TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Infotehnoloogia eriala

Ardi Türk

# Võtmesõnadel põhineva testimise metoodika ning raamistikud tarkvara automaattestimises

Bakalaureusetöö (6 EAP)

Juhendaja: dotsent Helle Hein

TARTU 2015

# Võtmesõnadel põhineva testimise metoodika ning raamistikud tarkvara automaattestimises

Käesoleva töö peamiseks eesmärgiks on anda ülevaade ja analüüs võtmesõnadel põhineva testimise metoodikast ja võimalustest ning ohtudest selle kasutusele võtmisel. Samuti tutvustatakse mitmeid metoodikat toetavaid raamistikke, neist detailsemalt kahte populaarsemat – FitNesse ja Robot Framework. Töös kasutatakse praktilisi näiteid ning autori kogemusi reaalsetest Eesti tarkvaraprojektidest, kus kirjeldatud metoodikat kasutati. Praktiliste soovituste kõrval tuuakse ära ka põhjused, miks paljud tarkvaraprojektid on ebaõnnestunud metoodika kasutuselevõtul, et lugeja oskaks nendest hoiduda. Töös kirjeldatakse ka tarkvara testimist ning automaattestimist üldiselt, kuna tutvustatav metoodika on nendega väga tihedalt seotud.

**Võtmesõnad:** võtmesõnadel põhinev testimine, automaattestimine, tarkvara testimine, Robot Framework, FitNesse

# Keyword driven testing methodology and frameworks in automated testing

The main goal of the thesis is to give an overview of keyword driven testing methodology, its possibilities and dangers. Multiple frameworks that use the methodology are introduced, two of most popular of them are introduced in more detail – FitNesse and Robot Framework. Examples and author's experience are used from real Estonian software projects, where the methodology was implemented. Along with practical recommendations the reasons why many software projects have failed in implementing this methodology are listed. It has been done so that the reader can avoid them as much as possible. Also an overview of software testing and automated testing is given, because the keyword driven testing methodology is closely connected to them.

**Keywords:** keyword driven testing, automated testing, software testing, Robot Framework, FitNesse

# Sisukord

# Sissejuhatus

Iga aastaga kasvab vajadus uute tarkvarasüsteemide järele ning läbi selle ka vajadus tarkvara arenduse järele (näiteks Eesti info- ja sidesektori lisandväärtuse reaalkasv 2014. aastal oli 14,9%). Samas kasvab ka arendatavates tarkvarades kasutatavate tehnoloogiate ning äriloogika keerukus tulenevalt soovist pakkuda tarkvaraga aina rohkem võimalusi ning digitaliseerida suuremat hulka tegevusi. Vaatamata mahu ja keerukuse jätkuvale kasvule, soovitakse tarkvara kvaliteeti hoida sama heal tasemel või isegi tõsta. Seetõttu tuleb tarkvara arendavatel firmadel pidevalt leida uusi meetodeid kvaliteedi hoidmiseks ja tõstmiseks olukorras, kus nõuded tarkvarale kasvavad aina kiiremini, kuid inimressursis sama suurt kasvu ei toimu. Võttes arvesse sellist valdkonna keerukuse ning tööjõu kasvu vahekorda, võiks võtmesõnadel põhineva testimise metoodika olla hea vahend kvaliteedi kiireks tõstmiseks, kasutades ära firmades juba olemasolevat ressurssi.

Tarkvara testimises on kaks suurt probleemi, mida käesoleva tööga soovitakse lahendada:

1) Vajadus pidevalt hoida või tõsta kvaliteeti, kasutades ära juba olemasolevat tööjõudu ning ajalisi piiranguid. Firmade tööjõust tegeleb automaattestimisega väike osakaal testijatest, mis tuleneb peamiselt selle testimise viisi kõrgest tehniliste teadmiste nõudest. Samas võimaldab automaattestimine suure osa oma tööst kerge vaevaga kiiresti korratavaks teha ning läbi selle saab sama ajaga üks testija rohkem teste läbi viia. Töös tutvustatav metoodika ning raamistikud aitavad väheste tehniliste teadmistega inimestel automaatteste kirjutada. Firmadel aitab see vajadusel ära kasutada kogu oma testimisressurssi automaattestide kirjutamiseks ja läbi selle tõsta rakenduse kvaliteeti ning testimise kiirust.
2) Selgete metoodikate puudumine testimisprotsessis. Kuigi paljud tarkvaraprojektid üritavad võimalikult suurel määral oma testimist automatiseerida, puudub paljudel teadmine, kuidas seda teha, kasutades võimalikult vähe raha, aega ning muud ressurssi. Seetõttu oleks vaja head emakeelset õppematerjali kasulikest metoodikatest, millega saavad projektiliikmed tutvuda.

**Lõputöö koostamisel on neli suuremat eesmärki:**

1. Anda ülevaade võtmesõnadel põhineva testimise metoodikast ja raamistikest, mis on hetkel testimise valdkonnas vähe tuntud ning mille kohta leiab vähe nii eesti- kui ka muukeelset kirjandust.
2. Anda selgeid juhtnööre tutvustatud metoodika kasutuselevõtuks ning sobiliku raamistiku valimiseks.
3. Ülevaates kasutada võimalikult palju näiteid reaalsetest tarkvaraprojektidest ning autori kogemusi metoodika kasutamisest testimise valdkonnas, et potentsiaalne metoodika kasutaja saaks ülevaate metoodikast just läbi Eestile omaste tarkvaraprojektide teostamise kogemuse.
4. Metoodikast ülevaatliku materjali loomise abil anda võimalus Eesti tarkvarafirmade töötajatele tutvuda metoodikaga ning läbi selle tõsta Eesti tarkvarafirmades automaattestimise teadlikku ning metoodilist kasutamist. Läbi metoodika tutvustamise ning sellest tuleneva juurutamise aidata firmadel paremini ära kasutada oma testimise tööjõudu.

Töö esimeses ja teises peatükis antakse üldine ülevaade tarkvara testimisest ning automaattestimisest, kuna nii testimist kui ka automaattestimist defineeritakse eri kontekstides erinevalt. Seetõttu on vajalik luua ühtne arusaam, kuidas need protsessid on defineeritud meetodi ning raamistike kirjeldamisel.

Töö kolmandas ja neljandas peatükis kirjeldatakse võtmesõnadel põhineva testimise metoodikat ning selle raamistikke. Kahes järgnevas peatükis tutvustatakse kahte kõige populaarsemat raamistikku detailsemalt. Viimases peatükis käsitletakse peamisi metoodika kasutuselevõtu ebaõnnestumise põhjuseid.

Autor tänab juhendaja Helle Heina sisukate märkuste ja igakülgse abi eest.

# 1. Tarkvara testimine

## 1.1 Tarkvara testimise definitsioon ning eesmärgid

Tarkvara testimist teostatakse ning defineeritakse eri firmades ning projektides erinevalt. Seega kõigepealt tuleks defineerida selle vajalikkus ning eesmärgid antud töö raames.

Tarkvara testimine on tarkvara kohta informatsiooni otsimine, et anda hinnang selle tarkvara kvaliteedile. Kindlasti ei saa testimine tagada tarkvara täielikku kvaliteeti, kuna erinevaid tarkvara kasutusvõimalusi on selleks liiga palju. Tarkvara testimine peaks suutma kindlustada, et kõik fikseeritud olulisemad rakenduse kasutusviisid oleksid teostatavad. Peamiselt aitab testimine üles leida võimalikult palju tooteriske ning abistab nende maandamisel, kuid samas ei suuda kindlustada, et kõik võimalikud riskid on kaotatud [1].

**Tarkvara testimise peamisteks eesmärkideks on:**

1) leida vigu arendatavast tarkvarast;
2) kontrollida, et leitud vead saaksid parandatud;
3) anda hinnang tarkvara üldisele kvaliteedile;
4) hinnata, kas arendatav tarkvara on see, mida klient ning lõppkasutaja soovivad ja ka vajavad.

Tarkvara testimist teostavad eri firmades ning projektides erinevad inimesed. Suurematel firmadel on selle jaoks eraldi töötajad, kuid mingil määral teostavad testimist kõik projektiga seotud inimesed (näiteks nii tarkvara testija, programmeerija, analüütik, projektijuht kui ka lõppkasutaja). Kõige suuremal määral ning läbimõeldumalt tegeleb testimisega tarkvara testija, kellele järgneb programmeerija, kes teeb tavaliselt suurel määral automaatselt käivitatavaid ühikteste oma töö esialgseks, kuid ka edasiseks pidevaks kontrollimiseks.

## 1.2 Tarkvara testimise vajalikkus

Testimine on lahutamatu osa tarkvara arendusest. Seda saab teostada igas arendusprotsessi sammus ning mida varasemas protsessis seda rakendatakse ja vigu leitakse, seda odavam on neid parandada.

*Tabel 1. Vea parandamise kulu sõltuvalt vea tekkimise faasist [2]*

| | | Vea leidmise faas | | | | |
|---|---|---|---|---|---|---|
| | | Analüüs | Arhitektuuri kavandamine | Arendamine | Testimine | Töökesk-kond |
| Vea tekkimise faas | Analüüs | 1x | 3x | 5-10x | 10x | 10-100x |
| | Arhitektuuri kavandamine | - | 1x | 10x | 15x | 25-100x |
| | Arendamine | - | - | 1x | 10x | 10-25x |

Tabelis 1 on esitatud uurimuse [2] tulemus, mis näitab, kuidas vea parandamise kulu kasvab olenevalt millisest arenduse faasist see leitakse. Ridadena on esitatud arendusfaas, kus viga tehti ning veergudena faas, millest viga leiti. Siit võib näha, et näiteks analüüsis tehtud vea leidmine testimise faasis läheb 10 korda kallimaks, kui sama vea leidmine kohe analüüsis, kuid kuni 10 korda odavamaks kui selle leidmine töökeskkonnas.

Antud tabel visualiseerib ilmekalt firmade suurt vajadust testimise järgi - kuigi testimine on kulukas, siis mittetestimine on tihtipeale veel kulukam. Muidugi ei kehti samad numbrid ühtselt igas projektis ning firmas, kuid see annab siiski hea arusaamise üldisest reeglist, et mida aeg edasi, seda kallimaks viga läheb. Siinkohal tuleks kindlasti mõista, et hilisemas arendusfaasis vigade parandamine võtab ka rohkem aega ning see aeg tuleb alati mingi teise protsessi arvelt. Peamine on testimise juures meeles pidada, et suur osa tema kasulikkusest väljendub pikemalt tulevikku vaadates.

Kuigi erinevates projektides on tarkvara, riistvara ning meeskonna ülesehitus väga erinev, võib kõik võimalikud tagajärjed ebapiisaval tarkvara testimisel välja tuua nelja suurema jaotusena [3]:

1) **Vigastused või surm** – see võib kõlada dramaatiliselt, kuid paljude kõrget ohutuse taset nõudvate süsteemide kasutamine võib lõppeda vigastuste või surmaga, kui nad ei tööta korralikult (näiteks lennuliikluse korralduse tarkvara).

2) **Raha kaotus** – siia alla kuulub nii klientide kaotus kui ka trahvid erinevate õigusaktide rikkumise eest.

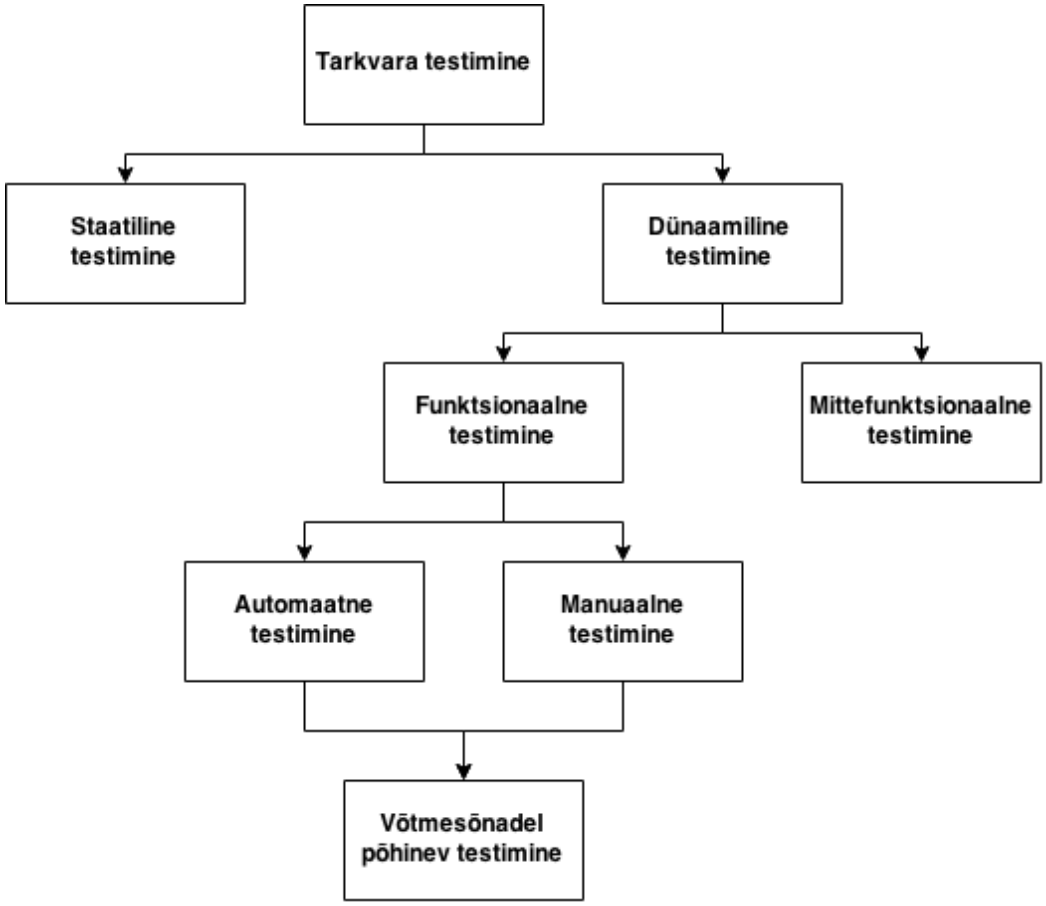
3) **Aja kaotus** – see võib juhtuda näiteks siis, kui erinevad protsessid tarkvaras võtavad kaua aega või toimub täielik tööseisak mõne rakenduse vea või tõrke tõttu.

4) **Kahju firma reputatsioonile** – kui organisatsioon ei ole võimeline oma klientidele teenust pakkuma tarkvara probleemi tõttu, siis kliendid kaotavad kindluse või usalduse selle organisatsiooni vastu (ning tõenäoliselt liiguvad edasi konkurentide klientideks).

Igas projektis on nimetatud tagajärgede esinemise tõenäosus erinev ning see ei ole kunagi kahe projekti puhul täpselt sama. Seetõttu tuleb testijatel iga uue projekti puhul testimise ohud ning võimalikud tagajärjed välja selgitada ning sellest tulenevalt teha kindlaks testimise vajalikkus antud kontekstis. Tuleb meeles pidada, et testimine ei ole kõigi vigade üles otsimine, vaid üks peamisteks testimise eesmärkidest on pakkuda kliendile seda, mida ta vajab ning vahel tuleb selleks testimise arvelt rõhku mujale panna.

## 1.3 Tarkvara testimise tasemed ning tüübid

Tarkvara testimisel on mitmeid eri tasemeid ning tüüpe, kuid käesoleva teema raames vaadatakse jaotust vastavalt Joonisele 1. Joonisel on näha, millisesse testimise harusse töö teema kuulub ning milliste testimise tüüpidega see seotud on.




*Joonis 1. Tarkvara testimise tüüpide tasemed*

Tarkvara testimise võib üldiselt jagada dünaamiliseks ja staatiliseks testimiseks. Staatiline testimine on testimine ilma rakendust käivitamata; enamasti toimub see erinevate ülevaatuste kaudu. Siia kategooriasse kuulub näiteks koodi ning tarkvara spetsifikatsioonide ülevaatus. Töös tutvustatav metoodika selle tüübi ega tema alamtüüpidega ei tegele.

Dünaamilise testimise jaoks käivitatakse rakendus või mõni selle komponent. Näiteks rakendusse sisse logimise dünaamiline testimine on kasutaja andmete sisestamine ning sisselogimise nupu vajutamine. Dünaamiline testimine omakorda jaguneb funktsionaal- ja mittefunktsionaalseks testimiseks. Mittefunktsionaalse testimise alla kuulub näiteks kasutajasõbralikkus, koormustaluvus ja turvalisus [4].

Funktsionaalne testimine on nõuetele vastavuse kontrollimine. Näiteks, kui rakenduse spetsifikatsioonis on öeldud, et rakenduses peab olema võimalik luua uusi taotlusi, siis funktsionaalne test on uue taotluse loomise testimine läbi taotluse väljade täitmise ning taotluse esitamise [4].

Funktsionaalne testimine jaguneb automaatseks ja manuaalseks testimiseks ning mõlema jaotuse korral on võimalik kasutada võtmesõnadel põhinevat testimist, kuid kõige suurem kasu tuleb metoodikast just automaattestimise juures. Manuaalse testimise juures on võimalik testjuhtumeid kirja panna ka muudel mugavamatel ning kiirematel viisidel.

Ülaltoodud joonis annab ülevaate, millisesse tarkvara testimise harusse tutvustatav metoodika kuulub ning milliseid suuremaid harusid see ei kata.

# 2. Tarkvara automaattestimine

## 2.1 Tarkvara automaattestimise definitsioon ning eesmärk

Töös tutvustatavat metoodikat vaadatakse automaattestimise kontekstis ning seetõttu tuleb lahti seletada ka tarkvara automaattestimise mõiste, selle tähtsus, head ja halvad küljed ning ohud. Kõiki käesoleva peatüki juures mainitud põhimõtteid tuleb tutvustatava metoodika rakendamisel pidevalt jälgida, kuna metoodikaga koostatakse funktsionaalsed automaattestid ning seda tüüpi testidele rakenduvad omadused ja ohud kehtivad ka võtmesõnadel põhineva testimise automatiseerimise kohta. Seega antud peatüki mõistmine ja soovituste järgimine mõjutab otseselt ka metoodika kasutuselevõtu edukust.

Automaattestimist võib defineerida kui inimese poolt välja mõeldud ning disainitud testlugude automaatset käivitamist. Täpsemalt öeldes - valitud raamistikus ja programmeerimiskeeles testjuhtumite kirjutamine, mis võimaldab soovitud formaadis ja tihedusega neid automaatselt käivitada.

Automaattestimine on tegevus, millel on selged eelised, nagu näiteks testide kiire käivitatavus, järjepidevus ja võimalus neid  korrata erinevatel aegadel ilma suureneva kuluta. Samas see ei ole triviaalne tegevus ja nõuab tõhusat planeerimist ja viimistletud testjuhtumite koostamist [5].

Enamuses tarkvaraprojektides on hulgaliselt teste, mida tuleb läbida pidevalt ja hoolikalt, et tagada tarkvara stabiilselt kõrge kvaliteet. Suure hulga selliste testide läbimine on tihti väga rutiinne ning kurnav töö, kuid samas tuleb alati hoolikalt jälgida tulemusi, kuna tuleb arvestada kõigi tehniliste ning äriliste nõuetega. Inimeste kasutamine nende testide läbi viimiseks on ressursi raiskamine ja samas ka ohtlik, sest samade sammude pidev ja pikaajaline täitmine väsitab inimese tähelepanu ning tekib oht mitte tähele panna olulisi erinevusi. Seetõttu on mõistlik sellised kohad automatiseerida, kuna kompuuter läbib nad kõik alati olulisi kontrolle silmas pidades (eeldades, et need on testi koodi sisse kirjutatud).

## 2.2 Tarkvara automaattestimise kasulikkus

Automaattestimisel on palju kasulikke külgi, mida mõjutab eri projektide meeskonna ülesehitus, äriline loogika, tehnoloogilised valikud jne, kuid siiski on ka omadusi, mis kehtivad enamuse projektide puhul ning need on järgnevalt lühidalt välja toodud.

