**Jussi Loppukaarre**

# Web-sovellusten testaaminen

Tietotekniikan kandidaatintutkielma

19. toukokuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

**Tekijä:** Jussi Loppukaarre

**Yhteystiedot:** loppujjy@student.jyu.fi

**Ohjaaja:** Tuomo Rossi

**Työn nimi:** Web-sovellusten testaaminen

**Title in English:** Web application testing

**Työ:** Kandidaatintutkielma

**Sivumäärä:** 21+0

**Tiivistelmä:** Tässä tutkielmassa perehdytään, kuinka IT-alan kirjallisuudessa ja tutkimuksissa web-sovelluksien funktionaalisten piirteiden testaamista on käsitelty. Tavoitteena on tuoda esille miten web-sovelluksien testaaminen eroaa ohjelmistotestaamisesta. Tämän lisäksi tarkastellaan myös havaittuja haasteita web-sovelluksia testatessa. Tutkielman lopussa lukijalla on käsitys kuinka web-sovelluksien testaaminen eroaa muusta sovellustestaamisesta ja mistä nämä erot johtuvat.

**Avainsanat:** Web-sovellus, testaaminen, ohjelmistotestaus

**Abstract:** This study explores how the literature and research in the IT field has addressed the testing of functional aspects of web applications. The objective of the study is to highlight how testing of web applications differs from software testing. In addition to this, we also take a look at the challenges identified when testing web applications. At the end of the dissertation, the reader has an understanding of how web application testing differs from other application testing and where these differences come from.

**Keywords:** Web application, testing, software testing

# Kuviot

# Sisällys

# 1 Johdanto

Uudelle vuosituhannelle siirryttyä saatavilla olevien web-sovellusten määrä on kasvanut voimakkaasti. Web-sovelluksia hyödynnetään niin arkielämässä kuin työelämässä tänä päivänä niin huomattavan paljon, että näistä on muodostunut melkein pakollinen osa nyky-yhteiskun nan rakennetta. Täten on tärkeää, että käytetty web-sovellus täyttää halutut toiminnallisuuskriteerit. Valitettavan yleisesti markkinapaine ja lyhyt markkinointiaika (time-to-market) johtavat web-sovellusten testetaamisen laiminlyöntiin (Di Lucca ja Fasolino 2006a, s. 219). Tämän vuoksi tutkielman aiheeksi päädyttiin valitsemaan web-sovellusten testaamisen.

Koska testaaminen on yleisesti hyvin laaja prosessi ja käsittää useita osia, tutkielmassa päädyttiin keskittymään web-sovelluksien tuotanto- tai kehitysvaiheen aikana tapahtuvaan funktionaalisten piirteiden testaamiseen. Vaikka ohjelmistotestaaminen ja web-sovellus testaaminen jakavat yhteisiä käytänteitä, ne eroavat kuitenkin huomattavasti. Tutkielman päätarkoituksena on tarkastella kuinka web-sovellusten testaaminen eroaa muusta ohjelmistotestaamisesta ja mitä eroja voidaan huomioida. Tutkielmassa web-sovellus käsitetään selainpohjaisena sovelluksena, joka voi muodostua yhdestä tai useammasta sivusta.

Web-teknologioiden nopea kehitys on jättänyt monia aukkoja ja yksi näistä käsittää websovellusten testaamisen. Täten on kannattavaa huomioida näitä aukkoja niin käytännön kuin tieteen kannalta. Tavoitteena on tuoda esiin havaittuja aukkoja, jotta näitä voidaan korjata ja tutkia tarkemmin tulevaisuudessa.

Tutkielman tutkimustrategia on toteutettu kirjallisuuskartoituksena. Esiin on nostettu yhteisiä testauskäytänteitä sekä web-sovelluksille ominaisia piirteitä. Tällä tiedolla on vastattu kysymyksiin:

1. Mitä testaaminen on?
2. Miten web-sovellukset vaikuttavat testauskäytäntöihin?
3. Minkälaisia haasteita web-sovellukset asettavat testaamiselle?

Tutkielmassa aineistona on käytetty IT-alan kirjallisuudessa ja tieteellisissä artikkeleissa esitettyjä käytänteitä sekä havaintoja. Aineiston kartoituksessa on hyödynnetty Google Scho-

lar -pavelua, IEEE Explore -kirjastoa ja Jyväskylän Yliopiston kirjaston JYKDOK palvelua. Teoksien tieteellinen oikeellisuus on tarkistettu Julkaisufoorumi-sivustoa käyttäen. Erityisesti tutkielmassa on hyödynnetty Di Luccan ja Fasolinon tuotoksia viimeisen kahdenkymmenen vuoden ajalta, sillä he ovat tutkineet web-sovellusten testaamista huomattavissa määrin.

