LAUREA AMMATTIKORKEAKOULU Uuden edellä

Laurea Liven toiminnallinen testaus

Keinänen, Jarmo

2015 Kerava

Laurea-ammattikorkeakoulu
Kerava

Laurea Liven toiminnallinen testaus

Jarmo Keinänen
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Lokakuu, 2015

Laurea-ammattikorkeakoulu **Tiivistelmä**
Kerava
Tietojenkäsittelyn koulutusohjelma

Keinänen, Jarmo

Laurea Liven toiminnallinen testaus

Vuosi 2015 Sivumäärä 49

Tämän opinnäytetyön aihe oli ohjelmistotestaus, tarkemmin sanottuna web-sovelluksen toiminnallinen testaus. Empiirinen osa tehtiin Laurea-ammattikorkeakoulun intranetin, Laurea Liven, hyväksymistestauksen kaltaisesta testauksesta.

Tutkimuksessa tehtiin kaksi iteraatiota. Ensimmäisellä iteraatiolla katsottiin, minkä tyyppisiä havaintoja löydettiin Laurea Livestä ja miten. Toisella iteraatiolla tutkittiin, löydettiinkö samoja havaintoja kahtena vuonna peräkkäin. Tutkimuksen tavoite oli analyysipäätelmä näistä asioista.

Tutkimuksen teoriaosuus koottiin alan kirjallisuudesta. Keskeiseen teoriaan kuuluu seuraavia osia: viisivaiheinen testausprosessi, V-malli, testitapaukset, toiminnallinen testaus, dynaaminen testaaminen, mustalaatikkotestaus, havaintoraportin teko ja web-sovelluksen testaus.

Varsinainen tutkimus tehtiin opiskelijoiden kurssilla tekemien raporttien pohjalta Laurea Liven testauksesta. Raporttien tärkeimmät kohdat koottiin analyysitaulukkoon.

Tutkimuksen tärkeimmät tulokset olivat listat löydetyistä havaintojen tyypeistä ja tavoista havaintojen löytämiseksi. Kahtena vuonna peräkkäin löytyi kaksi samaa havaintoa.

Asiasanat: ohjelmistotestaus, web-sovelluksen testaus, intranet

Laurea University of Applied Sciences **Abstract**
Kerava
Business Information Technology

Keinänen, Jarmo

The functional testing of Laurea Live

Year 2015 Pages 49

The topic of this thesis was software testing, functional testing of a web application to put it more accurately. The empirical part dealt with testing of the intranet of Laurea University of Applied Sciences, Laurea Live. The testing was done by students of the school during the course E0023 Testing and Introducing Web Services during the years 2013 and 2014 and it resembled acceptance testing.

The study was done in two iterations. During the first, the types of incidents that the students found was the focus of the study. During the second, the focus was what types of incidents the students found for two years in succession. The aim of the study was an analysis on the research questions.

The empirical study was done based on the reports that the students wrote on their testing of Laurea Live in the course. The important items of the reports were collected into an analysis table.

The main results of this study were a list of incident types found and another on ways to find them. The students found two incidents that were repeated in the two years.

Keywords: software testing, testing of a web application, intranet

Sisällys

1 Johdanto 6
2 Työn tausta ja rajaukset 6
3 Ohjelmistotestauksen teoriaa 7
3.1 Testauksen merkitys, käsitteitä ja periaatteita 7
3.1.1 Testauksen käsitteitä 8
3.1.2 Ohjelmistotestauksen periaatteita 9
3.2 Testausryhmä, testausprosessi ja organisaatio 12
3.2.1 Testitapaukset 13
3.2.2 Testausprosessi 15
3.3 Testaustasot, riskiohjattu testaus ja testaussuunnittelu 18
3.4 Toiminnallinen ja ei-toiminnallinen testaus 26
3.5 Testaustekniikat ja –tavat 26
3.6 Havaintoraportin teko ja havaintojen hallinta 29
3.7 Web-pohjaisten järjestelmien testaaminen 32
4 Tutkimusmenetelmät 38
5 Laurea Liven toiminnallisen testauksen toteutus 40
6 Tutkimustulokset 41
7 Yhteenveto ja johtopäätökset 42
8 Jatkokehitysehdotukset 45
Lähteet 46
Kuviot 48
Taulukot 49

1 Johdanto

Tässä opinnäytetyössä käsitellään www-sovelluksen toiminnallista testausta. Käytännön osuus on Laurea-ammattikorkeakoulun intranetsovelluksen, nimeltään Laurea Live, toiminnallisesta testauksesta. Teoreettinen osuus käsittää sovellustestauksen ja web-sovelluksen testauksen.