Kõige tähtsam automaattestimise juures on tema pikaajaline kasu erinevalt ainult manuaalsest testimisest, mida kasutades on hetke kulu küll väiksem, kuid regressiooni kasvades kulub igas tarnetsüklis järjest rohkem aega manuaalsele regressioontestimisele ning lõpuks ei ole see enam vajaliku ajaga teostatav. Regressioontestimine on testimise tüüp, mille eesmärk on kontrollida, et rakenduse uued funktsionaalsused töötavad nii nagu peavad arvestades seda, et need ei muudaks juba olemasolevat süsteemi mittesoovitud viisil [6].

Manuaalselt keerukaid teste saab mõnikord palju lihtsamalt ning kiiremini teha kasutades automaatteste. See on vajalik näiteks juhul, kui automaattestimise raamistikus on defineeritud meetodid serveripoolsetele teenustele ning funktsionaalsustele ligi pääsemiseks, mis käsitsi pole kas üldse võimalik või on raskesti teostatav. Samuti kasutatakse automaatteste enamasti koormus- ja jõudlustestide teostamisel, kuna on raske tekitada tuhandeid süsteemi kasutamisi arendusmeeskonnale vajalikul hetkel.

Manuaalselt kogu rakenduse pidev testimine kulutab palju inimressurssi ning läbi selle palju aega ning raha. Automatiseerimine säästab mõlemat ning aitab teostada suure hulga inimeste regressioontestimise tööd paralleelselt ning suure täpsusega.

Tulenevalt tarkvaralt oodatavate funktsionaalsuste pidevast kasvust, on paljude rakenduste äriloogika väga keeruline ning muutub ajas ainult keerulisemaks. Kui rakendus läheb väga suureks, siis ei jõua manuaalselt testides inimene kõiki äriliselt ning tehniliselt olulisi reegleid meeles pidada, kuid automaattestid vaatavad alati kõiki defineeritud reegleid ning maandavad teadmatuse ning tähelepanematuse riski [6].

Kuna automaattestimine säästab regressioontestimisele kuluvat aega, siis see tähendab seda, et testijatel on võimalik kulutada oma aega kasulikule testimisele. Kasuliku testimise all mõeldakse tegevusi, nagu näiteks uue funktsionaalsuse manuaalset testimist või parandatud vigade üle vaatamist, ehk tegevusi, mille tegemisel on vigade leidmise tõenäosus suur [4].

Oluline on eraldi välja tuua, et kuna automaattestid ei nõua inimese sekkumist, saab neid kergesti seadistada käivituma ka öösel. Kuna tihtipeale suurte rakenduste korral nõuab süsteemi täielik testimine palju aega ning arvutusressurssi, siis neid ei saa tööaegadel teostada. Sellistes olukordades on hea käivitada teste õhtuti või öösel, kuid kuna inimestelt ei saa oodata sellistel aegadel töötamist, siis on siinkohal parim lahendus ära kasutada automaattestimist.

Lisaks on suureks plussiks see, et iga testi ning testandmeid tuleb kirjutada ainult üks kord. Manuaalse testimise juures tuleb kõik olulised testid kindlasti üles kirjutada, et need oleks vajadusel kiiresti taaskorratavad ilma olulisi samme vahele jätmata. Kuna automaattestimise juures on testi disaini osa enamasti kõige aeganõudvam, siis saab paljud sellised juhtumid kirjutada juba kohe automaattestidena ning ei ole vaja edaspidi aeganõudvaid samme kordama hakata [6].

Automaattestimine on kasulik ka olukorras, kus test võtab kaua aega või on vaja käivitada kindlatel aegadel, siis saab automaattesti panna taustal käivituma eemal asuvas serveris. Sellisel juhul ei lähe raisku testide käivitamise aeg ning kohalik arvutusressurss [6].

Suur osa testimisest on inimesele rutiinne, kuid samas inimene vajab tähelepanelikult töötamiseks uusi väljakutseid ning vaheldust. Automaattestimine aitab selle jaoks hoida kokku aega. Tänu sellele püsib töötajal motivatsioon oma tööd teha ning firmadel ei toimu pidev tööjõu vahetus tulenevalt töökoha vähesest atraktiivsusest.

Kokkuvõtteks võiks öelda, et automaattestimisel on arvukalt häid külgi, mis aitavad tarkvaraprojekti arendamist lihtsamaks ning efektiivsemaks teha, kuid nende kasulike külgede väljendumine eeldab hoolikalt hoolikat planeerimist.

## 2.3 Tarkvara automaattestimise negatiivsed küljed

Kuigi automaattestimises on palju häid külgi, ebaõnnestuvad paljud projektid oma testide automatiseerimises. Sageli ei arvestata negatiivseid külgi, mis selle lähenemisega kaasnevad ning tehakse seetõttu valesid otsuseid. Näiteks hakatakse kogu testimist automatiseerima. Seetõttu tuleks järgnevaid mõtteid tutvustatava metoodikaga testide automatiseerimisel pidevalt jälgida, et kaitsta end ebaõnnestumise eest.

Vaatamata eelmises alampeatükis toodud põhjustele tuleb meeles pidada, et automaattestimine pole parim uute vigade leidmise meetod. Manuaalne testimine sobib selle jaoks paremini, kuna inimene mõtleb iseseisvalt, kuid arvuti teeb ainult nii, nagu on ette kirjutatud. Automatiseerimise kasu seisneb juba loodud komponentide nõuetele vastavuse pidevas jälgimises ning manuaalse regressioontestimise pealt kokkuhoidmises. See aitab tekitada kindlust kvaliteedi osas kohtades, mida käsitsi ei jõua hallata, kuid samas on olulised. Kuigi automaattestid vaatavad üle paljud kohad, mis tunduvad testijale olulised, ei garanteeri see vigade leidmist nendes kohtades, mida automaattestid ei kata.

Automaattestidele on omane anda palju valehäireid kohtades, kus tegelikult pole rakendus vigane, vaid viga seisneb näiteks rakenduse konfiguratsioonis või testandmetes. Valehäirete vähendamiseks tuleb hoolikalt teostada rakenduse esialgne konfiguratsioon ning see kergesti ühest kohast jälgitavaks teha. Samuti tuleb testandmeid koostada ning hooldada arvestades põhimõtet, et need oleks võimalikult läbinähtavad ning kergesti mõistetavad, mis võimaldab vigase testi korral kiiresti valehäire põhjuse tuvastada.

Samuti vajab automaattest vea avastamisel siiski ka inimlikku sekkumist, kuna testi ebaõnnestumine ei ole veel vearaport. Enamasti ei piisa, kui öelda programmeerijale, et see automaattest on vigane. Vea leidmisel tuleb seda tihtipeale veel lähemalt uurida, et saaks fikseerida võimalikult täpne veakoha. Lisaks tuleb koostada programmeerijale eraldi selgesti arusaadav vearaport, et viga oleks võimalik parandada ilma edasise aruteluta aja kokkuhoiu eesmärgil.

Kuigi automaattestimine hoiab pikas perspektiivis kulusid kokku, tuleb arvestada automaattestimise kasutusele võtmisel esialgse aja suure investeeringuga. Valitud raamistike kavandamine nõuab palju planeerimist, analüüsimist ning seadistamist ja sellega tuleb projekti alguses arvestada, vastasel juhul põhjustab esialgne ebapiisav planeerimine hilisemates etappides ootamatuid probleeme. Samuti tuleb koostatud testkogumikku aeg-ajalt hooldada,

kuna arendamises olev rakendus on pidevas muutumises ning sellest tulenevalt tuleb ümber defineerida ka testid.

## 2.4 Tarkvara automaattestimise ohud

Lisaks negatiivsetele külgedele on projekti testimise automatiseerimisel mitmeid ohte, mille ilmnemine oleneb selle kasutuselevõtust ning pidevast jälgimisest. Tarkvara automaattestimise ohud on näiteks:

1. Automaattestimist hakatakse rakendama kohtades, kus see pole enam mõistlik ehk üritatakse automatiseerida 100% rakendusest ning unustatakse manuaalse testimise olulisus ning kasulikkus. Automatiseerida tuleks ainult kohti, mis on selleks sobilikud ja mis toovad tulevikus ajalise võidu arvestades nii manuaalse testimise pealt säästetud kui ka automaatsete testide hooldamisele kuluvat aega.
2. Automaattestimisele kuluvat aega alahinnatakse. Kui hakatakse seadistama raamistikku, seda hooldama ja automaatteste tegema, siis sageli ei jõuta enam teist tüüpi testimistega tegeleda.
3. Kui alustada automaattestimisega ajal, mil on vaja uusi funktsionaalsusi manuaalselt pidevalt testida, siis on lihtne tekitada olukord, kus ei ole aega piisavalt automaattestidega tegeleda ning nad jäävad hooldamata ja loetakse lõpuks ebaõnnestunuks.
4. Võib tekkida ka olukord, kus tehakse valmis suur hulk teste, kuid ei arvestata aega hilisemale hooldusele ja testide uuendamisele või ei pöörata sellele piisavalt tähelepanu. See tekitab olukorra, kus automaatteste ei jälgita enam üldse või siis hakatakse järjest ebaõnnestuvaid teste välja võtma nende parandamise asemel.

Kokkuvõtteks võib öelda, et automaattestimine on omakorda nagu väike arendusprotsess, mis nõuab eraldi strateegiat, eesmärkide kavandamist, analüüsi, planeerimist, arendust ja testimist. Ehk see on kui omaette väikese rakenduse loomine suurema rakenduse jaoks ja seetõttu tuleks automatiseerimist käsitleda mitte kui kõrvalist tegevust, vaid suure olulise muudatusena ja täiendusena projektis, mis vajab palju tähelepanu ja planeerimist.

## 2.5 Testide valimine automatiseerimiseks

On palju erinevaid kohti, kus testide automatiseerimine on mõistlik ning kasulik. Selliste kohtade äratundmine nõuab hoolikat analüüsimist, kuid selle abistamiseks on järgnevalt välja toodud testimise tüübid, milles leidub tihti kasulikke automatiseerimise võimalusi [6]:

1. Ühiktestimine ehk individuaalsete ühikute valideerimine rakenduse lähtekoodis.
2. Regressioontestimine ehk kontrollimine, et eelnevalt testitud funktsionaalsused töötavad endiselt.
3. Funktsionaalne testimine ehk veendumine, et süsteem vastab funktsionaalsetele nõuetele.
4. Turvatestimine ehk kinnitamine, et süsteem vastab turvalisuse nõuetele [6].
5. Jõudlustestimine ehk kontrollimine, et süsteem tuleb toime kokkulepitud jõudluse nõuetega.
6. Koormustestimine ehk süsteemi koormustaluvuse piiride leidmine ning fikseerimine.
7. Stresstestimine ehk veendumine, et süsteemi koormustaluvuse piiride ületamisel süsteem sulgub ootuspäraselt ning samuti taastub sulgumisest korrapäraselt.
8. Samaaegsuse testimine ehk verifitseerimine, et süsteem suudab taluda paralleelseid lõimi ning kasutajaid.
9. Testidega kaetuse kontroll ehk mõõtmine, mitu protsenti süsteemi koodist käivitatakse testimise käigus.

Samamoodi nagu on hulgaliselt kohti, mida tasub automatiseerida, on ka hulgaliselt kohti, kus automaattestimine ei too kasu, vaid pigem kahju. Seda arvesse võttes tuleks enne automaatse lahenduse kasutuselevõttu põhjalikult projekti tundma õppida ning teha analüüs erinevate lähenemiste kasulikkusest ning kahjulikkusest. Selle teostamist aitab läbi viia töö edasistes peatükkides tutvustatav ATLM (*The Automated Testing Lifecycle Methodology*) mudel.

# 3. Võtmesõnadel põhinev testimine

## 3.1 Metoodika üldine kirjeldus ning eesmärgid

Paljudes organisatsioonides on tarkvara testimise osakaal umbes 30-50% kogu tarkvara arenduse kulust. Samas paljud usuvad, et tarkvara ei ole piisavalt hästi testitud enne, kui see tarnitakse kliendile. Sellise vastuolu põhjused seisnevad kahes faktis – esiteks, tarkvara testimine on väga keeruline ja teiseks, testimist teostatakse tavaliselt selgete metoodikateta [7].

Võtmesõnadel põhinev tarkvara testimine on metoodika, mida saab kasutada nii manuaalse kui ka automaatse testimise juures, et testidest arusaamist lihtsustada ning aidata seda teostada plaanipäraselt. Samas manuaalse testimise juures kasutatakse seda harva, kuna vaatamata sellele, et ilma suurema vaevata saab kirjutada testjuhtumid selle metoodika formaadis, on nende kirjutamiseks teisi alternatiivseid formaate, mis on mugavamad ning selgemini mõistetavad manuaalse testimise jaoks. Nagu eelnevalt mainitud, käsitleme seetõttu antud töö raames metoodikat ainult automaattestimise kontekstis.

Metoodika nimi tuleneb sellest, et testide kirjutamisel kasutatakse võtmesõnu. Võtmesõna on mingi fikseeritud sõna või lause (näiteks „Sisesta kasutajanimi“), mis sümboliseerib mingi tegevuse või kontrolli teostamist testitavas rakenduses. Võtmesõnu võib kirjutada vabalt valitud keele (näiteks eesti või inglise) kirjakeeles. Nende võtmesõnadega kaasnevad täpsemad tegevused rakenduses kirjeldatakse omakorda eraldi, kus kasutatakse konkreetses programmeerimiskeeles kirjutatud meetodeid.

Võtmesõnadel põhineva testimise peamiseks eesmärgiks on luua testkogumik, kus testid ning nendes kasutatavad andmed oleks kõigile loetavad. See saavutatakse tänu asjaolule, et defineeritavaid võtmesõnu kirjutatakse kõigile mõistetavas kirjakeeles. Võrdluseks võib tuua ühiktestid, mis on loetavad ainult heade tehniliste teadmistega ning rakenduse koodiga tuttavale isikule. Testide parema loetavuse teostamiseks jagab metoodika ka testide disaini ning arenduse ja käivitamise kaheks selgelt eristuvaks kihiks, peites kõik tehnilise informatsiooni testide lugeja eest.

Samuti on oluliseks eesmärgiks pakkuda võimalust luua teste arendusprotsessi võimalikult varajases etapis [8]. Selle tulemusel saavad testijad ära kasutada paremini aega, mil rakenduse funktsionaalsused pole veel valmis ja seega nende lõplikul valmimisel on juba funktsionaalsust kontrollivad automaattestid olemas. Tihti vajavad need testid peale funktsionaalsuse valmimist

küll täiendamist tulenevalt ootamatutest lahenduse muudatustest, kuid siiski saab need enamasti kiiresti peale valmimist käivitada ning olla kindel, et loodud funktsionaalsus käitub vastavalt ootustele ning täidab ärilised eesmärgid.

Tänu võtmesõnade kasutamisele ning selgeteks kihtideks jaotamisele saavad testijad tutvustatava meetodi abil:

1. kirjutada teste kirjakeeles, milles saab selgelt kirjeldada äriliselt olulised testjuhtumid kõigile arusaadavalt;
2. kirjutada automaatteste, omamata selleks häid tehnilisi teadmisi.

Läbi nende omaduste saavad teste lugeda kõik projekti ning kliendi meeskonna liikmed ning tagada sellega ühine arusaam testidega läbitavatest kasutusjuhtudest ning kõigil on võimalus täiendada puudulikke kohti. Samuti aitab kihtideks jaotamine muuta hooldamist enamasti lihtsamaks, kuna võimaldab testid raamistikus struktureerida selliselt, et muutuste korral oleks muudatusi vaja teostada võimalikult vähestes kohtades.

Võtmesõnadel põhinev testimine aitab defineerida vajaliku selge metoodika automaattestide planeerimiseks, teostamiseks ning hooldamiseks, mis aitab testprotsessi lihtsustada. Samal ajal aitab see kulusid vähendada kasutades paremini ära olemasolevat aega ning teisi ressursse. Läbi nende omaduste aitab meetod potentsiaalselt lahendada kahte peamist tarkvara testimise probleemi.

## 3.2 Võtmesõnadel põhinevate testide kirjutamine

Võtmesõnadel põhinev testimine kasutab testide kirjutamisel võtmesõnu, et sümboliseerida tegevusi ning kontrolle, mida soovitakse testiga teostada. Tihti nimetatakse võtmesõnadel põhinevat testimist ka tabelitel põhinevaks testimiseks. See tuleneb sellest, et metoodikas kasutatavates raamistikes toimub nende võtmesõnade kirjutamine tabulaarses formaadis (sarnaselt näiteks Microsoft Exceli tarkvaras tabelite loomisele).

*Tabel 2. Võtmesõnadel põhinev näidistest*

| Võtmesõna | Väärtus 1 |
|---|---|
| Ava rakendus veebilehitsejas | |
| Sisesta kasutajanimi | Testkasutaja |
| Sisesta parool | Parool123 |
| Logi kasutajana rakendusse | |
| Kontrolli avalehe avanemist | |

Tabelis 2 võib näha väga lihtsat võtmesõnu kasutatavat testi. Selliselt kirjutatud test peaks olema kõigile loetav ja mõistetav. Kogu tehniline keerukus on peidetud nende võtmesõnade taha. Enamus meeskonna liikmeid ning klient vaatavad teste ainult selles kihis ning sügavamatele tasemetele minnakse enamasti ainult testide täiustamise või parandamise eesmärgil tehnilise testija või programmeerija poolt.

Enamus raamistikke võimaldavad anda ka selgelt loetava nimekirja, mis võtmesõnad on juba loodud. Samuti on kõikides vahendites võimalik kasutada võtmesõnade otsingut. Isegi kui võtmesõnu tekib palju, on olemas võimalus leida endale sobiv või sobiliku puudumisel saab neid enamasti hõlpsasti juurde lisada.

*Tabel 3. Võtmesõna „Ava rakendus veebilehitsejas“ näite definitsioon*

| Meetod | Argument 1 | Argument 2 |
|---|---|---|
| *open_browser* | https://rakendus.com/test-login-leht/ | Firefox |
| *wait_until_page_contains_element* | xpath=.//*[@id='userPasswordLoginForm'] | |

Kui nüüd võtta Tabelist 2 võtmesõna „Ava rakendus veebilehitsejas“, siis see on omakorda defineeritud eraldi kohas tabeli kujul, mis kasutab käivitamise kihis konkreetses programmeerimiskeeles kirjeldatud meetodeid. Tabelis 3 on kujutatud, kuidas võtmesõna defineerimine võiks välja näha. Esimeses veerus on välja toodud näiteks Java’s või Python’is defineeritud meetodi nimi, mis kirjeldavat metoodikat kasutavasse töövahendisse on imporditud, tänu millele võtmesõnad neid kasutada saavad. Meetodile järgnevates tulpades saab anda meetodile nii palju parameetreid kui meetod nõuab.

```
def open_browser(self, url, browser='firefox', alias=None,remote_url=False,
            desired_capabilities=None,ff_profile_dir=None):

    if remote_url:
        self._info("Opening browser '%s' to base url '%s' through remote server at '%s'"
                % (browser, url, remote_url))
    else:
        self._info("Opening browser '%s' to base url '%s'" % (browser, url))
    browser_name = browser
    browser = self._make_browser(browser_name,desired_capabilities,ff_profile_dir,remote_url)
    try:
        browser.get(url)
    except:
        self._cache.register(browser, alias)
        self._debug("Opened browser with session id %s but failed to open url '%s'"
                    % (browser.session_id, url))
        raise
    self._debug('Opened browser with session id %s'
                % browser.session_id)
    return self._cache.register(browser, alias)
```

*Joonis 2. Python’is kirjutatud open_browser meetod [9]*

Joonisel 2 on esitatud meetodi *open_browser* definitsioon. Siit on näha, miks käivitamise kihis tegutsemine nõuab häid tehnilisi (näiteks programmeerimise) teadmisi.