Luvussa kaksi vastataan kysymyksiin mitä testaaminen on, miksi sovelluksia testataan sekä kuinka web-sovellusten testaaminen eroaa muusta ohjelmistotestaamisesta. Luvussa kolme käsitellään testausta kehitysvaiheessa kertomalla yleisistä testausmetodeista tai -strategioista ja testaustasoista. Luvussa neljä on koottu yhteen kirjallisuuden pohjalta havaittuja haasteita mitä web-sovelluksien testaaminen sisältää.

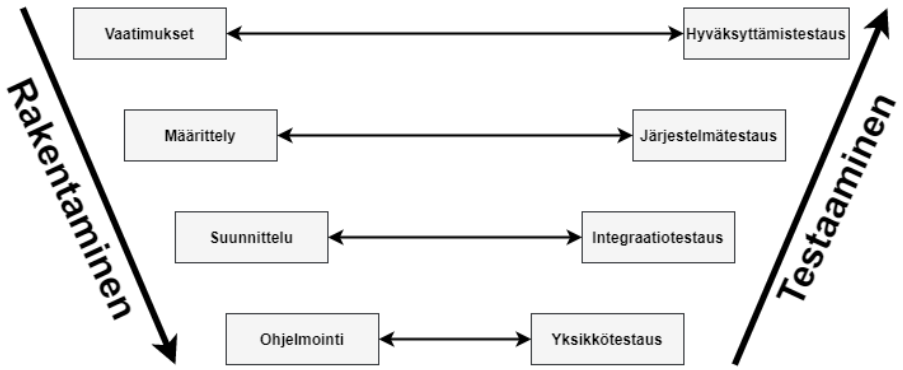
# 2 Testaaminen

Luvussa kaksi käsitellään testaamista yleisellä tasolla muutamalla kysymyksellä. Mitä testaaminen on ja miksi sovelluksia testataan? Näiden kysymysten jälkeen vastataan miten websovellus testaus eroaa tavanomaisesta sovellustestaamisesta.

## 2.1 Mitä testaaminen on?

Ohjelmistoja tuotetaan sekä päivitetään nykypäivänä nopeata tahtia eri tarkoituksiin. Onkin hyvin yleistä, että tuotantoprosessin aikana on syntynyt virheitä. Näille voi olla monia eri tekijöitä ja testaamisilla näitä virheitä pyritään kartoittamaan. Kasurinen (2013, s. 8) kertoo testaamisen olevan työtä, jolla varmistetaan tuotoksen olevan suunnitelman mukainen ja ominaisuuksien toimivan halutulla tavalla.

Testaaminen voidaan käsittää yhtenä prosessina projektin kokonaisuudessa. Tästä esimerkkinä voidaan pitää vesiputousmallia, jossa projektissa edetään vaihe vaiheelta. Kasurinen (2013, s. 10) pitää tätä mallia vanhanaikaisena useasta eri syystä, kuten testauksen painotuksesta vain yhteen vaiheeseen. Tämä voi asettaa mahdollisia haasteita jos testatessa havaitaan virheitä, jotka olisi mahdollisesti voitu poistaa aiemmassa vaiheessa.




Kuvio 1. V-malli ohjelmistokehityksestä

Kasurinen (2013, s. 9) toteaa, että testaamista tulisi käsitellä jatkuvana prosessina. Hän antaa v-mallin vaihtoehtoisena näkemyksenä projektin toteutukselle. Kuten kuvio 1 havainnollistaa sen sijaan, että testaaminen käsiteltäisiin vain yhtenä prosessina projektissa, jokaiselle ra-

kentamisprosessille on vastaava testausprosessi. Näitä testausprosesseja avataan tarkemmin luvussa kolme.

## 2.2 Miksi sovelluksia testataan?

Kuten aiemmin on kerrottu, on yleistä, että projektin aikana tuotokseen on syntynyt virheitä. Koska tarkoituksena on luoda laadullinen ohjelmisto, sen halutaan toimivan suunnitelman mukaisesti. Tämä voi kuitenkin antaa kapean kuvan siitä miksi sovelluksia testataan.

