Riku Lahdenperä

Miten testiautomaatiota hyödynnetään ohjelmistokehityksessä?

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelman

Insinöörityö

6.11.2022

Tiivistelmä

Tekijä:	Riku Lahdenperä
Otsikko:	Miten testiautomaatiota hyödynnetään ohjelmistokehityksessä?
Sivumäärä:	28 sivua
Aika:	6.11.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine:	Hyvinvointi- ja terveysteknologia
Ohjaajat:	Senior sovelluskehittäjä Laura Moilanen Yliopettaja Päivi Haho

---

Opinnäytetyön tarkoituksena oli selvittää, miten testiautomaatiota hyödynnetään, tarkastella testiautomaatiosta saatavia hyötyjä asiakas- ja potilastietojärjestelmän kehityksen kannalta sekä kehittää testiautomaatiotapauksia järjestelmässä kulkevien lähetteiden regressiotestitapauksien pohjalta.

Työn toteutus jaettiin kahteen osaan: tutkimus- ja kehitystyöhön. Tutkimustyö toteutettiin etsimällä erilaisia aineistoja ja lähteitä, joiden avulla opinnäytetyön tekijä pystyi perehtymään syvällisesti testiautomaation maailmaan. Käytännönläheinen kehitystyö toteutettiin Ranorex-nimisellä testiautomaatio-ohjelmistolla.

Kehitystyö aloitettiin valitsemalla regressiotestitapauksien joukosta automatisoitavia testitapauksia. Testitapauksien valinnassa painotettiin samankaltaisia testitapauksia, joissa olisi paljon askelia, jolloin automatisoinnilla saatavat hyödyt olisivat mahdollisimman suuret. Testitapauksien valinnan jälkeen aloitettiin automatisointi Ranorex-ohjelmalla.

Kehitys- ja tutkimustyön seurauksena selvisi, että testiautomaatio soveltuu parhaiten suurien ja pitkäikäisten ohjelmistoprojektien hyödynnettäväksi. Alkuvaiheessa testiautomaation kehittäminen ja käyttöönotto vievät paljon resursseja, minkä takia manuaalisesti tehtävä testaaminen sopii joihinkin projekteihin paremmin. Huomioitavaa on myös, ettei testiautomaatiolla voida korvata kaikkia manuaalisesti tehtäviä testejä.

Opinnäytetyöstä saatujen tuloksien perusteella pystyttiin päättelemään, että testiautomaatio tuo paljon erilaisia hyötyjä suurille ja pitkäikäisille ohjelmistoprojekteille. Testiautomaatio tarvitsee rinnalleen manuaalisesti tehtäviä testejä mahdollisimman hyvän testauskattavuuden saavuttamiseksi.

Avainsanat: testiautomaatio, potilastietojärjestelmä, regressiotestaus, automaatio

Abstract

Author:	Riku Lahdenperä
Title:	Use of Test Automation in Software Development
Number of Pages:	28 pages
Date:	6 November 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Health Technology
Supervisors:	Laura Moilanen, Senior Application Developer Päivi Haho, Principal lecturer

---

The purpose of the study was to find out how test automation is utilized, to examine the benefits of test automation in terms of the software development of the client and patient information system, and to develop test automation cases based on regression test cases of referrals passing through the client and patient information system.

The implementation of the study was divided into two parts, research and development. The research was carried out by searching for various materials and sources, which allowed the author to become deeply familiar with the world of test automation. Practical development work was carried out with test automation software called Ranorex.

The development started by selecting the test cases to be automated among the regression test cases. In the selection of test cases, emphasis was placed on similar test cases with several steps, in which case the benefits obtained by automation would be as substantial as possible. After the selection of the test cases, the automation was started with the Ranorex software.

As a result of the development and research work, it became clear that test automation is best suited for use in large and long-lived software projects. In the initial phase, the development and implementation of test automation takes a lot of resources, which is why manual testing is better suited in some projects. It is also worth noting that test automation cannot replace all manual tests.

Based on the results obtained, it was possible to conclude that test automation brings many different benefits to large and long-lived software projects. Test automation needs to be accompanied by manually performed tests to achieve the best possible testing coverage.

Keywords: Test Automation, Electronic Medical Record, Regression testing, Automation

Sisällys

1 Johdanto 1

2 Kohdeyritys ja yhteistyökumppanit 2

3 Tutkimuskysymykset ja menetelmä 3

4 Ohjelmiston kehitys ja testaus 4

4.1 Ketterä kehitys 5
4.2 Testauksen pääluokat 6
4.3 Testauksen eri menetelmät 8
4.4 Testauksen eri tasot 8
4.5 Regressiotestaus 11
4.6 Testiautomaatio 13
4.6.1 Testiautomaation hyödyt 15
4.6.2 Testiautomaation haasteet 15

5 Automaatiotestien tekeminen 16

5.1 Ranorex Studio ja Jenkins 17
5.2 Automatisoitavien regressiotestien valinta 18
5.3 Regressiotestien automatisointi 20

6 Automaatiotestauksen hyödyt 22

7 Pohdinta ja yhteenveto 24

Lähteet 26

1 Johdanto

Tässä opinnäytetyössä käsitellään Apotti-asiakas- ja potilastietojärjestelmän lähetetyönkulkujen virheiden havaitsemiseen käytettävien regressiotestien automatisointia ja siitä saatavaa hyötyä järjestelmän kehittämisen kannalta. Tarkoituksena on valita manuaalisesti tehtävien regressiotestitapauksien joukosta testejä, jotka automatisoidaan Ranorex-nimisellä ohjelmalla. Lopuksi pohditaan testiautomaatiosta saatavia hyötyjä.

Opinnäytetyö tehdään yritykselle Oy Apotti Ab, jonka omistavat Helsingin ja Uudenmaan sairaanhoitopiiri (HUS) ja useat eri kunnat kuten Helsinki, Vantaa ja Kerava. Apotin tuotteita ovat modulaarinen Apotti-asiakas- ja potilastietojärjestelmä ja sen tunnetuin sovellus Maisa-asiakasportaali.

Apotti-järjestelmää kehitetään jatkuvasti käyttäjien tarpeiden ja lakisääteisten vaatimuksien mukaan. Jatkuva kehittäminen vaatii paljon resursseja sidottavaksi kehitystyöhön, mikä vähentää ylläpitotyöhön käytettävien resurssien määrää. Järjestelmän kehittyessä testaamisen tärkeys korostuu entisestään, sillä järjestelmässä olevien riippuvuuksien ja kriittisten osa-alueiden määrä kasvaa. Toisin sanoen järjestelmän monimutkaisuuden kasvaessa myös mahdollisten häiriötilanteiden vaikutusalue kasvaa.

Lähete on lääkärin kirjoittama sähköinen tai paperinen asiakirja, jonka avulla varmistetaan erikoissairaanhoitoa tai tarkempia tutkimuksia tarvitsevan potilaan hoitoonpääsy. Apotti-järjestelmässä käyttäjä kirjaa lähetteen potilaalle, jolloin lähete kulkeutuu vastaanottavaan yksikköön. Lähetetyönkulkujen regressiotestaaminen on tärkeää, sillä vakava virhe voisi pahimmillaan estää potilaan pääsyn kiireelliseen hoitoon ja siten vaarantaa potilasturvallisuuden.

2 Kohdeyritys ja yhteistyökumppanit

Apotti-hanke alkoi vuonna 2012 suunnitteluvaiheella, jolloin hankkeelle luotiin perusta sekä määritettiin tavoitteet, aikataulu ja resurssit. Hankkeen tarkoituksena oli rakentaa maailman ensimmäinen sosiaali- ja terveydenhuollon yhdistävä asiakas- ja potilastietojärjestelmä. Tavoitteena oli kehittää ja yhtenäistää toimintatapoja sosiaali- ja terveyspalveluiden kustannuksien vähentämiseksi. [1; 2.]

Apotti-järjestelmän kehittämiseksi ja ylläpitämiseksi perustettiin vuonna 2015 Oy Apotti Ab, jonka omistajia ovat HUS Helsingin yliopistollinen sairaala, Helsinki, Vantaa, Kirkkonummi, Kauniainen, Kerava, Inkoo, Loviisa, Tuusula ja Siuntio. Apotti työllistää yli 550 henkilöä. [1; 2.]

HUS:n ja sen alueella toimivien kuntien käytössä oli satoja erilaisia tietojärjestelmiä, jotka eivät kommunikoineet keskenään. Järjestelmien välisen kommunikaation puute aiheutti potilastietojen puuttumisen hoito- ja palvelutilanteissa. Apotti-järjestelmän tarkoituksena oli korvata osa käytössä olevista järjestelmistä ja parantaa asiakas- ja potilastietojen liikkumista Apotti-järjestelmän ja käyttöön jääneiden järjestelmien välillä, mikä parantaisi palveluiden laatua sekä vähentäisi kustannuksia. [1; 2.]

Vuonna 2013 järjestetyssä hankintamenettelyssä Apotti-järjestelmän toimittajaksi valittiin yhdysvaltalainen Epic Systems Corporation, jonka Epic-järjestelmä katsottiin parhaiten täyttävän hankinnassa asetetut kriteerit. Valinnassa käytettiin kriteereinä hintaa ja laadullisia kriteereitä, joissa painopisteenä olivat käytettävyys, joustavuus, avoimuus ja mukautettavuus. Epic-järjestelmä muokattiin sopimaan Suomessa toimivaksi, jolloin siitä syntyi Apotti-järjestelmä. Epic-järjestelmää käytetään maailmanlaajuisesti yli 350 organisaatiossa. Apotti tekee jatkuvaa yhteistyötä etenkin muiden Euroopassa toimivien Epic-järjestelmää käyttävien organisaatioiden kanssa, joita löytyy esimerkiksi Tanskasta, Hollannista ja Englannista. [1; 2.]

