Tampereen yliopisto

Tuuli Laitinen

AINEISTO- JA AVAINSANAOHJATUT OHJELMISTOTESTAUSVIITEKEHYKSET

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintutkielma
Joulukuu 2021

TIIVISTELMÄ

Tuuli Laitinen: Aineisto- ja avainsanaohjatut ohjelmistotestausviitekehykset
Kandidaatintutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Joulukuu 2021

Ohjelmistotestauksen merkitys ohjelmistotuotannossa on suuri. Testauksen tarkoituksena on varmistaa kehitettävän ohjelmiston laatu ja toimivuus, joita ilman ohjelmiston arvo on minimaalinen. Tämän tutkielman aiheena on erityisesti testausautomaatiossa käytettävät ohjelmistotestausviitekehykset: aineisto-ohjattu viitekehys ja avainsanaohjattu viitekehys. Tutkielmassa on tarkoitus antaa lukijalle yleiskuva viitekehyksien ominaisuuksista sekä niiden hyödyistä ja haitoista toisiinsa verrattuna. Tämän avulla on tarkoitus luoda käsitys siitä, minkälaisissa tilanteissa kummankin viitekehyksen käyttäminen olisi kannattavaa.

Tutkielma on kirjallisuuskatsaus ja sen teossa on käytetty lähteinä erilaisia tieteellisiä tutkimuksia, artikkeleita ja kirjoja. Tutkielman aihetta on pohjustettu esittelemällä ohjelmistotestausta yleisesti avaamalla ohjelmistotestauksen merkitystä ohjelmistotuotannossa, sen eri tasoja, mahdollisia jakoja sekä testausautomaatiota. Tutkielmassa käydään läpi käsiteltävät viitekehykset ja niiden ominaisuuksia, joita vertaillaan testien rakenteen, käyttöönoton, ylläpidon, käytön ja käyttöympäristöjen näkökulmista. Keskustelussa tiivistetään vertailun tuloksia sekä pohditaan tulosten yleistettävyyttä ja tutkielmaan liittyviä jatkokehitysehdotuksia.

Tutkielmassa päätellään avainsanaohjatun viitekehyksen olevan aineisto-ohjatusta viitekehyksestä pidemmälle kehitetty versio. Molemmilla viitekehysten mukaan toteutetuilla testeillä on osittain rakenteellisia yhtäläisyyksiä, sillä ne hyödyntävät ulkoisia tiedostoja. Näissä tiedostoissa sijaitsee pääosin testissä käytettävä testidata, joista tiedostojen sisältöä käytetään joko testiä suorittavassa testikoodissa tai avainsanojen yhteydessä. Testien monimutkaisuus tuo omia haasteitaan, mutta myös hyötyjä testien toteuttamiseen, joka korostuu avainsanaohjattujen testien yhteydessä. Testausautomaatio vaatii enemmän työtä alku vaiheissa kuin manuaalinen testaus. Tällöin tarvetta testausautomaatiolle on perusteltava, jotta käyttöönotto olisi hyödyllistä. Aineisto-ohjattu viitekehys tuo yksinkertaisemman kosketuksen testausautomaatioon, jolloin sen käyttäminen olisikin hyödyllisempää pienemmissä testausautomaatiota vaativissa projekteissa. Avainsanaohjattu viitekehys soveltuu taas paremmin laajempiin projekteihin.

Avainsanat: ohjelmistotestaus, testausautomaatio, aineisto-ohjattu, avainsanaohjattu, viitekehys

Sisällysluettelo

1 Johdanto

Ohjelmistotuotannon yksi tärkeimmistä osa-alueista on ohjelmistotestaus, jos halutaan varmistua ohjelmiston laadusta, toimimisesta ja vaatimusten täyttämisestä. Ohjelmistojen testaamiseen on olemassa useita erilaisia viitekehyksiä, joista tässä tutkielmassa käsitellään aineisto-ohjattua ohjelmistotestausta ja avainsanaohjattua ohjelmistotestausta.

Teknologia on kehittynyt ja jatkaa kehittymistään nopeasti. Teknologian ja samalla ohjelmistojen käyttäjien vaatimukset ja oletukset ohjelmistojen toiminnasta nousevat, jonka takia ohjelmiston laatu on olennainen muuttuja sen menestykseen. Laadun perusteellinen tarkistaminen ilman testausta on mahdotonta, jolloin ohjelmistotestauksen merkitys ohjelmistotuotannossa kasvaa. Koska ohjelmistoja on erilaisia ja eri tarkoituksiin luotuja, yksi ja sama testausviitekehys ei välttämättä ole sopivin kaikkien ohjelmistojen testauksessa. Tämän takia eri viitekehyksiä on luotu erilaisille ohjelmistoille. Tämän tutkielman aihe on kiinnostava, koska viitekehystä valittaessa on kannattavampaa ottaa käyttöön viitekehys, joka sopii juuri testattavan tyyppiseen ohjelmistoon. Jotta viitekehyksistä osataan valita juuri se kannattavin, on niiden ominaisuuksista tiedettävä ja vertailtava niitä muihin viitekehyksiin. Avainsana- ja aineisto-ohjattujen testausten sisältäessä saman tapaisia ominaisuuksia, niiden vertailu ja erottelu toisistaan on mielestäni mielenkiintoista, jonka vuoksi valitsin nämä kaksi viitekehystä tutkielmaani.

Aikaisemmissa tutkimuksissa aihetta on käsitelty usein jonkin tietyn testaustyökalun yhteydessä tai sen näkökulmasta. Näistä esimerkkeinä toimii tässä tutkielmassa käytetyt Guptan ja Bajpain (2014) ja Wangin ja Hen (2014) teokset, joissa käsitellään aineisto- ja avainsanaohjattuja viitekehyksiä KeyDriver- ja QTP (Quick Test professional) -työkalujen yhteydessä. Kasurisen (2013) kirja ohjelmistotestauksesta sekä Kentin (2007) artikkeli kuvaavat taas aihetta laajemmin. Näissä teoksissa on keskitytty aiheeseen yleisemmällä tasolla, ja ne eivät kuvaile vain yhtä näkökulmaa.

Tässä tutkielmassa käsittelen ohjelmistotestausta, avainsana- ja aineisto-ohjattuja viitekehyksiä, sekä näiden viitekehysten eroja pääosin yleisellä tasolla. Vertailemisen tarkoituksena on erotella kummastakin viitekehyksestä sekä hyviä että huonoja puolia, joiden avulla voidaan päätellä, millaisissa tilanteissa on kannattavampaa käyttää toista viitekehystä ja milloin toista. Tavoitteenani tutkielmassa on antaa lukijalle selkeä käsitys aineisto- ja avainsanaohjatuista viitekehyksistä ohjelmistotestauksen ja -tuotannon kontekstissa ja niiden ominaisuuksista ja soveltuvuuksista tietynlaisien ohjelmistojen testaukseen.

