Lappeenrannan teknillinen yliopisto
School of Business and Management
Tietotekniikan koulutusohjelma

Kandidaatintyön loppuraportti

Ilari Sahi

ANDROID-SOVELLUSTEN TESTAUSAUTOMAATIO PK-YRITYKSESSÄ

Työn tarkastaja: TkT Ari Happonen

Työn ohjaajat: TkT Ari Happonen
Tuotantojohtaja, DI Ilkka Toivanen

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
School of Business and Management
Tietotekniikan koulutusohjelma

Ilari Sahi

Android-sovellusten testausautomaatio pk-yrityksessä

Kandidaatintyö

2018

46 sivua, 7 kuvaa, 5 taulukkoa, 1 liite

Työn tarkastaja: TkT Ari Happonen

Hakusanat: Android, testausautomaatio, DevOps
Keywords: Android, test automation, DevOps

Tämän kandidaatintyön tarkoituksena on tutkia parhaita käytänteitä Android-sovellusten testausautomaation toteuttamiseen ohjelmistoalan pk-yrityksessä. Työssä tutkitaan myös, miten testausautomaatio kytkeytyy DevOps-kulttuuriin. Tutkimusmenetelminä käytetään kirjallisuuskatsausta, kyselytutkimusta ja empiiristä tutkimusta. Empiirisessä tutkimuksessa evaluoidaan suosituimpia testausautomaation työkaluja.

Työn tuloksena saatiin käsitys, miten Android-sovellusten testausautomaatio kannattaa toteuttaa pk-yrityksessä. Tärkeäksi automatisointiin kannustavaksi tekijäksi nousi ohjeistus työkalujen käytöstä. Testausautomaation käyttöä kannattaa harkita projektikohtaisesti sen aloittamisen vaativien kustannusten takia. Työssä valittiin myös testaukseen suositellut työkalut. Testausautomaatio havaittiin olennaiseksi osaksi DevOps-kulttuuria.

ABSTRACT

Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science

Ilari Sahi

Test automation of Android applications in an SME

Bachelor's Thesis

46 pages, 7 figures, 5 tables, 1 appendix

Examiner: D.Sc. Ari Happonen

Keywords: Android, test automation, DevOps

The goal of this bachelor's thesis is to research best practices to automate testing of Android applications in an SME in the software industry. This thesis also examines how test automation is linked to the DevOps culture. Research methods used in this thesis are literature review, survey and empirical research. The most popular test automation tools are evaluated empirically.

As a result of this thesis, an understanding of how to implement test automation for Android applications in an SME was gained. Instructions for tool usage was identified as an important supportive factor for usage of automation. Use of test automation should be a project-specific decision due to its initial costs. Preferred test automation tools were also chosen in this thesis. Test automation was discovered to be a critical part of the DevOps culture.

SISÄLLYSLUETTELO

1 JOHDANTO ... 3

1.1 TYÖN TAUSTA ... 3

1.2 TYÖN TAVOITTEET JA RAJAUKSET ... 4

1.3 TYÖN RAKENNE ... 5

1.4 OCTO3 OY YRITYSESITTELY ... 6

2 OHJELMISTOJEN TESTAUSAUTOMAATIO ... 7

2.1 TESTAUSAUTOMAATIO YLEISESTI ... 7

2.2 DEVOPS JA TESTAUS ... 11

2.3 TESTAUSAUTOMAATIO ANDROID-ALUSTALLA ... 13

2.4 TESTILAITTEET PILVIPALVELUNA ... 14

3 TUTKIMUKSEN TOTEUTUS ... 16

3.1 KIRJALLISUUSKATSAUS ... 16

3.2 KYSELYTUTKIMUS ... 16

3.3 EMPIIRINEN TUTKIMUS ... 17

3.4 REFLEKTIO TUTKIMUKSEN TOTEUTUMISESTA ... 19

4 KYSELYN TULOKSET ... 21

5 TYÖKALUJEN EVALUOINTI ... 25

5.1 TUTKIMUSYMPÄRISTÖ ... 26

5.2 TUTKIMUKSEN TESTITAPAUKSET ... 27

5.3 EVALUOINNIN TULOKSET ... 29

6 JOHTOPÄÄTÖKSET ... 32

LÄHTEET ... 36

LIITTEET

SYMBOLI- JA LYHENNELUETTELO

API	Application programming interface
CI	Continuous integration
DevOps	Development & Operations
GUI	Graphical user interface
MTaaS	Mobile Testing-as-a-Service
Oy	Osakeyhtiö
Pk-yritys	Pieni tai keskisuuri yritys
TDD	Test-driven development
UI	User interface
UX	User experience

1 JOHDANTO

Asiakkaat eivät usein tiedä mitä he haluavat, koska eivät ole tietoisia nykypäivän mahdollisuuksista (Cagan 2006). Ohjelmistoalan yritysten pitäisi pystyä näyttämään asiakkaille mahdollisimman nopeasti uusia ratkaisuja ja iteroida tuotetta seuraavaan versioon palautteen perusteella. Ohjelmistotuotannon ketterät menetelmät ovat pyrkineet vastaamaan tähän haasteeseen ja ovatkin jossain määrin onnistuneet ratkaisemaan perinteisen ohjelmistotuotannon ongelmia.

Testausautomaatio on yksi modernin ohjelmistotuotannon kulmakivistä. Testausautomaation avulla ohjelmiston laadusta saadaan jatkuvasti tietoa, mikä edesauttaa nopeaa reagointia virhetilanteisiin. (Gotimer & Stiehm 2016, s. 13.) Testausautomaatio lisää myös luottamusta ohjelmiston laatuun ja tuotantovalmiuteen, jolloin asiakkaalle voidaan esitellä nopeaan tahtiin uusia versioita ohjelmistosta (Macharla 2017, s. 20). Tässä työssä tarkastellaan muun muassa testausautomaatiota Android-alustalla. Android on ohjelmistoalusta, joka sisältää käyttöjärjestelmän ja tärkeitä perusohjelmia. Androidia hyödynnettiin aluksi mobiililaitteilla, mutta nykyään se on käytössä myös älykelloissa, autoissa ja televisioissa. Google osti Androidin vuonna 2005. (Hoog 2011, s. 3–4; Sundar 2015.)

1.1 Työn tausta

DevOps tulee sanoista Development & Operations. Nimensä mukaisesti DevOps-kulttuurin tarkoituksena on yhdistää tuotteen kehitys- ja tuotantohaarat kokonaisuudeksi, jolloin pystytään tekemään nopeita päätöksiä ja ripeää ohjelmistokehitystä molemminpuolisen vuoropuhelun avulla. DevOps voidaan kuvata joukkona menetelmiä, jotka automatisoivat ohjelmistotuotantoprosessia. (Ravichandran, Taylor & Waterhouse 2016, s. 10–11; Atlassian 2017; Toivanen 2017.) DevOps-kulttuuri on kasvavassa suosiossa ja yritykset palkkaavat mielellään DevOps-asiantuntijoita töihin (Roche 2013, s. 40–41).

Jatkuva integraatio (continuous integration, eli CI) on olennainen osa DevOps-kulttuuria. Jatkuvassa integraatiossa sovelluskehittäjät integroivat ohjelmiston moduuleja toisiinsa ja

koostavat ohjelmiston, jonka toimivuus todennetaan testeillä päivittäin (Ståhl & Bosch 2014, s. 108). Testausautomaatio taas on olennainen osa jatkuvaa integraatiota (Ebert, Gallardo, Hernantes & Serrano 2016, s. 99).

Tämän työn aihe on peräisin OCTO3 Oy:ltä, joka on ohjelmistoalan pk-yritys (pieni tai keskisuuri yritys). OCTO3 Oy pyrkii kehittämään DevOps-menetelmiään tuotteiden laadun ja asiakastyytyväisyyden parantamiseksi. Yritys on mukana useissa mobiilialustoille kohdennetuissa projekteissa, joten kandidaatintyön puitteissa on luonnollista tutkia testausautomaatiota Android-alustalla. OCTO3 Oy hyödyntää DevOps-menetelmiä projektikohtaisesti, mutta mitään vakiintuneita käytänteitä mobiilikehityksessä ei ole vielä olemassa.

1.2 Työn tavoitteet ja rajaukset

Tämän työn tavoitteena on kartoittaa parhaita käytänteitä testausautomaation toteuttamiseen Android-alustalla. Työssä valitaan myös työkaluja OCTO3 Oy:n käyttöön, joita voidaan hyödyntää Android-sovellusten automaattisessa testauksessa. Työkalujen tutkimisella ja käyttöönotolla on tarkoitus parantaa mobiilikehitysprosessia, tuotteiden laatua ja asiakastyytyväisyyttä. Tällä hetkellä OCTO3 Oy:n sisällä Android-sovellusten testausautomaatiota on toteutettu projekti- ja tekijäkohtaisesti, mutta yrityksen koon kasvaessa on välttämätöntä määritellä tiettyjä prosesseja, kuten testausautomaation toteuttaminen, jotta kokonaisuus pysyy hallinnassa.

Tässä työssä on yksi tutkimuskysymys ja sillä kolme alakysymystä:
Mitkä ovat parhaat käytänteet Android-sovellusten testausautomaatioon ohjelmistoalan pk-yrityksessä?

- *Miten käytänteitä voidaan soveltaa käytännössä OCTO3 Oy:n kaltaisessa pk-yrityksessä?*
- *Millaiset testausautomaation työkalut ovat osa parhaita käytänteitä?*
- *Mikä on testausautomaation työkalujen rooli DevOps-kokonaisuudessa?*

Tässä tutkimuksessa parhailla käytänteillä tarkoitetaan toimenpiteitä tai toimintoja, joiden avulla Android-sovellusten testausautomaatio voidaan suorittaa käytännöllisimmin ja tehokkaimmin. Parhaita käytänteitä mitataan ja arvioidaan tehdyn tutkimuksen perusteella. Arviointiin vaikuttavat kirjallisuuskatsaus, kysely, sekä empiirinen tutkimus. OCTO3 Oy:n kaltaisella pk-yrityksellä tarkoitetaan ohjelmistoalan pientä tai keskisuurta yritystä, jonka liiketoiminta perustuu useaan erilaiseen tuotteeseen. Tarkempi kuvaus OCTO3 Oy:stä on annettu luvussa 1.4.

Mobiilisovellusten testausautomaatio on osa isoa DevOps-kokonaisuutta, mutta tässä työssä rajataan käsiteltäväksi vain testausautomaatio Android-alustalla. OCTO3 Oy:llä on tavoitteena rakentaa DevOps-kulttuurin mukainen mobiilikehitysprosessi, mutta kandidaatintyön puitteissa aihe on liian laaja. Työkalujen valinnan lisäksi työssä ennakoidaan lyhyesti tulevaa DevOps-kokonaisuutta ja arvioidaan työkalujen yhteensopivuutta muiden DevOps-kulttuurissa yleisesti käytettyjen työkalujen kanssa. Yhteensopivuuden arvioinnissa keskitytään lähinnä integraation helppouteen muun muassa raportoinnin osalta. Arviointi tehdään empiirisenä tutkimuksena. Evaluoitavat työkalut rajataan ilmaisiin ja avoimen lähdekoodin ohjelmistoihin. Työssä ei käydä läpi jokaista saatavilla olevaa työkalua, vaan keskitytään tarkastelemaan tällä hetkellä suosituimpia työkaluja. Suosituimpien työkalujen valinnassa huomioidaan työkalujen suosio ja yleisyys kirjallisuudessa, sekä kyselytutkimuksessa tehdyt havainnot.

1.3 Työn rakenne

Tämä työ alkaa kirjallisuuskatsauksella testausautomaatiosta. Siinä tarkastellaan testausautomaatiota yleisesti, osana DevOps-kulttuuria ja tarkemmin Android-alustalla. Tämän jälkeen työssä käydään läpi tutkimusmenetelmät ja työn toteutus. Seuraavaksi esitellään kyselyn ja työkalujen evaluoinnin tuloksia. Evaluoinnissa hyödynnetään empiiristä tutkimusta ja kirjallisuuskatsauksessa esitettyä evaluointimallia. Tämän jälkeen tarkastellaan tutkimustuloksia ja valitaan työkalut OCTO3 Oy:n käyttöön. Työn viimeinen luku kattaa johtopäätökset. Johtopäätöksissä käydään läpi tämän kandidaatintyön tuloksia ja tehdään yhteenveto.

1.4 OCTO3 Oy yritysesittely

OCTO3 Oy on keskisuuri ohjelmistoalan yritys, joka perustettiin Lappeenrannassa vuonna 2011. Yritys avasi toimiston Helsingin Ruoholahteen vuoden 2017 alussa (OCTO3 Oy 2017a). OCTO3 Oy:n ydinosaamiseen kuuluvat mobiilisovellukset, tietojärjestelmät, sekä sähköinen asiointi. Suurin osa yrityksen liikevaihdosta muodostuu sopimuskehityksestä ja konsultoinnista. Ennustettu liikevaihto vuodelle 2017 on noin 4,5 miljoonaa euroa. OCTO3 Oy:n asiakkaita ovat muun muassa Hyvis-ICT, Liikennevirasto ja Eksote. (OCTO3 Oy 2017b.) Yrityksellä on noin 60 asiantuntijaa töissä, joista noin 50 työskentelee Lappeenrannan toimipisteessä. Visma Consulting osti OCTO3 Oy:n koko osakekannan lokakuussa 2017. Visma Consulting on yksi Suomen johtavista IT-konsultointiyrityksistä ja yritysoston jälkeen yrityksellä on noin 220 työntekijää Suomessa. (Visma Consulting 2017.)

OCTO3 Oy:n asiakkaat sijaitsevat pääasiassa Helsingin ja Lappeenrannan alueilla, mutta yrityksellä on projekteja myös näiden kahden kaupungin välillä. Yrityksellä on asiakkaina muun muassa julkisen sektorin yrityksiä, startup-yrityksiä ja muita ohjelmistoalan yrityksiä. Projektit ovat moninaisia, sillä yritys tekee mobiili- ja web-kehityksen lisäksi esimerkiksi sulautettuja järjestelmiä. OCTO3 Oy ei siis tee yhtä omaa markkinoitavaa tuotetta, vaan sopimuskehittäjän roolissa yritys tarjoaa asiakkaille erilaisia ohjelmistoalan tuotteita ja palveluja. Projektit voidaan toteuttaa kokonaan yrityksen sisällä tai yhteistyössä muiden yritysten kanssa. Yritys voi myös vuokrata omia työntekijöitään toisten yritysten käyttöön yksittäisen projektin ajaksi.

