Tampereen yliopisto

Otto Kiander

AUTOMAATTITESTAUSMENETELMIEN VAIKUTUS TESTIKOODIN UUDELLEEN-KÄYTETTÄVYYTEEN

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
05/2020

TIIVISTELMÄ

Otto Kiander: Automaattitestausmenetelmien vaikutus testikoodin uudelleenkäytettävyyteen
Kandidaatintyö
Tampereen yliopisto
Informaatioteknologian ja viestinnän tiedekunta
2020

Ohjelmistojen automaattitestaus on perinteisen käsin tehtävän testauksen lisänä käytettävä testausmuoto, jonka avulla toisteiset testitapaukset suoritetaan automaattisesti. Tässä työssä tutkittiin sitä, kuinka erilaisten automaattitestausmenetelmien käyttö on vaikuttanut testikoodin uudelleenkäytettävyyteen ja tätä kautta automaattitestauksesta saatavaan hyötyyn tapaustutkimuksissa.

Automaattitestaus on yleistynyt yrityksissä työkalujen kehittyessä ja projektien laajuuden kasvaessa. Nykyään käytössä on useita eri testausmalleja, joiden avulla voidaan toteuttaa erilaisia testikehyksiä eri käyttötarkoituksia varten. Yleisiä automaattitestikehyksen käyttötarkoituksia on esimerkiksi regressiotestien suorittaminen ja käyttöliittymätestaus.

Toimiva automaattitestauskehys ohjelmistoprojektissa mahdollistaa ohjelmiston korkean laadun ylläpidon automaattisesti osana projektin elinkaarta ja vapauttaa työntekijöiden resursseja muita työtehtäviä varten. Hyvän automaattitestauskehyksen luomiseen vaaditaan oikein perustein valitut työkalut ja hyvä kokonaiskuva testattavasta projektista. Hyvän testikehyksen avulla testitapaukset voidaan luoda siten, että niitä voidaan käyttää uudelleen projektin eri moduuleissa tai jopa muissa projekteissa.

Työssä havaittiin, että tarkasteltavissa yrityksissä automaattitestikehysten käyttö oli pääasiassa tehostanut ohjelmoijien työntekoa. Hyvän automaattitestauskehyksen ylläpidon ja toiminnan yrityksissä mahdollistivat muun muassa riittävä resurssien saatavuus ja työntekijöiden kokemus automaattitestauksen parissa. Yrityksissä havaitut ongelmat automaattitestikehysten kanssa liittyivät usein työntekijöiden kokemattomuuteen tai testauksen suunnittelun yhteydessä tehtyihin virheisiin, kuten epäsopivien työkalujen valintaan. Työssä ei havaittu merkittäviä automaattitestauksen menetelmistä johtuvia eroja testikehysten toimivuudessa ja koodin uudelleenkäytettävyydessä.

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

csv	comma separated values
REST	representational state transfer

1. JOHDANTO

Ohjelmistojen testaus on osa jokaista ohjelmistokehitysprosessia. Ohjelmistoja testataan, jotta saadaan palautetta ohjelmiston tilasta ja laadusta. Mitä suurempi ohjelma on, sitä enemmän yksinkertaisia testitapauksia sen laadun varmistamiseen vaaditaan. Näitä tapauksia ei suuremmissa projekteissa enää tarvitse tuottaa yksitellen, vaan ohjelmoijien työtä on saatu tehostettua siten, että toisteiset testitapaukset voidaan automatisoida. Automaattitestien luominen on usein myös työlästä, joten niistä pyritään tekemään helposti uudelleenkäytettäviä.

Testikoodin uudelleenkäytettävyys kuvaa sitä, kuinka helposti samoilla testeillä saadaan testattua toista ohjelmaa. On kehitelty menetelmiä, joiden avulla testikoodi saadaan eristettyä testattavuudesta uudelleenkäytettävyyden parantamiseksi. Tällaisia menetelmiä on teollisuudessa käytössä useita.

Tässä opinnäytetyössä selvitän automaattitestauksen käyttöä teollisuudessa. Käsittelen tutkimuksia, joissa on kuvattu testiautomaation käyttöä teollisuudessa ja testiautomaation toteutuksen tehokkuutta. Vertailen myös tutkimuksissa havaittuja eroja aineisto-ohjatun ja avainsanaohjatun automaattitestauksen välillä. Tutkimuksen päätteeksi selvitän, kuinka hyvin yrityksissä ollaan työntekijöiden mielestä onnistuttu automattitestauksen toteutuksessa ja kuinka suuri vaikutus valituilla testaustyypeillä on ollut tehokkuuteen.

Tutkimuksessa selvisi, että valituissa yrityksissä automaattitestauskehys oltiin toteutettu vähintään tyydyttävällä tasolla. Eroja eri testausmenetelmien välillä ei valituissa tutkimuksissa juurikaan eritelty. Parhaisiin automaattitestaustuloksiin päästiin yrityksessä, jossa testikehys oli määritelty parhaiten testattavaa projektia vastaavaksi.