Esimerkiksi jos lähtökohtana on varmistaa, että tuotos ei sisällä virheitä niin Myersin (2012, s. 6) mukaan tämänkaltainen käsitys antaa vääristetyn kuvan testaamisesta. Hän kertoo, että testaamisen tarkoituksena on luoda tuotokselle lisäarvoa. Jos yllä mainittu esimerkki pätee, niin tällöin tuotokselle on luotu todennäköisesti vähän lisäarvoa. Myers (2012, s. 6) kertoo, että ohjelmistoa ei pitäisi testata näyttääkseen sen toimivan vaan oletuksella, että ohjelmisto sisältää virheitä.

Testauksen tarkoituksena kuitenkaan ei ole pelkästään varmistaa tuotoksen oikeellisuus ja luoda tälle lisäarvoa, vaan myös taloudellinen kannattavuus. Kasurinen (2013, s. 9) kertoo, että yritykset, jotka testaavat tuotostaan huolellisemmin saavat myös suuremman katteen. Hän jatkaa myös kertomalla, että virheen korjaaminen suunnitellessa tuotosta maksaa yritykselle 1-2 prosenttia siitä, mitä se maksaisi julkaisun jälkeen. Täten jos testaaminen on puutteellista se voi aiheuttaa yhtiölle pienemmän katteen tai mahdollisia sanktioita.

## 2.3 Web-sovellusten testaaminen

Web-sovelluksista on kehittynyt hyvin tehokas tapa hallinnoida eri järjestelmiä ja muodostunut elintärkeä nyky-yhteiskunnan kannalta. Di Lucca ja Fasolino (2006a, s. 219) kertovat, että kysyntä on nousussa web-sovelluksille, jotka täyttävät turvallisuus, skaalattavuus, luotettavuus ja saavutettavuus vaatimukset.

Usein web-sovelluksia testatessa voidaan hyödyntää samoja menetelemiä kuin muita sovelluksia testattaessa. Web-sovelluksien testaaminen ei ole kuitenkaan täysin samanlainen prosessi ja vastaan usein tulee monipuolisia haasteita. Doğan ym. (2014, s. 174-175) ja Faso-

lino ym. (2013, s. 35) kokoavat teoksissaan useita esimerkkejä havaituista haasteista. Nämä voidaan tiivistää

- asiakas-palvelin tai monitasoisesta arkkitehtuurista,
- heterogeenisestä suoritusympäristöstä,
- heterogeenisestä luoteesta kuten ohjelmointikielistä,
- dyynaamisesta luonteesta,

syntyviin haasteisiin.

Näitä haasteita on voitu lähestyä usealla menetelmällä. Yksi menetelmä on Di Luccan ja Fasolinon (2006b, s. 1173) esittämä tapa testata web-sovellusten ei-fuktionaalisia (non-functional) ja fuktionaalisia (functional) vaatimuksia eri perspektiiveistä. Testattavia ei-funktionaalisia ominaisuuksia voivat olla esimerkiksi suorituskyky, tietoturva ja käytettävyys, joille asiakas on voinut asettaa täytettäviä kriteerejä. Tutkielmassa ei keskitytä tämän enempää ei-funktionaalisen testaamisen käsittelyyn.

Funktionaalinen tai toiminnalinen testaaminen on usein ensimmäinen asia mikä tulee mieleen, kun testaamisesta puhutaan. Di Lucca ja Fasolino (2006b, s. 1173) kertovat, että funktinaalisen testaamisen tarkoituksena on löytää sovelluksen rakentamisen aikana syntyneitä virheitä. Näitä ovat esimerkiksi virheet koodissa, jotka eivät toteuta haluttua toimintoa oikein tai turvallisesti. Web-sovellus testaamisen kohdalla voidaan kuitenkin hyödyntää samoja käytänteitä kuin muun ohjelmistotestaamisen yhteydessä.

# 3 Testaaminen kehitysvaiheessa

Kuten aiemmassa luvussa todettiin, testaaminen voidaan jakaa funtionaaliseen ja ei-funktionaaliseen testaamiseen. Luku kolme käsitellään funktionaalista testaamista tarkemmin ja rajaa näkökulman kehitysvaiheeseen. Esiin tuodaan kolme yleisintä testausmetodia sekä testaustasot, jotka suoritetaan sovelluksen kehitysvaiheessa.

## 3.1 Yleiset testausmetodit

Kolme yleistä testausmetodia tai -strategiaa ovat mustalaatikko-, lasilaatikko ja harmaalaatikkotestaaminen. Näitä metodeja on tarkoituksena avata ensin ennenkuin aihe siirtyy testaustasoihin, joissa hyödynnetään kyseisiä metodeja.

