TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatika instituut
Tarkvaratehnika õppetool

Automatiseerimisvahendi valik mobiilirakenduse testimiseks agiilses keskkonnas

Magistritöö

Üliõpilane: Liina Kasvand
Üliõpilaskood: IABMM121773
Juhendaja: Jekaterina Ivask

Tallinn
2015

Autorideklaratsioon

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

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

(*kuupäev*) (*allkiri*)

Annotatsioon

Käesolev magistritöö kirjeldab automatiseerimisvahendi valikut mobiilirakenduse testimiseks agiilses keskkonnas. Töö eesmärgiks on uurida, millised omadused peaksid olema vahendil keskkonnas, kus nõuded arendatavale tarkvarale muutuvad kiiresti ning üritab leida neile omadustele vastava automatiseerimisvahendi funktsionaalse rakenduse testimise jaoks. Esialgu kirjeldatakse agiilset arendust üldiselt ning automatiseerimise raskuskohti selles, seejärel antakse ülevaade projektist ning rakendusest, mille tarvis automatiseerimisvahendit otsitakse. Lõpetuseks analüüsitakse teoreetilisi teadmisi agiilsest keskkonnast, viiakse läbi intervjuu pikaajalise kogemusega automatiseerijaga ning pannakse kirja punktid, millele peaks automatiseerimisvahend vastama.

Töö tulemusena testitakse nelja erinevat automatiseerimisvahendit, kirjeldatakse nende omadusi ning valitakse välja üks, mis vastab kõige enam seatud kriteeriumitele. Vahendit, mis välja valitakse, hakatakse kasutama kirjeldatud funktsionaalse rakenduse automatiseeritud testimiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 58 leheküljel, 5 peatükki, 21 joonist ja 2 tabelit.

Abstract

The current Master's thesis describes selection of automation tool for mobile application in agile environment. The purpose of this work is to investigate, which characteristics should automation tool have in environment, where requirements are not stable and change during project and to find automation tool for native application regarding this criteria.

Firstly agile development is described generally and difficulties of automation in it. Secondly project and application, for what automation tool is searched, is described. Lastly analysis of theoretical part is done, interview with automation tester is conducted and criteria, which characteristics automation tool should have, is written down.

As a result four automation tools are tested and analyzed against criteria set for agile environment and one automation tool is selected that covers most of this criteria. This tool will be used in a project to automate testing for native application.

The thesis is in Estonian language and contains 58 pages, 5 chapters, 21 figures and 2 tables.

Lühendite ja mõistete sõnastik

Android Studio	***Android Studio*** Ametlik IDE Android rakenduste loomiseks.
Android SDK	***Android Software Development Kit*** Osa Android Studiost.
BDD	***Behaviour Driven Development*** Agiilse arenduse meetod, kus tarkvara arendatakse huvirühmade perspektiivist lähtudes. BDD kirjeldab, kuidas soovitud nõue peaks käituma *(behave).*
Crystal	***Crystal*** Agiilsetest metodoloogiatest kõige kohanemisvõimelisem, mille põhiprintsiibiks on arvestamine iga projekti iseärasustega.
DSDM	***Dynamic Systems Development Method*** Agiilne metodoloogia, mis parandab maksumust ning kvaliteeti projektis tehes seda hinnates nõudeid printsiipidega “Peab”, “Peaks”, “Võib” ning “Seekord jääb välja”
Domeeni spetsiifiline keel	***Domain-specific language*** Keel, mis on spetsiaalselt loodud ühe domeeni jaoks.
Eclipse	***Eclipse*** Integreeritud arenduskeskkond, mis pakub arendajale võimaluse luua ning kompileerida tarkvara ning kasutada emulaatoreid.
Emulaator	***Emulator*** Seade või programm, mis imiteerib teise seadme või programmi tööd.

FDD	***Feature-Driven Development*** Agiilne tarkvaraarenduse metodoloogia, mille põhiprintsiip on arendamine tunnusjoone *(feature)* järgi.
Inkrementaalne mudel	***Incremental model*** Tarkvara arenduse meetod, kus tarkvara arendus on tükeldatud osadeks.
IDE	***Integrated Development Environment*** Tarkvara, mis on mõeldud tarkvara arendajale ning mis on varustatud vajalike osadega tarkvararakenduste loomiseks.
Jira	***Jira*** Atlassian poolt välja antud vigade haldamise tööriist.
Kaizen	***Kaizen*** Agiilne metodoloogia, mille põhiprintsiibiks on pidev parandamine.
Kanban	***Kanban*** Agiilne metodoloogia, mille põhiprintsiipideks on tehtavate tööde visualiseerimine, parasjagu tehtavate olevate tööde limiteerimine ning tähtsaimate töödega alustamine.
Kasutusmugavuse testimine	***Usability testing*** Tehnika, milles keskendutakse kasutajale ning testitakse tarkvara kasutaja vaatest.
Kernel	***Kernel*** Ressursijaotust ja muid põhifunktsioone hõlmav operatsioonisüsteemi keskne osa ehk südamik.
Koskmudel	***Waterfall model*** Tarkvara arenduse meetod, kus iga tegevus toimub etappidena ning jadamisi. Üks vanemaid tarkvaraarenduse meetodeid.

Lean	***Lean*** Agiilne meetod, mille põhiprintsiipideks on praagi elimineerimine, võimalikult viimasel hetkel otsustamine, tarnimine nii kiiresti kui võimalik, meeskonnaliikmetele võimu andmine.
Reliis	***Release*** Tarkvara avaldamisvalmis uuendus sama versiooni piires.
Robotframework	***Robotframework*** Üldine testide automatiseerimise raamistik vastuvõtutestimise jaoks.
Selenium	***Selenium*** Veebiautomatiseerimise tööriist.
Skript	***Script*** Käsujada, mida täidetakse ilma kasutajapoolse vahelesegamiseta.
Testiplaan	***Test plan*** Plaan, millega määratletakse testimise skoop, tegevused ning ajaplaan.
xUnit	***xUnit*** Ühisnimi mitme ühiktesti raamistiku tarvis.
XP	***Extreme programming*** Agiilse tarkvaraarenduse üks meetoditest, kus kasutatakse lühikesi iteratsioone ning tugevat suhtlust inimeste vahel.

Jooniste nimekiri

Joonis 1: Agiilne tarkvaraarendus [2] 15
Joonis 2 Scrum [3] 16
Joonis 3 Veebirakendus [20] 23
Joonis 4 Hübriid rakendus vasakul ja veebileht paremal [20] 24
Joonis 5 Mobiilirakenduse testimise protsess agiilses arenduses [22] 25
Joonis 6 OnTime funktsionaalne rakendus 29
Joonis 7 Calabash skripti näide 38
Joonis 8 Elementide määratlemine Calabashis 39
Joonis 9 Tulemuste kuvamine Calabashis käsureal 40
Joonis 10 HTML raport ebaõnnestunud testiloo kohta Calabashis 40
Joonis 11 HTML raport õnnestunud testiloo kohta Calabashis 40
Joonis 12 MonkeyTalk skripti sisestus tabeli kujulujul 43
Joonis 13 MonkeyTalk skripti sisestus MonekyTalk keeles 43
Joonis 14 MonkeyTalk skripti sisestus JavaScriptis 43
Joonis 15 Elementide määramine MonkeyTalkis 44
Joonis 16 MonkeyTalk tulemuste kuvamine HTML raportis 45
Joonis 17 OnTime peale instrumentimist MonkeyTalk Agendiga 47
Joonis 18 SeeTest skripti näide 48
Joonis 19 Elementide määratlemine SeeTestis 49
Joonis 20 SeeTest raport kasutajaliideses 50
Joonis 21 SeeTest raport HTML-is 50

Tabelite nimekiri

Tabel 1 Nõuded automatiseerimisvahendile agiilses keskkonnas 30
Tabel 2 Automatiseerimisvahendite tulemused 51

Sisukord

1. Sissejuhatus 12
1.1 Taust ja probleem 12
1.2 Ülesande püstitus 12
1.3 Metoodika 12
1.4 Ülevaade tööst 13
2. Agiilne tarkvaraarendus 14
2.1 Scrum 15
3. Tarkvara testimine 17
3.1 Tarkvara testimise meetodid 17
3.2 Testimine agiilses keskkonnas 18
4. Mobiiltarkvara 21
4.1 Mobiilsed operatsioonisüsteemid 21
4.1.1 Android operatsioonisüsteem 21
4.1.2 iOS operatsioonisüsteem 22
4.2 Mobiilirakendused 22
4.2.1 Funktsionaalsed rakendused 22
4.2.2 Veebirakendused 22
4.2.3 Hübriid rakendused 23
4.3 Mobiilirakenduste automatiseeritud testimine agiilses keskkonnas 24
5. Vahendi valimiseks kriteeriumite loomine 27
5.1 Projekti kirjeldus 27
5.1.1 Projekti liikmed 27
5.1.2 Tarkvara arenduse protsessi kirjeldus projektis 27
5.1.3 Automatiseerimine OnTime projektis 29
5.1.4 Rakenduse OnTime ülevaade 29
5.2 Kriteeriumid automatiseerimisvahendile 30
6. Automatiseerimisvahendi väljavalimine 33
6.1 Appium 34
6.2 Calabash 37
6.3 MonkeyTalk 42
6.4 SeeTest 47
6.5 Järeldus 51

7. Kokkuvõte 55
Summary 56
Kasutatud kirjandus 57

1. Sissejuhatus

Käesolev magistritöö keskendub testimisele agiilses keskkonnas mobiilsel platvormil. Eesmärk on leida automatiseerimisvahend, mis sobiks kõige enam keskkonda, kus nõuded pole projekti alguses päris paigas ning muutuvad selle vältel. Automatiseerimisvahendit otsitakse funktsionaalse rakenduse testimise tarvis.

1.1 Taust ja probleem

Aina enam võetakse tarkvara arenduses kasutusele agiilset metoodikat, kus nõudeid loodava tarkvara funktsionaalsusele ei seata jäigalt paika projekti alguses, vaid need muutuvad projekti elutsükli vältel. Sellest tulenevalt muutub kogu tarkvara testimine kaootilisemaks ning keerulisemaks, kuna testilugude kirjutamine ning haldamine peab kaasas käima muutuvate nõuetega. Automatiseerimine agiilses projektis omab omasid väljakutseid, kuna skriptide loomiseks ja haldamiseks kulub enamasti rohkem aega, kui manuaalseks testimiseks mõeldud testilugude jaoks. Samuti peab automatiseeritud skript andma vastuse automaatselt, ilma inimsekkumiseta.

Nutitelefonide kasutuselevõtt lõi võimaluse alla laadida ning kasutada erinevaid rakendusi. Olenevalt sellest, kuidas rakendus on arendatud, on need jagatud kolme erinevasse tüüpi — funktsionaalne-, hübriid- ning veebirakendus.

1.2 Ülesande püstitus

Käesoleva magistritöö eesmärgiks on leida kõige sobivam automatiseerimisvahend funktsionaalse rakenduse jaoks projektis, kus kasutatakse ühte agiilset metodoloogiat — Scrumi.

1.3 Metoodika

Eesmärgi saavutamiseks:

- Analüüsitakse agiilset keskkonda ning luuakse kriteeriumid, mida automatiseerimisvahend peaks toetama.

- Võetakse kasutusele neli automatiseerimisvahendit, Appium, Calabash, MonkeyTalk ning SeeTest.
- Võrreldakse vahendeid kriteeriumitega, mis agiilsele keskkonnale seati.

Magistritöö tulemusena valitakse välja üks automatiseerimisvahend, mis vastab kõige enam seatud kriteeriumitele. Valitud vahend võetakse kasutusele funktsionaalse rakenduse testimiseks.

1.4 Ülevaade tööst

Magistritöö on jagatud 5-e peatükki. Esimene peatükk kirjeldab agiilset tarkvaraarendust ning Scrumi lähemalt.

Teine peatükk kirjeldab testimist üldiselt — mis on manuaalne ja automatiseeritud testimine. Milliseid väljakutseid loob automatiseerimine agiilses keskkonnas.

Kolmas peatükk kirjeldab täpsemalt Android ning iOS operatsioonisüsteeme. Lisaks luuakse ülevaade erinevatest mobiilirakenduste tüüpidest.

Neljas peatükk loob ülevaate OnTime projektist ning rakendusest endast. Räägitakse lähemalt Scrum meetodist, mis kasutusele on võetud ning meeskonnast ja selle liikmetest. Analüüsitakse agiilset keskkonda ning luuakse kriteeriumid, mille põhjal automatiseerimisvahendeid võrrelda.

Viiendas peatükis tuuakse välja iga automatiseerimisvahendi vastavused loodud kriteeriumitele. Võrdluse põhjal valitakse välja automatiseerimisvahend, mis vastab kõige enam seatud kriteeriumitele.

Antud lõputöö koostaja on varasemalt töötanud manuaalse testijana ning automatiseerimisega kokku puutunud vaid põgusalt, seetõttu vaadatakse vahendeid ka alustava automatiseerija pilgu läbi.

2. Agiilne tarkvaraarendus

Käesolevas peatükis antakse ülevaade agiilsest tarkvaraarendusest ning magistritöö koostaja igapäevatöös kasutusel olevast agiilsest metoodikast Scrumist.

Agiilse tarkvaraarenduse manifesti kohaselt järgitakse agiilses tarkvaraarenduses järgnevaid printsiipe:

- Kõrgeim prioriteet on tagada kliendi rahulolu tarnides talle vajalikku tarkvara võimalikult kiiresti ja tihti.
- Nõuete muutumine, ka tarkvara arenduse lõpus, on oodatud. Taolised muudatused muudab agiilne protsess kliendi konkurentsieeliseks.
- Töötavat tarkvara tarnitakse nii tihti kui võimalik, alates mõnest nädalast kuni mõne kuuni, eelistatuna väiksema perioodi suunas.
- Äriinimesed ning arendajad peavad töötama igapäevaselt koos kogu projekti vältel. [1]

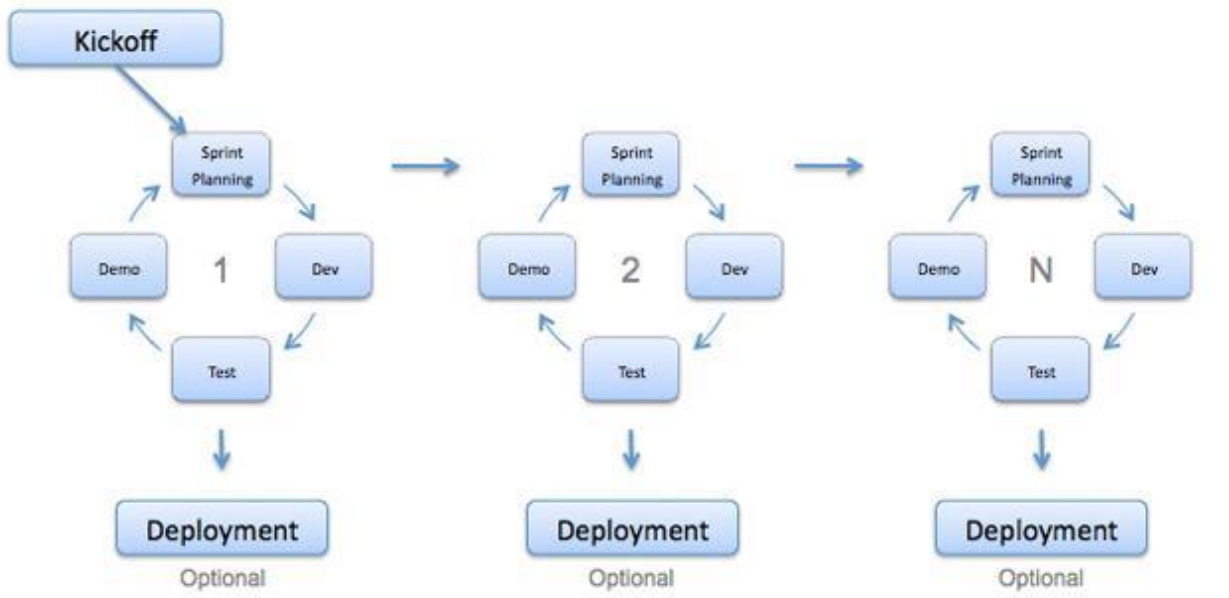
Agiilne tarkvaraarenduse mudel on ka inkrementaalne mudel, tarkvara luuakse kiiretes inkrementaalsetes tsüklites. Igale tsüklile liidetakse uus väiksem funktsionaalsus. Eelkõige kasutatakse agiilset tarkvaraarendust ajakriitiliste rakenduste loomiseks. Agiilsed metoodikad on näiteks XP, Scrum, Kanban, Kaizen, Lean, Crystal, DSDM, BDD ning FDD. Magistritöö autor kasutab enda igapäevases töös Scrumi.