OCTO3 Oy on jatkanut itsenäisenä yksikkönä osana Visma Consulting Oy:tä. Pääasiassa OCTO3 Oy raportoi emoyritykselle tuloksia ja pyrkii vastaamaan konsernin asettamiin tavoitteisiin, kuten erittäin hyvään asiakastyytyväisyyteen. Tässä tutkimuksessa OCTO3 Oy:n kaltaisella yrityksellä tarkoitetaan ohjelmistoalan pientä tai keskisuurta yritystä, jolla on samaan aikaan käynnissä useita erilaisia ohjelmistokehitysprojekteja. Tyypillisesti tämän kaltainen yritys tekee sopimuskehitystä. Mobiilikehitys suosituimmilla alustoilla on yksi olennainen vaatimus OCTO3 Oy:n kaltaiselle yritykselle.

2 OHJELMISTOJEN TESTAUSAUTOMAATIO

Ohjelmistojen testaus on äärimmäisen tärkeä osa ohjelmistotuotantoprosessia. Esimerkiksi avaruussukkuloissa käytetyn ohjelmiston laadusta pitää olla täysi varmuus, jotta sitä voidaan testata tuotannossa. TDD (test-driven development), eli testivetoinen kehitys on esimerkki siitä, miten pitkälle testaus on joissain tapauksissa viety. TDD-mallin mukaisesti testit kirjoitetaan ennen itse ohjelmakoodia. (Mäkinen & Münch 2014, s. 115.) Ohjelmistotestausta voidaan tehdä joko manuaalisesti, tai automaattisesti. Tässä luvussa käydään läpi testausautomaatiota yleisesti, osana DevOps-kulttuuria ja Android-alustalla.

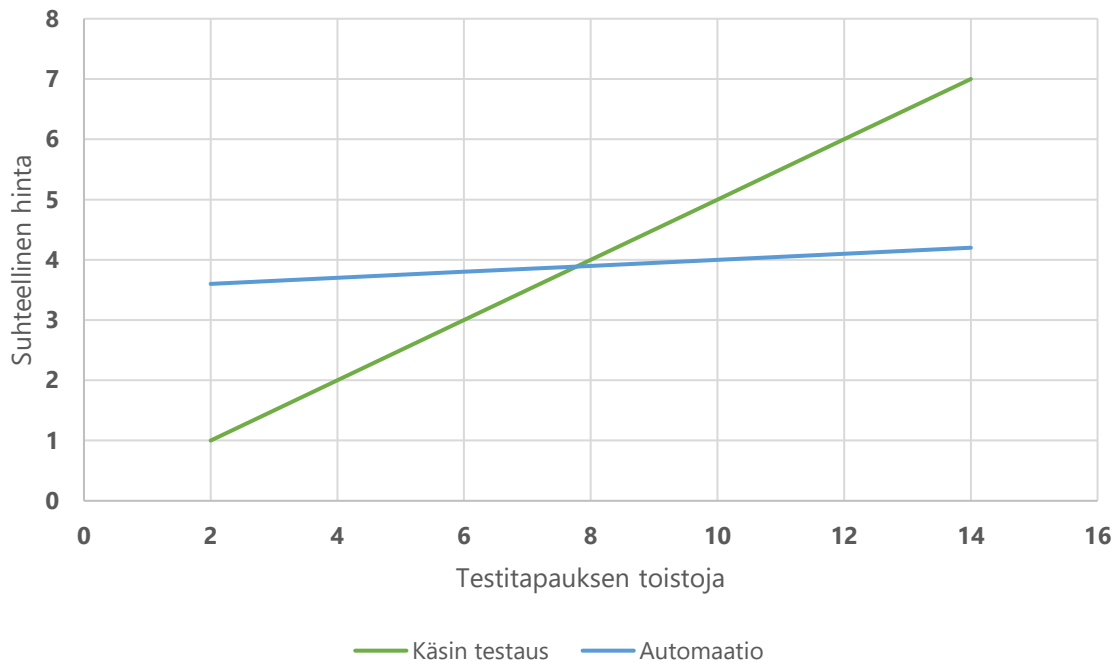
2.1 Testausautomaatio yleisesti

Usein testaajat voivat olla samoja henkilöitä, jotka koodaavat itse tuotetta. Testausautomaation kantavana tavoitteena on vähentää manuaaliseen testaukseen käytettävää työmäärää. Kun manuaalisen testaukseen tarvittava työ poistetaan lähes kokonaan, tuotteen kehitys nopeutuu. (Kasurinen 2013, s. 76.) Isoissa yrityksissä, kuten Facebookilla, testausautomaatio on kriittinen osa ohjelmistokehitystä (Feitelson, Frachtenberg & Beck 2013, s. 8). Vuonna 2009 tehdyssä tutkimuksessa todettiin kuitenkin, että vain 26 % otoksen testitapauksista oli automatisoitu (Kasurinen, Taipale & Smolander 2010, s. 9).

Testausautomaatiota ei hyödynnetä pelkästään toiminnallisten vaatimusten testaukseen, vaan sitä voidaan käyttää muun muassa suorituskykytestauksessa ja tietoturvallisuuden varmistuksessa (Gotimer & Stiehm 2016, s. 13). Ei-toiminnallisten vaatimusten testauksella voidaan automatisoida ohjelmiston hyväksymistestaus, joka manuaalisesti tehtynä maksoi eräälle yritykselle kolme miljoonaa dollaria jokaista uutta julkaisua kohden. (Humble & Farley 2011, s. 187–190.) Tässä työssä keskitytään toiminnallisten vaatimusten testaukseen.

Kuvassa 1 on esitetty testausautomaation suhteellinen hinta verrattuna manuaaliseen testaukseen. Testauksen automatisoinnin valmistelu ja aloittaminen vaativat merkittävää investointia (Toivanen 2017). Tämä näkyy myös kuvassa 1, jossa testausautomaation käyrä alkaa huomattavasti manuaalista testausta korkeammalta. Testausautomaatioon tarvittavien

resurssien merkittävä määrä on yksi syistä, minkä takia sitä ei ole otettu yrityksissä tai varsinkaan pienemmissä projekteissa käyttöön (Kasurinen et al. 2010, s. 14). Jos testitapauksia kuitenkin toistetaan useita kertoja, alkavat testausautomaation kustannukselliset hyödyt näkyä. Tutkimusten mukaan testitapaukset, jotka suoritetaan yli kahdeksan kertaa, kannattaa automatisoida taloudellisesta näkökulmasta (Kasurinen 2013, s. 78).

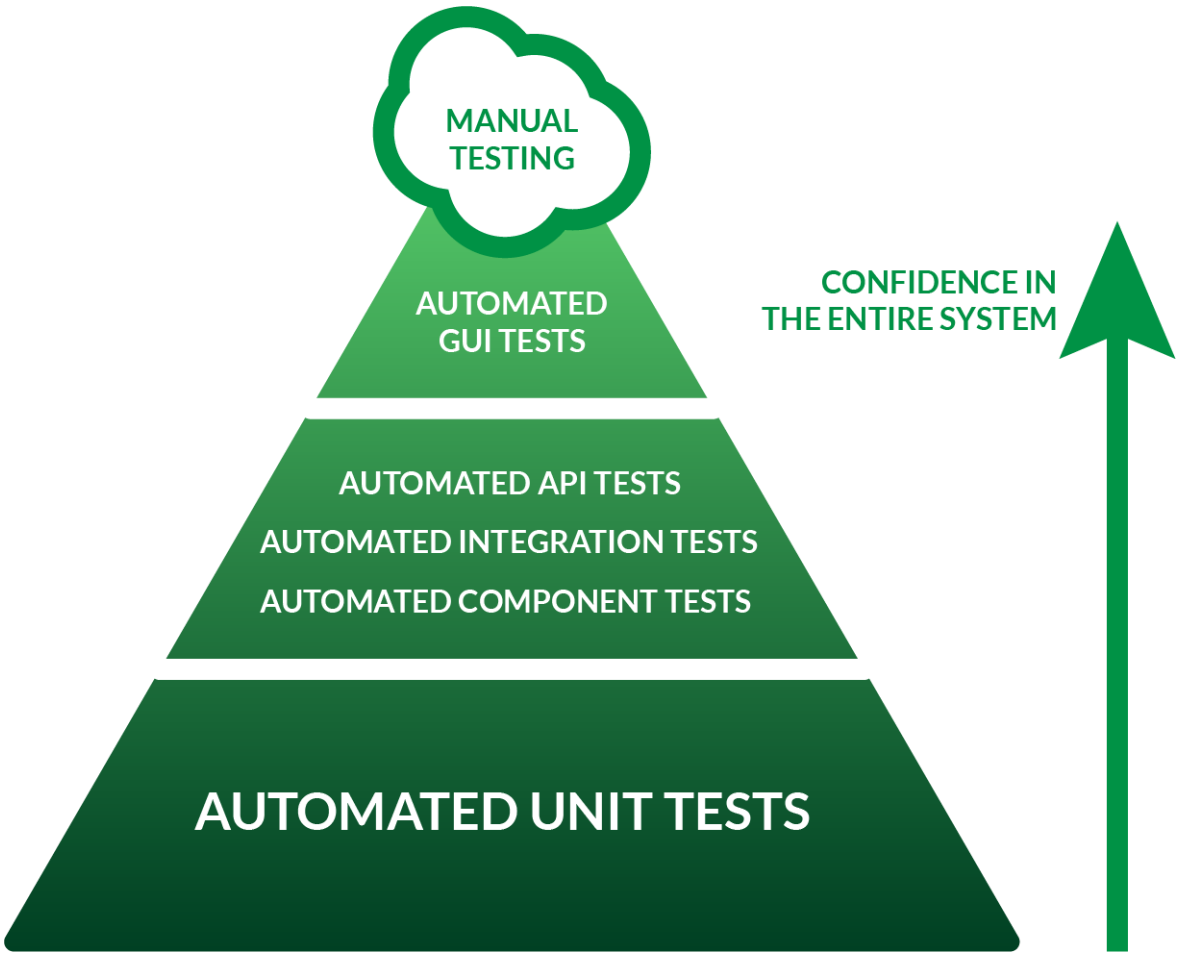


Kuva 1. Testauksen automatisoinnin suhteellinen hinta (Kasurinen 2013, s. 78)

Testausautomaation testitapauksia voidaan luoda muutamalla erilaisella tavalla. Testejä voidaan kirjoittaa joko manuaalisesti tai niitä voidaan luoda käyttämällä työkalua, joka tallentaa käyttäjän syötteet ja myöhemmin toistaa niitä (Marick 1998, s. 3). Haittapuolena manuaalisesti kirjoitetuissa testeissä on niiden vaatimat kustannukset, kun taas työkaluilla luotuja testitapauksia on usein hankala ylläpitää. Työkaluja käyttämällä voidaan kuitenkin säästää huomattavasti aikaa testien luonnissa. (Rafi, Moses, Petersen & Mäntylä 2012, s. 41.)

Kuvassa 2 on esitetty testauspyramidi, joka kuvaa testausautomaation testitapausten järjestystä ja määrää. Alimpana ja suurimpana ryhmänä on yksikkötestit. Yksikkötestien

jälkeen tehdään API- eli rajapintatestausta (application programming interface testing), sekä integraatiotestausta. Viimeiseksi automatisoidaan GUI- eli graafinen käyttöliittymätestaus (graphical user interface testing). Kaiken automatisoinnin jälkeen manuaaliselle testaukselle jää vielä tilaa. Pyramidin ideana on, ettei käyttöliittymätestaus todennäköisesti tule menemään läpi, jos yksikkötestauksessa ilmenee virheitä. Mitä ylemmäs pyramidissa päästään, sitä varmemmaksi tullaan ohjelmiston laadusta. (Nolan 2015, s. 82; Macharla 2017, s. 20.)



Kuva 2. Testauspyramidi (Nolan 2015, s. 82; Macharla 2017, s. 20)

Automaattisesta testauksesta huolimatta manuaalinen testaus on varsinkin tuotteen käyttöönoton yhteydessä tärkeää. Automaation paradoksi on tilanne, jossa hyvin pitkälle viedyssä automaatiossa ihmisen panos korostuu (Kaufman 2017). Automatisoidut testitapaukset eivät niinkään löydä uusia virheitä, vaan varmistavat ennalta määritettyjen

testitapausten virheettömyyden (Nikula 2017). Uusien ja kummallisten virheiden löytämisessä oikeiden ihmisten lähestymistapa on korvaamaton, eikä sitä tulisi unohtaa.

Yrityksen testausautomaation kypsyyttä voidaan arvioida esimerkiksi MPTA.BR-mallilla, joka ehdottaa neljää tasoa kypsyyden arvioinnille. Malli on esitetty taulukossa 1. Mallin esittämät neljä tasoa testausautomaation kypsyydelle ovat hallittu (managed), suunniteltu (designed), suoritettu (executed) ja analysoitu (analyzed). (Furtado, Meira & Gomes 2014, s. 284–285.) Ensimmäisellä tasolla vaaditaan testiprojektin suunnittelun ja seurannan automatisointia. Toisella tasolla vaatimuksina ovat muun muassa testauksen suunnittelun ja vianmäärityksen automatisointi. Kolmannella tasolla vaaditaan datan automaattista generointia ja testitapausten automaattista suoritusta. Korkeimmalla tasolla tavoitteena on hyödyntää työkaluja testitapausten tulosten vertailussa.

Taulukko 1. MPTA.BR-kypsyysmalli (Furtado et al. 2014, s. 285)

1 – Managed	The objective is to introduce the automation of planning and monitoring activities of the test project as well as configuration management practices.
2 – Designed	This maturity level focuses on the definition of automated practices in test design and debugging activities, as well as troubleshooting and static analysis tools.
3 – Executed	The objective of this level is to automate data generation, simulation, emulation, fault-sending, fault-injection and test case execution.
4 – Analyzed	The objective is to use tools to support a comparison of results and indicators.