Tutkijan henkilökohtainen kiinnostus on tärkeä tekijä tutkimusaiheen valinnassa (Uusitalo 1991, 57). Henkilökohtainen kiinnostukseni aiheeseen tulee etenkin siitä, että kuulemani mukaan ensimmäinen it-koulutuksen jälkeinen työpaikka voi hyvin olla ohjelmistojen testauksessa. Sitä silmällä pitäen kävin keväällä 2014 koulussa kurssin E0023 Verkkopalvelun testaus ja käyttöönotto.

Uusitalon (1991, 50) mukaan tutkimusta voidaan tehdä myös aineistopohjaisesti, jolloin lähtökohtana on aineisto, johon sitten yritetään keksiä tutkimusongelma. Tämä on tilanne tässä tutkimuksessa.

Teoriaa kirjoitettaessa vaikutti siltä, että nykyisessä tiedossa web-sovellusten toiminnallisesta testauksesta on vain vähän kehittämisen varaa. Siellä on kuitenkin aukko, jota yritän täyttää vastaamalla tutkimuskysymyksiini.

Yinin (1989, 38-40) mukaan tapaustutkimuksessa pyritään analyyttiseen yleistettävyyteen, ei tilastolliseen yleistettävyyteen. Keskeistä ovat aineistosta tehtyjen tulkintojen kestävyys ja syvyys (Eskola & Suoranta 1998, 67).

2 Työn tausta ja rajaukset

Aihe tutkimukseen on tullut lähinnä omasta kiinnostuksesta, josta kerroin jo johdannossa. Kerrottuani aiheesta tietojenkäsittelyn lehtorille, hän neuvoi minua käymään kurssin Verkkopalvelun testaus ja käyttöönotto ja kirjoittamaan opinnäytetyön sen jälkeen. Kurssilla sain perustiedot ohjelmistotestauksesta ja tein ryhmätyönä Laurea Liven, koulun intranetin, toiminnallisen testauksen. Näin pääsin myös tekemään osallistuvaa tutkimusta.

Keskityn tutkimuksessa ohjelmistojen, tarkemmin web-sovellusten, toiminnalliseen testaukseen. Tutkimusaineisto on Laurea-ammattikorkeakoulun intranetin, Laurea Liven, testauksesta, jonka opiskelijat tekivät kurssilla E0023 Verkkopalvelun testaus ja käyttöönotto vuosina 2013 ja 2014. Testaus oli hyväksymistestauksen kaltaista. Siitä sain aineiston tutkimukseeni, jonka pohjalta kirjoitin tutkimuksen empiriaosan.

Tämän opinnäytetyön tutkimusongelmana on: minkälaisia havaintoja löydetään tuotannossa olevasta web-sovelluksesta toiminnallisella testauksella? Tähän ongelmaan päädyin koottuani teoriaosuuden ja tutkimusaineiston. Teoria koostettiin ohjelmistotestausta käsittelevästä kirjallisuudesta.

Tutkimuskysymyksiä tässä opinnäytetyössä on kaksi:

- Minkä tyyppisiä havaintoja web-sovelluksen toiminnallisessa testauksessa löydetään?
- Millä tavoilla havainnot löydetään?

Opinnäytetyön johtopäätöksissä tuodaan esille tutkimuskysymyksien pohjalta analyysipäätelmä. Näihin tutkimuskysymyksiin olen päätynyt, koska empiriaosan tutkimusaineistolla on mahdollista vastata niihin. Teoriaan liittyväksi kysymykseksi valikoitui: mitä seikkoja pitäää ottaa huomioon web-sovelluksen toimi8nnallisen testauksen suunnittelussa?

Tein kaksi iteraatiota tutkimuksessa. Ensimmäisellä kierroksella tutkin minkä tyyppisiä havaintoja opiskelijat löysivät ja miten. Toisella tutkin, löysivätkö opiskelijat samoja havaintoja kahtena vuonna peräkkäin.

3 Ohjelmistotestauksen teoriaa

Tämän opinnäytetyön teoriaosuus on laaja, koska sen haluttiin käsittelevän ohjelmistotestauksen teoriaa yleisesti. Lähes kaikki teoria Laurea-ammattikorkeakoulun kurssilla E0023 Verkkopalvelun testaus ja käyttöönotto on tullut käsiteltyä. Tekstiä ei ole otettu suoraan kurssimateriaalista vaan yleisesti saatavilla olevasta kirjallisuudesta.

3.1 Testauksen merkitys, käsitteitä ja periaatteita

Testaaminen on ohjelman ajamista tarkoituksena löytää siitä vikoja (Myers, Badgett & Sandler 2012, 18). Ohjelmistoja testataan, jotta niissä olevat viat voitaisiin korjata ajoissa ennen kuin ohjelmisto julkaistaan eli ohjelmistojen laadun parantamiseksi. Julkaisun jälkeen vikojen korjaaminen on kallista. Ohjelmistoviat ovat mm. pudottaneet lentokoneita ja aiheuttaneet keskeytyksiä pörssikauppaan. (Choudhary & Kumar 2011, 3.)

