Tallinna Ülikool
Digitehnoloogiate Instituut

Automaattestide koostamine Robot Framework raamistikul pangandustarkvara näitel

Bakalaureusetöö

Autor: Dea Taur
Juhendaja: Inga Petuhhov

Autor: 2017
Juhendaja: 2017
Instituudi direktor: 2017

Tallinn 2017

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....	.....
(kuupäev)	(autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Dea Taur** (sünnikuupäev: **04.06.1991**)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose **„Automaattestide koostamine Robot Framework raamistikul pangandustarkvara näitel“**, mille juhendaja on **Inga Petuhhov** säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas, 03. mail 2017

Sisukord

Sissejuhatus

Testimine on loogiline osa iga tarkvara arendusprojektis. Mida spetsiifilisem ja suuremamahulisem on teostatav projekt, seda keerulisem ja ajakulukam on ka kaasnev testimine ning pidev kvaliteedi tagamine. Uue funktsionaalsuse lisamine tarkvarasse on seotud teatud riskidega. Et arendusmeeskonnad seavad endale eesmärgiks toimetada tarkvara täiendused kliendini kiirelt ja muretult, siis kerkib üles küsimus – kuidas muuta arendustsükkel kliendi tellimusest kuni selle üle andmiseni senisest kiiremaks kaotamata sealjuures toote kvaliteedis? Üks võimalikest vastustest peitub näiteks automaattestimise kasutamises.

Suurte süsteemide regressioonitestimist toetab mitmekülgne automaattestide kasutamine arendusprotsessis. Automaattestide olemasolu annab võimaluse igal ajahetkel kiiresti veenduda, et kriitiline osa tarkvarast ei ole saanud viga uuenduste tegemisel. Selliste testide loomine võib esialgu tunduda kulukam kui manuaalne testimine, ent ometi on ka märkimisväärseid kasutegureid, mis võivad üles kaaluda esialgse investeeringu. Näiteks on loodud testide läbimine kordades kiirem, kui suudaks seda teha ükski testija ning juba koostatud teste saab korduvalt kasutada erinevates keskkondades ja konfiguratsioonides.

Robot Framework on platvormist sõltumatu raamistik automaattestimiseks. Raamistikus kasutatakse märksõnadel põhinevat metoodikat (ingl k *keyword-driven testing methodology*), mis tagab selle, et testid on arusaadavad ka tehnilise taustata inimestele ning korrektse seadistuse korral ei vaja testide loomine erilisi programmeerimisoskusi. Samas on raamistik oma olemuselt paindlik ja tänu erinevatele raamistiku lisadele saab seda kohandada väga erisuguste projektide automaattestimiseks. (Robot Framework Foundation, 2017)

Eesti keeles ei leidu kahjuks väga palju materjali, mis annaks ülevaadet automaattestimisest ja selle vahenditest. Robot Framework raamistikku on varasemalt võrreldud teiste raamistikega kahe bakalaureusetöö raames (Annuste, 2016) (Türk, 2015). Samuti on Knowit Estonia OÜ Eestis korraldanud mõned selle raamistiku kasutamisele pühendatud inglise keelsed koolitused.

Antud töö teema on püstitatud autori isiklikust vajadusest tööalaste projektide teostamisel. Et tagada tarkvara elutsüklis nii uute arenduste kui töötavate lahenduste pidev kvaliteet, on erinevate tugisüsteemide, nagu seda on ka automaattestide olemasolu, lausa hädavajalik. Testitava tootena käsitletakse siinkohal pangandustarkvara, mida kasutavad oma töös mitmed Skandinaavia pangad. Töö autor soovib teada, kuidas kasutada Robot Framework raamistikku

nimetatud toote automaattestimisel ning selgitada välja, kas ja millistel tingimustel võiks tulevikus jätkuda automaattestimine nimetatud tarkvaral kasutades selleks just Robot Framework raamistikku.

Käesoleva bakalaureuse töö eesmärk on uurida automaattestide koostamise võimalusi Robot Framework nimelisel raamistikul ja anda praktiline hinnang raamistiku kasutamisele pangandustarkvara näitel. Antud töö on arendusuuring, kus eesmärgi saavutamiseks töö autor tutvustab üldisi automaattestimise põhimõtteid, annab ülevaate Robot Framework raamistikust ja testide koostamise protsessist pangandustarkvarale ning hindab raamistiku kasutamise võimalusi tulevikus.

Töö koosneb kolmest peatükist. Esimene peatükk keskendub automaattestimise üldiste põhimõtete uurimisele – selle peatüki käigus annab autor ülevaate automaattestimise olemusest, raamistike kategoriseerimisest ja valiku põhimõtetest ning peamistest automaattestimisega seotud muredest. Töö teine peatükk annab ülevaate Robot Framework raamistiku paigaldamisest, selle erinevatest osadest ning seal testide koostamisest ja sooritamisest. Kolmas peatükk käsitleb konkreetse tarkvara jaoks automaattestide loomist kasutades selleks Robot Framework raamistikku ning annab hinnangu raamistiku kasutamisele ja sobivusele edaspidise töö jätkamiseks pangandustarkvaral.

1 Automaattestimine

Käesolev peatükk annab ülevaate automaattestimisest – eesmärkidest, põhimõtetest ja kaasnevatest probleemidest. Lisaks tutvustatakse automaattestimiseks kasutatavaid vahendeid ning selgitatakse, kuidas neid klassifitseerida ning milliseid põhimõtteid tuleks järgida oma tööks sobivate vahendite valikul.

1.1 Automaattestimise mõiste

Automaattestimine on protsess, kus süsteem käivitab testitava objekti vastu eelnevalt (inimese poolt) koostatud testid. Testloo sammud, andmed ja oodatud tulemus defineeritakse kasutaja poolt, testide läbimise ja saadud tulemuste esitamise eest vastutab testimiseks kasutatav süsteem. (Lent, 2013)

Järgnevad sammud võtavad kokku automaattestimise olemuse ning annavad ülevaate, mida hõlmab automaattestimise protsess tervikuna (Basu, 2015):

1) automaattestimise skoobi määramine;
2) automatiseerimiseks kasutatavate töövahendite ja raamistike valimine;
3) detailse plaani loomine – strateegia, ajakava, standardid, saadused jmt;
4) automatiseerimist toetava infrastruktuuri loomine ning vahendite kasutamise väljaõpe;
5) automaattestide kavandamine, arendamine, sooritamine;
6) testimise järgne testimiskeskkonna koristus ehk algse olukorra taastamine.

Kuigi automatiseerida saab väga mitmesugust testimist, siis alati pole otstarbekas ja võimalik tervet tarkvara automaattestida. Automaattestimisest rääkides mõeldakse kõige enam üksuste testimisele (ingl k *unit testing*) ja funktsionaaltestimisele (ingl k *functional testing*). Osade funktsiooniväliste nõuete (ingl k *non-functional requirements*) täitmist, nagu seda tehakse pingutustestimise (ingl k *stress testing*), koormustestimise (ingl k *load testing*) ja jõudluse testimisega (ingl k *performance testing*), on üsna tülikas automaattestideta kontrollida. Ka soovitatakse regressioonitestimise faasi teste automatiseerida, sest neid kasutatakse töötsüklite jooksul korduvalt ning nende automatiseerimine aitab kokku hoida palju väärtuslikku aega. (Basu, 2015)

1.2 Automaattestimise vajalikkus

Tarkvaraprojektide ja -süsteemidega on seotud erinevate rollidega inimesed ning nende huvi automaattestimise vastu sõltub positsiooniga seotud eesmärkidest ja vaatenurkadest.

Ärilise poole huvi automaattestimise vastu on põhjendatud järgnevate eesmärkidega (Toledo, 2016):

- parandada tarkvara kvaliteeti;
- vähendada tootmisega seotud muresid;
- säilitada head kuvandit kliendi ees;
- vältida juriidilisi probleeme;
- hoida vigade parandamise kulud madalad.

Siinkohal on selge, et äri eest vastutavad inimesed otsivad finantsilist kindlust, võimalusi kulude vähendamiseks ning soovivad hoida häid suhteid kliendiga nii praegu kui tulevikus.

Infotehnoloogiline pool keskendub peamiselt toote kvaliteedile, tööprotsessidele ja efektiivsusele. Automaattestimine on nendele inimestele oluline seetõttu, et (Toledo, 2016):

- lihtsustab rutiinseid ülesandeid;
- annab võimaluse testida paralleelselt ja erinevatel platvormidel manuaalse sekkumiseta;
- sama aja- ja rahakuluga saab sooritada rohkem teste ning suurendada testimise skoopi;
- kui tervet süsteemi automaattestitakse järjepidevalt, siis on tõenäoline, et süsteemsed defektid leitakse töö varasemas faasis ning nende parandamine on lihtsam ja odavam;
- parandab üldist toote kvaliteeti.

2012. aastal küsitleti mitmeid ettevõtteid, et saada teada, miks ja millises ulatuses soovitakse kasutada automatiseerimist rakendustarkvara testimisel. Uuring kaasas ligi 700 ettevõtet Euroopast ja Põhja-Ameerikast, kellest enamike aastatulu ulatub üle 500 miljoni USA dollari. Uuringu tulemusel valminud raporti „2013 Trends in Automated Testing“ andmetel usuvad 86% vastanutest, et automaattestimine toob majanduslikku kasu mitmetes valdkondades. (Worksoft Inc., 2013)

Ettevõtted hindasid kõige olulisemaks järgmisi automaattestimisega kaasnevaid väärtusi:

1) Töötajate aja kokkuhoid ja efektiive kasutus (Worksoft Inc., 2013)

Automaattestide sooritamine võtab vähem aega kui manuaalne testimine. See ei nõua pidevat inimesepoolset sekkumist ja järelevalvet, küll aga hilisemat analüüsi saadud tulemustele. Automatiseerimise tulemusel saab töötaja loobuda korduvatest tegevustest (nt kui kasutatakse erinevaid konfiguratsioone ja lokalisatsioone) ning keskenduda töö olulisematele ja sisukamatele aspektidele, kus ei pea sama asja mitu korda järjest tegema.

2) Defektid avastatakse senisest varem ning seeläbi väheneb risk ärikasutajale (Worksoft Inc., 2013)

Automaattestimisega saab alustada juba väga varakult ja läbi erinevate arendusfaaside. Seetõttu saavad arendusmeeskonnad informatsiooni tekkinud probleemidest siis, kui nende lahendamine on veel lihtne ja odav. Tänu sellele on risk ärikasutajale väiksem, sest vead ja seosed rakendustarkvara teiste osadega avastatakse tõenäolisemalt juba arenduse käigus ja enne lõpp-kasutajani jõudmist.

3) Äriprotsessides on tagatud pidev kvaliteet (Worksoft Inc., 2013)

Kuna olemasolevate automaattestide käivitamine on kiire, siis ettevõtted saavad mis tahes ajahetkel veenduda, et kriitilised äriprotsessid on endiselt töökorras.

4) Tellitud arendus jõuab lõpp-kasutajani senisest kiiremini (Worksoft Inc., 2013)

Automaattestide kasutamine manuaalsete testide asemel kiirendab erinevate tööfaaside läbimist. Kogu tarkvara hõlmavate automatiseeritud regressioonitestidega saab tehtud töö kvaliteedis kiiresti veenduda ja tellimuse ärikasutajale senisest lühema ooteajaga üle anda.

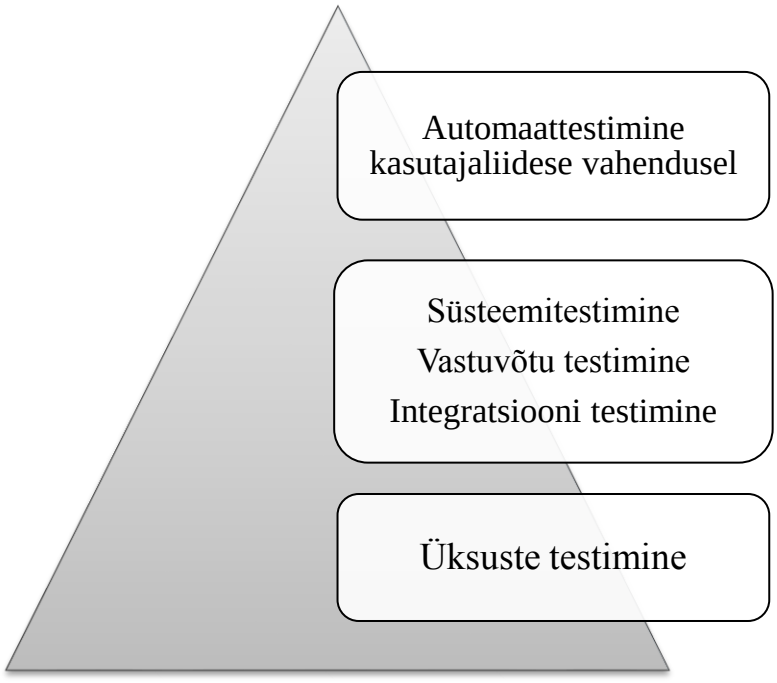
5) Suurem võimekus ja täpsus vigade leidmisel (Worksoft Inc., 2013)

Automaattestimine toetab keeruliste testide sooritamist kitsastes ajaraamides. Automatiseeritud piir-, koormus- ja jõudlustestimine aitab leida tarkvara murdepunkte kiiresti ja efektiivselt.

1.3 Automaattestimise püramiid

Agiilsete arendusmetoodikate levik infotehnoloogia maastikul on pannud aluse „ideaalsele automaattestimise püramiidile“. Antud käsitlus automaattestimisest annab suuniseid, milliseid automaatteste koostada ja milline võiks olla nende osakaal teiste püramiidi tasemetega võrreldes.

Püramiid (joonis 1) koosneb kolmest tasemest – üksuste testimine; süsteemitestimine (ingl k *system testing*), integratsiooni testimine (ingl k *integration testing*) ja vastuvõtutestimine (ingl k *acceptance testing*); testimine graafilise kasutajaliidese vahendusel (ingl k *graphical user interface testing*) (Toledo, 2016). Efektiivne automaattestimise strateegia eeldab tegutsemist kõigil kolmel tasemel (Cohn, 2009). Eelnevale lisaks võib strateegiat täiustada erinevate manuaalsete testimisviisidega (Toledo, 2016). Üks töövahend (ingl k *tool*), teek (ingl k *library*) või testimisraamistik võib samaaegselt katta automaattestimise eesmärgid mitmel erineval tasemel.



Joonis 1. Automaattestimise püramiid (Cohn, 2009).

Automaattestimise püramiidi baastasemel on üksuste testid ning neid koostatakse ja käivitatakse paralleelselt iga arendusversiooniga. Selliste testide koostamine on lihtne, odav ja

kiire. (Toledo, 2016) Samuti annavad need arendajale selgelt märku, millisest koodivahemikust tasuks vigu otsida (Cohn, 2009).

Python keeles kasutatakse üksuste testimiseks näiteks Python'i standardteeke unittest ja doctest (Gheorghiu, 2016). Java's on kasutusel sellised raamistikud nagu TestNG, SureAssert, JUnit ja paljud teised (Wikipedia, n.d.).

Teine tase keskendub erinevate süsteemi komponentide ja nende integratsiooni testimisele kaasamata protsessi kasutajaliidest (Cohn, 2009). Siin testitakse märkimisväärset osa tööloogikast ja äriprotsessist. Testid on küll aeglasemad ja keerukamad kui esimesel tasemel, ent siiski vähema hoolduskuluga kui kolmanda taseme testid ning annavad märku tekkinud vigadest piisavalt varajases arendusfaasis. (Toledo, 2016)

Java's kasutatakse integratsiooni- ja vastuvõtu testimiseks näiteks Arquillian testimisplatvormi ning samu eesmärke täidab ka juba eelnevalt mainitud TestNG (Marshall, 2015). Python'is on vastuvõtu testimiseks kasutusel näiteks behave ja TextTest nimelised töövahendid ning Robot Framework raamistik (Gheorghiu, 2016).

Kõige kõrgemal tasemel automaattestitakse läbi kasutajaliidese. Selle taseme testid on teiste tasemetega võrreldes kõige aeglasemad ja enamasti ka suurima hooldusvajadusega. Seetõttu soovitatakse kasutajaliideses automatiseerida vaid kõige olulisemaid funktsionaalsusi ja protsesse teekonna algusest lõpuni. (Toledo, 2016)

Python'is kasutatakse selle taseme automatiseerimiseks näiteks Automa tarkvara, PyUseCase ja ldtp raamistikke (Gheorghiu, 2016). Java's on kasutusel selleks näiteks Selenide, stevia, darcy raamistikud ja SikuliX teek (GitHub, Inc., 2017).