Jotta testausautomaatiosta on todellisia hyötyjä, se pitää toteuttaa oikein. Automaatiosta ei ole mitään hyötyä, jos testitapauksia ei ole suunniteltu ja toteutettu kunnolla. Jos testitapauksia on liikaa tai niissä on päällekkäisyyksiä, henkilöstön vaihtuessa kukaan ei lopulta tiedä mitä testataan, miten ja miksi. Tästä syntyy varsinkin ylläpidollisia lisäkustannuksia. Testausautomaatio ei myöskään ole pelkästään testitapausten automaattista suorittamista, vaan siihen lukeutuu esimerkiksi tulosten varmistus ja

raportointi. Ilman automaattista raportointia testausautomaation hyötyjä on vaikea mitata, eikä virheiden lähteitä saada paikannettua helposti. (Fewster 2001.)

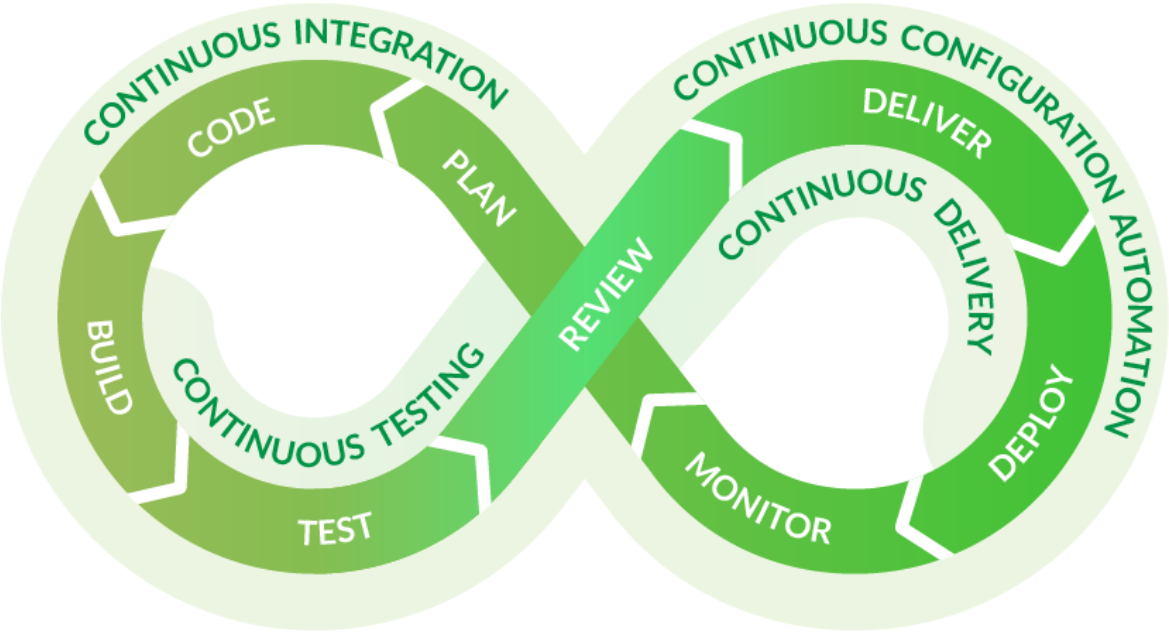
Testausautomaatiota tehdään nykyään myös tekoälyn ja koneoppimisen avulla. Appdiff julkaisi vuonna 2016 tekoälyyn perustuvan testausalustan, joka testaa automaattisesti muun muassa mobiilisovelluksen suorituskykyä ja käyttökokemusta (Appdiff 2016). Toisin kuin perinteisessä testausautomaatiossa, tekoälylle ei tarvitse kirjoittaa manuaalisesti testitapauksia, vaan tekoäly käyttää koneoppimista testien kehittämisessä ja suorittamisessa. Tekoäly oppii, mitkä sovelluksessa tapahtuvat toiminnot ovat toivottavia ja mitkä tulokset pitää luokitella virheiksi. (Merrill 2017.) Tulevaisuudessa tekoälyn merkitys niin testauksessa, kuin muillakin ohjelmistotuotannon osa-alueilla tulee varmasti kasvamaan huomattavasti. Nykyinen testausautomaatio voi menettää merkityksensä hyvinkin nopeasti, jos tekoälyn avulla suoritettava testaus yleistyy teollisuudessa.

2.2 DevOps ja testaus

Erään määritelmän mukaan DevOps on joukko menetelmiä, joiden tarkoituksena on nopeuttaa ohjelmistoon tehtyjen muutosten käyttöönottoa tuotannossa, samalla varmistaen tuotteen laadun (Zhu, Bass & Champlin-Scharff 2016, s. 33). Tyypillisesti DevOps-lähestymistavalla ohjelmistojen läpimenoaikaa voidaan parantaa 10–30:llä prosentilla ja kustannuksia pienentää 20:llä prosentilla. Eräässä case-tutkimuksessa DevOps-menetelmän ja erityisesti testausautomaation hyödyt näkyivät heti läpimenoajassa ja kustannuksissa, kun virheisiin pystyttiin reagoimaan aikaisemmin. Koodia ei enää tarvinnut korjata myöhemmissä vaiheissa ja tuotteen laatu parantui huomattavasti. (Ebert et al. 2016, s. 100.) Myös Yhdysvaltain puolustusministeriö on saavuttanut vastaavia etuja DevOps-menetelmillä (Gotimer & Stiehm 2016, s. 16–17).

Kuvassa 3 on esitetty DevOps-kulttuurin mukainen ohjelmistokehitysprosessi. Se on kuvattu silmukkana, jossa osa työvaiheista toistuu kaikkien muiden vaiheiden lomassa. Pääasialliset tehtävät ovat koodaus (code), testaus (test), läpikäynti (review), toimitus (deliver), levitys (deploy), seuranta (monitor) ja suunnittelu (plan). Jatkuvaa integrointia ja konfiguroinnin

automaatiota tapahtuu koko ajan taustalla. Jatkuva testaus käy läpi ohjelmistoa integraation aikana ja hyväksytysti läpi päässeet kokonaisuudet toimitetaan jatkuvasti eteenpäin.



Kuva 3. Ohjelmistokehityksen vaiheet DevOps-kulttuurin näkökulmasta (Toivanen 2017)

Kuten kuvasta 3 nähdään, testauksella on olennainen rooli DevOps-kulttuurissa. Testausautomaation merkitykselle saatiin vahvistusta vuonna 2016 tehdyssä tutkimuksessa, jossa testausautomaatio nousi toiseksi tärkeimmäksi toiminnoksi DevOps-kulttuuria rakennettaessa (Amaradri & Nutalapati 2016, s. 73). Testaukseen liittyen myös koodikattavuuden tarkastelu on tärkeää, jotta ohjelmiston laadunvarmistuksesta saadaan nopeasti riittävä käsitys (Ebert et al. 2016, s. 99). Yksi DevOps-kulttuurin selkeistä eduista on tylsien työvaiheiden, kuten manuaalisen testauksen vähentyminen, mikä lisää tuottavuutta ja työtehtävien mielekkyyttä (Toivanen 2017).

Kaikille yrityksille ei välttämättä sovi samanlainen DevOps-kokonaisuus, vaan menetelmiä tulee kustomoida tarpeen mukaan (Ebert et al. 2016, s. 100). DevOps-kulttuurin implementointi yrityksessä tuo mukanaan muutoksia muun muassa yrityksen prosesseihin, tuotteisiin, käytettyihin teknologioihin, organisaatiorakenteisiin ja yrityskulttuuriin. Kaikki nämä tekijät vaikuttavat optimaalisen DevOps-kulttuurin rakennukseen. Tästä syystä on

tärkeää tutkia ennakkoon, millaista DevOps-kulttuuria yritykseen kannattaa lähteä rakentamaan. (Zhu et al. 2016, s. 33–34.)

2.3 Testausautomaatio Android-alustalla

Android-sovellusten testaus jaetaan karkeasti yksikkö- ja laitetestaukseen (instrumented testing). Yksikkötestauksessa tarkastellaan yksittäisen moduulin tai funktion toimintaa (Kasurinen 2013, s. 51). Laitetestaus keskittyy integraatio- ja UI-testaukseen (Google & Open Handset Alliance 2017). Integraatiotestaus tarkastelee eri moduulien yhteensopivuutta (Kasurinen 2013, s. 54). UI- eli käyttöliittymätestauksessa (user interface testing) testataan ohjelman käyttöliittymää. Laitetestauksessa testataan siis laajempaa kokonaisuutta, kuin yksikkötestauksessa. Jotta kirjoitetut yksikkö- ja laitetestit voidaan suorittaa automaattisesti, tarvitaan usein jatkuvan integroinnin alusta, eli CI-palvelin. Jenkins-palvelin on yksi monista vaihtoehdoista, mutta se erottuu edukseen lukuisilla lisäosilla. (Nolan 2015, s. 37.) Jos projektia ohjelmoidaan vain yhdellä koneella, CI-palvelinta ei välttämättä tarvita. Kun projekti koostetaan paikallisessa ympäristössä, voi ympäristö käynnistää testit automaattisesti koostamisen jälkeen. Tämä lähestymistapa lisää rasitusta paikalliselle ympäristölle, mutta voi olla kätevä yhden miehen ohjelmointiprojekteissa.

Yksikkötestaus tehdään Java-virtuaalikoneella, kun taas laitetestaus toteutetaan fyysisellä laitteella tai emulaattorilla. Yksikkötestausta varten Androidiin on valmiiksi integroitu yksikkötestauskehys JUnit. Laitetestausta varten Androidille on saatavilla useita kehyksiä, kuten Espresso ja Robotium. (Google & Open Handset Alliance 2017; Nolan 2015, s. 9; Nolan, Cinar & Truxall 2014 s. 78.) Yksikkötestauksessa on tärkeää keskittyä juuri yksittäisen moduulin tai funktion testaukseen. Niinpä riippuvuus muihin ohjelman osiin tai ulkoisiin kirjastoihin tulisi minimoida. Tähän tarvitaan jäljiteltyä dataa (mock data). Androidilla voidaan hyödyntää Mockito-kirjastoa jäljitellyn datan käsittelyssä. (Nolan 2015, s. 45-57.) Testejä tehdessä kannattaa huomioida, ettei testattavana ole Androidin omat komponentit. Ei kannata esimerkiksi testata, voiko nappia koskettaa. Sen sijaan pitäisi testata, tapahtuuko nappia koskettamalla halutut asiat. Pitkät ja vaikealukuiset testitapaukset ovat epäkäteviä, minkä takia on hyödyllisempää kirjoittaa monia pienempiä testitapauksia. (Hellman 2014, s. 157-158.)

Mobiilitestauksessa on tärkeää testata sovellusta fyysisillä laitteilla. Tätä vaaditaan myös Androidilla laitetestauksessa. Taatakseen sovelluksen riittävän laadun, kehittäjän täytyy testata sovellusta useilla laitteilla, riippuen määritellyistä yhteensopivuusvaatimuksista. Fyysisillä laitteilla käsin tehtävä testaus on erittäin työlästä, joten DevOps-kulttuurissa testaus tulisi automatisoida etälaitteiden avulla. Kun uusi versio viedään testaukseen, tehdään testaus ja raportointi automaattisesti palvelimeen kytketyissä fyysisissä laitteissa. (Ravichandran et al. 2016, s. 76-77.) Emulaattorin tai fyysisen laitteen alustus oletustilaan testausta aloittaessa voi joissain tapauksissa olla tärkeää, esimerkiksi jos testaus kohdistuu useampaan, kuin yhteen sovellukseen. Ainakin Jenkins-palvelimelle saatavilla oleva lisäosa, jolla voi käynnistää koonnin yhteydessä Android-emulaattoreita, tarjoaa valinnan laitteen alustukselle.

Android kehittyy alustana jatkuvasti ja niin kehittyvät myös sen ympärille rakennetut työkalut. Tämän takia ei ole olemassa mitään vakiintunutta työkalukokonaisuutta, joka olisi käytössä jokaisessa projektissa. Uusia työkaluja kehitetään jatkuvasti lisää ja vanhemmat ratkaisut voivat poistua käytöstä yhteensopivuusongelmien ja tuen puutteen takia. On kuitenkin olemassa suosituksia käytettävistä testausautomaation työkaluista. Suositukset ovat sidonnaisia aikaan ja työkalutarjonta pitäisikin tarkastaa säännöllisesti.

2.4 Testilaitteet pilvipalveluna

Vuonna 2015 erilaisia Android-puhelimia oli yli 24 000 mallia, joten on lähes mahdotonta, että yrityksellä itsellään olisi nämä kaikki laitteet kytkettynä omaan palvelimeen (OpenSignal 2015). iOS-alustaan verrattuna Android-ekosysteemi on paljon laajemmin hajautunut, mikä asettaa haasteita testausautomaatiolle. Sovellusta tulee testata useilla eri laitteilla esimerkiksi siitä syystä, että laitteen ruudun koko ja resoluutio voi vaikuttaa huomattavasti siihen, miltä sovelluksen asettelu näyttää. Jos käyttöliittymässä ei ole otettu huomioon pieninäyttöisiä laitteita, voivat jotkut asiakkaat kokea ongelmia sovelluksen käytössä. Lisäksi Android-alustalla käytetty ajonaikainen ympäristö on erilainen eri Android-versioissa (Sinhal 2017). Ympäristön vaikutukset sovelluksen toimintaan saadaan

selville vain testaamalla sovellusta useilla eri Android-versioilla. (Baride & Dutta 2011, s. 1.)

Testilaitteiden tarjoamista pilvipalveluna ehdotettiin jo vuonna 2011 tehdyssä tutkimuksessa (Baride & Dutta 2011, s. 3). Myöhemmin esiteltiin MTaaS (Mobile Testing-as-a-Service), eli mobiilisovellusten testaus palveluna. Tällainen palvelu tarjoaa muun muassa testitapausten suoritusta ja monitorointia, skaalautuvan testausinfrastruktuurin ja palvelun käyttöön perustuvan laskutuksen. (Gao, Tsai, Paul, Bai & Uehara 2014, s. 158.) MTaaS mahdollistaa kustannustehokkaan mobiilitestauksen suorittamisen missä ja milloin tahansa jaetun laitekannan ansiosta (Gao et al. 2014, s. 166).