Luvussa 2 käsittelen yleisesti testaukseen liittyvää käsitteistöä ja määrittelen vertailussa tarvittavia käsitteitä ja teknologioita. Luvussa 3 esittelen tarkemmin vertailtavat testausmenetelmät. Luvussa 4 vertailen menetelmiä kirjallisuuslähteiden avulla. Luvussa 5 analysoin tutkimuksen tuloksia ja luvussa 6 kokoan yhteen tekemäni havainnot.

2. OHJELMISTOJEN TESTAUS

Ohjelmistojen testaus on ohjelmiston laadun varmentamista ja liittyy jokaiseen ohjelmistoprojektiin. Tässä luvussa käsitellään ohjelmistojen testaukseen liittyviä peruskäsitteitä.

2.1 Testauksesta yleisesti

Ohjelmistojen testaus on prosessi, jossa analysoidaan ohjelmistoa tavoitteena löytää eroavaisuuksia sen toteutuksen ja vaatimusten välillä sekä arvioidaan ohjelmiston ominaisuuksia [14, s. 76]. Ohjelmiston testauksen tavoitteena on saada tietoa ohjelmiston laadusta ja sen tilasta. Vaikka testaus liittyy ohjelmiston laatuun, täytyy laadunvalvonta ja testaus erottaa toisistaan omiksi osa-alueikseen. Hyvin testattu ohjelma voi olla huonolaatuinen, jos se on suunniteltu väärin perustein tai käyttötarkoitukseen nähden epäsopivalla tavalla. Laadunvalvonta on myös tiukasti kytköksissä ohjelmistontuotantoprosessiin, ja osa laadusta liittyykin itse prosessin eikä niinkään tuotteen arviointiin [12].

Ohjelman kehittämisen kannattavuuden mittarina käytetään yleensä investoinnin tuottoprosenttia (ROI). ROI:n laskemisessa kiinnitetään yleensä huomiota seuraaviin kriteereihin:

- Kuinka paljon aikaa ohjelman tuottamiseen on käytetty ja kuinka paljon aikaa tullaan vielä käyttämään?
- Mikä on ohjelman arvo sen valmistuessa?
- Kuinka paljon kustannuksia ohjelmasta aiheutuu nyt ja tulevaisuudessa?
- Kuinka paljon yritys hyötyy ohjelman toteuttamisesta?
- Kuinka suuri taloudellinen riski jäljellä olevan ohjelman toteuttamiseen liittyy? [5]

Kaikkiin näistä kriteereistä liittyy epävarmuustekijöitä, mutta testaamalla voidaan saada lisätietoa erityisesti ohjelman tuottamiseen käytettävästä ajasta ja projektissa vielä jäljellä olevista kustannuksista.

2.2 Testityypeistä

Ohjelmistojen testauksessa käytettävät testityypit voidaan jakaa karkeasti kahteen osa-alueeseen: korkean tason ja matalan tason testeihin. Korkea ja matala tässä tapauksessa kuvaavat testien abstraktiotasoa. Korkean tason testejä voivat olla esimerkiksi järjestelmätestit tai käyttöliittymätestit. Matalan tason testejä ovat esimerkiksi yksikkötestit.

Yksikkötestauksessa testataan yksittäisiä ohjelmistokomponentteja ja komponenttiryhmiä yksinkertaisilla testitapauksilla [14]. Testitapaukset voivat olla esimerkiksi komponentin käytön simuloimista yksinkertaisilla syötteillä ja toiminnan todentamista dokumentaatiota tai muita vaatimuksia vastaavaksi. Yksikkötestejä suoritetaan usein ja samanaikaisesti paljon, jotta voidaan varmistua ohjelman toimivan oikein komponenttitasolla. Kun ajetaan suuri määrä testejä, käytetään testiskriptejä. Testiskripti on yksityiskohtainen ohjeistus, jonka perusteella voidaan alustaa ja suorittaa jokin testitapaus ja arvioida sen tuloksia. [14]

2.3 Ohjelmiston laatu

Tuotteen laatua on hankala määritellä. Laatu mielletään osittain henkilökohtaiseksi kokemukseksi, mutta mahdollisia määritelmiä sille voidaan esittää eri näkökulmiin perustuen. Laadulla voidaan kuvata esimerkiksi sitä, kuinka korkeita standardeja tuote vastaa tai sitä, kuinka hyvin tehty tuote vastaa suunnitelmia, joiden pohjalta se on valmistettu. [8] Standardit voivat liittyä esimerkiksi kuluttajien tottumuksiin tai yrityksen sisäisiin tavoitteisiin.

Testauksessa pyritään analysoimaan ohjelmiston laatua erilaisten kriteerien pohjalta. Esimerkiksi Mili ja Tchier [20] esittävät laatua mittaaviksi kriteereiksi

- toiminnalliset ominaisuudet
- operatiiviset ominaisuudet
- käytettävyydelliset ominaisuudet
- taloudelliset ominaisuudet
- rakenteelliset ominaisuudet.

Toiminnalliset kriteerit kuvaavat ohjelmiston käyttäytymistä määritelmän mukaisesti. Toiminnallisesti laadukas ohjelma siis päätyy käyttäjän syötteiden perusteella ohjelmassa määriteltyyn tilaan lähes joka tapauksessa.