Hankintavaiheessa käyttöpalveluiden toimittajaksi valittiin Fujitsu Finland Oy. Valinta tehtiin noudattamalla julkista kilpailutusmenetelmää. Käyttöpalvelut tarkoittavat organisaatioiden kapasiteetti- ja ylläpitopalveluita, jotka voivat sisältää esimerkiksi konesalipalvelut, tietoliikenne- ja palomuuriratkaisut. [1; 2.]

3 Tutkimuskysymykset ja menetelmä

Opinnäytetyön päämääränä on perehtyä testiautomaation maailmaan syvällisesti tutkimalla aiheeseen liittyviä aineistoja ja käytännönläheisesti kehittämis- ja ylläpitotyön avulla. Tarkoitus on lisätä ymmärrystä ja osaamista testiautomaatiosta sekä samalla kehittää kohdeyrityksen käyttöön uusia testitapauksia. Asetettujen päämäärien saavuttamiseksi valittiin kaksi tutkimuskysymystä, joiden tarkoituksena on ohjata ja rajata opinnäytetyötä. Tutkimuskysymykset ovat:

- Miten testiautomaatiota hyödynnetään ohjelmistokehityksessä?
- Mitkä ovat testiautomaation hyödyt ohjelmistokehityksessä?

Opinnäytetyön tutkimusmenetelmänä käytetään konstruktiivista tutkimusotetta, joka on yksi tapa tehdä tapaustutkimus. Konstruktiivinen tutkimusote on kehitetty liiketaloustieteen alueelle, mutta sitä voidaan soveltaa muiden alojen käyttöön. Tarkoituksena on ratkaista tosielämän ongelma, testata ratkaisun toimivuus käytännössä ja reflektoida havaintoja takaisin olemassa olevaan teoriaan. [3.]

Potilastietojärjestelmät ovat erittäin laajoja kokonaisuuksia, joita kehitetään jatkuvasti vastaamaan paremmin käyttäjien toiveisiin, muuttuvien lakien määräyksiin tai muihin tarpeellisiin muutoksiin. Tosielämän ongelmana onkin, että jatkuva kehittäminen lisää järjestelmän testaamisen tarvetta. Manuaalisesti tehtävä testaus vie runsaasti järjestelmän ylläpidosta vastaavan tahon resursseja, joita voitaisiin hyödyntää organisaation muissa tärkeissä tehtävissä.

Ratkaisu ongelmaan on lisätä testiautomaation hyödyntämistä kohdeyrityksessä luomalla lisää testitapauksia sekä kasvattamalla testiautomaatiossa tarvittavaa osaamista. Testiautomaatiota hyödyntämällä pystytään vähentämään

testaamiseen tarvittavaa aikaa, lisäämään testauksen kattavuutta ja suorittamaan testaamista useammin kuin manuaalisesti tehtynä. Etenkin regressiotestien automatisoinnilla saadut hyödyt ovat suuret, sillä niiden tekeminen vie runsaasti aikaa, ja ne toistavat itseään, mikä manuaalisesti tehtynä lisää inhimillisen virheen mahdollisuutta.

Aikaisemmin saatuun teoriaan perusteellisesti perehtyminen on yksi konstruktiivisen tutkimusmenetelmän vaatimus. Aiheeseen perehtymisessä hyödynnetään tietokantahakuja, joiden tarkoituksena on löytää kattavaa ja laadukasta tietoa järjestelmän kehittämisestä, testaamisesta, testiautomaatiosta, sen hyödyistä, haitoista ja mahdollisesti kehitystyön aikana eteen tulevista sudenkuopista.

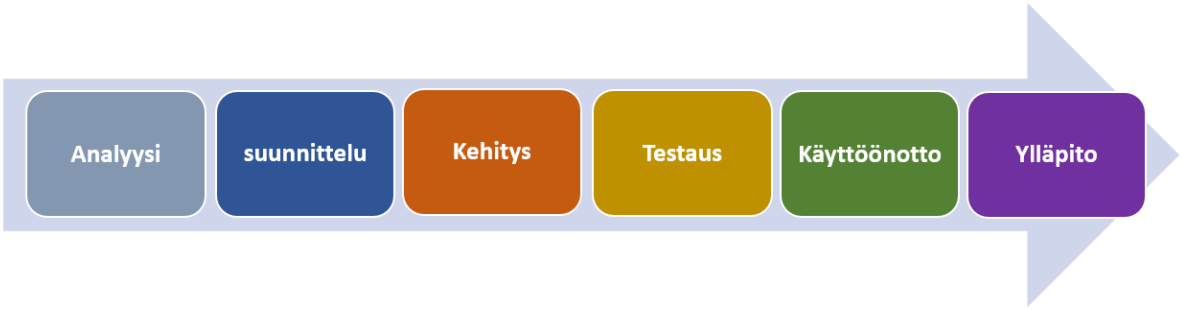
4 Ohjelmiston kehitys ja testaus

Erilaisista ohjelmistoista on tullut arkipäiväinen asia kaikkialla maailmassa. Ohjelmistojen yleistymisen ja kehittymisen myötä kilpailu, vaatimukset, asiakkaiden tarpeet ja laissa määritetyt säädökset ovat kasvaneet, mikä on johtanut ohjelmistokehityksessä käytettyjen menetelmien lisääntymiseen sekä muuttumiseen ketterämmiksi ja tehokkaammaksi kuin aikaisemmin. Teknologian kehittyminen ja tutkimuksista saatu tieto ovat lisänneet ohjelmistojen laatua, turvallisuutta ja käytettävyyttä.

Vaikka ohjelmistokehityksessä on otettu valtavia askelia eteenpäin, on kehityksen vaiheet pysyneet suhteellisen muuttumattomina. Kuten kuvassa 1 esitetään, on ohjelmistokehitys jakautunut kuuteen eri vaiheeseen, jotka ovat

- analyysivaihe, jonka aikana luodaan ohjelmistoprojektin perusta ja tehdään vaatimusmäärittely, käyttäjätutkimus, liiketoimintamalli, katsaus kilpailijoista ja budjettisuunnitelma.
- suunnitteluvaihe, jossa keskitytään ohjelmiston arkkitehtuurin suunnitteluun, asiakkaan tarpeisiin vastaamiseen ja ohjelmiston ulkonäön hahmottamiseen.
- kehitysvaihe, jonka aikana suunnitelman pohjalta luodaan toimiva ohjelmisto.

- testausvaihe, missä edellisessä vaiheessa luotu ohjelma testataan perusteellisesti.
- käyttöönottovaihe, jonka aikana ohjelmisto viedään oikeaan ympäristöön asiakkaan käyttöön.
- ylläpitovaihe, jossa ohjelmisto pidetään ajan tasalla ja korjataan mahdollisia virhetilanteita. [4.]



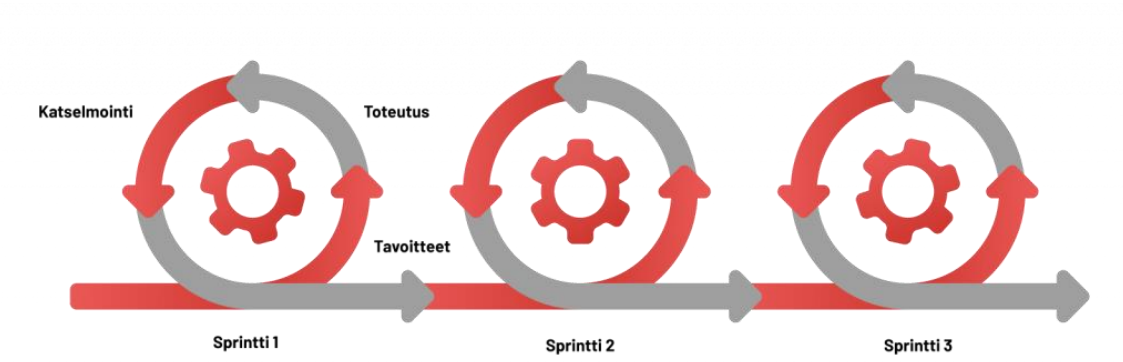
Kuva 1. Ohjelmistokehityksen vaiheet ovat analyysi, suunnittelu, kehitys, testaus, käyttöönotto ja ylläpito.

4.1 Ketterä kehitys

Ohjelmistojen kehittäminen toteutetaan usein jonkun ketterän kehittämisen mallin mukaan. Ketterän kehittämisen malleja luotiin vastapainoksi kurinalaisille ja suunnitelmapainotteisille menetelmille, joissa projektin alussa käytetään paljon aikaa suunnitelman luomiseen. Jälkeenpäin tulevat muutokset suunnitelmaan johtavat siihen, että projektin alussa tehty suunnittelu on ollut turhaa työtä. Etenkin suurien ohjelmistojen kehittämisessä tapahtuu suunnitelmien muutoksia, sillä suunnitteluvaiheessa ei aina tiedetä tarkalleen asiakkaan vaatimuksia ja tarpeita. Ketterien menetelmien ideana on poistaa kaikki turha työ ja keskittyä tekemään arvoa tuottavia asioita. [5.]