Ensimmäisenä käyn läpi tavat, joilla etsin erilaisia lähteitä tätä tutkielmaa varten. Ennen käsiteltäviä viitekehyksiä esittelen ohjelmistotestauksen perusteita yleisesti. Ohjelmistotestauksen tarkemman käsittelyn jälkeen esittelen kaksi ohjelmistotestauksessa käytettävää viitekehystä, joita vertailen tutkielmassa siitä seuraavassa luvussa. Keskusteluluvussa kertaan ja pohdin tutkielmassa esille tulleita tärkeimpiä huomioita. Viimeisenä tutkielman lopussa on yhteenveto työstä sekä luettelo tutkielmassa käytetyistä lähteistä.

2 Tutkimusmenetelmät

Tämä tutkielma on tehty kirjallisuuskatsauksena. Tutkielmassa käytettyjä lähteitä on etsitty Tampereen yliopiston Andor-hakupalvelussa (2021) olevista tietokannoista, Institute of Electrical and Electronics Engineers -järjestön (IEEE) sähköisestä kirjastosta (2021), Association for Computing Machinery -yhteisön digitaalisesta kirjastosta (2021), ProQuestin tietojenkäsittelytieteiden tietokannasta (2021) sekä Google Scholarista (2021). Hakuja tehdessä olen käyttänyt suomen- ja englanninkielisiä käsitteitä. Suomeksi käyttämiäni hakusanoja ovat olleet muun muassa ”aineisto-ohj*”, ”test*”, ”avainsan*”, ”avainsanaohjat*” ja ”ohjelmistotest*”. Englanniksi hakusanoja ovat taas olleet esimerkiksi ”data-driven”, ”data driven”, ”keyword-driven”, ”keyword driven”, ”software*”, ”software engineering”, ”software testing” sekä ”test automation”. Näitä hakusanoja olen yhdistellyt eri tavoin sulkujen sekä AND ja OR operaattoreiden avulla.

Hakukoneista saatuja tuloksia on ollut määrällisesti hyvin paljon ja osasta niistä voi huomata heti, kuinka ne eivät suoranaisesti käsittele aihettani, jolloin ne ovat karsiutuneet pois mahdollisten lähteiden seasta. Hakukoneissa rajasin mahdollisuuksien mukaan haettaviksi teoksiksi pääosin kirjoja, artikkeleita sekä konferenssijulkaisuja. Näiden lisäksi lisäsin esimerkiksi Andor-hakupalvelussa kriteeriksi sen, että ne ovat saatavilla kirjaston kokoelmissa. Vuosilukuina käytin vaihtelevasti vuodesta 2005–2010 vuoteen 2021.

Hakujen tuloksista luin ensimmäisenä otsikot ja jos otsikko vaikutti siltä, että teos voisi liittyä käsittelemääni aiheeseen luin teoksen abstraktin. Abstraktien perusteella sain paremman kuvan teoksen sisällöstä. Jos abstraktinkin perusteella teos vaikutti relevantilta aiheelleni, selasin teoksen kokonaan läpi tai etsin kohtia, joissa käsiteltiin aiheeseen soveltuvaa asiaa. Jos teos vaikutti aiheen kannalta olennaiselta, valitsin teoksen käytettävien lähteiden listaan. Monesti myös teoksen omista lähteistä löytyi muita teoksia, joita etsin ja otin mahdollisesti lähteeksi tutkielmaani. Näitä muiden teosten lähteitä etsin pääosin

ensin Andor-hakupalvelusta (2021), mutta lopuksi myös Google Scholarista (2021). Lopullisiin lähteisiin valikoituivat ne teokset, joissa käsitellään tutkielman aihetta tarpeeksi laajasti ja keskitytään aiheeseen riittävästi.

3 Ohjelmistotestaus

Ohjelmistotestauksen tarkoituksena on varmistaa, ettei kehitettävässä ohjelmistossa ole virheitä. Jos testauksen aikana kuitenkin löytyy virheitä, tarkoitus on myös varmistaa, että ne tulevat korjatuiksi. (Gupta ja Bajpai, 2014) Ohjelmistotestaus on yksi alue ohjelmistotuotannossa. Muita osa-alueita ohjelmistotuotannossa ovat esimerkiksi ohjelmiston vaatimusten määrittely, ohjelmiston suunnittelu, sen toteutus sekä laadun varmistaminen. Ohjelmistotuotanto kokonaisuudessaan tarkoittaa erilaisia ohjelmiston luomisessa käytettyjä tekniikoita, toimintatapoja, työkaluja ja periaatteita. Niitä käyttämällä on tavoitteena luoda asiakkaan vaatimukset täyttävä ja toimiva ohjelmisto, joka on saatu luotua jossakin vähintään osittain ennustettavissa olevassa ajassa ja budjetissa. (Haikala ja Mikkonen, 2011)

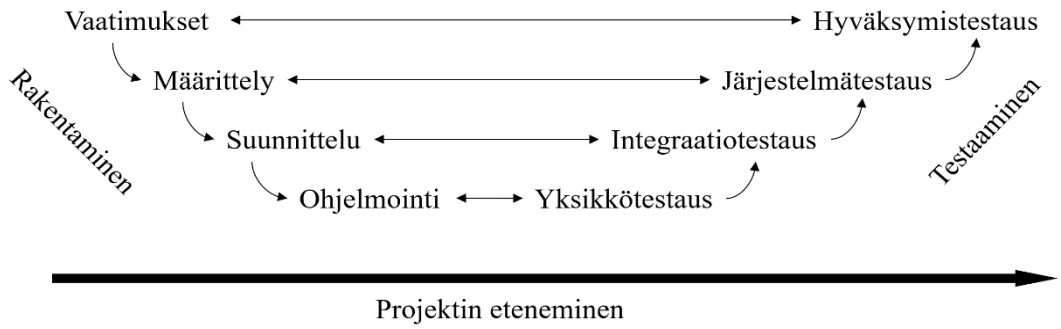
Koska ohjelmistotestauksen määritelmän mukaan ohjelmiston pitää olla toimiva ja samalla täyttää määritellyt vaatimukset, voidaan päätellä, että ohjelmistoa on testattava varmistaakseen nämä ominaisuudet. Pelkästään ohjelmistoa varten kirjoitettua koodia lukemalla testaaminen ei onnistu, jolloin erilaisia tähän tarkoitukseen luotuja työkaluja on hyödynnettävä. Ohjelmistojen testauksessa käytetään erilaisia prosesseja, kuten testitapausten suunnittelua ja koodausta, testien suorittamista ja arviointia, testien refaktorointia sekä tulosten raportointia. Nämä prosessit ovat kuitenkin helposti monimutkaisia ja kalliita. (Garousi ja Pfahl, 2016)