Nykyään on olemassa muutamia kaupallisia MTaaS-mallin mukaisia testauspalveluja, joiden kautta testejä voi suorittaa oikeilla laitteilla ilman, että yrityksen tarvitsee käyttää resursseja omaan testauslaitteistoon ja sen ylläpitoon. Testien suorituksen ja testauksen tulokset voi integroida lisäosien kautta muun muassa Jenkins-palvelimelle. Tällaisia palveluita ovat esimerkiksi Googlen Firebase Test Lab, Xamarin Test Cloud ja Amazonin AWS Device Farm. (Google 2017; Xamarin 2017.) Esimerkiksi Amazonin AWS Device Farm -palvelu tarjoaa käyttäjälle kuvakaappauksia, videoita, lokitiedostoja ja suorituskykydataa testilaitteilta testien suorituksen ajalta. Testilaitteisiin saa myös etäyhteyden, jos halutaan esimerkiksi tutkia tietyllä laitteella esiintyvää ongelmaa tarkemmin. Amazonin palveluun on kytketty yli 2 400 laitetta. (Lopes 2017, s. 35–37.)

Tässä työssä ei tutkita tarkemmin kaupallisia mobiilitestausta tarjoavia palveluja. Tällaisten palvelujen soveltuvuus OCTO3 Oy:n kaltaisten yritysten käyttöön vaikuttaa kuitenkin hyvältä, sillä palvelut ovat helppokäyttöisiä ja helpottaisivat huomattavasti esimerkiksi Android-sovellusten laitetestauksen automaattista toteutusta. Pk-yrityksen ei ole järkevää investoida mittavaan testilaitekantaan siitä syntyvien kustannuksien takia. Olisikin järkevämpää hyödyntää pilvipalvelua, jossa testilaitteet ovat monen yrityksen yhteisessä käytössä. Palvelun käytöstä syntyviä kustannuksia pitäisi kuitenkin arvioida ennen sen käyttöönottoa.

3 TUTKIMUKSEN TOTEUTUS

Tässä työssä dataa kerättiin kirjallisuuskatsauksella, kyselyllä ja empiirisellä tutkimuksella. Kerätty tieto on muun muassa tuloksia aikaisemmista tutkimuksista ja yksittäisten henkilöiden kokemuksia testausautomaatiosta ja siihen käytettävistä työkaluista. Virheellisten johtopäätösten välttämiseksi tutkimukselle tulee asettaa selvät tavoitteet ennen sen aloittamista ja tiedonkeruu tulee tehdä täsmällisesti ja hallitusti (Brown & Wallnau 1996, s. 49). Tutkimusdataa tulee kerätä useasta eri lähteestä, jolloin yhden lähteen virheellisyyden vaikutukset voidaan minimoida. Tämän seurauksena johtopäätökset ovat luotettavampia. (Runeson & Höst 2008, s. 114.)

3.1 Kirjallisuuskatsaus

Työn ensimmäisenä tutkimusvaiheena suoritettiin kirjallisuuskatsaus, jonka tuloksia käytiin läpi toisessa luvussa. Kirjallisuuskatsauksen tavoitteena oli luoda katsaus testausautomaatioon muun muassa Android-alustan ja DevOps-kulttuurin näkökulmasta. Tutkimuksessa pyrittiin käyttämään mahdollisimman paljon akateemisia lähteitä. Suurin osa käytetyistä lähteistä löydettiin Internet-hakupalveluiden kautta. Hakupalveluina käytettiin pääasiassa Google-hakukonetta, akateemiselle tiedonhaulle tarkoitettua Google Scholar -palvelua, sekä yliopiston tiedonhakupalvelua, Finnaa. Muutamia lähteitä löydettiin myös yliopiston kirjastosta.

Tietoa haettiin suomen- ja englanninkielisillä hakusanoilla. Hakusanoina käytettiin muun muassa seuraavia avainsanoja ja niiden yhdistelmiä: ”Android”, ”DevOps”, ”testausautomaatio”, ”ohjelmistotestaus”, ”Android-sovellukset”, ”test automation”, ”software testing”, ”continuous integration” ja ”software evaluation”.

3.2 Kyselytutkimus

OCTO3 Oy:n asiantuntijoiden kesken suoritettiin matalan kynnyksen kysely. Kysely oli verkkopohjainen ja vastaukset ovat anonyymejä. Kyselyllä oli tarkoitus kartoittaa

testausautomaation työkalujen käyttöä ja käyttämättömyyttä yleisellä tasolla. Tavoitteena oli kerätä tietoa kokemuksista ja ajatuksista testausautomaation osalta. Kyselyssä pyrittiin myös selvittämään, mitä piirteitä testausautomaation työkaluilta vaaditaan.

Kysymyksiä laadittaessa tulee ottaa huomioon viisi olennaista asiaa (Fowler 1995, s. 9):

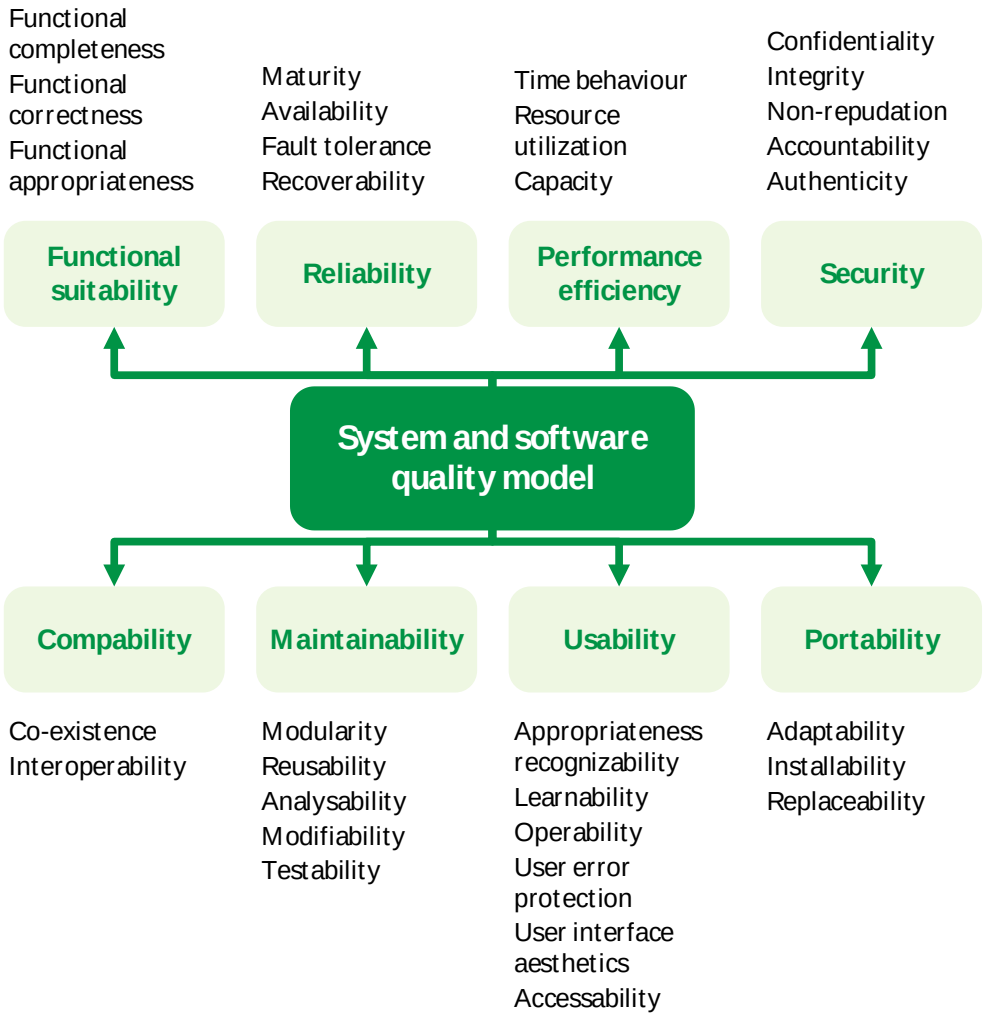
1. Tavoitteiden ja niiden saavuttamiseen tarvittavien vastausten määrittäminen.
2. Vastaajien tulee ymmärtää kysymys samalla tavalla. Kysymyksen laatijalla ja vastaajalla tulee olla yhteisymmärrys käsitteistä.
3. Vastaajien on mahdollista vastata kysymykseen. Vastaaminen voi olla vaikeaa, jos vastaajilla ei ole tarvittavaa tietoa, tai se on unohtunut.
4. Vastaukset ovat määrittelyn mukaisia. Vastaus voi olla näennäisesti oikein, mutta erota halutusta, jos vastaajan ja kysymysten laatijan ajatusmaailmoissa on eroavaisuuksia.
5. Vastaaja haluaa vapaaehtoisesti vastata kysymykseen oikein.

Kyselyn kysymykset laadittiin Fowlerin esittämän teorian mukaan. Kyselyn tulokset analysoidaan ja ne voivat vaikuttaa merkittävästi esimerkiksi työkalujen valintaan. Tuloksia voidaan myös hyödyntää sisäisesti OCTO3 Oy:llä. Jos kyselyn tuloksia arvioimalla paljastuu selkeitä ongelmakohtia liittyen testausautomaation käyttöön, yritys voi toimillaan yrittää vaikuttaa niihin.

3.3 Empiirinen tutkimus

Tämän työn pääpaino on testausautomaation työkalujen empiirisellä tutkimuksella. Työkaluja käytetään tuotantoympäristöä vastaavassa ympäristössä ja kokemusten perusteella tehdään johtopäätöksiä työkalun sopivuudesta OCTO3 Oy:n käyttöön. Empiirisen tutkimuksen läpivienti alkaa tutkittavien työkalujen valinnalla. Arvioijan tulee määrittää arviointikriteerit ja tehdä tarvittaessa lomake, jonka perusteella työkalua voidaan arvioida. Seuraavaksi määritellään ja suoritetaan tehtävät, joiden perusteella työkaluja voidaan arvioida. Lopuksi tulokset analysoidaan. (Kitchenham 1996, s. 27.) Tässä työssä arviointikriteerit valitaan kyselytutkimuksen perusteella. Erillistä arviointilomaketta ei tehdä, sillä kyseessä on pienimuotoinen, yhden henkilön suorittama tutkimus.

Tässä työssä empiirisen tutkimuksen tulosten analysoinnin pohjalla käytetään ISO/IEC 25010 -standardia, joka on esitetty kuvassa 4. Standardi on kehitetty vastaamaan tarpeeseen arvioida ja luokitella ohjelmistoja. Se määrittelee kahdeksan piirteen mallin, jotka on jaettu vielä alipiirteisiin. (ISO & IEC 2011.) Tämän mallin avulla työkaluja voidaan arvioida esimerkiksi säteittäistä kaaviota hyödyntäen. Mallista valitaan tietyt tarkasteltavat ominaisuudet ja jokainen työkalu pisteytetään valittujen ominaisuuksien osalta. Tuloksia analysoidessa pisteytykseen voidaan asettaa kertoimia, mikäli halutaan painottaa jotain tiettyä osa-aluetta.



Kuva 4. ISO/IEC 25010 -standardin määrittelemä laatumalli (ISO & IEC 2011)

Mallissa on paljon tämän tutkimuksen puitteissa tarpeettomia piirteitä, joten mallin kaikkia piirteitä ei painoteta evaluoinnissa. Tärkeimmät piirteet pyritään selvittämään aikaisemmin toteutetun kyselyn avulla. Kyselyssä vastaaja määrittelee, kuinka tärkeäksi kokee kunkin piirteen. Näiden vastausten perusteella voidaan rajata tietyt standardin mallin piirteet pois.

3.4 Reflektio tutkimuksen toteutumisesta

Tutkimus sujui kohtuullisen hyvin suunnitelman mukaan. Kaikki asetetut tavoitteet saavutettiin, mutta aikataululllisesti tutkimuksen toteutus viivästyi alkuperäisistä tavoitteista erinäisten syiden vuoksi. Vastaisuudessa aikataulutuksen merkitystä tulisi korostaa ja pyrkiä noudattamaan sitä mahdollisimman tarkasti. Haastavan aikataulun vuoksi esimerkiksi haastatteluja ei ehditty suorittamaan. Haastattelujen avulla olisi voitu luoda lisäarvoa empiiriselle tutkimukselle ja saada tukea kyselyn vastauksille.

Kirjallisuuskatsauksen aikana huomattiin, ettei testausautomaatiosta Android-alustalla ole tehty paljoakaan tutkimusta. Aiheesta löytyi julkaisuja kohtuullisesti, mutta yleisesti ottaen ne olivat yhden miehen suosituksia ja ohjeistuksia työkalujen käyttöön, eivätkä niinkään tutkimustuloksia tehdystä akateemisesta tutkimuksesta. Lähteitä löytyi huomattavasti paremmin verkkopohjaisista hakupalveluista, kuin fyysisistä kirjastoista. Tämä on ymmärrettävää, sillä tieto ohjelmistoista ja niihin liittyvistä parhaista käytänteistä vanhenee nopeasti. Kirjastot eivät välttämättä halua investoida kirjoihin, joita kukaan ei lainaa enää muutaman vuoden päästä julkaisusta. Odotetusti suomenkielisillä hakusanoilla ei löytynyt läheskään yhtä paljon tutkimuksen kannalta hyödyllisiä hakutuloksia, kuin englanninkielisillä hakusanoilla. Kirjallisuuskatsauksessa huomattiin myös, ettei DevOps-kulttuurin rakentamisesta tai siihen sopivasta testausautomaatiosta ole tehty merkittävästi tutkimusta. Tämä asetti haasteita tutkimuskysymyksiin vastaamiselle kirjallisuudessa esitetyn tiedon perusteella.