Ketterässä kehittämisessä tiimit työskentelevät iteraatioissa tai toiselta nimeltään sprinteissä. Sprintit ovat yhdestä neljään viikkoon kestäviä jaksoja, joissa tiimi käy kaikki sovelluskehityksen vaiheet läpi. Kuvasta 2 nähdään pelkistetty malli, kuinka sprintin alussa suunnitellaan tavoitteet, toteutetaan suunnitelma ja lopuksi katselmoidaan sprintin aikana tehty työ. Sprintin valmistuttua siirrytään seuraavaan sprinttiin, jossa samat vaiheet toistetaan uudelleen. Tiimin vastuulla

oleva kehitettävä tuote tai ominaisuus jaetaan pienemmiksi palasiksi, joita työstetään iteraatioittain. Kokonaisen tuotteen valmistumiseen voi kulua useampi iteraatio. [5.]



Kuva 2. Ketterän kehittämisen malleissa työskennellään sprinteissä, joissa käydään kaikki ohjelmistokehityksen vaiheet läpi. [6.]

Ketterän kehittämisen malleissa tiimissä tuotteen kehittämisen parissa työskentelee tyypillisesti 5–9 henkilöä, jotta kommunikaatio ja yhteistyö pysyvät yksinkertaisena ja selkeänä. Lisäksi tiimiin kuuluu asiakkaan edustaja (customer representative), jonka tehtävänä on vastata sovelluskehittäjiltä esille nouseviin kysymyksiin sekä iteraatioiden välissä tehdä uudelleen arviointi sidosryhmien kanssa tavoitteista ja asiakkaan tarpeista. [5.]

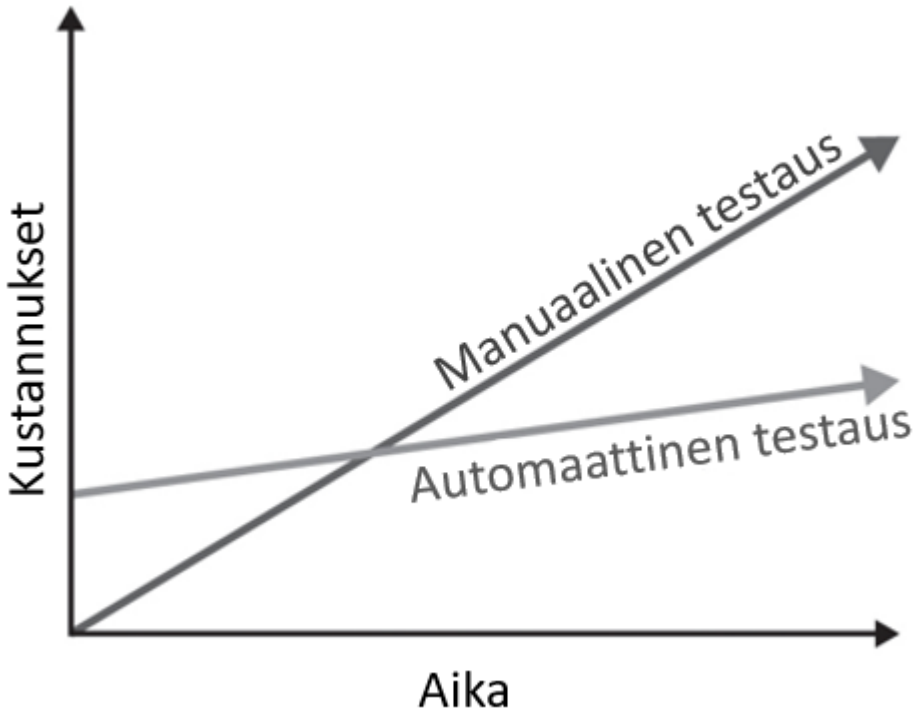
Kohdeyrityksessä työskennellään SAFe-mallin mukaisesti. SAFe-malli on yksi ketterän kehittämisen mallia hyödyntävistä menetelmistä.

4.2 Testauksen pääluokat

Ohjelmistotestaaminen voidaan jakaa kahteen pääluokkaan, manuaaliseen ja automaattiseen testaukseen. Manuaalisen testauksen suorittajana toimii ihminen, ja se soveltuu hyvin tilanteisiin, joissa vaaditaan luovuutta, harkintaa tai erilaisten päätösten tekoa. Esimerkiksi erilaiset käytettävyystestit ovat manuaalisesti tehtävää testausta, koska ihminen pystyy huomioimaan ja testaamaan ohjelmiston ulkoasua, toimintaa ja käytettävyyttä. Manuaalinen testaus soveltuu

myös hyvin pieniin projekteihin ja ohjelmistoihin, joita ei päivitetä julkaisun jälkeen. [7; 8.]

Nimensä mukaan automaattisessa testauksessa erillinen ohjelmisto hoitaa testaamisen automaattisesti. Automaattisen testauksen suurimmat hyödyt saadaan suurissa projekteissa, joiden ylläpito jatkuu julkaisun jälkeen. Tämä johtuu jatkuvasta ohjelmiston testaamisen tarpeesta, mikä manuaalisesti tehtynä vie paljon aikaa, lisää kustannuksia ja on altis inhimillisille virheille. Lisäksi lyhyemmissä projekteissa ei ole kannattavaa käyttää aikaa automaattisen testauksen suunnitteluun ja kehittämiseen. Testien automatisoinnilla pystytään vähentämään tai jopa kokonaan pääsemään eroon joistakin manuaalisen testauksen huonoista puolista. Kuten kuvasta 3 nähdään, manuaalisen testauksen kustannukset ovat huomattavasti pienemmät lyhytkestoisissa projekteissa, mutta ajan kasvaessa automaattisen testauksen hyödyt näkyvät kulujen vähenemisenä. [7; 8.]



Kuva 3. Testiautomaatio vähentää kustannuksia ajan kuluessa verrattuna manuaalisesti tehtävään testaukseen. [8.]

4.3 Testauksen eri menetelmät

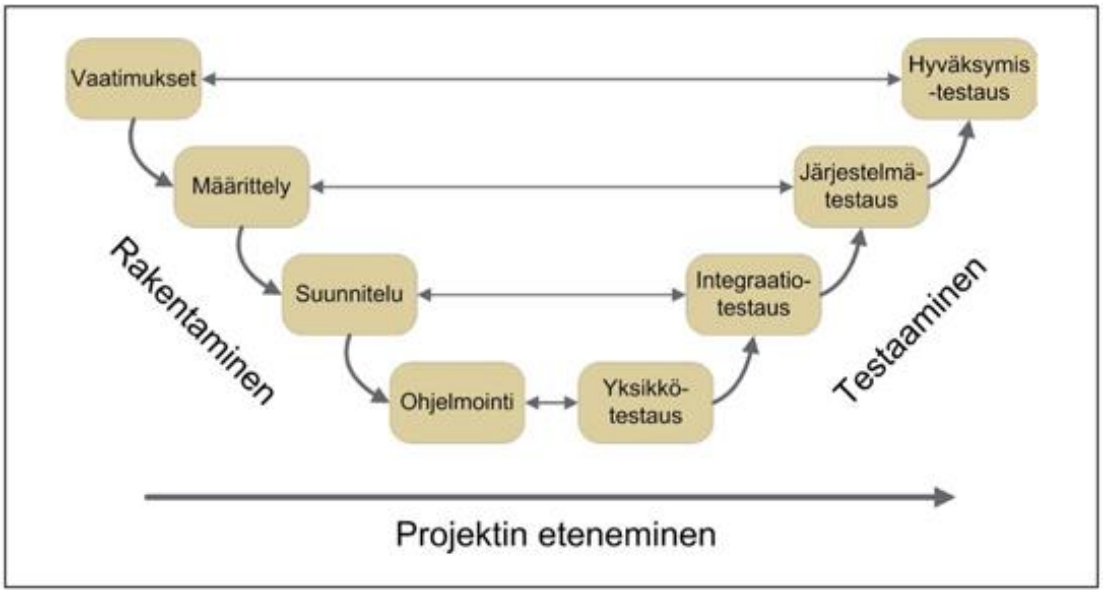
Testaaminen järjestelmän kehitysvaiheessa on todella tärkeää, sillä myöhemmässä vaiheessa havaitun virheen korjaaminen voi tulla jopa 40 kertaa kalliimmaksi kuin kehitysvaiheessa huomatun virheen korjaus. Kustannusten jyrkkä nouseminen johtuu pitkälti virheen selvittelystä, toistamisesta ja korjaamisesta. Järjestelmän perusteelliseen testaamiseen tarvitaan useita testejä, joilla testataan järjestelmän eri osa-alueita kuten turvallisuutta tai käytettävyyttä. Suoritettavat testit voidaan jakaa toiminnalliseen testaukseen (functional testing) ja ei-toiminnalliseen testaukseen (Non-functional testing). [9.]

Toiminnallinen testauksen tarkoituksena on varmistaa, että järjestelmä toimii asiakkaan tarvitsemalla tavalla. Vaaditut ominaisuudet määritetään suunnittelu-vaiheessa. Toiminnalliseen testaukseen kuuluu esimerkiksi käytettävyys-, järjestelmä- ja hyväksymistestaus. Toiminnallisen testauksen suorittajana toimii usein ihminen. [9.]

Ei-toiminnallinen testaus keskittyy selvittämään järjestelmän pinnan alla tapahtuvaa toimintaa, kuten järjestelmän suorituskykyä tai turvallisuutta. Testeille tulee tehdä tarkat määritelmät, kuten miten tuloksia mitataan, sillä testituloksien subjektiivinen tarkastelu voi aiheuttaa vääristyneen kuvan saaduista tuloksista. Ei-toiminnalisia testejä ovat esimerkiksi suorituskyky-, turvallisuus-, yhteensopivuus- ja käytettävyystestaus. Testaus suoritetaan usein automaattisesti, sillä esimerkiksi vasteaikojen mittaus on mahdoton tehdä manuaalisesti. [9.]