Ohjelmistotestaus on tärkeää ohjelmiston toteutuksen jokaisessa vaiheessa, sillä mitä myöhemmin virhe huomataan, sitä kalliimmaksi se tulee (Gupta ja Bajpai, 2014). Tämän takia ohjelmiston koodin kirjoittamisen aikana testausta onkin suoritettava eri vaiheissa, jolloin tehty virhe huomataan mahdollisimman ajoissa ja se saadaan korjattua ennen kuin sen pohjalta syntyy mahdollisesti uusia virheitä. Virheiden varhaista löytämistä painottaa myös Polo ja muut (2013). Löydettyjen virheiden määrä kertoo myös osittain kehitettävän ohjelmiston laadusta. Siitä myös osittain ohjelmistotestauksessa on kyse: testauksen avulla saadaan tietoa ohjelmiston sen hetkisestä laadusta (Gupta ja Bajpai, 2014). Bosen ja Thakurin (2014) mukaan testauksella onkin suuri merkitys laadukkaiden ohjelmistojen

tuotannossa. Laadukkaiden ohjelmistojen lisääntyneen kysynnän ja samalla niiden kasvaneen kompleksisuuden vuoksi testauksen onnistumisen tarve on suurempaa (Umar ja Zhanfang, 2019; Wang ja He, 2014). Ohjelmistojen testausta korostetaan esimerkiksi testivetoisessa ohjelmistokehityksessä.

Ohjelmiston testaaminen voidaan jakaa kahteen eri tyyppiin: staattiseen testaukseen ja dynaamiseen testaukseen. Staattisessa ohjelmistotestauksessa ohjelmiston koodia ei ajeta, jolloin testaus koostuu pääosin esimerkiksi koodin tai tiedostojen arvioinnista, koodin syntaksin oikeellisuuden tarkistuksesta tai koodin monimutkaisuuden analysoinnista. Dynaamisessa ohjelmistotestauksessa ohjelmiston koodia ajetaan joko osin tai kokonaisuudessaan. Dynaaminen ohjelmistotestaus voidaan jakaa vielä erikseen funktionaaliseen ja ei-funktionaaliseen testaukseen, jossa funktionaalinen testaus testaa ohjelmistolle asetettujen vaatimusten täyttymistä ja ei-funktionaalinen testaus ohjelmiston laadun näkökulmaa. (Gupta ja Bajpai, 2014)

Eri testattaville elementeille on olemassa erilaisia testauksen tasoja (Polo et al., 2013). Ohjelmistotestauksen jakaminen eri tasoihin perustuu testattavan osan kokoon suhteessa koko ohjelmiston kokoon. Yksikkötestaus koostuu yksittäisen osan, kuten luokan, olion tai moduulin, testaamisesta. Integrointitestauksessa taas yhdistellään pienempiä osia tai yksiköitä, jotta saadaan muodostettua osajärjestelmiä testausta varten. Laajimpana järjestelmätestaus sisältää koko ohjelmiston tai järjestelmän testauksen. (Haikala ja Mikkonen, 2011; Kasurinen, 2013) Kasurinen (2013) lisää testauksen tasoihin vielä viimeisenä suoritettavan hyväksymistestauksen, joka sisältää järjestelmätestauksen tapaisesti koko järjestelmän testausta. Tällä tasolla keskitytään kuitenkin erityisesti ohjelmiston laadun tasoon ja vaatimusten täyttämiseen asiakkaan kanssa yhteistyössä. Yksikkötestauksen avulla pyritään löytämään mahdolliset virheet mahdollisimman varhain jo kehityksen alussa, jolloin niitä ei pääse siirtymään integrointitestauksen tasolle. Integrointitestauksessa yhdistellessä osia, niiden yhteensopivuuksista saadaan tietoa, jolloin siinäkin ilmeneviin ongelmiin voidaan puuttua ennen koko ohjelmiston testaamista. Tätä tasojen suhdetta kuvaa testauksen V-malli (kuva 1).



Kuva 1: Testauksen V-malli (kuva muokattu lähteestä (Kasurinen, 2013)).

Testausta voidaan suorittaa joko manuaalisesti tai automatisoidusti (Garousi ja Pfahl, 2016). Manuaalisessa testauksessa testejä suoritetaan itse käsin, kun taas testausautomaatio tarkoittaa erilaisten automaatiotyökalujen ja/tai viitekehysten hyödyntämistä ohjelmiston testaamisessa (Umar ja Zhanfang, 2019). Suuremmissa projekteissa manuaalinen testaus ei ole kannattavaa, sillä laajamittainen manuaalinen testaus on hidasta. Pienemmissä ohjelmistoprojekteissa manuaalinen testaus voi taas olla riittävää. Regressiotestauksessa testataan jo toimivia osia niihin tehtyjen muutosten jälkeen, jotta varmistutaan siitä, että ohjelma toimii edelleen (Kasurinen, 2013). Automatisoitu testaus pyrkii vähentämään manuaalisesti tehdyn regressiotestauksen tarvetta, mutta ilman manuaalista testausta ei kuitenkaan pärjätä (Kasurinen, 2013; Bose ja Thakur, 2014). Manuaalista testausta tarvitaan siis aina.

Testauksen automatisointi ei ole välttämättä aina helppoa tai nopeaa ja siksi sen huonoina puolina pidetään yleisesti siihen kuluvaa aikaa ja rahaa sekä sen käyttöönottoon vaadittavan tietotaidon tasoa (Kasurinen, 2014; Gupta ja Bajpai, 2014; Umar ja Zhanfang, 2019). Tästä voidaan päätellä, että on tärkeää pohtia, kumpi testaustapa on kannattavampi. On hyvä kysyä, onko testauksen automatisoinnille niin suuri tarve, että on kannattavampaa käyttää resursseja automaation toteuttamiseen kuin jatkaa pelkän manuaalisen testauksen hyödyntämistä ohjelmiston testauksessa. Testausautomaation kannattavuus korostuukin niissä tapauksissa, joissa manuaalinen testaus on työlästä. Erityisesti esimerkiksi ketterät menetelmät ja jatkuva integraatio motivoivat testauksen automatisointia eri vaiheissa (Garousi ja Elberzhager, 2017). Koska testausautomaation hyviä puolia ovat useiden samanaikaisten testien ajaminen sekä toistuvien ja tärkeiden tehtävien automatisointi, aikaa yleensä säästyy (Umar ja Zhanfang, 2019). Umarin ja Zhanfangin

(2019) mukaan myös testien tarkkuus ja kattavuus, testaukseen kuluvan vaivan säästyminen sekä toistettavuus ovat hyötyjä, joita testausautomaatiosta saadaan. Aineisto- ja avainsanaohjattuja viitekehyksiä käytetään enemmän automatisoidun testauksen yhteydessä ja ne muodostavat osan testausautomaation viitekehyksistä (Kent, 2007; Gupta ja Bajpai, 2014).