Kyselyn kysymysten hiomiseen olisi voitu käyttää enemmän aikaa. Tässä muodossa kysely oli ehkä liian tiivis ja pitkä. Vastaamiseen pystyi helposti käyttämään ainakin 15 minuuttia, joten vastaamisen aikana keskittyminen voi jo alkaa herpaantua. ISO/IEC 25010 -standardin piirteitä on 31 ja vastaajaa pyydettiin arvioimaan jokaisen tärkeyttä. Piirteet eivät ole

yksiselitteisiä edes lyhyen kuvauksen kanssa, eikä kaikki vastaajat oletettavasti miettineet syvällisesti jokaisen piirteen tärkeyttä juuri tämän tutkimuksen kontekstissa. Osa piirteistä saattoi myös jäädä epäselviksi, mikä rikkoo Fowlerin esittämän teorian toista kohtaa vastaajan ja kysymyksen laatijan yhteisymmärryksestä. Olisikin voinut olla hyödyllisempää selvittää arviointimallin tärkeitä piirteitä haastatteluilla, eikä kyselyllä. Tällä tavalla piirteiden avaamiseen olisi voitu käyttää enemmän aikaa ja väärinymmärryksiltä olisi voitu välttyä.

Empiirisessä tutkimuksessa tutkimusympäristön pystyttämiseen kului odotettua enemmän aikaa. Tutkimuksessa käytettiin versionhallintajärjestelmää ja CI-palvelinta, joita ajettiin virtuaalikoneessa. Tuotantokäytössä nämä olisivat erillisessä koneessa, jolloin vältytään virtuaalikoneen rajoituksilta esimerkiksi muistin ja verkon käytön suhteen. Tämä olisi ollut järkevää myös tämän työn tutkimuksessa, mutta resurssien puutteen vuoksi päädyttiin virtuaalikoneeseen. Lisäksi joidenkin työkalujen dokumentaation puute asetti haasteita evaluoinnin suorittamiselle. Empiirisen tutkimuksen tuloksiin tulee suhtautua varauksella, sillä testauksen tarve on projektikohtaista. Valitut arviointiin käytetyt piirteet voivat olla joissain tapauksissa sopimattomat tai niitä ei ole painotettu sopivasti tietyn projektin kontekstissa. Yhden työkalun arviointiin käytettiin noin kahdeksasta kymmeneen tuntia, jolloin kaikki niiden erityiset hyödyt tai epätavalliset virheet eivät oletettavasti tulleet esille tässä työssä. Tutkimus luo kuitenkin pohjan työkalujen valinnalle.

4 KYSELYN TULOKSET

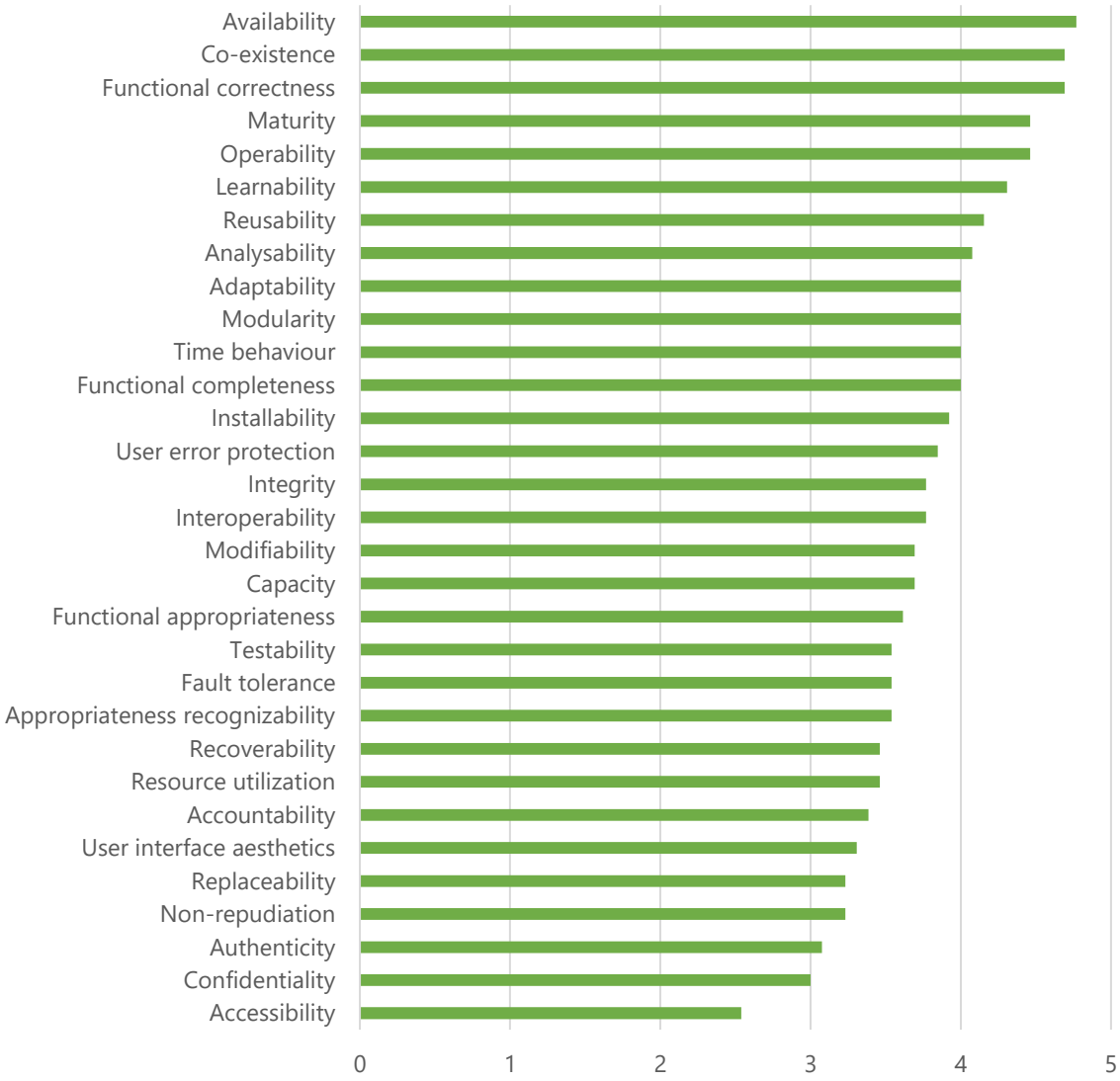
Kysely suoritettiin OCTO3 Oy:n asiantuntijoiden kesken ja siihen vastasi 13 henkilöä, joista 5 oli ollut mukana Android-projekteissa. Vastausprosentti on kohtalainen, noin 25. Kysely suoritettiin Google Forms -alustan kautta ja kysymykset ovat nähtävissä liitteessä 1. Linkki kyselyyn lähetettiin OCTO3 Oy:n henkilöstön sähköpostilistalle ja reaaliaikaiseen viestintään tarkoitetulle kanavalle. Kyselyyn kehotettiin vastaamaan myös henkilökohtaisesti.

Kyselyn tuloksena saatiin kysymyksestä riippuen kyllä- tai ei-vastauksia, tai lukuja yhden ja viiden väliltä. Kyselyn jokaisessa osassa oli myös mahdollisuus vapaaseen kommentointiin ja vapaita kommentteja saatiin muutamia jokaisessa osassa. Raakadataa analysoitiin Google Forms -alustan oman raportointityökalun ja Microsoft Excelin avulla. Google Forms -alusta tekee automaattisesti kyselydatasta erilaisia kuvaajia. Microsoft Excel -taulukkolaskentaohjelmaa käytettiin datan jatkoanalysointiin niissä tapauksissa, joissa valmiit kuvaajat eivät palvelleet tutkimuskohdetta. Esimerkiksi ISO/IEC 25010 -standardin piirteiden arvioinnissa Google Forms -alustalla jokaisesta piirteestä oli piirretty erillinen kuvaaja. Piirteiden esittämiseen yhdessä kuvaajassa käytettiin Microsoft Excel -ohjelmaa. Analyysin yksinkertaistamiseksi dataa analysointiin vastausten keskiarvon avulla. Johtopäätöksiä tehtiin kuvaajista osuuksien perusteella ja vapaita kommentteja hyödynnettiin johtopäätösten tukena. Virhetulkinnat voivat olla mahdollisia johtuen otoksen pienestä koosta. Joissain tapauksissa yksittäinen vastaus saattoi näyttää kuvaajassa merkittävältä, mutta virheanalyysiä pyrittiin välttämään vertaamalla kuvaajaa raakadataan ja keskiarvoihin. Myöskään yksittäisille vapaille kommenteille ei annettu suurta painoarvoa, vaan niitä käytettiin lähinnä analyysin tukena.

Android-projekteihin osallistuneet henkilöt olivat käyttäneet testausautomaatiota joissain projekteissa. Muissa projekteissa testausautomaatiota oli käytetty vaihtelevasti. Yleisimpiä syitä testausautomaation käyttöön olivat sillä saavutettavat laadulliset ja ajansäästölliset hyödyt, sekä testausautomaation asema projektivaatimuksena. Testausautomaation käyttö koettiin myös luonnolliseksi. Esimerkiksi muutosten tekeminen koettiin turvallisemmaksi, jos testausautomaatio on ollut käytössä. Suurimmat syyt testausautomaation

käyttämättömyyteen olivat sen aloituksen vaatimat ajalliset resurssit ja työkalujen tuntemattomuus. Tästä johtuen, jos testausautomaatio ei ole ollut vaatimuksena projektissa, asiakas ei välttämättä halua maksaa testausautomaation pystyttämistä. Myös automatisoinnin hyödyt pienessä projektissa kyseenalaistettiin. Pienessä projektissa testausautomaation ja ympäristön pystytys ei ole välttämättä vaivan arvoista. Vastaajista 92 % oli sitä mieltä, että testausautomaatiota tulisi käytettyä enemmän, jos yrityksellä olisi selkeät käytänteet ja ohjeistus testausautomaation käyttöön.

Kyselyn toisessa osassa käytiin läpi ISO/IEC 25010 -standardin piirteitä ja niiden merkitystä testausautomaation työkalujen kontekstissa. Kuten luvussa 3.4 on todettu, oltaisiin voitu saada laadukkaampia tuloksia, jos tämä osa olisi suoritettu haastatteluna. Kuvassa 5 on esitetty, miten tärkeäksi kukin laatumallin piirre koettiin asteikolla 1–5. Kuten kuvasta huomataan, lähes kaikki piirteet arvioitiin vähintään asteikolle 3, eli ne koettiin jokseenkin tärkeiksi. Toisaalta nämä piirteet ovat hyvän ohjelmiston peruspiirteitä, joten tulokset eivät yllätä. Tuloksilta olisi kuitenkin toivottu tarkempaa keskittymistä juuri testausautomaation työkalujen kontekstiin. Esimerkiksi tietoturva on asia, mihin testausautomaation työkalut eivät luonteensa vuoksi voi juuri vaikuttaa. Tietoturva koskettaa enemmänkin ympäristöä, missä työkaluja käytetään, kuin itse työkalua.



Kuva 5. Kyselyn tulokset työkaluilta vaaditut piirteet

Kyselyn tulosten perusteella valittiin kuusi piirrettä, joita hyödynnetään empiirisessä tutkimuksessa ja työkalujen evaluoinnissa. Joissain piirteissä yhdistettiin kaksi tai kolme standardin piirrettä, jotka ovat tässä kontekstissa lähes sama asia. Valitut piirteet ovat helppokäyttöisyys (learnability ja operability), toiminnallisuus (functional correctness ja functional completeness), vakaus (maturity), nopeus (time behaviour), saatavuus (adaptability, reusability ja availability) ja integroituminen (interoperability). Korkealle sijoittunut yhteiselo (co-existence) koettiin niin epäolennaiseksi tämän tutkimuksen kannalta, ettei sitä valittu evaluointikriteeriksi. Integroituminen taas osoittautui niin olennaiseksi tämän työn kontekstissa, että se otettiin mukaan yhdeksi kriteereistä.

Viimeisessä osassa selvitettiin Android-alustan testausautomaation työkalujen tuntemusta ja aikaisempaa käyttöä. JUnit on selkeästi tunnetuin ja käytetyin työkalu. Robotium, Espresso ja Appium ovat toiseksi käytetyimmät työkalut lähes samoilla osuuksilla. Näiden tulosten perusteella edellä mainitut neljä työkalua valittiin mukaan empiiriseen tutkimukseen. Listan ulkopuolelta ei noussut esiin mainintoja muista käytetyistä työkaluista. Kyselyn työkalulistan kasaus oli tehty pääosin virallisen Android-dokumentaation ja kirjallisuuskatsauksen pohjalta.

Otos kyselytutkimuksessa oli määrällisesti pieni, mutta yrityksen sisällä otos kuitenkin vastasi noin 25:tä prosenttia niistä työntekijöistä, jotka tekevät töitä ohjelmistokehityksen ja testauksen parissa päivittäin. Prosentuaalinen suhde on jo kohtuullisen hyvä ja tuloksia voidaan käyttää OCTO3 Oy:n sisäisessä analyysissä. Tulosten mallintaminen muihin yrityksiin ei ole välttämättä hyödyllistä, sillä usein yritykset palkkaavat itselleen juuri kyseiseen yritykseen sopivia henkilöitä. Tästä syystä OCTO3 Oy:n ja jonkun toisen, näennäisesti samankaltaisen yrityksen työntekijöiden välillä voi olla huomattavia eroja. Yleisesti luotettavampaa ja käytännöllisempää dataa olisi saatu, jos kysely olisi toteutettu useammassa, kuin yhdessä yrityksessä. Toteutetun kyselyn tuloksia voi hyödyntää vain OCTO3 Oy:n sisällä ja pienellä varauksella myös muissa OCTO3 Oy:n kaltaisissa yrityksissä.

