Ossi Lahti

Käyttöliittymän automaatiotestauksen hyödyt manuaalitestaukseen nähden

Tietotekniikan Kandidaatintutkielma

29. huhtikuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Ossi Lahti

Yhteystiedot: otvlahti@student.jyu.fi

Ohjaaja: Tytti Saksa

Työn nimi: Käyttöliittymän automaatiotestauksen hyödyt manuaalitestaukseen nähden

Title in English: Advantages of test automation in making of a user interface

Työ: Kandidaatintutkielma

Sivumäärä: 19+0

Tiivistelmä: Automaatiotestaus on laajasti tunnettu käsite sovellusmaailmassa, mutta sen konkreettisista hyödyistä kiistellään vielä tänäkin päivänä. Tässä tutkielmassa tarkastellaan testiautomaation näennäisiä hyötyjä manuaalitestaukseen verraten. Aiheen taustoja tutkitaan sekä liiketoiminnan että teknisten detaljien kannalta. Tutkielmassa käydään myös läpi testijärjestelmien toteutusmekanismeja ja siirtymää manuaalitesteistä automaatiotesteihin. Lähdekirjallisuuden perusteella automaatiotestauksen voidaan ennustaa olevan merkittävä osa tulevaisuuden työkulttuuria vähintäänkin keskisuurissa yrityksissä.

Avainsanat: testiautomaatio, ohjelmistotestaus, manuaalitestaus, käyttöliittymätestaus

Abstract: Automated software testing is a widely recognized concept among the software business world, but even until today, its concrete advantages are to be disputed. This bachelor's thesis discusses the main advantages of test automation, and compares it to manual software testing. The topic is studied in a business and a technical perspective. The thesis also discusses that how to implement a test suite and how the transition to automated tests has been like. On the basis of the sources, test automation could be forecasted to be a crucial part of work environments in the future, at least on the medium sized companies.

Keywords: test automation, software testing, manual testing, ui testing

Termiluettelo

DevOps	Toimintamalli sähköisten palveluiden ja ratkaisujen tuotantoon. Automatisoi testauksen, koodin ja ylläpidon integrointia (Ebert ym. 2016).
Agile	Toimintamalli ohjelmistokehityksessä. Päätavoitteena tehdä enemmän ohjelmistoon uusia ominaisuuksia ja vähemmän dokumentaatiota. Agile-menetelmässä pyritään olemaan enemmän asiakkaan kanssa tekemisissä, ja mukautumaan heidän vaatimuksiin (Fowler, Highsmith ym. 2001).
Scrum	Projektinhallinnan viitekehys, jonka tavoitteena on lähentää tuotekehitystiimin ja asiakkaan välistä yhteistyötä (Schwaber ja Sutherland 2011).

Sisällys

1 Johdanto

Web-käyttöliittymien testaaminen on ollut murroksessa yli vuosikymmenen ajan. Uusia JavaScriptin viitekehyksiä syntyy jatkuvasti, ja tätä myötä myös testaustyökaluja joudutaan päivittämään lyhyin aikavälein. Käyttöliittymien testaaminen on yksi kolmesta testauksen pääalueesta. Käyttöliittymän testaamista edeltää yksikkötestaaminen, sekä rajapintatason testaaminen (Cohn 2010). Käyttöliittymiä voidaan testata manuaalisesti sekä automaattisesti. Manuaalitestaamisessa pääkäyttäjät pyrkivät noudattamaan askel askeleelta laadittuja käyttöohjeita, joiden mukaan sovelluksen tärkeimpiä ominaisuuksia testataan.

Testausammattilaisten tulevaisuus näyttää lupaavalta, sillä automaatiopohjaisesta testaamisesta on tullut trendi. Agile-toimintatavan yleistyessä myös testiautomaation osaajille on huutava tarve. Testiautomaatio-osaamista tarvitaan muun muassa jatkuvien testien (engl. *Continuous Testing*) laatimisessa, ja ylläpidossa. Koska testauksen osuus on usein yli 60 prosenttia projektiin käytetystä ajasta (Kumar ja Mishra 2016), yritykset pyrkivät lisäämään tehokkuutta sovelluksen testaamiseen. Testauksen ollessa merkittävä osa projektikokonaisuutta, yritykset voivat tehdä huomattavia säästöjä sen oikein toteuttaessaan.