Agiilset tarkvaraarenduse mudelit on mõttekas kasutada juhtudel

- kui uued muudatused projektis on vajalikud, kuna väikeste muutuste vastuvõtmine igas tsüklis pole kulukas,
- kui varases tarkvara loome etapis pole kliendi nõuded loodavale tarkvarale veel täiesti kindlad,
- kui otsitakse vabadust teha suuri otsuseid hilisemates projekti etappides, kui projekti kulg ning kliendi soovid on juba selgemad. [2]

Agiilne meetod seega vähendab ebavajaliku tarkvara osade loomist, mis võib juhtuda koskmudeli puhul, milles nõuded luuakse tarkvara projekti alguses ning algsetest nõuetest kõrvalekaldumist ei soovita. Agiilne meetod aitab projektil tänases kiiresti muutuvas IT-maailmas ajaga kaasas käia. Lisaks aitab agiilne arendusmeetod kiirendada toote turustamist, kuna iga sprint luuakse uus reliisikandidaat. Agiilne tarkvaraarendus näeb suuresti välja, nagu kujutatud joonis 1-l, kus kogu arendustsükkel on jagatud erinevatesse sprintidesse. Iga sprindi alguses toimub sprindi planeerimise koosolek *(srpint planning)*, kus pannakse paika

funktsionaalsused, mis peavad valmima vastava sprindi lõpuks. Seejärel toimub arendus, kus arendatakse uued funktsionaalsused ning parandatakse vanad vead, kui neid on. Edasi toimub arendatud tarkvara testimine ning lõpuks demo valminud osast. Seejärel liigutakse edasi järgmisesse sprinti.



Joonis 1: Agiilne tarkvaraarendus [2]

2.1 Scrum

Scrum meetodit tutvustati esimest korda 1993. aastal Jeff Sutherland poolt ning on tänapäevaks saanud üheks kasutatavamaks agiilseks metoodikaks. Scrum on agiilse tarkvaraarenduse raamistik, mille puhul ei määratleta kindlaid nõudeid projekti alguses ning töö toimub iteratsioonides. Iteratsioonid Scrumis on enamasti kahe- kuni neljanädalased. Scrum meetodit on kujutatud joonisel 2. Scrumi põhilised tegevused on järgnevad:

- Toote omanik *(product owner)* koostab toote backlog-i *(product backlog)*, kuhu lisab ning prioritiseerib tegevused ning uued toote funktsioonid *(features)*, mis valmivale tarkvarale soovitakse lisada *(wish list)*.
- Iga sprindi alguses toimub sprindi planeerimine *(sprint planning)*, kus meeskond võtab toote backlog-ist teatud hulga ülesandeid, mis nad arvavad tulevase sprindiga tehtud saavat ning lisavad selle sprindi backlog-i. Sprindi planeerimise koosolekul määratakse

umbmäärane aeg, mis teatud ülesande täitmiseks kulub. Samuti määratakse, kes sellega tegelema hakkab.

- Igapäevaselt toimuvad Scrumis stand-up koosolekud. Stand-up koosolekutel räägib iga meeskonnaliige, mida eelmisel päeval tegi ning mis on plaanis teha käesoleval päeval. Räägitakse takistustest, mis esinevad sprindi backlog'i täitmisel ning esitatakse meeskonnaliikmetele küsimusi, mis ülesannete täitmisel ette tulevad.
- Scrum meister *(Scrum master)* on Scrum meeskonnas isik, kes juhib koosolekuid ning hoiab meeskonna fokusseerituna eesmärgil.
- Iga sprindi lõpuks peab olema valmis potentsiaalne reliisi kandidaat- versioon tarkvarast, mis oleks kõlbulik saata poodi (Google Play või iStore) või näidata huvirühmadele *(stakeholders).*
- Iga sprint lõppeb sprindi ülevaate *(sprint review)* ning sprindi retrospektiivi *(sprint retrospective)* koosolekutega. Sprindi ülevaate koosolekul mõeldakse, kuidas tõsta meeskonna efektiivsust, räägitakse läbi toote backlog, arutletakse tulevaseid töid. Sprindi retrospektiivi koosolekul tuuakse välja, mis läks sprindi jooksul hästi ning mis läks halvasti. Otsitakse võimalusi, kuidas parandada halvasti läinud asju.
- Järgmine sprint algab jällegi uue sprindi planeerimise koosolekuga, kus meeskond võtab uue osa ülesandeid toote backlog-ist ning lisab sprindi backlog-i. [3]

Joonis 2 Scrum [3]

3. Tarkvara testimine

Käesolevas peatükis antakse ülevaade tarkvara testimisest ning testimise kahest erinevast meetodist — manuaalsest ning automatiseeritud testimisest. Seejärel tutvustatakse tarkvara testimist agiilses keskkonnas ning lõpetuseks räägitakse lähemalt automatiseerimisest agiilses keskkonnas ning väljakutsetest, mis sellega kaasnevad. Antud peatükist saadakse sisend kriteeriumite loomiseks automatiseerimisvahendi valiku jaoks.

Tarkvara testimine on protsess, mille käigus valideeritakse ja verifitseeritakse, kas tarkvara produkt või rakendus

- vastab ärilistele ja tehnilistele nõuetele, mis olid disaini ja arenduse loomise aluseks,
- töötab nagu oodatud,
- töötab samade omadustega erinevas keskkonnas. [4]

Tarkvara testimine on protsess mitmetest erinevatest tegevustest, selle alla kuulub näiteks testiplaani loomine ja selle ajakohastamine, testilugude kirjutamine ja nende jooksutamine, vastavalt projektile erinevad testimise tüübid ja meetodid. Tarkvara on võimalik testida kahel viisil, manuaalselt ning automatiseeritud meetodil.

3.1 Tarkvara testimise meetodid

Manuaalse testimise puhul testib testija tarkvara käsitsi — manuaalselt. Manuaalset testimist kasutatakse enamasti juhtudel, kus sama testilugu peab läbima vähe kordi või kus inimlik lähenemine testimisele on oluline, nagu näiteks kasutusmugavuse testimine. Manuaalset testimist kasutatakse ka juhtudel, kus testitulemust on vaja kiiresti, programmi vastavad sammud läbitakse ning vastus saadakse kohe. Manuaaltestimise suurimaks miinuspooleks on selle aeglus ning piiratus — näiteks ei ole võimalik testida manuaalselt tarkvara vastupidavust koormusele või ei ole võimalik testida iseärasusi, mida inimsilmaga ei näe, nagu näiteks erinevad värvigammad või pikslitega mõõdetavad suurused.

Automaatse testimise puhul kasutatakse tarkvara testimiseks arvutiprogrammi. Selleks loob automatiseerija skriptid, millega öeldakse sammud ette, mida tarkvara läbima peab. Automaatset testimist kasutatakse näiteks regressiooni testimise puhul — kui sama testilugu on tarvis läbida projekti jooksul mitu korda või koormustestimise puhul — kui programmi tuleb koormata. [5] Kõige levinum viis tarkvara automatiseerimisel on testide läbimine

kasutajaliidese põhjal— automaattarkvara simuleerib kasutajat ning tema liigutusi kasutajaliidesel. Lisaks testisammude automaatsele läbimisele võidakse automatiseerimist kasutada aga veel järgnevate tegevuste puhul:

- Andmete ja skriptide genereerimine — automaattööriistad võivad koostada spetsiaalseid andmeid, nagu näiteks randomiseeritud e-maile või sõnumeid, mida saab kasutada testimise jaoks.
- Süsteemi konfigureerimine — tööriistad võivad aidata säilitada või taasluua süsteemi parameetreid.
- Simulatsioon — tööriistad võivad simuleerida alamsüsteeme või keskkonda, mida pole (veel) olemas või mis on liiga kallid testimise jaoks.
- Uurimise eesmärgil — automaattööriistad võivad aidata teha nähtamatu info inimestele nähtavaks. Näiteks võivad need statistiliselt uurida testitavat programmi, genereerida logi faili või jälgida süsteemi parameetreid.
- Tegevuste salvestamine ning katvuse analüüs — tööriistad võivad registreerida tegevusi, mis tehtud ning hiljem kirjeldada raportis, millised alad vajaksid veel testimist
- Testimise juhtimine — tööriistad võivad anda tagasisidet testitulemuste kohta, pakkuda testiideid või presenteerida mõõdikuid. [6]

Automatiseerimise põhilised miinused on suur raha- ja ajakulu skriptide loomisele ja uuendamisele ning inimfaktori puudumine- automatiseeritud test teeb seda, mis ette öeldud ega märka vigasid, mida manuaalne testija märkaks. Miinuspoolena veel on automatiseerimistarkvara tihtipeale üpris kulukas. [5]

3.2 Testimine agiilses keskkonnas

Agiilses tarkvaraarenduses peab testimine olema tagatud igas iteratsioonis, kuna iga iteratsiooni lõpus peab valmima töötav tarkvara. Iteratsiooni alguses valitakse kasutajalood, mis peavad olema realiseeritud selle iteratsiooni lõpuks, testimine on oluline kogu iteratsiooni vältel, mitte vaid iteratsiooni lõpus. [7] Kuna tarkvara projekti funktsionaalsus suureneb iga iteratsiooniga, suureneb ka testimise maht ning regressiooni oht— muudatused koodis võivad põhjustada uute või vanade vigade esiletulemist. Agiilses keskkonnas on tarkvara testijal erinevad väljakutsed, nende hulgas:

- vähene ja muutuv dokumentatsioon nõuete kohta,
- tarkvara testimisega alustatakse varakult ning testitakse pidevalt muutuvat tarkvara,

- nõuete muutumise tõttu on testilugusid etteulatuvalt luua keeruline,
- nõuete muutumise tõttu tuleb testilugusid pidevalt uuendada ning hallata,
- kuna kood muutub pidevalt on keeruline tagada, et kõik muudatused on testitud,
- testide automatiseerimisel on tarvis väga hoolikalt analüüsida, milliseid teste tasub automatiseerida. [8]

Automatiseerimine projektis on vajalik ning selle oskuslik planeerimine väga oluline. Kui tarkvara automatiseerimine on puudulik või kui selle ajakohastamine on liiga ajakulukas, jääb muude testimistegevuste jaoks liiga vähe aega. Cirilo Wortel, automatiseeritud testimise konsultant, annab mõningaid häid soovitusi, kuidas automatiseerimist projektis edukalt täide viia:

- Automatiseeritud testide loomine peaks olema küllaltki lihtne ning intuitiivne.
- Testimisvahend peaks olema paindlik, et muudatuste sisseviimine skriptis oleks võimalikult lihtne ning kiiresti tehtav.
- Testid peaksid olema loetavad ja arusaadavad kõigile. Testide dokumenteerimine ja vorming peaks olema ühetähenduslik.
- Testide jooksutamine peaks olema kiire ja usaldusväärne, kuna kiire tagasiside on peamine eesmärk. [8]

Automatiseeritud testimise puhul võib ette tulla mitmeid nõrku kohti, mis põhjustavad testide ebaõnnestumist ning mida tasuks vältida. Neist mõningad on järgnevad:

- Testkoodi kordus *(The Test Duplication)* — kood kordub mitmes kohas. Tagajärjeks mitmekordsete muudatuste sisseviimise vajadus, kui midagi koodis muutub.
- Välise ressursi kaasamise ohud *(Mystery Guest)* — kui test kasutab väliseid ressursse, nagu näiteks faili testandmetega *(test data)*, on raske öelda, mida test tegelikult verifitseerib. [9]
- Keeruline testi kood *(Complex Test Code)* — liiga palju keerulisi koodiridu, sellest tulenevalt raske verifitseerida koodi õigsust ja suurem potentsiaal vigade ilmnemiseks.
- Liiga palju koodi *(Can't See the Forest for the Trees)* — liiga palju testkoodi muudab segaseks, mida test verifitseerib.
- Tingimuslik testiloogika *(Conditional Test Logic)* — testid, mis sisaldavad tingimuslikke lauseid (if-lauseid või *loop*-e). Keeruline verifitseerida, kas tingimuslik lause on korrektne ning testib alati sama asja.

- Keeruline tühistamise loogika *(Complex Undo Logic)* — testid, mis rikuvad testkeskkonna, kuna jätavad peale jooksutamist mõned andmed alles. See võib viia andmete lekkimiseni ning järgnevate testide põhjuseta ebaõnnestumiseni.

Sellised nõrgad kohad skriptis viivad olukordadeni, kus muudatuste tegemine testitavas süsteemis lõhub ka testilood ning hoolduse kulud tõusevad kõrgele. Testide puhul, mis sõltuvad üksteisest, on oht sattuda tsüklisse- kui üks test ebaõnnestub, ebaõnnestuvad ka testid, mis sõltuvad eelnevast. Testid, mis vajavad inimsekkumist andmete kustutamiseks, muudavad hooldatavuse jällegi kulukaks.