5 TYÖKALUJEN EVALUOINTI

Evaluoitavat työkalut on valittu osittain kyselyn tulosten perusteella. Kyselystä nousi selvästi esiin neljä työkalua. Lisäksi testattavien työkalujen valinnassa hyödynnettiin vuonna 2017 tehtyä tutkimusta Android-sovellusten UI-testaukseen tarkoitetuista työkaluista. Tutkimuksessa suosituimmiksi testausautomaation työkaluiksi osoittautuivat Espresso ja Appium. Työkalujen suosiota mitattiin tehtyjen Google-hakujen määrällä ja suosiolla Stack Overflow -verkkosivustolla, joka on suosittu foorumi ohjelmointiin liittyvien kysymysten esittämiselle ja ratkaisemiselle. (Lämsä 2017, s. 16–20.) Evaluointiin nostettiin mukaan myös yksikkötestauskehys Robolectric, joka on mainittu kirjallisuudessa. Taulukossa 2 on listattu tutkimuksessa evaluoidut testausautomaation työkalut. Työkaluista käytettiin uusinta mahdollista versiota, jotta työkalut olisivat mahdollisimman tasavertaisessa asemassa.

Taulukko 2. Tutkimuksessa evaluoidut työkalut

Työkalun nimi	Versio	Testauskategoria
JUnit	4.12	Yksikkötestaus
Robolectric	3.5.1	Yksikkötestaus
Espresso	3.0.1	Laitetestaus
Appium	1.2.7	Laitetestaus
Robotium	5.2.1	Laitetestaus

Evaluointia varten tehtiin Android-projektina suhteellisen yksinkertainen taskulaskin. Laskimessa on perustoiminnallisuudet summalle, erotukselle, tulolle, osamäärälle ja desimaaleille. Käyttäjä näppäilee haluamansa luvut painamalla jokaiselle numerolle yksilöllisiä nappeja. Tulos näytetään käyttäjälle uudessa näkymässä. Tässä tutkimuksessa olisi voitu käyttää myös oikeaa, kehitteillä olevaa tuotetta ja luoda sille testitapauksia jokaisella työkalulla. Tutkimuksen tavoitteet kuitenkin saavutetaan myös pienellä ja yksinkertaisella testauskohteella. Työn tarkoitus ei ollut niinkään testata työkaluja todellisessa käytössä, vaan tutkia niiden ominaisuuksia ja potentiaalia tuotantokäyttöön.

5.1 Tutkimusympäristö

Työkalujen arviointi pyrittiin suorittamaan ympäristössä, joka vastaa tyypillistä kehitysympäristöä OCTO3 Oy:llä. Taulukossa 3 on esitetty ympäristön ohjelmistot. Tärkeimmät ympäristön ohjelmistot ovat CI-palvelin ja versionhallintajärjestelmä. CI-palvelimeksi valittiin Jenkins, sillä se on ollut aiemmin käytössä OCTO3 Oy:llä. Lisäksi Jenkins mainittiin kirjallisuuskatsauksessa muutaman kerran. Jenkinsille on saatavilla useita erilaisia lisäosia, jotka nopeuttavat testausautomaation aloitusta ja helpottavat sen hallintaa. GitLab valittiin versionhallintajärjestelmäksi, sillä se on jo OCTO3 Oy:n käytössä.

Taulukko 3. Tutkimusympäristön kuvaus

Ohjelmiston nimi	**Versio**	**Käyttötarkoitus**
Windows 10 Pro	1709	Evaluointiin käytetyn tietokoneen käyttöjärjestelmä
Android Studio	3.0.1	Android-kehitys
VMware Workstation 14 Player	14.0.0	Virtuaalikone taustajärjestelmille
Ubuntu	16.04	Virtuaalikoneen käyttöjärjestelmä
GitLab	10.2.0	Versionhallintajärjestelmä
Jenkins	2.73.3	CI-palvelin

Varsinainen Android-kehitys ja testitapausten luonti tehtiin Windows-ympäristössä. CI-palvelin ja versionhallintajärjestelmä sijaitsivat virtuaalikoneella, joka simuloi hyvin tilannetta, jossa kyseiset järjestelmät ovat yrityksen verkossa erillisellä palvelimella. Tutkimuksessa pyrittiin mahdollisimman autenttiseen kehitystilanteeseen. Kehittäjä kehittää sovellusta ja samalla testitapauksia omalla, henkilökohtaisella koneellaan. Kun vaaditut muutokset on tehty, koodi työnnetään versionhallintajärjestelmään. Versionhallintajärjestelmä ilmoittaa CI-palvelimelle, että saatavilla on uusi versio projektista. CI-palvelin hakee projektin uusimman version ja suorittaa sille ennalta määritetyt toimenpiteet. Tässä tapauksessa CI-palvelin koostaa projektin, ajaa yksikkö- ja laitetestit, sekä esittää testausraporttien tulokset.

Jenkins CI-palvelimella käytettiin Android Emulator -lisäosaa (versio 2.16-SNAPSHOT), joka käynnistää ennen testien ajoa yhden tai useamman Android-emulaattorin. Emulaattorin ominaisuudet, kuten käyttöjärjestelmän versio ja ruudun tarkkuus, voidaan määrittää erikseen. Testien koodikattavuuden esittämiseen käytettiin JaCoCo-lisäosaa (versio 2.2.1). Jenkins ajaa uudelle versiolle testit käynnissä olevissa Android-emulaattoreissa, minkä jälkeen se raportoi testauksen tulokset JUnit-raporttina ja koodikattavuuden JaCoCo-raporttina. Saatavilla on myös muita CI-palvelimia ja lisäosia testauksen tulosten raportointiin, mutta niiden tutkiminen ei ollut tämän kandidaatintyön pääpainona. Ympäristöstä pyrittiin tekemään sellainen, että se olisi helppo toistaa OCTO3 Oy:llä tällä hetkellä.

5.2 Tutkimuksen testitapaukset

Yksikkötestauksen ja laitetestauksen työkaluja testattiin erikseen ja niille kirjoitettiin luonnollisesti eri asioita testaavat testit. Testauskategorian sisällä käytettiin samoja testitapauksia. Taulukoissa 4 ja 5 on vastaavasti esitetty yksikkö- ja laitetestitapaukset. Testitapauksissa pyrittiin testaamaan työkalujen yleistä toimivuutta ja tyypillisiä Android-sovellusten testauskohteita.

Taulukko 4. Yksikkötestauksen testitapaukset

Testitapaus	Kuvaus
Y1	Laskumetodin testaus: 1 + 1 palauttaa 2.
Y2	Laskumetodin testaus: 1 / 2 palauttaa 0,5.
Y3	Laskumetodin testaus: 4 - 10 palauttaa -6.
Y4	Laskumetodin testaus: 6 * 101 palauttaa 606.
Y5	Tulosmetodin testaus: 10 palauttaa merkkijonon ”Result: 10”.

Yksikkötestitapauksissa testattiin kahta ohjelman metodia. Toinen metodi laskee laskuja kolmen parametrin perusteella, joista kaksi ovat laskussa käytettäviä numeroita ja kolmas on matemaattinen operaattori. Toinen metodi palauttaa laskun tuloksena käytettävän

merkkijonon annetusta parametrista. Molemmat metodit ovat puhdasta Java-koodia, eikä niissä ole käytetty ulkoisia tai Android-alustan kirjastoja.

Taulukko 5. Laitetestauksen testitapaukset

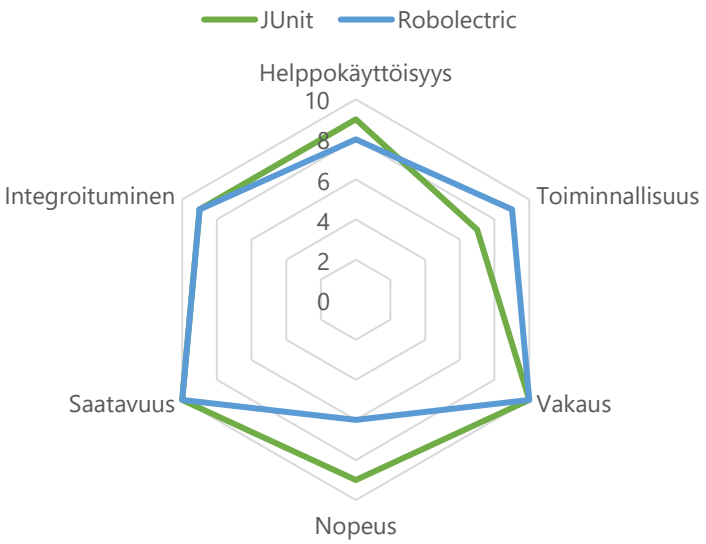
Testitapaus	Kuvaus
L1	Käyttäjä naputtelee 1 * 2, jolloin laskimen tekstikentässä on merkkijono ”1*2”.
L2	Käyttäjä painaa tulosnappia naputtelematta numeroita, eikä sovellus vaihda ruutua.
L3	Käyttäjä naputtelee 1 / 2 ja painaa tulosnappia, jolloin sovellus vaihtaa ruutua ja näyttää merkkijonon ”Result: 0,5”.
L4	Käyttäjä painaa ruudun oikeassa alareunassa olevaa nappia, jolloin ruudun alalaitaan ilmestyy ilmoitus.

Laitetestauksen testitapauksissa testattiin yleisesti taskulaskimen käyttöliittymää ja sovelluksen toimintaa Android-laitteella. Testitapauksina käytettiin sovelluksen oikeita käyttötapauksia, joissa käyttäjä laskee laskun ja käyttäjälle esitetään tulos. Testitapauksilla testattiin muun muassa miten arvioitavat työkalut käyttäytyvät, jos näkymään ilmestyy jotain uutta tai jos näkymä vaihtuu kokonaan uuteen.

Testitapauksia olisi oikeassa tuotantoprojektissa luonnollisesti enemmän ja ne olisivat monipuolisempia. Yksinkertaisen taskulaskimen testauksen haittapuolena on, ettei testitapauksista saatu kovin monimutkaisia. Tutkimuksen aikataulun puitteissa näillä testitapauksilla saatiin kuitenkin tarvittavat tai ainakin suuntaa antavat tulokset jokaisesta työkalusta. Testitapausten kirjoittaminen ja työkalujen dokumentaation lukeminen antoivat käsityksen työkalujen toiminnallisesta potentiaalista. Työkalujen testauksella saavutettiin yksinkertaisten ominaisuuksien todentamisen lisäksi myös arviot testauksen raportoinnin helppoudesta ja integroinnista CI-palvelimen kanssa.

5.3 Evaluoinnin tulokset

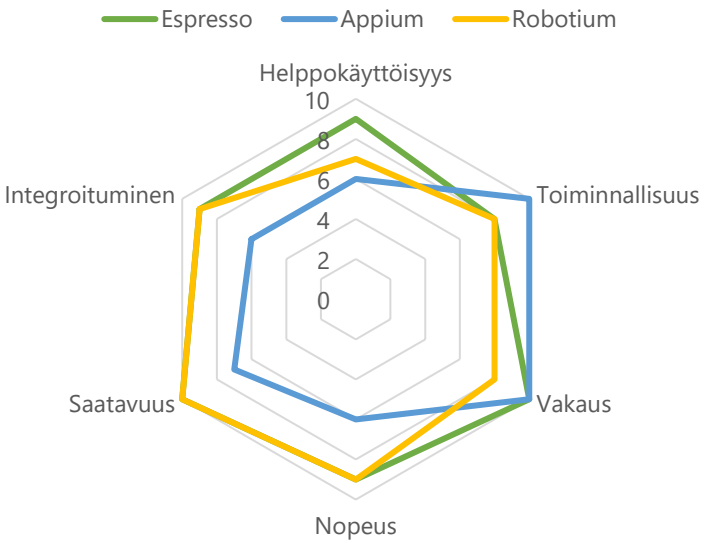
Työkalut evaluoitiin empiirisen tutkimuksen ja työkalujen dokumentaation perusteella. Evaluointikriteereinä olivat helppokäyttöisyys, toiminnallisuus, vakaus, nopeus, saatavuus ja integroituminen. Helppokäyttöisyydellä arvioitiin työkalun opittavuutta ja dokumentaatiota. Toiminnallisuudella arvioitiin työkalun ominaisuuksia ja niiden antamia oikeita tuloksia. Vakaus määriteltiin sillä, kaatuiko työkalu tai tapahtuiko siinä virheitä testauksen aikana. Nopeus mitattiin testitapausten suoritusnopeudella. Saatavuudessa tarkasteltiin, mille kaikille alustoille työkalu on saatavilla ja miten yksinkertainen työkalu oli asentaa tai ottaa käyttöön. Integroitumisella arvioitiin sitä, miten työkalu integroitui CI-palvelimen kanssa. Käytännössä siis arvoitiin, miten helppoa testejä oli suorittaa palvelimella ja miten helposti testausraportin sai esitettyä. Kuvissa 6 ja 7 on esitetty arvioinnit tämän tutkimuksen testausautomaation työkaluista. Jokainen työkalu arvioitiin valittujen piirteiden perusteella asteikolla 0–10.



Kuva 6. Yksikkötestauksen työkalujen arviointi

Kuvasta 6 nähdään, miten yksikkötestauksen työkalut suoriutuivat evaluoinnissa. JUnit pärjää hyvin nopeudessa, kun taas Robolectric toiminnallisuudessa. Molemmat työkalut ovat helppokäyttöisiä ja hyvin dokumentoituja. JUnit-kehyksellä on kuitenkin etua, sillä se on integroitu suoraan Android-alustaan ja sitä on dokumentoitu virallisessa Android-

dokumentaatiossa. Sen käyttämiseksi ei ole tarvetta ladata ulkopuolisia kirjastoja. JUnit on myös erittäin nopea verrattuna Robolectric-kehykseen. Robolectric-kehyksen etuna on sen toiminnallisuus. Vaikka se on käytännössä yksikkötestauskehys, sillä voidaan myös suorittaa laitetestausta tietyllä tasolla. Robolectric simuloi emulaattorin toimintaa ja mahdollistaa sellaisen Android-alustan ominaisuuksien testauksen, jota ei ole mahdollista tehdä pelkällä JUnit-kehyksellä. Tästä syystä esimerkiksi jäljitellyn datan luontiin ei välttämättä tarvita erillistä kirjastoa. (Robolectric 2017.) Koska Robolectric simuloi Android-emulaattoria, se on myös selkeästi hitaampi pelkkään JUnit-kehykseen verrattuna. Molemmat työkalut ovat vakaita, helposti saatavilla ja helposti integroitavissa esimerkiksi Jenkins-palvelimeen.