1.4 Automaattestimise raamistikud ja nende jagunemine metoodika alusel

Testimisraamistiku eesmärk on korraldada ja suunata erinevate komponentide (testid, andmed, teegid, draiverid jne) omavahelist koostööd, aga ka lihtsustada automaattestide kirjutamist ja nende haldamist tulevikus (Software Testing Help, 2016).

Testimisraamistik määrab testide formaadi, tagab struktuurse lähenemise ning ühenduse testide ja automatiseeritud testkoodi vahel, sooritab teste ning koostab saadud tulemuste põhjal raporti.

Lisaks eelnevale on raamistiku üheks oluliseks osaks draiver, mis tegeleb otseselt testitava komponendi liidese manipulatsiooniga. (Hendrickson, 2011)

Siinkohal käsitletakse testimisraamistike jagunemist neid iseloomustavate metoodiliste põhimõtete (nagu struktuurilised ja arhitektuurilised omadused) alusel. Sellise liigituse kohaselt eristatakse kuute tüüpi testimisraamistikke.

Moodulitel põhinev testimisraamistik (ingl k *module based testing framework*) lähtub objektorienteeritud programmeerimise põhimõtetest. Testitav objekt on jaotatud loogilisteks ja iseseisvateks osadeks ning testide planeerimisel käsitletakse iga moodulit eraldiseisva struktuuriüksusena. Mooduleid saab omavahel kokku kombineerida nii, et tekivad pikemad testimisskriptid ja nii saab korraga testida erinevaid funktsionaalsusi. Kuna moodulid ei sõltu üksteisest, siis muudatused ühes moodulis ei avalda mõju teistele moodulitele või testimisskriptidele. Kuigi raamistik on üsna lihtsasti hooldatav ja laiendatav, siis miinuseks võib pidada asjaolu, et testandmed on testide sisse kirjutatud ja erinevate andmetega testimiseks peab testimisskripte olulisel määral muutma. (Software Testing Help, 2016)

Teekidel põhineva arhitektuuriga testimisraamistik (ingl k *library architecture testing framework*) on väga sarnane eelnevale. Erinevus seisneb selles, et loogilisteks mooduliteks jagamise asemel otsitakse funktsionaalsuse ühisosa ning need eraldatakse testimisskriptist välja. Testimisskript saab aga teegi poole pöörduda, millal iganes vaja. Ehkki loodud testidel on kõrge taaskasutuse määr, on siingi testandmed kirjutatud testimisskriptide sisse. (Software Testing Help, 2016)

Andmekeskne testimisraamistik (ingl k *data driven testing framework*) tugineb testimisskriptide ja testandmete lahus hoidmisele. Tihti piisab sama testi sooritamisest erinevate andmetega. Testandmed on eraldiseisvas hoidlas ning informatsiooni salvestatakse võti ja väärtus paaridena. Võti märgitakse ära testimisskriptis ja testi täitmisel otsitakse sellele vastavaid väärtusi hoidlast. Selline raamistik vähendab loodavate testskriptide arvu, sest sama skripti saab kasutada korduvalt muutes ainult andmeid. Samas aga nõuab see paremat programmeerimisoskust ning tööd testandmete allikate ja nende töötlemise mehhanismidega. (Software Testing Help, 2016)

Märksõnakeskne testimisraamistik (ingl k *keyword-driven testing framework*) on eelneva raamistiku edasiarendus. Lisaks sellele, et testimisskriptid ja andmed on eraldi, siis siinkohal

lahutatakse ka koodikogumeid, mida testide sooritamisel kutsutakse välja märksõnade (ingl k *keywords*) kaudu. Raamistiku kasutamine ei nõua eelnevalt loetletud raamistikega võrreldes nii suurt programmeerimise kogemust ning keerukus tõuseb kasutamisel järk-järgult. (Software Testing Help, 2016)

Hübriid testimisraamistik (ingl k *hybrid testing framework*) ühendab eelnevate raamistike omadused. Selline häälestus ühendab erinevate raamistike parimad praktikad. (Software Testing Help, 2016)

Käitumiskeskse testimisraamistiku (ingl k *behaviour driven development framework*) kontseptsioon seisneb käitumispõhise arendusmetoodika ja vastuvõtutestimise ühendamises. Idee seisneb selles, et tänu raamistiku toele saab automaatteste kirjutada sellises vormingus, et nende kirjutamisega saavad hakkama ka tehnilise taustata inimesed. Käitumiskesksed testid kirjeldavad oodatud funktsionaalsust järgnevas vormis: Võttes arvesse et hetkel olukord on selline (näiteks „võttes arvesse, et pangakontol on 100 eurot“), kui sooritatakse mingit tegevust (näiteks „kui inimene maksab poes 60 euro eest“), siis tulemus on selline (näiteks „siis pangakontole jääb alles 40 eurot“). Testide fookus on rakenduse oodatud käitumisel, mitte aga teostatud tehnilisel lahendusel ning seetõttu aitavad sellised testid paremini selgitada tegelikke ootusi ja täpsustada nõudeid rakendusele. (Madaan, 2015)

Turult leiab ohtralt soovitusi ja pakkumisi erinevatele automaattestimise raamistikele. Valmistoodete kõrval on võimalik ehitada ise kohandatud lahendus. Alati ei pea raamistik ise teadma, kuidas testitava süsteemiga suhelda ning seetõttu võib raamistiku ehitada mõne olemasoleva töövahendi ümber.

Üks selline näide on Selenium WebDriver, mis korraldab suhtlust erinevate veebibrauserite ja raamistike vahel. Iga veebibrauseriga suhtlemine võib toimuda erinevalt, ent Selenium WebDriver’i abiga ei pea testija selleks lisapingutusi tegema. (Selenium Project, 2017) Veebirakenduste automaattestimisel kasutavad Selenium WebDriver’it muuhulgas näiteks sellised raamistikud nagu TestNg, Serenity ja Robot Framework. Samuti saab ise ehitada kohandatud ja enda vajadustele vastava raamistiku selle ümber. Selliseid koostöö näited on mitmeid ja mitte ainult veebirakenduste automaattestimiseks.

Testimisraamistike kõrvalt leiab veel mitmeid muid töövahendeid, mis otseselt või kaudselt on seotud automaattestimisega: tarkvarad, pistikprogrammid (ingl k *plug-ins*) veebibrauseritele ja

teistele programmidele, laiendused (ingl k *extensions*), teegid ja muu taoline. Töö autori hinnangul puudub üheselt mõistetav kriteeriumite kogu, et nimetatud töövahendeid täpsemalt rühmitada.

1.5 Valikukriteeriumid automaattestimise töövahendi valimisel

Internetis võib leida väga erinevaid soovitusi, millist automaattestimise töövahendit valida. Pakutud variandid ei ole enamasti universaalsed ning seetõttu tasuks enne lõpliku otsuse tegemist hoolikalt süveneda võimalustesse ja vajadustesse.

Õigete valikute tegemist mitmete testimisraamistike ja -vahendite vahel peetakse üheks automaattestimise suurimaks väljakutseks. Seetõttu tuleb selgelt määratleda eesmärke, uurida erinevaid vahendeid ja nende võimekust ning selgitada, kas vaadeldud vahendid vastavad ootustele. Et lihtsustada otsuse tegemist ja vältida ebavajalikke investeeringuid, soovitatakse läbi viia kontseptsiooni tõestus (ingl k *proof of concept*). Selle käigus tuleks seada ootused automaattestimise töövahendile, panna see proovile reaalses situatsioonis toote automaattestimisel, ja alles selle kogemuse naljalt teha lõplikke valikuid. Teoreetilisel tasandil tehtud otsused ei pruugi praktikas kõige paremateks osutuda ning seetõttu tuleks valiku tegemisel arvestada paljude erinevate aspektidega. (Basu, 2015)

Esimese asjana tuleks selgitada, mida on tarvis automaattestida ja millises keskkonnas see toimub (Basu, 2015). Testida võib ainult ühes või paralleelselt mitmes erinevas keskkonnas (nt Windows, Linux, iOs, Android rakendus, veebirakendus jne). Erinevate püramiidi tasemete automatiseerimiseks võib kasutada samu või erinevaid vahendeid. Lisaks funktsionaaltestimisele võib olla vajadus automatiseerida näiteks funktsioonivälised teste või tegeleda jõudluse mõõtmisega.

Samuti peaks kohe otsustama, milliseid kulutusi ollakse valmis tegema (Basu, 2015). Olenemata sellest, et kommertslahenduste litsentsid on enamasti tasulised ja vabavaralised mitte, on hinna kõrval veel muid nüansse, millele tasuks tähelepanu pöörata. Kommertslahenduste müüjad on tihti valmis pakkuma ka personaalset tuge lahenduse kasutajale, samal ajal kui vabavaralised lahendused võivad sõltuda vabatahtlikest ja huvigrupi sponsorlusest.

Töövahendite litsentside kõrval on teiseks oluliseks kuluallikaks personali koolitamine (Basu, 2015). Tasub uurida, milliste vahenditega on ettevõtte töötajad juba varem kokku puutunud. Täiesti uue asja õppimine, millega personalil varasemad kogemused puuduvad, võib osutuda pikaks ja kulukaks protsessiks.

Järgmisena saab juba seada täpsemaid nõudeid töövahenditele ja raamistikele. Esimesena on vaja määrata skriptimiskeel, mille valik võiks olla kooskõlas ettevõttes leiduva kompetentsi ja teostatava projekti arenduskeelega. (Basu, 2015). Seejärel tasuks tähelepanu pöörata sellistele omadustele nagu mitme töövahendi tugi ja võimekus tunda ära elemente erinevates keskkondades (Basu, 2015). Kui testimisobjekt toimib üle erinevate keskkondade, siis samaks asjaks peaksid võimelised olema ka valitud testimisvahendid. On mugav, kui sama testimisskripti saab kasutada kõikides keskkondades ning testimisvahend ei sea selles osas takistusi. Ka ei piisa mõnikord testimiseks ainult ühest spetsiifilisest töövahendist, vaid on vaja omavahel integreerida mitu erinevat vahendit (Basu, 2015).

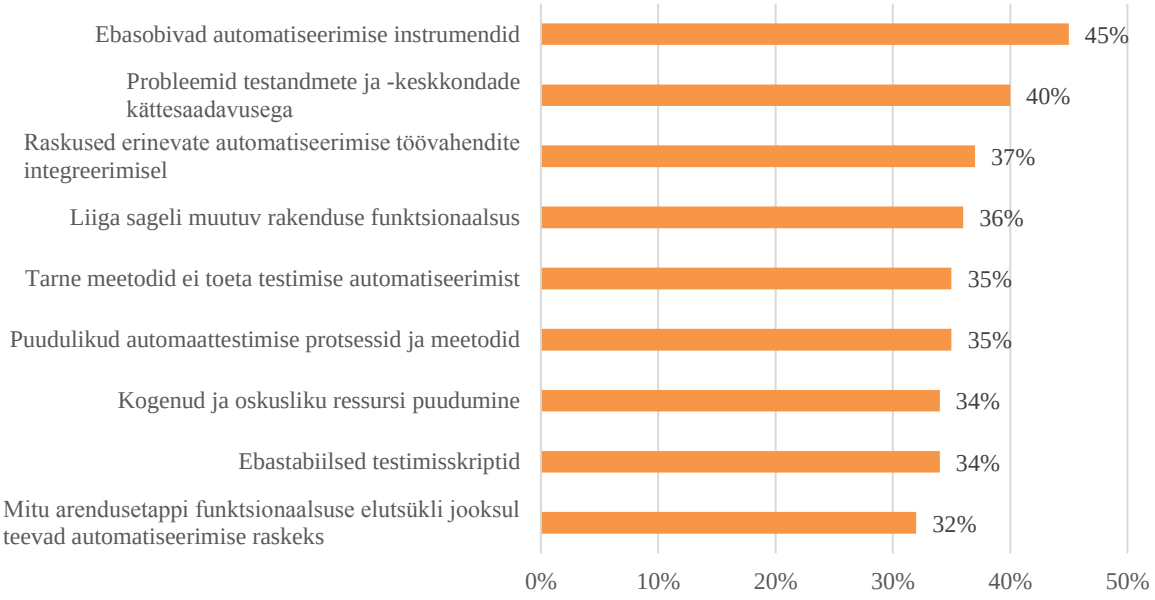
Eelnevale lisaks tasuks süveneda sellesse, kuidas toimub töövahendis testimisskriptide tõrgete analüüs ja millised on testimise tulemuste aruanded ning kas see toimub ootustele vastaval tasemel (Basu, 2015). Mida täpsemini annab töövahend informatsiooni vigase sisu osas, seda lihtsam on lahendada tekkinud tõrkeid. Aruanded peaksid olema kergesti loetavas vormis ja piisavalt sisukad.

1.6 Automaattestimise probleemid

World Quality Report 2016-17 keskendub testimise ja kvaliteedi tagamise (ingl k *quality assurance*) trendide uurimisele. 2016. aastal läbi viidud uuringus osales 1600 juhtivtöötajat 32 erineva riigi era- ja avaliku sektori ettevõtetest ja organisatsioonidest. (Camgemini, Sogeti, Hewlett Pacard Enterprise, 2016)

Kuigi testimise automatiseerimist peetakse oluliseks faktoriks testimise tõhususe hindamisel, siis uuringule tuginedes automatiseeritakse ainult 29% kogu testimisest, mis on tugev langus võrreldes eelmiste aastate tõusutrendidega. Muuhulgas mainitakse ka probleeme automatiseerimisprojektide edukuse ja investeeringute tasuvusega. (Camgemini, Sogeti, Hewlett Pacard Enterprise, 2016)

Joonis 2 annab täpsema ülevaate, mis on uuringus osalejate arvates peamised automaattestimise ebaedu põhjused. Mitmed tegurid on seotud eelkõige töövahenditest tulenevate probleemidega. Pea pooled (45%) vastanutest nõustusid, et kõige rohkem raskusi põhjustavad ebasobivad töövahendid ja sellele lisaks kogesid paljud (37%) probleeme erinevate töövahendite integreerimisel. Ka oldi hädas näiteks testandmete ja -keskkondade kättesaadavusega ja liiga tihti muutuva rakendusega. Üle kolmandiku vastanutest leidsid, et neil puuduvad automatiseerimist toetavad protsessid ja meetodid, aga ka pädevad inimesed nende rakendamiseks. (Camgemini, Sogeti, Hewlett Pacard Enterprise, 2016)



Joonis 2. Automaattestimise ebaedu põhjused 2016. aastal (Camgemini, Sogeti, Hewlett Pacard Enterprise, 2016).

Töö autori arvamusel võib antud uuringust teha mõningaid järeldusi, kuidas olla automaattestimises edukas, ning milliseid töökorralduslike samme selleks teha. Esiteks mängivad töövahendid väga tähtsat rolli kogu ettevõtmise õnnestumises ning nende valik peaks olema kaalutletud ning arvestama tegelike vajadustega. Samuti on oluline selgitada, mida ja kuidas soovitakse saavutada, ning vastavalt sellele kokku leppida automaattestimise meetodites ja protsessides. Alustamisel tuleks üle vaadata ka olemasolev töökorraldus ja veenduda, et see toetab automaattestimise lisamist tööprotsessi ning vajadusel viia sisse korrektuurid. Lisaks võiks panustada senisest rohkem inimeste koolitamisse ning eesmärki toetavate materjalide loomisesse.

2 Robot Framework raamistik

Järgneva peatüki eesmärk on tutvustada Robot Framework raamistiku kasutamist. Autor teeb ülevaate raamistiku arhitektuurist, paigaldamisest ja erinevatest komponentidest nagu teegid ja muud töövahendid ning nende koostööst raamistikuga. Samuti selgitatakse selles peatükis, kuidas antud raamistikul automaatteste kirjutada ja hiljem käivitada.

2.1 Ülevaade Robot Framework raamistikust

Robot Framework on märksõnakeskne testimisraamistik vastuvõtutestimiseks. Raamistik on avatud lähtekoodiga (Apache License 2.0 litsentsi alusel) ning põhineb Python arenduskeelel (Robot Framework Foundation, 2017).