The Test Automation Manifesto on loodud mitmete agiilsete projektide kogemuste põhjal ning võtavad arvesse eelnevaid nõrku kohti automatiseerimisel. The Test Automation Manifesto pakub välja järgnevad kriteeriumid, millised peaks olema automatiseeritud testid:

- Kompaktsed — nii lihtsad kui võimalikud, kuid mitte lihtsamad.
- Iseennast kontrollivad — testid raporteerivad enda tulemusi ise, inimsekkumine pole vajalik.
- Korratavad — teste saab korrata mitu korda järjest ilma inimsekkumiseta.
- Robustsed — testid annavad alati samu tulemusi, välised tegurid ei mõjuta neid.
- Hõlmavad — testid verifitseerivad kõiki testitava tarkvara nõudeid.
- Vajalikud — iga test on vajalik mingi spetsifikatsiooni jaoks.
- Selged — iga test ja rida selles on kõigile üheselt arusaadav.
- Praktilised — testide jooksutamine pole ülemäära ajakulukas.
- Konkreetsed — iga test, juhul kui see leiab vea koodis, viitab otseselt sellele veale.
- Iseseisvad — iga test jooksutab end ise.
- Hooldatavad — testid peaksid olema kergesti arusaadavad ja muudetavad.
- Järgitavad — iga test peaks olema seotud nõudega, mida see testib. [11]

4. Mobiiltarkvara

Käesolevas peatükis antakse ülevaade mobiilsest tarkvarast. Räägitakse lähemalt mobiilsetest operatsioonisüsteemidest ning luuakse ülevaade mobiilirakenduste erinevatest liikidest. Ülevaade luuakse eesmärgiga tutvustada lähemalt antud magistritöö ülesande püstituses toodud funktsionaalset rakendust, aga ka teisi mobiilirakenduste liike. Seda selleks, et hiljem oleks arusaadav, milliseid rakendusi valitud automatiseerimisvahendid toetavad.

Lõpetuseks räägitakse peatükis olulistest punktidest, mida peaks silmas pidama automatiseerimisvahendi valikul mobiilirakenduste jaoks. See alampeatükk on samuti sisend kriteeriumite loomiseks automatiseerimisvahendi valikul.

4.1 Mobiilsed operatsioonisüsteemid

2014 aasta neljandas kvartalis müüdi maailmas ligikaudu 377,4 miljonit mobiiltelefoni. Mobiilsetest operatsioonisüsteemidest oli kasutusel põhiliselt Android 76,6% ning iOS 19,7%. [12]

4.1.1 Android operatsioonisüsteem

Android on Google avatud lähtekoodiga operatsioonisüsteem, mis põhineb Linux kernelil. Android Open Source Project (AOSP) näeb ette mõningad standardid, mis muudavad loodava tarkvara "Androidile sobivaks" *(Android compatible).* Selleks, et kasutada mõningaid Androidiga kaasnevaid teenuseid, nagu näiteks Google Play, peab tarkvara olema testitud teatud AOSP poolt kindlaksmääratud testidega. Operatsioonisüsteemi avatus on arvatavasti ka põhjuseks, miks mitmed firmad selle enda jaoks leidnud on ning Android maailmas hetkel juhtival positsioonil mobiilsetest operatsioonisüsteemidest on. Üheks avatud koodi eesmärgiks on pakkuda arendajaile platvormi, millele luua uusi rakendusi. [13] Google Play rakenduste koguarv 2014. aastal oli üle 1,4 miljoni. [14] Nüüdseks on rakenduste allalaadimiste arv Google Play poest tõusnud 1,5 biljonini kuus. Igapäevaselt aga aktiveeritakse rohkem kui üks miljon Android seadet. [15] Android versioonide nimetused on erilised, kuna igal versioonil on magustoidu nimi ning nimed ise on alfabeetilises järjestuses- versioon 1.6 nimetuseks on Donut (sõõrik), sellele järgnes 2.0 Eclair (ekleer) ning sellele omakorda 2.2. Froyo (külmutatud jogurt). Nüüdseks on jõutud versiooninumbrini 5.0 Lollipop (pulgakomm). [16]

4.1.2 iOS operatsioonisüsteem

iOS on Apple operatsioonisüsteem, mis on mõeldud eranditult Apple enda toodetele- iPhone, iPod ja iPad. [17] Viimane versioon iOS-ist hetkel on 8.2, millel on olemas ka Apple kella (Apple watch) tugi. [18] Esimene iPhone tuli välja 2007 aastal ning sel ajal kandis selle operatsioonisüsteem nime iPhone OS. iOS on arendatud puutetundliku ekraani tarvis ning arvatavasti oli see operatsioonisüsteem tõukeks, miks puutetundlik ekraan nii populaarseks sai. Apple Store'i tutvustati 2008 aastal iOS versiooninumbriga 2, mis lubas ka kolmandate osapoolte rakendusi alla laadida. [17] 2014. aastal oli rakendustes arv Apple Store's üle 1,2 miljoni. [14] 2014. aasta oktoobris tõusis rakenduste allalaadimine Apple Store's rekordini— 7,8 miljonini päevas. [19]

4.2 Mobiilirakendused

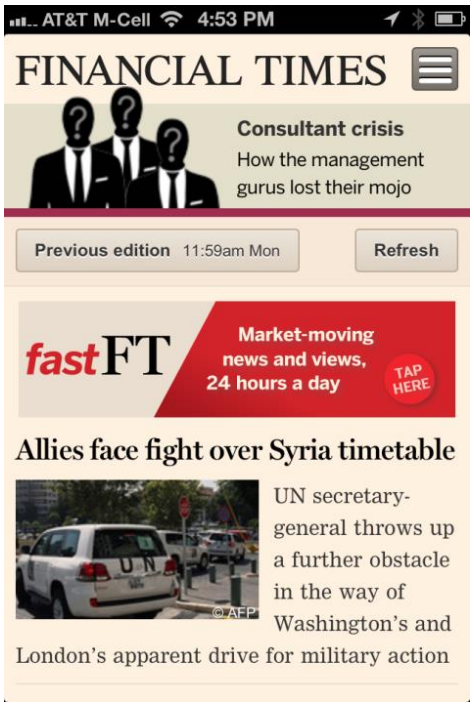
Mobiilirakendused jagunevad olenevalt sellest, kuidas nad loodud on, kolme erinevasse kategooriasse- funktsionaalsed rakendused *(native apps)*, veebirakendused *(web apps)* ning hübriid rakendused *(hybrid apps)*.

4.2.1 Funktsionaalsed rakendused

Funktsionaalsed rakendused töötavad kasutaja telefonis ja on kasutajale kättesaadaval ikoonidena telefoni menüüst. Funktsionaalsed rakendused laeb kasutaja alla enamasti rakenduste poest, nagu Google Play ja Apple Store. Need luuakse spetsiaalselt igale platvormile eraldi ning võivad kasutada seadme kõiki võimalusi, nagu näiteks kaamera, kompass või GPS. Funktsionaalsed rakendused saavad töötada ka offline-s, nagu näiteks kalendri rakendus.

4.2.2 Veebirakendused

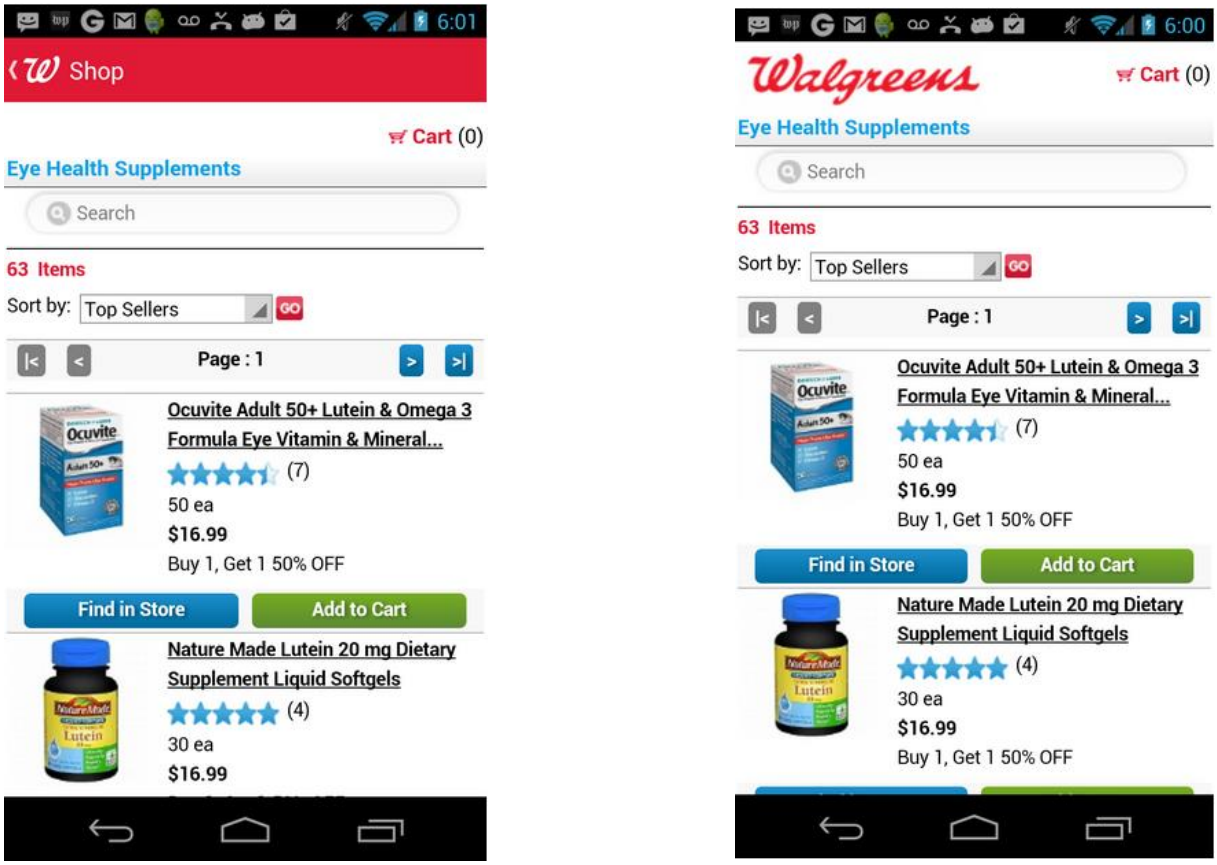
Veebirakendused on veebilehed, mis on kohandatud mobiilse seadme tarvis. Need avatakse brauseri teel ning on enamasti loodud HTML5-s. Veebirakendused näevad välja ja tunduvad kui funktsionaalsed rakendused, kuid tegelikkuses pole samamoodi loodud. Kasutajad avavad veebirakenduse brauseris URLi teel, misjärel pakutakse võimalust "installida" veebirakendus oma seadme koduekraanile *(homescreen)*. Seejärel luuakse sellele lehele järjehoidja *(bookmark)*. Vahe veebirakenduse ja veebilehe vahel on väga hägune, kuid need pole siiski samad. Veebirakendusena võib leida näiteks mitmeid uudistelehti. Financial Times-i veebirakenduse näidis on kujutatud joonisel 3.



Joonis 3 Veebirakendus [20]

4.2.3 Hübriid rakendused

Hübriid rakendused on osaliselt funktsionaalsed ning osaliselt veebirakendused. Sarnaselt funktsionaalsetele rakendustele saab hübriid rakendusi alla laadida rakenduste poodidest ning sarnaselt veebirakendustele kasutab hübriid rakendus töötamiseks brauserit ning HTML-i. Hübriid rakendused on populaarsed, kuna nõnda saavad firmad enda lehe esindatuse rakenduste poes ilma eraldi rakendust arendamata. Selline lähenemine hoiab kokku arenduskuludelt, kuna sama HTML koodi komponente saab kasutada erinevatel operatsioonisüsteemidel. Joonistel 4 on näide Walgreens hübriid rakendusest ning veebilehest Android operatsioonisüsteemil. Nagu näha, on nad küllaltki sarnased oma välimuselt. [20]



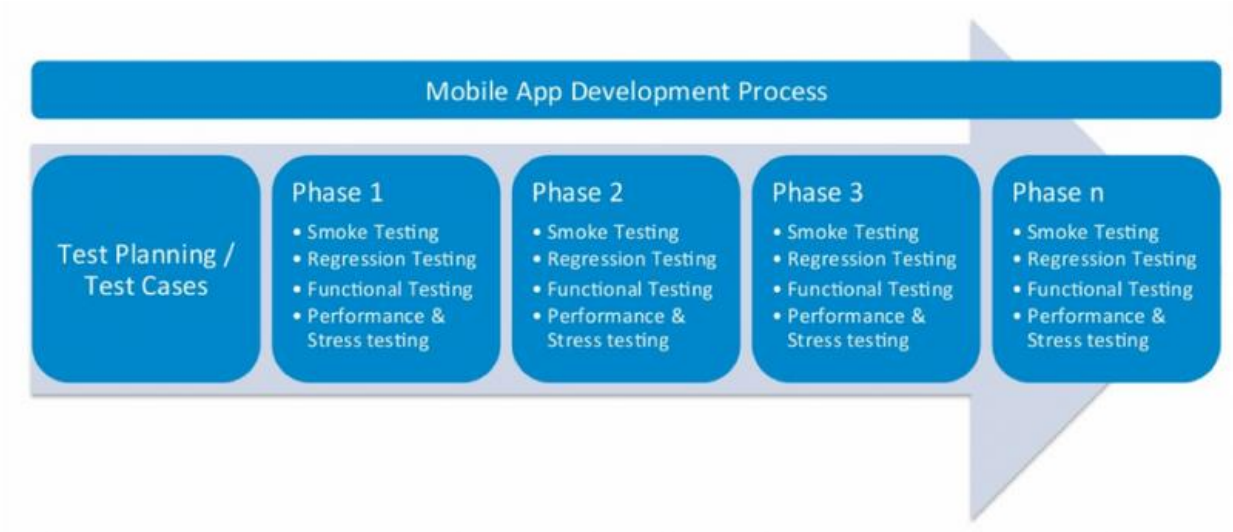
Joonis 4 Hübriid rakendus vasakul ja veebileht paremal [20]

4.3 Mobiilirakenduste automatiseeritud testimine agiilses keskkonnas

Testimine mobiilsel platvormil on mõneti erinev arvutil testimisest ning sellega kaasnevad erinevad väljakutsed, nagu näiteks erinevat tüüpi ühenduvused (WIFI, GPRS, GPS), mälu- ja töötlemispiirangud [21], muudetav orientatsioon *(landscape, portrait)*. Puutetundlik ekraan toob kaasa erinevaid liigutusi nagu koputus *(tap)* või kahekordne koputus *(double tap)*, ekraanil liuglemine *(swipe)* või ekraani liigutamine *(pan)*.