Julkaisun jälkeisten vikojen korjaamiseen tarvitaan uutta koodia, joka kasvattaa nopeasti ohjelmiston rivimäärää. Web-sovellusten viat ajavat käyttäjät ulos sovelluksesta ja kokeilemaan vaihtoehtoisia sovelluksia. Sovellukseen julkaisun jälkeen jäävät viat voivat tehdä sovelluksesta käyttökelvottoman ja pilata yrityksen maineen. (Farrell-Vinay 2008, 5 – 7.)

Testaus ei tule koskaan valmiiksi, mutta testausryhmä voi tehdä parhaansa siinä ajassa, joka sille on annettu. Kelvottoman julkaisun vieminen tuotantoon voi viedä yrityksen jopa konkurssiin. Testaus estää yritystä ja asiakkaita vahingoittumasta viallisten järjestelmien vuoksi. (Farrell-Vinay 2008, 3.)

3.1.1 Testauksen käsitteitä

Havainto (incident) on mikä tahansa tapahtuma, joka testauksessa havaitaan, joka vaatii tutkimista (ISTQB 2007, 21). Havainnoista kirjoitetaan lyhyet muodolliset raportit ja niistä tehdään keskitetty tietokanta (Spillner, Linz & Schaefer 2014, 193 – 194).

Häiriö (failure) tarkoittaa, että annettu vaatimus ei täyty. Se on eroavuus saadun tuloksen tai käyttäytymisen ja odotetun tuloksen tai käyttäytymisen välillä. Häiriö tapahtuu ohjelmassa olevan vian (fault, defect, bug) vuoksi. (Spillner ym. 2014, 7.)

Vika (fault, defect, bug) on komponentin vika, jonka vuoksi komponentti ei toimi vaaditulla tavalla. Vika voi aiheuttaa häiriöitä ohjelmassa (Spillner ym. 2014, 254). Vian syy on inhimillinen erehdys esimerkiksi ohjelmaa kirjoitettaessa (Spillner ym. 2014, 8).

Virhe (error) tarkoittaa ihmisen toimintaa, joka aiheuttaa väärän tuloksen. Esimerkki voisi olla kehittäjän tekemä viallinen ohjelmointi. Virhe on vian tai häiriön syy. Häiriöitä voivat lisäksi aiheuttaa ympäristön olosuhteet, kuten magnetismi ja säteily. (Spillner ym. 2014, 8.)

Mustalaatikkotestauksessa testien tekemiseen saadaan materiaalia ohjelmiston ulkopuolisista kuvauksista. Näitä ovat määrittelyt, vaatimukset ja testisuunnitelma. (Amman & Offutt 2008, 21.)

Lasilaatikkotestauksessa eli valkolaatikkotestauksessa testeihin saadaan materiaalia ohjelmiston lähdekoodista. Erityisesti käytetään haarautumia, ehtoja ja lauseita. (Amman & Offutt 2008, 21.) Tarkemmin sanottuna valkolaatikkotestauksessa ollaan kiinnostuneita siitä, kuinka hyvin testitapaukset ajavat tai kattavat ohjelman logiikkaa tai koodia. (Myers ym. 2012, 42).

Testauksen V-malli sisältää järjestelmän rakentamisen ja testaamisen päävaiheet eli testaustasot. Testauksen vaiheita on neljä: yksikkötestaus, integrointitestaus, järjestelmätestaus ja hyväksymistestaus. (Kasurinen 2013, 51.)

Hyväksymistestaus on viimeinen vaihe V-mallissa. Siinä on päätavoitteena osoittaa ohjelman täyttävän vaatimusmäärittelyssä asetetut vaatimukset. Siinä ei enää pitäisi löytyä vikoja vaan järjestelmän odotetaan toimivan. (Kasurinen 2013, 57.)

Testitapaus on kokonaisuus, jonka muodostavat syötteet, esiehdot suorittamiselle, odotetut tulokset ja suorituksen jälkiehdot. Se muodostetaan tiettyä tavoitetta tai testauksen kohdetta varten. (ISTQB 2007, 42.)

3.1.2 Ohjelmistotestauksen periaatteita

Vian löytämiseksi se pitää paikantaa ohjelmassa. Aluksi tiedetään vian vaikutus, mutta ei sen tarkkaa sijaintia ohjelmassa. Paikantaminen ja vian korjaaminen ovat ohjelmistokehittäjän tehtäviä, joita usein kutsutaan debuggaukseksi (debugging). (Spillner ym. 2014, 8.)