Tässä tutkielmassa perehdytään käyttöliittymien automaattitestaamisen hyötyihin manuaalitestaamiseen nähden. Työssä mietitään automaatiotestauksen hyötyjä erikokoisten yritysten, sekä eripituisten projektien kannalta. Tutkielmassa käydään myös läpi hieman testauksen historiaa ja sitä, mikä on nostanut testiautomaation suosioon. Tutkielmassa käydään läpi testiautomaation haasteita, sekä mietitään millaisissa tilanteissa manuaalitestaus olisi parempi vaihtoehto. Aihe on valittu kirjoittajan henkilökohtaisen mielenkiinnon ja ajankohtaisuuden vuoksi.

Tutkielman rakenne koostuu seuraavasti: toisessa luvussa avataan olennaisia käsitteitä tutkielmaan liittyen, luvussa kolme käsitellään testauksen historiaa ja siirtymistä automaatioon. Luvussa neljä pohditaan syvemmin lähdekirjallisuuden avulla automaatiotestaamisen hyötyjä manuaalitestaukseen verraten. Asiaa pohditaan erityisesti liiketoiminnan kannattavuuden näkökulmasta, sekä siten, millaisissa projekteissa automaatiosta olisi erityisesti hyötyä. Viimeisessä luvussa aiheesta tehdään yhteenveto ja johtopäätökset.

2 Testauskäsitteistöä

Sovelluksien testaamisella tavoitellaan dynaamista varmistusta ohjelmiston toimivuudelle. Toimivuudella tarkoitetaan laadunvarmistusta: sovelluksessa tulee olla kaikki halutut toiminnallisuudet, sekä sen tulee toimia virheettömästi. Testaus koostuu yleensä rajallisesta määrästä testitapauksia, joilla yritetään turvata sovelluksen oikeintoimivuus (Bourque ym. 1999).

Web-sovellusten ja muiden graafisten käyttöliittymien testaamisen voi jakaa karkeasti kahteen alalajiin: automaatiopohjaiseen testaamiseen, sekä manuaalitestaamiseen. Koska käyttöliittymätestaaminen on olennainen osa tätä tutkielmaa, sitä avataan tässä erikseen lyhyesti.

2.1 Käyttöliittymätestaus

Käyttöliittymätestaus on nimensä mukaisesti testausta, joka liittyy käyttöliittymän validointiin ja verifiointiin. Käyttöliittymätestaus on korkean tason testausta — se ei käsittele ohjelman syvää toiminnallisuutta lainkaan. Käyttöliittymätestauksella voidaan varmistaa esimerkiksi tiettyjen nappien tai sisäänkirjautumiskenttien ilmestyminen verkkosivuille. Käyttöliittymätestaukseen käytetään noin 10 prosenttia testaukseen kulutettavasta kokonaisajasta (Radziwill ja Freeman 2020).

Käyttöliittymätestausta edeltää integraatio- ja yksikkötason testaaminen. Intergaatio- tai palvelutason testaaminen vie noin 20 prosenttia, ja yksikkötestaus vie noin 70 prosenttia kaikesta testaukseen käytettävästä ajasta. Lukemat ovat peräisin Mike Cohnin (2010) kehittämästä testiautomaatiopyramidista. Käyttöliittymätestausta pidetään testaamisen haastavimpana vaiheena, sillä se vie paljon aikaa, sekä käyttöliittymän osien muuttaminen rikkoo herkästi testitapauksia (Contan, Dehelean ja Miclea 2018).

2.2 Manuaalitestaus

Manuaalitestauksella tarkoitetaan karkeasti sanottuna testausta, jonka tekee ihminen. Testatessaan ohjelmistoa testaaja asettuu loppukäyttäjän rooliin. Manuaalitestauksessa pyritään noudattamaan selkokielelle tehtyjä testitapauksia, jotka suoritetaan askel askeleelta (Kaur

ja Kumari 2011). Manuaalitestaus on automaation puutteen takia runsaasti aikaa kuluttava vaihe. Automaatiotestauksen rinnalla manuaalitestaus on epäluotettavampi vaihtoehto, sillä ihminen on virhealttiimpi kuin tietokone.