Raamistiku looja on Pekka Klärck. Esialgne versioon sai alguse 2005. aastal tema magistritöö raames koostöös Nokia Siemens Networks ettevõttega. Hilisemad versioonid ei ole enam olnud seotud konkreetse firmaga ning on seetõttu kättesaadavad kõigile soovijatele. (Bisht, 2013)

Robot Framework raamistikul on mitmeid eripärasid, mistõttu on see sobilik töövahend väga erinevate projektide automaattestimiseks. Esiteks on raamistik platvormist sõltumatu. Osaliselt on see saavutatud tänu mitmetele teekidele, mida on võimalik raamistikule lisaks paigaldada. Seetõttu saab selle raamistikuga automaattestida nii erinevate operatsioonisüsteemide töölaua rakendusi kui ka veebi- ja mobiilirakendusi. Lisaks raamistikku sisse ehitatud funktsionaalsusele on esindatud mitmed teegid ja tugi suhtlemaks väliste tootjate komponentidega. Olemasolev API toetab kohandatud teekide loomist Python'i või Java baasil. (Robot Framework Foundation, 2017)

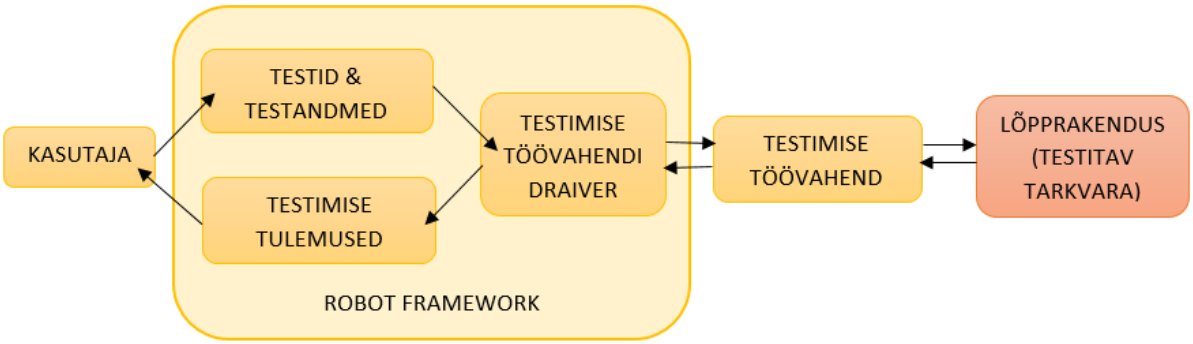
Teste kirjutatakse märksõnade abil, mis viitavad erinevatele teekides kirjeldatud meetoditele. Seetõttu näevad testid välja nagu oleks need kirjutatud lihtsas inglise keeles. Tänu sellele on testid loetavad ja kirjutatavad ka mitte-tehnilisele töötajaskonnale. (Mishra, 2014)

Testide formaadina kasutatakse tabeli kuju, mis aitab hoida testidel ühetaolist joont. Erinevaid märksõnu saab kokku kombineerida kõrgema taseme märksõnadeks. Ka on võimalik koostada andme- ja käitumiskeskseid teste ning kasutada keskkonnast sõltuvaid muutujaid. Testimisele eelnevat seadistamist (ingl k *test set up*) ja testide sooritamisele järgnevat testimiskeskkonna korrastamist (ingl k *test tear down*) saab üles seada nii üksiku testloo (ingl k *test case*) kui ka

terve testide komplekti (ingl k *test suite*) tasandil. Lisaks sellele saab testidele lisada silte (ingl k *tags*), et vajadusel töötada vaid nende testidega, mis on konkreetse märgistusega. Et kõiki andmeid hoitakse failide kujul kataloogides, siis sarnaselt tootekoodile saab teste lihtsasti versiooneerida ja muudatusi hallata paralleelselt testitavas rakenduses toimuvaga. (Robot Framework Foundation, 2017)

2.1.1 Raamistiku arhitektuur

Järgnev joonis (joonis 3) annab üldistatud ülevaate Robot Framework raamistikus seotud komponentidest ning info liiklusest nende vahel. Arhitektuuriliselt on tegemist moodulitel põhineva lahendusega, mistõttu on lihtne raamistikku kohandada ja edasi arendada vastavalt testitava objekti vajadustele (Bisht, 2013).



Joonis 3. Robot Framework raamistiku arhitektuuri ülevaade (Bisht, 2013).

Eelnevast joonisest (joonis 3) selgub, et kasutaja annab raamistikule tööks vajaliku sisendi testide ja testandmetega ning sellest sõltub otseselt testide täitmine (Bisht, 2013). Robot Framework raamistik on tuum, mis kogu protsessi haldab ja koos hoiab (Bisht, 2013), kusjuures raamistikul endal puudub otsene kontakt testitava tarkvaraga – suhtlust lõpprakendusega korraldavad erinevad teegid ja nende draiverid (Robot Framework Foundation, 2017).

Testide käivitamisel tagab draiver suhtluse raamistiku ja testimise töövahendi vahel. Läbi draiveri liigub info töövahendini, mis seejärel kasutab saadud teavet lõpprakenduse testimiseks vastavalt kasutaja poolt määratud testimisstsenaariumitele ja andmetele. Töövahendiks võib olla mingi muu teek, liides, pistikprogramm, tarkvara või muu säärane, mis ei ole raamistiku standardne osa, aga mille funktsionaalsust kasutatakse vajaduse tekkimisel läbi vastavate draiverite. Tagasiside sooritatud testidest ja tulemustest suunatakse töövahendi ja selle draiveri

vahendusel tagasi raamistikuni. Seal toimub edasine teabe töötlemine, mille tulemusel saab kasutaja testimise raporti toimunu kohta. (Bisht, 2013) Mõningatel juhtudel saab vahelt ära jätta eraldiseisva töövahendi, sest draiver oskab juba ise suhelda rakenduse kasutajaliidesega (Robot Framework Foundation, 2017).

2.1.2 Raamistiku teegid

Erinevad teegid annavad Robot Framework raamistikule tegeliku võimekuse testimiseks. Teegid sisaldavad kõige madalama taseme märksõnu, mille abil suhelda testitava süsteemiga. Iga testlugu kasutab alati mingisugust märksõna mõnest teegist, sh ka nendest, mis sisaldavad kasutaja poolt loodud kõrgema taseme märksõnu. (Robot Framework Foundation, n.d.)

Teegid jagatakse kolme gruppi – standardsed, välised ja kasutaja poolt loodud teegid. Standardsed teegid on juba raamistikuga kokku komplekteeritud ja neid ei pea eraldi installeerima. Järgnevas tabelis (tabel 1) on välja toodud kõik standardsed teegid ning nende peamised ülesanded (Robot Framework Foundation, n.d.).

Tabel 1. Standardsed teegid ja nende ülesanded.

Teek	**Ülesanne**
BuiltIn	Tihedalt kasutust leidvad märksõnad
Collections	Python'i loendite ja sõnastike haldamine
DateTime	Kuupäevade ja aja haldamine
Dialogs	Seisakute tegemine testide sooritamisel ning kasutajalt sisendite küsimine
OperatingSystem	Erinevate operatsioonisüsteemiga seotud ülesannete täitmine (nt failidega seotud operatsioonid)
Process	Erinevate protsesside algatamine, kulgemise jälgimine ja lõpetamine
Screenshot	Ekraanitõmmiste tegemine masinas, kus teste sooritatakse
String	Stringide verifitseerimine ja muutmine
Telnet	Suhtlus üle Telneti ühenduse
XML	XML dokumentide verifitseerimine ja muutmine

BuiltIn teek võetakse kasutusse automaatselt ja seetõttu on sealsed märksõnad alati saadaval. Ülejäänud teekide (sh standardsed, välised ja kasutaja poolt loodud) märksõnade kasutamiseks peab vastavad teegid testi importima. (Robot Framework Foundation, 2017)

Kõik, mis ei kuulu standardsete teekide nimekirja, loetakse välisteks. Nende paigaldamine ja kasutusloogika võib olla täiesti erinev standardsetest teekidest ja muuhulgas ka eeldada mingite eraldiseisvate kolmandate komponentide kasutamist. (Robot Framework Foundation, 2017)

Järgnevalt on välja toodud mõned näited välistest teekidest (Robot Framework Foundation, n.d.):

- Selenium2Library – kasutatakse veebirakenduste testimiseks;
- SwingLibrary – Java rakenduste testimiseks, mis kasutavad Swing graafilist kasutajaliidest;
- iOS library – kasutatakse iOs operatsioonisüsteemil rakenduste testimiseks;
- Database Library (Python) – Python'il põhinev teek andmebaaside testimiseks;
- Django Library – Python'i veebiraamistiku Django testimiseks;
- robotframework-faker – võltside testandmete genereerimine.

Eelnevas loetelus on kirjas vaid mõned välised teegid. Hetkel on saadaval umbes 30 teeki mitmetele platvormidele ja erinevate ülesannete täitmiseks.

Kui olemasolevad teegid ei täida testimisvajadusi, siis on võimalik kohandatud teeke ise juurde luua. Robot Framework raamistikul on selle tarbeks kolm erinevat API't – Static API, Dynamic API ja Hybrid API – ning nende kasutamiseks on tarvis elementaarseid teadmisi kas Python või Java keelest (Robot Framework Foundation, 2017).

2.1.3 Raamistiku töövahendid

Robot Framework raamistikul on peale tuumikmehhanismi, mis käivitab teste, ka mitmeid muid abivahendeid, et toetada erinevaid testimisprotsessi osasid. Valdavat enamust arendatakse eraldiseisvalt konkreetsest raamistikust, ent mõned neist käivad juba ka raamistikuga koos. (Robot Framework Foundation, n.d.)

Raamistiku paigaldamisel tuleb kohe kaasa 4 töövahendit – Rebot, Libdoc, Testdoc ja Tidy. Rebot genereerib logisid ja raporteid XML väljundi põhjal. Libdoc ja Testdoc aitavad dokumentatsiooni loomisel – esimene neist on testimisteekidele ja ressursifailidele, teine aga testlugudele. Tidy oskab muuta ja korrastada testandmete faile. (Robot Framework Foundation, n.d.)

Robot Framework raamistiku enda redaktor kannab nime RIDE (vt Lisa 1). Sellele lisaks eksisteerib mitmeid pistikmooduleid (sh ka süntaksi esiletõstmiseks) teistele redaktoritele nagu näiteks Sublime, Eclipse, Vim, Gedit, Notepad++ ja teised. (Robot Framework Foundation, n.d.)

Jenkins on serverisüsteem keskkondade ja toote versioonide pidevintegratsiooni (ingl k *continuous integration*) haldamiseks ja automatiseerimiseks (Jenkins, n.d.). Robot Framework pistikmoodulit Jenkins'ile kasutatakse selleks, et panna kaks süsteemi omavahel suhtlema ning avaldada testimise tulemused Jenkins keskkonnas. (Robot Framework Foundation, n.d.).

Maven ja Ant on töövahendid tarkvara paigaldusprotsesside automatiseerimiseks (The Apache Software Foundation, n.d.). Robot Framework pistikmooduliga saab neis kasutada raamistiku funktsionaalsust nii, et midagi täiendavat pistikmoodulile lisaks paigaldama ei pea (Robot Framework Foundation, n.d.).

Ülejäänud töövahendid on loodud tugi- ja haldusülesannete lihtsustamiseks. Näiteks Babot'i abil saab käivitada teste paralleelsessioonidena. Fixml parandab katkiseid väljundifaile ja Robot Framework Hub on veebiserver, mis tagab ligipääsu kogu raamistiku funktsionaalsusele veebibrauseri vahendusel. Siinkohal on välja toodud vaid mõned näited ja tugivahendeid erinevate probleemide lahendamiseks on veelgi. (Robot Framework Foundation, n.d.)

2.2 Robot Framework raamistiku ja lisade paigaldamine

Robot Framework raamistik on loodud Python arenduskeelel ja vajab seda ka töötamiseks. Python'ile lisaks (nii versioon 2 kui 3) toetab raamistiku kasutamist ka Jython (Python Java platvormil), IronPython (Python .NET platvormil) ja PyPy (Python'i alternatiivkeel). Raamistiku paigaldamine eeldab, et operatsioonisüsteemis on tugi vähemalt ühele nimetatud programmeerimiskeeltest. (Robot Framework Foundation, 2017)

Interpretaatori valik sõltub soovitud testimisteekidest ja testimiskeskkonnast – kui mõned töövahendid sobivad kõigi keeltega kasutamiseks, siis mõningad teegid on mõeldud kasutamiseks ainult Python'iga ja teised jälle ainult mõne muu arenduskeelega. Eriliste nõuete puudumisel soovitatakse kasutada eelkõige Python'it, sest see on kõige kaugemale arenenud teostusega ning teiste raamistikus toetatud keeltega võrreldes märkimisväärselt kiirem. Samuti on Python juba installeeritud mitmetes UNIX-il põhinevates operatsioonisüsteemides (nt OS

X, Linux). Windows operatsioonisüsteemil raamistiku kasutamiseks tuleb alustada Python'i paigaldamisega. (Robot Framework Foundation, 2017)

Paigaldamise saab kokku võtta järgmiste etappidega (Robot Framework Foundation, 2017):

0) Python'i (või muu eelnevalt nimetatud arenduskeele) paigaldamine ja seadistamine;
1) Robot Framework raamistiku paigaldamine;
2) Robot Framework raamistiku väliste teekide paigaldamine;
3) raamistiku kasutamiseks vajalike töövahendite paigaldamine (RIDE redaktor või pisikprogrammid teistele redaktoritele).

Python'i paigalduspaki ja -juhendi Windows operatsioonisüsteemile leiab aadressilt https://www.python.org/downloads/. UNIX operatsioonisüsteemidel on Python juba eelnevalt installeeritud ning seda sammu eraldi läbima ei pea.

Kui Python on paigaldatud ja töövalmis, saab jätkata Robot Framework raamistiku paigaldamisega. Kõige lihtsamini saab seda teha paketihalduriga pip. See on saadaval nii Python'is kui ka Jython, IronPython ja PyPy arenduskeeltes ning töötab kõikidel operatsioonisüsteemidel. Windows operatsioonisüsteemis on pip standardne Python'i osa (Python 2-s alates versioonist 2.7.9 ja Python 3-s alates versioonist 3.4). Teistel operatsioonisüsteemidel töötades tuleb pip manuaalselt paigaldada. Koodinäide 1 on kirjas, kuidas Python'iga installeerida pip paketihaldurit ja mõned variandid, kuidas käsurealt alustada raamistiku paigaldamist. Java tarbeks on loodud ka eraldiseisev jar-arhiiv, mis sisaldab Jython'it ja Robot Framework raamistikku ning ei vaja töötamiseks midagi rohkemat kui Java enda olemasolu. (Robot Framework Foundation, 2017)

```
#pip paketihalduri paigaldamine
python get-pip.py

#raamistiku viimase versiooni paigaldamine
pip install robotframework

#raamistiku valitud versiooni paigaldamine
pip install robotframework==2.9.2

#raamistiku paigaldamine kui operatsioonisüsteemis on mitu Python'i
versiooni
python -m pip install robotframework
python3 -m pip install robotframework
```

Koodinäide 1. Paketihalduri pip ja Robot Framework raamistiku paigaldamine Python'is.

Raamistikule teekide paigaldamine sõltub sellest, mida ja kuidas soovitakse testida. Kuna tegemist on väliste tootjate komponentidega, siis nende paigaldamine võib üksteisest erineda. Näiteks Selenium2Library teegi paigaldamine on väga lihtne kui kasutada selleks eelmistele näidetele sarnaselt pip paketihaldurit (koodinäide 2). (Robot Framework Foundation, 2017)

Samas vajab see teek töötamiseks ka vastava draiveri paigaldamist. Veebibrauseri draivereid Python'ile saab alla laadida järgnevalt lingilt: https://pypi.python.org/pypi/selenium.

```
# Selenium2Library paigaldamine
pip install robotframework-selenium2library
```

Koodinäide 2. Selenium2Library paigaldamine pip paketihalduriga.

Töövahendite või nende pisikmoodulite paigaldamine sõltub kasutaja harjumustest ja eelistustest. Raamistiku kasutamiseks ei pea ilmtingimata nt RIDE või teiste redaktorite pistikmooduleid kasutama, kuigi omajagu mugavust lisavad need küll. Vajadusel saab töö tehtud ka operatsioonisüsteemide standardsete tekstiredaktoritega.

RIDE redaktori kasutamiseks on vaja kõigepealt paigaldada wxPython ja seejärel RIDE rakendus. RIDE graafiline kasutajaliides on arendatud wxPython'i abil ning rakendus vajab seetõttu seda ka töötamiseks. Kuigi raamistik ise alustas hiljaaegu Python 3 toega, siis mitmed teised teegid ja töövahendid ei ole veel nii kaugele jõudnud. Nii toetab hetkel ka RIDE ainult Python 2-te alates versioonist 2.6. Lisaks sellele toetab see ainult standardset Python'it, mitte aga Jython, IronPython või PyPy programmeerimiskeeli. (GitHub, Inc., 2017)

Sobiva wxPython'i versooni paigalduspaketi saab alla laadida järgnevalt veebiaadressilt: https://sourceforge.net/projects/wxpython/files/wxPython/. RIDE arendajad soovitavad kasutada wxPython'i 2.8.12.1 versiooni, sest selles on Unicode teksti kodeering ametlikult toetatud. RIDE paigaldamiseks Windows operatsioonisüsteemil on olemas graafiline paigaldusprogramm. Kuna pip paketihaldur on juba eelnevalt paigaldatud, siis piisab ka sellest, kui käsurealt anda sisse vastav korraldus (koodinäide 3). Selle käsuga saab RIDE paigaldada kõikides operatsioonisüsteemides eeldusel, et pip paketihaldur on juba paigaldatud. (GitHub, Inc., 2017)

```
# RIDE paigaldamine
pip install robotframework-ride
```

Koodinäide 3. RIDE redaktori paigaldamine käsurealt.