Testausautomaatio on syntyessään koostunut tallentamiseen ja tallenteiden toistamiseen perustuvasta tekniikasta. Tekniikassa tallennetaan käyttäjän tekemiä toimintoja ohjelmistolle ja tallenteiden avulla toiminnot toistetaan samalla tavalla niin monta kertaa kuin tarpeellista. Seuraavaksi testausautomaation kehityksessä eriteltiin data itse ajettavasta koodista eli aineisto-ohjattu testaus syntyi. Avainsanaohjattu testausviitekehys kehittyi seuraavaksi avainsanaohjatusta viitekehyksestä, kun myös tehtävät toiminnallisuudet eriteltiin testiä ohjaavasta koodista. (Kent, 2007; Gupta ja Bajpai, 2014) Kent (2007) määrittelee kaksi ensimmäistä testausviitekehystä perustasoisiksi ja avainsanaohjatun testauksen edistyneemmäksi viitekehykseksi. Myöhemmin testausautomaation yhteyteen on kehittynyt myös muitakin viitekehyksiä.

4 Viitekehykset

Ohjelmistotestausta voidaan tehdä useilla erilaisilla testausmenetelmillä. Tässä tutkielmassa on kuitenkin tarkoituksena käsitellä kahta erityisesti testausautomaatioon liittyvää viitekehystä: aineisto-ohjattua viitekehystä ja avainsanaohjattua viitekehystä. Näiden kahden lisäksi muita testausautomaatioon liittyviä viitekehyksiä ovat nykyään esimerkiksi lineaarinen automaatioviitekehys, kirjastoarkkitehtuuriviitekehys, modulaarinen viitekehys sekä useita viitekehyksiä yhdistelevä hybridiviitekehys (Umar ja Zhanfang, 2019).

4.1 Aineisto-ohjattu ohjelmistotestaus

Aineisto-ohjattu ohjelmistotestaus tarkoittaa kehiteltävän ohjelmiston testaamista testikoodilla, josta on eroteltu testitapauksessa käytettävä testidata. Tämä data sijaitsee erillisissä datatiedostoissa, joista testikoodi ottaa sitä käyttöönsä erilaisilla kutsuilla. (esimerkiksi Cocchiaro, 2018; Kent, 2007; Umar ja Zhanfang, 2019) Testidata voi olla säilöttynä esimerkiksi tavallisessa tekstitiedostossa, tietokannassa, Excel-taulukossa, JSON-tiedostossa tai CSV-tiedostossa, josta testikoodissa toteutetut, toimintoja kuvaavat funktiot, vastaanottavat dynaamista testidataa. Tämän erottelun tarkoituksena on muun muassa

mahdollistaa helpompi uusien testitapausten lisääminen testaukseen sekä testikoodin vaatimien ylläpitotoimenpiteiden ja yleisen koodin määrän vähentäminen. (Cocchiaro, 2018) Testin kulun ollessa omassa testikoodissa, joka lukee dataa erillisistä tiedostoista, testikoodi toimii kuin ajurina datalle (Bose ja Thakur, 2014).

Testaus, jossa testidata on eroteltuna testikoodista, mahdollistaa testidatan muokkaamisen ilman, että tarvitsee koskea itse testikoodiin. Tämä tekee yleisesti datan muuttamisesta helppoa (Gupta ja Bajpai, 2014). Koska testikoodissa olevat toiminnot pysyvät samoina, datan sisältöä muuttamalla saadaan helposti testattua erilaisia tai samanlaisia tapauksia nopeasti.

Aineisto-ohjatun ohjelmistotestauksen hyödyt ja haitat liittyvät pääosin datan erillisyyteen. Testidatan mahdollisuus sisältää monia erilaisia yhdistelmiä erilaisista datoista kasvattaa testien kattavuutta (Kent, 2007). Samassa testissä voidaan siis testata esimerkiksi useampia ohjelmiston testattavia ominaisuuksia. Kent (2007) huomauttaa myös, kuinka tässä käytettävät tavat testien parempaan kattavuuteen eivät ole mahdollisia manuaalisessa testauksessa. Suurimmat hyödyt Wangin ja Hen (2014) mukaan aineisto-ohjatussa ohjelmistotestauksessa ovat useiden testien suorittamisen mahdollisuus samalla koodilla sekä testidatan muuttaminen ilman testikoodiin koskemista. Guptan ja Bajpain (2014) mukaan taas hyötynä ovat myös yksittäisten testipakettien koon pienentyminen ja sitä kautta testauksen ylläpidon helpottuminen. Testaustavassa, jossa käytettävä testidata ja testikoodi ovat samassa tiedostossa, data on kovakoodattuna koodiin. Tämän takia aineisto-ohjatussa testauksessa etuna on myös kovakoodauksen vähentyminen.

Testikoodin ja testidatan yhdessä käyttämisen sujuvuus vaatii paljon tietotaitoa ja osaamista testejä toteuttavilta henkilöiltä (Gupta ja Bajpai, 2014), mikä asettaa rajoitteita testaukselle. Testien tekemiseen vaadittavia tehtäviä ei voida jakaa sellaisille henkilöille, joilla ei ole tarkempaa tietoa esimerkiksi ohjelmoinnista, vaan henkilön on ymmärrettävä ja osattava itse ohjelmoida koodia tai koota testidataa. Edellisessä luvussa esitellyssä testauksen V-mallissa (kuva 1) kuvatuilla korkeammilla tasoilla tarkoituksena on testata ennemmin ohjelmiston kokonaisuutta, kuin pelkästään jotain tiettyä toimintoa. Tällöin toistuva datan syöttäminen testattavaan järjestelmään ei ole testauksen päätavoitteen kannalta olennaista. Tämän vuoksi aineisto-ohjattu testaus ei välttämättä ole parhain testaustapa laajemmille testeille (Kent, 2007), vaan sitä on kannattavampaa käyttää esimerkiksi yksikkötestauksessa testattaessa jotakin tiettyä datankeruutoimintoa.

Taulukko 1: Esimerkki datatiedostosta.

Testitapaus	**Parametri 1**	**Parametri 2**	**Parametri 3**	**Odotettu tulos**
Lisää henkilö	Maija	Korhonen	*maija.korhonen@email.com*	Lisäys onnistunut
Lisää henkilö	Matti	Virtanen	*matti.virtanen@email.com*	Lisäys onnistunut
Lisää henkilö	Aji367s	Mjsy48bd	*aji367sMjsy48bd.email.com*	Lisäys epäonnistunut

Taulukossa 1 on esimerkki datatiedoston sisällöstä, jota voitaisiin käyttää aineisto-ohjatussa testissä. Tässä esimerkissä testataan nettisivun sisäänkirjautumista erilaisilla arvoilla. Ensimmäinen sarake kertoo mitä testitapausta ollaan suorittamassa ja seuraavat kolme saraketta sisältävät siinä testissä käytettävät kolme parametria. Viimeisenä datasta nähdään näillä parametreilla suoritetun testin odotettu tulos, jota voidaan vertailla testissä saatuun tulokseen. Datatiedostoa päästään käsittelemään esimerkiksi ohjelman 1 mukaisella tavalla. Esimerkki on tehty Laukkasen (s. 50, 2006) teoksessa esiintyvän esimerkin pohjalta.

```
data = open('testidata')

for testitapaus in data
   testi = testitapaus.getData('Testitapaus')
   etunimi = testitapaus.getData('Parametri 1')
   sukunimi = testitapaus.getData('Parametri 2')
   sahkoposti = testitapaus.getData('Parametri 3')
   odotettu_tulos = testitapaus.getData('Odotettu tulos')
   // Käytetään testidataa
   // Vertaillaan saatua tulosta odotettuun tulokseen
endfor
```

Ohjelma 1: Esimerkki testikoodista pseudokoodilla esitettynä. Kommenteilla esitelty puuttuvia toimintoja.