Manuaalitestaus on joka tapauksessa tärkeä osa projektia, ja joissain tapauksissa käytännöllisempi virheiden havainnointiin (Dukes, Yuan ja Akowuah 2013). Tämä johtuu siitä, että ihminen pystyy havainnoimaan testitapausten ulkopuolelta sellaisia asioita, jotka eivät ole sovelluksessa haluttuja toimintoja. Manuaalitestausta voidaan käyttää esimerkiksi sisäänkirjautumisen validointiin. Sillä voidaan varmistaa esimerkiksi virheilmoitusten toimintaa, tai että järjestelmään ei päästä murtautumaan käyttäen invalidia salasanaa.

Graafisen käyttöliittymän testauksessa manuaalitestaus voi olla black box -testaamista, white box-testaamista, tai hyväksymistestausta (acceptance testing). Black box -testaamisessa sovellus kuvitellaan mustana laatikkona, jolloin testaajalla ei ole tiedossa sovelluksen rakennetta tai koodia (Nidhra ja Dondeti 2012). White box -testaamisessa testaajalla on tiedossa sovelluksen lähdekoodi, tästä syystä testaajana on usein koodari itse. Hyväksymistestauksessa lopulta katsotaan, päteekö ohjelmiston toiminnallisuus liiketoiminnan vaatimuksiin (Nidhra ja Dondeti 2012).

2.3 Automaatiotestaus

Automaatiotesteillä tarkoitetaan sovelluksen ulkopuolisella työkalulla automaattisesti suoritettavia testitapauksia. Testitapaukset koostuvat ihmisen luomista skripteistä, joita työkalu ajaa. Työkalu ajaa testitapaukset itsenäisesti, ja vertaa tuloksia haluttuihin tuloksiin (Huizinga ja Kolawa 2007). Automaatiotestaus on oikein toteutettuna luotettava testaustapa, sillä testitapaukset ovat toistettavia, ja täten tekevät aina saman asian. Automaatiotestausta voidaan käyttää melkein kaikessa, missä manuaalitestaustakin. Esimerkiksi sisäänkirjautumisen yhteydessä työkalu asettaisi käyttäjän ennaltamääritellyn skriptisyötteen suoraan sisäänkirjautumiskenttiin, ja validoisi sen oikeellisuuden. Yleinen harhaluulo on, että automaatiolla pystytään korvaamaan kaikki testit - näin ei kuitenkaan ole. Manualitestausta tarvitaan uusien ominaisuuksien testaamiseen. Automaatiotestit luodaan, kun ominaisuuden toiminta on validoitu ja verifioitu toimivaksi (Pettichord 1999).

Kun aletaan toteuttamaan testiautomaatiojärjestelmää, pitää ensiksi miettiä, millä työkalulla työtä aletaan tekemään. Tämän jälkeen lähdetään pohtimaan sitä ominaisuutta projektista, jonka testaus halutaan automatisoida. Tämän jälkeen luonnollisesti suunnitellaan testitapaukset, jonka jälkeen ne toteutetaan. Suunnitteluvaiheessa tehdään testausskriptit, jota testiautomaatiotyökalut hyödyntävät (Mosley ja Posey 2002). Työ jatkuu toteutuksen jälkeen ylläpidon muodossa. Manuaalitestauksen suunnitteluvaiheeseen pääsy on nopeampaa, sillä työkalun hankintaa, tai testiautomaation rajaamista projektin eri osa-alueelle ei tarvitse pohtia.

3 Transitio käyttöliittymätestauksen automaatioon

Tässä osiossa käsitellään siirtymää manuaalitestauksesta automaatiotestaukseen. Perehdytään myös hieman automaatiotestauksen teknisiin haasteisiin, sekä siihen, miten automaattisia testitapauksia luodaan.

