mudelitest, testimismeetoditest ja -liikidest ning vahenditest. Lisaks kirjeldasin automaattestimise olemust ning selle plusse ja miinuseid.

Samuti viisin lisaks teoreetilisele ülevaatele ja ühe töövahendi praktilisele tutvustusele läbi küsitluse Eesti tarkvarafirmade hulgas. Uuringu eesmärgiks oli saada infot kasutatavate testimismeetodite, -vahendite ja eesmärkide kohta, samuti ka testija olemasolu ja projektimeeskonna suuruse ja nendevaheliste seoste olemasolu kohta.

Uuringu tulemusena selgus, et automaattestimist kasutatakse rohkem kui pooltes 22-st küsitluses osalenud ettevõtetes. Veidi enam kui pooltel juhtudel testitakse rohkem kui 16% võrra tarkvaraprojekti üldmahust. Samuti ilmnes, et enamikel juhtudel viib põhjalikuma testimise läbi siiski keegi teine peale koodi kirjutaja. Testimise kasutamise osakaalu, eesmärkide ja erinevate töövahendite kasutamise osas on kindlasti ruumi arenguks.

Automaattestimisvahendite valik on lai ning pakub väga erineval tasemel töövahendeid. Automattestimisvahendi *Selenium* praktiline katsetamine näitas, et automatiseeritud testimisega lihtsal tasemel alustamine ei nõua spetsiaalseid süvendatud teadmisi ja on jõukohane tarkvaraarendusega kursis olevatele inimestele. Testide automatiseerimine võiks olla veel populaarsem ning sellele tuleks kindlasti tähelepanu juhtida.

The Practical Owerview and Use of Automated Testing Tools on the Example of Selenium

Meeli Pällin

Bachelor Thesis

Abstract

The aim of this Bachelor thesis was to emphasize the relevance of testing in a software project, to demonstrate different methods of testing, and to provide a brief introduction to the possibilities offered by automated testing. In the thesis, I gave an overview of different software development models, methods and types of testing, and testing tools. And described the nature of automated testing and its pros and cons.

In addition to the theoretical overview and the practical introduction of one of the automated testing tools, I conducted a survey among Estonian software companies. The goal of the survey was to collect information about the used testing methods, tools, and objectives as well as about the existence of a tester, the size of a project team, and the connections between them.

According to the results, more than half of the 22 companies who participated in the survey use automated testing. In slightly more than half the cases, the companies test over 16% of the total size of the software project. In addition, it appears that in most cases, the more thorough testing is conducted by someone other than the person who wrote the code. The extent of testing, the objectives of testing, and the use of different testing tools can surely be improved.

The selection of automated testing tools is wide and suited to many different levels. The practical use of *Selenium* showed that using automated testing on an elementary level is feasible for people who are familiar with software development and requires no specialist knowledge. The automation of testing could and should be more popular and it is certainly a topic that needs more attention.

Kasutatud materjalid

[1] Patton, R. Software testing. Sams Publishing 2001, 390.

[2] Software testing overview (07.05.2012)
http://en.wikipedia.org/wiki/Software_testing

[3] HTTP Test Tool (05.05.2012)
http://htt.sourceforge.net/cgi-bin/cwiki/bin/public

[4] Apache JMeter (05.05.2012)
http://jmeter.apache.org/

[5] Spiraalmudeli pilt (10.05.2012)
http://leansoftwareengineering.com/2008/05/05/boehms-spiral-revisited/

[6] Seleniumi tutvustus (07.05.2012)
http://qtpselenium.com/

[7] Seleniumi komponentide võrdlus (07.05.2012)
http://qtpselenium.com/selenium-tutorial/difference-between-ide-rc-webdriver/

[8] Seleniumi koduleheküIg (07.05.2012)
http://seleniumhq.org/

[9] LabVIEW tutvustus (05.05.2012)
http://sine.ni.com/nips/cds/view/p/lang/en/nid/209043