4.4 Testauksen eri tasot

Järjestelmän testaaminen voidaan jakaa eri testaustasoihin testattavan ominaisuuden mukaan. Kehitysvaiheessa on kolme päävaihetta: yksikkötestaus, integrointitestaus ja järjestelmätestaus. Lopuksi on vielä hyväksymistestaus, jossa testaaminen suoritetaan järjestelmän julkaistavaan versioon tai siitä tarkkaan versioon. Testaustasojen kuvaamiseen käytetään kuvassa 4 esitettyä V-malliksi kutsuttua kaaviota. [10.]



Kuva 4. V-malli kuvaa järjestelmän rakentamisen vaiheiden heijastumista testaustasoihin projektin edetessä. [10.]

Yksikkötestauksen tarkoituksena on testata uutta yksittäistä ominaisuutta, jolloin varmistetaan ominaisuuden toimivuus. Usein yksikkötestauksen suorittaa uuden ominaisuuden tekijä, joka tietää, miten uuden ominaisuuden tulisi toimia ja miten se on rakennettu. Ominaisuuden tekijän on helppo tehdä tarvittavat korjaukset ominaisuuteen, mikäli testauksesta saatavat tulokset eivät ole haluttuja. Yksikkötestauksen tueksi voidaan rakentaa testikomponentteja tai testitynkiä (mock objects, test studs), joiden avulla pystytään varmistamaan uuden ominaisuuden toiminta tilanteissa, joissa siltä vaaditaan vuorovaikutusta muiden ominaisuuksien kanssa, kuten tiedon hakua tietokannasta, kuunnella verkon yli tapahtuvia pyyntöjä tai reagoida järjestelmän toimintaan. Testikomponenttien hyödyt ovat suuret, sillä niiden avulla pystytään varmistamaan, että toteutettavat testit tapahtuvat aina samalla tavalla. Testikomponentteja pystytään hyödyntämään esimerkiksi järjestelmässä tapahtuvan virheen simuloimisessa. [10.]

Integrointitestauksessa aloitetaan sovittamaan järjestelmän eri osia yhteen. Tarkoituksena on varmistaa järjestelmän eri osien toimiminen yhtenä kokonaisuutena. Uusi ominaisuus tai moduuli liitetään ennestään testattuun osaan järjestelmää, jolloin moduulien toimintaa kokonaisuutena voidaan testata. Osa

moduuleista voi vielä tarvita testitynkiä kunnollisen testauksen takaamiseksi. Tässä vaiheessa ei kuitenkaan testata koko järjestelmän toimivuutta, vaan keskitytään pienempien kokonaisuuksien testaamiseen. Integrointitestaus voidaan toteuttaa kolmella eri tavalla: alhaalta ylöspäin, ylhäältä alaspäin ja voileipätestauksena. Alhaalta ylöspäin -mallissa järjestelmään tuodaan ensimmäisenä matalan tason moduulit, jotka ovat vuorovaikutuksessa suoraan käytettävän raudan tai järjestelmän kanssa. Ensimmäisen moduulin testauksen jälkeen järjestelmään tuodaan seuraava moduuli, joka hyödyntää edellistä moduulia. Moduuleja ikään kuin lisätään toistensa päälle niin kauan, että kaikki kokonaisuuden moduulit on lisätty ja testattu. Ylhäältä alaspäin menetelmässä testaus tapahtuu päin vastaisessa järjestyksessä kuin alhaalta ylöspäin menetelmässä. Testaus aloitetaan järjestelmähierarkian huipulta ja edetään kohti rautatasoa lähellä olevia moduuleja. Voileipätestauksessa hyödynnetään molempia edellä mainittuja menetelmiä eli moduuleja tuodaan järjestelmähierarkian molemmista päistä samaan aikaan. [10.]

Järjestelmätestaus on kehitysvaiheen viimeinen vaihe, jonka tarkoituksena on testata testiympäristössä olevaa järjestelmää kokonaisuutena ja varmistaa, että se täyttää kaikki asetetut vaatimukset. Tässä vaiheessa kaikki testityngät ja testikomponentit tulee olla siivottuna järjestelmästä. Järjestelmätestaamista voidaan toteuttaa monella eri tavalla, sillä eri testausmenetelmien käyttämisellä saadaan erilaisia tuloksia. Esimerkiksi käyttäjätestaus ja kuormitustestaus tuottavat hyvin erilaista tietoa järjestelmästä ja sen toimimisesta. [10.]

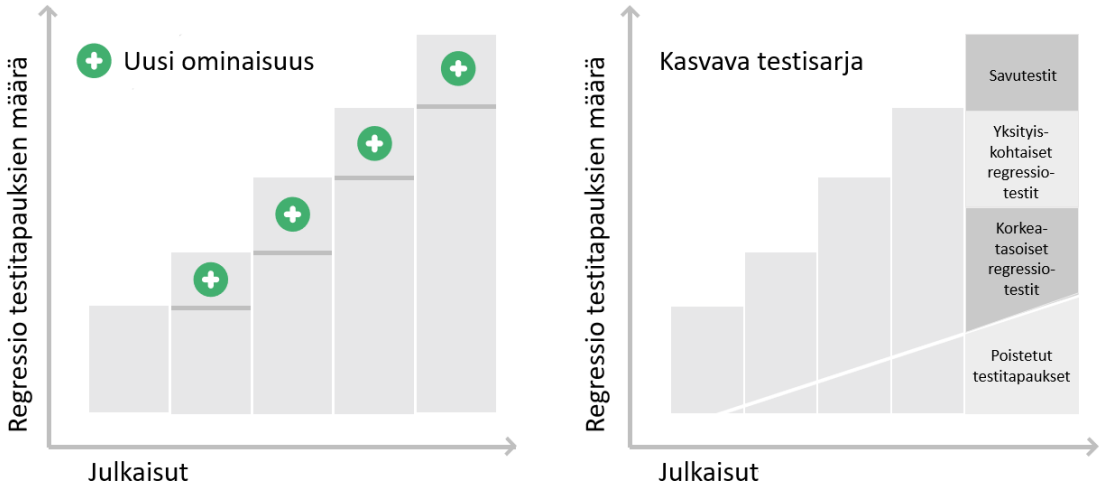
Viimeisenä vaiheena on hyväksymistestaus, jonka tarkoituksena on toimia virallisena tarkastuksena. Järjestelmä hyväksytään asiakkaan toimesta ja järjestelmä luovutetaan lakiteknisesti asiakkaan omistukseen. Hyväksymistestauksessa järjestelmää käytetään sen kohde ympäristössä toisin kuin järjestelmätestauksessa. [10.]

4.5 Regressiotestaus

Ohjelmistojen jatkuvan kehittymisen takia on tärkeää huolehtia, ettei järjestelmään tehdyillä uusilla ominaisuuksilla ole vaikutusta järjestelmässä jo ennestään oleviin vanhoihin ominaisuuksiin. Järjestelmän kehittyessä myös sen monimutkaisuus kasvaa, mikä voi johtaa tahattomiin muutoksiin. Vanhojen ominaisuuksien testausta kutsutaan regressiotestaamiseksi. Regressiotestissä toistetaan aina samat vaiheet, jolloin saadun lopputuloksen pitäisi olla sama. Mikäli lopputulos ei ole sama kuin ensimmäisellä kerralla tehtynä, voidaan päätellä, että testattava ominaisuus on muuttunut. Regressiotestaus voidaan tehdä manuaalisesti tai automaattisesti. [11, s. 154–156; 12, s. 356–357.]

Manuaalisesti tehtynä regressiotestaaminen vaatii paljon aikaa. Etenkin suuremmissa projekteissa, joissa testitapauksia on paljon. Resurssien kuluttamisen lisäksi ihmisen tekemänä testauksessa on aina inhimillisen virheen mahdollisuus, etenkin kun samaa asiaa toistetaan useaan otteeseen. Tämän takia regressiotestien automatisoinnilla saatu hyöty isoissa projekteissa on suuri. Ohjelmiston kehittyessä myös sen testaamiseen vaadittavien regressiotestitapausten määrä kasvaa, mikä vie entistä enemmän resursseja. Onkin tärkeää säännöllisesti päivittää vanhoja testitapauksia vastaamaan järjestelmässä tapahtuneita muutoksia ja poistaa vanhentuneita testitapauksia käytöstä. Kuvassa 5 olevasta vasemmanpuoleisesta kaaviosta nähdään, kuinka uusien julkaisujen myötä testitapausten määrä kasvaa. Ikuinen testitapauksien lisääntyminen ei ole kestävää, sillä jossain vaiheessa kaikkien testien testaaminen vaatii liikaa työtunteja.

Oikeanpuoleisessa kaaviossa poistetaan vanhentuneita ja päivitetään vanhoja testejä korvaamaan uusia testitapauksia, jolloin testitapausten määrä pysyy maltillisena. [13.]



Kuva 5. Uusien ominaisuuksien lisääminen julkaisujen myötä lisää testitapausten määrää. Testitapauksia pitää säännöllisesti päivittää tai poistaa käytöstä, jotta testitapausten määrä pysyy sopivana. [13.]