### 3.1.1 Mustalaatikkotestaaminen


Kuvio 2. Mustalaatikkotestaaminen

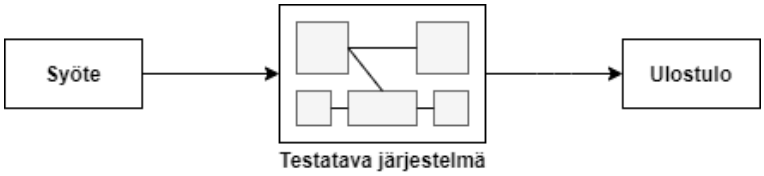
Mustalaatikkotestaaminen (black box testing) on yleisesti hyvin suosittu funktionaalisten ominaisuuksien testausmetodi. Tätä metodia usein on hyödynnetty, kun on haluttu testata järjestelmän tai ohjelman käyttäytymistä. Kuten kuvio 2 havainnollistaa, testaaja asettaa ohjelmistoon syötteitä, joista tuotettuja ulostuloja verrataan odotettaviin tuloksiin ja itse syötteitä käsittelevä ohjelmisto tai koodi ei ole huomioitavana kohteena. Myers (2012, s. 9) kertoo, että tällöin on keskitytty löytämään virheitä, joita on syntynyt teknisten tietojen pohjalta luoduista syötteistä.

Mustalaatikkotestaamisen valitseminen testausmetodiksi ohjelmistolle on hyvin tehokas ja saatavat hyödyt ovat monipuolisia. Khan (2011, s. 32) on esittänyt näitä hyötyjä olevan esimerkiksi:

- puolueeton näkemys, kun testaaja ja suunnittelija ovat riippumattomia toisistaan
- testaus on suoritettu käyttäjän näkökulmasta

- testitapaukset on helppo luoda käyttämällä sovellusta kuin loppukäyttäjä
- testaajan ei tarvitse tuntea ohjelmointikieliä
- sallii suuren koodialuuen testaamisen
- nopea testitapausten suoritus

### 3.1.2 Lasilaatikkotestaaminen




Kuvio 3. Lasilaatikkotestaaminen

Lasilaatikko- tai valkolaatikkotestaaminen (white box testing) on mustalaatikkotestaamisen vastakohta. Kun mustalaatikkotestaamisella on haluttu testata sovelluksen käyttäytymistä syötteisiin, niin lasilaatikkotestaamisella on pyritty tutkimaan sovelluksen rakennetta ja toiminnalisuutta. Ricca ja Tonella (2001, s. 25) huomauttavat, että mustalaatikkotestaamisen tuloksia voidaan täten täydentää lasilaatikkotestaamisen avulla. Kuten kuvio 3 havainnoi, lasilaatikkotestaamisessa testaaja kykenee näkemään kuinka mikäkin komponentti reagoi annettuun syötteeseen. Tällöin on tärkeää, että testaaja ymmärtää koodin täydellisesti (Ehmer ja Khan 2012, s. 12). Ilman tätä tietämystä on haastellista hahmottaa mikä komponentti aiheuttaa virheellisen ulostulon.

### 3.1.3 Harmaalaatikkotestaaminen

Harmaalaatikkotestaaminen (grey box testing) on yhdistänyt mustalaatikko- ja lasilaatikkotestaamisen hyödylliset osa-alueet. Tällöin testaamisessa huomoidaan sovelluksen käyttäytymisen loppukäyttäjän näkökulmasta sekä rakenteen vaikutuksen (Di Lucca ja Fasolino 2006a, s. 227). Yleisesti harmaalaatikkotestaamista hyödynnetään, kun kaikkia sovelluksen komponentteja ei voida käsitellä. Kasurinen (2013, s. 43) antaa kirjassa *Ohjelmistotestauksen käsikirja* tästä esimerkin, jossa oma järjestelmä voidaan testata lasilaatikkona, mutta järjestelmään yhdistettyä toisen osapuolen ohjelmiston rajapintaa voidaan käsitellä vain mustalaatikkona.

## 3.2 Testaustasot

Kehitysvaiheessa testaaminen on yleisesti jaettu yksikkö-, integraatio- ja järjestelmätestaamiseen. Tässä alaluvussa avataan, mitä nämä testaustasot ovat ja kuinka niitä käsitellään web-sovelluksien yhteydessä. Myös regressiotestaamisen käsitettä avataan lyhyesti.

### 3.2.1 Yksikkötestaaminen

Yksikkötestaaminen (unit testing) on ensimmäinen tapa testata tuotettu osuus koodista. Kuten luvussa 2 annetussa v-mallissa esitettiin, yksikkötestaamisella pyritään varmistamaan ohjelmoinin myötä syntyneen sovelluksen oikeellisuus. Tällöin testataan yhtä pientä sovelluksen osaa, kuten oliota tai funktiota. Kasurisen (2013, s. 37) mukaan ohjelmoija varmistaa, että testattava funktio tai olio voidaan suorittaa ja se kykenee käsittelemään annettuja syötteitä oikein.