Tarkvara testimise protsess mobiilsel platvormil agiilses keskkonnas näeb välja suuresti, nagu kujutatud joonis 5-l. Igas faasis toimub esialgne tarkvara suitsutestimine *(smoke testing)*, kus kontrollitakse, kas põhifunktsioonid reliisil töötavad. Regressioonitestimisega *(regression testing)* kontrollitakse, ega uuendused koodis uusi vigu tarkvaras pole põhjustanud. Funktsionaalne testimine *(functional testing)* testib tarkvara vastavust nõuetele ja kas selle funktsioonid korrektselt töötavad. Jõudlus- ja stress testimine *(performance and stress testing)* kontrollivad tarkvara vastupidavust koormusele. Jõudlustestimise eesmärk on kaardistada

tarkvara jõudluse tase ning skaleeritavus. Stress testimine näitab, millise koormuse käes ja miks tarkvara lakkab töötamast. [22]



Joonis 5 Mobiilirakenduse testimise protsess agiilses arenduses [22]

Testimistüübid, mida agiilses keskkonnas automatiseerida, sõltuvad suuresti projektist- selle suurusest, eelarvest, ajagraafikust ning inimressurssidest. Testimistüübid, mida tasuks kaaluda automatiseerimisel on kasutajaliidese põhine testimine *(UI testing)*, monkey testing ja ühiktestimine *(unit testing)*. [20] Kasutajaliidese põhine testimine testib tarkvara kasutaja vaatest, ühiktestimine arendaja vaatest, kus tarkvara on jagatud tükkideks ning integratsiooni pole veel toimunud. [23] Monkey testimise käigus ei peeta silmas ühtegi konkreetset testilugu. Programm (Monkey) annab tarkvarale erinevaid, juhuslikke sisendeid ning sellega kontrollitakse, kuidas tarkvara neile sisenditele vastata suudab. [24]

Automatiseerimisvahendi valikul mobiilirakenduste puhul tuleks silmas pidada järgnevaid nõudeid:

- Jailbraking ning rootimine — rooting (Android puhul) ning jailbraking (iOS puhul) tähendab luba andmist seadme alamsüsteemidele eesmärgiga eemaldada mõned tarkvaralised ja riistvaralised limiteeringud.
- Tõeline objekti märkamine *(true object recognition)* — kui elementi on võimalik määratleda tema klassi *(class)* või id *(id)* järgi, pole tähtsust kui tema asukoht graafiliselt muutub, testilugu sellest ei ebaõnnestu. Kui aga objekti on võimalik määratleda vaid graafiliste tunnuste põhjal, hakkab iga graafilise muudatuse järel automatiseeritud test ebaõnnestuma.

- Integratsioon juba olemasoleva *Integrated Development Environments*-iga (IDE-ga) — IDE on näiteks Eclipse, kus automatiseerija saab luua skripti, seda kompileerida ning kasutada emulaatorit. Kui automatiseerija on varasemalt kasutanud programme automatiseerimiseks, on lihtsam jätkata vana tuntud vahendiga.
- Kõrge skriptide kasutatavus — skriptid peaksid olema kasutatavad üle operatsioonisüsteemide ning erinevate seadmete peal. Väga tihti luuakse rakendused mitmele platvormile (enamasti Android ning iOS platvormile). Seetõttu on suureks aja kokkuhoiuks kui automatiseerimisvahend suudab toetada mõlemaid platvorme ning platvormiülese testimise näol — ühe testiskriptiga saab testida nii Android platvormil kui ka iOSil.
- Füüsilise seadme ning emulaatori/simulaatori kasutatavus — kas automatiseerimisvahend toetab füüsilise seadme ning emulaatori ning simulaatori peal testimist. Kui iOS erinevaid seadmeid pole turul väga palju, siis Android seab selles osas väljakutseid, kuna erinevaid seadmeid turul on väga palju. Kindlasti pole võimalik töökohal osta kõiki füüsilisi seadmeid, mistõttu on emulaatori ja simulaatori toetus oluline, kuna testida tuleb rakendust võimalikult laialdaselt.
- Veebirakenduste toetus — veebirakendused ning hübriid rakendused on tõusuteel asendamaks funktsionaalseid rakendusi. Funktsionaalsed rakendused on loodud töötama konkreetsel operatsioonisüsteemil, veebirakendusi aga saab kasutada kõigil operatsioonisüsteemidel, kuna põhinevad HTML5-l.
- Raporteerimine — kui täpselt suudab automatiseerimisvahend kirjeldada testimise tulemusi.
- Liigutuste toetus — kas puutetundliku ekraaniga kaasnevate puudutuste, nagu liuglemine *(swipe)*, kerimine *(scroll)* või ekraani puudutamine *(tap)* on toetatud.
- Manuaalne või plaanitud käivitus — kas automatiseeritud vahend suudab ka automaatselt teste täita, näiteks kell kaks öösel automatiseeritud testi käivitada.
- Integreeritavus teiste programmidega — kas automatiseerimisvahendit on võimalik integreerida teiste vahendite, nagu jõudlustestide vahenditega või Jenkinsiga, pideva integratsiooni serveriga. [24]

5. Vahendi valimiseks kriteeriumite loomine

Käesolevas peatükis antakse ülevaade Scrum projektist ning funktsionaalsest rakendusest OnTime. Lõpetuseks luuakse kriteeriumite punktid, mille vastu automatiseerimisvahendeid võrreldakse.

5.1 Projekti kirjeldus

Projekt nimi on OnTime ning arendatakse funktsionaalset mobiilirakendust nimega OnTime, mida on lähemalt kirjeldatud peatükis 5.1.4 Rakenduse OnTime ülevaade. Projektis kasutatakse Scrum raamistikku ning projekt sai alguse 2014. aasta novembris. Selle aja jooksul on loodud esimene reliis 1.0, mis on alates 2015. aasta maikuust saadaval ka Android Play Store's.

5.1.1 Projekti liikmed

OnTime meeskonda kuuluvad toote omanik, Scrum meister, neli arendajat, kaks manuaalset testijat ning üks automatiseerija.

- Toote omaniku ülesanneteks on pidada ülevaadet toote backlog-i üle ning suhtlemine huvigruppidega. Tema paneb paika reliisi plaane ning annab viimase sõna, kas reliis on valmis poodi üleslaadimiseks.
- Scrum meister viib läbi igapäevaseid stand-up koosolekuid. Samuti vastutab Scrum meister sprint ülevaate ning sprint retrospektiivi koosolekute eest. Tema annab nõu, kui tundub, et mingi ülesanne backlog-ist võib venida pikemaks või ilmneda takistusi.
- Arendajad loovad uusi funktsionaalsusi ning parandavad neile raporteeritud vigu.
- Automatiseerija ülesandeks on automatiseerida testid ning hoida neid kaasajastatuna.
- Manuaalsed testijad loovad testilugusid ning testivad tarkvara manuaalselt testilugude järgi või teevad uurivat testimist *(exploratory testing)* ilma testilugudeta.

5.1.2 Tarkvara arenduse protsessi kirjeldus projektis

Projektis ei ole varasemalt kasutatud automatiseerimist, kuna rakenduse funktsionaalsus oli veel väike. Alates reliis 1.0-st võetakse kasutusele pidev integratsioon ning automatiseeritud testimine. Pidev integratsioon tähendab, et arendajate tehtud muudatused integreeritakse põhikoodi *(repository)* mitu korda päevas ning seetõttu valmib igapäevaselt tarkvarast 1-3 tarkvara versiooni. Kasutusel on Jenkins, pideva integratsiooni server, kuhu tarkvara versioonid üles laetakse.

Scrum sprindid OnTime projektis on kahenädalased. Sprint algab alati teisipäeval ning lõppeb esmaspäeval. Esimesel sprindi päeval toimub sprindi planeerimise koosolek. See kestab tavaliselt neli tundi ning selle jooksul valitakse toote backlogist ülesanded uue sprindi jaoks. Toote backlogi, nagu ka vigade haldus, toimub Jiras. Sprindi planeerimisel osalevad kõik Scrum meeskonna liikmed. Sprindi backlogi lisatakse nii arenduse ülesanded, uued funktsionaalsused ning vanade vigade parandused, kui ka neile vastavad testimise ülesanded. Igale ülesandele määratakse kaal, kus 1 tähendab 8 tundi ühe inimese tööd. Kui uue funktsionaalsuse loomiseks kulub 3 päeva 2 arendaja tööd, on kaaluks 6.

Peale sprindi planeerimist algab sprint. Selle käigus võetakse sprindi backlogist ülesanded ning täidetakse need. Igapäevaselt toimub hommikuti stand-up koosolekud, kus osalevad arendajad, testijad ning Scrum meister. Stand-up'id on kuni 15-minutilised. Stand-up'idel räägivad kõik liikmed, mida tehti eelneval päeval ning mida plaanitakse teha käesoleval päeval. Räägitakse läbi takistused oma ülesannete täitmisel ning küsitakse küsimusi, kui neid on. Kui sprindi jooksul plaanitakse uus funktsionaalsus, loovad manuaalsed testijad sellele testilood. Testilugude haldamiseks on kasutusele võetud TestLink. Kui uus funktsionaalsus on valmis, testivad manuaalsed testijad vastu testilugusid tarkvara. Vead kantakse Jirasse ning seejärel parandatakse arendajate poolt samas sprindis või kantakse edasi järgnevasse sprinti. Igapäevaselt loovad arendajad kas uusi funktsionaalsusi või parandavad vigu lisades iga muudatuse Jenkins'isse ning luues mitu versiooni tarkvarast päevas (pidev integratsioon).

Iga sprint peetakse ka grooming koosolek, mille käigus toote omanik, arendajad ning testijad vaatavad üle toote backlogi. Toote backlog Jiras on hinnatud nõnda, et listis kõige peal olevad ülesanded on kõige olulisemad ning võetakse järgnevasse sprinti. Grooming koosoleku jooksul hinnatakse uuesti üle viimased toote backlogis olevad ülesanded ning lisatakse uusi kui tarvis. Vajadusel eemaldatakse ülesanded, mis enam pole vajalikud.

Sprindi lõpus toimub sprindi ülevaate koosolek, mille käigus räägitakse läbi saavutused, mis eelmisel sprindil tehtud sai ning muudetakse vastavalt toote backlogi. Sprindi ülevaate koosolekul osalevad kõik Scrum meeskonna liikmed.

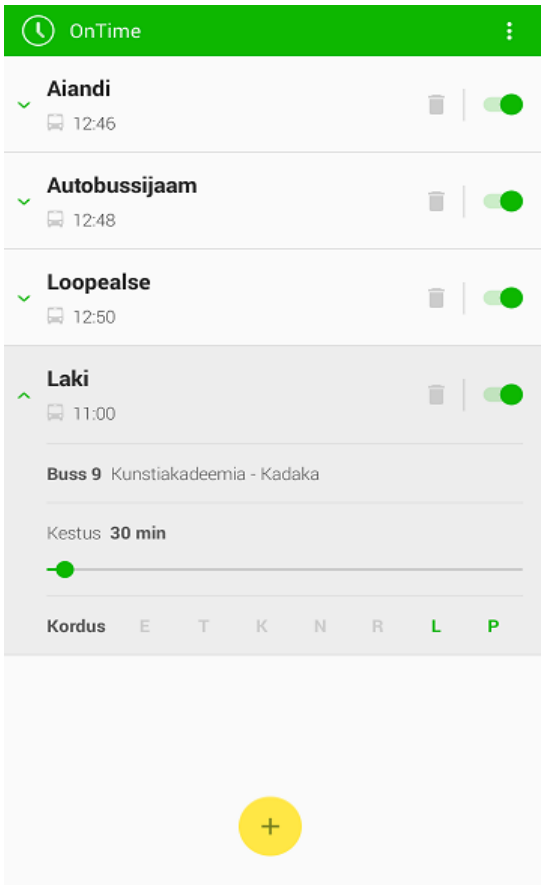
Sprindi lõpus toimub ka sprindi retrospektiivi koosolek, kus arutatakse läbi, mis läks hästi ja mis halvasti. Iga meeskonnaliige toob välja enda jaoks olulised punktid. Scrum meister juhib seda koosolekut ning peale sprindi retrospektiivi koosolekut on iga halvasti läinud asja juures lahendus ning inimene, kes sellega edaspidiselt tegelema hakkab.

5.1.3 Automatiseerimine OnTime projektis

Hetkel on OnTime rakenduse funktsionaalsus veel väike, seetõttu pole mõtet regressiooniteste automatiseerida. Selle loomine ning haldamine oleks antud projekti puhul liialt ajakulukas. Kuna pideva integratsiooni tõttu luuakse igapäevaselt 1-3 uut tarkvaraversiooni, on tarvis nende versioonide põhifunktsionaalsusi testida. Seetõttu võeti vastu otsus automatiseerida suitsutestid OnTime rakenduses.

5.1.4 Rakenduse OnTime ülevaade

OnTime on FOB Solutions OÜ poolt loodud funktsionaalne rakendus, mis hetkel on loodud Android operatsioonisüsteemile, kuid tulevikus plaanitakse toetama hakata ka iOS operatsioonisüsteemi. OnTime võimaldab ühistranspordi kasutajal luua alarmi talle sobivaks ajaks, mis tuletab meelde, millal kodunt või töölt välja peaks astuma, et soovitud transpordile jõuda. OnTime on mõeldud kasutamiseks Android kellaga, kuid töötab ka vaid telefonil. Hetkel on toetatud vaid Tallinna transport kuid eesmärgiks on laieneda ka muudesse Eesti linnadesse ning välismaale. OnTime on kujutatud joonisel 6.



Joonis 6 OnTime funktsionaalne rakendus

5.2 Kriteeriumid automatiseerimisvahendile

Selleks, et saada realistlikumat ülevaadet, mida automatiseerija agiilses keskkonnas automaattestide programmilt eeldab, viidi läbi intervjuu Indrek Kahu-ga, FOB solutions OÜ pikaajalise kogemusega automatiseerijaga. Tuginedes eelnevates peatükkides kogutud informatsioonile ja läbiviidud intervjuule, on loodud nõuete punktid, mida automatiseerimisvahend võiks toetada agiilses keskkonnas. Eelnevat kogutud informatsiooni analüüsiti ning kohandati OnTime projektile ning agiilsele keskkonnale vastavaks. Tulemused on toodud tabelis 1.

Tabel 1 Nõuded automatiseerimisvahendile agiilses keskkonnas

Operatsioonisüsteemide toetus	Selleks, et kahele platvormile ei peaks looma eraldi testilugusid, võiks automatiseerimisvahend toetada vähemalt kahte laialtlevinuma operatsioonisüsteemi Android ning iOS platvormiülest testimist.
Mobiilirakenduste toetus	Milliseid rakendusi (veebi-, hübriid- ning funktsionaalseid rakendusi) automatiseerimisvahend toetab. Olenevalt projektist võib see punkt saada määravaks automatiseerimisvahendi valimisel.
Füüsilise seadme ning emulaatori/simulaatori toetus	Kas testimisvahend toetab füüsilise seadme peal testimist ning emulaatori/simulaatori kasutamist.
Kasutatavad keeled	Selleks et automatiseerija ei peaks vahendi valikul hakkama õppima täiesti uut keelt, võiks programm toetada ka mõnda tuntud keelt. Lisaks on üldlevinud keelte kohta lihtsam leida materjali internetist.
Maksumus	Kui palju maksab automatiseerimisvahend, arvestada tuleb kogu projekti eelarvega ning kui palju on automatiseerimiseks ette nähtud summa.
Lähtekoodi vajalikkus	Kas testitava rakenduse lähtekoodile ligipääs või selles muudatuste tegemine on vajalik.