Operatiiviset ominaisuudet kuvaavat muita ohjelman toiminnasta havainnoitavia ominaisuuksia, kuten latenssia tai tehokkuutta. Näitä ominaisuuksia ei välttämättä määritellä ohjelman määrittelydokumentaatiossa, mutta ne vaikuttavat silti ohjelman koettuun laatuun.

Käytettävyydelliset ominaisuudet kuvaavat ohjelmiston mukautuvuutta ja sopivuutta käyttäjän tarpeisiin. Nämä ominaisuudet todennäköisesti määritellään pääpiirteittäin ohjelman toteutuksessa. Käytettävyydelliset ominaisuudet koostuvat osittain samoista asioista kuin operatiiviset ominaisuudet, mutta sisältävät enemmän käyttäjän henkilökohtaisiin ja kulttuurillisiin tottumuksiin liittyviä yksityiskohtia.

Taloudelliset ominaisuudet kuvaavat ohjelmiston kehittämiseen, käyttämiseen ja jatkokehittämiseen kuluvia resursseja. Nämä ominaisuudet kuvaavat usein sitä, kuinka hyviä päätöksiä ohjelman suunnitteluvaiheessa tai ennen sitä on onnistuttu tekemään ohjelman suhteen.

Rakenteelliset ominaisuudet liittyvät ohjelmiston sisäiseen rakenteeseen. Nämä ominaisuudet mittaavat sitä, miten hyviä ratkaisuja ohjelman toteutuksessa on tehty esimerkiksi ohjelman ymmärrettävyyden ja modulaarisuuden näkökulmasta. Myös ohjelman testattavuus voidaan määritellä rakenteelliseksi ominaisuudeksi.

Edellä mainituista laadun kriteereistä tässä tutkimuksessa tarkastellaan lähemmin toiminnallisia, operatiivisia ja käytettävyydellisiä ominaisuuksia, sillä valitussa aineistossa niitä käsitellään eniten.

Ohjelmiston kokonaislaadusta kertovat erilaiset testauksen mittasuureet (metrics). Mittasuureita tarvitaan testaamiseen liittyvän informaation tulkintaan. Tärkeitä testien laadun mittareita ovat esimerkiksi testikattavuus ja bugien löytymissuhde. [13] Korkea testikattavuus kertoo yleensä korkeammasta ohjelman laadusta.

2.4 Automaattitestaus

Automaattitestaus on ohjelmiston testauselinkaaren pituuden ja hinnan pienentämiseksi kehitetty prosessi, jossa toisteiset ja yksinkertaiset testitapaukset tehdään automaattisesti osana ohjelmiston kehitystä [19]. Tällaisia helposti automatisoitava testitapauksia voivat olla esimerkiksi regressiotestit. Regressiotestaus on valikoivaa testausta, jossa varmistetaan, että järjestelmään kohdistuvat muutokset eivät aiheuta ei-toivottuja muutoksia järjestelmässä [15].

Testiautomaation käyttö on sitä kannattavampaa, mitä suurempi projekti on kyseessä, sillä automaatiotyökaluihin turvautuminen vaatii taakseen mittavan alkuinvestoinnin, jonka tuotto realisoituu hitaasti [19]. On esitetty useita arvioita siitä, kuinka monen testi-iteraatiokerran jälkeen automaattitestien käyttö on kannattavampaa kuin manuaalisten testien, mutta tarkkaa lukuarvoa tälle ei voida antaa, sillä kannattavuus vaihtelee tapauskohtaisesti.

On kuitenkin tutkittu, että tärkeimpiä edellytyksiä onnistuneelle testiautomaatiolle on johdon tuki automaatiolle realististen tavoitteiden ja suunnitellun investoinnin tuottoprosentin muodossa [9]. Konkreettiset tavoitteet auttavat testaajia käyttämään resursseja tehokkaasti, ja johdon ymmärrys automaattitestauksesta edesauttaa sen sisällyttämistä projektiin jo suunnitteluvaiheessa. Huonosti suunnitellun testiautomaation käyttö projekteissa on johtanut yrityksissä kannattamattoman suuren aikaosuuden käyttöön projektin testauksessa [19]. Tällaisissa projekteissa oli usein joko valittu käyttötarkoitukseen sopimaton työkalu tai eriytetty testauksen suunnittelu ohjelman muusta suunnittelusta.

3. TESTAUSMENETELMIÄ

Tutkimuksessa käsitellään aineisto-ohjattua testausta ja avainsanaohjattua testausta automaation näkökulmasta. Kumpikin testausmenetelmistä soveltuu myös automatisoimattomaan testausprosessiin.