Web-sovellusten yksikkötestaaminen ei ole yhtä yksinkertaista, kuin muun ohjelmistotestaamisen kohdalla. Vaikka mahdollisuutena on testata jokainen olio ja funktio, jotka muodostavat web-sovelluksen, tämä ei kuitenkaan saata olla mielekästä. Di Lucca ja Fasolino (2006a, s. 230) mukaan testattava yksikkö voi olla web-sivu, jota voidaan käsitellä erikseen asiakas- ja palvelinsivuina.

Asiakassivulla tarkoitetaan sovelluksen käyttöliittymää. Tällöin testauksen kohteena ovat sen eri komponentit, kuten lomakkeet, kuvat tai renderöity rakenne. Di Luccan ja Fasolinon (2006a, s. 231) mukaan asiakassivujen testaamisella pyritään varmistamaan sivun sisältävän käyttäjän määritellemän ja odetetun sisällön, hyperlinkkien osoittavan oikeaan kohdesivuun, linkit sivuille, joita ei ole vielä olemassa, toiminnot, kuten painikkeiden ja lomakaiden oikeellisuus sekä visuaalisen puolen oikeellinen renderöinti. He jatkavat kertomalla, että näitä voidaan käsitellä musta-, lasi- ja harmaalaatikko metodeilla. Esimerkiksi sivulla oleva lomake voidaan testata ensin mustalaatikkona ja jos syöteestä saadaan virheitä, voidaan testamista tarkentaa lasi- tai harmaalaatikkona.

Palvelinsivu sen sijaan käsittelee käyttöliittymän "takana"toimivan logiikan eli backendin. Di Luccan ja Fasolinon (2006a, s. 231) mukaan tällöin yksikkötestaaminen keskittyy havaitsemaan esimerkiksi rajapintojen, kuten Java Servlet tai COTS suorittamisessa, virheellisesti

suoritetun datan tallentamisesta tietokantaan, virheellisten linkkien ja dynaamisesti luotujen sivujen virheitä.

### 3.2.2 Integraatiotestaaminen

Integraatiotestaaminen (integration testing) on yksikkötestaamista seuraava testaustaso. Siinä missä yksikkötestaaminen on keskittynyt funktion, olion tai web-sivun oikeellisuuden varmistamiseen, Kasurisen (2013, s. 39) mukaan integraatiotestaamisella pyritään varmistamaan, että luodut funktiot voidaan kytkeä yhteen yksi kerrallaan ja nämä kykenevät kommunikoimaan halutulla tavalla.

Web-sovellusten yhteydessä integraatiotestaminen voidaan käsittää suurinpiirtein samalla tavalla, mutta yksittäisien funktioiden yhdistämisen sijaan yhteen on kytketty web-sivuja. Tällöin on voitu tukeutua suunnitteludokumentteihin, jotta integroitavat web-sivut kyetään määrittelemään (Di Lucca ja Fasolino 2006b, s. 1177). Integroitavat web-sivut ovat siis yhteydessä toisiinsa jollain loogisella tavalla. Di Lucca ja Fasolino (2006b, s. 1177) kertovat näiden yhteyksien voivan olla esimerkiksi hyperlinkit, uudelleenohjaus riippuvuussuhteet tai rakenteelliset suhteet.

Integraatiotestaamisen tarkoituksena on siis tutkia kuinka yhteen kytketyn ohjelmiston rakenne ja käyttäytyminen toimivat. Tetausmetodeina voidaan hyödyntää niin musta- kuin lasilaatikkotestaamista. Esimerkiksi jos web-sovellus vaatii profiiliin kirjautumista voidaan testata asiakassivua mustalaatikkona ja tähän integroitua palvelinsivua lasilaatikkona. Di Lucca ja Fasolino (2006b, s. 1178) huomauttavat, että integraatiotestaamiselle harmaalatikkotestaaminen olisi kuitenkin soveliaampi metodi, koska tämä käsittää niin rakenteen kuin käytöksen tarkastelun.

### 3.2.3 Järjestelmätestaaminen

Yksikkö- ja integraatiotestaamisella voi varmistaa yksittäisen komponentin laatu ja näiden kyky kommunikoida, mutta nämä eivät kata koko järjestelmän toiminnallisuutta (Baresi ja Pezzè 2006, s. 107). Järjestelmätestaamisen tarkoituksena onkin varmistaa, että tuotettu ohjelmisto kokonaisuudessan vastaa haluttuja toiminnallisuuksia ja laatua.