Edaspidi saab RIDE redaktorit käivitada käsurealt `ride.py` korraldusega. Windows operatsioonisüsteemi alt töötades on võimalik tekitada ka vastav otsetee operatsioonisüsteemi töölauale. (GitHub, Inc., 2017)

2.3 Testide koostamine Robot Framework raamistikus

Järgnevalt selgitatakse, kuidas hoitakse raamistikus testidega seotud informatsiooni, millistest komponentidest testid koosnevad ja kuidas neid struktureerida. Lisaks tutvustatakse erinevatest meetoditest lähtuvaid testlugude kirjutamise stiile ning testimismallide kasutamist.

2.3.1 Testide formaat ja komponendid raamistikus

Robot Framework raamistikus hoitakse teste tabelikujulises vormingus, kasutades selleks HTML, TSV *(tab-separated values)*, reST *(reStructuredText)* või lihtteksti formaati. Formaadi valik sõltub kontekstist, ent erinõuete puudumisel soovitatakse kasutada just lihtteksti formaati. Raamistik valib õige süntaksianalüsaatori vastavalt faililaiendile. Lihtteksti puhul on selleks kas .txt või .robot. (Robot Framework Foundation, 2017)

Lihtteksti formaadis teste on teise formaatidega võrreldes kõige lihtsam redigeerida ning samuti on see sobilik versioonihalduses kasutamiseks. Üks peamine eelis teiste formaatidega võrreldes on see, et mitmetel redaktoritel on sellele formaadile Robot Framework pistikmoodul, mis tõstab esile süntaksi ning võimaldab märksõnade automaatset jätkamist. Samuti on just see formaat RIDE redaktoris kasutusel. Faililaiendi .robot kasutamine lihtsustab testimisfailide eristamist muudest süsteemis olevatest failidest. (Robot Framework Foundation, 2017)

Tabelikujulise vormingu hoidmiseks lihtteksti failiformaadis on kaks võimalust – tabeliveergude eraldamine toimub kas tühikute või tühiku ja püstkriipsu abil. Tühiku ja püstkriipsu puhul on visuaalselt lihtsam testi vaadata, eriti kui mõne lahtri sisu pikaks venib. Tühikutega eraldamisel peab kahe tabeli lahtri vahel olema minimaalselt kaks tühikut. Töötlemise mõttes ei ole vahet kumba varianti kasutada, sest raamistik oskab mõlemat samal tasemel käsitleda. (Robot Framework Foundation, 2017)

Kasutades näiteks RIDE redaktorit (vt Lisa 1), ei pea vormingule ja võimalikele vormistusvigadele mõtlema. Selle töö teeb juba redaktor ise ära pakkudes selleks sobivat vormingut ja kontrollides sisestatu õigsust.

Kood on tõstutundetu ja ladina tähestikus. Testi erinevaid osasid identifitseeritakse tabeli nimetuse alusel. Igas failis on üks kuni mitu tabelit. Nimetus märgitakse ära tabeli esimese rea esimeses lahtris ning tabelit hakatakse nimetusega kooskõlas olevate andmetega täitma alates teisest reast. Parsimisel ignoreeritakse tühje ridu; koodi osasid, mis on enne esimese tabeli nimetust ja tabeli nimetuse järel tabeli esimeses reas; ning tabelit ja selle sisu, mille nimetus ei ole kooskõlas struktuuri reeglitega. Järgnevalt (tabel 2) on välja toodud tabeli nimetused, mille alusel raamistik erinevaid testi osasid ära tunneb. (Robot Framework Foundation, 2017)

Tabel 2. Tabeli nimetused Robot Framework raamistikus testi erinevate osade identifitseerimiseks (Robot Framework Foundation, 2017)

Kood	Tähendus	Kasutusala
`***Settings***`	Seaded	1) Metaandmete defineerimine testlugude ja testide komplektide jaoks 2) Teekide, ressursifailide ja muutujate failide importimine
`***Variables***`	Muutujad	Muutujate defineerimine
`***Test Cases***`	Testlood	Testlugude koostamine saadaval olevatest märksõnadest
`***Keywords***`	Märksõnad	Kõrgema taseme (kasutaja poolt loodud) märksõnade loomine

2.3.2 Testlugude struktuur ja erinevad kirjutamise stiilid

Robot Framework raamistikus kasutatakse testide kirjutamiseks erinevaid märksõnu. Oma ülesehituselt sarnaneb see lihtsale inglise keelele. Seetõttu on need testid arusaadavad ka inimesele, kellel puudub tehniline taust. (Robot Framework Quick Start Guide, 2017)

Robot Framework raamistikus eristatakse kasutatava metoodika alusel kolme liiki teste – testid on kas märksõnakesksed, käitumiskesksed või andmekesksed. Esimest kahte nimetatud meetodit kasutatakse töövoogude testimiseks. Andmekeskseid teste saab kasutada aga töövoogude testimiseks muutuvate andmetega. (Robot Framework Foundation, 2017)

Märksõnakeskses testloos kutsutakse välja ja läbitakse märksõnu jadamisi ning testi lõpus kontrollitakse, kas süsteemis on oodatud tulemus (Robot Framework Foundation, 2017).

Testlood pannakse kokku erinevatest märksõnadest ja nende argumentidest. Testides kasutatavaid märksõnu saab jagada kahte gruppi – need, mis pärinevad erinevatest teekidest (sisesed, välised ja kasutaja poolt loodud teegid) ehk madala taseme märksõnad ning kasutaja poolt loodud kõrgema taseme märksõnad. (Robot Framework Quick Start Guide, 2017)

Kõik madalama taseme märksõnad on defineeritud teekides kasutades selleks valitud programmeerimiskeelt. Et testloos mõnda neist teekidest kasutada, tuleks need eelnevalt importida. Ainuke erand on BuiltIn teek, mille märksõnad on saadaval automaatselt ning mida ei pea eraldi testi importima. Järgnevas näites (koodinäide 4) kaasatakse testi kolm erinevat teeki: standardne teek DateTime, väline teek Selenium2Library ja kasutaja enda teek lib/KasutajaTeek.py. (Robot Framework Quick Start Guide, 2017)

```
*** Settings ***
Library           DateTime
Library           Selenium2Library
Library           lib/KasutajaTeek.py
```

Koodinäide 4. Teekide lisamine testloosse.

Kõrgema taseme märksõnade loomist peetakse Robot Framework raamistiku üheks olulisemaks omaduseks. See annab kasutajale võimaluse kerge vaevaga defineerida enda töös vajalikke märksõnu, et lihtsustada tüüpilisemate sammude korduvat kasutamist ning hoida testid lihtsad ja kergesti loetavad. (Robot Framework Quick Start Guide, 2017)

Kõrgema taseme märksõnade loomiseks kasutatakse märksõnade tabelit. See erineb tavalisest testloost vaid selle poolest, et identifitseerimiseks kasutatakse teistsugust tabeli nimetust (`***Keywords***`) ning raamistik ei käivita neid kui eraldiseisvaid testlugusid, vaid teiste testlugude sees. Kõrgema taseme märksõnad pannakse kokku kas madalama taseme märksõnadest või teistest kasutaja poolt loodud märksõnadest. (Robot Framework Foundation, 2017)

Koodinäide 5 kirjeldab, kuidas luua kõrgema taseme märksõna kasutades selleks Selenium2Library teegist pärinevaid madalama taseme märksõnu. Eesmärk on avada veebibrauser, navigeerida Google otsingumootorisse ning alustada otsingut sõnaga „Robot Framework“. Edaspidi piisab terve protsessi läbimiseks sellest, et testis kasutada märksõna „Search Robot Framework from Google“.

```
*** Settings ***
# Selenium2Library teegi importimine
Library           Selenium2Library

*** Keywords ***
Search Robot Framework from Google #märksõna nimetus
# Märksõna ja argumendi paarid
    Open Browser    googlechrome
    Go To           https://www.google.ee/
    Input Text      id=lst-ib    Robot Framework
    Click Button    name=btnG
```

Koodinäide 5. Kõrgema taseme märksõna loomine.

Testlugudes või märksõnades kasutatavad sisendid tasuks defineerida muutujatena. Seda võib teha kas ühe testi või märksõna sees, terve komplekti tasandil või muutujate failis. Kuigi koodis ei erista väikeseid ja suuri tähti, siis soovitatakse globaalsetes muutujatest kasutada läbivalt suurtähti ning väiketähti muutujates, mida kasutatakse vaid kindlates testlugudes või märksõnades. Muutuja nimi on alati loogeliste sulgude vahel ning sulgude ette lisatakse muutuja tüübile vastav identifikaator. Eristatakse nelja tüüpi muutujaid: skalaartüüpi muutujat tähistatakse $-märgiga, massiivi @-ga, sõnastikku &-ga ja keskkonna muutujat %-ga. (Robot Framework Foundation, 2017) Koodinäide 6 selgitab, kuidas eelnevas näites püsiprogrammeeritud andmed asendada defineeritud muutujatega.

```
*** Settings ***
Library           Selenium2Library

*** Variables ***
# Muutujate defineerimine
${BRAUSER}          googlechrome
${URL}              https://www.google.ee/
${OTSINGUSONA}      Robot Framework

*** Keywords ***
Search Robot Framework from Google
    Open Browser    ${BRAUSER}
    Go To           ${URL}
    Input Text      id=lst-ib    ${OTSINGUSONA}
    Click Button    name=btnG
```

Koodinäide 6. Muutujate defineerimine kõrgema taseme märksõnas.

Teste hoitakse testi failis ning need on automaatselt osa mõnest testide komplektist. Ühte komplekti võib kuuluda üks kuni mitu testi – struktuur on paindlik. Näiteks ühe testi (koodinäide 7) juurde võivad lisaks testloo sammudele kuuluda ka seaded (milliseid tegevusi sooritada enne ja pärast testimist jpm) ning kõrgema taseme märksõnade ja muutujate

defineerimine. See tähendab aga seda, et nende kasutamine on võimalik ainult konkreetse testloo raames. (Robot Framework Foundation, 2017)

Järgnevas testis (koodinäide 7) kõigepealt imporditakse vajalik teek ning määratakse, mis juhtub enne ja mis pärast testlugu. Seejärel defineeritakse muutujad, mida omakorda kasutatakse kas märksõnades või testloos. Pärast seda kirjeldatakse märksõnad ning alles siis on kõik testlooks vajalikud komponendid olemas. Testlugu on märgistatud sildiga „Example“. Testlugu otsib Google otsingumootorist sõnapaari „Robot Framework“ ja ootab kuni otsingutulemuste arv on lehel nähtav. Ootamine on vajalik selleks, et test ei oleks kiirem kui otsingumootor ning ei põhjustaks vale-negatiivset testi tulemust. Kui tulemuste arv on veebilehele jõudnud, veendutakse, kas otsingusõna on lehel esindatud. Kui otsingusõna leitakse, siis lõppeb test positiivse tulemusega. Kui mingil põhjusel otsingusõna lehelt ei leita, siis test lõppeb negatiivse tulemusega. See on märguanne, et tasuks uurida, kas funktsionaalsus on ikka töökorras või mitte.

```
*** Settings ***
# Selenium2Library teegi importimine
# Enne testlugu brauser avatakse ja pärast testi suletakse
Library           Selenium2Library
Test Setup        Open Web Browser
Test Teardown     Close All Browsers

*** Variables ***
${BRAUSER}          googlechrome
${URL}              https://www.google.ee/
${OTSINGUSONA}      Robot Framework

*** Keywords ***
#  Märksõna  Open  Web  Browser  avab  muutujates  defineeritud
veebibrauseri
#  Märksõna  Search  Robot  Framework  from  Google  otsib  Google
otsingumootorist  muutujates defineeritud väärtust
Open Web Browser
    Open Browser    ${BRAUSER}
Search Robot Framework from Google
    Go To           ${URL}
    Input Text      id=lst-ib    ${OTSINGUSONA}
    Click Button    name=btnG
```

```
*** Test Cases ***
# Test ootab kuni tulemuste arv kuvatakse lehele
# Alles seejärel test veendub, et leht sisaldab otsitud märksõna
[Tags] Example   # [Tags] defineerib testloole sildi
Search Robot Framework from Google
Wait Until Page Contains Element    id=resultStats
Page Should Contain                 ${OTSINGUSONA}
```

Koodinäide 7. Testi näide.

Kõiki nimetatud osasid (muutujad, märksõnad, seaded) võib määratleda ka terve testide komplekti tasandil nii, et need on kättesaadavad rohkematele testlugudele. Ressursifailides hoitakse kõrgema taseme märksõnu koos muutujatega ja muutujate failides hoitakse muutujaid koos sobilike väärtustega ning need saavad olla ka globaalselt kättesaadavad kõikidele testide komplektidele ja testlugudele. (Robot Framework Foundation, 2017)

Kui eelnevalt kirjeldatud näited olid märksõnakesksed testlood, siis järgnev näide (koodinäide 8) on käitumiskeskne testlugu. Sellistes testlugudes kirjeldatakse süsteemi kriteeriumeid ja samme lisades märksõnadele erinevaid prefikseid nagu `Given`, `When`, `Then`, `But` ja `And`. `Given` eesliitele järgnev märksõna kirjeldab süsteemi esialgset seisukorda, `When` näitab, milliseid tegevusi sooritatakse ning `Then` defineerib, milline on oodatud tulemus süsteemis pärast tegevuste sooritamist (Robot Framework Foundation, 2017). Kokku on testis kasutusel neli erinevat märksõna (`login page is open`, `login button is pressed` jt) ja nende tegelik funktsionaalsus on kirjeldatud väljaspool testi.

```
*** Test Cases ***
Failed Login
   Given login page is open
   When invalid username and password are inserted
   And login button is pressed
   Then error page should be open
```

Koodinäide 8. Käitumiskeskne testlugu.

Testimismallide kasutamine testlugudes võimaldab muuta märksõnakesksed testid andmekeskseteks. Sellised testid on kasulikud siis, kui sama testimisstsenaariumit on vaja läbida mitmeid kordi erinevate andmetega (Robot Framework Foundation, 2017)

Järgnev näide (koodinäide 9) on andmekeskne testlugu. Testis on defineeritud märksõna ning sedasama märksõna kasutatakse kõikides testlugudes mallina. Kokku koostatakse malli abil seitse erinevat testlugu, kõigil ühine eesmärk – testida sisselogimise ebaõnnestumist. Testloo sammud on kõikidel juhtudel samad, aga vahetuvad muutujad, millega süsteemi proovitakse

sisselogida. Märksõnas `Failed Login` on defineeritud kaks muutujat argumentidena – kasutajanimi ja parool. Et loodud märksõna on defineeritud kui testlugude mall, siis seetõttu eeldatakse, et igale testloole antakse samuti kaasa kaks sisendit, millest üks tähistab kasutajanime ja teine parooli. Testandmetena on kasutusel neli erinevat väärtust – korrektsed kasutajanimi (kasutaja123) ja parool (parool456), mis töötavad antud süsteemi sisselogimisel, tühi sisend ja vale sisend (ValeSisend). Esimesed kolm neist on defineeritud muutujatena, viimane on testlugude sisse kirjutatud. Mitte ühelgi juhul ei sisestata kasutajanime ja parooli õiges kombinatsioonis ja seetõttu peaks süsteemi sisselogimine ebaõnnestuma kõikidel kordadel. Kuna eesmärk on testida negatiivseid stsenaariumeid, siis ebaõnnestunud sisselogimise korral on testimise tulemus positiivne. Kui valitud andmetega saab süsteemi sisselogida, siis selle testloo tulemus on negatiivne.

```
# Märksõna Failed Login defineerimine testlugude mallina
*** Settings ***
Test Template         Failed Login
*** Variables ***
${VALID USER}         kasutaja123
${VALID PASSWORD}     parool456
${EMPTY}

# Esimeses veerus on testloo nimi
# Teises veerus on argumendi ${USERNAME} väärtus
# Kolmandas veerus on argumendi ${PASSWORD} väärtus
*** Test Cases ***    USERNAME             PASSWORD
Invalid username      ValeSisend           ${VALID PASSWORD}
Invalid password      ${VALID USER}        ValeSisend
Invalid both          ValeSisend           ValeSisend
Empty username        ${EMPTY}             ${VALID PASSWORD}
Empty password        ${VALID USER}        ${EMPTY}
Empty both            ${EMPTY}             ${EMPTY}
Values mixed          ${VALID PASSWORD}    ${VALID USER}

*** Keywords ***
Failed Login
    [Arguments]    ${USERNAME}    ${PASSWORD}
    Input username   ${USERNAME}
    Input password   ${PASSWORD}
    Sumbit login
    Error page should be open
```

Koodinäide 9. Testimismalli kasutamine andmepõhises testloos.