3.1 Siirtymä automaatiotesteihin

The art of software testing-kirjan (Myers, Sandler ja Badgett 2011), katsotaan olleen modernin ohjelmistotestaamisen pohja. Se on ensimmäinen kirja maailmassa, joka keskittyi pelkästään sovellustestaamiseen. Myers erotti debuggaamisen, eli virheenjäljityksen testaamisesta (Myers, Sandler ja Badgett 2011). Tämän johdosta sovelluskehittäjät saivat inspiraation erottaa tärkeät kehitysvaiheet toisistaan.

Testaus kasvatti suosiotaan merkittävästi 20 vuoden aikana. Vuonna 1986 Paul Rook kehitti idean V-mallista, joka kuvaa projektin eri vaiheissa suoritettavaa testaamista. V-malli on johdettu vesiputousmallista, ja sen avulla pystytään helposti seuraamaan, missä testauksen vaiheessa ollaan menossa (Rook 1986).

Merkittävin testausideologia syntyi vuonna 2003, kun testilähtöinen kehitys sai alkunsa. Ajatus testilähtöisen kehityksen taustalla on, että uudelle ominaisuudelle täytyy laatia testit, ennen kuin se toteutetaan (Beck 2003). Testitapaukset kulkevat aina sovelluksen mukana, ja kaikki testitapaukset pitää ajaa aina, kun uusia ominaisuuksia tehdään lisää. Käyttöliittymätestaaminen kasvatti nopeasti suosiotaan siitä alkaen, kun Selenium kehitettiin vuonna 2003. Se on web-sovellusten testaamiseen tarkoitettu työväline. Selenium hyödyntää yksinkertaisia skriptejä, joilla se suorittaa testejä selaimessa (Holmes ja Kellogg 2006). Selenium sulautetaan selaimeen pääosin JavaScriptiä käyttämällä. Selenium on tänä päivänä yksi suosituimpia JavaScript-pohjaisten web-käyttöliittymien automaatiotestaustyökaluja Robot Frameworkin ohella.

3.2 Manuaalitestauksen toteuttaminen

Manuaalitestausta kehutaan siitä, että sen avulla pystytään parhaiten selvittämään sovelluksen virheet ja haavoittuvuudet. Tämä johtuu siitä, että manuaalitestaus vaatii testaajalta laajaa tuntemusta sovelluksen halutusta toimivuudesta (Itkonen, Mantyla ja Lassenius 2009). Manuaalitestauksen suurin heikkous on, että testejä ei voida toistaa.

Manuaalitestauksen toteutus lähtee liikkeelle testitapausten suunnittelusta, jolloin mietitään ominaisuuksia, jotka on välttämätöntä testata. Testitapaus koostuu ennakkoasetelmasta, syötteen ohjeistuksesta vaiheittain, sekä toivotusta lopputuloksesta (Sharma 2014). Kun samojen ominaisuuksien pariin liittyvät testitapaukset yhdistetään kokoelmaksi, syntyy testijoukko (engl. *test suite*). Vähintään aina, kun projektin seuraavaa vaihetta toimitetaan asiakkaalle, testataan testijoukot bugien varalta läpi. Yleensä testijoukon parissa työskentelee tiimin sisältä useampi koodari. Koska manuaalitestaus ei vaadi koodausta, laadunvarmistamisen kannalta sitä voi tehdä esimerkiksi projektipäälliköt tai asiakaspalvelijat. Tämä on samalla yksi huonoista puolista manuaalitestauksessa: se sitoo merkittävän määrän resursseja yhden asian äärelle (Sharma 2014), mikä on kaikesta muusta työajasta pois.

3.3 Automaatiotestien toteutusmekanismit

Automaation parhaita puolia ovat tehokkuus ja ajansäästö. Yksinkertaisten testitapausten automatisointi tuo kehittäjille runsaasti enemmän aikaa keskittyä uusien toimintojen luontiin. Jotta automaatiotestejä voisi toteuttaa, on testitapauksen luojan tunnettava haluttu toiminnallisuus läpikotaisin. Tutkimusten mukaan useimmat viat sovelluksissa huomataan manuaalitesteissä, ja automaatiotestit nähdään ratkaisuna toistuvien testitapausten korvaamiseen. Tämä antaa testaajille ja kehittäjille aikaa tehdä luovia testitapauksia (Itkonen, Mantyla ja Lassenius 2009).