Web-sovelluksien yhteydessä käsite ei muutu suuresti. Kokonaisuudessa tällöin testataan kaikki web-sivut ja pyritään löytämään virheitä toimivasta kokonaisuudesta. Di Lucca ja Fasolino (2006a, s. 233) mukaan tyypillinen järjestelmätestaaminen koostuu käyttäjän toimintojen, asiakas- ja palvelinsivujen sekä linkkien kattavuus kriteerien täyttämisestä.

### 3.2.4 Regressiotestaaminen

Yleisesti regressiotestaaminen ei ole testaustaso. Tämä on pikemminkin käytänne, jota voi hyödyntää kaikilla edellämainituilla testaustasoilla. Tarkoituksena on varmistaa, että valmiiseen komponenttiin tai ohjelmistoon tuotettu muutos ei aiheuta taantumaa edelliseen versioon verrattuna. Oletukena on, että ennalta korjatut virheet eivät esiinny ja uusia virheitä ei muutoksen jälkeen synny. (Kasurinen 2013, s. 43). Esimerkiksi jos on päädytty muuttamaan komponenttien välistä kytkentää, halutaan varmistaa, että korjattuja virheitä ei synny uudessa kytkennässä.

# 4 Web-sovellus testauksen haasteet

Luku neljä käsitellee kirjallisuuskartoituksen pohjalta havaittuja haasteita, joita on kohdattu web-sovelluksia testatessa. Ensin perehdytään testausmetodien ominaisiin haasteisiin ja tämän jälkeen testaustasoissa havaittuihin haasteisiin.

## 4.1 Testausmetodien haasteet

Web-sovellus testaamisen kohdalla mustalaatikkometodia voi hyödyntää melkein samalla periaattella kuin muun ohjelmistotestaamisen. Kuitenkin Di Luccan ja Fasolinon (2006a, s. 226, 237-238) mukaan dynaamisesti luotujen komponenttien testaaminen voi olla kallista, koska komponentin olosuhteita on vaikea hahmottaa ja toistaa. Toisena ongelmana he havaitsevat olevan sopivan testausmallin valinta, jotta sovelluksen käyttätyminen ja testitapaukset voidaan määrittää.

Lasilaatikkometodia täytyy myös tarkentaa web-sovelluksien kohdalla. Web-sovelluksien monimutkaisemman arkkitehtuurin ja rakenteen takia kattavuuskriteereille täytyy määritellä sopivat mallit kuvaamaan rakenteellista tietoa eri tasoilla (Di Lucca ja Fasolino 2006a, s. 226-227).

Harmaalaatikkometodin haasteet ovat hyvin samanlaisia web-sovellusten ja muiden sovellusten tapauksessa. Ehmerin ja Khanin (2012, s. 14) mukaan yleisiä haasteita ovat testien rajautuminen saatavissa olevaan koodiin, virheiden havaitseminen hajautetusta ohjelmistosta, monet väylät voivat jäädä testaamattomiksi sekä testitapaukset voivat olla tarpeettomia jos testaaja ei ole tietoinen ennestään suoritetuista testeistä.

Yleisesti testausmetodien heikkoudet sekä web-sovelluksien monimutkainen rakenne ja luonne lisäävät haasteita. Testausmetodeja täytyisi tällöin tarkentaa, että testattavat kriteerit täyttyisivät.

## 4.2 Testaustasojen haasteet

Vuosituhannen alussa web-sovellusten yksikkötestaaminen oli haasteellista, koska testattava yksikkö oli vaikea rajata. Di Lucca ym. (2002, s. 331) määrittelivät, että web-sivu tai sen muodostavat komponentit voidaan käsittää testattavana yksikkönä. Tämä oli yksinkertainen ja tehokas ratkaisu, mutta web-sovelluksia hyödyntävien teknologioiden edetessä uusia haasteita on kartoitettu.

Varsinkin dynaamisesti rakennetuista asiakasivuista on syntynyt haasteista. Di Lucca ja Fasolino (2006b, s. 1177) kertovat perusongelman olevan dynaamisten web-sivujen saatavuus, joiden tuottaminen vaatii kykyä tunnistaa ja toistaa käyttäjän syötteestä sekä sovelluksen tilasta rakennettu sivu. Toisena haasteena voidaan nähdä heidän havaitsema "tilan rähdysongelma". Tällöin dynaamisia web-sivuja voidaan luoda huomattava määrä riippuen mahdollisten syötteiden sekä sovelluksen tilan kombinaatiosta.