Vikojen korjaaminen tuo ohjelmaan usein uusia vikoja. Siksi täytyy toistaa korjausten jälkeen samat testitapaukset, jotka paljastivat viat. Niiden lisäksi täytyy yrittää paljastaa mahdolliset sivuvaikutukset uusilla testitapauksilla. (Spillner ym. 2014, 8.)

Yhtäkään ohjelmistoa, jossa ei ole vikoja, ei ole olemassa eikä sellaista todennäköisesti tule lähitulevaisuudessa. Sellainen on mahdollista vain, jos järjestelmä on triviaalia monimutkaisuuden tasoa. Kuitenkin on myös olemassa monia ympäri vuorokauden luotettavasti toimivia ohjelmistojärjestelmiä. (Spillner ym. 2014, 10.)

Vaikka kaikki testitapaukset olisi ajettu eikä uusia vikoja löydettäisi, niin ei voida turvallisesti sanoa, että ohjelmassa ei ole vikoja tai että uudet testitapaukset eivät löytäisi niitä. Vain hyvin pienissä ohjelmissa voidaan tietää, että vikoja ei ole. (Spillner ym. 2014, 10.)

Testauksella ei voida todistaa, että ohjelmassa ei ole vikoja. Sen todistamiseksi pitäisi ajaa ohjelma kaikilla mahdollisilla syötteillä ja kaikissa mahdollisissa oloissa. Se ei ole käytännössä mahdollista. (Spillner ym. 2014, 13.)

Spillner ym. (2014, 33) toteavat, ”Viimeisten 40 vuoden aikana useat periaatteet testaukselle ovat tulleet hyväksytyiksi yleisiksi säännöiksi testaustyölle.” Listaan nämä periaatteet alle.

Periaate 1: ”Testaus osoittaa vikojen olemassaolon, ei niiden poissaoloa.” Testaus voi näyttää, että tuotteessa on vikoja, ei todistaa sitä, että vikoja ei ole. Vaikka testeissä ei löytyisi vikoja, se ei ole todiste siitä, että niitä ei olisi. (Spillner ym. 2014, 33.)

Periaate 2: ”Kaiken kattava testaaminen on mahdotonta.” Ei ole mahdollista ajaa tyhjentävää testiä, joka sisältäisi kaikki arvot kaikilla syötteillä ja niiden yhdistelmillä kaikkien erilaisten esiehtojen yhdistelmien kanssa. Se vaatisi astronomisen korkean määrän testejä. Jokainen

testi on vain näyte. Siksi testausta täytyy kontrolloida riskin ja prioriteetit huomioon ottaen. (Spillner ym. 2014, 34.)

Periaate 3: testaaminen pitäisi aloittaa mahdollisimman aikaisin. Testauksen pitäisi alkaa mahdollisimman aikaisin ohjelmiston elinkaaressa ja keskittyä määritettyihin tavoitteisiin. Näin vikoja löydetään aikaisessa vaiheessa. (Spillner ym. 2014, 34.)

Periaate 4:n mukaan viat eivät jakaudu tasaisesti vaan kasautuvat yhteen. Useimmat viat löytyvät muutamasta kohdasta testattavasta ohjelmasta. Siksi monia vikoja löytyy yhdestä paikasta. Tavallisesti muita vikoja on lähistöllä. (Spillner ym. 2014, 34.)

Periaate 5 on hyönteismyrkyn paradoksi. Hyönteiset ja bakteerit tulevat vastustuskykyisiksi tuholaismyrkyille. Samoin testit menettävät tehoaan toistettaessa eikä niillä löydetä uusia vikoja. Vanhoja tai uusia vikoja voi olla paikoissa, joita ei ole käyty läpi testitapauksilla. Siksi uusia ja muokattuja testitapauksia täytyisi kehittää. (Spillner ym. 2014, 34.)

Periaate 6 sanoo, että ”testaus on kontekstista riippuvaista.” Testaus täytyy sovittaa sovelluksen käytön ja ympäristön riskeihin. Siitä syystä kahta sovellusta ei pitäisi testata täsmälleen samalla tavalla. Testauksen intensiteetti, lopetusehdot yms. pitäisi päättää erikseen jokaiselle ohjelmistojärjestelmälle sen käyttöympäristöstä riippuen. (Spillner ym. 2014, 35.)

Periaatteen 7 mukaan ajatus, että vikojen löytymättömyys merkitsee, että järjestelmä on käyttökelpoinen, on harha. Häiriöiden löytäminen ja vikojen korjaaminen eivät takaa järjestelmän täyttävän käyttäjien odotukset ja tarpeet. Käyttäjien täytyisikin osallistua kehitystyöhön ja prototyyppien käyttöön. (Spillner ym. 2014, 35.)