3.1 Aineisto-ohjattu testaus

Aineisto-ohjattu testaus on testauksen muoto, jossa testiskriptille syötettävä data ohjaa skriptin suorittamia testejä. Usein tämä data on hyvin yksinkertaista, esimerkiksi csv-muotoista dataa. [21] Aineisto-ohjautuvuudella tarkoitetaan sitä, että aineisto ja testikoodi erotetaan toisistaan, jotta uusia testipermutaatioita voidaan ajaa saman testikoodin avulla. Testimetodit toimivat näin monikäyttöisinä kehyksinä, joiden avulla voidaan testata ohjelmaa monella tavalla tekemättä metodeihin muutoksia. [10]

Testityökalujen ja -datan eristäminen toisistaan mahdollistaa myös samojen testiskriptien uudelleenkäyttöön. Näin tarvittavan testikoodin määrä vähenee ja sen ylläpito on helpompaa. [6]

Taulukossa 1 on kuvattu yksinkertaisen sisäänkirjautumistestin luomiseen tarvittavat testitapaukset. Kyseistä sisäänkirjautumistestiä voidaan ajaa millä tahansa kuvausta vastaavalla datalla, joka on yhteensopivaa testattavan järjestelmän kanssa.

Taulukko 1. *Esimerkki aineisto-ohjatusta sisäänkirjautumistestistä [11]*

Kuvaus	**Testidata**	**Odotettu ulostulo**
Testaa sisäänkirjautuminen oikealla käyttäjänimellä ja salasanalla	Hyväksyttävä käyttäjänimen ja salasanan yhdistelmä	Käyttäjä kirjautuu sisään onnistuneesti
Testaa sisäänkirjautuminen-väärällä käyttäjänimellä ja salasanalla	Virheellisen käyttäjänimen ja salasanan yhdistelmä	Käyttäjää huomautetaan virheestä sisäänkirjautumisen yhteydessä
Testaa sisäänkirjautuminen oikealla käyttäjänimellä, mutta väärällä salasanalla	Hyväksyttävä käyttäjänimi ja virheellinen salasana	Käyttäjää huomautetaan virheestä sisäänkirjautumisen yhteydessä

Aineisto-ohjatun testauksen eduiksi luetellaan:

- testiskriptien monikäyttöisyys
- testidatan muokattavuus
- useiden datasarjojen käyttö saman kohteen testaamiseen. [21]

Kuten kaikessa automaattitestauksessa, myös aineisto-ohjatussa testauksessa hankalinta on liittää automaatio osaksi projektia. Jotta automatisoitu, itseohjautuva testiskripti saadaan toteutettua siten, että sen käyttämiseen tarvitaan vain erilaisia datasarjoja, tarvitaan huolellisesti suunnitellut testitapaukset ja hyvin määritelty testattava ohjelma.

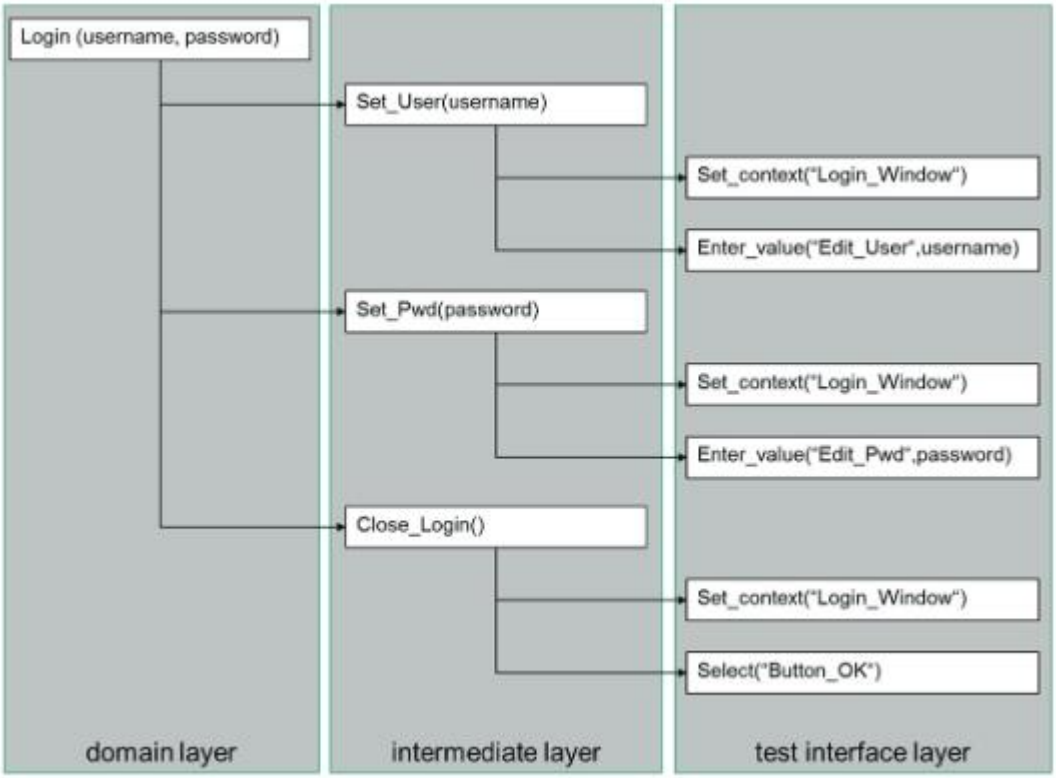
Toimivan aineisto-ohjatun testiautomaation hyvä puoli on myös se, että uusia testejä voidaan luoda muokkaamalla testeissä käytettäviä datasarjoja. Testattavaa ohjelmaa käyttänyt pystyy siis luomaan testitapauksia testiskriptille testidataa muokkaamalla vaikka ei tietäisi ohjelman sisäisesta toiminnasta. [21]