Seadme rooting ning jailbraking vajadus	Kas automatiseerimisvahend nõuab seadmel muudatuste tegemist.
Testskriptide loomise võimalused	Kuidas on võimalik luua skripte. Kas salvestus ja taasesitus (*record and play back)* meetod on toetatud, kus automatiseerimisvahend salvestab kasutaja liigutused skriptina ning seejärel saab sama skripti järgi mängida (*play back).* Kas automatiseerimisvahendil on olemas GUI (*Graphical User Interface),* mille abil teste kirjutada või on võimalik skriptida vaid terminaliaknas. Üldjuhul agiilses keskkonnas vaid salvestus ning taasesitus meetod ei sobi, kuna skriptid on enamasti nende tööriistadega pikad ning raskesti hallatavad.
Elementide määratlemine ning tõelise elemendi määratlemine *(true object recognition)*	Mil moel määratletakse rakenduse elemente, mida skriptis kasutatakse. Kas elementide määratlemine nende id või klassi järgi (tõelise elemendi määratlemine) on tagatud.
Integreeritavus teiste programmidega	Kas vahendit on võimalik integreerida muude projektis kasutatavate vahenditega, nagu näiteks Jenkins.
Mitme seadme *(multi-device)* tugi	Kas testiskripti on võimalik jooksutada korraga mitme seadme peal.
Tulemuste kuvamine	Kuidas ja kui selgelt on kuvatavad automaattestide tulemused. Testide jooksutamine ning tagasiside agiilses keskkonnas peab olema kiire ja usaldusväärne.
Skriptimine ja muudatuste tegemine	Kui kerge on luua ja teha muudatusi skriptis. Kas skriptid on selged ja arusaadavad kõigile.
Skripti vastavus The Test Automation Manifesto kriteeriumitele	Kas testivahendis loodav skript on: ● Kompaktne — nii lihtne kui võimalik. ● Iseennast kontrolliv — test raporteerib tulemused automaatselt.

	● Korratav — testi saab korrata mitu korda järjest ilma inimsekkumiseta. ● Robustne — test annab alati samu tulemusi. ● Selge — iga test ja rida selles on kõigile üheselt arusaadav. ● Praktiline — testi jooksutamine pole ülemäära ajakulukas. ● Konkreetne — test, mis leiab vea koodis, viitab otseselt sellele veale. ● Iseseisev — test jooksutab end ise. ● Hooldatav — test on kergesti arusaadav ja muudetav. ● Järgitav — test on seotud nõudega, mida see testib.

6. Automatiseerimisvahendi väljavalimine

Antud magistritöö raames uuriti lähemalt nelja automatiseerimisvahendit- Appium, Calabash, MonkeyTalk ning SeeTest. Need vahendid langesid valikusse, kuna toetavad kõik vähemalt iOS ning Android operatsioonisüsteeme.

OnTime koodis loodi viga, peites inglise keele valik ära, et näha, kuidas vahendid kuvavad tulemusi, kui testiskripti sammud läbivad ja ebaõnnestuvad. Testilood, mida automatiseeriti, on toodud alljärgnevalt.

ID	**001**
Lühikirjeldus	Testiloo eesmärk on vahetada rakenduse keel inglisekeelseks
Eeltingimused	Rakendus ei ole avatud
Sammud	1. Ava rakendus 2. Keeldu tarkvara uuendusest 3. Kliki raadionupul 4. Vali “Inglise keel”
Oodatav tulemus	Inglise keel on valitud rakenduse keeleks

ID	**002**
Lühikirjeldus	Testiloo eesmärk on lisada uus meeldetuletus määratud kellaajaks määratud bussipeatusese ning -liini jaoks
Eeltingimused	Rakendus ei ole avatud
Sammud	1. Ava rakendus 2. Keeldu tarkvara uuendusest 3. Kliki “pluss” nupul 4. Vali kellaajaks 11:00 5. Vali Buss number “9” 6. Vali suund “Kunstiakadeemia- Kadaka” 7. Vali peatus “Laki” 8. Vali “L” ja “P”
Oodatav tulemus	Uus meeldetuletus valitud andmetega on loodud

Kuna eelnevalt on inglise keel koodisiseselt peidetud, peab testilugu 001 ebaõnnestuma ning 002 õnnestuma.

6.1 Appium

Appium on avatud koodiga funktsionaalsete testide automatiseerimiseks mõeldud raamistik. Appium kujutab endast test-serverit, mis muudab testiskripti testitavale seadmele arusaadavaks. Appium kasutab Selenium JSON Wire Protocol-i, mistõttu saab automatiseerimiseks kasutada kõiki keeli, mida toetab ka Selenium.

Operatsioonisüsteemide toetus

Appium on platvormiülene testimisvahend ning selle abil on võimalik automatiseerida Android, iOS ning Firefox operatsioonidele mõeldud rakendusi.

Mobiilirakenduste toetus

Appium toetab kõigi kolme erineva mobiilirakenduse, veebi-, hübriid- ja funktsionaalsete rakenduste automatiseerimist. Veebirakendusi on võimalik testida ka reaalsetel brauseritel (Androidil Chrome ning iOS-l Safari).

Füüsilise seadme ning emulaator/simulaator toetus

Toetatud on nii füüsilised seadmed (iPhone, iPad, Android ja Firefox seadmed) kui ka simulaatorid (iOS ja Firefox OS) ja emulaatorid (Android).

Kasutatavad keeled

Automatiseerija saab kasutada talle sobilikku keelt nagu näiteks Java, Objective-C, JavaScript, PHP, Ruby, Python või C#. Seda tänu Appiumi klient-server tüüpi lahendusele- klient, mis jooksutab automatiseerija valitud keeles teste ning server, mis tegelikult testid teostab.

Maksumus

Appium on vabavaraline ning avatud koodiga tarkvara. Sauce Labs Inc. pakub tasulist teenust SauceLabs, kus on võimalik seadmetel nii manuaalset kui automatiseeritud testimist teha. Manuaalne testimine toimub klaviatuuri ning hiire abil.

Lähtekoodi vajalikkus

Lähtekoodis muudatuste tegemine pole vajalik, Appium järgib põhimõtet, testida sama rakendust, mis laetakse üles poodi.

Seadme rooting/jailbraking vajadus

Seadme rooting/jailbraiking vajadus puudub.

Testskriptide loomise võimalused

Appium Inspector vahendi abil on salvestamine ja taasesitamine võimalik OSX-ile mõeldud versioonist Appiumil. Keelt, milles skriptid salvestatakse on võimalik valida ning vahetada Ispector'i *Language Selection Dropdown* abil. Kuna Appium on vahelüli testiskripti ning rakenduse vahel, on võimalik kirjutada teste automatiseerijale sobivas keskkonnas ning võtta kasutusele talle tuntud IDE, näiteks Eclipse.

Antud magistritöö puhul kasutati RIDE *robotframework* vahendit, milles skripte luuakse Python keeles.

Elementide määratlemine

Appium kasutab tõelist elementide määratlemist nende klasside või id-e järgi. Elemente on võimalik määrata Appium Inspector abil. Antud tööriista ei õnnestunud magistritöö testimise käigus Windowsil tööle saada, see viga on ka raporteeritud paljude teiste Appiumi kasutajate poolt. Seetõttu kasutati magistritöö puhul Android Studio SDK kaustas olevat vahendit "Uiautomatorviewer", mis annab samuti elementide klassid, id-d ning muud omadused.

Integreeritavus teiste programmidega

Appiumi on võimalik integreerida pideva integratsiooni vahendite, nagu Jenkinsi või Travisega.

Mitme seadme tugi

Testimine mitmel seadmel korraga on võimalik.

Tulemuste kuvamine

Kuna automatiseerija saab kasutada Appiumiga erinevaid skriptimisvahendeid, sõltub tulemuste kuvamine skriptimisvahendist endast. Appium kuvab terminaliaknas vaid ühenduvuse ning veasõnumeid.

Ride vahendit kasutades kuvatakse tulemused jooksvalt kasutajaliidese logis ning HTML raportis.

Skriptimine ja muudatuste tegemine

Skriptimine ning muudatuste tegemine sõltub taaskord automatiseerija valitud vahendist.
Ride vahend võimaldab muuta Pythoni koodis ning tabeli näol testiskripte. Skriptide loomine ning uuendamine on intuitiivne ning kiire.

Skripti vastavus The Test Automation Manifesto kriteeriumitele

Skripti vastavust The Test Automation Manifesto kriteeriumitele on raske hinnata, kuna erinevad vahendid annavad erinevaid tulemusi. Antud tulemused on siiski toodud Ride vahendit kasutades:

- Kompaktne — Skriptist Pythonis ning tabelis on kerge aru saada kõigil meeskonnaliikmetel.
- Iseennast kontrolliv — testiraport kuvatakse automaatselt peale testi jooksutamist.
- Korratav — kasutades “loop-i” on võimalik jooksutada testi mitu korda.
- Robustne — testimine andis alati samu tulemusi.
- Selge — iga test ja rida selles on kõigile üheselt arusaadav.
- Praktiline — testilugude jooksutamine on kiire.
- Konkreetne — Ride raportis viidatakse puuduvale elemendile „EN“.
- Iseseisev — test jooksutab end ise.
- Hooldatav — testiskript on tabeli kujul kergesti arusaadav ja muudetav.
- Järgitav — skripti sidumist nõudega ei pakuta.

Plussid

Kuna Appium töötab vahelülina rakenduse ning testiskripti kirjutamiseks kasutatava vahendi vahel, siis võib automatiseerija valida vahendi, millega ta harjunud on, nagu näiteks Eclipse.

Miinused

Aeganõudev ülesseadmine- installida on vaja väga palju erinevaid osasid.

6.2 Calabash

Calabash on Xamarin poolt välja antud automatiseerimise raamistik, mille keel põhineb Rubyl. Calabash on mõelud funktsionaalsete testide automatiseerimiseks. Calabash on ainuke valitud automatiseerimisvahenditest, millel puudub GUI- testimine toimub terminaliaknas ning testiskriptide loomine vabalt valitud tekstieditoris, näiteks Notepadis. Xamarin pakub arendajaile mõeldud vahendit Xamarin Studiot, tasuta alla laadimiseks ning nii Windows kui iOS keskkondades.

Operatsioonisüsteemide toetus

Calabash toetab nii iOS kui ka Android platvormiülest testimist. Toetatud on nii iPhone, iPad kui ka iPod touch ning Android erinevad seadmed. Hetkel teisi operatsioonisüsteeme ei toetata.

Mobiilirakenduste toetus

Calabash toetab hübriid ning funktsionaalseid rakendusi. Antud hetkel veebirakendused ning HTML5 ei ole toetatud.

Füüsilise seadme ning emulaator/simulaator toetus

Automatiseerimine on võimalik nii füüsilise seadme kui ka emulaatori ja simulaatori peal.

Kasutatavad keeled

Calabashi skriptide loomise keeleks on Gherkin (põhineb Ruby-l), mis kasutab *Behaviour Driven Development (BDD)* filosoofiat, on domeeni spetsiifiline ning loetav ka mitteprogrammeerijatele. Gherkin keele näide on toodud joonisel 7, mis loob uue alarmi OnTime rakenduses.

```
  Feature: Notification feature
  Scenario: As a user I can set new notification
  And I press "Dismiss"
  And I press "addNewNotificationButton"
  Given I set the time to "11:00" on TimePicker with index 1 do
|11:00, 1|
```

```
And I press "button1"
Then I scroll down
Then I touch the "9" text
Then I touch the "Kunstiakadeemia - Kadaka" text
Then I touch the "Laki" text
And I press "saturdayC"
And I press "sundayC"
And I wait for 3 seconds
```

Joonis 7 Calabash skripti näide

Calabash on varustatud eeldefineeritud testisammudega Rubys, mis aitavad algajal automatiseerijal alustada, kuid suuremate ning keerukamate projektide jaoks pole need piisavad. Gherkinis loodud testid jooksutatakse Cucumber raamistikus.

Maksumus

Calabash on avatud lähtekoodiga tarkvara ning see on tasuta kättesaadav. Xamarin pakub muid tasulisi teenuseid, nagu Xamarin Test Cloud, mille vahendusel on võimalik testida sadadel erinevatel reaalsetel seadmetel või automatiseerimiskoolitused.

Lähtekoodi vajalikkus

Koodile ligipääs on vajalik ning mõningad muudatused on tarvis koodis teha iOS platvormile mõeldud rakenduste puhul. iOS operatsiooni puhul tuleb rakendusse lisada Xamarin Test Cloud Agent ning teha väiksemad muudatused koodis. Android rakendusel on tarvis lubada vaid interneti ligipääs koodile.

Seadme rooting/jailbraking vajadus

Rooting ning jailbroking vajadus puudub.

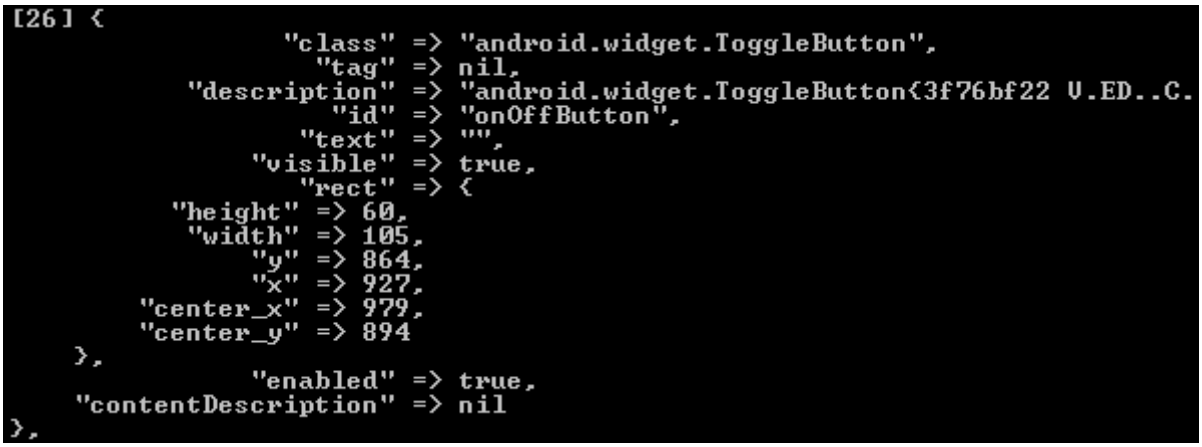
Testskriptide loomise võimalused

Salvestamine ning taasesitamine on võimalik vaid vanematel kui iOS 7 seadmetel andes käskluse Calabash konsoolil. Testsikripti võib kirjutada automatiseerijale sobilikus tekstieditoris, näiteks WordPad või Notepad. Testiskriptid salvestatakse kausta “Features”

laiendusega .feature. Testide jooksutamiseks Windows-il tuleb anda käsureal käsklus calabash-android run rakenduse_nimi.apk ning iOS-il cucumber.

Elementide määratlemine

Calabash toetab Query süntaksi, mis võimaldab luua dünaamilis testilugusid. Query kuvab parasjagu ekraanil olevate elementide id-d, klassid ning muud kirjeldused, mille kaudu saab testskriptis elementi määratleda. Seetõttu on tõeline objektide määratlemine toetatud. Näide Query süntaksiga saadud elemendi "Button" kirjetest on toodud joonisel 8.



Joonis 8 Elementide määratlemine Calabashis

Integreeritavus teiste programmidega

Integreeritavus pideva integratsiooni tööriistade nagu Jenkins, TFS ja TeamCity on olemas. Samuti on võimalik integreerida erinevate NUnit tööriistadega.