2.4 Testide sooritamine Robot Framework raamistikus

Robot Framework raamistiku teste saab käivitada käsurealt või valitud redaktori, nagu RIDE, vahendusel. Käsureal saab argumendina kaasa anda testandmeid ning erinevate märgete lisamisega saab mõjutada testide sooritamist ja loodavaid väljundeid. (Robot Framework Foundation, 2017)

Koodinäide 10 tutvustab kolme erinevat viisi valitud testide (komplekti) poole pöördumiseks. Tee võib olla osaline või täielik ning viidata üksikule testile (sh üksikute testide jadale) või tervele testide komplektile.

```
robot testid.robot
robot tee/minu/testideni/
robot c:/robot/testid.robot
```

Koodinäide 10. Testide käivitamine käsurealt kolmel erineval viisil.

Testide koostamise järgselt kogub Robot Framework raamistik saadud informatsiooni kokku. Tavaliselt koostatakse selle põhjal kolm dokumenti – väljundifail XML formaadis, raport HTML-is ja logi. Hiljem on neid võimalik kokku liita üheks tulemuste aruandeks (Robot Framework Foundation, 2017)

Väljundifail XML-is sisaldab kogu informatsiooni testimise tulemuste kohta ning ka logifail (vt Lisa 2) ja raport (vt Lisa 3) HTML-is koostatakse selle põhjal. Logis on struktureeritult kirjas kõik detailid testide sooritamise kohta ning see annab täpse ülevaate kaasatud testimiskomplektidest, -mallidest ja märksõnadest. Raport seevastu on pigem ülevaatlikum dokument kogu protsessist ja sisaldab põhjalikke statistilisi andmeid. (Robot Framework Foundation, 2017)

Standardsete valikute kõrval on võimalik koostada mitmesuguseid infofaile. Samuti saab muuta tulemuste detailsust lisades käsureal erinevaid argumente. Käesoleva töö kontekstis ei pea autor vajalikuks lähemalt tutvustada nimetatud iseärasusi.

3 Automaattestide koostamine pangandustarkvarale

Järgnev peatükk keskendub automaattestimise korraldamisele pangandustarkvaral kasutades selleks eelnevas peatükis tutvustatud Robot Framework raamistikku. Peatükk algab metoodika ja kõnealuse tarkvara tutvustusega. Töö automaattestidega on jaotatud kolme ossa – ettevalmistav, kavandamise ja arendamise etapp. Peatüki viimases osas antakse hinnang tehtud tööle ja selle käigus toimunule ning selgitatakse, kuidas jätkata Robot Framework raamistiku ja pangandustarkvara koostööga tulevikus.

3.1 Metoodika tutvustus

Käesoleva peatüki jooksul soovib töö autor jõuda arusaamale, kas Robot Framework raamistik on sobilik lahendus pangandustarkvara automaattestimiseks. Peatüki jooksul viiakse läbi nn kontseptsiooni tõestus, mis selgitab kas kõnealune raamistik on sobilik töövahend pangandustarkvara automaattestimiseks, või tuleks valida midagi muud oma eesmärkide saavutamiseks.

Seatud eesmärgi saavutamiseks selgitab töö autor peatüki jooksul järgnevaid aspekte:

- pangandustarkvara automaattestimise üldine eesmärk;
- ootused Robot Framework raamistikule;
- Robot Framework raamistiku töövahendite valimine;
- skoobi määramine, testide kavandamine;
- testide arendamine;
- testide sooritamine.

Töös on loetletud punktid jaotatud kolme ossa. Esimene osa on ettevalmistav etapp, mis selgitab **miks** on vaja automaattestida kõnealust tarkvara, ja milliseid Robot Framework raamistiku vahendeid selleks kasutada. Sellele järgneb kavandamise etapp, kus selgub, **mida** kavatsetakse automatiseerida ehk koostatakse plaan testlugudest ja nendele vajalikest märksõnadest. Viimaseks on arenduse etapp, mis näitab, **kuidas** eelnenud etapis planeeritu ellu viia. Kolme etapi järgselt antakse hinnang saadud kogemusele ja selgitatakse, kas ja kuidas tehtut edaspidi kasutada saaks.

3.2 Pangandustarkvara tutvustus

Töö autor kasutab Robot Framework raamistiku pangandustarkvara automaattestimisel. Tegemist on era- ja äriklientide teenindamiseks mõeldud terviklahendusega, mille abil saavad pangad oma klientidele kiirelt ja kvaliteetselt erinevaid pangateenuseid pakkuda. Lahendus koosneb panga töötajatele mõeldud süsteemist ehk tellerirakendusest ja iseteeninduskeskkonnast panga klientidele ehk internetipangast. Tarkvara on konfigureeritav vastavalt ärivajadustele ning korraga saab kasutada kas tervet tarkvara või ainult üksikuid mooduleid. Seega võivad erinevatel klientidel kasutuses olevad rakendused teineteisest kardinaalselt erineda.

Tellerirakendus ja internetipank on kasutajale kättesaadavad veebibrauseri vahendusel. Tellerirakendust kasutavad panga töötajad oma igapäevasteks töötoiminguteks. Kõige tüüpilisemad tegevused tellerirakenduses on järgmised:

- kliendi registreerimine, kliendi andmete ja internetipanga kasutusõiguste haldamine;
- pangakontode avamine, sulgemine ja muud kontodega seotud toimingud;
- sularaha ja rahaülekannetega seotud tegevused (sh erinevad maksed, püsikorraldused);
- laenudega seotud toimingud;
- hoiustega seotud toimingud;
- kliendisuhtluse korraldamine (pangateadete saatmine erinevatele huvigruppidele, kliendi päringutele vastamine).

Internetipanka kasutab panga klient informatsiooni saamiseks ja erinevate tegevuste sooritamiseks. Internetipanga kõige tihedamini kasutust leidev funktsionaalsus on järgmine:

- ülevaade kontodest, nende jääkidest ja tehtud toimingutest (sh detailne ülevaade laekunud ja tehtud maksete kohta);
- riigisiseste ja väliste maksete tegemine;
- püsikorralduste ja määratud maksete haldamine;
- valuutakurssidega tutvumine ja välisvaluuta vahetus;
- pangaga suhtlemine (saabunud teadete lugemine, teadete ja päringute saatmine).

Eelnevalt loetletud funktsionaalsust kasutatakse igapäevaselt paljude erinevate inimeste poolt ning seetõttu on need tarkvara osad ka kõige ärikriitilisemad. Toote klientide igapäevased

äritoimingud saaksid tugevalt kahjustada, kui mõni nimetatud toote osadest peaks saama uute arenduste käigus märkamatult kahjustatud. Selline funktsionaalsus tuleks esmajärjekorras vajalikul määral automaattestidega katta.

3.3 Ettevalmistav etapp

Eelnevas peatükis tutvustatud pangandustarkvara automaattestimise vajadus tuleneb järgnevatest asjaoludest:

- terve süsteemi regressioonitestimine manuaalsete testimismeetoditega võtab väga palju aega;
- tarkvara vastutab mitmete ärikriitiliste protsesside eest;
- sama tarkvarauuendust on vaja testida korduvalt erinevates keskkondades, et olla veendunud lahenduse toimimises ka erinevate süsteemi konfiguratsioonide korral;
- arendusmeeskonnad kulutavad suure osa oma ajast selleks, et igas arendustsüklis veenduda, et kogu ülejäänud süsteem ei ole saanud haavata tehtud muudatustest;
- ka väikese arenduse puhul peab klient tellimuse täitmist pikalt ootama, sest tervet süsteemi hõlmav regressioonitestimine võtab ühepalju aega olenemata muudatuse suurusest.

Automaattestimise kaasamisega tööprotsessidesse soovitakse täita järgnevaid eesmärke:

- vähendada regressioonitestimiseks kuluvat aega;
- hoida ärikriitilisi protsesse töökorras;
- vähendada rutiinseid kordusi testijate töös;
- anda arendusmeeskondadele tagasisidet süsteemsete defektide kohta juba arenduse käigus;
- täita kliendi tellimusi senisest kiiremini;
- teha tarneid tihti ja senisest lühema ettevalmistusajaga.

Automaattestida soovitakse nii tellerirakendust kui ka internetipanka, sest mõlemast leiab tarkvara ärikriitilisi osasid. Kontseptsiooni tõestuse raames alustatakse testide automatiseerimist tellerirakenduses ning raamistiku sobivuse korral jätkatakse internetipangaga. Eesmärk on automatiseerida läbivteste (ingl k *end-to-end tests*) ning selle käigus kontrollida, kas tegevused rakenduses annavad oodatud tulemusi.

Automaattestimise raamistiku valikul lähtuti järgnevatest tingimustest:

- võimaldab automaattestida veebirakendusi;
- saab kasutada läbi kasutajaliidese toimuvate funktsionaaltestide koostamiseks;
- ei sõltu operatsioonisüsteemist;
- põhineb Python arenduskeelel;
- on vabavaraline ning ei vaja investeeringut litsentsitasudeks jmt.

Robot Framework raamistik suudab vastata kõikidele seatud tingimustele. Eelnevate kriteeriumite kõrval valiti Robot Framework raamistik esimeseks katsealuseks seetõttu, et kasutatava metoodikaga on testide kirjutamine lihtne ning ei nõua raamistiku kasutajalt pikaajalist programmeerimiskogemust. Nii on tagatud see, et piisava koolitusega suudavad automaattestimisse panustada paljud inimesed üle erinevate tiimide ning see ei jää vaid mõne üksiku automaattestimise entusiasti ja programmeerijate pärusmaaks.

Kuna kõnealust tarkvara kasutatakse veebibrauseri vahendusel, siis lisaks raamistikule endale on tarvis paigaldada ka Selenium2Library teek koos veebibrauserite draiveritega. Selle teegi abil tuvastatakse erinevaid veebilehe elemente. Kuna töö autoril puuduvad varasemalt kujunenud eelistused tekstiredaktorite osas, siis kasutatakse testide arendamiseks Robot Framework raamistiku redaktorit RIDE.

3.4 Planeerimise etapp

Planeerimise etapis valitakse välja automaattestimiseks sobivad tarkvara osad ning pannakse kirja valitud testlood ja nende märksõnad. Oma töö planeerimine enne automatiseerimisega alustamist aitab luua selgemat pilti erinevatest töö etappidest ja nende mahukusest.

Märksõnade valikul lähtutakse sellest, et kaetud saaksid tüüpilisemad tegevused, mida süsteemis korduvalt erinevates situatsioonides teha saab (nt lehtedele navigeerimine, erinevate nuppude vajutamine), ja et need lihtsustaksid testlugude kirjutamist. Testlugude valikusse lisatakse sellised stsenaariumid, mida süsteemis kõige tihedamini tehakse või millel on suur mõju teistele süsteemi osadele.

3.4.1 Märksõnade valik

Järgnevas tabelis (tabel 3) on plaan märksõnadest, mida soovitakse koostada arendamise etapis. Neid märksõnu kasutatakse testlugude sees. Seetõttu ei pea testlugudes pikalt ja tehniliselt mingisugust tegevust kirjeldama, vaid piisab sellest, kui kasutada õiget märksõna, mis oma nimetusega juba annab piisava selgituse, mida süsteemis parasjagu tehakse.

Tabel 3. Märksõnade plaan.

Ülesanne	**Kirjeldus**
Ebaõnnestunud sisselogimine	1) Tellerirakenduse avamine; 2) kasutajanime ja parooli sisestamine; 3) sisse logimise nupu vajutamine. Märksõna kontrollib, kas lehel on kuvatud info vigaste andmete kohta.
Õnnestunud sisselogimine	1) Tellerirakenduse avamine; 2) kasutajanime ja parooli sisestamine; 3) sisse logimise nupu vajutamine. Märksõna kontrollib, kas süsteem on tellerirakenduse avalehel.
Leheküljele navigeerimine	1) Peamenüü jaotise valimine; 2) alammenüü jaotise valimine. Märksõna kontrollib, kas süsteem jõudis oodatud leheküljele.
Kirjete loetelu kuvamine	1) Otsingu nupu vajutamine leheküljel. Märksõna kontrollib, et tulemuste tabel oleks lehel kuvatud.
Kirje detailandmete vaatamine	1) Tulemuste nimekirjast ühe kirje valimine; 2) vaata/muuda nupu vajutamine. Märksõna kontrollib, kas avati õige detailandmete lehekülg.
Uue kirje lisamine	1) Leheküljel vajutatakse uue kirje loomise nuppu. Märksõna kontrollib, kas leheküljel on avanenud õige detailandmete vorm.
Kirje salvestamine	1) Detailandmete vormil vajutatakse salvestamise nuppu. Märksõna kontrollib kirje salvestamist.
Kirje kinnitamine	1) Detailandmete vormil vajutatakse kinnitamise nuppu. Märksõna kontrollib kirje kinnitamist.

Ülesanne	Kirjeldus
Andmete sisestamine detailandmete vormil	1) Avatud vormil nõutud lahtrite täitmine.
Detailandmete vormilt lahkumine	1) Leheküljel vajutatakse tagasi nuppu. Märksõna kontrollib, et süsteem navigeeriks õigele lehele.
Kliendi valimine lehekülje päises	1) Otsingu nupu vajutamine lehekülje päises; 2) otsitava kliendi nime(osa) sisestamine; 3) kliendi valik tulemuste nimekirjast. Märksõnal tuleb põhiaknas veenduda, kas klient sai korrektselt valitud.
Süsteemist väljalogimine	1) Väljalogimise nupu vajutamine. Märksõna peab veenduma, et väljalogimine õnnestus.

3.4.2 Testlugude valik

Järgnevas tabelis (tabel 4) on plaan automatiseeritavatest testlugudest. Loodud plaan annab ülevaate testlugude ülesannetest, täitmise eeldustest ja kirjelduse testloo sammudest. Valiku tegemisel lähtuti põhimõttest, et tegemist oleks funktsionaalsusega, mida süsteemis kõige tihedamini läbitakse, ja mille katki minemine põhjustaks ärilisest vaatest väga suuri katkestusi panga töös. Testlood planeeriti selliselt, et need oleksid läbivtestid. See tähendab seda, et iga testlugu algab ja lõppeb töövoo loogilistes punktides ning katab konkreetset funktsionaalsust terves ulatuses. Kõikides loetletud testlugudes (va ebaõnnestunud sisselogimine, õnnestunud sisselogimine) eeldatakse, et süsteemi ollakse sisse logitud.

Tabel 4. Testlugude plaan.

Ülesanne	Eeldus	Kirjeldus
Ebaõnnestunud sisselogimine	Puuduvad	Sisselogimine toimub ebakorrektsete andmetega ning kasutajale kuvatakse sellekohast teadet.
Õnnestunud sisselogimine	Süsteemi kasutajakonto olemasolu	Sisselogimine toimub korrektsete andmetega ning kasutajale ei kuvata infot vigastest andmetest.

Ülesanne	Eeldus	Kirjeldus
Kliendi registreerimine	Puuduvad	Navigeeritakse kliendi registreerimise vormile, sisestatakse andmed, salvestatakse andmed ja kontrollitakse tulemust.
Kliendi andmete muutmine	Klient on registreeritud	Valitakse klient, avatakse kliendiandmete vorm, muudetakse andmed ja kontrollitakse tulemust.
Pangakonto avamine	Klient on registreeritud	Valitakse klient, navigeeritakse pangakonto avamise vormile. Avanenud vormil määratakse konto parameetrid ja konto salvestatakse.
Pangakonto sulgemine	Konto olemasolu	Valitakse klient, navigeeritakse pangakontode vormile, valitakse kliendi konto ja vajutatakse konto sulgemise nuppu. Avanenud vormil prinditakse sulgemise vorm ning kinnitatakse konto sulgemine.
Pangaülekanne kliendi kontolt	Positiivse saldoga konto olemasolu	Valitakse klient, navigeeritakse maksekorralduse vormile. Vorm täidetakse maksekorralduse andmetega ning salvestatakse ja kinnitatakse.
Sularaha sissemakse kliendi kontole	Konto olemasolu	Valitakse klient, navigeeritakse uue sularaha sissemakse vormile. Vormil sisestatakse nõutud andmed ning tehing salvestatakse ja kinnitatakse.
Sularaha väljamakse kliendi kontolt	Positiivse saldoga konto olemasolu	Valitakse klient, navigeeritakse uue sularaha väljamakse vormile. Vormil sisestatakse nõutud andmed ning tehing salvestatakse ja kinnitatakse.
Süsteemist väljalogimine	Süsteemi on sisse logitud	Süsteemist logitakse välja vajutades väljalogimise nuppu.