3.2 Avainsanaohjattu testaus

Avainsanaohjattu testaus on testaustyyppi, jossa testidatan ja testiajojen oletettujen tulosten lisäksi testidatatiedostoihin säilötään ohjelman toimintaan liittyviä avainsanoja (keyword). Avainsanoja tulkitaan avustavien skriptien avulla, joita kutsutaan testiajon kontrolliskriptistä. [12]

Avainsanat luodaan siten, että ne kuvaavat matalan tason testitapauksia, kuten yksittäisten moduulien testejä. Avainsanoja yhdistelemällä näistä tapauksista kootaan suurempia testikokonaisuuksia. Avainsanaohjatun testauksen tavoitteena on kattaa suurin osa mahdollisista testitapauksista avainsanoilla tai niistä kootuilla yhdistelmillä. Näin saadaan koottua havainnollistava malli sovelluksen kokonaistestauksesta siten, että sen ymmärtämiseen ei tarvitse ohjelmointiosaamista. [16]

Kuvassa 1 on havainnollistettu avainsanojen hierarkiaa yksinkertaisen sisäänkirjautumistapauksen avulla. Sisäänkirjautumisen yhteydessä tarvittavat toiminnot, kuten käyttäjätunnuksen ja salasanan syöttäminen on koostettu omiksi helposti hahmotettaviksi kokonaisuuksikseen. Kokonaisuuksien sisäistä toteutusta ei tarvitse tietää, jotta voi käyttää näitä toiminnallisuuksia testitapausten kokoamiseen.



Kuva 1. *Esimerkki avainsanahierarkiasta avainsanaohjatussa testauksessa [16, s. 12]*

IEEE:n määritelmässä [16] avainsanaohjatun testauksen eduiksi luetellaan

- helppokäyttöisyys
- ymmärrettävyys
- ylläpidettävyys
- testi-informaation uudelleenkäytettävyys
- tuki testiautomaatiolle
- mahdolliset säästöt kustannuksissa ja aikataulussa.

Avainsanoille tulee lisätä toteutus testikoodiin, jotta avainsanapohjainen malli voidaan yhdistää testiautomaatiokehykseen. Toteutuksessa käytettävä työkalu voidaan valita vapaasti, sillä testausmalli ei ole riippuvainen ohjelman toteutuksesta. [16]

3.3 Menetelmien vertailu

Avainsanaohjatun ja aineisto-ohjatun automaattitestauksen eroavaisuudet liittyvät pääosin niiden tarjoamaan testikoodin abstraktioon. Avainsanaohjatussa automaattitestauksessa testikoodi kootaan avainsanojen alle, kun taas aineisto-ohjatussa testauksessa tätä ei tehdä. Testityyppeihin liittyvät suurimmat eroavaisuudet on koottu taulukkoon 2.

Avainsanaohjattujen testitapausten luomiseen ei tarvita ohjelmointiosaamista, vaan kuka tahansa testattavan ohjelman toiminnallisuudet tunteva voi luoda testitapauksia olemassa olevia avainsanoja hyödyntäen. Uusien avainsanojen luontiin ohjelmointiosaamista kuitenkin tarvitaan, joten esimerkiksi testattavaa järjestelmää laajennettaessa ohjelmointiosaaminen on eduksi uusien testien suunnittelussa.

Taulukko 2. *Suurimpia eroja avainsanaohjatun ja aineisto-ohjatun automaattitestauksen välillä.*

	Avainsanaohjattu	**Aineisto-ohjattu**
Testitapaukset liittyvät testattavan järjestelmän toimintaan		x
Korkea testikoodin abstraktio	x	
Helppo ottaa käyttöön		x
Helppo ylläpitää	x	

Aineisto-ohjattu testikoodi liittyy suoraviivaisemmin kirjoitettuun ohjelmaan, eikä yhtä järjestelmää varten kirjoitettu koodi ole yhtä helposti hyödynnettävissä toisen järjestelmän testaamisessa. Koska testitapauksia ei koota erillisten käsitteiden alle, on aineisto-ohjatun testauksen käyttöönotto helpompaa, mutta ylläpito voi suuremman projektin yhteydessä olla työläämpää.

4. VERTAILUN TOTEUTTAMINEN

Luvussa käsitellään automaatiotestauksen vertailua eri tapaustutkimusten valossa. Käsiteltäväksi on pyritty valitsemaan sellaisia tutkimuksia, joissa testausmenetelmien väliset erot tulevat parhaiten esille.