Aina ei kannata testata kaikkia regressiotestejä uudelleen, sillä se on työlästä ja vie aikaa. Regressiotestaamiseen voidaan käyttää neljää eri tekniikkaa, joilla pystytään vähentämään testaamiseen käytettäviä resursseja ja testattavien testitapausten määrää tilanteen mukaan. Näitä tekniikoita ovat: testaa kaikki uudelleen (Retest all), regressiotestin valinta (Regression test selection), testitapausten priorisointi (Test case prioritization) ja hybridi (Hybrid). [12, s. 357; 14.]

Nimensä mukaan testaa kaikki uudelleen tekniikassa testataan kaikki järjestelmän regressiotestit uudelleen. Tämä tapa sopii pienempiin projekteihin, joiden testaamiseen ei käytetä suuria määriä regressiotestejä. Suurempien ohjelmistojen kohdalla tapaa käytetään isojen päivitysten aikana, jolloin varmistetaan, ettei julkaistuun versioon pääse mitään mitä sinne ei haluta. Manuaalisesti tehtynä tämä tekniikka on työläs ja aikaa vievä, minkä takia automaatiotestauksen käyttämisellä saadaan suuri hyöty. [14.]

Regressiotestin valinnassa valitaan testattavaksi ne osat järjestelmästä, joihin tehdyt muutokset ovat vaikuttaneet. Relevanttien testitapausten testaaminen vie vähemmän resursseja kuin kaikkien testitapausten testaaminen, mutta lisää riskiä häiriöiden pääsemisestä julkaistuun ohjelmistoon. [14.]

Testaaminen voidaan aloittaa kriittisimmästä testitapauksesta ja edetä kohti vähemmän kriittisiä tapauksia. Menetelmässä valitaan joukko testattavia testitapauksia, eikä suinkaan testata kaikkia testejä. Tätä tapaa kutsutaan testitapausten priorisoinniksi. Menetelmä voidaan jakaa kahteen tapaan, yleinen testitapauksen priorisointi (general test case prioritization) ja versiokohtainen testitapauksen priorisointi (version-specific prioritization). [14.]

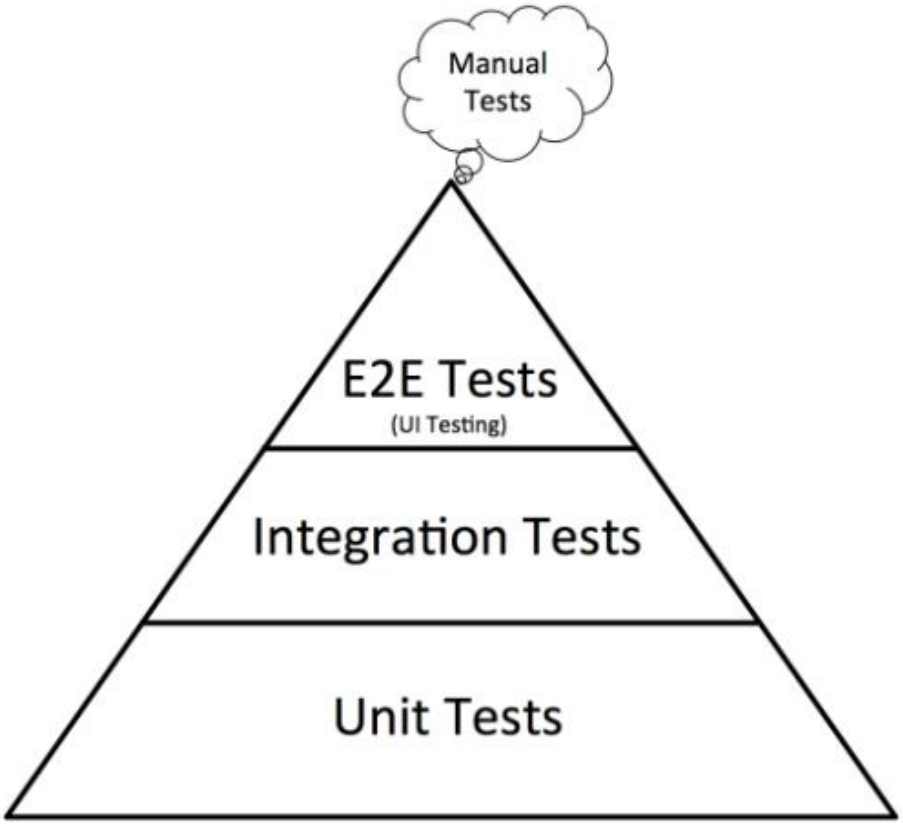
Hybridi menetelmä on yhdistelmä testitapauksen priorisointia ja regressiotestin valintaa, jossa ensin testataan kriittisimmät testitapaukset, minkä jälkeen testataan valittu osa järjestelmästä. [14.]

4.6 Testiautomaatio

Testiautomaatiolla tarkoitetaan toimintaa, missä erillinen automaatio-ohjelmisto hoitaa testitapauksen suorittamisen ilman ihmisen puuttumista testausvaiheeseen. Vaikka automaatio-ohjelmisto suorittaa testauksen, on ihmisellä silti paljon vastuuta toimivan testiautomaatiokokonaisuuden luomisessa. Testitapauksien luominen, niiden ylläpito ja testauksissa tapahtuneiden virheiden analysointi jää ihmisen tehtäväksi. Testiautomaatio onkin vain testien ajamiseen käytettävä työkalu, jonka hyödyntämisestä päätetään yrityksen määrittelemässä testausstrategiassa. Testiautomaatio ei aina ole kaikille tai kaikkiin tilanteisiin sopiva ratkaisu. Onkin tärkeää miettiä testiautomaation hyötyjä ja haittoja ennen kuin sitä lähdetään ottamaan käyttöön. Testiautomaatio soveltuu parhaiten laajoille ja pitkäaikaisille projekteille, jossa testausta suoritetaan jatkuvasti. [15; 16; 17.]

Järjestelmän testaamiseen käytettävät automaattiset testit voidaan jakaa kolmeen eri osa-alueeseen kuvassa 6 esitetyn testiautomaatiopyramidin

mukaisesti. Alimmalla tasolla on yksikkötestaus (unit testing), keskimmäiseltä tasolta löytyy integraatiotestaus (integration testing) ja ylimmällä tasolla on käyttöliittymätestaus (E2E test, UI testing). Puhekuplan sisällä oleva manuaalinen testaus ei kuulu pyramidiin, mutta on esitetty kuvassa muistutuksena lisätestaamisen tarpeesta ja tärkeydestä. Pyramidin alimmilla tasoilla pitäisi olla enemmän ajettavia testitapauksia, koska niiden testaaminen tapahtuu nopeasti. Ylemmillä tasoilla testien monimutkaisuus kasvaa, mikä hidastaa niiden ajamista. Testitapauksien monimutkaisuuden kasvaessa myös niiden haavoittuvuus kasvaa, koska testeissä on enemmän riippuvuuksia eri komponenttien välillä. Riippuvuuksien takia ylemmillä tasoilla olevien testien ylläpitoon kuluu enemmän aikaa kuin alemmilla tasoilla. [18; 19.]



Kuva 6. Testiautomaatiopyramidi kuvaa tarvittavien testitapauksien määrää tasojen koko erolla. Ylemmillä tasoilla testitapauksien monimutkaisuus ja ylläpidon tarve kasvaa. [19.]

4.6.1 Testiautomaation hyödyt

Manuaalisesti tehtävä testaus on ja tulee olemaan välttämätöntä ohjelmistokehityksessä, mutta sen rinnalle huolellisesti rakennetusta testiautomaatiosta saadaan valtavasti hyötyä. Laajoissa ja pitkäikäisissä projekteissa saadut hyödyt vain kasvavat, koska testitapauksien määrää pystytään lisäämään helposti valmiiden moduulien avulla. [20.]

Automaation suurimmat hyödyt piilevät sen nopeudessa, tarkkuudessa ja toistamisessa. Sama pätee myös testiautomaatioon. Huolellisesti suunnitellun testitapauksen ajaminen voi viedä muutaman minuutin, kun taas saman testin manuaalinen testaus voi viedä kymmeniä minuutteja. Säästetyn ajan määrä kasvaa merkittäväksi, kun suoritettavia testitapauksia on useita satoja. Automaatiolla pystytään poistamaan myös testauksen aikana sattuvat inhimilliset virheet, sillä tietokone ei kärsi väsymyksestä tai keskittymisen puutteesta. [20.]

Nopeasti ajateltuna testiautomaatiosta saatavat hyödyt liittyvät vain aikaan, tarkkuuteen ja kustannuksien pienenemiseen, mutta todellisuudessa automaatiosta saadaan muitakin hyötyjä. Testauskattavuuden paraneminen testitapauksien lisääntymisen myötä ja testien tiheämmästä suorittamisesta johtuva testituloksien laadun paraneminen ovat myös merkittäviä testiautomaatiosta saatavia hyötyjä. Testiautomaatiosta saatavat hyödyt vähentävät ohjelmiston tuotantoversioon pääsevien virheiden mahdollisuutta ja samalla vähentävät sinne pääsevien virheiden vakavuutta. [20.]

4.6.2 Testiautomaation haasteet

Testiautomaatiosta saatavat hyödyt kasvavat projektin kasvamisen myötä, mutta hyötyjen mukana tulee myös haasteita. Etenkin testitapauksien kehittämisessä ja testiautomaation käyttöönotossa piilee useita suden kuoppia, joihin kannattaa varautua ennakkoon suunnittelemalla huolella, mitä halutaan testata ja miten testit suoritetaan. Vanhan sananlasku ”Hyvin suunniteltu on puoliksi tehty” pitää paikkansa myös testiautomaation hyödyntämisessä. [20.]