3.5 Arendamise etapp

Arendamise etapis valmisid planeeritud märksõnad ja testlood ning defineeriti vajalikud seaded ja muutujad. Lisaks planeeritud märksõnadele ja testlugude valmis mitmeid märksõnu, mida saab kasutada vaid konkreetsete vormide testimisel.

Automaattestide arendamisel valiti alati kõige lühem tee konkreetse tegevuse sooritamiseks ning erinevate vormide täitmisel kasutati vähima suurusega testandmete komplekte. Samuti

jälgiti arendamisel seda, et märksõnades ja testlugudes toimuks pidev süsteemi seisundi kontroll. Automaattestid, mis sisestavad rakenduses andmeid ja klõpsavad erinevaid nuppe, aga oma tegevuse käigus midagi ei kontrolli, on madala kasuteguriga. Näiteks võib automaattest küll vajutada ettenähtud nupule, aga süsteemi vea tõttu sattuda valele või katkisele lehele. Kui automaattest ei kontrolli seda, mis parasjagu süsteemis toimub, siis tekkinud vead jäävad avastamata. Sellisel juhul tõestab automaattest vaid seda, et tegevust saab sooritada, ent ei taga ootuspärast tegevuse tulemust.

3.5.1 Üldised sätted ja muutujad

Järgnevas koodinäites (koodinäide 11) kirjeldatakse üldisi automaattestimise sätteid ja testides kasutatavaid muutujaid. Komplekti tasandil defineeritakse see, et kõigepealt avatakse veebibrauser (Google Chrome) ja navigeeritakse süsteemi avalehele. Alles seejärel käivitatakse testid. Kui kõik testid on sooritatud, siis veebibrauser suletakse. Samuti on siinkohal defineeritud Selenium2Library teegi kasutamine ning seetõttu ei pea seda teeki eraldi igasse testi importima.

Loetletud muutujaid kasutatakse testides (vt Lisa 4). Muutujate defineerimine sellisel kujul on kasulik seetõttu, et kui muutuja väärtus peaks muutuma, siis parandus on vaja sisse viia vaid ühes kohas, mitte aga igas testis, mis kasutab konkreetset muutujat.

Antud juhul on testide komplekti tasandil on defineeritud need muutujad, mis puudutavad üldisi sätteid või mida kasutatakse testlugudes. Enamjaolt on tegemist erinevate süsteemi poolt antavate teadetega. Mitmed muutujad on defineeritud ka kasutaja poolt loodud märksõnade argumentidena.

```
*** Settings ***
Suite Setup       Open Browser    ${URL}    ${BROWSER}
Suite Teardown    Close All Browsers
Library           Selenium2Library

*** Variables ***
${BROWSER}        gc
${URL}            localhost
${VALID_USERNAME}    dea
${VALID_PASSWORD}    dea
${UNSUCCESFUL_LOGIN_MESSAGE}    Wrong username/password
${TIMEOUT_MESSAGE}    Your session has timed out!
${LOGOUT_MESSAGE}    You have logged out.
${SAVE_MESSAGE_NEW_CLIENT_REG}    New customer added to registry.
```

```
${SAVE_MESSAGE_UPDATE_CLIENT_REG}    Customer data updated!
${SAVE_MESSAGE_NEW_ACCOUNT}    Account opened successfully!
${SAVE_MESSAGE_CASH_DEPOSIT}      Order is saved and needs to be
confirmed!
${CONFIRM_MESSAGE_CASH_DEPOSIT}    Order is confirmed!
${SAVE_MESSAGE_PAYMENT_ORDER}      Order is saved and needs to be
confirmed!
${CONFIRM MESSAGE PAYMENT ORDER}    Order is confirmed!
${SAVE_MESSAGE_CASH_WITHDRAWAL}     Order is saved and needs to be
confirmed!
${CONFIRM_MESSAGE_CASH_WITHDRAWAL}    Order is confirmed!
${CLOSE_MESSAGE_ACCOUNT}    Account is closed!
```

Koodinäide 11. Üldised sätted ja muutujad testide komplekti tasandil.

3.5.2 Märksõnade koostamine

Automaattestide arendamise käigus valmis 15 märksõna. Planeeritud märksõnade kõrval kirjutati ka mitu sellist märksõna, mida kasutatakse konkreetsetel vormidel andmete sisestamisel, ning mis ei olnud esialgses plaanis. Järgnevalt selgitatakse kolme universaalse märksõna toimimist. Kõikide märksõnade detailsed kirjeldused ja informatsioon testides kasutamise kohta on saadaval töö lisas (vt Lisa 4).

`Navigate to page` (koodinäide 12) märksõna kasutatakse menüüs navigeerimiseks. Testloos kasutatakse seda nii, et lisaks märksõna nimetusele antakse argumentidena kaasa kolm väärtust – peamenüü nimetus, alammenüü nimetus ja leheküle nimi, kuhu välja peaks jõudma. Märksõnas on defineeritud järgmised sammud. Kõigepealt vajutatakse peamenüü linki ja siis oodatakse kuni alammenüü on nähtav, sest vastaselt juhul on test liiga kiire rakenduse jaoks ja põhjustab vale-negatiivset tulemust. Seejärel vajutatakse alammenüü linki ning viimaks kontrollitakse, kas rakendus on jõudnud õigele lehele.

```
***Keywords***
Navigate to page
    [Arguments]    ${mainmenu}    ${submenu}    ${pagetitle}
    Select Frame    name=menu
    Click Link    link =${mainmenu}
    Wait Until Element Is Visible    link=${submenu}
    Click Link    link=${submenu}
    Page Should Contain    ${pagetitle}
    Select Frame    name=WORK
```

Koodinäide 12. Navigate to page märksõna.

`Fill in payment order form` (koodinäide 13) märksõna ülesandeks on maksekorralduse vormil andmete sisestamine. Märksõna kasutades tuleb argumentidena kaasa anda makse saaja nimi, tema pangakonto number, makse summa ja selgitus. Andmete

sisestamisel järel vajutatakse tasu arvestamise nuppu ning oodatakse kuni leheküljel kuvatakse teadet tasu arvestamise kohta.

```
***Keywords***
Fill in payment order form
    [Arguments]      ${ben_name}      ${ben_account}      ${amount}
${explanation}
    [Documentation]    Maksekorralduse vormil andmete sisestamine
    Input Text    name=beneficiary_name    ${ben_name}
    Input Text    name=ben_acc_number    ${ben_account}
    Input Text    name=amount    ${amount}
    Input Text    name=explanation    ${explanation}
    Click Button    name=fee_calc
    Wait Until Page Contains    Fee calculated!
```

Koodinäide 13. Fill in payment order form märksõna.

Märksõna `Confirm record` (koodinäide 14) kinnitab kirje selle salvestamise järgselt ning kontrollib, et leheküljel oleks kuvatud korrektne kinnitamise teade.

```
***Keywords***
Confirm record
    [Arguments]    ${confirm_message}
    [Documentation]    Kirje kinnitamine
    Wait Until Element Is Enabled    name=btn_confirm
    Click Button    name=btn_confirm
    Page Should Contain    ${confirm_message}
```

Koodinäide 14. Confirm record märksõna.

3.5.3 Testlugude koostamine

Arenduse käigus valmis kokku 10 testlugu. Järgnevalt on kirjeldatud kolm testlugu. Ülejäänud testide kohta leiab ülevaate käesoleva töö lisast (vt Lisa 4).

Järgnevas kahes testloos (koodinäide 15) automaattestitakse süsteemi sisse logimist. Esimesel juhul tehakse seda vigaste andmetega, sisselogimine ei õnnestu ja vormil kuvatakse teadet vigaste andmete kohta. Teisel juhul kasutatakse korrektseid andmeid ning sisselogimine peaks õnnestuma. Mõlemal juhul on tegemist andmekesksete testidega, kus defineeritud märksõna kasutatakse mallina ning testloos määratakse vaid sobivad testandmed. Dokumenteerimiseks on kasutatud `[Documentation]` silti, et anda kaasa lühike kirjeldus testis toimuvast.

```
***Test Cases***
T1: Login failed
    [Documentation]    Ebaõnnestunud sisselogimine
    [Template]    Failed log in to system
    valekasutajanimi    valeparool
    valeparool    valekasutajanimi

T2: Successful login
    [Documentation]    Õnnestunud sisselogimine
    [Template]    Successful log in to system
    ${validusername}    ${validpassword}
```

Koodinäide 15. Testlood - süsteemi sisselogimine ebakorrektsete ja korrektsete andmetega.

Järgneva testloo (koodinäide 16) raames registreeritakse süsteemis klient. Testlugu on ära kirjeldatud vaid nelja erineva märksõna abil, kusjuures kolm neist on universaalsed märksõnad, mida saab kõikides testides kasutada ning üks konkreetselt registreerimise vormiga seotud. Testlugu algab vormile navigeerimisega, kus vajutatakse uue kliendi registreerimise nuppu. Avanenud vorm täidetakse testloos märgitud andmetega ja salvestatakse. Eraisiku registreerimiseks on vaja määrata klienditüüp ning teada tema isikukoodi, ees- ja perekonnanime, telefoninumbrit jaa sünniaega. Viimasena kontrollitakse, et lehel oleks kuvatud salvestamise teade.

```
T3: New client registration
    [Documentation]    Uue kliendi registreerimine
    Navigate to page    Customer    Registry    REGISTRY
    New record    REGISTRY
    Fill new user form (private person)    Private individual (10)
37810019940    Oskar    Ohakas    5550100203    01.10.1978
    Save record    ${SAVE_MESSAGE_NEW_CLIENT_REG}
```

Koodinäide 16. Testlugu – uue kliendi registreerimine.

3.6 Robot Framework raamistiku sobivus pangandustarkvara automaattestimiseks

Automaattestide kirjutamise käigus selgus mitmeid olulisi nüansse kõnealuse pangandustarkvara kohta, mis mõjutavad automaattestimise korraldamist. Töö autori arvates on tegemist selliste aspektidega, mis tuleks läbi mõelda ja lahendada enne lõpliku automaattestimise töövahendi valimist.

Kuna tarkvara on konfigureeritav vastavalt kliendi vajadustele, siis seetõttu võivad ka testimiskeskkonnad erineda seal kasutuses oleva funktsionaalsuse poolest. Standardne funktsionaalsus on kättesaadav kõikidel tingimustel ning ei sõltu sellest, milline keskkond on

parasjagu testimiseks valitud. Konfigureeritav funktsionaalsus võib olla kättesaadav ainult siis, kui testimiseks on valitud spetsiifiline testimiskeskkond.

Olukorra täpsustamiseks tuleks autori arvates funktsionaalsust hinnata kahest aspektist lähtuvalt. Esiteks tuleks välja selgitada, kui suur osa kogu tarkvara funktsionaalsusest on konfigureeritav ja ainult teatud tingimustel testitav, ning seejärel määrata konfigureeritava funktsionaalsuse kriitilisus. Situatsiooni lahendamiseks on mitu võimalust. Esiteks võiks kaaluda skoobi kitsendamist – näiteks automaattestitakse ainult sellist funktsionaalsust, mis kuulub standardse funktsionaalsuse hulka või vastab mõnele muule kindlale kriteeriumile (kriitiline, enim kasutatav, kergesti haavatav teistest süsteemi muudatustest jne). Robot Framework raamistik pakub ka testide sildistamist. Automaattestide käivitamisel saab täpsustada, millise sildiga teste sooritada. Kui testil puudub hetkel kasutuses oleva testimiskeskkonna silt, siis see test jäetakse vahele. Lisaks võiks kaaluda sellise mehhanismi ehitamist, mis enne testi sooritamist kontrollib, kas konkreetne funktsionaalsus, mida soovitakse testida, on hetkel kastutatavas keskkonnas üldse olemas.

Samuti on osa funktsionaalsusest seotud kindlate kasutajaõigustega. Kui sisse loginud kasutajal puuduvad vastavad õigused, siis konkreetne funktsionaalsus ei ole kasutajale kättesaadav. Seega peaks automaattestide jaoks süsteemis eksisteerima sobiv kasutaja või peaks selle looma testidele eelneva seadistamise käigus.

Järgmiseks mõttekohaks on see, mida teha sellise funktsionaalsusega, mis on üksteisest tugevalt sõltuv ja ajaliselt distantseeritud. Sellistel juhtudel on üsna keeruline protsessi algusest lõpuni automaattestida. Näiteks selleks, et panga kliendile saaks teller teha tähtajalise hoiuse, peab eelnevalt olema täidetud mitu tingimust: isik on kliendina registreeritud, tal on avatud pangakonto ja sellel on piisavalt raha. Üks lahendus on kombineerida mitu testlugu üheks suuremaks testlooks, mida läbitakse kindlas järjekorras. Näiteks alustatakse kliendi registreerimisega, seejärel avatakse talle konto, siis tehakse sellele kontole sularaha sissemakse ning viimase sammuna alles testitakse tähtajalise hoiuse avamist. Teine variant on see, et tähtajalise hoiuse avamise test seab andmebaasi ise üles ka vajalikud andmed ning pärast testi lõppemist taastab andmebaasis esialgse olukorra.

Ajaliselt distantseeritud funktsionaalsus tähendab seda, et mingi protsess on küll käivitatud, aga see realiseerub alles siis, kui kätte on jõudnud nö „õige aeg“. Näiteks kui teller tööpäeva lõpus

algatab makse välismaale, siis kinnitatuks saab märkida selle alles järgmise tööpäeva hommikul ja selle hetkeni on makse töötlemisel. Selline olukord tuleneb rahvusvaheliste arveldussüsteemide töökellaaegadest. Kui automaattest tahab veenduda, et makse jõuab pärast makse tegemist ka õigesse staatusesse, siis peab test ka vajadusel ootama „õige aja“ saabumist või sekkuma süsteemi ajakäsitlusse.

Robot Framework raamistiku kasutamisel ilmnesid mitmed üllatavad asjaolud. Esimeseks komistuskiviks on raamistiku paigaldamine. Selleks, et kogu lahendus korrektselt tööle saada, tuleb paigaldada mitmeid erinevaid komponente. See protsess tuleb läbida õiges järjekorras ning valides erinevate osade õigeid versioone, sest kõik ei sobi omavahel kokku. Vastasel juhul ei hakka lahendus lihtsalt korralikult tööle. Raamistiku enda paigaldamine on küll lihtne, ent väliste teekide ja töövahendite paigaldamine nõuab täpset reeglite järgimist. Lisaks sellele kipuvad asjakohased juhendid olema liiga pealiskaudsed. Näiteks Selenium2Library teegi esmakordsel kasutamisel selgus tõsiasi, et teek vajab ka vastava veebibrauseri draiverit, et oskaks brauseriga suhelda. Samas ei leidnud töö autor vastavat informatsiooni paigaldamise juhendist ning see erines ka varasemast Robot Framework raamistiku kasutamise kogemusest, kus teek ei vajanud töötamiseks draiverit.

Töö autor leiab, et testide koostamine märksõnadel põhineva metoodikaga on küllaltki lihtne ning usub, et sellega võiks hakkama saada pea igaüks. Testide kirjutamine eeldab õigete märksõnade kasutamist funktsionaalsuse sammude kirjeldamistel. Kui valmistada ette piisav kogus kõrgema taseme märksõnu ning testija on kogenud raamistiku kasutaja, siis ühe testi automatiseerimise võiks võtta sama palju aega, kui manuaalse testi ettevalmistamine paberil. Raamistiku kasutamise alguses võtab testlugude ja selleks sobivate märksõnade loomine kindlasti rohkem aega, kui ühekordselt manuaalsete testide ettevalmistamine ja läbimine. Samas on ajavõit saavutatud juba järgmiste automaattestide kasutuskordadel manuaalse testimise asemel.

Terve automaattestide komplekti läbimine ühes keskkonnas võtab aega umbes 1 minut, mis on oluline ajavõit võrreldes manuaalse testimisega. Samas tekkisid töö autoril mõned kahtlused testide stabiilsuse osas. Näiteks esineb olukordi, kus ühel testide käivitamisel element leitakse üles ning teisel korral mitte. Üks oluline märkus siinjuures on see, et Robot Framework raamistik on oma tegevuses kiirem kui toode ise. Seetõttu peab teste kirjutama nii, et need pidevalt arvestaks tingimuste realiseerumisega enne järgmise sammu tegemist. Ootehetkede

lisamiseta oleks testide käivitamise aruandes mitmeid vale-negatiivseid tulemusi. Näiteks peab test ootama, et nupp oleks lehel aktiivne enne kui üritab seda nuppu vajutada. Lisaks eelnevale leidis autor, et automatiseerimise lihtsustamiseks ja stabiilsuse tagamiseks oleks vaja luua hulk kõrgema taseme märksõnu, mis oskaks kohanduda süsteemis toimuvaga ja vastavalt selle seisundile õigesti reageerida. Üks selline näide on kliendi valimine lehekülje päises. Kui otsingutulemusi on rohkem kui üks, siis kasutaja peab ise otsingutulemuste nimekirjast valima sobiva kliendi. Samas kui tulemuste arv on üks, siis valitakse see klient automaatselt ja kasutaja ei pea seda eraldi otsingutulemustes kinnitama. Selliste juhtumitega võiks ka osata arvestada testides kasutatav märksõna.