Web-sovellusten yksikkö- ja integraatiotestaaminen jakavat yhden erityisen haasteen, tynkien käyttö. Nämä ovat komponentteja, jotta testattava osuus kyetään ajamaan testausympäristössä. Di Lucca ja Fasolino (2006a, s. 246) mukaan asiakas- ja palvelinsivuille täytyy luoda sopivat tyngät sekä huomioida kuinka nämä toteutetaan dynaamisesti luoduille websivuille.

Integraatiotestaamiselle on web-sovelluksien kuin muiden sovelluksien kohdalla yhteinen haaste, järjestys missä komponentit integroidaan. Di Lucca ym. (2002, s. 314) mukaan voi olla hankalaa määritellä strategia järjestyksen toteuttamiselle, jos web-sivujen yhdistämisestä syntyy syklinen polku. He esittävät nelivaiheisen strategian, jonka avulla järjestys voidaan asettaa ja integraatio voidaan toteuttaa:

1. luodaan kaavio, missä solmut kuvaavat web-sivuja ja linkit näiden yhteyksiä
2. jokaiselle linkille annetaan painoarvo, joka koostetaan linkin kautta annettujen parametrien määrästä
3. luodaan asyklinen kaavio poistamalla linkit joiden painoarvo on pieni
4. lasketaan asyklisesta kaaviosta topologinen järjestys ja toteutetaan web-sivujen integraatio tämän mukaan

Järjestelmätestaamisen yhteydessä yleisesti hyödynnetään mustalaatikkotestaamista. Donley ja Offutt (2009, s. 18) huomauttavat, että monet testausrakenteet korvaavat yksikkötestaamisen järjestelmätestaamisella hyödyntäen edellämainittua metodia. Tällöin tulisi huomioida Ehmerin ja Khanin (2012, s. 18) havaitsemia mustalaatikkometodin heikkouksia, kuten toteutettavien testien rajallisuus ja testitapauksien luominen epäselvistä määritelmistä.

Kuten testausmetodien kohdalla niin myös testaustasojen yhteydessä web-sivujen monimutkainen rakenne ja luonne luovat haasteita. Tällöin erityisesti web-sivujen rakennetta ja liitäntöjä täytyy huomioida.

# 5 Yhteenveto

Tutkielmassa on annettu käsitys testaamisesta yleisellä tasolla. Tutkielmassa olen tuonut esille, mitä testaaminen on? Miten web-sovellukset vaikuttavat testauskäytäntöihin? Minkälaisia haasteita web-sovellukset asettavat testaamiselle?

Luvussa kaksi pyrkimyksenä oli vastata ensimmäiseen kysymykseen, mitä testaaminen on? Kasurinen (2013, s. 8-9) esitti teoksessa *Ohjelmistotestauksen käsikirja*, jonka mukaan testaaminen voidaan käsittää jatkuvana työnä varmistaa, että tuotos vastaa suunnitelmia ja odotuksia. Myersin (2012, s. 6) näkemys teoksessa *The art of software testing* oli, että testaamisella pyritään luomaan tuotokselle lisäarvoa ja todistamaan tuotoksen sisältävän virheitä. Molemmat näkemykset ovat valideja ja nämä olisi hyödyllistä yhdistää. Testaamisella täten pyrittäisiin varmistamaan suunnitelman mukainen toteutus, mutta myös huomioida, että tuotos ei ole täydellinen.

Luvussa kolme esiteltiin yleisiä kehitysvaiheessa hyödynnettyjä metodeja ja huomioitavia piirteitä testaustasoissa. Testausmetodeja on kyetty käyttämään näiden periaatteiden mukaan melkein samalla tavalla niin web-sovelluksien kuin muiden sovelluksien testaamisen yhteydessä. Sen sijaan varsinainen ero havaittiin testaustasojen yhteydessä. Erona havaittiin Di Luccan ja Fasolinon (2006a, s. 230) esittämä tapa yksikkö- ja integraatiotestaamiselle, jossa testaaminen suoritetaan asiakas- ja palvelinsivuille yksittäisen funktion tai olion sijasta. Kyseinen tapa voi olla hyödyllinen, koska web-sivut yleensä muodostuvat useista komponenteista ja näiden kaikkien testaaminen tavallisten käytäntöjen mukaan ei saata olla mielekästä.