Kuva 7. Laitetestauksen työkalujen arviointi

Kuvassa 7 on esitetty laitetestauksen työkalujen arviointi. Kuvasta nähdään, että Espresso-kehys pärjäsi yleisesti ottaen parhaiten evaluoinnissa. Se oli helppokäyttöisyydeltään selkeästi muita kehyksiä parempi. Espresso on Googlen tukema ja se on myös dokumentoitu virallisessa Android-dokumentaatiossa. Espresson etu on sen niin sanottu tietoisuus sovelluksen tilasta. Muita työkaluja käyttäessä käyttäjän pitää kirjoittaa koodia, jolla odotetaan esimerkiksi sovelluksen siirtymistä näkymästä toiseen. Tälle ei ole tarvetta Espresso-kehystä käytettäessä. Espresso on Robotiumin kanssa suunnilleen yhtä nopea. Espresso ja Robotium ovat muutenkin hyvin samankaltaisia toiminnallisuudeltaan,

saatavuudeltaan ja integroituvuudeltaan. Molemmista löytyy tarvittavat toiminnot olennaisen testauksen tekoon ja molemmat työkalut tuottavat raportointiin tarvittavat tiedostot. Koska Espresso- ja Robotium-kehykset integroidaan suoraan Android-projektiin, testejä on helppo suorittaa myöhemmin CI-palvelimella. Robotiumin arviointia alentavana tekijänä on Espressoa niukempi dokumentointi ja mahdollinen tuen loppuminen. Robotium-kehystä on päivitetty viimeksi vuonna 2016, kun taas muita tässä tutkimuksessa testattuja työkaluja päivitetään jatkuvasti.

Appium eroaa Espressosta ja Robotiumista siten, ettei sitä integroida Android-projektiin, vaan sitä kehitetään itsenäisenä projektina. Appium on järjestelmäriippumaton testauskehys, eli sillä voidaan testata sekä Android-, että iOS-sovelluksia. Tällä ominaisuudella on mittavia etuja, mutta myös haittoja. Ensinnäkin Appium-kehyksen käytön aloitus on huomattavasti työläämpää verrattuna muihin työkaluihin. Toiseksi Appium tarvitsee toimiakseen oman palvelimen, sekä erillisen projektin. Appium on kohtuullisen hyvin dokumentoitu, mutta sen käyttöönotto ja opittavuus eivät kuitenkaan olleet muiden työkalujen tasolla. Appium on myös muihin työkaluihin nähden jopa erittäin hidas. Eräällä testikerralla testitapausten läpikäynnissä kului Appium-kehyksellä noin 30 sekuntia, kun Espresso suoritti samat testit viidessä sekunnissa. Appium-testejä ei ole myöskään yhtä helppoa integroida CI-palvelimelle. Integrointi ja testien raportointi ovat mahdollisia, mutta vaativat enemmän työtä muihin työkaluihin verrattuna. Tämä johtuu pääosin siitä, että Appium-kehyksen testit ovat erillinen kokonaisuus itse Android-projektista.

Appiumilla voidaan kuitenkin saavuttaa huomattavia etuja sen järjestelmäriippumattoman luonteen takia. Appium-kehyksen testejä voidaan kirjoittaa kuudella eri ohjelmointikielellä ja samoja testejä voidaan hyödyntää niin sovelluksen Android-, kuin iOS-versiossa. (Appium 2017.) Yleensä mobiilisovellusprojekteissa tehdään versiot Androidille ja iOS:lle ja nämä versiot voivat olla lähes identtisiä. Jos yhtä testiä voidaan käyttää molemmilla alustoilla, voidaan saavuttaa huomattavia kustannussäästöjä. Toisaalta erillisen testausprojektin ylläpito voi tuottaa lisää työtä.

6 JOHTOPÄÄTÖKSET

Kandidaatintyön alussa esitettiin tutkimuskysymys ja sille kolme alakysymystä. Tämän tutkimuksen pääasiallisena tarkoituksena oli tutkia Android-sovellusten testausautomaation parhaita käytänteitä ja työkaluja, niiden soveltuvuutta OCTO3 Oy:n kaltaiseen pk-yritykseen ja testausautomaation roolia DevOps-kulttuurissa. Tutkimuskysymyksiin vastattiin kirjallisuuden, kyselyn ja empiirisen tutkimuksen perusteella. Seuraavaksi käydään läpi tutkimuskysymykset yksi kerrallaan ja vastataan niihin tässä työssä tehdyn tutkimuksen perusteella.

Ensimmäisellä kysymyksellä pyrittiin selvittämään parhaita Android-sovellusten testausautomaation käytänteitä ohjelmistoalan pk-yrityksessä. Parhaita käytänteitä on haastavaa määritellä täydellisesti, sillä käytänteet ja tekniikat kehittyvät jatkuvasti. Tutkimuksessa selvisi, että testausautomaation käyttö tulisi aina päättää projektikohtaisesti. Pienemmissä tai muuten testausautomaatiolle sopimattomissa projekteissa ei ole välttämättä järkevää rakentaa testausautomaation vaatimaa ympäristöä tai itse testejä. Testausautomaation käytöstä kannattaa sopia asiakkaan kanssa, sillä siitä syntyy pienissä projekteissa mittavia lisäkustannuksia manuaaliseen testaukseen verrattuna.

Jos testausautomaatiota päätetään käyttää, olisi hyvä, että yrityksellä olisi valmis ohjeistus sen aloittamiseen. Jokaisen ohjelmistosuunnittelijan ei kannata tutkia uudestaan samoja asioita. Ohjeistuksessa voitaisiin käydä läpi ainakin ympäristön pystytys ja suositeltujen työkalujen käyttöönotto. Ympäristössä kannattaa käyttää CI-palvelinta, joka on sijoitettu sitä varten tarkoitetulle tietokoneelle. Tällä voidaan helpottaa testitapausten ajoa taustalla, eikä rasitus kohdistu työntekijän henkilökohtaiseen tietokoneeseen. Myös testitapausten suorituksen raportointi ja koodikattavuuden mittaus kannattaa hoitaa CI-palvelimella, josta raportit voidaan käydä tarkastamassa testauksen jälkeen. Käytettävien työkalujen ja ympäristön valinnassa kannattaa huomioida työntekijöiden aikaisempi kokemus. Jos edes osa työntekijöistä on käyttänyt aikaisemmin jotain tiettyä työkalua, säästetään heti opetteluun käytettyä aikaa.

Toisella kysymyksellä selvitettiin parhaiden käytäntöjen sovellusta käytännössä OCTO3 Oy:n kaltaisessa pk-yrityksessä. Ensimmäiseksi yrityksen kannattaa panostaa sisäiseen CI-palvelimeen, jotta testausautomaatiolle saadaan kätevä pohja. Yleinen ohjeistus koetaan tärkeäksi tekijäksi, joka kannustaisi testausautomaation käyttöön. Testausautomaation hyödyt tunnistetaan selvästi, mutta testausautomaation aloitus koetaan työlääksi, osittain puuttuvan ohjeistuksen takia. Saatavilla on ulkopuolisten ihmisten tekemiä ohjeistuksia Android-sovellusten testausautomaatiokokonaisuuksista, mutta ne voivat olla vanhentuneita tai vaikeasti ymmärrettäviä. Siksi olisi hyvä, että yritys tekisi sisäiset ohjeet testausautomaation aloittamiseen. Jos Android-projektissa päätetään käyttää testausautomaatiota, näistä ohjeista olisi helposti nähtävissä työvaiheet ja vaatimukset sen käyttöön. Ohjeita tulee kuitenkin ylläpitää. Ohjelmistoja täytyy päivittää ja jopa vaihtaa mahdollisten yhteensopivuusongelmien takia. Yrityksen kannattaa antaa vastuu ohjeiden ja ympäristöjen ylläpidosta työntekijöille, jotta voidaan taata ohjeiden ajantasaisuus uuden projektin alkaessa. Yrityksen koosta ja ympäristöjen monimuotoisuudesta riippuen vastuullisia työntekijöitä voi olla esimerkiksi yhdestä viiteen. Vastuulliset työntekijät kannattaa valita mielenkiinnon ja kokemuksen perusteella. Yritys voi myös palkata uuden työntekijän juuri tällaiseen työtehtävään. Uusi työntekijä voisi mahdollisesti ylläpitää myös muiden ympäristöjen laajaa DevOps-kokonaisuutta ja ohjeistuksia. Työntekijä voisi lisäksi kouluttaa muita työntekijöitä aiheesta. Tämän tutkimuksen tuloksia ei sovellettu muihin, OCTO3 Oy:n kaltaisiin yrityksiin. Tästä syystä kannattaisi tehdä jatkotutkimusta, jossa tutkimusaineistoa ja tuloksia laajennetaan muihin ohjelmistoalan pk-yrityksiin.

Kolmannella kysymyksellä haluttiin kartoittaa, mitkä työkalut ovat osa parhaita käytänteitä. Tutkimus on suoritettu pääosin empiirisenä. Yleinen johtopäätös suoritetusta tutkimuksesta on se, että kannattaa käyttää parhaiten tuettuja työkaluja. Usein ne ovat niitä, jotka ovat laajimmin käytössä ja näin ollen apua käyttöön on helposti saatavilla. Tässä tutkimuksessa parhaiten tuetut työkalut suoriutuivat keskimääräisesti parhaiten arvioinnissa. Yksikkötestauksen työkaluista JUnit on kirjoitushetkellä Android-dokumentaation suosittelema. Robolectric pärjäsi myös hyvin evaluoinnissa. Se, kumpaa työkalua kannattaa käyttää, riippuu projektin rakenteesta. Jos yksikkötestauksen kohteet ovat selkeästi Android-alustasta erillisiä metodeja, eikä niissä käytetä Android-alustan omia kirjastoja, on JUnit suositeltavampi. Jos taas testauskohteet ovat sekalaisia ja niissä on myös Androidin omia

metodeja, on Robolectric-kehyksen käyttö suositeltavaa. Näin vältytään jäljitellyn datan luomiseen tarvittavan kirjaston käytöltä.

Laitetestauksen kehyksistä Espresso on vertailussa vahvin. Myös se on JUnit-kehyksen tapaan esitelty Android-dokumentaatiossa ja on tällä hetkellä yksi suosituimmista laitetestauksen kehyksistä. Kannattaa kuitenkin harkita Appium-kehyksen käyttöä, jos yrityksellä on paljon projekteja, joissa sovelluksista tehdään sekä Android-, että iOS-versio. Haittapuolena on se, että Appium-kehyksen käyttöönotto vaatii huomattavasti enemmän resursseja Espressoon verrattuna. Ajallisesti Appium-kehystä käyttäessä ensimmäisen testin kirjoitus ja ajo kesti ainakin 10 kertaa kauemmin Espresso-kehykseen verrattuna, johtuen vaikeammasta opittavuudesta ja tarpeesta uudelle ympäristölle. Jos Android-sovellusta kehitetään natiivisti, tutkimuksen perusteella Espresso-kehys on suositeltavampi sen helppokäyttöisyyden ja virallisen tuen takia. CI-palvelimena Jenkins pärjäsi tutkimuksessa hyvin.

Viimeisellä kysymyksellä selvitettiin testausautomaation roolia DevOps-kulttuurissa. Testausautomaatio osoittautui odotetusti olennaiseksi osaksi DevOps-toimintatapoja. Testausautomaatiolla saadaan nopeasti tuloksia kaikkien osapuolten nähtäville ja sillä pystytään varmistamaan tuotteen laatu. Myös muutosten tekeminen havaittiin turvallisemmaksi, jos testausautomaatio on käytössä. Testausautomaatio integroituu CI-palvelimeen, joka on olennainen osa DevOps-kulttuuria. Tässä työssä toteutetun kirjallisuuskatsauksen mukaan vaikuttaa siltä, että tutkimusta DevOps-toimintojen rakentamisesta esimerkiksi pk-yrityksille ei ole juurikaan saatavilla. Kirjallisuudessa todetaan, että DevOps-kulttuuri tulee rakentaa aina yrityskohtaisesti, mutta oikeat tavat tietyille yritystyypeille jäivät kysymyksen tasolle.

Tämän tutkimuksen tuloksia ei kannata yleistää liikaa. Tutkimus on rajoittunut vain yhteen yritykseen, mikä korostaa eroja yritysten välillä. Vaikka yritykset toimisivat samalla toimialalla ja samojen asiakkaiden parissa, voivat niiden sisäiset ominaisuudet, kuten käytettävät tuotantoprosessit vaihdella huomattavasti. Jos yrityksellä on jo vakiintuneet prosessit testausautomaation toteuttamiseen, voi niiden muuttaminen vaatia laajempaa taustatyötä ja tutkimusta. Toisaalta tässä tutkimuksessa evaluoidut, parhaimpiin käytäntöihin

kuuluvat testausautomaation työkalut ovat todennäköisesti sovellettavissa myös OCTO3 Oy:n ulkopuolella. Vaikka evaluointikriteerit valittiin OCTO3 Oy:n työntekijöiden tekemän arvioinnin perusteella, ovat valitut kriteerit yleispäteviä ja niiden avulla saatiin luotettava arvio suositeltavimmista työkaluista. Jos yrityksellä kuitenkin on vakiintuneet ympäristöt käytössä, täytyy työkalujen osalta tehdä lisätutkimusta siitä, miten ne soveltuvat kyseiseen ympäristöön ja miten ne saadaan integroitua muiden käytettyjen työkalujen kanssa. Isomman yrityksen tapauksessa työkalut valitaan todennäköisesti monivaiheisen valintaprosessin kautta, jolloin tässä työssä esitetty tutkimus ei välttämättä riitä perustelemaan työkalun paremmuutta.