Myers ym. antavat kymmenen periaatetta lisää. Periaatteen 1 mukaan ”Tarpeellinen testitapauksen osa on odotetun tuloksen määrittely”. Testitapauksessa täytyy olla kaksi osaa: syötteet ohjelmalle ja ohjelman odotetut tulokset näillä syötteillä. (Myers ym. 2012, 13 – 14.)

Toinen periaate sanoo, että ”ohjelmoijan pitäisi välttää oman ohjelmansa testaamista”. Ohjelmoija on haluton löytämään vikoja omasta työstään. Ohjelmoija on voinut ymmärtää vaatimuksen määrittelyn väärin, jolloin testatessaan hän ei löytäisi määrittelyn toteutuksesta ohjelmassa vikaa. Testaaminen on tehokkaampaa toisen tekemänä. Debuggaus eli tunnettujen vikojen korjaaminen on sitä vastoin tehokkaampaa alkuperäisen ohjelmoijan tekemänä. (Myers ym. 2012, 14 – 15.)

Kolmas periaate sanoo, että ”ohjelmoivan organisaation ei pitäisi testata omia ohjelmiaan”. Organisaatioilla on samanlaisia psykologisia ongelmia, kuin yksittäisillä ohjelmoijilla. Lisäksi

näiden organisaatioiden tulosta mitataan usein kyvyllä tuottaa ohjelma tietyssä ajassa ja tietyillä kustannuksilla. Ohjelman luotettavuutta taas on vaikeaa mitata. Siksi testaamisen voidaan katsoa laskevan todennäköisyyttä saavuttaa aikataulun ja kustannusten vaatimukset. Organisaatio tietenkin löytää vikoja myös omasta ohjelmastaan, mutta työn antaminen objektiivisen itsenäisen osapuolen tehtäväksi on taloudellisempaa. (Myers ym. 2012, 15.)

Neljäs periaate kuuluu näin: Minkä tahansa testausprosessin pitäisi sisältää jokaisen testin tulosten huolellisen tarkastuksen". Myöhemmissä testeissä löytyy usein aikaisemmissa testeissä huomaamattomia vikoja. (Myers ym. 2012, 15.)

Viides periaate on seuraava. "Testitapauksia täytyy kirjoittaa syötteille jotka ovat vääriä ja odottamattomia, kuten myös niille, jotka ovat oikeita ja odotettuja". Monet häiriöt löytyvät ohjelman käytössä, kun sitä käytetään tavalla, joka on uusi tai odottamaton. Siksi odottamattomilla ja väärillä syötteillä testaamalla löydetään paljon häiriöitä. (Myers ym. 2012, 15 – 16.)

Kuudes periaate on seuraava: "Ohjelman tutkiminen sen katsomiseksi, että se ei tee, mitä sen pitäisi tehdä, on vain puolikas taistelusta; toinen puoli on sen katsominen, tekeekö ohjelma mitä sen ei pitäisi tehdä". Tämä on edellisen periaatteen seuraus. Ohjelmat täytyy tarkastaa epätoivottujen sivuvaikutusten varalta. Esimerkiksi palkanmaksuohjelma ei oikeiden palkkashekkien lisäksi saa tulostaa shekkejä olemattomille työntekijöille. (Myers ym. 2012, 16.)

Seitsemännessä periaatteessa sanotaan, että "vältä poisheitettäviä testitapauksia jos ohjelma ei todella ole poisheitettävä ohjelma." On tavallista, että istutaan päätteellä, keksitään testejä lennossa ja annetaan syötteitä ohjelmalle. Ongelma tästä tulee, jos testejä ei säilytetä. Testit ovat arvokas investointi. Kun ohjelmaa tarkastetaan uudelleen, esim. vian korjauksen jälkeen, testitapaukset täytyy keksiä uudelleen, kun regressiotestauksessa pitäisi ajaa samat testit kuin viimeksi. (Myers ym. 2012, 16.)

Periaate 8: "Älä suunnittele testausta äänettömän oletuksen vallitessa, että virheitä ei löydy". Testaaminen on prosessi, jossa ajetaan ohjelmaa tavoitteena löytää vikoja. Jopa laajan testaamisen jälkeen voidaan yhä olettaa että vikoja on edelleen; niitä vain ei ole löydetty vielä. (Myers ym. 2012, 16 – 17.)

Periaate 9 on seuraava: "Todennäköisyys virheiden olemassa ololle ohjelman osassa on suhteessa osasta jo löydettyjen vikojen määrään". Otetaan esimerkiksi ohjelma, jossa on komponentit A ja B. Komponentista A on löytynyt 5 vikaa ja komponentista B vain yksi. Jos osaa A ei ole alistettu tiukemmalle tutkimukselle, niin sieltä todennäköisesti löytyy enemmän vikoja lisää, kuin osasta B. (Myers ym. 2012, 17 – 18.)