[10] WATIR koduleheküIg (05.05.2012)
http://watir.com/

[11] V-mudeli pilt (10.05.2012)
http://www.certitudo-gmbh.de/bilder/v_model_e.gif

[12] Petuhhov, I. Loengumaterjal. Tarkvaraprotsess, üldised protsessimudelid. 2008. (05.05.2012)
http://www.cs.tlu.ee/~inga/SE_materjal/Tarkvara_protsess_2008.pdf

[13] Pan, J. Software Testing (05.05.2012)
http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/

[14] Testimistüübid ja –tasemed (06.05.2012)
http://www.exforsys.com/tutorials/programming-concepts/types-and-levels-of-testing-in-programming.html

[15] Visual Studio Test Professional 2010 ülevaade (05.05.2012)

http://www.microsoft.com/visualstudio/en-us/products/2010-editions/test-professional/overview

[16] Testimisvahendite ülevaade (06.05.2012)
http://www.softwareqatest.com/qatweb1.html

[17] Koskmudeli ülevaade (04.05.2012)
http://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/6118423

[18] Mercury ülevaade (05.05.2012)
http://www.testinggeek.com/mercury-quality-centre-introduction

[19] Testimistüübid (04.05.2012)
http://www.the-software-experts.de/e_dta-sw-training-test.htm

[20] W3C töövahendi ülevaade (05.05.2012)
http://www.w3.org/Graphics/SVG/WG/wiki/Test_Suite_Overview

[21] Koskmudeli pilt (04.05.2012)
http://www.waterfall-model.com/

Lisa 1 – Küsitluse blankett

Uuring tarkvarafirmade seas.

1. Kui palju on firmas töötajaid?
    a. Kuni 10
    b. 11 kuni 30
    c. Rohkem kui 30

2. Keskmiselt kui suur on ühe tarkvaraprojektiga tegeleva meeskonna suurus.
    a. Kuni 5 inimest
    b. 6 kuni 10 inimest
    c. Rohkem kui 10 inimest

3. Kas projekti meeskonnas on tavaliselt liige, kelle ainsaks ülesandeks projektis on testimine?
    a. Jah, projektimeeskonnas on eraldi testija/testijad
    b. Ei, igaüks testib enda kirjutatud koodi ise
    c. Ei, eraldi testijat pole, ent koodi testib siiski keegi teine peale selle kirjutaja
    d. Mitte ükski eelnev variant

4. Protsendiliselt kui suur maht tarkvaraprojekti tööst kulutatakse testimisele.
    a. Kuni 15%
    b. 16 kuni 30%
    c. Rohkem kui 30%

5. Milliseid testimismeetodeid teie ettevõttes enim kasutatakse?
    a. Valge kasti meetod
    b. Musta kasti meetod
    c. Mõlemaid

6. Milliseid testimisvahendeid teie ettevõttes peamiselt kasutatakse?

a. Selenium
b. Mercury
c. HTTP Test Tool
d. LabVIEW
e. Visual Studio Test Professional
f. WATIR
g. DBUnit
h. Mitte ühtegi eelnevatest

7. Juhul, kui eelmises küsimuses ei olnud Teie poolt kasutatavaid testimisvahendeid, siis palun lisage need.
    a. Vaba vastusega küsimus

8. Keskmiselt kui suur protsent testimisest viiakse läbi automaatteste kasutades.
    a. Kuni 30%
    b. 31 kuni 60%
    c. Rohkem kui 60%
    d. Automaatteste ei kasutata

9. Mille jaoks teie ettevõttes automaattestimist kasutatakse?
    a. Rakendusliideste testimine
    b. Koodi ulatuse testimine
    c. Lõimumise testimine
    d. Süsteemi nõuetele vastavuse testimine
    e. Jõudlus- ja koormustestimine
    f. Stabiilsuse testimine
    g. Kasutatavuse testimine
    h. Ei kasutata automaattestimist