4.2 Avainsanaohjattu ohjelmistotestaus

Avainsanaohjattu ohjelmistotestaus vie aineisto-ohjatun ohjelmistotestauksen viitekehyksen hieman pidemmälle. Tässä viitekehyksessä itse testikoodi ja -data ovat myös jaettuna erillisiin tiedostoihin. Lisäksi testeissä hyödynnetään avainsanoja, jotka viittaavat

erillisissä tiedostoissa toteutettuihin testattaviin toimintoihin. Testi koostuu siis kokonaisuudessaan testikoodista, avainsanojen kirjastosta ja testidatasta (Gupta ja Bajpai, 2014). Testikoodi lukee testidatan sisältävästä tiedostosta kokonaisuuden, jossa on annettu suoritettavaksi haluttu toiminto eli avainsana ja mahdollisesti lisäksi tarvittava testidata, jota käytetään parametreina avainsanaa vastaavaa metodia kutsuttaessa. Nämä avainsanojen toteutukset toteutetaan niin sanotuilla kääreillä (engl. wrapper), jotka sisältävät ohjelmistolle tehtävät toiminnot sen kyseisen avainsanan käytön aikana. (Kent, 2007)

Testauksessa käytettävät avainsanat kuvaavat siis ohjelmistolle tehtäviä toimintoja (Kent, 2007; Pajunen et al., 2011; Takala et al., 2009), ja näiden avainsanojen luonti riippuu testattavasta ohjelmistosta (Bose ja Thakur, 2014). Avainsanojen taakse kirjoitetaan avainsanaa kuvaavan toiminnon koodi, jota kutsutaan testin pääkoodissa testidatasta saadun datan mukaan. Esimerkiksi käyttöliittymien testauksessa avainsanat voivat kuvata käyttäjän tekemiä toimintoja kuten jonkin elementin klikkaaminen tai syötteen antaminen.

Avainsanaohjatussa ohjelmistotestauksessa on aineisto-ohjatun ohjelmistotestauksen tapaisesti hyötynä itse testikoodin pituuden lyhyys verrattuna perinteisempiin testikoodeihin. Myös muutoksia tehtäessä niitä tarvitsee tehdä pelkästään tarvittaviin osiin, jolloin kaikkiin testin osiin ei tarvitse tehdä muutoksia. (Pajunen et al., 2011) Yksi suurimpia hyötyjä kyseisessä viitekehyksessä on sen riippumattomuus käytettävästä testaustyökalusta tai sovelluksesta avainsanojen ollessa tallennettuna ulkoiseen tietolähteeseen (Bose ja Thakur, 2014; Gupta ja Bajpai, 2014; Umar ja Zhanfang, 2019). Tämä antaa enemmän vapautta päättää hyödynnettävistä työkaluista ja valita eri tilanteisiin niistä sopivin. Gupta ja Bajpai (2014) listaavat lisäksi testaukseen vaadittavien tiedostojen huoltamisen olevan helppoa ja hyötynä olevan myös skaalautuvuus, avainsanojen kuvainnollisuus, uudelleenkäytettävyys, riippumattomuus ohjelmointikielestä ja muutoksiin sopeutuminen.

Avainsanaohjattua viitekehystä käytettäessä testejä voidaan suunnitella ja toteuttaa osittain jo samanaikaisesti ohjelmiston toteuttamisen aikana (Gupta ja Bajpai, 2014; Takala et al., 2009), jolloin testauksen suunnittelun ja toteutuksen aloittamista ei tarvitse odottaa ohjelmiston valmistumiseen asti. Koska testikomponentit toteutetaan erikseen, kaikilla testauksessa mukana olevista ei tarvitse olla tietoa tai osaamista ohjelmoinnista (Pajunen et al., 2011; Gupta ja Bajpai, 2014). Tällöin työtä voidaan jakaa tarkemmin, jolloin yksi työntekijä keskittyy vain tiettyyn tehtävään testauksen toteuttamisessa. Testauksessa käytettävät avainsanat ovat myös yleensä nimetty niin, että niistä on helppo ymmärtää mitä sen avainsanan määrittelemän toiminnon on tarkoitus tehdä (Kent, 2007).

Tämä vahvistaa vielä enemmän sitä, että testitapauksia ymmärtää niiden ohjelmoijien lisäksi myös testien suunnittelijat ja loppukäyttäjät.

Avainsanaohjatun viitekehyksen rakenteessa on hyvänä puolena myös avainsanojen abstraktius (Pajunen et al., 2011; Gupta ja Bajpai, 2014). Avainsanoja voidaan luoda hierarkkisesti, jos esimerkiksi yhtä laajempaa toimintoa kuvaava avainsana jaetaan pienempiin osiin, joita kuvaa taas uusi avainsana ja niin edelleen. Avainsanoja voidaan myös yhdistää keskenään, joka lisää avainsanojen abstraktiutta yhä enemmän. (Pajunen et al., 2011) Gupta ja Bajpai (2014) jopa luokittelevat avainsanoja korkeamman tason avainsanoihin ja alemman tason avainsanoihin. Korkeamman tason avainsanoja voidaan muodostaa alemman tason avainsanoista, mutta yhdistäminen voi joissain tilanteissa vaatia ohjelmointitaitoja. Abstraktio lisää testitapausten ymmärrettävyyttä ja niiden rakentaminen on helpompaa (Pajunen et al., 2011).

Haittana viitekehyksessä on sen vaatiman työn määrä ja samalla siihen kuluva aika, sekä tiettyjen asioiden tekemiseen vaadittava tietotaito (Gupta ja Bajpai, 2014; Thummalapenta et al., 2012). Avainsanojen yhteydessä sopivan testaustyökalun löytäminen ja sen toimimaan saaminen ohjelmiston kanssa voivat olla haasteellisia (Takala et al., 2009), joka lisää jo valmiiksi normaalia suurempaa työmäärää. Nämä haitat ovatkin yleisiä varsinkin automatisoidun testauksen yhteydessä.

Avainsanaohjattu testaus soveltuu hyvin tilanteisiin, joissa testataan jotakin tiettyä toimintoa useaan kertaan. Siksi aineisto-ohjatun testauksen tavoin avainsanaohjattua testausta voidaan käyttää luonnollisemmin testauksen V-mallin (kuva 1) kahdella alemmalla tasolla kuin ylemmillä tasoilla. Avainsanojen yhdistely mahdollistaa siis myös helpommin suurempien kokonaisuuksien testaamisen, jolloin yksikkötestauksen lisäksi integraatiotestaus sopii kuvaan.