Periaate 10 on, että ”testaus on äärimmäisen luova ja älyllisesti haastava tehtävä”. Suuren ohjelman testaamisessa tarvittava luovuus ehkä ylittää sen suunnitteluun tarvitun luovuuden. (Myers ym. 2012, 18.)

3.2 Testausryhmä, testausprosessi ja organisaatio

Testausryhmän jäsenillä on erilaisia rooleja ja taitoja. Testauksessa tarvitaan spesialisteja, joiden tiedot kattavat koko testausaktiviteettien kentän (Spillner ym. 2014, 172). Kaikille yhteisesti käytetään nimitystä testaaja (tester).

Testauspäällikkö (test manager) on testauksen suunnittelun ja kontrollin asiantuntija, jolla on tietoa ja kokemusta testauksesta, laadun johtamisesta, projektin johtamisesta ja henkilöstöjohtamisesta. Testauspäällikkö suunnittelee testauspolitiikan, testauksen lähestymistavan ja suunnitelman. Hän hankkii testausresurssit. Hän valitsee testausstrategiat ja -menetelmät, esittelee tai parantaa testaustyökaluja, järjestää valmennuksen työkaluja varten sekä päättää testausympäristöstä ja testausautomaatiosta. Hän optimoi testausta tukevat prosessit, jotta muutokset voitaisiin jäljittää ja testien uudelleentoteutettavuus varmistaa. Hän arvioi testaussuunnitelmassa olevia mittareita ja käyttää niitä. Hän sopeuttaa testisuunnitelmat säännöllisesti pohjautuen testituloksiin. Hän tunnistaa sopivat mittarit testauksen etenemisen mittaamiseen sekä arvioi testauksen ja tuotteen laadun. Hän kirjoittaa testausraportit ja kommunikoi niistä. (Spillner ym. 2014, 172.)

Testisuunnittelija/-analyytikko (test designer/analyst) on testausmenetelmien ja testispesifikaation asiantuntija, jolla on tietoa ja kokemusta ohjelmistotestauksesta, ohjelmistotekniikasta ja määrittelymenetelmistä. Hän katselmoi vaatimukset, määrittelyt ja mallit testattavuutta varten ja testitapausten kirjoittamiseksi. Hän luo testausmääritelmät. Hän valmistelee ja hankkii testausdatan. (Spillner ym. 2014, 173.)

Testiautomatisoija (test automator) on testausautomaation asiantuntija, jolla on tietoa testauksen perusteista, ohjelmointikokemusta, ja syvä tietämys testityökaluista ja skriptikielistä. Hän automatisoi testejä saatavilla olevia työkaluja käyttäen. (Spillner ym. 2014, 173.)

Testihallinnoija (test administrator) on asiantuntija testiympäristön asentamisessa ja käytössä. Hänellä on järjestelmän hallinnoijan tietämys. Hän asentaa ja tukee testiympäristöä koordinoiden yleisen järjestelmän hallinnon ja verkkohallinnon kanssa. (Spillner ym. 2014, 173.)

Testaaja (tester) on asiantuntija testaamisessa ja häiriöiden raportoinnissa. Hänellä on tietämystä informaatioteknologian perusteista, testauksen perusteista, testityökalujen käytöstä ja

ymmärrys testin kohteesta. Hän katselmoi testisuunnitelmat ja testitapaukset. Hän käyttää testaustyökaluja ja monitorointityökaluja. Hän ajaa testit ja tekee lokit mukaan lukien tulosten arviointi sekä dokumentointi ja paljastaa niiden puutteita. (Spillner ym. 2014, 173.)

Erityisesti järjestelmätestauksessa on usein tarvetta lisätä ryhmään IT-asiantuntijoita ainakin väliaikaisesti. Heitä voivat olla tietokantahallinnoijat, tietokantasuunnittelijat, verkkoasiantuntijat, tai spesialistit järjestelmän alalta tai yrityksen toimialalta. (Spillner ym. 2014, 174.)

3.2.1 Testitapaukset

Testit määritellään testitapauksiksi. Testille annetaan nimi. Sille annetaan kuvaus, eli mitä testi tarkistaa ja yleinen kuvaus, mitä tapahtuu testin aikana. Testille voidaan määritellä, mitä vaatimuksia se kattaa. Sille määritellään esivaatimukset, syötteet ja odotetut tulokset. (Fournier 2009, 154 – 155.)

Testitapauksien määrittely tapahtuu kahdella askeleella. Loogiset testitapaukset määritellään ensin. Sitten määritellään konkreettiset fyysiset testitapaukset eli todelliset syötteet ohjelmalle. Päinvastoin voidaan myös toimia eli fyysisistä testitapauksista loogisiin testitapauksiin. Näin toimitaan, jos testin kohde on riittämättömästi määritelty ja testin määrittely täytyy tehdä kokeellisella tavalla. Fyysisten testitapauksien kehittäminen kuuluu testien toteutukseen. (Spillner ym. 2014, 23.)