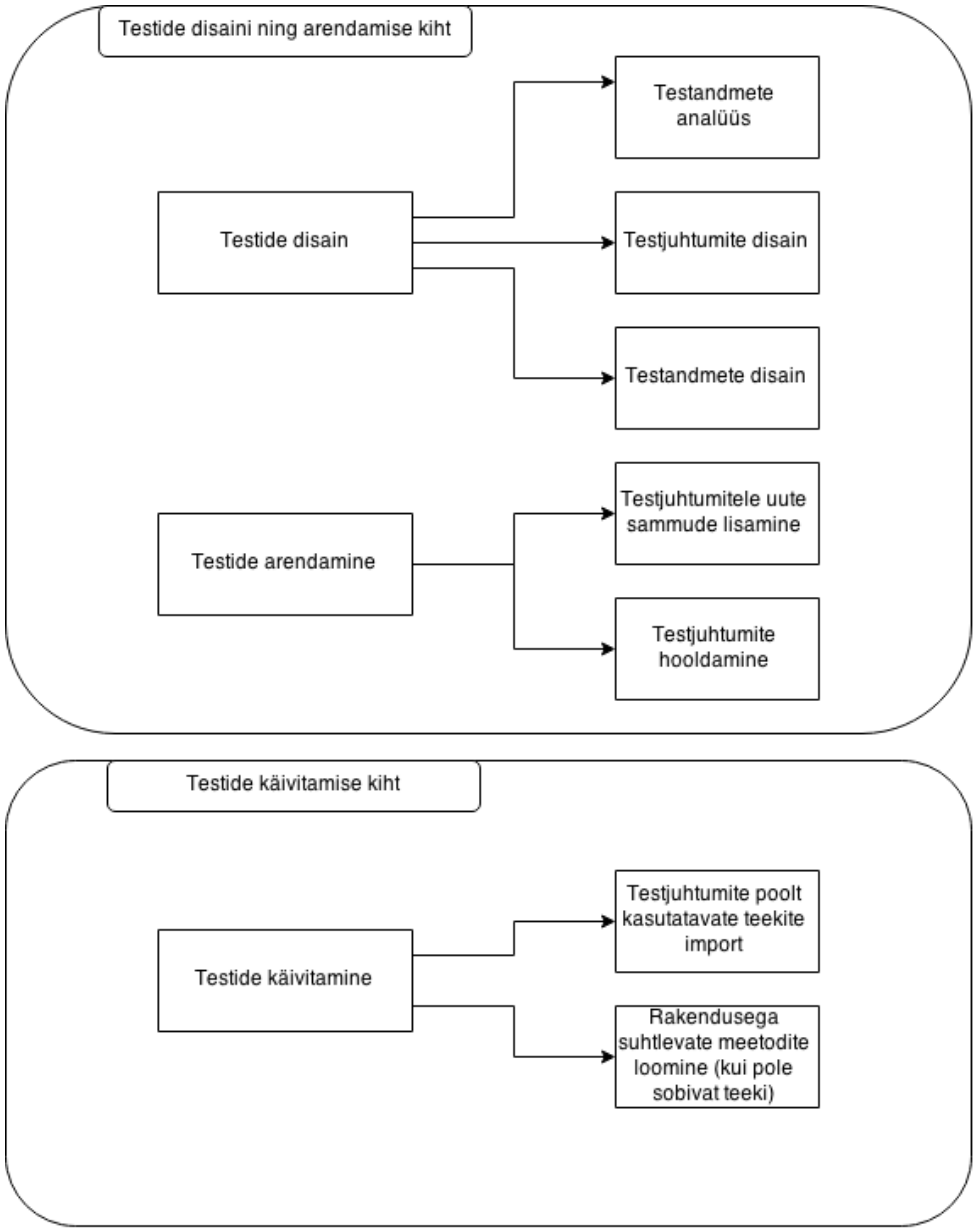
Nii võtmesõnadega testide kirjutamine kui ka võtmesõnade defineerimine toimub testide disaini ning arendamise kihis. Käivitamise kihis defineeritakse programmeerimiskeeles tehtavad meetodid, mis rakendusega ühenduvad. Täpsemalt saab kihtide kohta lugeda järgnevates peatükkides.

Võtmesõnu kavandama ning arendama peaks testija, kellel on head teadmised rakenduse valdkonnast, kasutusjuhtudest, ohtudest ning ärilistest nõuetest. Ainult läbi nende teadmiste saab luua testkogumiku, mis toob rakenduse kvaliteedile kõige suuremat kasu. Juhul, kui heade valdkonnateadmistega testija oskab ka programmeerida, võib ta kirjutada ka käivitamise kihis meetodeid ning struktureerida peidetud tehnilist poolt.

Kuna võtmesõnu võib kirjutada endale sobivas keeles, siis tuleb neid kirjutama hakates otsustada, milline keel sobiks projekti kontekstis kõige paremini. Siinkohal ei tasuks alati oma emakeele kasuks otsustada, kuna juhul kui tulevikus liitub näiteks mõni emakeelt mitterääkiv isik, siis ei ole tal võimalik testidest aru saada. Samuti tuleb arvestada, et kui tegemist on projektiga, mis tõenäoliselt kasvab väga suureks ning sellega võivad hakata tegelema meeskonnaliikmed ka teistest riikidest, siis pole ka neil võimalik teste mõista ning hooldada. Sellistel juhtudel on soovitatav teste kirjutada hoopis näiteks inglise keeles, mis on enamasti kõigi valdkonnas tegelevate inimeste poolt mõistetav.

## 3.3 Testide disaini ja arendamise ning käivitamise kihid

Eelmises paragrahvis käsitletu põhjal jaotab metoodika testide kirjutamise kahte eraldi kihti, testide käivitamise ning testide disaini ja arendamise kihti. Mõlemas kihis toimuvad kindlad tegevused, mis aitavad luua selge testide kavandamise, loomise, edasiarendamise ning käivitamise töövoo.




*Joonis 3. Võtmesõnadel põhineva testimise kihtide jagunemine*

Joonisel 3 on näha võtmesõnadel põhineva testimise kahte suuremat kihti ning nendes toimuvaid tegevusi. Testide disaini ja arendamise kihis toimub testide kirjeldamine kõigile loetavas formaadis. Enamasti tegutsevad selles kihis testijad, kellel on kogemus ja teadmine testjuhtumite koostamiseks. Kõigepealt analüüsitakse selles kihis testimiseks vajalikke andmeid, seejärel disainitakse nende põhjal testjuhtumid ning testandmed. Peale testide disaini saab arenduse faasis testjuhtumid ning testandmed sisestada valitud raamistikku ning samas kihis ka neid edasi täiendada ning uuendada. Enamusel järgnevates peatükkides kirjeldatud meetodit toetavatel raamistikel on testi sisestamise jaoks ka kasutajasõbraliku kasutajaliidese võimalus. Tänu sellele on kõigile nähtavas kihis testjuhte võimalik raamistikku kirjutada ka tehnilisi teadmisi mitte omaval isikul.

Käivitamise kihis toimub raamistiku ühendamine testitava rakendusega. Projektis kasutatavatele eri tehnoloogiatele ligi pääsemiseks on vaja erinevaid võimalusi ning just selles kihis defineeritakse või luuakse kõik vajalikud võimalused. Võtmesõnadel põhineva testimise raamistikus kasutatavad teegid ning kirjeldatud meetodid on esimesele kihile kättesaadavad ning neid kasutatakse võtmesõnade defineerimisel. Seda kihti planeerivad, loovad ning hooldavad enamasti kas heade tehniliste teadmistega testijad või programmeerijad. Kuigi metoodika üheks peamiseks omaduseks on väike tehniliste teadmiste nõue, on käivitamise kihi koostamiseks siiski olulised ka tehnilised teadmised. Peale käivitamise kihi esialgset seadistamist on edaspidi harva vajadus uuendusteks selles kihis. Tihedamat hooldust ning uuendust vajab ainult esimene kiht, kus tehnilised teadmised pole vajalikud. Muidugi oleneb see kõik käivitamise kihis kavandatud arhitektuurist.

Käivitamise kiht on enamasti kirjutatud tuntumates programmeerimiskeeltes nagu Python või Java, kuid raamistikel on tavaliselt vähe piiranguid selles osas ning iga raamistik toetab paljusid programmeerimiskeeli.

Sellise kahekihilise eristuse kõige suurem väärtus seisneb tehnilise, keeruliselt mõistetava loogika mujale peitmises ning kõigile kergesti mõistetava informatsiooni nähtavale jätmises. Mõlemad kihid on vajalikud rakenduse testimiseks, kuid samas on nad üksteisest sõltumatud ja neid saab vaadelda eraldi. See tähendab, et ühes programmeerimiskeeles kirjutatud käivitamise kihti saab asendada teises keeles kirjutatud käivitamise kihiga (eeldades, et meetodite nimed on samad) ilma, et testide disaini ning arenduse kihis tuleks muudatusi teha. Samuti kehtib see vastupidi - eeldusel, et disaini ja arenduse kiht kasutavad samu võtmesõnu, võivad sama käivitamise kihi peal töötada erinevad testkogumikud.

## 3.4 Võtmesõnadel põhineva testimise kasulikkus

Suuremal osal firmadest on olemas mitmeid konkurente, kes võistlevad samade klientide saamise nimel. Klientide endale võitmiseks on vaja oma toode võimalikult kiiresti turule saada ning klientidega sidemeid looma hakata. Samas turule minev toode peab olema kvaliteetne, sest vastasel juhul kliendid ei soovi toodet kasutama hakata. Seega testimist ei saa kõrvaldada toote valmimise kiirendamiseks, kuid seda tuleb muuta efektiivsemaks. Kuna võtmesõnadel põhinev testimine aitab säästa firmadel suures pildis palju aega läbi oma tööjõu parema ärakasutamise ning tootest selgema ühise pildi loomise, siis selle nurga alt vaadates on metoodikal suur potentsiaal tuua firmale hea eelis oma konkurentide ees.

Võtmesõnadel põhineva testimise metoodika peamised head omadused saab kokku võtta järgmiselt:

- Metoodikaga automaattestide kirjutamine sobib kasutamiseks ka uutele töötajatele – see ei nõua eelnevat kogemust ja tänu sellele saab iga uus töötaja vajadusel kiiresti teste looma või hooldama hakata. Muidugi tuleb siin meeles pidada, et nende testide disaini peaks looma keegi suurema kogemusega testija.
- Testide disaini kihis saavad teste kirjutada ka tehniliste teadmisteta töötajad. Kuna testide disaini kihis toimub testide kirjutamine kasutades võtmesõnu, milleks on tihtipeale kõnekeelsed laused, siis selles kihis tegutsemine ei eelda mingeid tehnilisi teadmisi.
- Metoodika aitab luua ühise arusaama projektis tehtavatest kvaliteedi kontrollidest. Loodud testid on nii projekti tellijale kui ka kõigile meeskonna liikmetele selgelt loetavad ning neid üheskoos üle vaadates saab kergesti leida veel kontrollimata ohukohad või ärireeglid.
- Teste saab looma hakata enne rakenduse valmimist ning vajalikud meetodid saab käivitamise kihis taustal hiljem juurde lisada.
- Hooldus nõuab enamasti vähem aega kui teiste testimise automatiseerimise lahenduste korral. Samas see eeldab läbimõeldud testide struktureerimist disaini ning arenduse ja meetodite käivitamise kihis.
- Metoodikal põhinevad raamistikud ei ole enamasti sõltuvad kindlast programmeerimiskeelest või tehnoloogiast tulenevalt metoodikale omasest kihtideks jaotamisest.

- Võtmesõnad on uuesti kasutatavad üle testkogumiku.
- Kuna võtmesõnadel põhinevad testid sarnanevad manuaalse testimise jaoks loodavate skriptidega, on hea jälgida, millised manuaalsed testid on juba automaattestidega kaetud.

## 3.5 Võtmesõnadel põhineva testimise puudused

Nagu metoodikal on palju kasulikke omadusi, on sellel olemas ka negatiivseid külgi. Üritades metoodikat kasutusele võtta, tuleks heade külgede kõrval kindlasti arvestada ka ohtudega, et metoodika kasutusele võtul oleks suurem võimalus õnnestuda. Kindlasti tuleks mõelda, kas mõni negatiivne metoodika külg või oht saab projekti kontekstis suureks takistuseks.

Võtmesõnadel põhineva testimise peamisteks negatiivseteks külgedeks on:

- Algne raamistiku seadistamine ning vajalike meetodite loomine võib olenevalt rakendusest võtta palju aega ning planeerimist. Seetõttu tuleb planeerida esialgseks kasutuselevõtuks piisavalt aega. Läbimõtlemata struktuuriga raamistiku ülesseadmine võib hiljem palju rohkem arendusaega võtta.
- Õppimiskõver on järsk esialgse seadistuse teostamiseks ning testide struktureerimiseks kui pole varasemat kogemust metoodikaga.
- Kuigi testjuhtumite kirjutamine testide disaini ning arendamise kihis ei nõua tehnilisi teadmisi, nõuab käivitamise kihiga tegelemine põhjalikke tehnilisi teadmisi. Meeskonnas peab olema keegi, kellel on vajadusel võimalus ning oskused sellega tegeleda.
- Metoodikat kasutavates raamistikes olevad raportid on vähem konfigureeritavad kui näiteks ühiktestide puhul. Konfigureerimise puhul sõltub meeskond raamistike poolt pakutavatest võimalustest ning ise juurde kirjutada funktsionaalsust on keeruline.
- Erinevatel metoodikat kasutatavatel raamistikel on väike IDE valiku võimalus ning seega tihtipeale on valida ainult 1-2 keskkonda, milles raamistikku kirjutama hakata. Näiteks Robot Framework'il on ainult RIDE ning Eclipse's olev pistikprogramm.
- Metoodika sobib peamiselt ainult süsteemi testide tegemiseks, näiteks konkureeriva BDD (*behaviour driven development*) lähenemise korral saab lisaks teha andmetel põhinevaid ühikteste.
- Võtmesõnu on keeruline dokumenteerida viisil, et kogu meeskonnal oleks selge, milliseid võtmesõnu tuleks kusagil kasutada. Mõnikord nõuab see kasutaja tutvumist pika nimekirja võtmesõnadega. Samas enamasti saab eelmiste testide näitel uusi kirjutada.

## 3.6 Metoodikaga kaasnevad ohud

Lisaks erinevatele negatiivsetele külgedele on metoodikal ka mitmeid ohte, mis võivad ilmneda teatud olukordades ja millega tuleks arvestada. Sellisteks ohtudeks on näiteks:

- Testide halb struktureerimine põhjustab hoolduse kulu kasvamise. Selle tulemusel võib juhtuda isegi olukord, kus hooldamine muutub niivõrd kulukaks, et sellest tulenev kulu ületab tulu. Seetõttu tuleb kogu struktuur algusest peale läbi mõelda ning kooskõlastada automaattestide struktureerimisega tegelenud isikutega.
- Tehnilisele testijale võib võtmesõnade kasutamine tunduda pigem piiravana. Kuigi reaalseid piiranguid on metoodikat kasutavatel rakendustel vähe, võib esialgu ainult testide disaini ning arenduse kihti vaadates tunduda, et võimalused on piiratud. Tegelikult on peidus käivitamise kiht, mis on ühenduses konkreetse programmeerimiskeelega, kus saab toestada kõike, mida valitud programmeerimiskeel võimaldab. Sellise arvamusega inimestele tuleks tutvustada käivitamise kihi loogikat.
- Lähenemise kasutuselevõtt võib nõuda juhtide tuge suurt toetud tulenevalt selle lähenemise kasutuselevõtu esialgsest suurest ajakulust. Seetõttu tuleb osata selgelt seletada selle otsene ajaline ning rahaline kasu projektijuhtidele. See tähendab, et tuleb osata seda lähenemist juhtkonnale müüa.
- Meeskonnaliikmetel puudub huvi teste arutada ning koos üle vaadata. Selle riski vähendamiseks tuleb eelnevalt meeskonnaga kokku leppida, et teste hakatakse koos arutama ning üle vaatama teatud kindla aja tagant.

## 3.7 Testimisprotsessi üldine kirjeldus kasutades võtmesõnadel põhinevat testimist

### 3.7.1 Kolme Amigo protsessi kasutamine testimisprotsessis

Et saada suurimat kasu tutvustavast metoodikast, tuleks üritada testimise protsessi alustada mitte siis, kui funktsionaalsus on juba valminud, vaid juba funktsionaalsuse disaini faasis. Selle teostamiseks on soovitatav tutvuda näiteks Kolme Amigo [10] protsessiga.

Kolme Amigo protsess on põhimõtteliselt koosolek, mille jooksul analüütik tutvustab testijale ning programmeerijale uut funktsionaalsust ja leitakse koos vigased, puudulikud või täiendamist vajavad kohad. Peamiseks põhjuseks, miks seda protsessi sobib hästi kasutada võtmesõnadel põhineva testimise juures, on asjaolu, et kohtumise tulemusel tekib ühine arusaam ja sõnavara rakenduse kohta. See aitab luua paremaid teste eriti võtmesõnadel põhineva testimise kontekstis, kuna võtmesõnad saab kirjutada neilt koosolekutelt saadud arusaama ning sõnavara alusel.

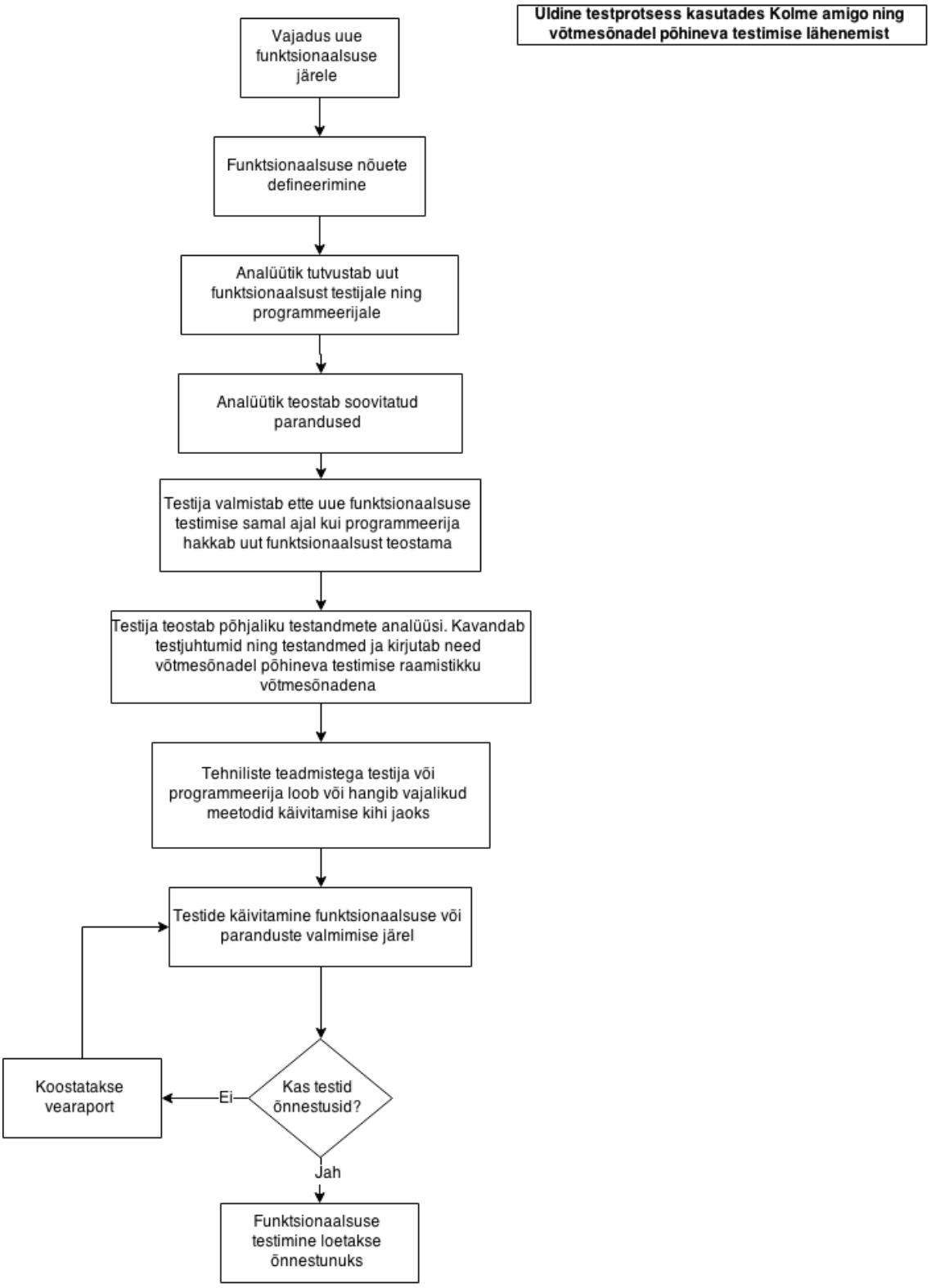
Kolm peamist probleemi, mida Kolme Amigo lähenemine aitab lahendada, on järgmised [10]:

1. Segadus (milliseid kriteeriumeid tuleb funktsionaalsusel täita, et lugeda see õnnestunuks?)
2. Keerukus (milliseid muudatusi võib testija oodata peale programmeerija tehtud muutusi?)
3. Duplikatsioon (kui palju testimisest katavad ühiktestid ning kui palju tuleb manuaalselt või automaatselt testijatel endil teha?)