Taulukko 2: Esimerkki datatiedostosta.

Avainsana	**Parametri 1**	**Parametri 2**	**Parametri 3**	**Odotettu tulos**
lisää_henkilö	Maija	Korhonen	*maija.korhonen@email.com*	Lisäys onnistunut
lisää_henkilö	Matti	Virtanen	*matti.virtanen@email.com*	Lisäys onnistunut
lisää_henkilö	Aji367s	Mjsy48bd	*aji367sMjsy48bdemail.com*	Lisäys epäonnistunut

Aineisto-ohjatun testin mukaisesti testidata sijaitsee esimerkiksi taulukkomuodossa erillisessä tiedostossa (taulukko 2). Avainsanaohjatun testin datatiedostossa sijaitsee lisäksi käytettävä avainsana. Taulukossa on listattuna myös käytettävät parametrit, joita on tässä esimerkissä kolme, sekä testin vaatimien toimien tekemisen jälkeen odotettu tulos. Testitapaukset voivat koostua useammista suoritettavista avainsanoista, jolloin testissä testitapausta käydessä suoritetaan useampia avainsanojen kutsuja (Laukkanen, 2006).

Avainsanat on toteutettu erillisissä kooditiedostoissa, joista niitä kutsutaan testikoodissa. Koko testiä ajava testikoodi (ohjelma 2) sisältää datatiedoston luvun ja sen läpikäynnin. Datatiedosto käydään läpi testitapaus kerrallaan ja siitä luetaan halutut avainsanat, joita kutsutaan parametrien kanssa. Avainsanaa vastaavan funktion palauttamaa arvoa verrataan datatiedostossa määriteltyyn odotettuun tulokseen, jonka perusteella määritellään, menikö testi läpi vai ei. Esimerkki on tehty Laukkasen (s. 59, 2006) teoksessa esiintyvän esimerkin pohjalta.

```
data = open('testidata')

// Käydään testitapaukset läpi
for testitapaus in data
   avainsanat = testitapaus.getKeywords()
   // Käydään testitapauksen avainsanat läpi ja suoritetaan ne
   for avainsana in avainsanat
      executeKeyword(avainsana)
      // Tehdään vertailu testin onnistumisesta
   endfor
endfor
```

Ohjelma 2: Esimerkki testikoodista pseudokoodilla esitettynä. Kommenteilla esitelty puuttuvia toimintoja.

5 Viitekehysten erot ja yhtäläisyydet

Aineisto-ohjatulla ja avainsanaohjatuilla viitekehyksillä on osittain samoja piirteitä ja ominaisuuksia, mutta erojakin on monia. Seuraavaksi eri alalukujen yhteydessä käsittelen viitekehysten ominaisuuksia vertaillen niitä toistensa kanssa ja luvun lopussa taulukossa 3 on vertailun tuloksien tiivistelmä.

5.1 Testin rakenne

Rakenteellisesti viitekehysten mukaan toteutetut testit ovat osittain samankaltaisia. Testien kulut määritellään testikoodissa, joka hyödyntää erillisiin tiedostoihin tallennettuja tietoja. Tiedostoista haetaan testeissä käytettävää dataa tai toimintoja suorittavia metodeja. Molemmissa testaustavoissa myös testin tulos päätellään testin pääkoodissa, kuitenkin käyttäen hyödyksi datatiedostosta saatua odotettua tulosta.

Erona viitekehyksissä on tehtävien toiminnallisuuksien toteutus. Aineisto-ohjatussa viitekehyksessä testiä suorittava koodi sisältää myös testidatalle tehtävät toiminnallisuudet, kun taas avainsanaohjatussa viitekehyksessä testissä käytetään avainsanoja. Testikoodissa on kutsuja erillisiin kooditiedostoihin, joissa sijaitsee avainsanojen mukaan toteutetut metodit, joita käytetään testidatan yhteydessä.

5.2 Käyttöönotto

Molempien viitekehysten käyttöönotto vaatii runsaasti töitä. Ennen valmiita ja suoritettavia testejä, niitä varten on luotava testin kulun määrittelevä koodi ja tässä koodissa kutsuttavat ulkopuolisissa lähteissä sijaitsevat tiedot. Molemmissa viitekehyksissä käytettävä ulkoinen testidata on muodostettava, aineisto-ohjatussa testissä tehtävät toimenpiteet pitää lisätä testin koodiin ja avainsanaohjatussa testissä avainsanat täytyy valita ja tarvittaessa lisäksi toteuttaa. Manuaaliseen testaukseen verrattuna käsiteltävissä viitekehyksissä on siis enemmän helposti työläitä työvaiheita, jolloin niiden käyttöönotto kannattaa vain perustellusti.

Koska aineisto-ohjatun viitekehyksen mukaan toteutetussa testissä käytetään ulkoisesti pelkkää testidataa, testien tekijöiden tarvitsee osata ohjelmoida (Bose ja Thakur, 2014). Testien kehittäjiltä vaaditaan tiettyjä ohjelmointitaitoja, jotta testikoodista ja -datasta saadaan yhteensopivia. Koska avainsanaohjatussa testauksessa työnjako on laajempaa, työryhmän osaamisen rajoittava tekijä poistuu (Gupta ja Bajpai, 2014). Työt voidaan jakaa selkeästi asian osaaville työntekijöille, jolloin kaikkien työntekijöiden ei tarvitse osata samoja asioita kuin muut.

5.3 Testin ylläpito

Testien ylläpitoon voi sisältyä esimerkiksi asioiden dokumentointia, tietojen ajan tasalla pitämistä, niiden säilömistä tai niihin muutosten tekemistä. Ylläpito on tärkeää, sillä sen avulla varmistetaan testien pysymistä toimivina ja uudelleenkäytettävinä.

Molemmissa viitekehyksissä testidataan tehtävien muutosten helppouden vuoksi ylläpito helpottuu testeihin tehtävien muutosten näkökulmasta. Muutoksia tehtäessä täytyy kuitenkin avainsanaohjatussa viitekehyksessä huomioida esimerkiksi muokatun avainsanan liittyminen muihin avainsanoihin. Jos avainsana on osana muita avainsanoja, muokkauksen aiheuttamat muutokset on huomioitava myös niissä. Tällöin ylläpito voi vuorostaan hankaloitua.

Avainsanaohjatun viitekehyksen mukaan toteutetuissa testeissä ylläpito hankaloituu mitä enemmän avainsanoja on. Ymmärrettävästi mitä enemmän ylläpidettäviä komponentteja ja kokonaisuuksia on, myös työn määrä ja kompleksisuus kasvaa. Esimerkiksi myös säilömisen näkökulmasta suurempi määrä erilaisia komponentteja vaikeuttaa ylläpitoa. Tiiviimmin ja yhtenäisemmin toteutetut aineisto-ohjatut testit ovat tältä kannalta siis mahdollisesti ylläpidettävämpiä kuin avainsanaohjatut testit.