Jokaiselle testille on määrättävä alkutilanne eli ennakkoehdot. Tässä määrätään ympäristön tilanne. Ennen testin ajamista täytyy tietää odotetut tulokset ja käyttäytyminen. Tuloksiin kuuluvat tulosteet, muutokset globaaliin dataan ja tiloihin ja testitapauksen muut seuraukset. (Spillner ym. 2014, 23.)

Odotettujen tulosten määräämiseksi tarvitaan testioraakkeli eli mekanismi odotettujen tulosten ennustamiseksi. Sellaisena voivat toimia määrittelyt. On kaksi mahdollisuutta. Ensinnäkin, odotettu data voidaan saada testin kohteen määrittelystä. Toiseksi, jos funktiot toiseen suuntaan kulkemiseen ovat olemassa, voidaan käyttää niitä testin jälkeen ja verrata tulosta alkuperäiseen syötteeseen. (Spillner ym. 2014, 23.)

Testitapaukset voidaan jakaa kahteen tapaukseen. Ensiksi, testitapaus voi tutkia määriteltyä käyttäytymistä ja tulosteita. Tähän kuuluvat testitapaukset, jotka tutkivat poikkeusten ja virheiden määriteltyä käsittelyä. Toiseksi, testitapaus voi tutkia kohteen reaktiota virheellisiin ja odottamattomiin syötteisiin ja olosuhteisiin, joilla ei ole määriteltyä poikkeuksen käsittelyä. (Spillner ym. 2014, 23.)

Konkreettisia testitapauksia voidaan käyttää ilman muutoksia testien ajamiseen. Testitapauksien ja määrittelyjen välinen yhteinen jäljitettävyys on varmistettava ja tarvittaessa päivittävä. Testien ajaminen täytyy kuvata etukäteen. (Spillner ym. 2014, 25.)

Taulukko 1 luettelee erilaisia testitapauksien luokkia. Luettelon on antanut Myers ym. (2012, 122.) Eri luokkia on tässä listassa 15 kappaletta. Ei ole varmaa, onko tässä kaikki mahdolliset luokat vai voisiko niitä löytyä vielä lisää.

Kategoria	**Kuvaus**
Fasiliteetti	Varmistetaan, että toiminnallisuus tavoitteissa on toteutettu.
Volyymi	Annetaan ohjelmalle tavallista suurempi määrä dataa käsiteltäväksi.
Rasitus	Annetaan ohjelmalle tavanomaista suurempi kuorma, yleensä yhdenaikaista käsittelyä.
Käytettävyys	Selvitetään, kuinka hyvin käyttäjä voi vuorovaikuttaa ohjelman kanssa.
Turvallisuus	Yritetään ohittaa ohjelman turvatoimet.
Suorituskyky	Tarkistetaan, täsmääkö ohjelma täyttää vasteajan ja suoritustehon vaatimukset.
Tallennus	Varmistetaan ohjelman täyttävän tallennustarpeensa oikein.
Kokoonpano	Tarkistetaan, että ohjelma toimii oikein suositelluilla kokoonpanoilla.
Yhteensopivuus/ muuntaminen	Tarkistetaan, onko ohjelman uusi versio yhteensopiva aikaisempiin versioihin.
Asentaminen	Varmistetaan, että asennusmenetelmät toimivat kaikilla suositelluilla alustoilla.
Luotettavuus	Tarkistetaan, vastaako ohjelma luotettavuusvaatimuksia, kuten käytettävyysaika ja MTBF.
Ylläpidettävyys	Varmistetaan, tarjoaako ohjelma mekanismit datan saamiseen tilanteissa, jotka vaativat teknistä tukea.
Toipuminen	Testataan, toimivatko järjestelmän toipumistoiminnot, kuten suunniteltu.
Dokumentaatio	Validoidaan käyttöohjeiden oikeellisuus.
Toiminto	Varmistetaan erikoisten ohjelman käyttöön tai ylläpitoon tarkoitettujen menettelyjen tarkkuus.

Taulukko 1. Testitapausten kategoriointia. (Myers ym. 2012, 122)

Testitapausten suunnittelu on tärkeää, koska ohjelman täydellinen testaaminen on mahdotonta. Siksi ilmiselvä strategia on tehdä testeistä niin kokonaisia kuin mahdollista annetussa ajassa ja annetuilla resursseilla. Näillä rajoituksilla kysymys on: ”Millä mahdollisten testitapauksien joukolla on suurin mahdollisuus havaita eniten virheitä?” (Myers ym. 2012, 41.)