Selle protsessi heade omadustena võib välja tuua [10]:

1. Kohtumise tulemusel tekib ühine sõnavara projekti meeskonnas.
2. Tekib ühine arusaam nõuetest.
3. Tekib ühine arusaam vajalikest testidest.
4. Toode läheb arendusse alles siis, kui tekib üksmeel, et funktsionaalsus on piisavalt täpselt kirjeldatud.

### 3.7.2 Näide testimisprotsessist kasutades Kolme Amigot ning võtmesõnadel põhinevat testimist




*Joonis 4. Üldine testprotsess kasutades Kolme Amigo ning võtmesõnadel põhineva testimise lähenemist*

Kasutades võtmesõnadel põhinevat metoodikat, on oma testprotsessi võimalik üles seada väga erinevatel viisidel. Järgnevalt on toodud üks võimalus, kuidas võiks protsess üldiselt välja näha kasutades soovitatud protsesse ja meetodeid.

Joonisel 4 kujutatud protsessi alguseks loeme kõigepealt vajadust luua uus funktsionaalsus. Selle jaoks tuleb soov enamasti projekti kliendilt või analüütikutelt. Selles sammus testija ning programmeerija ei sekku, kuid mõlemat tuleks saabunud funktsionaalsuse soovist teavitada, et nad oskaks oodata selle arenduse tulekut ning oma tööde ajakava sellest lähtuvalt planeerida.

Peale selge uue funktsionaalsuse soovi fikseerimist on soovitatav analüütikul defineerida selle funktsionaalsuse nõuded. Siinkohal paneb analüütik kirja kogu informatsiooni, mis tema hinnangul on vajalik, et see funktsionaalsus valmis arendada. Selles sisalduvad ärilised nõuded, arhitektuurilised muudatused (näiteks andmemudelis) ning oodatavad testjuhtumid, mis tuleks kindlasti läbida. Selles sammus tuleks ka kokku leppida kohtumine programmeerijate ning testijatega või nende rollide esindajatega, et tutvustada uut funktsionaalsust ning arutada puudujääke.

Enne funktsionaalsuse arendusse minekut toimub analüütiku, programmeerija ning testijate vahel koosolek, kus analüütik tutvustab loodavat funktsionaalsust. Järgnevalt testija ning programmeerija aitavad leida puuduvaid nõudeid ning erijuhtumeid, mida pole spetsifikatsiooni dokumentides veel mainitud. Kokkusaamis(t)e tulemusel tekib arendusmeeskonna liikmetel ühine arusaam ning sõnavara uuest funktsionaalsusest ning selle vajadustest, mida saab hiljem võtmesõnade loomisel kasutada.

Edasi testija valmistab testimist ette ehk hangib kõik testimiseks vajamineva informatsiooni ning testandmed. Näiteks hangib analüütikult tutvustatud spetsifikatsiooni ja uurib, kas on vaja näiteks testandmetena erinevate omadustega kasutajate andmeid.

Järgnevalt analüüsib testija põhjalikult hangitud materjali, loob endale selge detailse pildi kirjeldatud protsessist ning kavandab nende põhjal testjuhtumite ning testandmete kogumikud võtmesõnadel põhineva testimise raamistikus. Raamistikus kasutatavate võtmesõnade loomisel tuleb testijal kasutada analüütiku ning programmeerijaga suhtlemisel saadud ühist arusaamist ning sõnavara testimise all oleva funktsionaalsuse kohta. Selle sammu lõpuks on võtmesõnade raamistikus kirjeldatud võtmesõnadena testid ning nendes tehtavad sammud.

Enne rakenduse testide käivitamist tuleb heade tehniliste teadmistega testijal või programmeerijal leida käivitamise kihi jaoks sobivad teegid või luua ise vajalikud meetodid.

Lõpuks, kui uus funktsionaalsus on valminud, käivitab testija loodud testid sobival rakenduse versioonil ning vigade ilmnemisel tehakse testimise raport. Kui uurimise tulemusel selgub, et viga seisneb testides, tehakse disaini ja arenduse või/ja käivitamise kihti parandused. Soovitatav on proovida ka kohe peale testide valmimist ja enne funktsionaalsuse valmimist teste käivitada, et välistada vale positiivse tulemuse võimalus hilisema käivitamise korral.

## 3.8 Konkureerivatest metoodikatest

Võtmesõnadel põhineva testimise peamiseks konkureerivaks metoodikaks on käitumisel põhinev arendamine (*behaviour driven development*). Järgnevalt tuuakse lühidalt selle metoodika kirjeldus ning erinevused ja sarnasused võtmesõnadel põhineva metoodikaga. Kummalgi meetodil pole selgeid eeliseid teineteise ees, kuna projektide kontekstid on väga erinevad ning metoodika negatiivsed küljed ühes olukorras olevad võivad kiiresti muutuda positiivseks teises kontekstis.

Käitumisel põhinev arendus on välja arenenud testidel põhinevast arendusest (*test driven development*) ning selle loomise eesmärgiks on anda paremad võimalused testide loomiseks ühiktestide tasemel ärilistest eesmärkidest lähtudes [11].

Käitumisel põhineva arendamise puhul kasutatakse testimisel rakenduse lähtekoodi klasse, samas kui võtmesõnadel põhinev testimine kasutab rakenduse teenuseid. Mõlemal lähenemisel kasutatakse kirjakeelseid lauseid tegevuste märkimiseks. Käitumisel põhineva arenduse testi näide on järgmine:

> @tag_sisse_logimine
> **Scenario Outline:** Kontrolli, et saad rakendusse sisse logida
> **Given** Ma käivitan http://rakendus.com lehe
> **When** Ma täidan „Kasutajanimi“ väärtusega „<Kasutaja>“
> **And** Ma täidan „Parool“ väärtusega „<Parool>“
> **Then** Ma satun esilehele

Kahe metoodika peamisteks sarnasusteks saab loetleda järgnevaid omadusi:

1. Mõlemad aitavad luua paremat suhtlust eri rollide vahel, kuna teevad teste paremini loetavaks kasutades kirjakeelsete lausetena kirjutatud teste.
2. Mõlema metoodika puhul keskendutakse testimisele lähtudes ärilistest vajadustest. Seega mõlemad eeldavad, et testide kirjutaja on äriloogikaga tuttav.

Kahe metoodika peamised erinevused on järgnevad:

1. Võtmesõnadel põhinevat testimist kasutavad enamasti testijad. Käitumisel põhinevat arendust kasutavad enamasti programmeerijad. See tuleneb põhiliselt käitumisel

põhineva arenduse suuremast tehniliste teadmiste nõudest ja sellest, et testija soovib automatiseerimist teostada pigem süsteemitestide tasemel.

2. Käitumisel põhineva arenduse puhul kirjutatakse suurem osa testimise loogikast ühiktestidele omasel kujul ning on seetõttu suuremate tehniliste teadmiste nõudega.
3. Käitumisel põhineva arenduse puhul saab kasutada sisukamaid lauseid, kuna seal ei arvestata nii palju korduvkasutamisega. See tähendab, et on võimalik mõelda just antud testi jaoks kõige sobivamaid lauseid ning ei pea samal ajal mõtlema, et need sobiks ka teistele testidele.
4. Võtmesõnadel põhineva testimise puhul toimub testides kasutavate lausete suurem korduvkasutamine. Käitumisel põhineva arenduse puhul koostatakse testide sisu keskendudes ainult konkreetse testi vajadustele. Võtmesõnadel põhineva testimise korral mõeldakse pidevalt ka sellele, kuidas saaks koostatud võtmesõnu teistes testides kasutada. See teeb käitumisel põhineva arenduse hooldamise raskemaks.
5. Käitumisel põhineva arenduse korral pole vaja rakendust käivitada, sest testitakse läbi lähtekoodi klasside ning meetodite.

Kokkuvõtteks võib öelda, et kuigi metoodikate korral kirjutatakse teste väga erinevates tasemetes (üks lähtekoodi tasemel ning teine süsteemi tasemel), on nende eesmärk sama. Mõlema eesmärgiks on koostada paremini loetavaid teste, lähtudes ärilistest vajadustest ning eesmärkidest ja seetõttu on nad konkureerivad lähenemised.

# 4. Võtmesõnadel põhineva testimise raamistikud

## 4.1 Võtmesõnadel põhineva testimise metoodika raamistikest

Testide automatiseerimise raamistikku võib defineerida kui abstraktsete ideede, protsesside, protseduuride kogumikku ja keskkonda, milles automaattestid disainitakse, luuakse ja teostatakse. Automaattestimise raamistiku eesmärgiks on viia projekti testkogumik seisu, kus kõigi testide käivitamine toimub üheainsa nupuvajutusega. Võid öelda, et raamistik annab erinevate metoodikast tulenevate ideede teostamiseks vajalikud vahendid [12].

Raamistiku ülesanneteks on järgmised tegevused:

1. Kirjeldada formaat, kuidas teste sisestatakse.
2. Pakkuda võimalusi raamistiku ühendamiseks testimist vajava rakendusega.
3. Võimaldada testide käivitamist.
4. Pakkuda võimalust tulemuste raportite koostamiseks eri osapooltele.

Võtmesõnadel põhineva testimise metoodika kogub järjest rohkem populaarsust ja kasutajaid ning tänu sellele suureneb ka raamistike valik, mis toetuvad sellele metoodikale. Erinevatel raamistikel on palju erinevaid funktsionaalsusi, kuid kõik nad üritavad eri viisidel toetada eelnevalt käsitletud ideede ning põhimõtete teostamist.

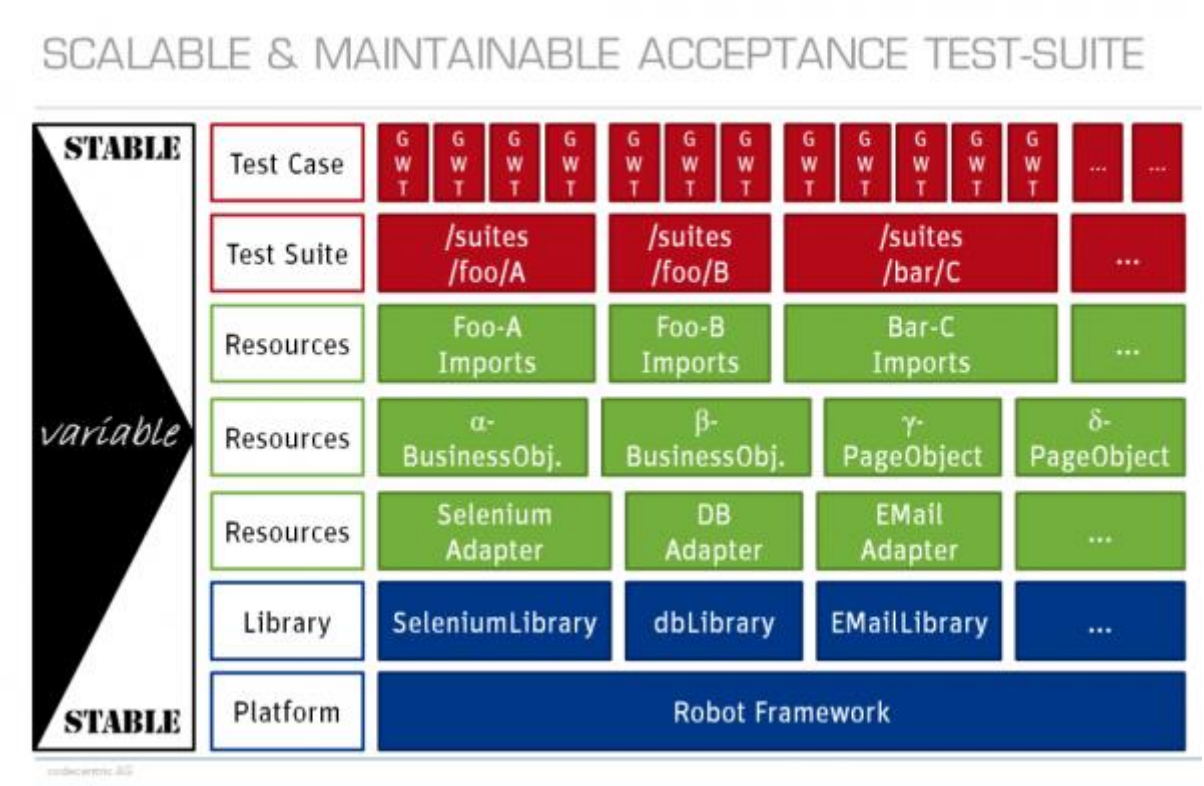
Võtmesõnadel põhineva testimise metoodikat toetavad raamistikud on näiteks:

- Robot Framework [13]
- FIT/FitNesse [14]
- EMOS Framework [15]
- TestComplete [16]
- SAFS [17]
- Certify (Worksoft) [18]
- TestArchitect (LogiGear) [19]

Töö raames keskendume **Robot Framework** ning **FIT** automaattestimise raamistikule, kuna need on käesoleva töö kirjutamise hetkel kaks kõige laialdasemalt kasutusel olevat võtmesõnadel põhinevat testimise raamistikku nii Eestis kui ka väljaspool Eestit. Oma suure kasutatavuse tõttu on neil väga head abimaterjalid ning suur kasutajate kommuun.

## 4.2 Raamistike tasemed

Tavapärases võtmesõnadel põhinevas raamistikus olevad tasemed on välja toodud Joonisel 5. Joonisel on tasemed välja toodud stabiilsuse seisukohast, mis näitab, kui tihti tuleb antud tasemes muutusi teha rakenduse muudatustest tulenevalt.




*Joonis 5. Võtmesõnadel põhineva testimise raamistikus olevad kihid ning nende stabiilsus [20]*

Igas raamistikus on kõige aluseks platvorm, nt Robot Framework, mis annab vahendid metoodika kasutamiseks ning testide loomiseks ja käivitamiseks. See tase on kõige stabiilsem ning ei vaja muudatusi tulenevalt rakenduse muudatustest.

Iga platvorm kasutab rakendusega ühenduseks erinevaid teeke. Need püsivad samuti enamasti muutumatuna ning teekides tehakse harva muudatusi.

Kõige rohkem muutub erinevate ressursside kiht. Selles kihis on näiteks defineeritud, kus mõni nupp või meetod asub ning millised on tema parameetrid. Need on kõige tihedamini muutuvad rakenduse osad ning seetõttu see kiht on kõige vähem stabiilsem.

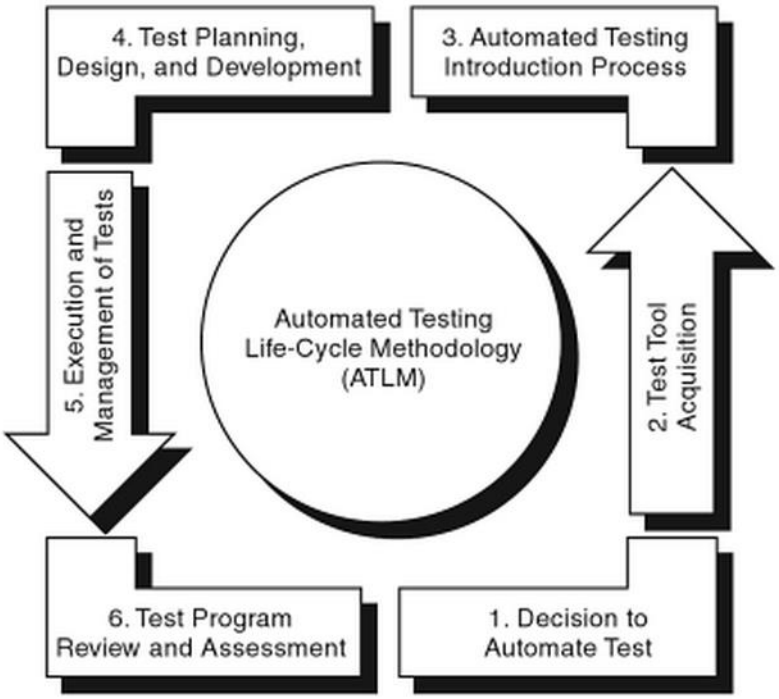
Järgmisena tulevad testjuhtumid, mida tuleb mõnikord muuta, kuid need muudatused peaks toimuma ainult siis, kui äriloogika muutub. Testjuhtumite kirjeldusse ei tohiks panna midagi kergesti muutuvat (nagu näiteks nuppude asukoht HTML kihis). Kasutajaliidese abil testimises nimetatud raamistikega kasutatakse *Page Object'*e [21], mis aitavad hoida kasutajaliidese teste hooldatavana.

Viimane tase on testide komplekt, mis võimaldab erinevaid teste grupeerida. Enamasti grupeeritakse näiteks erinevate funktsionaalsuste või äriliste tegevuste järgi (näiteks taotluste loomised). See tase püsib üsna muutumatuna, muutused võivad olla tingitud äriliste vajaduste muutumisest.

Kõige olulisem on silmas pidada, et raamistik tuleks üles seada nii, et käivitamise kihis tuleks peale esialgset seadistamist võimalikult vähe midagi uut kirjutada või seni kirjutatut parandada.

## 4.3 Sobiliku raamistiku valimine

Projektile sobiliku automaattestimise raamistiku valimine on üks tähtsamaid edasisi samme, kuna raamistikku vahetada on keeruline. Sobiliku raamistiku valimisel võiks kasutada näiteks ATLM (*Automated Test Life-Cycle Methodology*) mudelit (Joonis 6).




*Joonis 6. Automated Test Life-Cycle Methodology (ATLM) [22]*

Antud mudel sisaldab endas vajalikke samme edukaks automaattestimise tööriista kasutuselevõtuks. Raamistiku reaalse kasutuselevõtu korral tuleks lugeda meetodi kohta lähemalt joonisel viidatud materjalist, kus on hulgaliselt materjali iga sammu teostamiseks.

Meetodil on kuus suuremat sammu, mis jagunevad järgmiselt:

1. Otsus automatiseerida teste – Esimeses sammus defineeritakse ootused automaattestidele ning potentsiaalsed kasud. Samuti pannakse kokku ettepanek juhtkonnale automatiseerimise kasutuselevõtuks.
2. Testvahendi valimine ning hankimine – Selles sammus valitakse lähtuvalt oma eesmärkidest ning ootustest sobiv vahend. Siin tuleks arvestada nii rakenduse tehnoloogilisi valikuid kui ka kõiki organisatoorseid piiranguid ning võimalusi. Samuti tuleks selles sammus hinnata igat tööriista eraldi, kasutades meetodi [22].