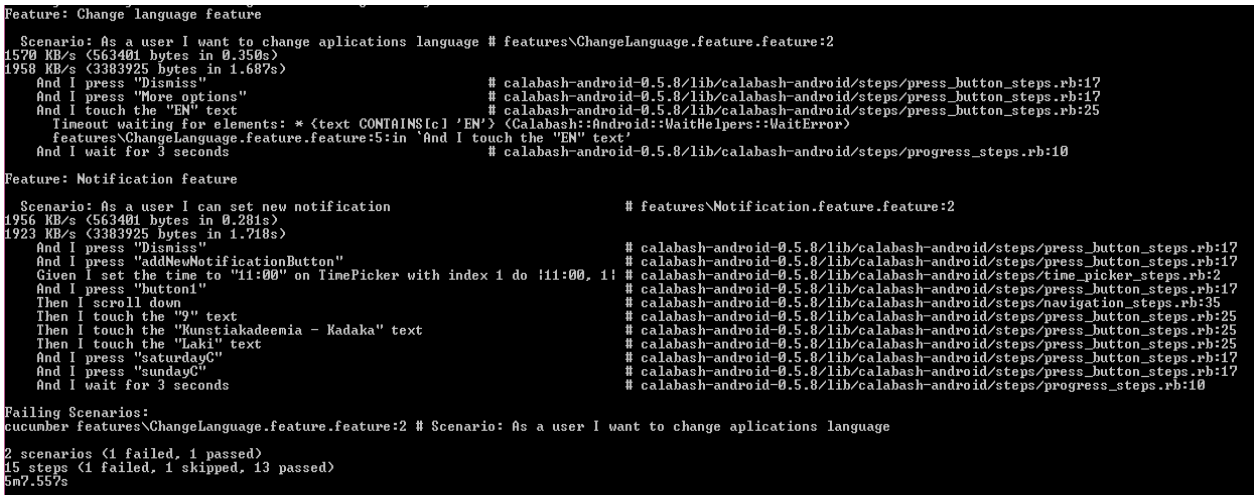
Mitme seadme tugi

Olemas on mitme seadme tugi, mis tähendab, et ühte testiskripti on võimalik jooksutada mitme seadme peal korraga.

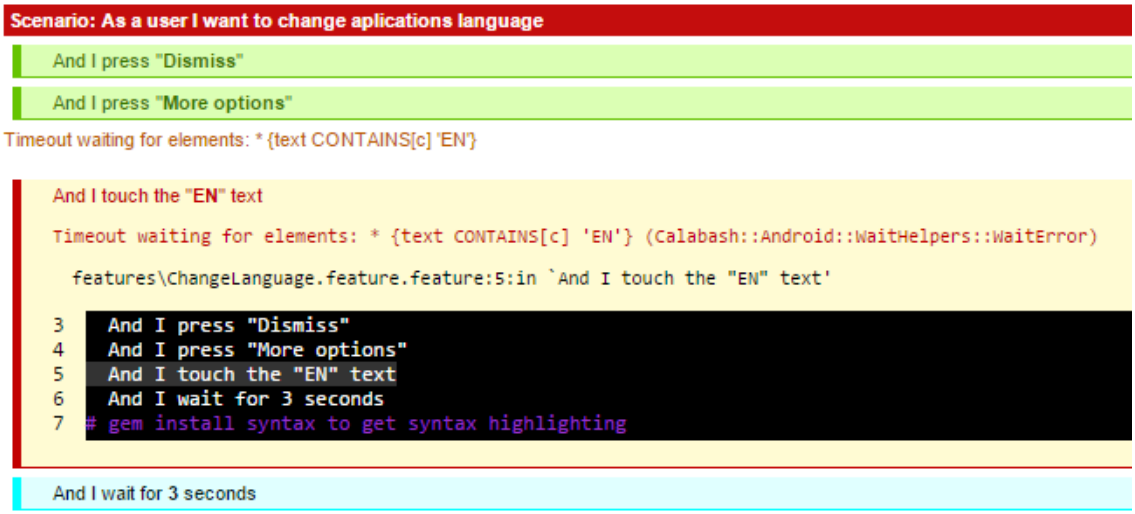
Tulemuste kuvamine

Tulemusi on võimalik kuvada kahel viisil — käsureal või eksportida HTML raportisse. Käsureal raport on kuvatud joonisel 9. HTML raport testiloo 001 kohta on kujutatud joonisel 10 ning testiloo 002 kohta joonisel 11.

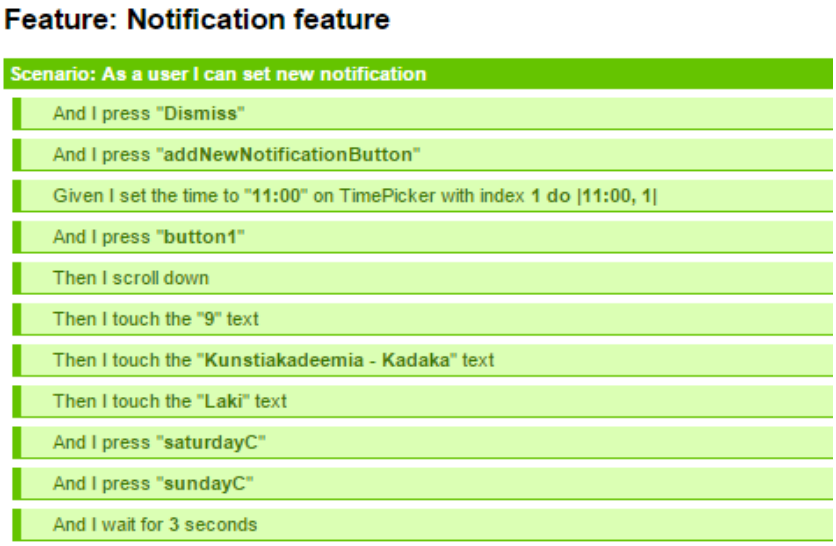
Joonistelt on näha, et Calabash ei leia esimese testiloo puhul, mis ebaõnnestuma pidi, elementi „EN" ning saab taimauti. Teine testilugu, mis peabki õnnestuma, on läbitud ilma vigadeta. Vea korral luuakse ka ekraanipilt ekraanist, kus viga tuvastati.



Joonis 9 Tulemuste kuvamine Calabashis käsureal



Joonis 10 HTML raport ebaõnnestunud testiloo kohta Calabashis



Joonis 11 HTML raport õnnestunud testiloo kohta Calabashis

Skriptimine ja muudatuste tegemine

Skriptis muudatuste tegemine on kerge ning intuitiivne, kui eeldefineeritud sammud on olemas. Kui eeldefineeritud Gherkin sammudest jääb puudu, on võimalik lisada Ruby libs kausta Rubys kirjutatud sammud, millega implementeeritakse Gherkinis antud käsklusi.
Näiteks on olemas eeldefineeritud Ruby samm kellaaja valmise kohta:

```
Given /^I set the time to "(\d\d:\d\d)" on TimePicker with index ([^\"]*)$/ do |time, index|
  set_time("android.widget.TimePicker index:#{index.to_i-1}", time)
end
```

Gherkinis käib kellaaja valik järgneva lausega:
Given I set the time to "11:00" on TimePicker with index 1 do |11:00, 1|

Tänu sellisele implementatsioonile on testidest arusaamine võimalik kõigil meeskonnaliikmetel ning skriptide uuendamine ja loomine intuitiivne.

Skripti vastavus The Test Automation Manifesto kriteeriumitele

Calabashis kirjutatud testiskript on:

- Kompaktne — iga meeskonna liige saab Gherkini käsklustest aru ja keel on lihtne.
- Iseennast kontrolliv — testiraport kuvatakse automaatselt peale testi jooksutamist.
- Korratav — kasutades “loop-i” on võimalik jooksutada testi mitu korda.
- Robustne — testide jooksutamine andis alati samu tulemusi.
- Selge — iga test ja rida selles on kõigile üheselt arusaadav.
- Praktiline — testilugude jooksutamine on kiire.
- Konkreetne — raportis viidatakse testiloo 001 puhul otse puuduvale elemendile “EN”.
- Iseseisev — test jooksutab end ise.
- Hooldatav — test on kergesti arusaadav ja muudetav.
- Järgitav — test on seotud nõudega, mida see testib.

Plussid

Suureks plussiks on Calabashi puhul see, et kolmandad osapooled, nagu kaamera, on toetatud. Gherkin keel tagab arusaamise testiskriptist kogu projektimeeskonnale. Calabashil on suur kogukond nii Android kui iOS kasutajaid Google Groups näol, kust abi küsimisel saab vastuse enamasti väga kiiresti. Testilugu on otseselt seotud selle nõudega (stsenaariumi näol).

Miinused

Veebirakendused ei ole endiselt toetatud. Testides vigast koodi (Test 001), annab Calabash taimauti, kuna püüab testiloos mainitud „EN“ elementi leida (teised testimise all olevad automatiseerimisvahendi kuvavad kohe, et element „EN“ on puudu).

6.3 MonkeyTalk

MonkeyTalk, endise nimega FoneyMonkey, on CloudMonkey LLC poolt loodud avatud lähtekoodiga automaatiseerimisvahend. Raamistik koosneb kahest tööriistast, MonkeyTalk IDE ning MonkeyTalk Agendid (iOS ja Android). MonkeyTalk IDE põhineb Eclipse'l ning pakub võimalusi funktsionaalsete testide salvestamiseks, muutmiseks ja ettemängimiseks.

Operatsioonisüsteemide toetus

MonkeyTalk toetab nii iOS (alates versioon 4.0) kui ka Android (alates versioon 2.2) operatsioonisüsteemi. Android seadet on võimalik ühendada nii WIFI kui USB vahendusel, iOS seadet vaid WIFI vahendusel. Platvormiülene testskriptide jooksutamine on võimalik loogiliselt identsete rakenduste vahel. iOS ja Android rakendused ei pea olema visuaalselt samasugused, vaid nende loogika peab ühtima.

Mobiilirakenduste toetus

Funktsionaalne- ja hübriid rakendus on toetatud täielikult. Veebirakendusi on võimalik testida kasutades MonkeyTalki brauser MTBrowser vahendit. Otse Safari ning Chrome brauserites pole võimalik testida, kuna MonkeyTalk Agent ei ole nendega lingitud.

Füüsilise seadme ning emulaator/simulaator toetus

Testimine on võimalik nii füüsilisel seadmel kui ka emulaatoril ja simulaatoril. Toetatud on erinevad iPhone mudelid, iPad, iPod touch ning enamus Android seadmed. Lisaks on võimalik testida ka *device pool* CloudMonkey LabManager-il, mis on varustatud erinevate mobiilseadmetega, kuid see teenus on juba tasuline.

Kasutatavad keeled

Tööriist kasutab MonkeyTalk keelt, mis on kergesti arusaadav ka eelneva programmeerimisoskuseta inimestele. MonkeyTalk keel on *keyword-driven*, mis tähendab, et kasutaja saab lisada ka ise keelde uusi käsklusi. Nagu mainitud eelpool, on MonkeyTalk keelt

võimalik eksportida ka JavaScripti ning muuta ja uuendada JavaScriptis. MonkeyTalk kasutajaliides koosneb kolmest osast, tabeli vaade (joonis 12), kus on sõnadena kirjas komponendid, ID-d ning tegevused, MonkeyTalk vaade (joonis 13), kus testiskript on kuvatud MonkeyTalk keeles ning JavaScript vaade (joonis 14), kus skript on JavaScriptis.

Row		Component	MonkeyID	Action	Arguments
1		Button	Dismiss	tap	
2		Image	addNewNotification...	tap	
3		Button	OK	tap	
4		Table	listView	SelectIndicator	18
5		Table	listView	SelectIndicator	4
6		Table	listView	SelectIndicator	5
7		Toggle	saturdayC	on	
8		Toggle	sundayC	on	

Joonis 12 MonkeyTalk skripti sisestus tabeli kujulujul

```
Button Dismiss tap
Image addNewNotificationButton tap
Button OK tap
Table listView SelectIndicator 18
Table listView SelectIndicator 4
Table listView SelectIndicator 5
Toggle saturdayC on
Toggle sundayC on
```

Joonis 13 MonkeyTalk skripti sisestus MonekyTalk keeles

```
load("libs/OnTime.js");


OnTime.Katse.prototype.run = function() {
   /**
    * @type MT.Application
    */
   var app = this.app;


   app.button("Dismiss").tap();
   app.image("addNewNotificationButton").tap();
   app.button("OK").tap();
   app.table("listView").selectIndicator("18");
   app.table("listView").selectIndicator("4");
   app.table("listView").selectIndicator("5");
   app.toggle("saturdayC").on();
   app.toggle("sundayC").on();
};
```

Joonis 14 MonkeyTalk skripti sisestus JavaScriptis

Maksumus

MonkeyTalk Community on avatud koodiga ning tasuta tarkvara. MonkeyTalk Professional on tasuline ning pakub lisaks erinevaid vahendeid, nagu sisse ehitatud emulaator või automaatne

instrumenteerimine (koodis muudatuste tegemine ning MonkeyTalk Agendi lisamine). Antud magistritöös käsitletakse tasuta MonkeyTalk Community vahendit.

Lähtekoodi vajalikkus

Lähtekoodile ligipääs on vajalik, kuna MonkeyTalki kasutamiseks on tarvis installida testitava rakenduse koodi MonkeyTalk Agent, mis võimaldab suhtlust MonkeyTalk IDE-ga. Hilisemaid muudatusi lähtekoodis teha pole tarvis, piisab kui Agent on installitud arenduse alguses loodava rakenduse koodi. Siiski enne poodi saatmist tuleks see lähtekoodist eemaldada.

Seadme rooting/jailbraking vajadus

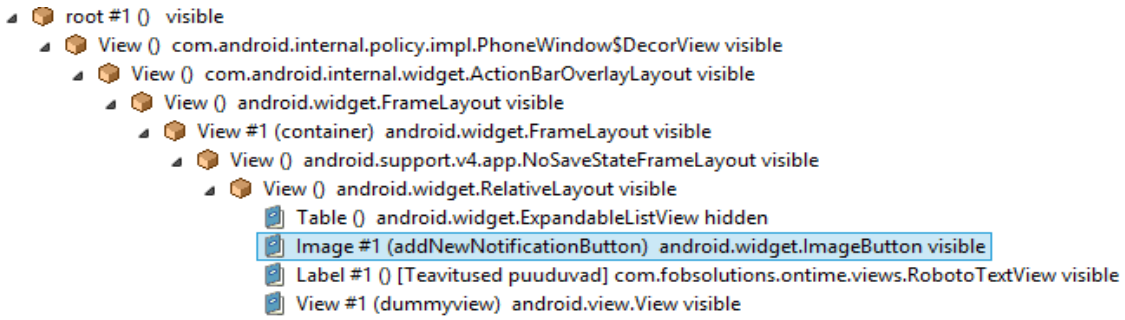
Testitavas seadmes pole tarvis muudatusi teha.