4.1 Huomioita vertailusta

Avainsanaohjatun testauksen ja aineisto-ohjatun testauksen vertailu on hankalaa, sillä menetelmiä käytetään ohjelmistoprojekteissa usein hyvin eri tavalla. Kummankin menetelmän tarkoituksena on tuottaa uudelleenkäytettävää testikoodia, jonka avulla testejä voi tarvittaessa luoda myös ilman ohjelmointiosaamista. Eroavaisuutena menetelmien välillä on testikoodin yhteys järjestelmään. Avainsanaohjatussa testauksessa luodaan abstrakti malli kuvaamaan testitapauksia. Tämä malli ei liity testattavan ohjelman rakenteeseen, eikä sen sisäistä toteutusta ole määritelty. Aineisto-ohjatussa automaattitestiskriptissä täytyy ottaa huomioon ohjelman sisäinen rakenne, sillä skriptissä käytettävät datan arvot ohjaavat skriptin suoritusta. Kummassakaan tapauksessa testitapausten määrittelijälle ei kuitenkaan ole väliä testikoodin sisäisellä toteutuksella, eikä toteutukseen käytetyillä työkaluilla.

Vertailussa keskitytään testien luomiseen ja uudelleenkäytettävyyteen liittyviin huomioihin käsitellyissä tapauksissa. Tapauksissa on käytetty usein monia testityökaluja ja -menetelmiä, joiden avulla voidaan luoda sekä aineistoon, että avainsanoihin liittyviä testejä. Näistä menetelmistä otetaan huomioon vain ne, joista voidaan selkeästi sanoa, minkälaiseen tarkoitukseen niitä on käytetty.

4.2 Käsiteltävät tapaukset

Tutkimuksessani käsittelen kolmea automaattitestausta käsittelevää tutkimusta. Tutkimuksissa käsitellään automaattitestausta eri tavoin Spotifyn, Scanian ja keskikokoisen ohjelmistoprojektin konteksteissa. Tutkimuksissa on keskenään hyvin erilainen näkökulma automaattitestaukseen, mutta yhteistä kaikissa on automaation onnistuneisuuden vertailu.

Ensimmäisessä tutkimuksessa Alégroth ja Feldt [1] käsittelevät pitkän aikavälin kokemuksia automaattisesta käyttöliittymätestauksessa Spotifyn tuotteiden yhteydessä. Toisessa tutkimuksessa käsitellään [4] avainsanaohjatun testauskehyksen käyttöönottoa Scanian Södertäljen yksikössä. Kolmannessa tutkimuksessa Berlowski et al. [3] käsittelevät korkeasti automatisoitua testausprosessia keskikokoisessa ohjelmistoyrityksessä.

Tutkimuksissa käytetyistä automaattitestaustyökaluista mainitaan esimerkiksi Selenium ja REST-rajapintaan pohjautuva automaattitestauskehys. Selenium on laajasti käytössä oleva avoimen lähdekoodin automaattitestausteknologia, jota käytetään pääasiassa web-testaukseen [11]. REST-rajapinta taasen on moderni web-rajapinta, jonka avulla saadaan muun muassa hyödynnettyä web-resursseja tekstuaalisessa muodossa, jolloin niiden käyttö testauksessa helpottuu [18].

Kaikissa tutkimuksissa on haastateltu testiautomaation parissa työskenteleviä työntekijöitä ja haastattelun perusteella on pyritty arvioimaan automaation toimivuutta tarkastellun projektin yhteydessä. Vain Scaniaa käsittelevässä tutkimuksessa eritellään käytetty automaattitestauksen tyyppi, kun muissa tutkimuksissa puhutaan vain yleisesti automaattitestauksesta.

5. TULOSTEN ANALYSOINTI

Scanian automaatiota käsittelevässä tutkimuksessa selvisi, että käyttöliittymän testien automatisointi on lyhentänyt viivettä ohjelmoijille annettavan palautteen saamisessa. Kuitenkin haastateltavien mielestä osittain työntekijöiden kokemattomuuden ja liian vähäisten automatisointiresurssien vuoksi automaatiotestikehyksestä on päässyt muodostumaan liian monimutkainen. [4] Automaatiota käyttöönotettaessa ei siis olla täysin tiedetty kuinka laaja lopullinen ohjelma tai tarvittava automaatio todellisuudessa tulee olemaan ja testityökalu on valittu liian heikoin perustein.

Tutkimuksessa myös kerrotaan, että koko ohjelman regressiotestit olisi ollut mahdollista suorittaa automaattisesti, jos resurssit olisivat riittäneet automaatiokehyksen laajentamiseen. Toisaalta osa mahdollisesti automatisoivista testeistä kirjoitettiin käsin työkalujen välisten yhteensopivuusongelmien vuoksi. [4]

Mitä enemmän testitapauksia joudutaan kirjoittamaan käsin, sitä huonommin testikoodi on uudelleenkäytettävissä. Testaustyökalun valintaan olisi kuulunut panostaa tämän projektin yhteydessä huomattavasti enemmän resursseja, jotta projektille oltaisiin saatu valittua sopivampi työkalu, jonka avulla testit oltaisiin pystytty automatisoimaan. Tähän ei kenties pystytty resurssien puutteen tai henkilökunnan kokemattomuuden vuoksi.