Tässä tutkimuksessa laitetestaus on toteutettu testausympäristöön kytketyillä Android-emulaattoreilla ja fyysisillä Android-laitteilla. Jatkotutkimukselle olisi tarvetta maksullisten testauspalvelujen evaluoinnissa. Jos testit voidaan suorittaa helposti fyysisillä laitteilla pilvipalvelun kautta maksua vastaan, se vähentäisi oman testilaitteiston ylläpidon tarvetta yrityksen sisällä. Jatkotutkimusta voisi tehdä myös Android-sovellusten testausautomaation käyttöönoton kustannuksista ja vaikutuksista oikeissa ohjelmistoprojekteissa. Tässä työssä keskityttiin toiminnallisten vaatimusten testaukseen, mutta jatkotutkimuksissa voisi ottaa kantaa myös ei-toiminnallisten vaatimusten, kuten suorituskyvyn testauksen automatisointiin. Yhtenä jatkotutkimuskohteena voisi myös olla asiakkaiden osallistaminen testausautomaation kehitykseen. Asiakkaat voisivat auttaa kohdentamaan testausta juuri niihin osa-alueisiin, joissa on tyypillisesti eniten kriittisimpiä ongelmia.

Tätä työtä vastaavaa tutkimusta toteutettaessa toiselle ohjelmistoalan pk-yritykselle kannattaa huomioida työntekijöiden tarpeet ja aikaisemmat kokemukset testausautomaatiosta. Kuten tässä tutkimuksessa aikaisemmin todettu, työntekijät voivat olla saman toimialan yrityksissä erilaisia, joten kyselytutkimus kannattaa toteuttaa huolella ja mahdollisesti hyödyntää myös haastatteluja tarkempien tutkimustulosten saavuttamiseksi. Työkaluja valittaessa kannattaa keskittyä yrityksen tarpeisiin ja projektien tyypillisiin piirteisiin. Joistain työkaluista voi olla toisia enemmän etua, jos yritys tekee esimerkiksi verkkopohjaisia sovelluksia useille eri mobiilialustoille. Työkalujen tarjonta pitää myös muistaa tarkistaa tutkimusta aloittaessa, sillä mitä todennäköisimmin tarjolla on jo jotain uusia, hyvin toimivia ratkaisuja, joita ei nostettu esille tässä tutkimuksessa.

LÄHTEET

Amaradri, A. S. & Nutalapati, S. B. Continuous Integration, Deployment and Testing in DevOps Environment. Faculty of Computing, Blekinge Institute of Technology. Karlskrona, Sweden. 115 s.

Appdiff. 2016. Appdiff Launches First AI-Powered Mobile Testing Platform. [verkkodokumentti]. [viitattu 17.12.2017]. Saatavissa <https://blog.appdiff.com/appdiff-launches-first-ai-powered-mobile-testing-platform-15002908873e>

Appium. 2017. About Appium. [verkkodokumentti]. [viitattu 5.12.2017]. Saatavissa <http://appium.io/slate/en/master>

Atlassian. 2017. What is DevOps? [verkkodokumentti]. [viitattu 23.11.2017]. Saatavissa <https://www.atlassian.com/devops>

Baride, S. & Dutta, K. 2011. A Cloud based Software Testing Paradigm for Mobile Applications. ACM SIGSOFT Software Engineering Notes. Volyymi n:o 36. Lehti n:o 3, s. 1-4.

Brown, A. W. & Wallnau, K. C. 1996. A Framework for Evaluating Software Technology. IEEE Software. Volyymi n:o 13. Lehti n:o 5, s. 39-49.

Cagan, M. 2006. Market Research Capabilities and Limitations. [verkkodokumentti]. [viitattu 3.12.2017]. Saatavissa <https://svpg.com/market-research-capabilities-and-limitations/>

Ebert, C., Gallardo, G., Hernantes, J., Serrano, N. 2016. DevOps. IEEE Software. Volyymi n:o 33. Lehti n:o 3, s. 94-100.

Feitelson, D. G., Frachtenberg, E., Beck, K. L. 2013. Development and Deployment at Facebook. IEEE Internet Computing. Volyymi n:o 17. Lehti n:o 4, s. 8-17.

Fewster, M. 2001. Common Mistakes in Test Automation. 2001. Grove Consultants.

Fowler, F. J. 1995. Improving Survey Questions: Design and Evaluation. Thousand Oaks: Sage Publications, Inc. 193 s.

Furtado, A. P. C. C., Meira, S. R. L. & Gomes, M. W. 2014. Towards a Maturity Model in Software Testing Automation. Teoksessa: Mannaert, H., Lavazza, L., Oberhauser, R., Kajko-Mattsson, M. & Gebhart, M. ICSEA 2014: The Ninth International Conference on Software Engineering Advances. Nice, France. 12-16.10.2014. s. 282–285.

Gao, J., Tsai, W., Paul, R., Bai, X. & Uehara, T. 2014. Mobile Testing-As-A-Service (MTaaS) – Infrastructures, Issues, Solutions and Needs. Teoksessa: Clarke, P. J., Zulkernine, M., Paul, R. A. & Xu, J. 2014 IEEE 15th International Symposium on High-Assurance Systems Engineering. Miami, Florida, USA. 9.–11.1.2014. s. 158–167.

Google. 2017. Firebase Test Lab for Android. [verkkodokumentti]. [viitattu 25.10.2017]. Saatavissa <https://firebase.google.com/docs/test-lab/>

Google & Open Handset Alliance. 2017. Test Your App. [verkkodokumentti]. [viitattu 23.10.2017]. Saatavissa <https://developer.android.com/studio/test/index.html>

Gotimer, G. & Stiehm, T. 2016. DevOps Advantages for Testing: Increasing Quality through Continous Delivery. CrossTalk. Volyymi n:o 29. Lehti n:o 3, s. 13-18.

Hellman, E. 2014. Android Programming: Pushing the Limits. Hoboken: John Wiley & Sons, Inc. 432 s.

Hoog, A. 2011. Android Forensics: Investigation, Analysis and Mobile Security for Google Android. Waltham: Elsevier, Inc. 371 s.

Humble, J. & Farley, D. 2011. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Upper Saddle River: Addison-Wesley. 512 s.

ISO & IEC. 2011. ISO/IEC 25010. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Geneva: International Organization of Standardization.

Kasurinen, J., Taipale, O., Smolander, K. 2010. Software Test Automation in Practice: Empirical Observations. Advances in Software Engineering. Volyymi 2010. 18 s.

Kasurinen, J. P. 2013. Ohjelmistotestauksen käsikirja. 1. p. Jyväskylä: Docendo. 236 s.

Kaufman, J. 2017. What is the 'Paradox of Automation'? [verkkodokumentti]. [viitattu 23.10.2017]. Saatavissa <https://personalmba.com/paradox-of-automation/>

Kitchenham, B. 1996. DESMET: A method for evaluating Software Engineering methods and tools. Keele: Department of Computer Science, University of Keele. Raportti n:o TR96-09. 52 s.

Lopes, R. 2017. Solutions for Automated Mobile Testing Cloud Service. Teoksessa: Sousa, A. A. & Oliveira, E. Proceedings of the 12th Doctoral Symposium in Informatics Engineering. Porto, Portugal. 31.1.2017. s. 30–40.

Lämsä, T. 2017. Comparison of GUI testing tools for Android applications. [diplomityö]. Oulun yliopisto, Tietojenkäsittelytieteen laitos. Oulu. 105 s.

Macharla, P. 2017. Android Continuous Integration: Build-Deploy-Test Automation for Android Mobile Apps. Berkeley: Apress. 152 s.

Marick, B. 1998. When Should a Test Be Automated? Teoksessa: Proceedings of the 11th International Software/Internet Quality Week, s. 1-20.

Merrill, P. 2017. Will AI bots steal your QA testing job? [verkkodokumentti]. [viitattu 17.12.2017]. Saatavissa <https://techbeacon.com/will-ai-bots-steal-your-qa-testing-job>

Mäkinen, S. & Münch, J. 2014. Effects of Test-Driven Development: A Comparative Analysis of Empirical Studies. Kirjassa: Software Quality: Model-Based Approaches for Advanced Software and Systems Engineering. Cham: Springer, s. 155-169.

Nikula, U. 2017. Testausautomaatio ja testauksen työkalut. Opintojakso CT60A0220, C-ohjelmoinnin ja testauksen periaatteet, Lappeenrannan teknillinen yliopisto, School of Business and Management, Tietotekniikan koulutusohjelma, lukuvuosi 2016-2017. [luentokalvot].

Nolan, G. 2015. Agile Android. Berkeley: Apress. 102 s.

Nolan G., Cinar O., Truxall, D. 2014. Android Best Practices. Berkeley: Apress. 223 s.

OCTO3. 2017a. OCTO3:n uusi Helsingin toimisto! [verkkodokumentti]. [viitattu 15.10.2017]. Saatavissa <http://www.octo3.fi/uutiset-octo3-uusi-helsingin-toimisto/>

OCTO3. 2017b. Referenssit. [OCTO3:n www-sivuilla]. [viitattu 15.10.2017]. Saatavissa <http://www.octo3.fi/referenssit/>

OpenSignal. 2015. Android Fragmentation Visualized. [verkkodokumentti]. [viitattu 24.10.2017]. Saatavissa <https://opensignal.com/reports/2015/08/android-fragmentation/>

Poston, R. M. & Sexton, M. P. 1992. Evaluating and Selecting Testing Tools. IEEE Software. Volyymi n:o 9. Lehti n:o 3, s. 33-42.

Rafi, D. M., Moses, K. R. K., Petersen, K. & Mäntylä, M. V. 2012. Teoksessa: Proceedings of the 7th International Workshop on Automation of Software Test. Zurich, Switzerland. 2-3.6.2012. Piscataway: IEEE Press. s. 36-42.

Ravichandran, A., Taylor, K., Waterhouse, P. 2016. DevOps for Digital Leaders: Reignite Business with a Modern DevOps-Enabled Software Factory. Berkeley: Apress. 173 s.

Robolectric. 2017. Robolectric: test-drive your Android code. [verkkodokumentti]. [viitattu 5.12.2017]. Saatavissa <http://robolectric.org/>

Roche, James. 2013. Adopting DevOps Practices in Quality Assurance. Communications of the ACM. Volyymi n:o 56. Lehti n:o 11, s. 38-43.

Runeson, P. & Höst M. 2008. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering. Volyymi n:o 14. Lehti n:o 2, s. 131-164.

Sinhal, A. 2017. Closer Look At Android Runtime: DVM vs ART. [verkkodokumentti]. [viitattu 17.12.2017]. Saatavissa <https://android.jlelse.eu/closer-look-at-android-runtime-dvm-vs-art-1dc5240c3924>

Ståhl, D. & Bosch, J. 2014. Continuous Integration Flows. Kirjassa: Continuous Software Engineering. Cham: Springer, s. 107-115.

Sundar, Pichai. 2015. You say you want a mobile revolution… [verkkodokumentti]. [viitattu 3.12.2017]. Saatavissa <https://googleblog.blogspot.fi/2015/05/io-2015-mobile-revolution.html>

Toivanen, I. 2017. OCTO3 DevOps. [PowerPoint-esitys].

Visma Consulting. 2017. Visma osti OCTO3 Oy:n ja jatkaa vahvaa kasvua IT-konsultointimarkkinalla. [lehdistötiedote]. [viitattu 15.10.2017] Saatavissa <http://media.visma.fi/pressreleases/visma-osti-octo3-oy-n-ja-jatkaa-vahvaa-kasvua-it-konsultointimarkkinalla-2191826>

Xamarin Inc. 2017. Xamarin Test Cloud: Mobile app testing made easy. [verkkodokumentti]. [viitattu 25.10.2017]. Saatavissa <https://www.xamarin.com/test-cloud>

Zhu, L., Bass, L. & Champlin-Scharff, G. 2016. DevOps and Its Practices. IEEE Software. Volyymi n:o 33. Lehti n:o 3, s. 32-34.

LIITE 1. Kysely

Tästä liitteestä on jätetty kuvaukset, selittävät tekstit ja kuvat pois.

Kyllä/en -vastaukset.

Oletko ohjelmistosuunnittelija tai -kehittäjä?

Oletko ollut mukana Android-projekteissa?

Vastaus asteikolla 1–5, missä 1: ”En missään projektissa” ja 5: ”Kaikissa projekteissa”.

Olen käyttänyt testausautomaation työkaluja Android-projekteissa.

Olen käyttänyt testausautomaation työkaluja muissa projekteissa.

Kyllä/en vastaus.

Käyttäisin työkaluja (enemmän), jos OCTO3 Oy:llä olisi olemassa selkeät käytännöt ja ohjeet niiden käyttämiseen (esim. wiki-sivu).

Vastaus asteikolla 1–5, missä 1: ”Ei pidä lainkaan paikkaansa” ja 5: ”Pitää täysin paikkansa”.

Olen käyttänyt testausautomaation työkaluja, koska

... se on ollut projektivaatimus.

... siitä on hyötyä.

... automatisointi säästää aikaa (ja kustannuksia).

... automatisointi parantaa tuotteen laatua (ja asiakastyytyväisyyttä).

... se on luonnollista.

En ole käyttänyt testausautomaation työkaluja, koska

... sitä ei ole vaadittu.

... automatisoinnin aloitus vie liikaa aikaa.

... automatisoinnista ei ole hyötyä.

... en tiedä, miten automatisoinnin voisi toteuttaa (en tunne työkaluja).

... työkaluja on vaikea käyttää.

(jatkuu)

LIITE 1. (jatkoa)

Vastaus asteikolla 1–5, missä 1: ”Ei lainkaan tärkeä” ja 5: ”Välttämätön”.

Arvioi ISO/IEC 25010 -standardin piirteiden tärkeyttä testausautomaation työkalujen kontekstissa (jokaiselle piirteelle oli lyhyt suomenkielinen kuvaus).

Functional completeness	Functional correctness	Functional appropriateness
Maturity	Availability	Fault tolerance
Recoverability	Time behaviour	Resource utilization
Capacity	Confidentiality	Integrity
Non-repudiation	Accountability	Authenticity
Co-existence	Interoperability	Modularity
Reusability	Analysability	Modifiability
Testability	Appropriateness recognizability	Learnability
Operability	User error protection	User interface aesthetics
Accessibility	Adaptability	Installability
Replaceability		

Vastaukset annettiin valitsemalla valintaruutu.

Olen kuullut seuraavista työkaluista…

Olen käyttänyt seuraavia työkaluja…

JUnit	Robotium	Espresso
Calabash	UI Automator	Appium
Katalon Studio	Hamcrest	JaCoCo
Robolectric		

Kyselyssä oli myös mahdollisuus vapaaseen kommentointiin jokaisessa kohdassa.