Automaatiotestejä voidaan tehdä monella eri tavalla. Automaatiotestien rakenne koostuu skripteistä - ja skriptejä on paljon erilaisia: lineaarisia, jaettuja, rakenteellisia, aineisto-ohjattuja (engl. *data-driven*), sekä avainsanaohjautuvia (engl. *keyword-driven*) (Fewster ja Graham 1999). Ylläpidettävyyden ja helppokäyttöisyyden näkökulmasta avainsanaohjatut ja aineisto-ohjatut automaatiotestit ovat paras ratkaisu (Laukkanen ym. 2006).

Automaatiotestit ovat tärkeä osa DevOps-viitekehyksiä. Jatkuva integrointi ja jatkuva toimitus on olennainen osa muun muassa suuressa suosiossa olevaa Scrumia. Jatkuvan toimituksen kannalta on todella tärkeää, että koodin laatu varmistetaan jokaisen julkaisun yhteydessä. Tällöin jatkuva testaaminen on kriittinen osa projektia, sillä kaikki testitapaukset ajetaan jokaisen julkaisun yhteydessä (Battina 2020).

4 Automaation hyödyt manuaalitestaukseen verraten

Oikean testausmenetelmän valintaan pätee moni yksityiskohta, kuten projektin ikä ja yrityksen koko. Tässä osiossa käsitellään testiautomaation kannattavuutta yritysten liiketoiminnan kannalta, sekä millaisiin tilanteisiin automaatio sopii parhaiten.

4.1 Testiautomaation kannattavuus liiketoiminnan kannalta

Yleinen mielikuva testiautomaatiosta on, että se ratkaisee kaikki testaamiseen liittyvät tehokkuusongelmat, ja sen avulla säästää paljon rahaa. Usein tehokkuusongelmien ratkaiseminen ei ole niin yksinkertaista. Testiautomaatiojärjestelmä on olennainen osa tuotekehityssykliä. Se tekee projektista monimutkaisemman, ja testiautomaatio-osaajia täytyy kouluttaa ja rekrytoida menestyvän järjestelmän luomiseksi (Hoffman 1999).

Hoffman (1999) tarkastelee tutkimuksessaan automatisoitujen käyttöliittymätestien hyödyllisyyttä sijoitetun pääoman kannalta. Hänen laskelmissaan käytetään oletuksia, eikä reaalimaailman dataa. Tutkimuksessa oletetaan esimerkiksi kyseessä olevan uusi tuote, ja yhden henkilön palkkaukseen kuluvan sata tuhatta euroa vuodessa. Laskelmissa tarkastellaan sijoitusten kannattavuutta 12 kuukauden ja 24 kuukauden jaksoissa. Tutkimuksesta käy ilmi, että testiautomaatiojärjestelmä ei ole aina tarpeellinen.

Miettiessä projektin mittakaavaa ja elinikää, automaatiotesteistä on enemmän haittaa lyhytkestoisimmissa projekteissa. Hoffmanin tekemän kustannushyötyanalyysin mukaan (1999) vuodessa testiautomaatiojärjestelmä tuo 13-prosenttiset tappiot. Tämä johtuu siitä, että alkuinvestoinnit järjestelmää luotaessa ovat niin suuret. Kun testiautomaatiojärjestelmä on ollut kaksi vuotta käytössä, siitä tulee voitollinen. Samainen kustannushyötyanalyysi osoitti, että 24 kuukauden jälkeen sijoitetun pääoman tuotto (ROI) oli 55 prosenttia (Hoffman 1999). Tutkimuksessa oletetaan, että 12 kuukauden jälkeen ainoastaan yksi työntekijä huolehtii automaatiotestien ylläpidosta, joten reaalimaailman tapauksessa tuotto pääomalle voi olla pienempi.

Garousi ja Yildirim kollegoineen (2018) toteutti empiirisessä tutkimuksessaan laajat testiau-

tomaatiojärjestelmät kahdelle isolle lakiasioiden hallintaan keskittyvälle yritykselle. Tutkimuksen jälkeen tehtiin havainto, joka osoittaa automatisoidun käyttöliittymätestausjärjestelmän olleen positiivinen asia testaustiimille. Kävi ilmi, että automaatiotestit onnistuen toteuttaessaan testaukseen kuluva aika pystytään lyhentää kahdesta päivästä yhteen tuntiin (Garousi ja Yildirim 2018).