Spotifyn automaatiota käsittelevässä tutkimuksessa [1] tarkastellaan testiautomaatiota pidemmällä aikavälillä. Erityisesti käyttöliittymätestauksessa käytetty testiautomaatiokehys on ollut kauan jo jatkuvan kehityksen kohteena. Kehys koostuu pienistä uudelleenkäytettävistä testimoduuleista. Suurinta osaa testeistä käytetään korkean tason testauksessa, kuten järjestelmä- ja hyväksymistestauksessa.

Haastateltujen mukaan järjestelmä oli jo vuosia toiminut hyvin suurimmassa osassa sen käyttötarkoituksista. Kuitenkin mobiililaitteiden käyttöliittymätestit ja toiminnallisuudet, joihin liittyi dynaamisia hakutuloksia, jouduttiin testaamaan toisten työkalujen avustuksella. Kuitenkin kenties aikaavievimmäksi osaksi uusien käyttöliittymätestien luomisessa nostettiin testiaineiston muokkaaminen sopivaksi kullekin alustalle.

Jälleen työkalun valintaan olisi voitu panostaa enemmän, jotta tämänkaltaisilta ongelmilta oltaisiin vältytty myöhemmin. Yhdeksi suurimmista työkalun valintakriteereistä nostettiinkin erään työntekijän aiempi kokemus testikehyksen käytöstä. Vaikka kehys vuosien kehityksen jälkeen toimii pääasiassa hyvin, oltaisiin todennäköisesti parempiin

tuloksiin päästy kartoittamalla käytettävää työkalua paremmin automaattitestikehyksen kehityksen alkuvaiheissa.

Keskikokoisen yrityksen testiautomaatiota käsittelevässä tutkimuksessa [3] testiautomaatiota käytettiin pääasiallisesti testattavan ohjelman aikatauluttamattomien julkaisujen nopeaan laadunvarmistukseen sekä yleisesti osana testausta. Nykyisellä kehyksellä tutkimuksen aikana saatiin sovellukselle 47,4 prosentin julkaisuvalmiusaste koko projektin ajalta.

Tutkimuksessa haastateltujen mielestä kehys oli toimiva ja monipuolinen. Tutkimuksen kehys rakentui kahdella eri teknologialla tehdyistä automaattitesteistä: REST-kehyksellä tehdyistä yksikkö- ja regressiotesteistä ja Seleniumilla tehdyistä käyttöliittymätesteistä. Yhden käyttöliittymätestin luomiseen ja ylläpitoon kuuluva aika oli yhteensä kolminkertainen vastaavan regressiotestin luomiseen ja ylläpitoon verrattuna. Keskimäärin Seleniumilla tehdyn testin ylläpitoon myös kului noin kolmasosa kaikesta sen parissa käytetystä ajasta, kun taas REST-kehyksellä luodulla testillä vastaava arvo oli noin 15 prosenttia.

Tutkimuksissa tehtyjä keskeisiä havaintoja on koottu taulukkoon 3.

Taulukko 3. *Vertailtavissa tutkimuksissa esitettyjä havaintoja.*

	Scania	Spotify	Keskikokoinen projekti
Automaattitestaus on tehostanut toimintaa	x	x	x
Nykyinen automaattitestauskehys toimii käyttötarkoituksessaan	x	x	x
Nykyisessä automaattitestauskehyksessä on vielä parannettavaa	x	x	x
Käytettävät automaattitestaustyökalut on valittu hyvin			x
Automaatiotestaukseen käytetään riittävästi resursseja		x	x

Kaikissa tutkimuksissa havaittiin, että automaatiotestaus oli tehostanut toimintaa pitkällä aikavälillä. Senhetkisten automaatiotestauskehysten toimintaan oltiin myös pääosin tyytyväisiä. Pisimpään kehityksen kohteena olleeseen Spotifyn automaattitestikehykseen oltiin kaikista tyytyväisimpiä, kun taas Scanian tapauksessa kehyksen todettiin toimivan tarkoituksessaan, mutta ei aina luotettavasti.

Kussakin tutkimuksessa oltiin sitä mieltä, että omassa testikehyksessä on vielä parannettavaa, mutta tämä korostui Scanian tutkimuksen kohdalla. Kehystä pidettiin liian monimutkaisena ja valittuja testaustyökaluja epäsopivina käyttötarkoitustaan varten.

6. YHTEENVETO

Tässä tutkimuksessa vertailtiin käytössä olleiden automaattitestauskehysten toimintaa yrityksissä niiden parissa työskentelevien työntekijöiden haastattelujen perusteella. Tutkimuksessa myös vertailtiin kehyksissä käytettyjä automaattitestausmenetelmiä ja näiden välisiä eroja. Tutkimuksessa selvisi, että automaattitestauksen toteuttaminen on ollut kannattavaa kaikissa tarkastelluissa yrityksissä riippumatta käytetyistä menetelmistä. Menetelmien välisiä eroja ei voitu vertailla luotettavasti valituissa kohdetutkimuksissa.