Eelnevale ülevaatele tuginedes võib öelda, et Robot Framework raamistikus on küll lihtne teste koostada, ent kogu lahenduse muretu ülevalhoidmine vajab teadlikku lähenemist ja oma vajadustele vastavaks kohandamist. Töö autor leiab, et Robot Framework raamistik võiks sobida töös käsitletud pangandustarkvara automaattestimiseks siis, kui seda kasutada vaid kõige üldisemate äriloogikate automaattestimisel kasutajaliidese vahendusel. Samas ei täida antud raamistik kogu automaattestimise vajadust, vaid vajab endale ka mõnda muud töövahendit, mis tegeleks automaattestimise püramiidi madalamatel tasemetel ja kus testimise ei toimuks rakenduse kasutajaliidese vahendusel. Töö autor arvab, et selline lahendus, kus Robot Framework raamistikku kasutatakse koos Selenium2Library teegiga, võiks kõige paremini sobida väikeste infosüsteemide ja veebirakenduste automaattestimiseks, mille keerukus ei ole väga kõrge ja mille kasutajaliides muutub harva.

Kokkuvõte

Bakalaureusetöö „Automaattestide koostamine Robot Framework raamistikul pangandustarkvara näitel“ eesmärk oli tutvustada Robot Framework raamistiku võimalusi, proovida järgi raamistiku kasutamist ning hinnata seda pangandustarkvara näitel.

Töö tulemusel valmis ülevaade:

- automaattestimise printsiipidest, vahenditest ja nende valiku põhimõtetest;
- Robot Framework raamistikust, selle lisadest ja nende kasutamisest;
- Robot Framework raamistiku kasutamisest pangandustarkvara automaattestimisel.

Töö autor selgitas pangandustarkvara näitel, kuidas korraldada automaattestimise raamistiku kasutamist, tööd automaattestidega planeerida ning automaatteste koostada. Töö tulemusel hindas töö autor kõnealuse tarkvara automaattestimise ja raamistiku koostöö võimalusi ja kaasnevaid probleeme. Samuti valmis komplekt automaatteste, mida saab edaspidi kasutada pangandustarkvara automaattestimisel.

Töö tulemusel leidis autor, et kuigi automaattestide koostamine Robot Framework raamistikul märksõnadel põhineva metoodika abil on lihtne, siis loodud automaattestidega võib esineda mõningaid tehnilisi probleeme ning töö raamistikuga vajab selget strateegiat. Autori arvates võiks raamistik sobida pangandustarkvara üldiste äriprotsesside automaattestimiseks kasutajaliidese vahendusel, kuid vajab endale kõrvale veel mõnda automaattestimise vahendit täitmaks kõiki automaattestimise vajadusi. Eelkõige soovitab autor Robot Framework raamistikku kasutada väiksemate infosüsteemide ja veebirakenduste automaattestimisel.

Töö autor plaanib valminud automaatteste kasutada pangandustarkvara regressioonitestimisel ning võimalusel jätkata uute testide loomist nii tellerirakendusele kui ka internetipangale.

Summary

The topic of this Bachelor Thesis is „Test Automation with Robot Framework. The Case of Banking Software“. The purpose of this is to give an overview on Robot Framework and explain how to use this framework for test automation of a banking software. Robot Framework is a generic test automation framework with keyword-driven testing approach and it is used for acceptance testing. It can be used for all sorts of operating systems and applications. Due to used methodology, writing tests is simple, does not require good programming skills and can be done by a non-technical person as well.

The first chapter of this Bachelor Thesis gives a general overview on test automation principles. The author explains what is test automation overall and what befits it has. Test automation pyramid, different types of automation frameworks, tool selection procedure and challenges of test automation are introduced.

The second chapter focuses on Robot Framework. The author introduces architecture, libraries and tools used in Robot Framework. There is also an overview how to install Robot Framework and related components, what is the structure of automated tests in Robot Framework, how to write and later execute them.

The third chapter shows how to use Robot Framework for specific software test automation. The idea is to conduct a proof of concept to decide if this framework is suitable tool for test automation on such software. In this chapter, the banking solution is introduced and work is divided in to 3 phases. The first phase explains why this software needs test automation and why is Robot Framework picked for this job. The second phase is about planning keywords and test cases. The third phase is the development and here is explained how earlier plans really come to life. The final part of this chapter gives an overview on experience of using Robot Framework and clarifies how to use this framework in future for test automation.

In conclusion, test automation with Robot Framework is simple, but there can be some technical issues. It suits for automated testing of baking software, when it is used just for general business logic testing via user interface. The author recommends using Robot Framework for test automation of small web applications and information systems.

Kasutatud kirjandus

Annuste, T. (2016). Automaattestimise raamistike analüüs ja võrdlus Telia Eesti AS näitel. Tallinn.

Basu, A. (2015). *Software quality assurance, testing and metrics.* Delhi: Prentice-Hall of India Pvt.Ltd.

Bisht, S. (2013). *Robot Framework Test Automation.* Birmingham, Mumbai: Packt Publishing Ltd.

Camgemini, Sogeti, Hewlett Pacard Enterprise. (2016). *World Quality Report 2016-17.* Camgemini, Sogeti, Hewlett Pacard Enterprise.

Cohn, M. (17. 12. 2009. a.). *The Forgotten Layer of the Test Automation Pyramid*. Allikas: Mountain Goat Software: https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid

Gheorghiu, G. (31. 10. 2016. a.). *PythonTestingToolsTaxonomy*. Allikas: The Python Wiki: https://wiki.python.org/moin/PythonTestingToolsTaxonomy

GitHub, Inc. (17. 01. 2017. a.). *Installation Instructions*. Allikas: GitHub: https://github.com/robotframework/RIDE/wiki/Installation-Instructions#installation

GitHub, Inc. (18. 03. 2017. a.). *Java test automation*. Allikas: GitHub: https://github.com/atinfo/awesome-test-automation/blob/master/java-test-automation.md#web-ui-test-automation

Hendrickson, E. (01. 12. 2011. a.). *Test Obsessed*. Allikas: From the mailbox: selecting test automation tools: http://testobsessed.com/2011/12/selecting-test-automation-tools/

Jenkins. (kuupäev puudub). Allikas: https://jenkins.io/

Lent, J. (28. 02. 2013. a.). *FAQ: Automated software testing basics*. Allikas: TechTarget: http://searchsoftwarequality.techtarget.com/feature/FAQ-Automated-software-testing-basics

Madaan, D. (10. 03. 2015. a.). *Behaviour Driven Testing – An introduction*. Allikas: Women Testers: http://www.womentesters.com/behaviour-driven-testing-an-introduction/

Marshall, A. (12. 02. 2015. a.). *The 8 Useful Java Testing tools (& Frameworks) for Programmers, Developers and Coders*. Allikas: IDRsolutions: https://blog.idrsolutions.com/2015/02/8-useful-java-testing-tools-frameworks-programmers-developers-coders/

Mishra, M. (04. 12. 2014. a.). *Introduction to Robot Framework*. Allikas: Agile Test Tools And Techniques: http://agiletesttoolsandtechniques.blogspot.com.ee/2014/12/introduction-to-robot-framework_3.html

Robot Framework Foundation. (06. 01. 2017. a.). *Robot Framework User Guide*, Version 3.0.1. Allikas: Robot Framework: http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

Robot Framework Foundation. (kuupäev puudub). *Robot Framework*. Allikas: Robot Framework: http://robotframework.org/

Robot Framework Foundation. (kuupäev puudub). *Robot Framework documentation*. Allikas: Robot Framework: http://robotframework.org/robotframework/

Robot Framework Quick Start Guide. (19. 01. 2017. a.). Allikas: GitHub: https://github.com/robotframework/QuickStartGuide/blob/master/QuickStart.rst

Selenium Project. (23. 02. 2017. a.). *Selenium WebDriver*. Allikas: SeleniumHQ Browser Automation: http://www.seleniumhq.org/docs/03_webdriver.jsp#

Software Testing Help. (14. 12. 2016. a.). *Most Popular Test Automation Frameworks with Pros and Cons of Each – Selenium Tutorial #20*. Allikas: Software Testing Help: http://www.softwaretestinghelp.com/test-automation-frameworks-selenium-tutorial-20/

Software Testing Help. (25. 08. 2016. a.). *Why Do We Need Framework for Test Automation?* Allikas: Software Testing Help: http://www.softwaretestinghelp.com/why-do-we-need-test-automation-framework/

The Apache Software Foundation. (kuupäev puudub). *Apache project list*. Allikas: The Apache Software Foundation: https://www.apache.org/index.html#projects-list

Toledo, D. F. (2016). *A Complete Introduction to Functional Test Automation.* Abcstracta.

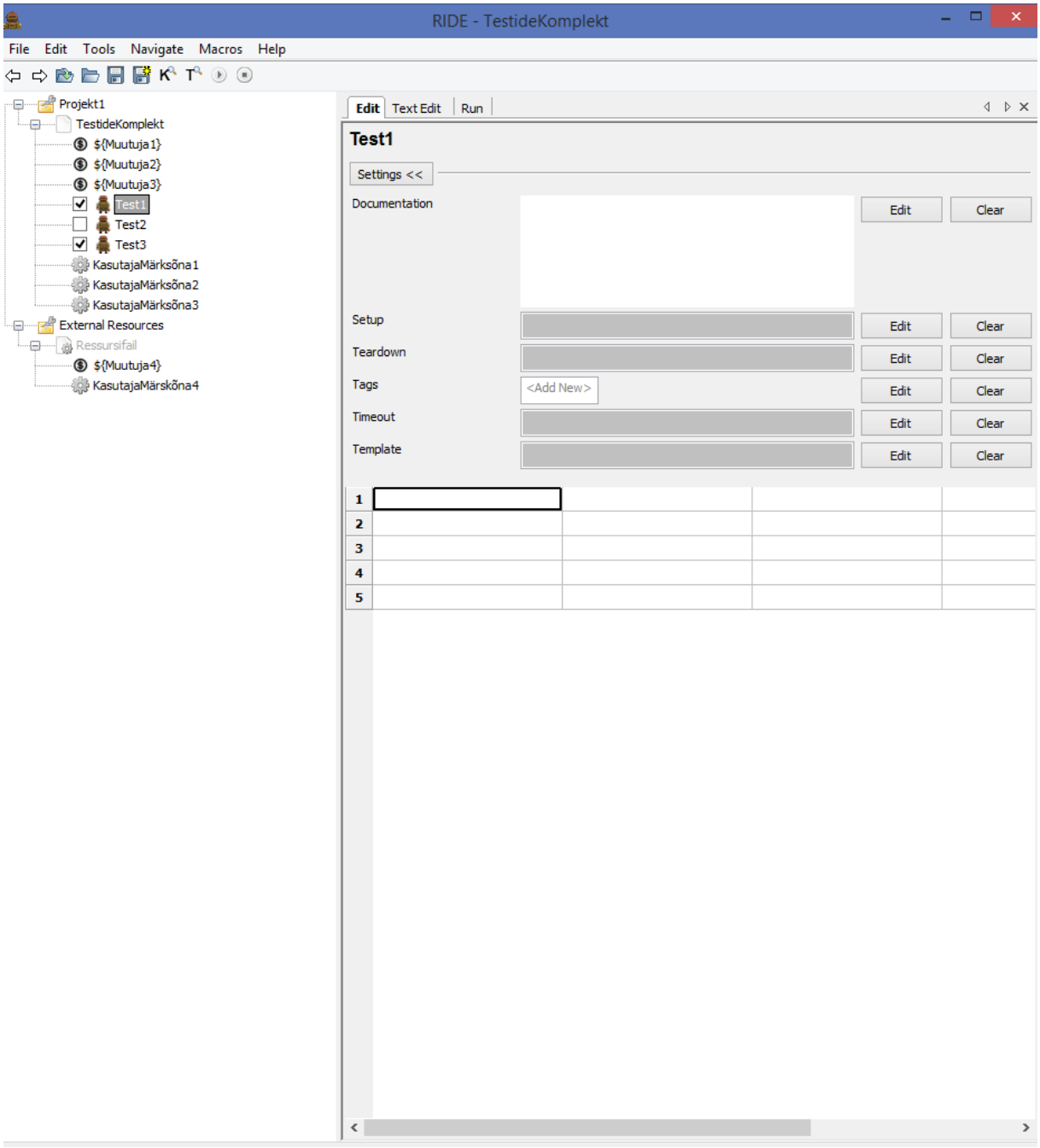
Türk, A. (2015). Võtmesõnadel põhineva testimise metoodika ning raamistikud tarkvara automaattestimises. Tartu.

Wikipedia. (kuupäev puudub). Allikas: List of unit testing frameworks: https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

Worksoft Inc. (2013). *2013 Trends in Automated Testing.* Worksoft Inc. Allikas: https://www.worksoft.com/2013-trends-in-automated-testing-for-enterprise-systems

Lisad

Lisa 1. RIDE kasutajaliides



Lisa 2. Testimise logifail Robot Framework raamistikus

Pangandustarkvara Test Log

REPORT
20170425 08:52:09 GMT+03:00
1 minute 3 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	4	2	2	00:00:11	
All Tests	4	2	2	00:00:11	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Pangandustarkvara	4	2	2	00:00:19	
Pangandustarkvara.Automaattestid	4	2	2	00:00:19	

Test Execution Log

SUITE Pangandustarkvara 00:00:19.081

Full Name: Pangandustarkvara
Source:
Start / End / Elapsed: 20170425 08:51:50.146 / 20170425 08:52:09.227 / 00:00:19.081
Status: 4 critical test, 2 passed, 2 failed
4 test total, 2 passed, 2 failed

SUITE Automaattestid 00:00:19.042

Full Name: Pangandustarkvara.Automaattestid
Documentation: See on automaattestide komplekt. Testide komplekti tasandil on seadistatud:

- Selenium2Library teegi importimine
- Veebibrauseri avamine ja testkeskkonna avalehele navigeerimine enne komplektis olevate testide sooritamist
- Veebibrauseri sulgemie pärast komplektis olevate testide sooritamist

Source:
Start / End / Elapsed: 20170425 08:51:50.177 / 20170425 08:52:09.219 / 00:00:19.042
Status: 4 critical test, 2 passed, 2 failed
4 test total, 2 passed, 2 failed

SETUP Selenium2Library.Open Browser ${url}, ${browser} 00:00:07.886

TEST T1: Login failed 00:00:01.653

TEST T2: Successful login 00:00:02.750

Full Name: Pangandustarkvara.Automaattestid.T2: Successful login
Documentation: Õnnestunud sisselogimine
Start / End / Elapsed: 20170425 08:51:59.874 / 20170425 08:52:02.624 / 00:00:02.750
Status: PASS (critical)

KEYWORD Successful log in to system ${validusername}, ${validpassword} 00:00:02.749