Web-sovelluksien testaaminen käsittää useita haasteita, joita oli tarkoitus avata luvussa neljä. Testausmetodien kohdalla haasteita voivat aiheuttaa web-sovelluksien dynaamiset komponentit tai rakenne. Di Luccan ja Fasolinon (2006a, s. 226) mukaan perinteisiä metodeja täytyisi sopeuttaa näihin erityisiin piirteisiin. Testaustasojen tapauksessa dynaamisuuden havaittiin aiheuttavan myös selviä haasteita. Integraatiotestaamisen kohdalla huomattavana haasteena havaittiin integraatiojärjestys, johon Di Lucca ym. (2002, s. 314) esittivät nelivaiheisen strategian, jolla järjestys ongelma voidaan ratkaista. Funktionaalisia piirteitä järjestelmätestatessa haasteena havaittiin yleisesti hyödynnetyn mustalaatikkometodin heikkoudet.

Tutkielman tavoite selvän yleiskuvan antamiseksi web-sovellusten funktionaalisten piirteiden testaamisen eroista täytti tekijän asettaman tavoitteen. Kuitenkin tutkielmassa on selviä merkkejä kuinka sitä olisi mahdollista parantaa. Hyödynnetty materiaali ei ole täysin ajankohtaista, mutta toimivat silti validina lähteenä. Myös rajaaminen vain funktionaalisten piirteiden tarkasteluun jättää huomattavan osan testaamisesta käsittelemättä. Mahdollisena jatkotutkimuksena voidaan ottaa huomioon myös ei-funktionaalisia piirteiden rooli. Toisena mahdollisuutena olisi myös vertailla Di Luccan ym. (2002, s. 314) esittämää strategiaa tämän hetkisten integraatio käytänteiden mukaan.

# Lähteet


Baresi, Luciano, ja Mauro Pezzè. 2006. “An Introduction to Software Testing”. *Electronic Notes in Theoretical Computer Science* 148 (1): 89–111. https://doi.org/https://doi.org/10.1016/j.entcs.2005.12.014.

Di Lucca, Giuseppe, ja Anna Fasolino. 2006a. “Web Application Testing”. Teoksessa *Web Engineering,* 219–260. Berlin: Springer. https://doi.org/10.1007/3-540-28218-1_7.

Di Lucca, Giuseppe A., Anna R. Fasolino, F. Faralli ja U. De Carlini. 2002. “Testing Web applications”. Teoksessa *International Conference on Software Maintenance, 2002. Proceedings.* 310–319. https://doi.org/10.1109/ICSM.2002.1167787.

Di Lucca, Giuseppe A., ja Anna Rita Fasolino. 2006b. “Testing Web-based applications: The state of the art and future trends”. *Information and Software Technology* 48 (12): 1172–1186. https://doi.org/https://doi.org/10.1016/j.infsof.2006.06.006.

Doğan, Serdar, Aysu Betin-Can ja Vahid Garousi. 2014. “Web application testing: A systematic literature review”. *Journal of Systems and Software* 91:174–201. https://doi.org/https://doi.org/10.1016/j.jss.2014.01.010.

Donley, Blaine, ja Jeff Offutt. 2009. “Web Application Testing Challenges”. *Software Engineering, George Mason University.*

Ehmer, Mohd, ja Farmeena Khan. 2012. “A Comparative Study of White Box, Black Box and Grey Box Testing Techniques”. *International Journal of Advanced Computer Science and Applications* 3 (kesäkuu): 12–15. https://doi.org/10.14569/IJACSA.2012.030603.

Fasolino, Anna Rita, Domenico Amalfitano ja Porfirio Tramontana. 2013. “Web application testing in fifteen years of WSE”. Teoksessa *2013 15th IEEE International Symposium on Web Systems Evolution (WSE),* 35–38. https://doi.org/10.1109/WSE.2013.6642414.

Kasurinen, Jussi Pekka. 2013. *Ohjelmistotestauksen käsikirja.* 1. p. Jyväskylä: Docendo.

Khan, Mohd. 2011. “Different Approaches To Black box Testing Technique For Finding Errors”. *International Journal of Software Engineering  Applications* 2, numero 4 (lokakuu): 31–40. https://doi.org/10.5121/ijsea.2011.2404.

Myers, Glenford J. 2012. *The art of software testing.* 3rd ed. Hoboken: John Wiley  Sons. https://ebookcentral.proquest.com/lib/jyvaskyla-ebooks/detail.action?docID=697721.

Ricca, Filippo, ja Paolo Tonella. 2001. “Analysis and testing of Web applications”. Teoksessa *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001,* 25–34. https://doi.org/10.1109/ICSE.2001.919078.