Tulevaisuudessa tutkimusta voitaisiin jatkaa luontevasti joko selvittämällä tarkemmin automaattitestauksen onnistumiseen ja epäonnistumiseen johtavia syitä tai selvittämällä automaattitestauksessa käytettävien menetelmien välisiä eroja käytännössä. Automaattitestauskehyksen toteutukseen liittyviä syitä voitaisiin vertailla muiden tapaustutkimusten avulla tai kohdennetusti jonkin tietyn yrityksen kohdalla. Menetelmien välisiä eroja voitaisiin tutkia esimerkkitestikehyksen avulla, joka toteutetttaisiin eri menetelmin.

Tutkimuksessa oltaisiin voitu keskittyä enemmän automaattitestausmenetelmien vertailuun tai jättää ne kokonaan ulkopuolelle vertailusta. Jos menetelmiä ei oltaisi käsitelty ollenkaan, olisi tutkimus ollut suoraviivaisempi ja oltaisiin voitu keskittyä tarkemmin yritysten automaattitestauksen toteutukseen. Tutkimuksen olisi voinut myös rakentaa kokonaan menetelmien vertailun ympärille, jos menetelmiin liittyen olisi löytynyt yksityiskohtaisempaa aineistoa.

LÄHTEET

[1] Alégroth E, Feldt R. On the long-term use of visual gui testing in industrial practice: a case study. Empirical Software Engineering. 2017 Dec;22(6):2937–71.

[2] Arya, K. V, Verma, H. Keyword driven automated testing framework for web application. In: 2014 9th International Conference on Industrial and Information Systems (ICIIS). IEEE; 2014. p. 1–6.

[3] Aziz Y. Exploring a keyword driven testing framework : a case study at Scania IT [Internet] 2017. (UPTEC STS). Saatavissa: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-326139.

[4] Berłowski J, Chruściel P, Kasprzyk M, Konaniec I, Jureczko M. Highly Automated Agile Testing Process: An Industrial Case Study. 2015 Nov 30;10(1):69–87.

[5] Cantor M. Calculating and improving ROI in software and system programs. Communications of the ACM. 2011 Sep 1;54(9):121–30.

[6] Cocchiaro C. Selenium framework design in data-driven testing : build data-driven test frameworks using Selenium WebDriver, AppiumDriver, Java, and TestNG . Birmingham, England: Packt Publishing; 2018.

[7] Everett, G. D., McLeod, R. Software testing: testing across the entire software development life cycle. Piscataway, New Jersey: IEEE Press; 2007.

[8] Garvin, D. What Does “Product Quality” Really Mean? Sloan Management Review (pre-1986) [Internet]. 1984 Oct 1;26(1):25–43. Saatavilla: http://search.proquest.com/docview/2081489739/

[9] Graham D, Fewster M. Experiences of test automation case studies of software test automation. Upper Saddle River, NJ: Addison-Wesley; 2012.

[10] Gundecha U. Data-Driven Testing. In: Selenium Testing Tools Cookbook [Internet]. Packt Publishing; 2012. Saatavilla: https://app.knovel.com/hotlink/pdf/rcid:kpSTTC0002/id:kt00U5VMO8/selenium-testing-tools/data-driven-testing?kpromoter=Summon

[11] Gundecha U, Cocchiaro C. Learn selenium : build data-driven test frameworks for mobile and web applications with selenium web driver 3 . Birmingham ;: Packt; 2019.

[12] Homès, B. Fundamentals of software testing. Hoboken, New Jersey: ISTE/Wiley; 2012.

[13] Hutcheson ML. Software testing fundamentals: methods and metrics / Marnie L. Hutcheson. Hoboken: Wiley; 2003.

[14] IEEE Standard Glossary of Software Engineering Terminology (610.12-1990). New York, USA: IEEE; 1990.

[15] ISO/IEC/IEEE 29119-1:2013(E): Software and systems engineering Software testing Part 1: Concepts and definitions. IEEE; 2013.

[16] ISO/IEC/IEEE 29119-5 First edition 2016-11-15: ISO/IEC/IEEE International Standard - Software and systems engineering -- Software testing -- Part 5: Keyword-Driven Testing. IEEE; 2016.

[17] Kontio, J., Conradi, R. Software Quality - ECSQ 2002 Quality Connection -7th European Conference on Software Quality, Helsinki, Finland, June 9-13, 2002. Proceedings. 1st ed. 2002. Berlin, Heidelberg: Springer Berlin Heidelberg; 2002.

[18] Kurtz J, Wortman B. ASP.NET Web API 2: Building a REST Service from Start to Finish . 2nd ed. Berkeley, CA: Apress; 2014.

[19] Lewis, W. E. Software testing and continuous quality improvement. 2nd ed. CRC Press; 2004.

[20] Mili, A., Tchier, F. Software testing: concepts and operations. Hoboken, New Jersey: John Wiley & Sons, Inc.; 2015.

[21] Mosley DJ, Posey BA. Just enough software test automation. Upper Saddle River, NJ: Yourdon Press; 2002.

[22] Wiklund K, Eldh S, Sundmark D, Lundqvist K. Impediments for software test automation: A systematic literature review. Software Testing, Verification and Reliability. 2017