5.4 Käyttö ja käyttöympäristöt

Molempien viitekehysten käyttö eroaa pääosin vain avainsanaohjatun testauksen avainsanojen käytössä. Käytettävät datatiedostot ovat pääpiirteiltään saman tyylisiä keskenään ja niiden hyödyntäminen testikoodissa toimii samalla periaatteella molemman laisissa testeissä. Käytön aikana tehtävät muutokset aineisto-ohjatussa testissä tekee kuitenkin henkilö, joka on toteuttanutkin testit, mutta avainsanaohjatussa testissä esimerkiksi avainsanoihin kohdistuvat muutokset toteuttaa henkilö, joka on vastuussa avainsanoista, eikä koko testistä.

Testien uudelleenkäytettävyys on olennainen osa viitekehyksiä, joka korostuu etenkin testausautomaatiossa. Samoja testejä voidaan suorittaa ja toistaa useita kertoja peräkkäin joko samoilla tai erilaisilla datoilla. Molemmissa viitekehyksissä käyttöä helpottaakin testitapauksissa käytettävien syötteiden helppo muuttaminen, joka tekee käytöstäkin sujuvampaa.

Viitekehysten mukaan toteutettujen testien sovelluskohteita ohjaa muun muassa se, kuinka samaa testiä voidaan suorittaa useita kertoja eri syötteillä. Tällaisista testeistä onkin hyötyä esimerkiksi web-sovellusten tai nettisivujen kirjautumistoimintojen testaamisessa. Kirjautumista voidaan yrittää useita kertoja antamalla oikeanlaisia tai virheellisiä syötteitä, jolloin toiminnon toimintaa voidaan testata mahdollisimman kattavasti. Myös muita ohjelmistoja, joihin on tarkoitus syöttää paljon tietoa ja saada siitä tietty tulos, voidaan testata esimerkiksi aineisto-ohjatun testauksen avulla. Yksinkertainen esimerkki tällaisesta olisi muun muassa laskin. Monimutkaisemmat ja useampia toimintoja vaativat

testit voidaan paremmin jakaa avainsanoihin, kun taas esimerkiksi nelilaskimen muutamat toiminnot voidaan yksinkertaisemmin toteuttaa aineisto-ohjatun testin kautta. Testejä voidaan toteuttaa erilaisilla testausautomaatiotyökaluilla, joita ovat esimerkiksi Selenium (2021) ja Quick Test Professional eli nykyään Micro Focus Unified Functional Testing (UTF) (2021).

Taulukko 3: Viitekehyksien vertailun tiivistelmä.

	Yhtäläisyyksiä	**Eroavaisuuksia**
Testin rakenne	Erillisten tiedostojen hyödyntäminen Testin tuloksen päättely pääkoodissa	Testattavien toiminnallisuuksien toteutus
Käyttöönotto	Työlästä Ulkoisten testidatojen muodostus	Työntekijöiden ohjelmointitaidot
Testin ylläpito	Testidataan tehtävien muutosten helppous	Avainsanojen huomioinnin tarve muutoksien yhteydessä Ylläpidettävyys avainsanojen määrän kasvaessa
Käyttö ja käyttöympäristöt	Testien uudelleenkäytettävyys Esimerkiksi web-sovellusten testaus Testausautomaatiotyökalut	Avainsanojen käyttö Muutokset koodiin tekevä henkilö

6 Keskustelu

Tämän tutkielman tarkoituksena oli tutustuttaa lukija aineisto-ohjatun ja avainsanaohjatun viitekehysten ominaisuuksiin ja mahdollisesti antaa perusteluja, miksi jompaakumpaa testausviitekehystä tulisi käyttää tietynlaisen ohjelmiston testaamisessa. Viitekehysten esittelylukujen ja vertailuluvun perusteella voitaisiinkin päätellä, että avainsanaohjattu viitekehys on aineisto-ohjatusta viitekehyksestä pidemmälle kehittynyt viitekehys. Yleisesti viitekehyksiä erottaa testeissä tehtyjen toimintojen koodin sijainti; avainsanaohjatussa testissä koodi on avainsanan määrittelyssä ja aineisto-ohjatussa testissä testikoodin

seassa. Aineisto-ohjatusta testistä saataisiinkin avainsanaohjattu testi, kun testin suorittaman toiminnallisuuden toteutus poistettaisiin testikoodista ja säilöttäisiin erilliseen tiedostoon avainsanalla kuvattuna.

Eroavaisuuksia viitekehyksissä on kuitenkin niin käyttöönoton kuin ylläpidon puolella testien rakenteen lisäksi. Testien käyttöönotto ja toteuttaminen vaativat yhdeltä henkilöltä enemmän osaamista aineisto-ohjatuissa testeissä, kun taas työt jakautuvat avainsanaohjatuissa testeissä useammalle ammattilaiselle. Avainsanaohjatussa viitekehyksessä testien ylläpidossa on taas enemmän työtä, etenkin mitä enemmän avainsanoja on käytössä, mitä enemmän avainsanoja on käytetty muiden avainsanojen yhteydessä ja mitä enemmän muutoksia niiden toteutuksiin tehdään niiden elinkaaren aikana. Aineisto-ohjatussa testauksessa ylläpidettävät komponentit ovat paremmin yhdessä paikassa ja niiden ylläpito voi olla tietyissä tilanteissa selkeämpää.

Aikaisempiin tutkimuksiin verrattaessa tutkielmassa ei luotu paljoa uutta tietoa. Tarkoituksena olikin koota yhteen tietoa viitekehyksistä, jotta lukijalle syntyisi laajempi käsitys näiden kahden testaustavan suhteesta toisiinsa. Omana päätelmänäni kuitenkin esitän sen, että vaikka molemmat viitekehykset omaavatkin samoja piirteitä, avainsanaohjattu testi sopii paremmin esimerkiksi laajempien web-sovellusten testaukseen. Aineisto-ohjattu viitekehys taas on kannattavampaa tarpeeksi suurilla, mutta myös tarpeeksi pienillä ohjelmistoilla, jolloin testausautomaation hyödyntäminen on tarpeellista, mutta ei aivan avainsanojen käytön laajuudella.

Tutkielmasta voisi testien eri ominaisuuksien vertailun perusteella tehdä vain alustavia johtopäätelmiä tai havaintoja. Näiden mukaan voitaisiin alustavasti päättää kumpaa viitekehystä olisi kannattavampi käyttää ja millaisessa tilanteessa. Jotta päätöksille olisi enemmän tukea ja perusteita, olisi tarkempi taloudellisen näkökulman tutkiminen ja konkreettisen kokemuksen kuvaaminen tarpeellista. Myös erilaisten työkalujen käyttäminen voitaisiin ottaa osaksi vertailua, jos halutaan tietoa myös käytettävistä työkalusta ja niiden ominaisuuksista. Koska vertailu on vain ominaisuuksien vertailua, ei yleistystä voi kovinkaan laajalti tehdä perustellusti.