Haasteet alkavat jo suunnitteluvaiheessa, sillä ensimmäisenä täytyy päättää, mitä testiautomaatio-ohjelmistoa käytetään. Ohjelmistoja on lukuisia erilaisia, ilmaisesta maksulliseen, helposta monimutkaiseen ja kaikkea siltä väliltä. Ohjelmistoihin perehtyminen voi viedä runsaasti aikaa. Täydellisen ohjelmiston löytämisen jälkeen, tulee mietittäväksi kuka ohjelmistoa osaa käyttää. Kaikki testiautomaatio-ohjelmat eivät ole helppokäyttöisiä. Opettelu voi viedä paljon aikaa työntekijöiltä etenkin, jos heiltä ei löydy aikaisempaa kokemusta testiautomaatiosta. Vaihtoehtoisesti yritys voi palkata valmiiksi osaavia työntekijöitä, mutta molemmissa tapauksissa testiautomaation käyttöönoton kustannukset kasvavat. Testiautomaation kustannukset ovat alkuvaiheessa huomattavasti suuremmat verrattuna manuaaliseen testaukseen. [20.]

Huonosti tehty testaussuunnitelma heijastuu testitapauksien kehitysvaiheeseen. Testiautomaation käyttöönotossa suurin osa ajasta kuluu testitapauksien luontiin. Etenkin monimutkaisien testitapauksien luominen vie runsaasti aikaa, minkä takia tulisi huolellisesti suunnitella, mitä halutaan testata ja miten. Jokainen testi ei sovellu automatisoitavaksi. Testitapauksien tulisi olla mahdollisimman luotettavia ja helposti ylläpidettäviä, jotta ylläpidolliset kustannukset pysyvät matalina. Täsmälliset ja määrätietoiset askeleet testitapauksessa ovat tärkeitä, sillä ylimääräiset tai puuttuvat vaiheet testitapauksien sisällä kasvattavat testin hajoamisen riskiä. [20.]

5 Automaatiotestien tekeminen

Regressiotestien automatisointi toteutetaan Ranorex Studio -nimisellä ohjelmistolla, jolla pystytään hyödyntämään aikaisemmin tehtyjen testitapauksien osia eli moduuleja. Valmiit testitapaukset ajetaan automaattisesti jokaisena yönä, jolloin saadaan kattavaa tietoa testattavan toiminnon toimimisesta. Testitapauksien ajoista saadut tulokset tulevat näkyviin Jenkins-nimiseen sovellukseen, jossa pystytään tutkimaan tarkemmin, mitä testitapauksien ajon aikana on tapahtunut.

Automaatiotestien tekeminen alkaa suunnittelemalla, mitä halutaan testata. Testattavan toiminnon tulisi olla järjestelmän käytön kannalta tärkeä, koska tärkeisiin toimintoihin ei tehdä muutoksia usein, joten toiminnon testaamiseen luodun testitapauksen ylläpitäminen ei vie liikaa resursseja. Lisäksi järjestelmän käytön kannalta tärkeän toiminnon testaaminen päivittäin on hyödyllisempää kuin vähäpätöisempien toimintojen testaus.

Seuraavana vaiheena on testitapauksien luominen ja niiden kehittäminen. Testitapauksia luodessa on suositeltavaa nimetä tapaukset kuvaamaan testin tarkoitusta, etenkin silloin, kun testitapauksia on määrällisesti paljon, jotta testitapaukset löytyvät nopeasti tarvittaessa. Testitapauksien kehittäminen vie suurimman osan ajasta, koska jokainen automaattisesti suoritettava askel täytyy lisätä testitapauksiin erikseen. Aikaa kuluu myös runsaasti askelien toimimisen testaamiseen. Ranorex tuo helpotusta askelien lisäämiseen moduulien muodossa.

Moduulit ovat yksittäisten vaiheiden suorituksia, jotka yhdessä muodostavat kokonaisen testitapauksen. Esimerkiksi sisäänkirjautuminen ja uloskirjautuminen järjestelmään ovat yksittäisiä moduuleja. Niitä tarvitaan melkein jokaisessa testitapauksessa, joten on paljon tehokkaampaa tehdä molemmista yksi valmis moduuli, joita hyödynnetään jokaisessa testitapauksessa kuin ohjelmoida jokaiseen testitapaukseen sisäänkirjautuminen ja uloskirjautuminen erikseen. Testitapauksien määrän kasvaessa myös moduulien määrä kasvaa, jolloin kokonaan uuden testitapauksen tekemisessä pystytään hyödyntämään ennestään valmiita moduuleja ja siten säästämään runsaasti resursseja. Moduulien hyödyt tulevat myös esille testitapauksien ylläpidossa, sillä yksittäisen moduulin korjaus on nopeampaa kuin testitapauksien korjaaminen yksitellen.

5.1 Ranorex Studio ja Jenkins

Ranorex Studio on vuonna 2007 perustetun itävaltalaisen Ranorex-yhtiön kehittämä ohjelmisto, jonka tarkoituksena oli mullistaa testaajien ja kehittäjien välinen yhteistyö ja samalla parantaa ohjelmistojen laatua. Ranorex ostettiin vuonna 2017 yhdysvaltalaisen Ideran toimesta. [21.]

Automaatiotestitapauksien tekeminen vaatii usein käyttäjältä entuudestaan ohjelmointiosaamista, mikä voi hidastaa testiautomaation kehitystä ja käyttöönottoa. Ranorex helpottaa vähemmän ohjelmoimista osaavien taakkaa useilla eri ominaisuuksilla, kuten mahdollisuudella tallentaa käyttäjän tekemiä askelia suoraan testitapaukseen, jolloin käyttäjän on helppo lisätä tai poistaa tarpeettomia kohtia. Erilaisten elementtien tunnistaminen on myös helppoa elementtien jäljittämiseen tarkoitetulla työkalulla, jonka avulla pystytään paikantamaan elementtien sijainti. [22.]

Vaikka testitapauksien luominen on tehty helpoksi ilman ohjelmointia, on käyttäjän mahdollista halutessaan hyödyntää ohjelmointiosaamistaan. Ranorexin koodieditori tunnistaa yleisimmät ohjelmointikielet. [22.]

Testiautomaatiotapausten ajamisesta saatujen tuloksien tarkasteluun käytetään Jenkins-nimistä avoimen lähdekoodin automaatiotyökalua. Web-pohjaista Jenkinsiä pystytään käyttämään monenlaisiin eri tarkoituksiin, kuten versionhallintaan, kokoonpanonhallintaan, jatkuvaan testaukseen, käyttöönottoon ja monitorointiin. Riittää, että lataa ja asentaa tarvitsemaansa toimintaan tarkoitetun laajennuksen. Jenkinsiä käytetään laajasti ympäri maailman jatkuvan integraation työkaluna ja sillä on yli miljoona aktiivista käyttäjää. [23.]

5.2 Automatisoitavien regressiotestien valinta

Opinnäytetyön tekijä työskentelee kohdeyrityksessä lähetteiden parissa, minkä vuoksi automatisoitavat regressiotestit kohdistuvat järjestelmän sisäisiin lähetteisiin ja niiden työnkulkuihin. Lähete on asiakirja, jolla potilas lähetetään jatkotutkimuksiin tai hoidettavaksi toiseen yksikköön, esimerkiksi tilanteissa, joissa potilas on hakeutunut terveyskeskukseen hoitoon, ja käynnin aikana todetaan tarve magneettikuvaukselle. Mikäli terveyskeskuksessa ei ole magneettikuvauslaitetta, potilaalle kirjoitetaan lähete toiseen yksikköön toimenpiteen suorittamiseksi.

Lähetteisiin kohdistuvien regressiotestien tarkoituksena on varmistaa, että lähetteet kulkevat järjestelmässä suunnitellusti, näkyvät oikeassa paikassa ja niitä koskevat ilmoitukset toimivat, kuten on tarkoitettu. Lähetteiden regressiotestit soveltuvat automatisoinnin kohteeksi hyvin, sillä manuaalisesti tehtynä ne vievät aikaa, niissä on toistoa eikä lähetteiden työnkulkuihin tehdä suuria muutoksia. Lisäksi regressiotesteissä tehdään paljon samoja asioita, joten ensimmäisen testin automatisoinnin jälkeen pystytään hyödyntämään sen valmiita moduuleja muiden testien automatisoinnissa.

Automatisoitavia regressiotestejä valittaessa haluttiin saada mahdollisimman suuri hyöty automatisoinnista. Alussa tarkasteltiin lähetteisiin kohdistuvia regressiotestejä, vertailtiin niiden suorittamiseen tarvittavien askelien määrää ja järjestettiin samoja vaiheita sisältävät testit keskenään. Automatisoitavaksi valittiin testejä, joissa perusterveydenhuollon yksiköstä tehdään lähete erikoissairaanhoidon yksikköön. Valittutuissa testeissä oli paljon yhtäläisyyksiä toistensa kanssa, mikä nopeuttaa ja helpottaa testien automatisointia.

Valittujen testien testauksessa käytettävä työnkulkua esitetään kuvassa 7. Testien alussa työnkulku on sama. Lähetteen määräys tapahtuu kunnan yksikössä, jonka jälkeen lähete ohjautuu oikeaan HUS-yksikköön. Lisätietopyynnön tekeminen on vaihtoehtoista, mutta valmiiden testitapauksien aikana on tarkoitus testata myös lisätietopyynnön tekeminen ja siihen vastaaminen. Vastauksen saamisen jälkeen lähete joko palautetaan tai hyväksytään. Lähetteen hyväksymisen yhteydessä testataan myös hoitopalautteen toiminta.