Kumarin ja Mishran (2016) tekemässä tutkimuksessa tarkastellaan testiautomaation merkitystä matemaattisten mallien kautta. Kuten Garousi ja Yildirim (2018) ja Hoffman (1999), myös Kumar ja Mishra osoittavat tutkimuksessaan, että automaatiotestaus tuo yritykselle tuntuvia säästöjä tulevaisuudessa. Tutkimus näin ollen tukee väitettä, että automaatiotestauksen laatu ja tehokkuus on paljon paremmalla tasolla kuin manuaalitestauksessa.

4.2 Oikean testausmenetelmän valinta

Kaikkiin projekteihin ei tarvita automaatiotestausta. Kuten edellä mainittiin, testiautomaatiojärjestelmät alkavat maksamaan itseään takaisin noin kahden vuoden kuluttua. Testaus kuuluu joka tapauksessa hyvin suunnitellun sovelluksen perusperiaatteisiin, joten se on aina läsnä, vähintään manuaalisena.

Testausmenetelmän valintaan vaikuttaa projektin elinkaaren lisäksi esimerkiksi sovelluksen teema. Jos tehdään videopeliä, on huomattavasti vaikeampaa toteuttaa sille automaatiotestausta verrattaen esimerkiksi pankin verkkosivujen rakenteeseen. Politowskin ym. (2021) tekemä kyselytutkimus vahvistaa väitettä, että automaatio on harvemmin käytetty vaihtoehto videopelien testaamisessa. Pelikehittäjien keskuudessa testauksen merkitys kuitenkin ymmärretään tärkeänä osana projektia (Politowski, Petrillo ja Guéhéneuc 2021).

Lazic ja Mastorakis (2008) ehdottavat laajassa tutkimuksessaan, että saadakseen testauksesta oikeaoppista ja hyödyllistä, on otettava käyttöön testausmetriikkaa. Metriikan avulla voidaan varmistaa, että testaaminen on tehokasta ja hoituu asiaankuuluvalla tavalla. Testauksesta voidaan kerätä objektiivista statistiikkaa esimerkiksi sovelluksen eri osa-alueiden bugien määrästä, tai bugien korjausnopeudesta (Lazic ja Mastorakis 2008). Muuta hyödyllistä tietoa metriikoista voisi saada muun muassa bugien tai haavoittuvuuksien vakavuudesta.

Miten yritys pystyy valitsemaan oikean työkalun, jotta automaatiotestaus on mahdollisimman tehokasta? Rafin ym. (2012) tekemän kirjallisuuskatsauksen ja kyselytutkimuksen mukaan parhaimpaan ratkaisuun päästään työkalulla, jota on mahdollisimman helppo käyttää, ja siihen on helppo luoda yksinkertaisia, helposti ylläpidettäviä ja toistettavia testitapauksia. Parhaan työkalun pitäisi olla myös helposti integroitavissa suurimpaan osaan koodausympäristöjä, sekä tukea testiautomaation asteittaista toimitusta (Rafi ym. 2012).

Lähdekirjallisuuden perusteella on selvää, että testiautomaatiojärjestelmän luominen vaatii mittavia alkuinvestointeja. Tämä puoltaa ajatusta siitä, että pienten yritysten puolesta testauksen automaatio ei olisi järin kannattavaa. Manuaalitestauksen puoleen kannattaa myös kääntyä tapauksissa, jos automaatiotestit olisi liian kompleksisia ylläpitää. Testiautomaatiojärjestelmää luotaessa täytyy ottaa huomioon, että vähintään yhden työntekijän tulisi kyetä ylläpitämään testijoukkoa.

5 Yhteenveto

Tässä kandidaatintutkielmassa käytiin läpi olennaiset erot automaatiotestauksen ja manuaalitestauksen välillä. Tutkielmassa käytiin ydinterminologiaa läpi, sekä pyrittiin antamaan lukijalle näkemyksiä testauksen kehityksestä historiasta nykypäivään. Tutkielman pääideana oli eritellä, millaisia hyötyjä automaatiotestauksesta voisi olla sovelluskehityksessä, sekä millaiset tekijät vaikuttavat testausmenetelmän valintaan.