7 Yhteenveto

Ohjelmistotestaus on tärkeä osa ohjelmiston kehittämistä. Sen avulla pystytään havainnoimaan ja tulkitsemaan kehitettävän ohjelmiston laatua, jolloin voidaan myös tehdä johtopäätöksiä tarvittavista toimenpiteistä. Näillä mahdollisilla tarvittavilla toimenpiteillä on

tarkoituksena ohjelmiston valmiiksi saaminen tai kehittäminen, mutta lisäksi tarkoituksena on viedä ohjelmistoa laadukkaampaan suuntaan ja varmistaa, että se toimii tarkoituksenmukaisesti.

Aineisto- ja avainsanaohjatut viitekehykset ovat erityisesti testausautomaation yhteydessä käytettyjä viitekehyksiä ja niiden vahvuutena on useiden eri testitapausten läpikäynti ja tämän tehokkuus. Viitekehyksissä on komponenteiltaan paljon yhtäläisyyksiä, mutta esimerkiksi viitekehysten hyödyntämiseen tarvittavissa tietotaidoissa ja työnjaossa on eroja. Avainsanaohjatun viitekehyksen mukaan tehdyt testit vaikuttavat aineisto-ohjatun viitekehyksen mukaan tehdyiltä testeiltä, joissa testin skriptissä suoritettavat toiminnot ovat jaettuna vielä erikseen omaan avainsanojen tiedostoon. Tämä tuo lisää haastavuutta ja työtä testien toteuttamiseen ja ylläpitoon, mutta avainsanaohjatulle viitekehykselläkin löytyy käyttökohteita esimerkiksi suuremmissa ohjelmistoprojekteissa. Koska viitekehysten mukaan toteutettu testaus vaatii keskivertoa enemmän työtä ja aikaa, niiden käyttöönottoa on harkittava ennen päätöksen tekoa. Tarkoituksena tutkielmassa olikin luoda selkeämpää kuvaa viitekehysten ominaisuuksista ja asioista, joita ottaa päätöksenteossa huomioon.

Aineisto- ja avainsanaohjatut viitekehykset soveltuvat esimerkiksi web-sovellusten toimintojen testaamiseen. On mielenkiintoista nähdä kuinka tulevaisuudessa ohjelmistotestauksen viitekehykset kehittyvät ja miten erilaisia ohjelmistojen tyylejä on otettu huomioon viitekehyksiä luotaessa ja kehitettäessä. Teknologian ja siinä samalla ohjelmistojen kehittyessä myös testauksen on pystyttävä vastaamaan uusiin vaatimuksiin, joten muutosta ja kehitystä on odotettavissa.

Lähdeluettelo

ACM. (2021). *ACM Digital Library. https://dl.acm.org/* (Haettu 11.12.2021)

Andor. (2021). *Ex Libris Discovery*. *https://andor.tuni.fi/discovery/search?vid=358FIN_TAMPO:VU1&lang=fi* (Haettu 11.12.2021)

Bose, L. & Thakur, S. (2014). GRAFT: Generic & Reusable Automation Framework for agile testing. *5th International Conference – Confluence the Next Generation Technology Summit (Confluence)*. 761–766. *https://doi.org/10.1109/CONFLUENCE.2014.6949235*

Cocchiaro, C. (2018). *Selenium Framework Design in Data-Driven Testing* (1. p.). Packt Publishing.

Garousi, V. & Pfahl, D. (2016). When to automate software testing? A decision-support approach based on process simulation. *Journal of Software: Evolution and Process, 28*(4), 272–285. *https://doi.org/10.1002/smr.1758*

Garousi, V. & Elberzhager, F. (2017). Test Automation: Not Just for Test Execution. *IEEE Software, 34*(2), 90–96. *https://doi.org/10.1109/MS.2017.34*

Google Scholar. (2021). *Google Scholar. https://scholar.google.com/* (Haettu 11.12.2021)

Gupta, R. & Bajpai, N. (2014). A Keyword-Driven Tool for Testing Web Applications (KeyDriver). *IEEE Potentials, 33*(5), 35–42. *https://doi.org/10.1109/MPOT.2012.2202135*

Haikala, I. & Mikkonen, T. (2011). *Ohjelmistotuotannon käytännöt* (12., uudistettu painos.). Talentum.

IEEE Xplore. (2021). *IEEE Xplore. https://ieeexplore.ieee.org/Xplore/home.jsp* (Haettu 11.12.2021)

Kasurinen, J. P. (2013). *Ohjelmistotestauksen käsikirja* (1. p.). Docendo.

Kent, J. (2007). Test Automation: From Record/Playback to Frameworks. *EuroSTAR*. 2007, Stockholm, Sweden.

Laukkanen, P. (2006). *Data-Driven and Keyword-Driven Test Automation Frameworks.* [diplomityö, Teknillinen korkeakoulu] *http://eliga.fi/Thesis-Pekka-Laukkanen.pdf*

Pajunen, T., Takala, T. & Katara, M. (2011). Model-Based Testing with a General Purpose Keyword-Driven Test Automation Framework. *IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. 242–251. *https://doi.org/10.1109/ICSTW.2011.39*

Polo, M., Reales, P., Piattini, M. & Ebert, C. (2013). Test Automation. *IEEE Software, 30*(1), 84–89. *https://doi.org/10.1109/MS.2013.15*

ProQuest. (2021). *Basic Search – Computer Science Database – ProQuest*. *https://www.proquest.com/compscijour* (Haettu 11.12.2021)

Selenium. (2021). *Selenium*. *https://www.selenium.dev/* (Haettu 11.12.2021)

Takala, T., Maunumaa, M. & Katara, M. (2009). An adapter Framework for Keyword-Driven Testing. *2009 Ninth International Conference on Quality Software*. 201–210. *https://doi.org/10.1109/QSIC.2009.35*

Thummalapenta, S., Sinha, S., Singhania, N. & Chandra, S. (2012). Automating Test Automation. *2012 34th International Conference on Software Engineering*. 881–891. *https://doi.org/10.1109/ICSE.2012.6227131*

Umar, M. & Zhanfang, C. (2019). A Study of Automated Software Testing: Automation Tools and Frameworks. *International Journal of Computer Science Engineering, 8*(6), 217–225. *http://www.ijcse.net/docs/IJCSE19-08-06-011.pdf*

UTF. (2021). *Automate Functional Testing from UI to the API | UTF One | Micro Focus*. *https://www.microfocus.com/en-us/products/uft-one/overview* (Haettu 11.12.2021)

Wang, X. & He, G. (2014). The research of data-driven testing based on QTP. *9th International Conference on Computer Science & Education*. 1063–1066. *https://doi.org/10.1109/ICCSE.2014.6926625*