3. Automaattestimise tutvustamine projekti meeskonnale – Kolmandas sammus toimub valitud automaattestimise lahenduse tutvustamine meeskonnale ning kasutusele võtmine projektis. Kasutusele võtmine jaguneb omakorda kaheks suuremaks osaks; esiteks testprotsessi analüüs ning seejärel testvahendi kaalumine.
    a. Testprotsessi analüüsitakse ning tutvustatakse meeskonnale, võttes arvesse erinevaid nõudeid, keskkondi, inimressurssi, kasutatavaid tehnoloogiaid ja rakenduse funktsionaalsusi. Läbi selle selgitatakse välja, kas kõik meeskonnale vajalik on olemas ning samuti tekib meeskonnal ühtne arusaam protsessist [22].
    b. Testvahendi kaalumisel analüüsitakse, kas kõik testprotsessi analüüsi käigus arvestatud kriteeriumid saavad vahendiga täidetud, kas projektis on piisavalt aega, et testvahendit kasutusele võtta, arvestades vahendi esialgse installeerimise ning seadistamise aega [22].
4. Testimise planeerimine, disain ning arendamine.
    a. Testide planeerimise osas tuleb välja mõelda pikemaajaline kava testide teostamiseks ning samuti defineerida testide standardid ning loomise juhendid. Selles sammus kasutatakse kõikide eelnevate sammude tulemusi, et välja selgitada erinevad tegemist vajavad ülesanded, määrata inimestele kohustused ning kuidas igale ülesandele lähenetakse (näiteks kui põhjalikult igat testimise tüüpi teostatakse). Samuti planeeritakse ning seadistatakse selles faasis ka testkeskkond [22].
    b. Disaini etapis mõeldakse põhjalikumalt, kui süviti testimine toimub, arvestades projekti ajapiiranguid ning kuidas igale testimise viisile lähenetakse (näiteks funktsionaaltestimise korral fikseeritakse millised harud ja funktsionaalsused kaetakse). Siinkohal tuleks defineerida testide koostamiseks standardid, mida meeskond saaks järgida, kuid samas olla valmis neid kohendama vastavalt vajadusele.
    c. Testide arendamise osas tuleb defineerida standardid automaattestimise vahendisse testide kirjutamiseks, kindlustamaks, et hiljem oleks testid hooldatavad ning arusaadavad [22].
5. Testide käivitamine ning haldamine. Peale testide planeerimist, disaini ning arendamist, tuleb koostatud testid valitud raamistikus käivitada vastavalt koostatud plaanile ning hinnata testide tulemusi. Siinkohal märgitakse ka leitud vead üles ning edastatakse programmeerijatele läbi kokkulepitud kanalite ning formaadi. Iga arendustsükli lõpus

tuleks kogu koostatud testkogumik võimalusel alati uuesti läbi jooksutada kindlustamaks, et tehtud parandused pole vahepeal midagi ootamatult katki teinud [22].

6. Testimisprotsessi ülevaade ning hindamine. Kuigi see on ära toodud viimase sammuna, tuleb protsessi ülevaatust ning hindamist teostada kogu arenduse vältel kindlustamaks, et vajalikke parandusi tehakse pidevalt. Protsessi alguses tuleb defineerida ka erinevad meetrikad, mida hakatakse jälgima vaatamaks, kas koostatud automaattestimise lahendus saavutab oodatud tulemuse. Arenduse vältel tuleks need meetrikad mingi regulaarse ajavahemiku tagant fikseerida ning lõpuks nende põhjal koostada lõplik hinnang lahenduse edukusele.

Soovitatud on ka, et esimestes sammudes lisaks tööriistade valikule keskendumisele üritada juba mõned testid valmis disainida. See annab parema arusaamise rakenduse vajadustest ning formaadist, kuidas oleks vajalik teste luua. Samuti on oluline üritada kavandatud teste valitud raamistikes luua. Tänu sellele saab raamistikku kergemini ning targemalt valida ja vältida hilisemaid üllatusi ning pettumusi.

# 5. FIT/FitNesse

## 5.1 FIT ning FitNesse üldine kirjeldus

**FIT** (Framework for Integrated Tests) on võimas automaattestimise raamistik, mille eesmärk on peamiselt lahendada järgnevaid probleeme [23]:

- Süsteemi tellijad ei mõista piisavalt hästi ärilisi vajadusi (sest ei ole selget viisi nende peale mõelda ning neid väljendada).
- Süsteemi arendaja ei mõista piisavalt hästi ärilisi vajadusi ja seetõttu süsteem ei lahenda õiget probleemi.
- Süsteem on madala kvaliteediga, seda on raske mõista ja sisaldab tihti vigu, millele kasutajatel tuleb tihti alternatiivseid lahendusi leida. Uuendustesse suhtutakse kahtlevalt, kuna need toovad tihti rohkem uusi probleeme kui kasu.
- Mida aeg edasi, seda raskem on süsteemi muuta ning see muutub hapraks. Arendajad hakkavad kartma muudatusi teha ning hakkavad pigem koodi juurde lisama kui vana koodi üle vaatama.

Raamistiku eesmärgiks on aidata luua teste, mis näitavad selgelt kõigile osapooltele, et ärilised eesmärgid on projektiga saavutatud. Seega FitNesse on mõeldud just äripoole testide kirjutamiseks. Äripoole test on näiteks test, mis kontrollib juhtu, kus kasutaja lisab taotluse ning seejärel esitab selle. Ehk selle asemel, et testida taotluse lisamist ning esitamist eraldi, mõeldakse välja suurem kasutaja tegevuste protsess kliendi ärilistest vajadustest lähtuvalt.

Nagu teistelegi võtmesõnadel põhinevatele testimise raamistikele kohane, kasutab ka FIT tabulaarset sisendi vormi, mis lubab inimestel, kel pole programmeerimise tausta, kiiresti ning lihtsalt uusi teste luua (eeldades, et ta on teadlik olemasolevatest võtmesõnadest). See aitab täita FIT kõige suuremat eesmärki, kus nii arendajad kui ka klient saavad teste kirjutada ning koos üle vaadata ja läbi selle luua ühtne pilt projekti vajadustest.

**FitNesse** on Wiki veebiserver, mis loob mugava võimaluse FIT testide loomiseks, käivitamiseks, organiseerimiseks ning hooldamiseks. FitNesse abil saab projektis seadistada ühise veebiserveri, kus kõik saavad mugavalt veebilehitseja kaudu teste käsitleda ning hoida end kursis nende arenguga. Põhimõtteliselt on FitNesse FIT raamistikule mõeldud kasutajaliides [14].

Nii FIT kui ka FitNesse on mõlemad vabavaralised tarkvarad, mille lähtekoodi ei oma ükski firma või asutus. Tänu sellele on kogunenud raamistikule ka suur kasutajate kommuun, kes kirjutavad raamistikule erinevaid uuendusi ning täiendusi ja tänu sellele toetavad need tarkvarad palju erinevaid tehnoloogiaid ning programmeerimiskeeli.

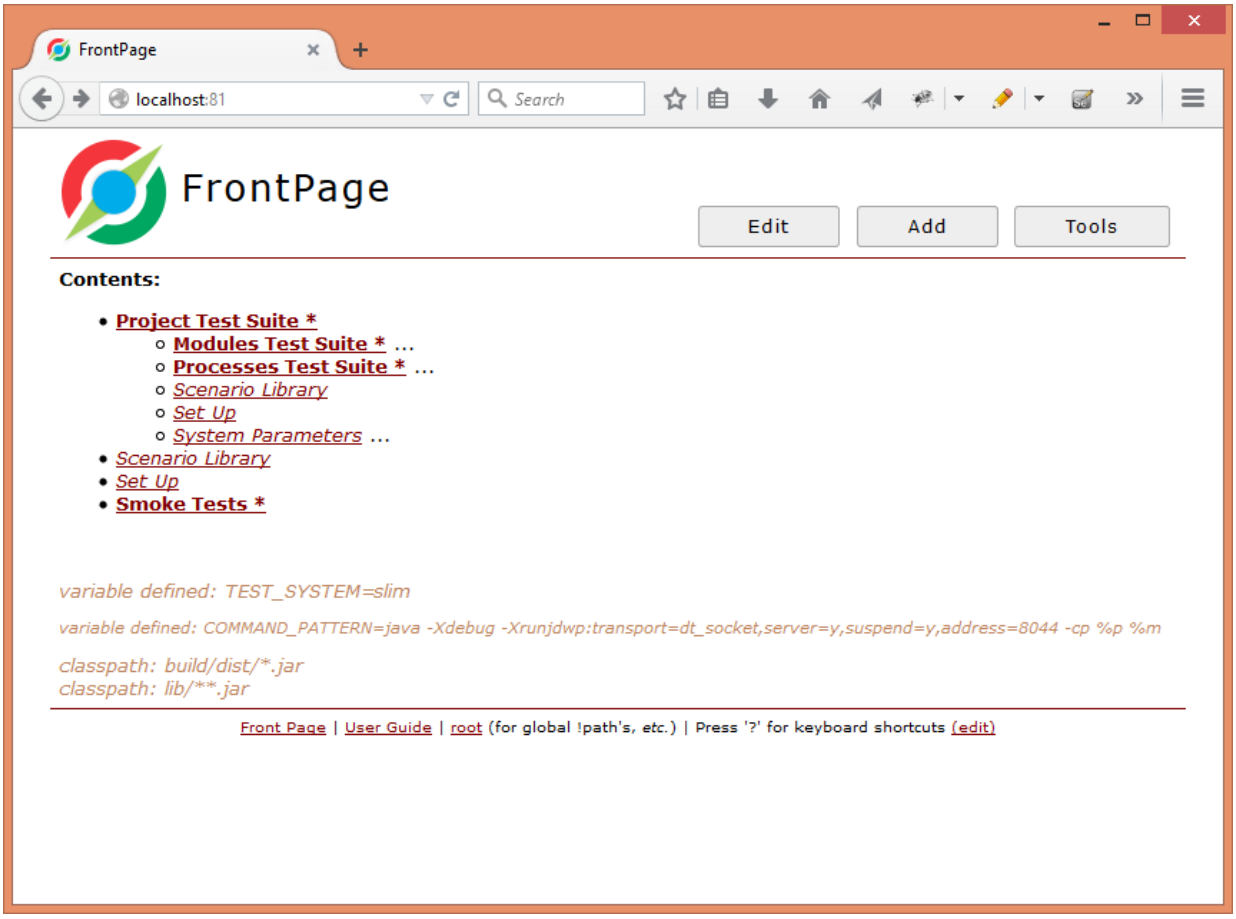
Näiteks FIT poolt toetatavate programmeerimiskeelte alla kuuluvad [23]:

- C# ja .Net platvorm
- C++
- Delphi
- Lisp
- Objective C
- Perl
- Python
- Ruby
- Smalltalk

## 5.2 Näide raamistiku kasutamisest reaalse tarkvaraprojekti näitel

### 5.2.1 FitNesse kasutajaliides ning põhilised funktsionaalsused

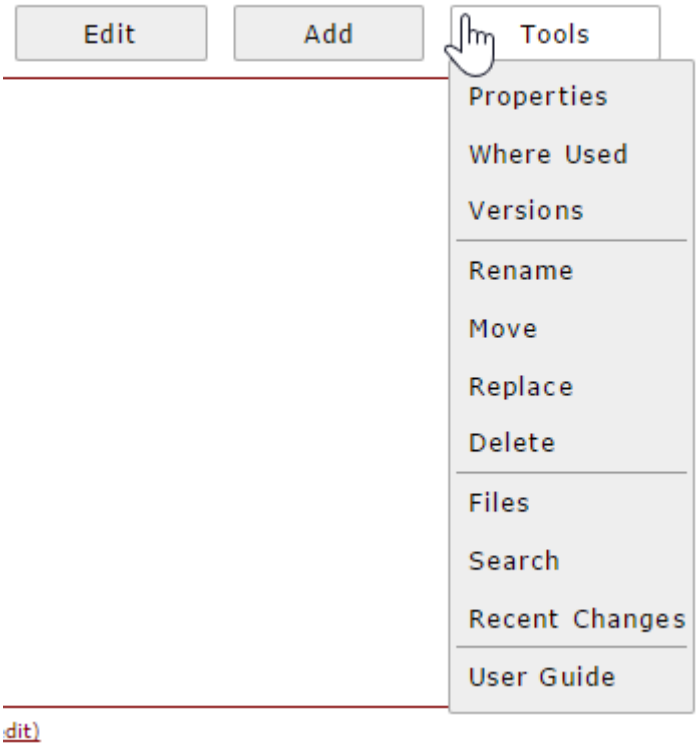
Joonisel 7 on esitatud FitNesse keskkonnas loodud projekti esileht, kus on näha testkogumiku hierarhia kaks esimest taset ning tähtsamad nupud. *Tools* nupu all on erinevad kergesti kasutatavad funktsionaalsused testlehtedega toimetamiseks. Uue lehe loomiseks on nupp *Add*, mille alt saab lisada erinevaid lehe tüüpe ja *Edit* nupp on olemasoleva lehe muutmiseks. Raamistiku kirjelduses kasutatavad näited pärinevad reaalsest Eesti tarkvaraprojektist finantsasutusele.




*Joonis 7. FitNesse raamistiku esileht reaalsest Eesti tarkvaraprojektist*

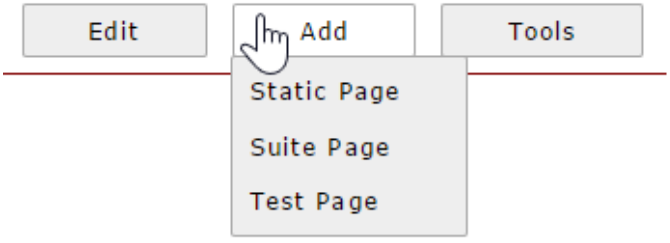
Joonisel 8 on näha peamised funktsionaalsused, mida FitNesse lubab testlehtedega toimetamisel. *Properties* võimaldab ligi pääseda lehe seadistuste juurde, kus saab muuta lehe tüüpi või kuvatavaid objekte ning muid lehe omadusi. *Where Used* on funktsionaalsus leidmaks kohad, kus antud lehte kasutatakse. Edasi järgnevad funktsionaalsused lehe eri versioonide nägemiseks ning lehe ümbernimetamiseks, liigutamiseks, asendamiseks ning kustutamiseks.

Samuti väärib mainimist otsimise funktsionaalsus, mille kaudu saab testlehti ning võtmesõnu otsida.




*Joonis 8.* Tools *nupu kaudu ligipääsetavad funktsionaalsused*

Joonisel 9 on näha erinevaid lehe tüüpe, mida FitNesse liideses on võimalik kasutada. *Static Page* lehe tüüp on tavaline Wiki lehekülg, mida ei saa testi(de)na käivitada. See on peamiselt koht, kuhu saab panna kasutajale abistavat materjali, mis testide jooksutamisel ei tohiks käivituda. *Suite Page* võimaldab konkreetset testide kogumikku käivitada. Kõik testi tüüpi lehed, mis asuvad selle lehe all, käivitatakse. *Test Page* on tavaline testlehe tüüp, kuhu pannakse üksikud testjuhtumid.




*Joonis 9. Erinevad lehe tüübid FitNesse'is*

### 5.2.2 Testide loomine

Testlehtede loomine ning muutmine FitNesse liideses toimub *Wiki* [24] tüüpi lehtedele omase süntaksiga. Uue testjuhtumi kirjapanekuks tuleb kõigepealt luua *Test Page* tüüpi leht ning minna tema muutmise vaatesse (kasutades *Edit* nuppu).

Võtame nüüd konkreetse testjuhu, kus soovitakse kontrollida, et siseriikliku makse korral peab valuuta olema EUR. Selleks tuleks koostada vastav test ning kontrollida vastussõnumit ja andmebaasi õiget seisu. Selline test on FitNesse liideses kirjutatuna järgmine (Joonis 10):

```
!|script|
|Send domestic payment request xml and receive response  with response code |RJCT                    |
|Validate response to client message reason code is                          |AM11                    |
|Validate response to client message error code  is                          |paymPaymentCurrencyErr  |
|Validate response to client message reason definition is                    |Invalid payment currency|
|Get order data of the saved order and validate data saved to database                                |
```

*Joonis 10. FitNesse liideses kirjutatud testi script kontrollimaks, et makse valuuta oleks EUR*

Jooniselt on näha, et uue testi skripti alustamiseks piisab *!|script|* kirjutamisest esimeseks reaks ning seejärel saab selle alla hakata võtmesõnu ning nende väärtusi kirjutama. Tabeli formaati aitab sümboliseerida *Wiki* keskkonnas püstkriips, mis eristab tulpasid ning iga uue rea tekst algatab tabelis uue rea. Uue testi skripti alustamiseks piisab eelmise lõppu tühja rea jätmisest. Lisades sellele testile juurde ka kirjelduse, pealkirja ja mõned parameetrid, näeb see välja muutmise vaates, nagu on kujutatud Joonisel 11.

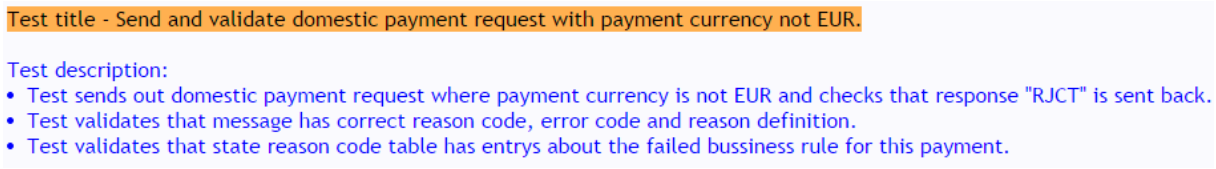
```
!define NotValidCurrency (USD)
!define domesticPaymentPaymentCurrencyNotEur (<?xml version=.....</Document>)

!style_test_title[Test title - Send and validate domestic payment request with payment currency not EUR.]

!style_blue[Test description:
 * Test sends out domestic payment request where payment currency is not EUR and validates correct response code.
 * Test validates that response message has correct reason code, error code and reason definition.
 * Test validates that state reason code table in database has entrys about failed bussiness rule for this payment.
]

-!|script|Echo fixture                                |
|$msg=|echo|${domesticPaymentPaymentCurrencyNotEur }  |

!|script|
|Send domestic payment request xml and receive response with response code |RJCT                    |
|Validate response to client message reason code is                        |AM11                    |
|Validate response to client message error code  is                        |paymPaymentCurrencyErr  |
|Validate response to client message reason definition is                  |Invalid payment currency|
|Get order data of the saved order and validate data saved to database                              |
```

*Joonis 11. Kogu testjuhtum muutmise vaates*

Kasutades testides parameetreid, tuleb eelnevalt need defineerida testi *Wiki* leheküljel või sellele lehele kaasatud lehekülgedel (näiteks *ScenarioLibrary* leheküljel). Parameetri loomiseks

tuleb kirjutada algusesse käsk *!define*, millele järgneb parameetri nimi peale tühikut ning seejärel lisatakse sulgudesse parameetri väärtus. Näitena võib Jooniselt 11 näha, et on määratud vale valuuta väärtus parameetris *NotValidCurrency* ning seejärel parameeter, kus on defineeritud saadetav XML fail, mis omakorda kasutab esimest parameetrit (parameetrile viidatakse *Wiki* lehtedel kasutades süntaksit *${parameetri nimi}*). Ruumi kokkuhoiu eesmärgil XML (*Extensible Markup Language*) [25] väärtust käesolevas töös ei ole välja toodud. Skripti sees parameetrile väärtuse andmine toimub süntaksiga *|$parameeter=|väärtus|* (nagu oli Joonisel 11 antud *msg* parameetrile väärtus).

Pealkirjade ning kirjelduste paremaks vormistamiseks saab luua eri stiilid FitNesse CSS (*Cascading Style Sheets*) [26] failides. Antud näites on loodud pealkirja jaoks stiil *test_title*, mida saab välja kutsuda kasutades *!style_test_title[pealkiri]* süntaksit. Samuti on kirjeldus paremaks lugemiseks siniseks värvitud, kasutades samuti CSS failis defineeritud stiili muutmist. Et ühel testlehel olevaid erinevaid testjuhte kerimisel ning testide jooksutamisel selgemalt eristada, tuleb vajadusel muuta testi pealkirja ja kirjeldust paremini nähtavaks. Joonisel 12 on näha näiteprojektis tehtud lahenduse tulemust testlehel.