Tutkielma osoitti, että ohjelmistojen testaaminen on kehittynyt merkittävästi 1980-luvun taitteesta. Kun siirrytiin moderneihin projektinhallintatyökaluihin, kuten Scrumiin, on testauskin mullistunut vanhasta. Vuonna 1979 mietittiin vielä debuggaamisen ja testaamisen erottamista toisistaan, kun nykypäivänä pyörii automaatiotestit jatkuvasti tuotantoketjuissa. Jatkuva testaaminen todettiin tutkielmassa erittäin hyödylliseksi ja tärkeäksi osaksi sovelluksen laadunvarmistusta.

Suunnittelu ja toteuttaminen on tärkeimpiä vaiheita testiautomaatiojärjestelmän luomisessa, kuten tutkielmassa tuli selväksi. Ilman hyvää suunnittelua järjestelmän ylläpito tulevaisuudessa voi olla haastavaa, tai jopa mahdotonta. Tästä johtuen testiautomaatiojärjestelmän odotetaan olevan aluksi heikosti kannattava, koska ei voida tietää paljonko korjauksia järjestelmä tarvitsee tulevaisuudessa. Suunnittelun ja toteutuksen oikein onnistuessa automaatiotestaus voi olla suuri menestys yrityksen liiketoiminnalle.

Tutkielman päähavaintona oli se, että automaatiotestauksen luotettavuus, toistettavuus, tehokkuus ja nopeus puoltaa kehittäjiä automatisoimaan testejä. Käyttöliittymien muokkaaminen voi tuoda haasteita automaation toiminnalle, sillä työkalulle annetut skriptit voivat olla vahvasti sidottuja entiseen käyttöliittymään. Tästä syystä suositellaan automatisoimaan ensiksi sellaiset käyttöliittymät, joita ei hienosäädetä kovin usein. Testauksen automaation paras puoli on sen nopeus manuaalitestaukseen nähden. Parhaassa tapauksessa testausaika saadaan lyhenemään manuaalitestauksen kahdesta päivästä automaatiotestien yhteen tuntiin.

Tutkielmassa selvisi myös, että testiautomaatiojärjestelmän kehitys vaatii kattavia alkuinvestointeja, paljon inhimillistä pääomaa sekä aikaa. Tämä puoltaa sitä ajatusta, että sellaisen järjestelmän luominen olisi helpompaa suurelle tai keskikokoiselle yritykselle, jolla on

tarpeeksi resursseja henkilöstön palkkaamiseen ja lisenssien ostamiseen. Tarvitaan kuitenkin lisää tutkimuksia siitä, voisiko esimerkiksi pienet yritykset toteuttaa testiautomaatiota, ja miten kannattavaa se olisi heidän liiketoiminnalleen.

Kaiken kaikkiaan automaatiotestauksessa on merkittäviä ja konkreettisia etuja manuaalitestaukseen nähden liiketoiminnan sekä henkilöstön keskittämisen kannalta. Tulevaisuuden tutkimuksissa voisi keskittyä siihen, millainen testiautomaatiotyökalu olisi kaikkein paras tai voisiko automaatiotestejä korvata vielä paremmilla vaihtoehdoilla.

Lähteet

Battina, Dhaya Sindhu. 2020. “Devops, A New Approach To Cloud Development & Testing”. *International Journal of Emerging Technologies and Innovative Research (www. jetir. org), ISSN,* 2349–5162.

Beck, Kent. 2003. *Test-driven development: by example.* Addison-Wesley Professional.

Bourque, Pierre, Robert Dupuis, Alain Abran, James W Moore ja Leonard Tripp. 1999. “The guide to the software engineering body of knowledge”. *IEEE software* 16 (6): 35–44.

Cohn, Mike. 2010. *Succeeding with agile: software development using Scrum.* Pearson Education.

Contan, Andrei, Catalin Dehelean ja Liviu Miclea. 2018. “Test automation pyramid from theory to practice”. Teoksessa *2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR),* 1–5. IEEE.