Kuva 7. Testitapauksissa lähetteen käsittelyssä käytettävä työnkulku on alkuun sama, mutta testien loppu riippuu siitä, mitä lähetteelle tehdään.

5.3 Regressiotestien automatisointi

Opinnäytetyön tekijällä ei ollut ennen automatisointiprojektin alkua aikaisempaa kokemusta testiautomaatiosta, joten projektin ensimmäinen vaihe oli kattavan perehdytyksen saaminen kohdeyrityksen testiautomaatiokokonaisuudesta ja

siihen käytetyistä ohjelmistoista. Perehdytyksen sekä automatisoitavien testien valitsemisen ja suunnittelun jälkeen oli vuorossa itse automatisoinnin aloittaminen. Testien automatisoinnin toteuttamisessa käytettiin Ranorex-ohjelmistoa.

Automatisoinnin alussa luotiin testitapaukset Ranorex-ohjelmistoon. Testitapauksien sisälle lisättiin valmiita moduuleja ja rakennettiin uusia, joiden sisälle tallennettiin automaattisesti ajettavat askeleet. Moduulien ideana on helpottaa ylläpitotyötä, selkeyttää testitapauksien rakennetta ja nopeuttaa uusien testitapauksien luomista. Testitapauksien rakentamisessa pystyttiin hyödyntämään joitakin valmiita moduuleja, mutta kaikkia tarvittavia moduuleja ei ollut valmiina. Tarvittavat moduulit täytyi rakentaa itse.

Moduulien sisälle tulevien askelien tekemiseen käytettiin enimmäkseen nauhoitustyökalua. Nauhoitustyökalu nimensä mukaan nauhoittaa käyttäjän tekemiä klikkauksia tai näppäinten painalluksia ja tuo käyttäjän tekemät askeleet moduuliin automaattisesti. Vaikka nauhoittamalla saadaan askeleet nopeasti moduuliin, on ongelmana eri elementtien löytyminen ajon aikana. Jokaisella elementillä on oma polkunsa, jonka avulla elementti tunnistetaan. Poluissa olevat virheet estävät elementtien löytymisen, minkä takia testitapauksen vaiheita nauhoittaessa kuluu aikaa elementtien polkujen tarkistamiseen. Kaikkia polkuja ei tarvitse tarkistaa, sillä osa nauhoitustyökalun etsimistä poluista ovat toimivia.

Moduulien rakentamisessa täytyi olla koko ajan tarkkana, koska askelten lisääminen ja niiden toimivuuden tarkistaminen ei vielä tarkoittanut moduulin toimivuutta. Testiajon aikana kävi usein ilmi, että tietokone aloitti jonkin vaiheen ajamisen liian nopeasti. Usein syynä oli järjestelmässä tapahtuvan latauksen tai sivun vaihdon takia johtuva pieni viive, jota ei ollut huomioitu moduulissa erikseen. Moduuleissa olevat askeleet ajetaan yksitellen ensimmäisestä viimeisimpään. Tietokone hyppää suoraan askeleen valmistumisen jälkeen seuraavaan askeleeseen, minkä takia järjestelmässä tapahtuvat viiveet aiheuttavat virheen testin ajossa. Oikean elämän esimerkkinä voidaan käyttää hissin tilaamista. Hissin tilaamisen jälkeen täytyy vielä odottaa hissin saapumista ja ovien avautumista ennen kuin sen kyytiin pääsee. Testiajoa tehneeltä tietokoneelta puuttui

odottamisaskel, minkä takia se yritti hissin kyytiin heti hissin tilaamisen jälkeen. Ongelma oli helppo korjata lisäämällä odotusaika askelien väliin.

Askelien lisäämisen yhteydessä täytyi jatkuvasti tehdä testiajoja testitapauksen toimivuuden varmistamiseksi, koska ennen testiajoa oli mahdoton tietää, toimivatko lisätyt askeleet vai eivät. Askelia lisättiinkin moduuleihin vähän kerrallaan, jonka jälkeen suoritettiin lisättyjen askelien testaus ja mahdolliset korjaukset. Koko testitapausta ei aina tarvinnut ajaa uudelleen, sillä Ranorex tarjoaa vaihtoehtona tiettyjen askelien ajamista. Koko testitapauksen ajaminen uudelleen ja uudelleen veisi valtavasti turhaa aikaa.

Askelien lisäämisen ja testaamisen jälkeen oli vielä tehtävä kommentointi lisättyihin askeliin. Kommenteilla kerrotaan, mitä kommentoitu askel tekee. Kommentoinnilla pyritään helpottamaan testitapauksen selkeyttä ja auttamaan sen ymmärtämisessä. Testitapauksen tekijälle testitapauksen toiminta on selkeää, mutta henkilölle, joka ei ole aikaisemmin ollut tekemisissä kyseisen testitapauksen kanssa voi olla hankala hahmottaa, mitä testitapauksessa tapahtuu. Selkeät kommentit auttavat testitapauksien ylläpitämisessä.

Opinnäytetyön kirjoitushetkellä testitapauksien automatisointi on vielä kesken. Automatisoitaviksi valitut regressiotestit olivat monivaiheisia eikä kaikista vaiheista ollut valmiita moduuleja hyödynnettäviksi, mikä on lisännyt työn määrää huomattavasti. Osasyynä viiveelle kehitystyössä on myös ollut tekijän automatisointiosaamisen puute projektin alkuvaiheessa. Osaamisen puute hidasti työskentelyä etenkin projektin alussa, vaikka automatisointiin saatiinkin kattava perehdytys. Lisäksi automatisointityön aloitus venyi, koska sen edelle priorisoitiin kohdeyritykselle kriittisempiä työtehtäviä.

6 Automaatiotestauksen hyödyt

Kohdeyrityksessä hyödynnetään testiautomaatiota laajasti järjestelmän eri osa-alueiden testaamisessa, mutta manuaalisesti tehtävien testien osuus on silti suurempi kuin automaattisesti ajettavien testien. Testiautomaation

hyödyntämisen lisäämisellä pystytään saavuttamaan paljon erilaisia hyötyjä, jotka mahdollistavat entistä tehokkaamman järjestelmän kehityksen ja ylläpidon.

Jatkuvan kehittämisen takia järjestelmä on jatkuvasti muutoksen alla. Järjestelmän monimutkaisuuden ja laajuuden takia kaikkia muutoksien vaikutuksia ei aina huomata niitä testatessa. Virheiden löytäminen ennen muutoksien vientiä tuotantoympäristöön lisää järjestelmän testaamisen tarvetta. Manuaalisesti tehtävä järjestelmän testaaminen kuluttaa runsaasti kohdeyrityksen resursseja, joita voitaisiin käyttää muiden tehtävien edistämiseen. Testiautomaatiolla pystytään kasvattamaan järjestelmän testaamista ja samalla vähentämään kohdeyrityksen resurssien käyttöä testaamisessa.

Automaattisesti ajettavien testien ajaminen tapahtuu jokaisena yönä, mikä kasvattaa huomattavasti testeistä saatujen tuloksien määrää ja laatua. Testien tuloksien vilkaiseminen Jenkins-käyttöliittymästä vie vain muutaman minuutin. Jenkins esittää testin viimeisimpien ajojen tuloksia erilaisilla kuvilla, joten testin viimeisimmistä tuloksista saa selkeän kuvan ilman tarkempaa tutkimista. Automaattisen testauksen prosessi on huomattavasti nopeampaa kuin saman prosessin tekeminen manuaalisesti.

Ihmisen tekemänä testejä ei pystytä suorittamaan yhtä useasti kuin automaattisesti. Inhimilliset virheet testauksen aikana voivat vaikuttaa testaustuloksen vääristymiseen etenkin silloin, kun kyseistä testiä suoritetaan harvakseltaan. Automaation avulla saadaan jatkuvasti testituloksia, mikä parantaa testaustuloksien laatua ja vähentää virheellisten tuloksien vaikutusta.

Jenkins-käyttöliittymän avulla pystytään tarkastelemaan testitapauksien ajamisessa tapahtuneita virheitä. Testitapauksen ajon aikana tapahtuneet vaiheet tallentuvat raportille, josta niitä pystytään lukemaan. Virheraportti nopeuttaa virheen syyn selvittämistä, koska raportilta näkee, missä testin vaiheessa virhe on sattunut. Virheen etsimiseen ei kulu ylimääräistä aikaa, ja korjaaminen voidaan aloittaa askeleesta, jossa virhe on tapahtunut.

Testiautomaation hyödyt kohdeyritykselle ovat huomattavat. Laajan ja monimutkaisen potilastietojärjestelmän automaattinen testaaminen vähentää, tukee ja nopeuttaa ihmisen tekemää työtä järjestelmän testaamisessa. Testauskokonaisuuden tehokkaampi toteuttaminen vähentää tuotantoympäristöön pääsevien virheiden määrää sekä samanaikaisesti nopeuttaa virheiden havaitsemista. Myös virheiden selvittäminen ja korjaus nopeutuvat. Edellä mainitut hyödyt mahdollistavat kohdeyrityksen resurssien tehokkaamman käytön. Vapautuneet resurssit voidaan hyödyntää muissa työtehtävissä, mikä omalta osaltaan helpottaa ja edistää potilastietojärjestelmän jatkokehitystä sekä ylläpitoa.