*Joonis 12. Muudetud testi pealkirja ning kirjelduse stiilid*

### 5.2.3 Võtmesõnade defineerimine

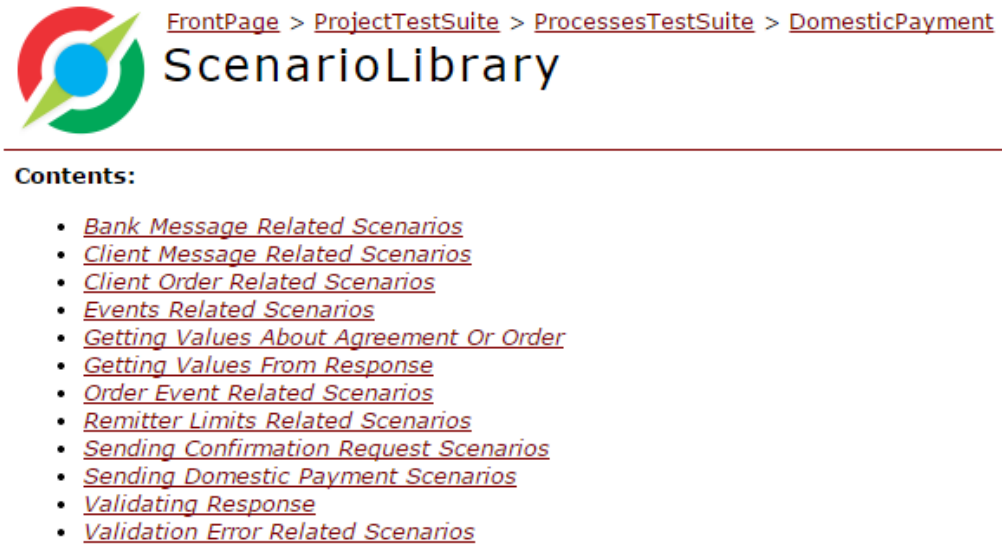
Testis kasutatavad võtmesõnad on defineeritud eraldi *ScenarioLibrary* lehtedel, mille võtmesõnu saavad kasutada kõik testid, mis asuvad kas samal või madalamal tasemel kui loodud leht. See toimub automaatselt ainult juhul, kui lehe nimi on *ScenarioLibrary*. Muudel juhtudel tuleb käsitsi lisada igale lehele soovitud stsenaariumite kogumike kasutamine kasutades süntaksit *!include lehe nimi*. Joonisel 13 esimene rida |*Send domestic payment request xml and receive response  with response code* |*RJCT*| võtmesõna on defineeritud oma *ScenarioLibrary* lehel järgmiselt:

```
!|scenario      |Send domestic payment request xml and receive confirmation request and response message with state code|expectedState|
|addMessage     |$msg                                                                                                                |
|$sentMsgJmsId= |sendCommModule                                                                                                      |
|$ack=          |getMessageBody                                |ACK                                                                  |
|isXml          |$ack                                                                                                                |
|$confReq=      |getMessageBodyByXpath|/ConfirmationRequest/OrderConfirmation/UserCode/text()[.='${RemitterUserCode}']                |
|isXml          |$confReq                                                                                                            |
|$statusMsg1=   |getMessageBodyByXpath|/Document/CstmrPmtStsRpt/OrgnlPmtInfAndSts/TxInfAndSts/TxSts/text()[.='@expectedState']       |
|isXml          |$statusMsg1                                                                                                         |
```

*Joonis 13. Võtmesõna defineerimine*

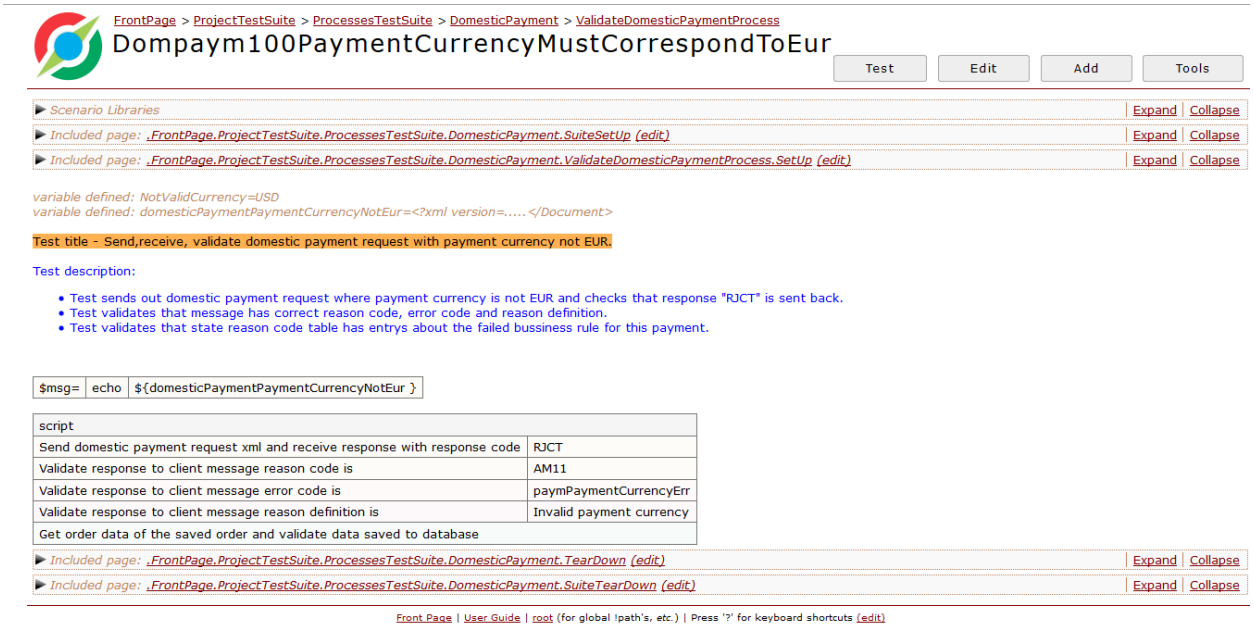
Esimene rida määrab võtmesõna pealkirja ja fakti, et see on võtmesõna defineeriv stsenaarium. Edasi on kasutatud meetodeid nagu *addMessage*, mis viitavad käivitamise kihis Java meetoditele ning ka neid saab parameetritesse panna hilisemaks kasutamiseks (Joonis 8).

Võtmesõnu tuleks üritada võimalikult palju grupeerida, sest samal lehel paljude võtmesõnade defineerimisel võib haldamine muutuda keeruliseks. Võtmesõnu saab FitNesse abil grupeerida luues *Static Page* tüüpi lehe *ScenarioLibrary* alla ning seejärel kaasata tavatüüpi lehed kasutades käsku *!include lehe nimi*. Selle tulemusel on kõik need võtmesõnad kasutatavad testides, mis on kaasatud *ScenarioLibrary* alla. Näide grupeeritud võtmesõnadest on esitatud Joonisel 14.




*Joonis 14. Grupeeritud võtmesõnad*

### 5.2.4 Testide käivitamine




*Joonis 15. Valminud testi vaade peale muutmist*

Eelnevalt kirjeldatud muudatuste salvestamise tulemust FitNesse liideses on esitatud Joonisel 15.

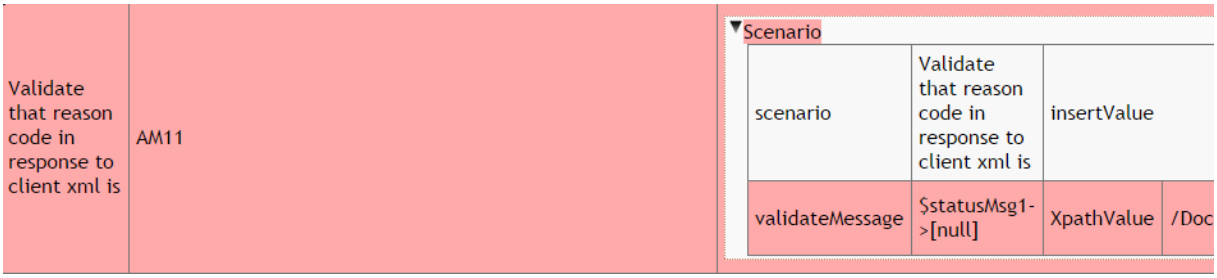
Joonisel 15 kujutatud lehel on näha, et lehe testlehele on kaasatud (*Included page*) *SetUp* ning *TearDown* nimelised lehed, mis on osa FitNesse sisse kirjeldatud funktsionaalsusest. Kui testi lehega asub samal tasemel *SetUp* nimega leht, siis enne iga testi käivitamist teostatakse sellel lehel defineeritud tegevused (näiteks vajaliku kasutajate seisu defineerimine andmebaasis). *TearDown* lehel defineeritud tegevused teostatakse iga testi järel (näiteks andmebaasist testi jooksul tekitatud andmete eemaldamine).

Testi saab käivitada vajutades nuppude menüüs olevat *Test* nuppu, mille tulemusel kõik testlehel ning kaasatud testlehtedel olevad võtmesõnad käivitatakse.

Käivitades kuvatakse kõigepealt raport, kus on näha palju teste õnnestus ning kui palju ebaõnnestus. Samuti näidatakse testide jaoks kulunud aeg. See osutub kasulikuks, kui on soovi testkogumiku jooksutamise kiirust parandada. Testlehe raport on esitatud Joonisel 16.


Joonis 16. Testlehe käivitamise raport

Testide käivitamisel käivitatakse iga võtmesõna eraldi ning iga rea taust muutub punaseks või roheliseks, olenevalt selles reas tehtud toimingute tulemusest. Vaadates päringule vastuseks tulnud veakoodi või vastuse sõnumit, saab iga rea korral lihtsalt uurida ka vea täpsemat põhjust. Seda saab teostada, minnes võtmesõna realt sügavamale loogikasse sisse, mida FitNesse võimaldab otse *Wiki* keskkonnas teha. Joonisel 16 on esitatud võtmesõnade eri tasemete vaade testi ebaõnnestumise korral.




*Joonis 17. Võtmesõnade eri tasemete vaatamine ebaõnnestumise korral*

Soovides käivitada suuremal hulgal teste korraga, tuleb luua *Suite* tüüpi leht ning liigutada *Test* tüüpi lehed *Suite* lehe alla või luues selle alla uued soovitud testlehed.

# 6. Robot Framework

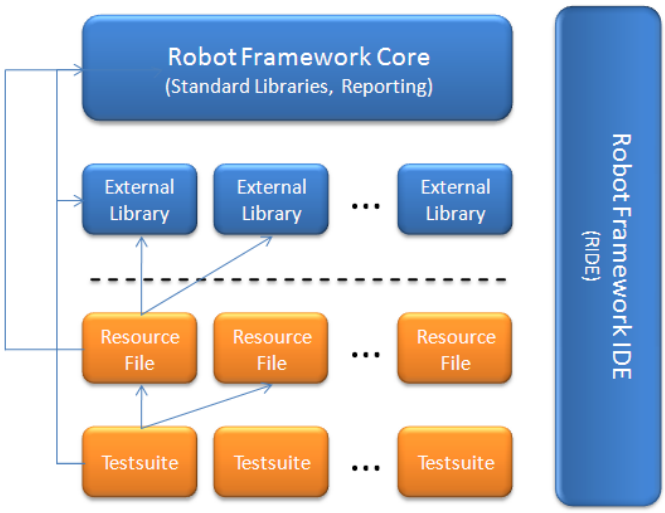
## 6.1 Robot Framework ning tema arhitektuur

Robot Framework on üldine automaattestimise raamistik vastuvõtu testimiseks (*acceptance testing*) ning vastuvõtu testimisel põhineva arenduse (*acceptance test driven development*) jaoks. Raamistikul on kergesti kasutatav tabulaarses formaadis testandmete sisestamise võimalus ja ta toetab võtmesõnadel põhineva testimise lähenemist. Raamistiku testimise võimalusi saab laiendada kasutades Python'i või Java teeke [13].

Raamistiku peamisteks tunnusteks on:

- Mugav ja kergesti kasutatav tabulaarne süntaks
- Ei sõltu kasutatavast platvormist ning rakenduses kasutavatest tehnoloogiatest
- Detailsed logid
- Andmetel põhinevad testid
- Mugav testide loomise kasutajaliides (RIDE)
- Selgelt ja kiirelt arusaadavad testraportid
- Lokaalsest masinast eemal asuvate teekide kasutamise võimalus

Raamistik ning enamus sellele loodud teeke ning tööriistu on vabavaralised. Raamistikul on suur kasutajaskond üle maailma ning see on aktiivselt kasutusel ka Eesti testijate kommuuni seas. Tänu sellele on abimaterjali raamistiku kohta hulgaliselt ning tekkivatele küsimustele saab raamistikule pühendatud lehekülgedelt kiiresti vastuseid.




*Joonis 18. Raamistiku ülesehitus [13]*

Joonisel 18 on esitatud raamistiku arhitektuur. Kõige aluseks on raamistiku tuum, kus asub põhiloogika, mis loob võimalused kasutada raamistikule omaseid funktsionaalsusi. Seal asuvad näiteks raamistikuga standardina kaasas olevad teegid ning raportite genereerimise vahendid.

Raamistiku tuumaga saab ühendada erinevaid väliseid teeke, olenevalt testitava raamistiku vajadustest ning temas kasutatavatest tehnoloogiatest. Järgmisena kasutatakse raamistikes eri tüüpi ressursifaile, kus defineeritakse peamiselt võtmesõnu ja parameetreid. Need failid kasutavad raamistiku tuumas olevaid standardteeke ja väliseid teeke.

Enamus Joonisel 18 kujutatud kihte aitab mugavalt struktureerida ning luua raamistikuga enim kasutatav kasutajaliides RIDE, mis tuleb eraldi alla laadida. RIDE pakub mitmeid võimalusi, mida paljud raamistikud ei toeta. See on üks peamistest põhjustest, miks raamistikku on saatnud niivõrd suur edu. Lähemalt saab mainitud liidesest lugeda järgmistes paragrahvides.

Kõige nähtavamaks osaks raamistiku juures on testkogumikud. Need koosnevad testidest, mis on koostatud varem loodud võtmesõnadest ning parameetritest. Neid teste saavad lõpuks lugeda kõik arendusmeeskonna liikmed ning ka klient.

Läbi kirjeldatud jaotuse on selgelt näha võtmesõnadel põhineva testimise metoodika kasulikkus. Kogu tehniline keerukus on peidetud testkogumike taha, kus sügavamale minnes keerukus kasvab kihi kaupa. Tänu kihilisele keerukuse kasvule saab väikeste tehniliste teadmistega testija lisaks testide kirjutamisele ka võtmesõnu kirjutada.

## 6.2 Robot Framework raamistiku kasutajaliides RIDE

RIDE on Robot Framework'i jaoks loodavate automaattestide arendamise keskkond. See on hea vahend nii kogenud automaattestide loojale kui ka sellega alustajale. RIDE edu juures on tähtsat rolli mänginud järgmised omadused:

1. See on vabavaraline.
2. Väga lihtne kasutajaliides liigsete funktsionaalsusteta.
3. Võimaldab mugavalt kirjutada tabulaarses formaadis teste.
4. Pakub olulisi võimalusi, mida teiste tuntud raamistike kasutajaliidesed ei võimalda.

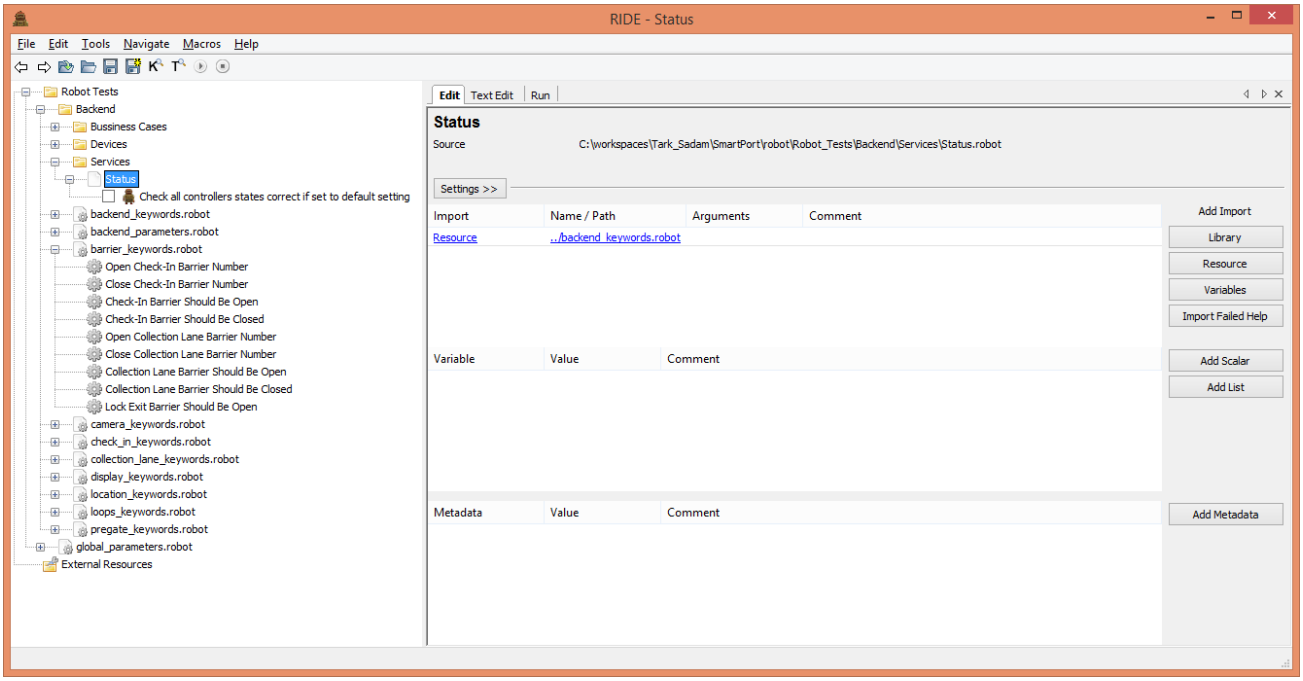
Kõige suuremaks eeliseks teiste raamistike vahendite ees on see, et see oskab pakkuda võtmesõnade sisestamisel võimalikke võtmesõnu juba eelnevalt kirjeldatud võtmesõnade seast. Kuna rakenduses tekib tihtipeale arenduse käigus hulgaliselt võtmesõnu, on keeruline neid kõike meeles hoida ja RIDE pakub sellele väga mugava lahenduse.

Lisaks võtmesõnadele oskab RIDE pakkuda võtmesõnade kirjeldamisel ka käivitamise kihis kirjutatud meetodeid samal viisil nagu võtmesõnu. See teeb testide kirjutamise palju efektiivsemaks, kuna ei pea aega kulutama kontrollimaks, kas kirjutatud võtmesõna või meetod reaalselt eksisteerib.

Väga oluline RIDE eelis teiste sarnaste vahendite ees on testide jooksutamise järel pakutavad võimalused. Peale testide kirjutamist ning nende käivitamist pakub RIDE võimaluse vaadata väga detailseid Robot Framework'i poolt koostatuid testide jooksutamise raporteid ning logisid. Need on HTML formaadis ning võimaldavad saada kiiret ülevaadet testide tulemustest. Nende detailsust saab konfigureerida ning läbi selle nad teha sobivaks eri vajadustega isikute jaoks. Näiteks kliendile edastada ühe detailsuse astmega ning testkogumikku arendada teisega. Loodavad logid aitavad vigade korral kiiresti probleemne koht üles leida ning vearaport koostada.

## 6.3 RIDE kasutajaliides

Ekraanitõmmis kasutajaliidesest RIDE on esitatud Joonisel 19. Erinevad testkogumiku tasemed ning struktuur on vahendis kujutatud operatsioonisüsteemide kataloogipuudele sarnaselt ning nendes liikumine on seetõttu mugav ning intuitiivne.




*Joonis 19. RIDE kasutajaliides*

Testide struktureerimiseks on peamiselt võimalik luua viit liiki objekte:

1. Parameeter
2. Võtmesõna
3. Test
4. Testkogumik
5. Kataloog

Võtmesõnades on defineeritud tegevused, mida teostatakse kindlat võtmesõna välja kutsudes. Nende defineerimisel saab kasutada nii teisi võtmesõnu kui ka käivitamise kihis defineeritud programmeerimiskeele meetodeid. Nende defineerimisel tuleb olla ettevaatlik, et ei tekiks tsüklilist väljakutse ahelat.

Testide loomisel kasutatakse defineeritud võtmesõnu ning parameetreid. Nende koostamisel tuleks lähtuda põhimõttest, et neid tuleb hiljem pidevalt hooldada ja seetõttu üritada võimalikult

palju muutuvat loogikat kirjeldada võtmesõnade kihis, kus saab kõigi testide jaoks ühest kohast muudatusi teha.

Testkogumikud tuleks luua vastavalt eelnevalt väljamõeldud struktuuriloogikale. Näiteks grupeerida äriliselt sarnaste eesmärkidega või sama funktsionaalsuse testjuhtumid. Testkogumike loomisel arvestada, et rakendus kasvab kiiresti, kuid samas enamasti ettearvatavalt ning leida selline lahendus, et see oleks tuleviku töid arvestades eskaleeruv.