Testskriptide loomise võimalused

MonkeyTalk pakub salvestamise ja taasesitamise võimalust, kirjutades testskriptid tabelisse, MonkeyTalk keelde ning JavaScripti. Hiljem on võimalik neid muuta kus kasutaja ise soovib. Võimalik on kasutada ka omale sobilikku skriptide loomise vahendit ning jooksutada neid Ant või Java runneris. Andmepõhine *(data-driven)* testimine on võimalik CSV failide abil, mis võimaldab lihtsalt muuta iga iteratsiooni jaoks sisendeid.

Elementide määratlemine

Elementide määratlemine toimub tänu MonkeyTalk Agendile automaatselt kasutajaliideses. Selleks tuleb liikuda rakenduse ekraanile, mille elemente kuvada soovitakse ning valida MonkeyTalkis *"ComponentTree"*. MonkeyTalk kasutab tõelist objektide määratlemist—element tuvastatakse selle id ning klassi, mitte vaid graafilise tunnuse järgi. *ComponentTree* elementide määratlemise aken on kuvatud joonisel 15.



Joonis 15 Elementide määramine MonkeyTalkis

Integreeritavus teiste programmidega

MonkeyTalki on võimalik integreerida Jenkins'iga ning teiste programmidega, mis toetavad xUnit standardit.

Mitme seadme tugi

Testimine mitmel seadmel korraga on toetatud.

Tulemuste kuvamine

Tulemusi kuvatakse MonkeyTalk kasutajaliideses jooksvalt logina. Samuti luuakse kasutajaliideses kausta "Reports" alla HTML raportid, joonis 16 kujutab testiloo 001 tulemust HTML vaates.

Joonis 16 MonkeyTalk tulemuste kuvamine HTML raportis

Logina antakse sama tulemus järgnevalt:

16:31:38.623: Button Dismiss tap
16:31:39.174: Image "More options" tap
16:31:39.794: Menu * select EN
16:31:42.568: FAILURE: Menu selection failed. Unable to find item with title: EN
16:31:42.604: Completed Script Playback - FAILURE Menu selection failed. Unable to find item with title: EN

Skriptimine ja muudatuste tegemine

Skriptimine on MonkeyTalk abil kiire, kuna salvestamine ning järgi mängimine ei nõua skripti kirjutamiseks aega. Muudatuste tegemine on samuti kerge, kuna kasutajaliideses pakub võimalust luua uuendiso nii tabeli kujul, MonkeyTalk keeles ning soovi korral ka JavaScriptis. Testides OnTime salvestamine ning järgi mängimine ei salvestanud 100% kõiki samme, seega tuli manuaalselt lisada samme juurde.

Skripti vastavus The Test Automation Manifesto kriteeriumitele

MonkeyTalkis kirjutatud testiskript on:

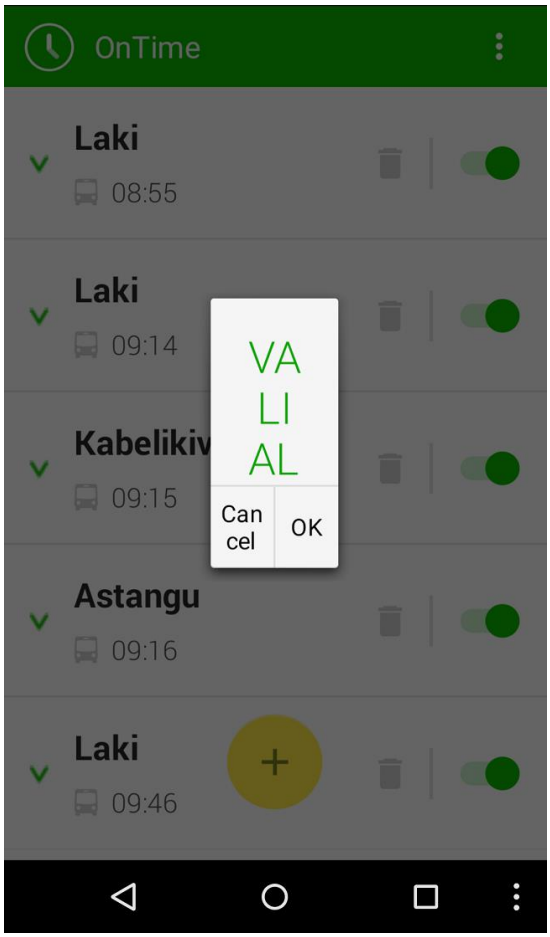
- Kompaktne — skripti iga rida loob tegevuse.
- Iseennast kontrolliv — test raporteerib tulemused automaatselt.
- Korratav — JavaScriptis on võimalik luua loop-e, et korrata testi.
- Robustne — testimine andis alati samu tulemusi.
- Selge — iga test on arusaadav kogu meeskonnale MonkeyTalk kerge keelekasutuse tõttu.
- Praktiline — testilood füüsilisel seadmel jooksevad kiiresti ilma liigse ajakuluta. Emulaatorit ei õnnestunud OnTime testimisel tööle saada.
- Konkreetne — testilugu 001 andis teate, et elementi “EN” ei leitud.
- Iseseisev — test jooksutab end ise.
- Hooldatav — test on kergesti arusaadav ja muudetav.
- Järgitav — sellist võimalust MonkeyTalk ei paku.

Plussid

Väga kergesti muudetavad ja lühikesed skriptid ning arusaadav kasutajaliides. Raportid kuvatakse jooksvalt kasutajaliideses ning HTML raportitena. Vigane testilugu 001, mis peab ebaõnnestuma, annab kohe märguande vea kohta.

Miinused

Väga keeruline ülesseadmine *(set-up)* projektidele, mis on tehtud Android Studios, kuna MonekyTalk põhineb Eclipse-l. OnTime rakendus on loodud Android Studios ning peale instrumentimist MonkeyTalk Agendiga, ei kuvatud rakendust seadmes korralikult, ajavaliku funktsionaalsust ei ole võimalik üldse kasutada (joonis 17). Tööriista suurimaks miinuseks on see, et kolmandate osapoolte testimine (kaamera, brauser jne) pole toetatud.



Joonis 17 OnTime peale instrumentimist MonkeyTalk Agendiga

6.4 SeeTest

SeeTest on Experitest Ltd poolt välja antud automatiseerimisvahend, mis antud magistritööks valitud vahenditest on ainukesena tasuline. Ainukesena valitud automatiseerimisvahenditest pakub SeeTest võimalust luua skripte ka näiteks WindowsPhone või BlackBerry operatsioonisüsteemide rakendustele.

Operatsioonisüsteemide toetus

SeeTest toetab iOS, Android, Windows Phone, BlackBerry, Windows Mobile ning Symbian opertsioonisüsteemis automatiseerimist. Platvormiülene testimine on samuti võimalik.

Mobiilirakenduste toetus

Toetatud on kõik kolm rakenduse tüüpi- funktsionaalne-, hübriid- ja veebirakendus. SeeTest võimaldab kasutada ka kolmandaid osapooli nagu kaamera või veebibrauser.

Füüsilise seadme ning emulaator/simulaator toetus

SeeTest toetab testimist nii füüsilise seadme kui ka emulaatori ning simulaatori peal. Emulaatoril salvestus ja taasesitamise võimalus puudub.

Kasutatavad keeled

SeeTest skriptimiskeel on domeenipõhine ning koosneb käsklustest, mis antakse rakendusele ette. Joonis 18 kujutab endast SeeTestis kirjutatud OnTime testilugu 002, kus luuakse uus alarm.

1	☑	Set device adb:Nexus 5
2	☑	Click on NATIVE : xpath=//*[@text='OnTime']
3	☑	Click on NATIVE : text=Dismiss
4	☑	Click on NATIVE : id=addNewNotificationButton
5	☑	Set time = 11.00 from element NATIVE : xpath=//*[@class='android.widget.TimePicker' and @hidden='false']
6	☑	Click on NATIVE : text=OK
7	☑	Click on NATIVE : text=Kadaka - Kunstiakadeemia
8	☑	Click on NATIVE : text=Kunstiakadeemia - Kadaka
9	☑	Click on NATIVE : text=Laki
10	☑	Click on NATIVE : text=L
11	☑	Click on NATIVE : text=P
12	☑	Click on NATIVE : xpath=(//*[@class='android.widget.RelativeLayout'])[2]
13	☑	

Joonis 18 SeeTest skripti näide

SeeTest võimaldab testiskripti konverteerida erinevatesse programmeerimiskeeltesse, nagu Java, Ruby, Perl, Python, C# ja muud. Selle abil on võimalik siduda SeeTest keskkonda erinevate pideva integratsiooni keskkondadega ning jooksutada skripte ka muudes keskkondades, nagu Eclipse.

Maksumus

SeeTest maksab 3500 dollarit aastas (umbes 3110 eur). Pakett sisaldab endast toetust Android, iOS, BlackBerry ning WindowsPhone operatsioonisüsteemidele, plug-ine UFT (QTP), RFT, TestComplete, C#, MSTest/VisualStudio/TFS, Java, Perl, Python, Ruby, Selenium-ile, e-mail supporti ning 5-seadme ühenduvust (järjestikuselt). Lisadena on võimalik osta erinevate keelte toetust 500 dollari eest aastas (see hõlmab ka eesti keelt), plug-ini seadmete samaaegseks testimiseks (17500 dollarit aastas) ning installerit OSX jaoks (1000 dollarit aastas).

Lähtekoodi vajalikkus

SeeTesti puhul ei ole vaja ligipääsu lähtekoodile ning koodis muudatuste tegemise järele vajadus puudub.

Seadme rooting/jailbraking vajadus

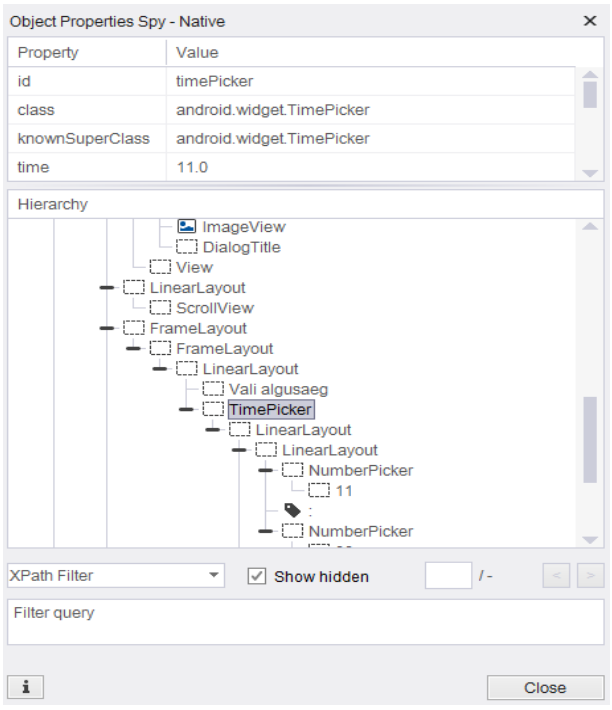
Seadme rooting/jailbraking vajadus puudub.

Testskriptide loomise võimalused

SeeTest on mõeldud eelkõige töötamaks salvestus ja taasesitamise tööriistana. Automatiseerija saab aga ka ise skripte või samme skripti tabelisse lisada. Skriptide loomine toimub SeeTest graafilises kasutajaliideses.

Elementide määratlemine

Elementide määratlemine SeeTestis toimub kahel moel, automaatselt ning manuaalselt instrumentides. Manuaalselt saab seda teha käsurealt ning seda võimalust tuleks kasutada kui rakendus sisaldab elemente, mis on privaatsussätete tõttu peidetud. Enamus elemente saab SeeTest kätte automaatselt. Elementide klassid, id- ja muud omadused kuvatakse SeeTest graafilises kasutajaliideses. SeeTesti määratleb elemendid nähtava osa järgi, nagu pildid ja teksti ning ka elemendi id järgi. Näide SeeTesti elementide määratlemisest on toodud joonisel 19.



Joonis 19 Elementide määratlemine SeeTestis

Integreeritavus teiste programmidega

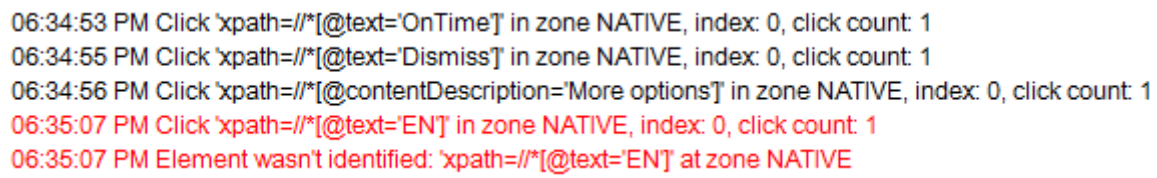
SeeTesti on võimalik integreerida erinevate pideva integratsiooni vahenditega, nagu Jenkins.

Mitme seadme tugi

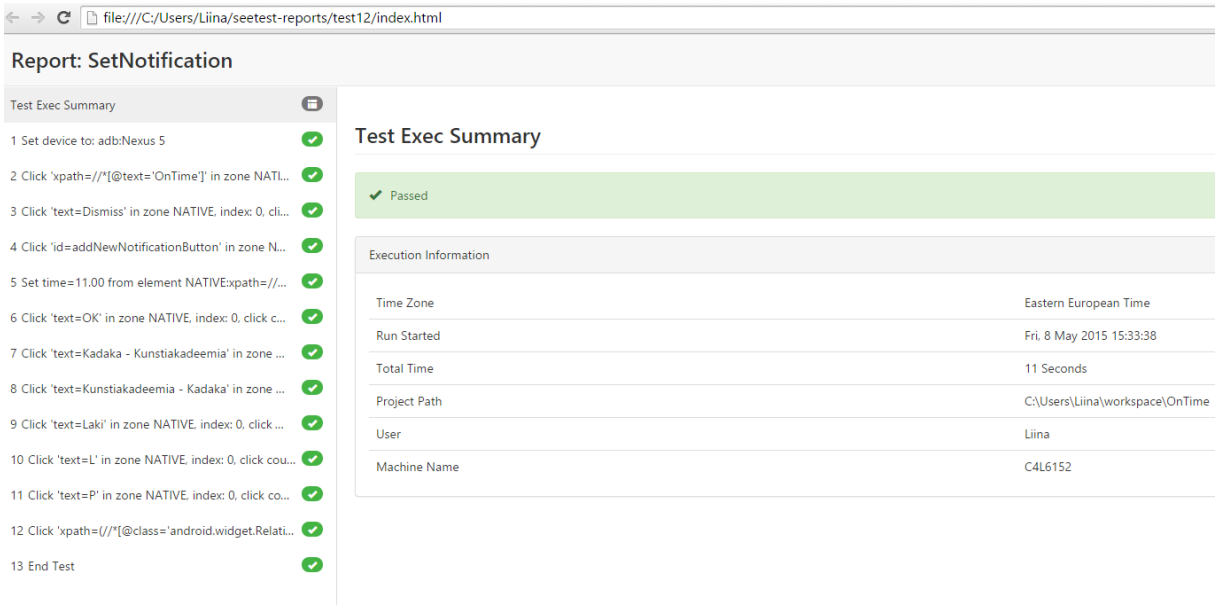
SeeTest pakub mitme seadme tuge- teste saab korraga jooksutada, see aga on omakorda tasuline teenus.