7 Pohdinta ja yhteenveto

Suurien ja pitkäikäisten sovelluksien ja ohjelmistoprojektien kehittämisessä automaation hyödyntäminen testauksessa on suositeltavaa sen tarjoamien hyötyjen takia. Testiautomaation käyttöönotto vaatii paljon aikaa ja vaivaa, mutta hyvin tehdyllä testausstrategialla pystytään selkeyttämään testiautomaatioon liittyviä käytänteitä ja helpottamaan testiautomaation käyttöönottoa. On myös muistettava, että testiautomaatio vaatii osaavia työntekijöitä kehittämään ja ylläpitämään testiautomaatiokokonaisuutta. Testiautomaation hyödyt tulevat esiin pitkällä aikavälillä, minkä takia kustannukset ovat alussa korkeat, mutta laskevat huomattavasti testitapauksien määrän kasvaessa. Suurissa projekteissa voi olla jopa useita satoja testejä, minkä takia automatisoinnista saatu hyöty kasvaa pitkässä juoksussa huomattavaksi. Vaikka testiautomaation hyödyt ovat laajat, ei silti kaikkia testejä tulisi automatisoida. Osa testeistä vaatii ihmisen havainnointi- tai päätöksentekokykyä. Manuaalisesti ja automaattisesti tehtävää testausta tulisi hyödyntää yhdessä molempien tapojen parhaiden puolien hyödyntämiseksi ja laajan testauskattavuuden sisältävän kokonaisuuden luomiseksi.

Opinnäytetyön aikana perehdyttiin testiautomaation maailmaan syvällisesti tutkimalla erilaisia aineistoja sekä käytännönläheisesti kehittämis- ja ylläpitotyön kautta. Vaikka kehittämistyö jäikin vielä keskeneräiseksi, saatiin siitä selkeä kuva, minkä takia testitapauksien kehittäminen vie paljon resursseja. Myös valmiiden automaatiotestitapauksien ylläpitotyö auttoi ymmärtämään

testiautomaation hyviä ja huonoja puolia. Käytännön tekemisen yhteydessä opitut ja koetut asiat olivat linjassa tutkittujen aineistojen johtopäätöksien kanssa, mikä ei sinänsä ollut yllättävää. Testiautomaatiota on kuitenkin hyödynnetty jo pidemmän aikaa, minkä takia uusien ja mullistavien löytöjen etsimisessä täytyisi porautua pintaa syvemmälle.

Konstruktiivista tutkimusmenetelmää hyödyntämällä pyrittiin selvittämään testiautomaation hyödyntämistä ja siitä saatavia hyötyjä ohjelmistokehityksen kannalta. Lähtökohtana oli resurssien kuluminen testaamiseen, jonka helpottamiseen suunniteltiin testiautomaation hyödyntämistä. Opinnäytetyön aikana selvisi, että testiautomaatiolla pystytään vähentämään resurssien kulumista ja samalla parantamaan testauskattavuutta. Vaikka teoreettinen kontribuutio jäi opinnäytetyössä vähäiseksi, pystytään silti toteamaan, että tehty tutkimus- ja kehitystyö vahvistaa aikaisempien tutkimuksien tuloksia.

Opinnäytetyön tekemisessä olisi pitänyt varata enemmän aikaa kehittämiselle ja kirjoittamiselle, mikä olisi mahdollisesti johtanut laajempaan ja kattavampaan opinnäytetyöhön. Huomioitavaa on myös, ettei opinnäytetyön tekijällä ollut aikaisempaa kokemusta aihealueeseen liittyen, minkä takia tutkimus- ja kehittämistyö vaativat suunniteltua enemmän aikaa. Osaamisen puute testiautomaatiossa esti sukeltamisen aihealueen syvään päähän, mikä olisi voinut paljastaa uusia puolia testiautomaatiosta. Opinnäytetyöhön suunniteltu aikataulu ei ollut joustava työn venymiselle, mikä johti kiireeseen opinnäytetyön valmistumisessa.

Opinnäytetyön aikana esille nousseiden johtopäätöksien perusteella pystytään toteamaan, ettei testiautomaation tekemisessä kannata kiirehtiä. Perusteellinen suunnittelu jo projektin alkuvaiheessa antaa hyvän pohjan testiautomaation hyödyntämiselle. Onnistuneen testiautomaatiokokonaisuuden luominen vaatii suunnittelua, aikaa, rahaa ja ennen kaikkea osaavia työntekijöitä. Hyvin toteutettu testiautomaatiokokonaisuus antaa yritykselle paljon hyötyjä, mutta on kuitenkin muistettava, ettei testiautomaatio sovi käytettäväksi jokaisessa projektissa eikä jokainen testi sovellu automatisoitavaksi.

Lähteet

1. Oy Apotti Ab. Verkkoaineisto. Oy Apotti Ab. <https://www.apotti.fi/apotti/apotti-yrityksena/>. Luettu 14.8.2022.

2. Usein kysyttyä. Verkkoaineisto. Oy Apotti Ab. <https://www.apotti.fi/apotti/usein-kysyttya/>. Luettu 14.8.2022.

3. Lukka, Kari. 2001. Kari Lukka: Konstruktiivinen tutkimusote. Verkkoaineisto. <https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/>. Luettu 6.10.2022.

4. 6 Phases of the software engineering life cycle. 2020. Verkkoaineisto. Erbis. <https://erbis.com/blog/6-phases-of-the-software-development-life-cycle/>. Päivitetty 19.5.2020. Luettu 13.10.2022.

5. Sewell, Landon. 2014. Agile software development. E-kirja. World Technologies.

6. Mitä ketterä ohjelmistokehitys tarkoittaa Hurjalla. 2021. Verkkoaineisto. Hurja. <https://www.hurja.fi/blogi/mita-kettera-ohjelmistokehitys-tarkoittaa-hurjalla/>. Päivitetty 16.9.2021. Luettu 8.10.2022.

7. Jose, Boby. 2021. Test Automation: A manager's guide. E-kirja. BCS, The Chartered Institute for IT.

8. Axelrod, Arnon. 2018. Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects. E-kirja. Apress.

9. Tonislava, Docheva. 2021. The complete guide to test automation: best strategies, frameworks and tools. Verkkoaineisto. <https://www.getxray.app/blog/the-complete-guide-to-test-automation-best-strategies-frameworks-and-

tools?utm_term=&utm_campaign=GG_SEARCH_TOFU+-+DSA+Blog+Posts+-+EU1&utm_source=adwords&utm_medium=ppc&hsa_acc=9970092548&hsa_cam=17994134125&hsa_grp=141374238713&hsa_ad=615661432990&hsa_src=g&hsa_tgt=dsa-1391637226698&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gclid=CjwKCAjwv4SaBhBPEiwA9YzZvL-HUBT16a_6udB6vg0XrU5bS3qKOJ5k6Di2pav4_NUKMSxyzL2COR-hoCb_oQAvD_BwE>. Päivitetty 18.7.2021. Luettu 8.10.2022.

10. Kasurinen, Jussi Pekka. 2013. Ohjelmistotestauksen käsikirja. E-kirja. Docendo.

11. Rajlich, Vaclav. Software Engineering : The Current Practice, CRC Press LLC, 2011. ProQuest Ebook Central.

12. Chopra, Rajiv. Software Quality Assurance, Mercury Learning & Information, 2018. ProQuest Ebook Central.

13. Regression Testing Guide. Verkkoaineisto. Ranorex. <https://www.ranorex.com/resources/testing-guides/regression-testing/> Luettu 27.8.2022.

14. Khrupa, Anna. 2020. Regression Testing: How, When, and Why You Should Do It. Verkkoaineisto. <https://testfort.com/blog/regression-testing-how-when-and-why-you-should-do-it>. Päivitetty 28.5.2020. Luettu 27.8.2022.

15. Guberna, Dragos. Testiautomaatio-opas. Verkkoaineisto. VALA Group Oy. <https://www.valagroup.com/fi/oppaat/testiautomaatio-opas/>. Luettu 25.9.2022.

16. Hamilton, Thomas. 2022. What is Automation Testing? Test Tutorial. Verkkoaineisto. <https://www.guru99.com/automation-testing.html>. Luettu 25.9.2022.

17. What is Automated Testing. Verkkoaineisto. Smartbear. <https://smart-bear.com/learn/automated-testing/what-is-automated-testing/>. Luettu 25.9.2022.

18. Bose, Shreya. 2020. Testing Pyramid: How to jumpstart Test Automation. Verkkoaineisto. <https://www.browserstack.com/guide/testing-pyramid-for-test-automation>. Luettu 25.9.2022.

19. Knott, Danlie. 2015. The Mobile Test Pyramid. Verkkoaineisto. <https://www.ministryoftesting.com/dojo/lessons/the-mobile-test-pyramid>. Luettu 25.9.2022.

20. Da Silva Paternoster, Michaël. 2022. Pros and Cons of Automated Testing. Verkkoaineisto. <https://uilicious.com/blog/pros-cons-automated-testing/#what-are-the-pros-of-automated-testing>. Luettu 11.10.2022.

21. Ranorex Studio Features. Verkkoaineisto. Ranorex. <https://www.ranorex.com/company/>. Luettu 3.10.2022.

22. About Ranorex. Verkkoaineisto. Ranorex. <https://www.ranorex.com/features/>. Luettu 3.10.2022.

23. Saurabh. 2022. What is Jenkins? Verkkoaineisto. <https://www.edureka.co/blog/what-is-jenkins/>. Päivitetty 28.3.2022. Luettu 3.10.2022.