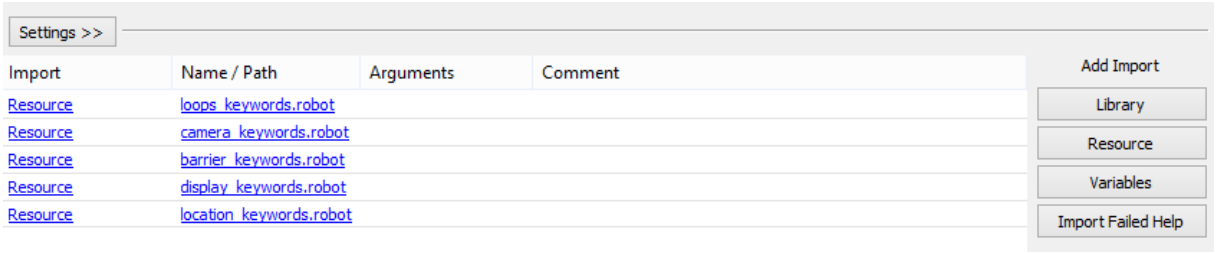
Kataloogid võimaldavad erinevaid testkogumikke struktureerida vastavalt rakenduse ning planeeritava struktureerimise loogikale. Ka siin tuleks eskaleerumise võimaldamisest kinni pidada.


*Joonis 20. Testkogumike ning kataloogide võimalused*

Joonisel 20 on näha lisavõimalusi, mida pakuvad nii kataloogid kui ka testkogumikud. Loodavates objektides saab kirjutada ka dokumentatsiooni. Selle jaoks on eraldi lahter RIDE liideses ning on soovitatav oma objektide juures seletada vähemalt nende eesmärk ning sisu. Testkogumik ning kataloogid pakuvad võimalust enne ja peale iga testkogumiku käivitamist välja kutsuda mingid võtmesõnad. See on kasulik näiteks juhul, kui enne testkogumikku on vaja kõigile kogumikus asuvatele testidele luua ühised testandmed ning hiljem need eemaldada.

Samuti on võimalus sama loogikaga enne ja pärast igat testi võtmesõnu välja kutsuda. See võib kasuks tulla siis, kui on vaja iga testi alguses rakenduse seis viia algseisu.

Samuti lubab RIDE oma objekte märksõnadega märgistada. Tänu nendele saab näiteks käivitada ainult kindlat märksõna sisaldavad testid või neid otsida märksõnade alusel.




*Joonis 21. Teekide, parameetrite ning ressursside lisamine*

Joonisel 21 on näha kuidas saab võtmesõnadele, testkogumikele ning kataloogidele lisada objekte mujalt. Võtmesõnade defineerimisel saab käivitamise kihilt meetodeid kasutada importides *Library* nupuga soovitud meetodi fail. Samal viisil saab importida võtmesõnu ning nende kogumikke, kasutades nuppu *Resource*.

Ei ole soovitatav alati kõiki meetodeid importida. Importides ainult vajaliku, on võimalik kiiresti aru saada, milliseid ressursse kasutavad teatud struktuuriüksused ning nende laadimisel ei laeta mitte vajalikke meetodeid.

Kirjeldatud funktsionaalsused on peamised, mis raamistiku kasutamise alguses vaja lähevad ning nende teadmistega saab juba hakata teste looma, struktureerima ning käivitamise kihiga ühendama. Tuleb arvestada, et enne alustamist tuleks välja mõelda planeeritava struktuuri plaan, et koostatud kogumikud oleksid hooldatavad mugavalt ning kiiresti. Samuti peaksid nad olema kõigile võimalikult kiiresti arusaadavad, et uus töötaja saaks neid kiiresti luua.

## 6.4 Testide loomine

Vaatamaks, kuidas toimub testide loomine Robot Framework raamistikus kasutades RIDE liidest, vaatame kõigepealt, kuidas luuakse võtmesõnu ning parameetreid ning seejärel kuidas neid saab kasutada testide kirjutamisel.

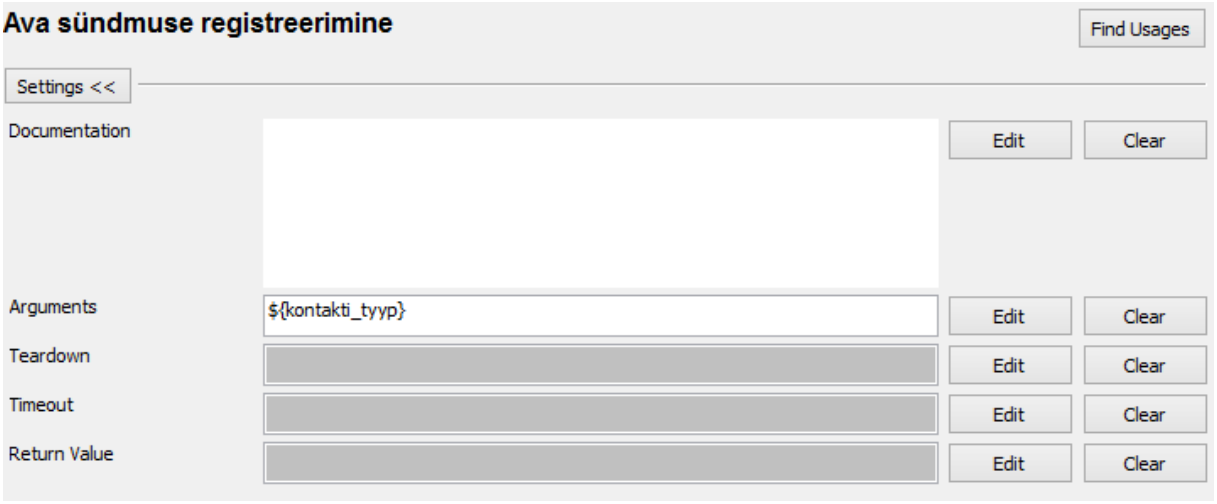
Enne testi käivitamist tuleb vajalikud võtmesõnad defineerida. See toimub RIDE liideses samuti tabulaarses formaadis nagu on kujutatud Joonisel 22. Kujutatud võtmesõnas on kirjeldatud tegevused, et avada rakenduses teostatud mingi sündmuse registreering.

| | | | | | |
|---|---|---|---|---|---|
| 1 | **Wait Until Page Contains** | Sündmuste otsimine | | | |
| 2 | **Nupp.On olemas** | Lisa uus sündmus | | | |
| 3 | **Nupp.Vajuta** | Lisa uus sündmus | | | |
| 4 | **Rippmenüü.Vali väärtus** | Kontakti registreerimine | ${kontakti_tyyp} | | |
| 5 | **Nupp.On olemas** | Vali kontakti registreerimine | | | |
| 6 | **Nupp.Vajuta** | Vali kontakti registreerimine | | | |

*Joonis 22. Võtmesõna kirjeldamine RIDE liideses*

Joonisel 22 oleva võtmesõna defineerimisel on kasutatud nii käivitamise kihis olevat programmeerimiskeeles kirjutatud meetodit kui ka teisi võtmesõnu. Tumesinisega on kujutatud teised võtmesõnad ning nende järel võtmesõna käivitamisele kaasa antavad parameetrid ning helesinisega programmeerimiskeeles tehtud meetodid ning neile kaasaantavad parameetrid.

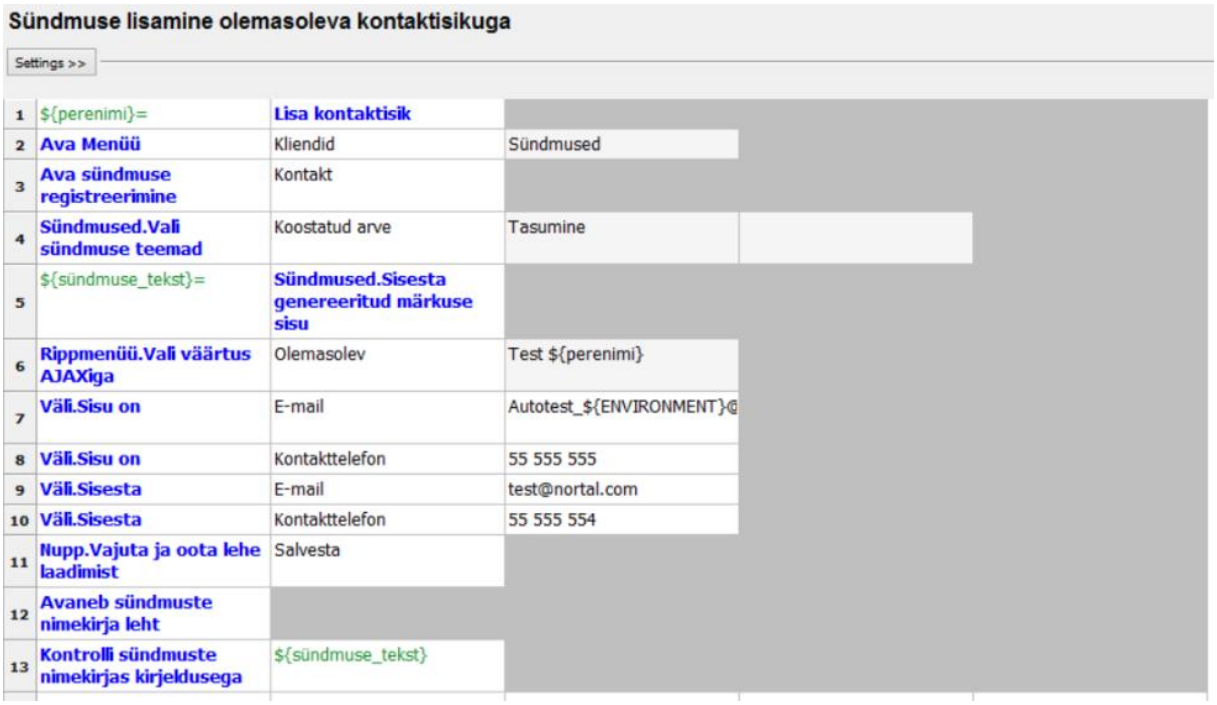
Vajutades võtmesõnale peale on võimalik liikuda selle definitsioonile ning näha iga meetodi ning võtmesõna kohta, kust see pärit on ning milliseid argumente ta nõuab.




*Joonis 23. Lisavõimalused võtmesõnade defineerimisel*

Joonisel 23 on näha võtmesõnade kirjeldamise juures olevad lisavõimalused. Kõigepealt saab lisada võtmesõna kirjeldava teksti, mis on nähtav igal pool, kus lisainfot nende kohta näidatakse. Kirjeldatavat teksti on soovitatav alati lisada. Järgmisena on võimalik määrata, milliseid argumente võtmesõna nõuab, kui teda välja kutsutakse (mitu argumenti saab lisada eraldades nad püstkriipsudega). Lisaks on võimalik iga võtmesõna puhul määrata tegevused, mida tuleb peale võtmesõna käivitamist teostada (määrata võtmesõna, mille see välja kutsub). Samuti on võimalik määrata võtmesõnale ajalimiit, mille jooksul tuleb oma tegevused lõpetada või võtmesõna ebaõnnestub. Viimasena saab lisada väärtuse, mille võtmesõna peale käivitamist väljastab.

Testide loomine toimub raamistikus samuti tabulaarses formaadis, nagu võtmesõnadel testimise metoodikat kasutavatele raamistikele kohane. Kirjeldatud võtmesõna võib näha kasutuses Joonisel 24. Joonisel on näha, kui hästi on test arusaadav tänu läbimõeldud võtmesõnadele. Testi esimesel real on näha, et kui võtmesõna väljastab parameetri, siis pannes esimesse lahtrisse parameetri koos võrdusmärgiga, on võimalik anda parameetrile väärtus, mis on testi edasistes sammudes kasutatav.



**Sündmuse lisamine olemasoleva kontaktisikuga**

Settings >>

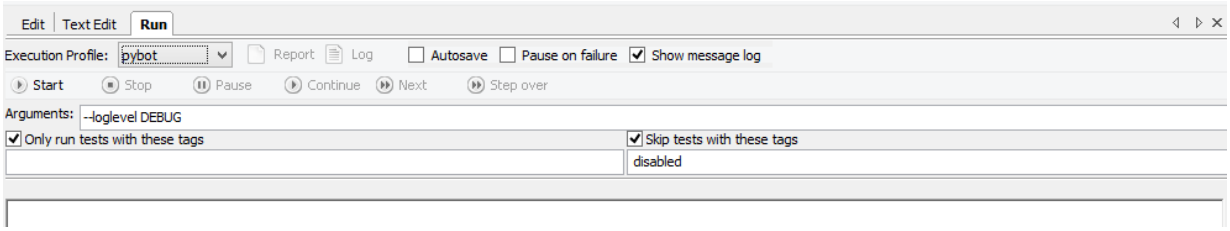
| | | | |
|---|---|---|---|
| 1 | ${perenimi}= | **Lisa kontaktisik** | |
| 2 | **Ava Menüü** | Kliendid | Sündmused |
| 3 | **Ava sündmuse registreerimine** | Kontakt | |
| 4 | **Sündmused.Vali sündmuse teemad** | Koostatud arve | Tasumine |
| 5 | ${sündmuse_tekst}= | **Sündmused.Sisesta genereeritud märkuse sisu** | |
| 6 | **Rippmenüü.Vali väärtus AJAXiga** | Olemasolev | Test ${perenimi} |
| 7 | **Väli.Sisu on** | E-mail | Autotest_${ENVIRONMENT}@ |
| 8 | **Väli.Sisu on** | Kontakttelefon | 55 555 555 |
| 9 | **Väli.Sisesta** | E-mail | test@nortal.com |
| 10 | **Väli.Sisesta** | Kontakttelefon | 55 555 554 |
| 11 | **Nupp.Vajuta ja oota lehe laadimist** | Salvesta | |
| 12 | **Avaneb sündmuste nimekirja leht** | | |
| 13 | **Kontrolli sündmuste nimekirjas kirjeldusega** | ${sündmuse_tekst} | |

*Joonis 24. Võtmesõna kasutav test*

Testi tasemel tuleks vältida tehniliste sisendite andmist. Näiteks, kui on vaja vajutada nuppu, siis see ei kirjelda testis nupu asukohta HTML struktuuris, vaid see tuleks kirjeldada võtmesõna sees. Sellest tulenevalt saab vähem universaalseid võtmesõnu luua, kuid nende hooldamine on tänu sellele kergem.

## 6.5 Testide käivitamine

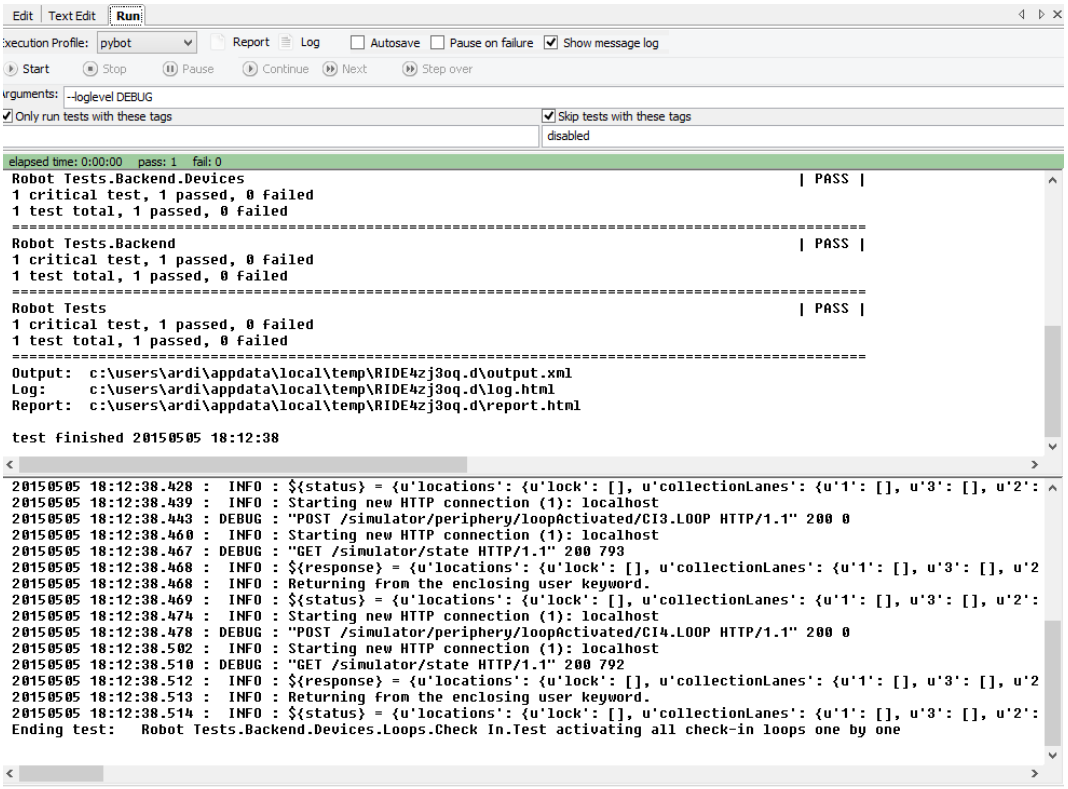
Peale testide kirjeldamist ning nendes olevate võtmesõnade defineerimist on võimalik need testid käivitada kasutades testide juuresolevat *Run* vaadet. Joonisel 25 on kujutatud selle vaate võimalused.




*Joonis 25. Testide käivitamise vaade*

Lisaks testi käivitamisele on võimalik siin määrata, milliste märksõnadega teste käivitada ning millised vahele jätta. Samuti on võimalik kaasa anda argumente käivitamisele, näiteks logimise aste, mida tulemuste vaatamisel soovitakse näha.

Käivitades loodud testi, saab näha õnnestunud ning ebaõnnestunud sammude kohta infot otse RIDE liidesest.




*Joonis 26. RIDE liideses käivitatud test*

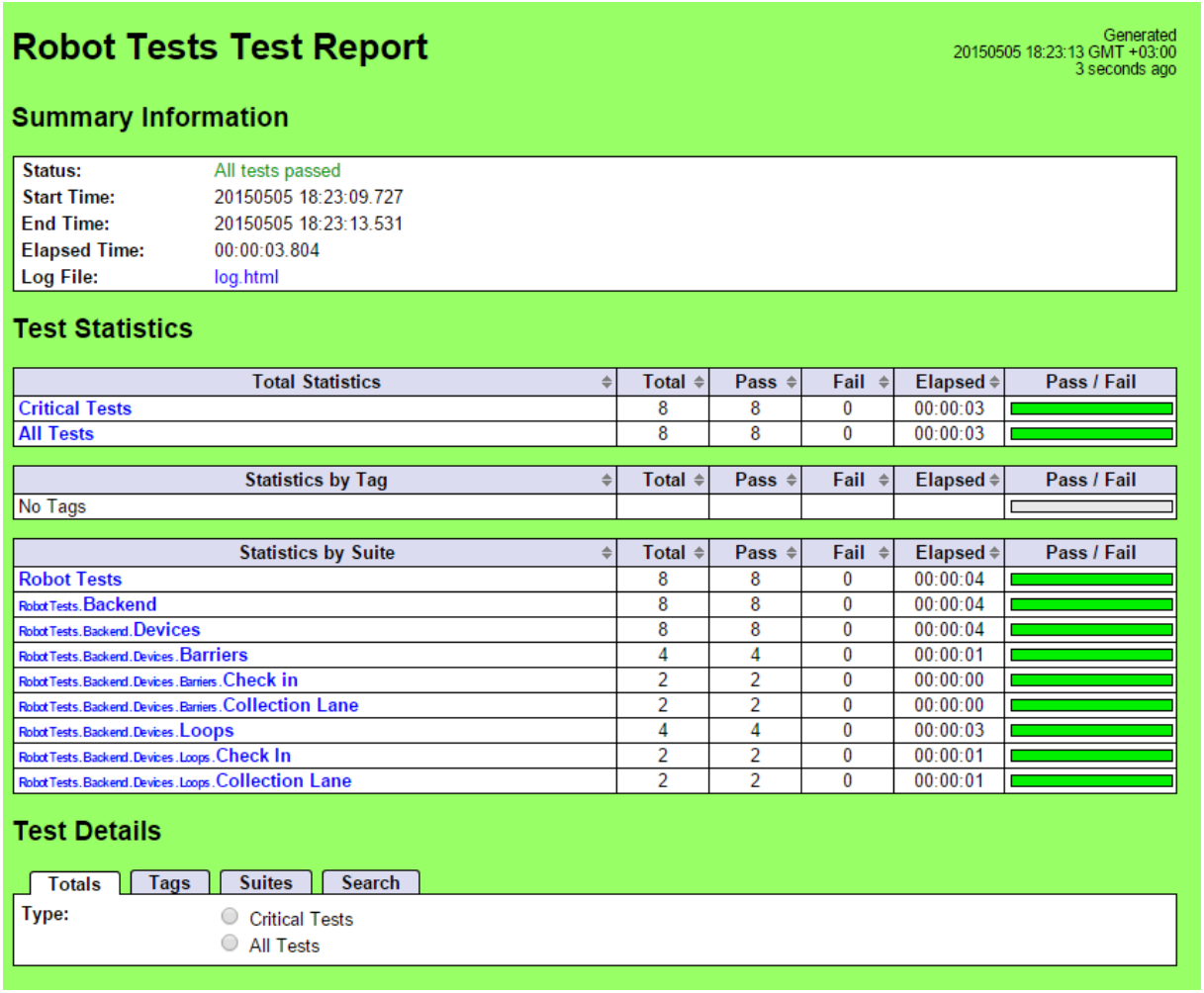
Joonisel 26 on toodud õnnestunud testi vaade. RIDE liideses testide käivitamise juures tuleb arvestada ka sellega, et kui lihtsalt käivitada, ilma midagi määramata, käivitab see kõik testid, mis liidesesse on imporditud. Seda käivitatavat kogumikku saab vähemaks määrata kolmel viisil:

1. Defineerides märksõnad, mille olemasolul test käivitatakse.
2. Defineerides märksõnad, mille olemasolul testi ei käivitata.
3. Testide struktuuri vaates (vasakul olev struktuuri puu) märgistades need testid, millised soovitakse käivitada.