Tulemuste kuvamine

Tulemusi kuvatakse nii SeeTest graafilises kasutajaliideses logina kui ka HTML kujul. Joonis 20 kujutab endast SeeTest raportit graafilises kasutajaliideses testiloo 001 kohta, kus testilugu peab ebaõnnestuma, joonis 21 kujutab HTML raportit testiloo 002 kohta, mis peab õnnestuma.



Joonis 20 SeeTest raport kasutajaliideses



Joonis 21 SeeTest raport HTML-is

Skriptimine ja muudatuste tegemine

SeeTest põhineb salvestamisel ja taasesitamise funktsioonil. Muuta saab skripti vaid tabelis, kuid seda saab teha kasutades vaid domeenipõhiseid SeeTest käsklusi. Muudatusi sisse viia on kerge.

Skripti vastavus The Test Automation Manifesto kriteeriumitele

SeeTestis kirjutatud skript on:

- Kompaktne — lühike tabeli näol skript.
- Iseennast kontrolliv — test raporteerib tulemused automaatselt.
- Korratav — ei leidnud viisi kuidas korrata ilma inimsekkumiseta teste.
- Robustne — test annab alati samu tulemusi.
- Selge — iga test ja rida selles on üheselt arusaadav kogu meeskonnale.
- Praktiline — testide jooksutamine on kiire.
- Konkreetne — testilugu 001 puhul viidati logis, et teksti “EN” ei suudetud leida.
- Iseseisev — test jooksutab end ise.
- Hooldatav — test on kergesti arusaadav ja muudetav kõigi poolt.
- Järgitav — seda võimalust ei pakuta.

Plussid:

Väga kerge ülesseadmine. Vahendit saab kasutada ka manuaalne testija, kes varasemalt pole automatiseerimisega kokku puutunud.

Miinused:

Kõrge hind. Skripte ei saa kirjutada ise vaid ainult eksportida soovitud keelde.

6.5 Järeldus

Tabel 2 Automatiseerimisvahendite tulemused

Kriteerium	**Appium**	**Calabash**	**MonkeyTalk**	**SeeTest**
Operatsiooni-süsteemide toetus	Android iOS Firefox OS	Android iOS	Android iOS	Android iOS Windows Phone BlackBerry Symbian

Rakenduste toetus	Funktsionaalne rakendus Hübriid rakendus Veebirakendus	Funktsionaalne rakendus Hübriid rakendus	Funktsionaalne rakendus Hübriid rakendus Veebirakendus MonkeyTalk brauseril	Funktsionaalne rakendus Hübriid rakendus Veebirakendus
Seadme toetus	Füüsiline seade Simulaator Emulaator	Füüsiline seade Simulaator Emulaator	Füüsiline seade Simulaator Emulaator	Füüsiline seade Simulaator Emulaator
Kasutatavad keeled	Java, Objective-C, JavaScript PHP Ruby Python C#	Gherkin Ruby	MonkeyTalk JavaScript	Domeenipõhine keel
Maksumus	Vabavaraline	Vabavaraline	Vabavaraline	3500$ aastas
Lähtekoodile ligipääs	Pole vajalik	Vajalik iOS rakendustel	Vajalik	Pole vajalik
Seadme rooting/ jailbraking vajadus	Puudub	Puudub	Puudub	Puudub
Testskriptide loomise võimalused	Salvestus ja taasesitamine (iOS versioonist Appiumil) Skriptimine	Skriptimine	Salvestus ja taasesitamine Skriptimine	Salvestus ja taasesitamine
Elementide määratlemine	Appium Inspector Tõeline objekti määratlemine	Query süntaks abil Tõeline objekti määratlemine	Graafilise kasutajaliidesega Tõeline objekti määratlemine	Graafilise kasutajaliidesega Tõeline objekti määratlemine
Integreeritavus teiste programmidega	Jenkins, Travis jne.	Jenkins, TFS, TeamCity jne.	Jenkins, xUnit	Jenkins, Eclipse jne.
Mitme seadme tugi	Olemas	Olemas	Olemas	Olemas
Tulemuste kuvamine	Oleneb valitud skriptimise vahendist	Jooksvalt terminalis HTML raport	Jooksvalt kasutajaliideses logina HTML raport	Jooksvalt kasutajaliideses logina HTML raport

Muudatuste tegemine	Oleneb valitud skriptimise vahendist	Intuitiivne. Kui eeldefineeritud sammud olemas, arusaadav kõigile.	Kasutajaliideses. Mugav ning intuitiivne.	Kasutajaliideses. Mugav ning intuitiivne.
Skripti vastavus The Test Automation Manifesto kriteeriumitele	Oleneb valitud skriptimise vahendist	95% Taimaut vea tõttu võtab aega rohkem, kui peaks kui taimaut pole defineeritud	90% Skripti pole võimalik siduda nõudega	80% Skripti pole võimalik siduda nõudega. Skripti ei saa korrata automaatselt

OnTime rakenduse testimiseks valiti Calabash ning seda eelkõige järgnevatel põhjustel:

- Testid on otseselt seotud nõudega *Scenario* kirjelduse abil.
- *Behaviour Driven Development* ning Gherkin keel annavad võimaluse testiskriptidest aru saada kõigil Scrum meeskonna liikmetel.
- Gherkin keeles on võimalik automatiseerida ka algajal automatiseerijal.
- Testiskriptid on lühikesed ja kompaktsed, iga rida täidab mingit eesmärki.
- Google Groups ja suur kogukond nii Android kui ka iOS platvormile, küsimustele reageeritakse väga kiiresti.
- Elementide määratlemine toimub elemendi tõelise kirje, id või klassi järgi. See muudab testid vähem lõhutavateks, kui muudatus peaks tulema graafilises pooles.
- Testimine Calabashis on robustne — testide jooksutamine annab alati samu tulemusi.
- Calabashi on võimalik siduda pideva integratsiooni vahendite, nagu Jenkins-iga.
- On vabavaraline.
- Toetab nii füüsilisel seadmel kui ka emulaatoril ja simulaatori testimist ning teeb seda mõlemal korrektselt.
- Toetab mitmel seadmel testimist korraga (*multi device* tugi).
- Testiskriptis on kerge muudatusi teha ning kuna skriptid on eraldi notepad vormingus, saab luua erinevaid versioone hõlpsasti (küll pole nende haldus ehk kõige kasutajasõbralikum).

Miinuspoolena võib tuua välja taimauti, mis Calabash annab, kui test ebaõnnestub (testilugu 001). Taimauti saab Calabashis määrata, seega see testimist ei sega.

Projektispetsiifiliselt ei sobi OnTime testimiseks MonekyTalk, kuna ei toeta kolmandaid osapooli (Android kella) ning koodi muudatused ning Agendi lisamine OnTime koodi muudavad rakenduse kasutuskõlbmatuks.

Kuna OnTime on vabavaraline rakendus, pole automatiseerimise jaoks võimalik kasutada ka SeeTest vahendit, kuna oleks liialt suur kulu. Miinuspoolena veel pole võimalik SeeTest-is kirjutada testiskripte ise, toetatud on vaid salvestus ja taasesitamine skriptimise viis.

Appiumi on agiilsuse poole pealt keeruline hinnata, kuna skriptide pikkus ja haldus olenevad sellest, millise vahendi skriptimiseks automatiseerija valib. Appium pakub küll rohkem valikuid, nagu veebirakenduse testimine ning rohkemad toetatud keeled, kuid valituks ei osutunud Appium, kuna erinevate osapoolte kasutamine (skriptimisvahend, Appium server, testitav rakendus), lisavad keerukust arusaamisel, miks test ebaõnnestub. Ride vahendit kasutades pole võimalik ka siduda nõuet testskriptiga.

7. Kokkuvõte

Antud magistritöö eesmärgiks oli valida automatiseerimisvahend Scrum projektis funktsionaalsele Android mobiilirakendusele OnTime. Eesmärgi täitmiseks tehti järgevad tegevused:

- Loodi ülevaade agiilsest tarkvaraarendusest ning uuriti lähemalt, milliseid väljakutseid see esitab automatiseerimisele.
- Viidi läbi intervjuu FOB Solutions automatiseerija, Indrek Kahuga, et saada reaalsemat ülevaadet automatiseerimisest ning nõuetest, mida vahendid peaksid tagama.
- Teoreetilise materjali ning läbiviidud intervjuu põhjal koostati kriteeriumid, mida automatiseerimisvahend peaks täitma.
- Võeti kasutusele neli automatiseerimisvahendit, Appium, Calabash, MonekyTalk ning SeeTest. Need vahendid valiti, kuna toetavad kõik Android ning iOS operatsioonisüsteeme.
- Võrreldi automatiseerimisvahendeid vastu loodud kriteeriume.
- Automatiseerimisvahenditel loodi kaks skripti, millest üks pidi ebaõnnestuma ning teine õnnestuma. Seda selleks, et saada ülevaadet, kuidas vahendid käituvad kahe stsenaariumi puhul ning kui hea on tulemuste kajastamine.

Tulemusena valiti välja Calabash kuna vastas kõige enam agiilsuse nõuetele ning sobis enim projekti. Calabashi suurimateks plussideks on:

- Otsene nõude sidumine testiskriptiga stsenaariumi kirjeldamise näol.
- Lühikesed, arusaadavad skriptid- iga rida skriptis täidab mingi käskluse.
- Gherkin keel, mida kasutatakse Calabashis skriptide loomiseks ning mis on arusaadav ka mitteprogrammeerijale.
- Testitulemused olid alati konstantsed.
- Nii füüsilise seadme kui ka emulaatori ning simulaatori toetus.
- Mitme seadme tugi- sama testi on võimalik jooksutada korraga mitmel seadmel.
- Integreeritavus pideva intergatsiooni tööriistadega, nagu Jenkins.
- Elementide määratlemine käib tõelise objekti tunnuste järgi.

Calabashi suurimaks miinuseks võib tuua välja taimauti, mis anti ebaõnnestunud testiloo puhul. Calabashis saab seada taimauti pikkust, seega see ei jää testimist segama.

Summary

Current Master's thesis objective was to select best automation tool for native Android application, OnTime, in Scrum project. Next actions were made in order to fulfill the purpose:

- An overview of Agile development and the challenges it creates for automation, was made.
- An interview with Indrek Kahu, FOB Solutions automation tester, was conducted in order to get better overview what is needed by automation tools.
- Based on analysis of theoretical part and interview, criteria was created, what automation tool should fulfill in Agile environment.
- Four automation tools, Appium, Calabash, MonkeyTalk and SeeTest, were taken into use. These four tools were selected as they all support Android and iOS operationsystems.
- These tools were compared against created criteria.
- Two scripts, one that fails and the other that passes, were created on selected tools. The purpose of that was to have an idea, how tools react and report results in both scenarios.

From four automation tools Calabash was selected as best tool for OnTime project for the next reasons:

- It Describes requirement for script through "Scenario" feature.
- It has short scripts that are easily created and changed.
- Calabash uses Behaviour Driven Development and Gherkin language that all project members will undestand.
- Test results are always constant.
- Support for physical device and also emulator and simulator.
- Multi-device support.
- Support for continious integration tools like Jenkins.
- Element recognition is done by true-object recognition.

Biggest drawback is the timeout that is given for testcase 001. As Calabash allows to set timeout lenght, it will not disturb testing.

Kasutatud kirjandus

1. Principles behind the Agile Manifesto [WWW] http://www.agilemanifesto.org/principles.html (02.03.2015)
2. What is Agile model- advantages, disadvantages and when to use it? [WWW] http://istqbexamcertification.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/ (02.03.2015)
3. Learn about Scrum https://www.scrumalliance.org/why-scrum (12.05.2015)
4. What is software testing? [WWW] http://istqbexamcertification.com/what-is-a-software-testing/ (02.03.2015)
5. Manual testing VS automated testing [WWW] http://www.slideshare.net/Softwarecentral/manual-testing-vs-automated-testing (07.03.2015)
6. Agile test automation [WWW] http://www.satisfice.com/articles/agileauto-paper.pdf (07.03.2015)
7. What are testing and agile development activities? [WWW] http://istqbexamcertification.com/what-are-testing-and-agile-development-activities/ (08.03.2015)
8. Agile testing: the changing role of testers [WWW] http://www.allaboutagile.com/agile-testing-the-changing-role-of-testers/ (08.03.2015)
9. More agile testing: learning journeys for the whole team / Janet Gregory, Lisa Crispin. First edition. Indiana : RR Donnelley, 2014.
10. Refactoring Test Code [WWW] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.486&rep=rep1&type=pdf (15.03.2015)
11. The Test Automation Manifesto [WWW] http://xunitpatterns.com/~gerard/xpau2003-test-automation-manifesto-paper.pdf (15.03.2015)
12. Smartphone OS Market Share, Q4 2014 [WWW] http://www.idc.com/prodserv/smartphone-os-market-share.jsp (15.03.2015)
13. Frequently aksed questions [WWW] http://source.android.com/source/faqs.html (15.03.2015)
14. Google Play has more apps than Apple now, but it's still behind in one key area [WWW] http://www.businessinsider.com/google-play-vs-apple-app-store-2015-2 (19.03.2015)

15. Android, the world's most popular mobile platform [WWW] http://developer.android.com/about/index.html (15.03.2015)
16. Android history [WWW] http://www.android.com/history/ (15.03.2015)
17. iOS: A visual history [WWW] http://www.theverge.com/2011/12/13/2612736/ios-history-iphone-ipad (19.03.2015)
18. Apple to release iOS 8.2 with Apple Watch support on Monday [WWW] http://appleinsider.com/articles/15/03/09/apple-releases-ios-82-with-apple-watch-support (19.03.2015)
19. Apple's iOS App Store reaches record 7,8M daily downloads [WWW] http://appleinsider.com/articles/14/11/24/apples-ios-app-store-reaches-record-78m-daily-downloads (19.03.2015)
20. Mobile: Native Apps, Web Apps, and Hybrid Apps [WWW] http://www.nngroup.com/articles/mobile-native-apps/ (19.03.2015)
21. Selecting the right mobile test automation strategy: challenges and principles [WWW] http://www.cognizant.com/InsightsWhitepapers/Selecting-the-Right-Mobile-Test-Automation-Strategy-Challenges-and-Principles.pdf (24.03.2015)
22. Mobile app testing in an agile environment - testobject [WWW] https://testobject.com/blog/2013/11/mobile-app-testing-in-agile-environment.html (24.03.2015)
23. Software testing types [WWW] http://www.aptest.com/testtypes.html (29.03.2015)
24. What is Monkey Testing [WWW] http://www.exforsys.com/tutorials/testing-types/monkey-testing.html (29.03.2015)
25. Top 10 Mobile Application Testing Automation Tool Requirements https://northwaysolutions.com/blog/top-10-mobile-application-testing-automation-tool-requirements/#.VVDEp_mqpBe (11.05.2015)
26. Intervjuu Indrek Kahu'ga, FOB Solutions OÜ arendaja ning automatiseerija