Dukes, LaShanda, Xiaohong Yuan ja Francis Akowuah. 2013. “A case study on web application security testing with tools and manual testing”. Teoksessa *2013 Proceedings of IEEE Southeastcon,* 1–6. https://doi.org/10.1109/SECON.2013.6567420.

Ebert, Christof, Gorka Gallardo, Josune Hernantes ja Nicolas Serrano. 2016. “DevOps”. *Ieee Software* 33 (3): 94–100.

Fewster, Mark, ja Dorothy Graham. 1999. *Software test automation.* Addison-Wesley Reading.

Fowler, Martin, Jim Highsmith ym. 2001. “The agile manifesto”. *Software development* 9 (8): 28–35.

Garousi, Vahid, ja Erdem Yildirim. 2018. “Introducing automated GUI testing and observing its benefits: an industrial case study in the context of law-practice management software”. Teoksessa *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW),* 138–145. IEEE.

Hoffman, Douglas. 1999. “Cost benefits analysis of test automation”. *STAR West* 99.

Holmes, Antawan, ja Marc Kellogg. 2006. “Automating functional tests using selenium”. Teoksessa *AGILE 2006 (AGILE’06),* 6–pp. IEEE.

Huizinga, Dorota, ja Adam Kolawa. 2007. “Initial Planning and Infrastructure”. Teoksessa *Automated Defect Prevention: Best Practices in Software Management,* 74. https://doi.org/10.1002/9780470165171.ch3.

Itkonen, Juha, Mika V Mantyla ja Casper Lassenius. 2009. “How do testers do it? An exploratory study on manual testing practices”. Teoksessa *2009 3rd International Symposium on Empirical Software Engineering and Measurement,* 494–497. IEEE.

Kaur, Manjit, ja Raj Kumari. 2011. “Comparative study of automated testing tools: Testcomplete and quicktest pro”. *International Journal of Computer Applications* 24 (1): 1–7.

Kumar, Divya, ja Krishn Kumar Mishra. 2016. “The impacts of test automation on software’s cost, quality and time to market”. *Procedia Computer Science* 79:8–15.

Laukkanen, Pekka, ym. 2006. “Data-driven and keyword-driven test automation frameworks”. *Master’s thesis. Helsinki University of Technology.*

Lazic, Ljubomir, ja Nikos Mastorakis. 2008. “Cost effective software test metrics”. *WSEAS Transactions on Computers* 7 (6): 599–619.

Mosley, Daniel J, ja Bruce A Posey. 2002. *Just enough software test automation.* Prentice Hall Professional.

Myers, Glenford J, Corey Sandler ja Tom Badgett. 2011. *The art of software testing.* John Wiley & Sons.

Nidhra, Srinivas, ja Jagruthi Dondeti. 2012. “Black box and white box testing techniques-a literature review”. *International Journal of Embedded Systems and Applications (IJESA)* 2 (2): 29–50.

Pettichord, Bret. 1999. “Seven steps to test automation success”. *Star West, November.*

Politowski, Cristiano, Fabio Petrillo ja Yann-Gaël Guéhéneuc. 2021. “A Survey of Video Game Testing”. Teoksessa *2021 IEEE/ACM International Conference on Automation of Software Test (AST),* 90–99. IEEE.

Radziwill, Nicole, ja Graham Freeman. 2020. “Reframing the test pyramid for digitally transformed organizations”. *arXiv preprint arXiv:2011.00655.*

Rafi, Dudekula Mohammad, Katam Reddy Kiran Moses, Kai Petersen ja Mika V Mäntylä. 2012. “Benefits and limitations of automated software testing: Systematic literature review and practitioner survey”. Teoksessa *2012 7th International Workshop on Automation of Software Test (AST),* 36–42. IEEE.

Rook, Paul. 1986. “Controlling software projects”. *Software engineering journal* 1 (1): 7–16.

Schwaber, Ken, ja Jeff Sutherland. 2011. “The scrum guide”. *Scrum Alliance* 21 (1).

Sharma, RM. 2014. “Quantitative analysis of automation and manual testing”. *International journal of engineering and innovative technology* 4 (1).