Käivitamise ajal on võimalik teste ka peatada või jätta kauakestvad vahele. Liidese sees saab peale testide käivitamist infot selle kohta, kui palju eri taseme teste õnnestus või ebaõnnestus, kuhu väljundid paigutati ning kui palju kulus testide käivitamiseks aega. Samuti kuvatakse testides käivitunud võtmesõnade logi, kus on näha näiteks millised võtmesõnades välja kutsutud sõnumi vastused tulid (mis on abiks vigade leidmisel).

## 6.6 Väljastatavad testide raportid

Üks suuremaid eeliseid konkurentide ees on RIDE tugi kuvada Robot Frameworki poolt väljastatavaid põhjalikud ning hästi loetavaid HTML raporteid. Liidese sees oleva info lugemine on tavaliselt ebamugav ning raport annab parema visuaalse väljundi, mille abil on kiiremini võimalik vigu ning nende põhjuseid tuvastada.



**Robot Tests Test Report**

Generated
20150505 18:23:13 GMT +03:00
3 seconds ago

**Summary Information**

| | |
|---|---|
| Status: | All tests passed |
| Start Time: | 20150505 18:23:09.727 |
| End Time: | 20150505 18:23:13.531 |
| Elapsed Time: | 00:00:03.804 |
| Log File: | log.html |

**Test Statistics**

| Total Statistics | Total | Pass | Fail | Elapsed | Pass / Fail |
|---|---|---|---|---|---|
| Critical Tests | 8 | 8 | 0 | 00:00:03 | |
| All Tests | 8 | 8 | 0 | 00:00:03 | |

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

| Statistics by Suite | Total | Pass | Fail | Elapsed | Pass / Fail |
|---|---|---|---|---|---|
| Robot Tests | 8 | 8 | 0 | 00:00:04 | |
| Robot Tests.Backend | 8 | 8 | 0 | 00:00:04 | |
| Robot Tests.Backend.Devices | 8 | 8 | 0 | 00:00:04 | |
| Robot Tests.Backend.Devices.Barriers | 4 | 4 | 0 | 00:00:01 | |
| Robot Tests.Backend.Devices.Barriers.Check in | 2 | 2 | 0 | 00:00:00 | |
| Robot Tests.Backend.Devices.Barriers.Collection Lane | 2 | 2 | 0 | 00:00:00 | |
| Robot Tests.Backend.Devices.Loops | 4 | 4 | 0 | 00:00:03 | |
| Robot Tests.Backend.Devices.Loops.Check In | 2 | 2 | 0 | 00:00:01 | |
| Robot Tests.Backend.Devices.Loops.Collection Lane | 2 | 2 | 0 | 00:00:01 | |

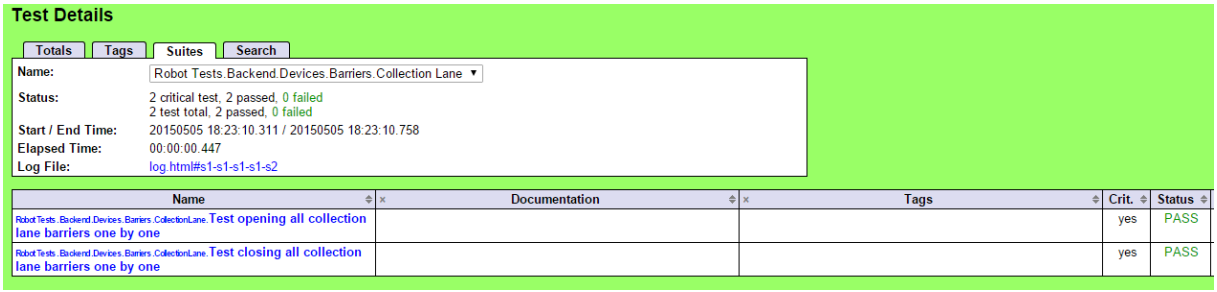
**Test Details**

Totals | Tags | Suites | Search

Type: Critical Tests / All Tests

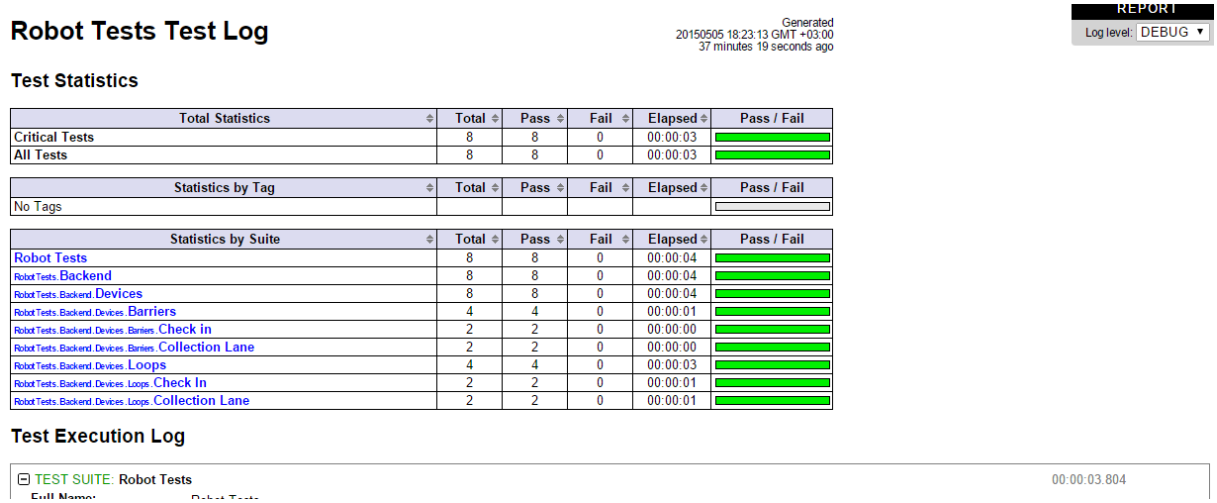
*Joonis 27. Testide käivitamise raport*

Joonisel 27 on näha üht sellist raportit. Iga testi tulemused on selgesti eristatud ning lähemaks uurimiseks tuleb soovitud kogumiku peale vajutada, mille tulemusel saab näha eraldi võtmesõnade tulemusi ning andmeid. HTML raportist saab sama info mis liidesestki, kuid see on paremini struktureeritud ning visuaalselt kergemini haaratavam.



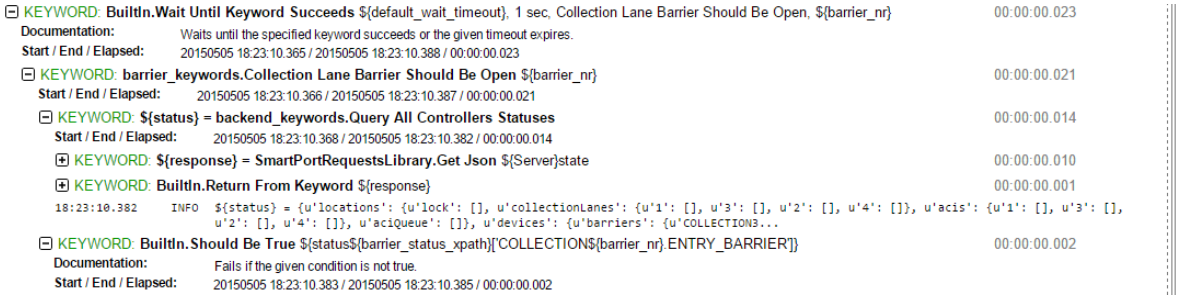

*Joonis 28. Täpsem testi käivitamise raport*

Joonisel 28 on esitatud testi detailsem raporti vaade. Siit on näha kogu vajalik informatsioon iga võtmesõna ning testi käivitamise kohta. Soovides mõne võtmesõna käivitamise kohta täpsemat infot, saab edasi vajutada võtmesõna peale, mis viib käivitamise logi vaatesse, kus on võtmesõnades toimunud tegevuste väljundid ning väärtused täpsemalt välja toodud.




*Joonis 29. Testide käivitamise logi*

Joonisel 29 on toodud samade testide käivitamise logi. Siin on samuti testide tulemused, kuid lisaks ka käivitamise logi. Joonisel on ka näha, et saab valida logimise taset vastavalt hetkevajadustele.




*Joonis 30. Testi täpsem logi*

Joonisel 30 on näha võtmesõna logi detailsemalt, kus on esitatud käivitamise kihis välja kutsutud programmeerimiskeele meetodi väljundeid. Hetkel on selleks REST sõnumi vastus, kust saab välja lugeda kõik väärtused, millega võtmesõnad tegelesid ning kui mõni neist on vale, siis saab seda kiiresti välja filtreerida.

Robot Framework ning tema liidesed pakuvad veel palju rohkem erinevaid võimalusi, kuid töös käsitletud võimalused on peamised, mida on vaja teada, kui alustada raamistikuga tööd. Alustuseks tuleks neid vahendeid kasutades koostada esialgne kogumik, mida saab hakata järjest edasi arendama nii uute võtmesõnade lisamisega kui ka olemasolevate täiustamisega. Samuti tuleks pidevalt jälgida testide struktuuri ning kui on märke, et seda tuleks ümber teha, siis tuleks see teostada võimalikult varakult. Nii saab vältida erinevaid hiliseid ebameeldivaid aega raiskavaid hoolduse tegevusi.

# 7. Metoodika kasutuselevõtu ebaõnnestumise peamised põhjused

Kirjeldatud metoodikat on üritanud kasutusele võtta paljud projektid ning iga õnnestumise kohta on tihti mitu ebaõnnestumist. Sellel on tavaliselt palju erinevaid põhjuseid olenevalt projekti kontekstist, kuid samas on välja kujunenud teatud mustrid, mille järgi on näha, kus kõige rohkem vigu tehakse. Mitmed metoodika kasutuselevõtu ebaõnnestumise põhjused kattuvad automaattestimise kasutuselevõtu probleemidega. Kuna testide automatiseerimise ning metoodika automatiseerimine on niivõrd lähedalt seotud, tuleb arvestada metoodika kasutuselevõtul testide automatiseerimise põhimõtete ning hoiatustega.

Antud peatükis välja toodud punkte tuleks võtta kui soovitusi, mille peale mõelda, kui üritada metoodikat ise kasutusele võtta. Mõeldes projekti alguses ning selle jooksul nendele punktidele, kasvab õnnestumise tõenäosus kindlasti, kuid samas igal projektil on oma eripärad ja need tuleks välja selgitada ning ka arvesse võtta.

Järgnevalt on välja toodud mõned suuremad vead:

1. Koostatud testkogumikule ei teostata piisavalt tihti hooldust. Halvasti struktureeritud ning planeeritud testkogumik nõuab pidevat hooldamist. Ilma pideva hoolduseta väheneb testkogumikust tulenev kasu järsult, kuna paljud olulised testid hakkavad valesid negatiivseid tulemusi näitama ja tihti takistavad ka ülejäänud testide käivitamist [27].
2. Võtmesõnadel põhineva testimise metoodika kasutusele võtmisel ei hangita selleks piisavalt häid teadmisi. Tihti kasutatakse metoodika raamistikke toetudes oma hetketeadmistele ning ei uurita, mis on selle metoodika eripärad ning peamised põhimõtted teiste metoodikate kõrval.
3. Kirjeldatud metoodika kasutuselevõtmisel ei tehta ka olemasolevale test- ja arendusprotsessile vajalikke muudatusi. Näiteks, metoodikat ei saa edukalt kasutusele võtta kaasamata projekti liikmeid, kes regulaarselt tutvuksid koostatud testidega. Eriti võiks siinkohal koostööd teha analüütikutega.
4. Valitakse oma projektile vale raamistik või lähenemine. Raamistikku tuleks hoolikalt valida ning samas küsida endalt pidevalt, kas kirjeldatud metoodika on üldse projektile sobilik ning vajalik. On võimalik, et projektile sobib hoopis mõni teine metoodika.

5. Tihti ei alustata oma testide automatiseerimisega piisavalt varakult. Alguses väikest kogumikku üles eskaleerida on lihtsam, kui üritada hiljem väga suurt rakendust automaattestidega katta.
6. Automaatse testimise kõrvalt ei tehta piisavalt manuaalset testimist, kuna kõike automatiseerida pole kunagi võimalik ja mõistlik. Selleks on liiga palju erinevaid variatsioone funktsionaalsustel ja nende testimiseks kuluv ajakulu ei tasu ära [27].
7. Ei ole inimesi, kes tehnilist kihti pidevalt toetaks. Paljudes projektides pannakse käivitamise kihti koostama ning hooldama programmeerijad, kellel pole selleks tihti piisavalt motivatsiooni ning aega. Et metoodika kasutuselevõtmine õnnestuks, peaks meeskonnas olema ka heade tehniliste teadmistega testija, kes suudab hallata kogu automaattestide kogumikku ning ei sõltu teistest rollidest [4].

# 8. Kokkuvõte

Käesoleva töö peamiseks eesmärgiks oli luua materjal, mis annaks võtmesõnadel põhineva testimise kohta praktilise ülevaate ning kirjeldaks nii metoodikat üldiselt kui ka tema kasutusvõimalusi. Töös esitati ülevaade metoodikast ning võtmesõnadel põhineva testimise seos tarkvara testimisega automaattestimise kontekstis.

Lisaks oli eesmärgiks anda selgeid juhtnööre tutvustatud metoodika ning raamistike reaalseks kasutuselevõtuks ja selgitada, millal ei ole mõistlik antud metoodikat kasutada ning kuidas valida endale sobivaim raamistik. Selleks tutvustati printsiipe, kuidas üldiselt automaattestimisega alustada ja millised võivad olla kõige sagedasemad probleemid. Metoodika juures kirjeldati selle erinevaid aspekte, kasulikkust, puudusi ja ohte ning ka seda, milline võiks kogu testprotsess välja näha peale metoodika rakendamist. Raamistike juures tutvustati kahe kõige populaarsema raamistiku põhilisi funktsionaalsusi, nende häid ja halbu külgi. Lõpuks toodi välja ka põhjused, miks paljud tarkvaraprojektid antud metoodika kasutuselevõtul ebaõnnestuvad.

Töös kasutati näiteid reaalsetest tarkvaraprojektidest ning autori kogemusi metoodika kasutamisest testimise valdkonnas, et anda ülevaade metoodikast kasutades reaalseid Eesti tarkvaraprojekte. Raamistike tutvustamisel kasutati näiteid kolmest eri projektist, kuid samas keskenduti peamiselt siiski metoodika ning raamistike ülevaatlikule tutvustamisele. Paljud välja toodud ohud ning soovitused tuginevad autori kogemustele eri projektides.

Pikemas perspektiivis on töö eesmärgiks arendada Eesti tarkvarafirmades automaattestimise metoodikate kujunemist. Hetkel kättesaadavate eestikeelsete materjalide hulk on väike ning teemad on laiali erinevate allikate vahel. Kõiki olulisi aspekte ei katnud algusest lõpuni ka ükski leitud inglisekeelne materjal. Käesoleva tööga on loodud juhend, millest võiks peamiselt kasu olla tarkvara testijatele.

# 9. Kasutatud kirjandus


[1] Cem Kaner, James Bach, Bret Pettichord. Lessons learned in software testing, Wiley Computer Publishing. 2002.

[2] Laurent Bossavit. The Leprechauns of Software Engineering. 2012.

[3] Why is Software Testing Important to a business? (06.05.2015) [online] http://www.softwaretesting.com.au/Why_is_Software_Testing_important.php

[4] Rashmi Gupta, Neha Bajpai. A keyword-driven tool for testing Web applications (KeyDriver), International Journal of Advanced Computer Science and Applications, Vol. 3, No. 3, 2012.

[5] Pierre F. Tiako. Software Applications: Concepts, Methodologies, Tools, and Applications, Langston University. 2009.

[6] Elfriede Dustin, Thom Garrett, Bernie Gauf. Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality, Addison-Wesley Professional. 2009.

[7] Jim Heumann. Generating Test Cases From Use Cases (06.05.2015) [online] http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jun01/GeneratingTestCasesFromUseCasesJune01.pdf

[8] Tobias Walter. Keyword-Driven Test Automation Framework (06.05.2015) [online] http://www.ranorex.com/blog/keyword-driven-test-automation-framework

[9] Selenium2Library (06.05.2015) [online] http://rtomac.github.io/robotframework-selenium2library/doc/Selenium2Library.html

[10] Elfriede Dustin, Thom Garrett, Bernie Gauf. Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality, Addison-Wesley Professional. 2009.

[11] Behaviour Driven Development) (06.05.2015) [online] http://guide.agilealliance.org/guide/bdd.html

[12] What Is An Automated Test Framework? (06.05.2015) [online]
http://www.automatedtestinginstitute.com/home/index.php?option=com_content&id=1096:faq-framework&Itemid=25

[13] Robot Framework (06.05.2015) [online]
http://robotframework.org/

[14] FitNesse (06.05.2015) [online]
http://www.fitnesse.org/

[15] EMOS Framework (06.05.2015) [online]
http://emos-framework.sourceforge.net/

[16] TestComplete (06.05.2015) [online]
http://smartbear.com/product/testcomplete/overview/

[17] Software Automation Framework Support (06.05.2015) [online]
http://safsdev.sourceforge.net/home.htm

[18] Worksoft Certify (06.05.2015) [online]
https://www.worksoft.com/products/worksoft-certify

[19] TestArchitect (06.05.2015) [online]
http://testarchitect.logigear.com/

[20] Andreas Ebbert-Karroum. How to Structure a Scalable And Maintainable Acceptance Test Suite (06.05.2015) [online]
https://blog.codecentric.de/en/2010/07/how-to-structure-a-scalable-and-maintainable-acceptance-test-suite/

[21] Page Objects (06.05.2015) [online]
https://code.google.com/p/selenium/wiki/PageObjects

[22] Elfriede Dustin, Jeff Rashka, John Paul. Automated Software Testing: Introduction, Management, and Performance, Addison-Wesley Professional. 1999.

[23] Rick Mugridge, Ward Cunningharm. Fit for Developing Software, Prentice Hall. 2008.

[24] Wiki (06.05.2015) [online]
http://en.wikipedia.org/wiki/Wiki

[25] XML (06.05.2015) [online]
http://www.w3schools.com/xml/

[26] Cascading Style Sheets (06.05.2015) [online]
http://www.w3schools.com/css/css_intro.asp

[27] Amit Srivastava. Test Automation and Software Development, Technology Review. 2002.

**Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks**

Mina, Ardi Türk (sünnikuupäev: 27.11.1989)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

**„Võtmesõnadel põhineva testimise metoodika ning raamistikud tarkvara automaattestimises“**,

mille juhendajaks on Helle Hein,

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

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.

Tartus, 14.05.2015