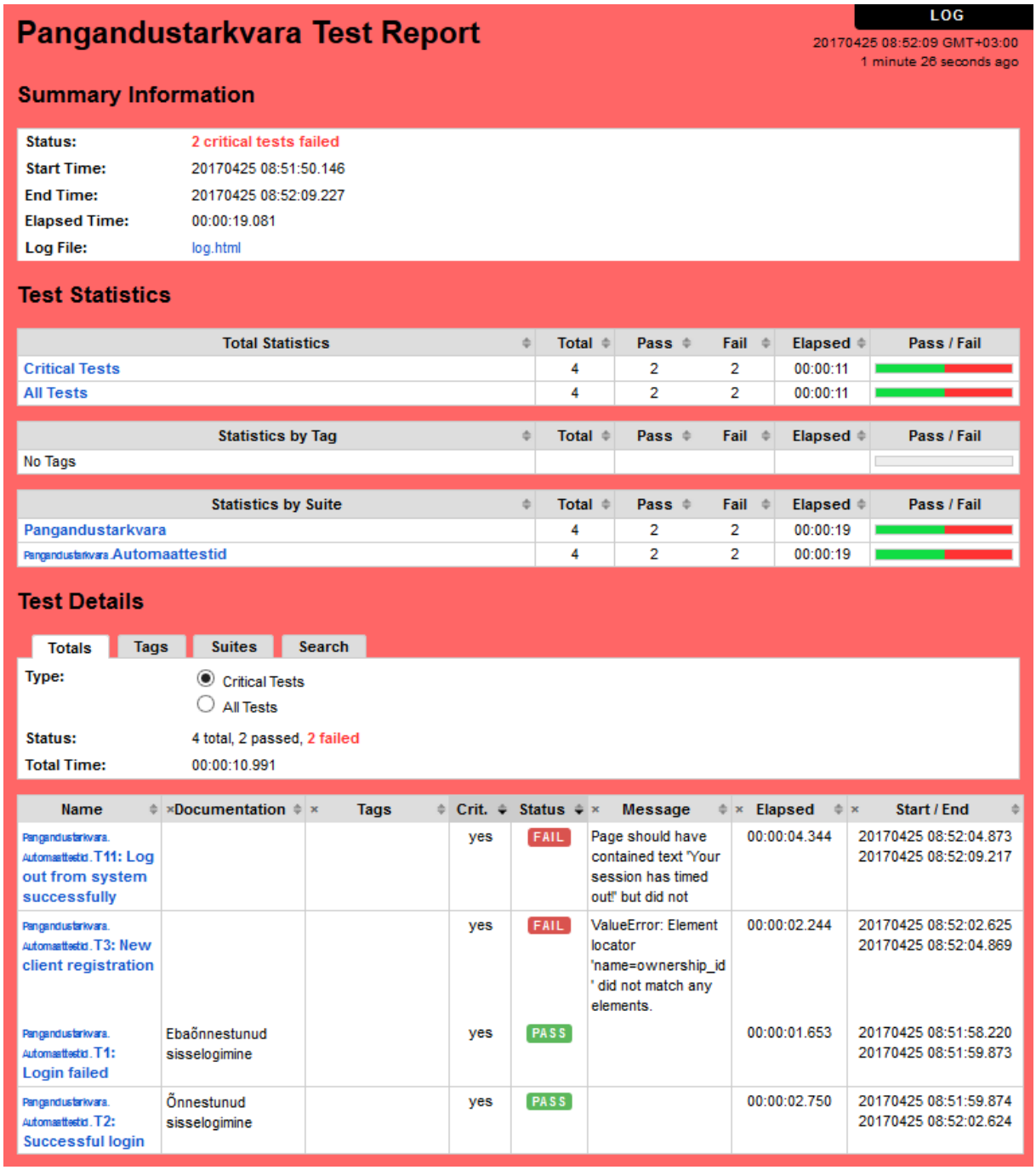
Start / End / Elapsed: 20170425 08:51:59.875 / 20170425 08:52:02.624 / 00:00:02.749

KEYWORD Selenium2Library.Input Text id=user-name, ${username} 00:00:00.098
KEYWORD Selenium2Library.Input Text id=user-pw, ${password} 00:00:00.117
KEYWORD Selenium2Library.Click Button name=nupp 00:00:02.438
KEYWORD Selenium2Library.Page Should Not Contain ${unsuccessful_login_message} 00:00:00.094

TEST T3: New client registration 00:00:02.244

TEST T11: Log out from system successfully 00:00:04.344

Lisa 3. Testimise raport Robot Framework raamistikus



Pangandustarkvara Test Report

LOG
20170425 08:52:09 GMT+03:00
1 minute 26 seconds ago

Summary Information

Status:	2 critical tests failed
Start Time:	20170425 08:51:50.146
End Time:	20170425 08:52:09.227
Elapsed Time:	00:00:19.081
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	4	2	2	00:00:11	
All Tests	4	2	2	00:00:11	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Pangandustarkvara	4	2	2	00:00:19	
Pangandustarkvara.Automaattestid	4	2	2	00:00:19	

Test Details

Totals | Tags | Suites | Search

Type: (•) Critical Tests ( ) All Tests

Status: 4 total, 2 passed, 2 failed

Total Time: 00:00:10.991

Name	Documentation	Tags	Crit.	Status	Message	Elapsed	Start / End
Pangandustarkvara.Automaattestid.T11: Log out from system successfully			yes	FAIL	Page should have contained text 'Your session has timed out!' but did not	00:00:04.344	20170425 08:52:04.873 20170425 08:52:09.217
Pangandustarkvara.Automaattestid.T3: New client registration			yes	FAIL	ValueError: Element locator 'name=ownership_id ' did not match any elements.	00:00:02.244	20170425 08:52:02.625 20170425 08:52:04.869
Pangandustarkvara.Automaattestid.T1: Login failed	Ebaõnnestunud sisselogimine		yes	PASS		00:00:01.653	20170425 08:51:58.220 20170425 08:51:59.873
Pangandustarkvara.Automaattestid.T2: Successful login	Õnnestunud sisselogimine		yes	PASS		00:00:02.750	20170425 08:51:59.874 20170425 08:52:02.624

Lisa 4. Automaattestid

```
# Seaded
*** Settings ***
Documentation     See on automaattestide komplekt.
...               Testide komplekti tasandil on seadistatud:
...               - Selenium2Library teegi importimine
...                - Veebibrauseri avamine ja testkeskkonna avalehele
navigeerimine enne komplektis olevate testide sooritamist
...               - Veebibrauseri sulgemine pärast komplektis olevate
testide sooritamist
Suite Setup       Open Browser    ${URL}    ${BROWSER}
Suite Teardown    Close All Browsers
Library           Selenium2Library

#Testides ja seadetes kasutatavate muutujate defineerimine
*** Variables ***
${BROWSER}        gc
${URL}            localhost
${VALID_USERNAME}    dea
${VALID_PASSWORD}    dea
${UNSUCCESFUL_LOGIN_MESSAGE}    Wrong username/password
${TIMEOUT_MESSAGE}    Your session has timed out!
${LOGOUT_MESSAGE}    You have logged out.
${SAVE_MESSAGE_NEW_CLIENT_REG}    New customer added to registry.
${SAVE_MESSAGE_UPDATE_CLIENT_REG}    Customer data updated!
${SAVE_MESSAGE_NEW_ACCOUNT}    Account opened successfully!
${SAVE_MESSAGE_CASH_DEPOSIT}      Order is saved and needs to be
confirmed!
${CONFIRM_MESSAGE_CASH_DEPOSIT}    Order is confirmed!
${SAVE_MESSAGE_PAYMENT_ORDER}      Order is saved and needs to be
confirmed!
${CONFIRM_MESSAGE_PAYMENT_ORDER}    Order is confirmed!
${SAVE_MESSAGE_CASH_WITHDRAWAL}     Order is saved and needs to be
confirmed!
${CONFIRM_MESSAGE_CASH_WITHDRAWAL}    Order is confirmed!
${CLOSE_MESSAGE_ACCOUNT}    Account is closed!

#Testlood
*** Test Cases ***
T1: Login failed
    [Documentation]    Ebaõnnestunud sisselogimine
    [Template]    Failed log in to system
    valekasutajanimi    valeparool
    valeparool    valekasutajanimi

T2: Successful login
    [Documentation]    Õnnestunud sisselogimine
    [Template]    Successful log in to system
    ${VALID_USERNAME}    ${VALID_PASSWORD}

T3: New client registration
    [Documentation]    Uue kliendi registreerimine
```

```
    Navigate to page    Customer    Registry    REGISTRY
    New record    REGISTRY
    Fill new user form (private person)     Private individual (10)
37810019940    Oskar    Ohakas    5550100203    01.10.1978
    Save record    ${SAVE_MESSAGE_NEW_CLIENT_REG}

T4: Upate client data
    [Documentation]    Kliendi andmete muutmine
    Navigate to page    Customer    Search    CLIENT SEARCH
    Input Text    name=cust_name    Oskar Ohakas
    Show records
    Select From List By Index    name=search_result    1
    View/change record    REGISTRY
    Input Text    name=phone_number    55020304
    Save record    ${SAVE_MESSAGE_UPDATE_CLIENT_REG}
    Navigate back to previous page    CLIENT SEARCH

T5: Open bank account
    [Documentation]    Pangakonto avamine
    Navigate to page    Accounts   Current accounts    CURRENT ACCOUNTS
    Select customer on top of screen    Oskar Ohakas
    New record    CURRENT ACCOUNT OPENING
    Select From List By Value    name=acc_type    Bank account
    Save record    ${SAVE_MESSAGE_NEW_ACCOUNT}
    Navigate back to previous page    CURRENT ACCOUNTS

T6: Cash deposit to client's account
    [Documentation]    Sularaha sissemakse kliendi kontole
    Navigate to page    Payments    Cash deposit    CASH DEPOSITS
    Select customer on top of screen    Oskar Ohakas
    New record    CASH DEPOSIT
    Fill in cash deposit form    1000.00    Sularaha sissemakse Oskar
Ohakas kontole
    Save record    ${SAVE_MESSAGE_CASH_DEPOSIT}
    Confirm record    ${CONFIRM_MESSAGE_CASH_DEPOSIT}
    Navigate back to previous page    CASH DEPOSITS

T7: Payment from client's account
    [Documentation]    Pangaülekanne kliendi kontolt
    Navigate to page    Payments    Payment orders    PAYMENT ORDERS
    Select customer on top of screen    Oskar Ohakas
    New record    PAYMENT ORDER
    Fill in payment order form    Maali Maasikas    EE22220001107773747
45.00    Ülekanne Maalile
    Save record    ${SAVE_MESSAGE_PAYMENT_ORDER}
    Confirm record    ${CONFIRM_MESSAGE_PAYMENT_ORDER}
    Navigate back to previous page    PAYMENT ORDERS

T8: Cash withdrawl from client's account
    [Documentation]    Sularaha väljamakse kliendi kontolt
    Navigate to page    Payments    Cash withdrawal    CASH WITHDRAWALS
    Select customer on top of screen    Oskar Ohakas
    New record    CASH WITHDRAWAL
```

```
    Fill in cash withdrawal form    65.00
    Save record    ${SAVE_MESSAGE_CASH_WITHDRAWAL}
    Confirm record    ${CONFIRM_MESSAGE_CASH_WITHDRAWAL}
    Navigate back to previous page    CASH WITHDRAWALS

T9: Close bank account
    [Documentation]    Pangakonto sulgemine
    Navigate to page    Accounts    Current accounts    CURRENT
ACCOUNTS
    Select customer on top of screen    Oskar Ohakas
    Select From List By Index    name=search_result    1
    Click Button    name=close_btn
    Wait Until Element Is Visible    name=confirm_dialog_message
    Click Button    name=confirm_close
    Wait Until Element Is Not Visible    name=confirm_dialog_message
    Page Should Contain    ${CLOSE_MESSAGE_ACCOUNT}

T10: Log out from system successfully
    Log out
    Go Back
    BuiltIn.Sleep    3s
    Page Should Contain    ${TIMEOUT_MESSAGE}

#Testides kasutatavate märksõnade loomine
*** Keywords ***
Failed log in to system
    [Arguments]    ${username}    ${password}
    [Documentation]    Ebaõnnestunud sisselogimine
    Input Text    id=user-name    ${username}
    Input Text    id=user-pw    ${password}
    Click Button    name=login_btn
    Page Should Contain    ${UNSUCCESFUL_LOGIN_MESSAGE}

Successful log in to system
# Süsteemi sisse logimiseks on vaja kasutajanime ja parooli
    [Arguments]    ${username}    ${password}
    [Documentation]    Õnnestunud sisselogimine
    Input Text    id=user-name    ${username}
    Input Text    id=user-pw    ${password}
    Click Button    name=nupp
    Page Should Not Contain    ${UNSUCCESFUL_LOGIN_MESSAGE}
    Page Should Contain    ${username}

Navigate to page
# Lehele navigeerimiseks vajutatakse peamenüü linki, alammenüü linki
ja kontrollitakse, kas süsteem jõudis õigele leheküljele.
    [Arguments]    ${mainmenu}    ${submenu}    ${page_title}
    [Documentation]    Leheküljele navigeerimine
    Select Frame    name=menu
    Click Link    link =${mainmenu}
    Wait Until Element Is Visible    link=${submenu}
    Click Link    link=${submenu}
    Page Should Contain    ${page_title}
```

```
    Select Frame    name=WORK

Show records
    [Documentation]    Kirjete loetelu kuvamine
    Wait Until Element Is Enabled    name=btn_search
    Click Button    name=btn_search
    Page Should Contain Radio Button    name=record_id

View/change record
# Detailandmete vormi avamisel kontrollitakse, kas süsteem on jõudnud
korrektsele leheküljele
    [Arguments]    ${edit_record_page}
    [Documentation]    Kirje detailandmete vaatamine
    Wait Until Element Is Enabled    name=btn_edit
    Click Button    name=btn_edit
    Page Should Contain    ${edit_record_page}

New record
# Uue kirje sisestamise vormi avamisel kontrollitakse, kas süsteem on
jõudnud korrektsele leheküljele
    [Arguments]    ${new_record_page}
    [Documentation]    Uue kirje lisamine
    Wait Until Element Is Enabled    name=btn_new
    Click Button    name=btn_new
    Page Should Contain    ${new_record_page}

Save record
#  Kirje  salvestamisel  kontrollitakse,  kas  leheküljel  kuvatakse
salvestamise teadet
    [Arguments]    ${save_message}
    [Documentation]    Kirje salvestamine
    Wait Until Element Is Enabled    name=btn_save
    Click Button    name=btn_save
    Page Should Contain    ${save_message}

Confirm record
#  Kirje  kinnitamisel  kontrollitakse,  kas  leheküljel  kuvatakse
kinnitamise teadet
    [Arguments]    ${confirm_message}
    [Documentation]    Kirje kinnitamine
    Wait Until Element Is Enabled    name=btn_confirm
    Click Button    name=btn_confirm
    Page Should Contain    ${confirm_message}

Select customer on top of screen
# Kliendi valimiseks on vaja sisestada kliendi nimi
    [Arguments]    ${customername}
    [Documentation]    Kliendi valimine lehekülje päises
    Wait Until Element Is Visible    name=custfind
    Click Button    name=custfind
    Select Window    CUSTOMER SEARCH
    Input Text    name=company_name    ${customername}
    Click Button    name=otsi
```

```
    Wait Until Element Is Visible    name=foundcust
    Select From List By Index    name=foundcust    1
    Click Button    name=valik
    Select Window
    Select Frame    name=WORK

Navigate back to previous page
# Detailandmete vormilt lahkumisel „Tagasi" nupu abil kontrollitakse, kas süsteem jõudis oodatud lehele
    [Arguments]    ${back_page_title}
    [Documentation]    Detailandmete vormilt lahkumine
    Click Button    name=btn_back
    Page Should Contain    ${back_page_title}

Log out
# Märksõnas on defineeritud välja logimise nupu vajutamine ning väljalogimise teate otsimine leheküljelt
    [Documentation]    Süsteemist väljalogimine
    Select Frame    name=WORK
    Click Image    /img/logout.png
    Page Should Contain    ${LOGOUT_MESSAGE}

Fill new user form (private person)
# Uue eraisiku registreerimiseks on vaja valida klienditüüp ning lisada kliendi isikukood, eesnimi, perenimi, telefoninumber ja sünniaeg
    [Arguments]    ${clienttype}    ${idcode}    ${firstname}    ${lastname}    ${phone}    ${regdate}
    [Documentation]    Kliendi andmete sisestamine kliendi registreerimise vormil (eraisik)
    Select From List    name=ownership_id    ${clienttype}
    Input Text    name=registration_number    ${idcode}
    Input Text    name=firstname    ${firstname}
    Input Text    name=lastname    ${lastname}
    Input Text    name=phone_number    ${phone}
    Input Text    name=registration_date    ${regdate}

Fill in cash deposit form
# Sularaga sissemaksel sisestatakse vormil makse summa ja selgitus
    [Arguments]    ${amount}    ${payment_info}
    [Documentation]    Sularaha sissemakse vormil andmete sisestamine
    Input Text    name=amount    ${amount}
    Input Text    name=description    ${payment_info}
    Click Button    name=fee_calc
    Wait Until Page Contains    Fee calculated!

Fill in payment order form
# Maksekorralduse koostamisel sisetatakse makse saaja nimi, pangakonto, summa ja selgitus
    [Arguments]    ${ben_name}    ${ben_account}    ${amount}    ${explanation}
    [Documentation]    Maksekorralduse vormil andmete sisestamine
    Input Text    name=beneficiary_name    ${ben_name}
```

```
    Input Text    name=ben_acc_number    ${ben_account}
    Input Text    name=amount    ${amount}
    Input Text    name=explanation    ${explanation}
    Click Button    name=fee_calc
    Wait Until Page Contains    Fee calculated!

Fill in cash withdrawal form
# Sularaha väljamaksel sisestakase vormil väljamakse summa
    [Arguments]    ${amount}
    [Documentation]    Sularaha väljamakse vormil andmete sisestamine
    Input Text    name=amount    ${amount}
    Select From List By Index    name=account_select    1
    Click Button    name=fee_calc
    Wait Until Page Contains    Fee calculated!
```