Testidatan valintaan täytyy käyttää ajattelua. Menestyksellinen testistrategia koostuu sekä mustalaatikko- että lasilaatikkotestaustekniikoista. Niitä käsitellään osassa Testaustekniikat ja –tavat. (Myers ym. 2012, 41.)

Vaatimuksiin perustuvassa testauksessa suunnitellaan vähintään yksi testitapaus jokaista vaatimusta kohti. Yleensä enemmän, kuin yksi testitapaus tarvitaan testaamaan toiminnallinen vaatimus. Jos vaatimukseen liittyvät testit menevät läpi moitteetta, katsotaan vastaava toiminnallisuus hyväksytyksi. (Spillner ym. 2014, 70 – 71.)

3.2.2 Testausprosessi

Vaatimusten pitäisi ohjata koko ohjelmistokehitysprosessia. Siksi ne ovat tärkeitä testauksessakin. Niiden ollessa selvillä tiedetään mitä pitää testata. Hyvät vaatimukset ovat selviä eli ymmärrettäviä, täysilukuisia, järkevän yksityiskohtaisia, varmistettavissa olevia, oikeita, johdonmukaisia ja yksimerkityksisiä. (Fournier 2009, 68 – 70.)

Spillner ym. (2014, 19) esittävät kirjassaan perusprosessin ohjelmistotestaukselle. Se sisältää viisi tehtävää:

1. Suunnittelu ja valvonta (planning and control).
2. Analyysi ja suunnittelu (analysis and design).
3. Toteutus ja ajaminen (implementation and execution).
4. Lopetuskriteerien arviointi ja raportointi (evaluation of exit criteria and reporting).
5. Testauksen lopetustoimet (test closure activities).

Testauksen suunnittelu alkaa ohjelmistokehitysprosessin alussa. Suunnitelmaa voidaan muokata projektin aikana. Suunnittelussa päätetään testauksen tehtävästä ja tavoitteista sekä käytettävistä resursseista. Resursseihin kuuluvat yksittäiset työntekijät, mihin tehtäviin heitä tarvitaan ja milloin. Pitää päättää käytettävästä ajasta, laitteista ja työkaluista. Suunnittelussa tehdään kaikesta tästä testaussuunnitelma. Valvonnassa tarkkaillaan testauksen kulkua ja verrataan sitä projektisuunnitelmaan. Eroavuuksista raportoidaan ja ryhdytään toimenpiteisiin tavoitteisiin pääsemiseksi uudessa tilanteessa. Testaussuunnitelmaa ylläpidetään muuttuvissa tilanteissa. Testausjohtamisen tehtäviin kuuluvat testausprosessin, testausinfrastruktuurin ja testausvälineistön hallinnointi ja ylläpito. Edistymisen seuranta voidaan pohjata raportointiin, jota saadaan työntekijöiltä ja työkalujen automaattisesta datasta. (Spillner ym. 2014, 19 – 20.)

Tärkeä tehtävä suunnittelussa on testausstrategian tai lähestymistavan päättäminen. Testaukseen täytyy päättää priorisointi riskien perusteella. Tehtävät täytyy jakaa alajärjestelmille riskin ja häiriön vaikutusten vakavuuden perusteella. Kriittiset alajärjestelmät täytyy testata vähemmän kriittisiä tarkemmin. Testausstrategiassa testaussuunnitelmassa päätetään testaustekniikoista, joita testauksessa käytetään. Testausstrategia riippuu vaatimuksista luotettavuudelle ja turvallisuudelle. Korkean riskin tuote pitää testata vähemmän kriittistä tarkemmin. (Spillner ym. 2014, 20 – 22.)

Jos testaustyökaluja ei vielä ole, niiden valinta ja hankinta täytyy aloittaa aikaisin. Testiympäristön osat, joissa alajärjestelmiä voidaan testata yksittäin, täytyy usein ohjelmoida niin, että ne ovat valmiit, kun testattavat osat valmistuvat. Myös testattavan kohteen täytyy olla testattavissa. Kohteen rajapintoja täytyy pystyä käyttämään testiympäristössä. Kohde täytyy ohjelmistokehityksessä ohjelmoida täyttämään nämä vaatimukset. (Spillner ym. 2014, 20 – 22.)

Testien toteutuksen ja ajamisen vaiheessa testit täytyy muuttaa konkreettisiksi testitapauksiksi. Testit ajetaan ja niistä kirjoitetaan lokitiedot. Testit pitäisi ryhmitellä testijoukkoihin (test suites), mikä edistää testien ymmärtämistä ja tehokasta ajamista. (Spillner ym. 2014, 25.) Testijoukko on joukko testitapauksia, jossa usein edellisen testin jälkiehdot toimivat seuraavan testin esiehtoina (ISTQB 2007, 48).

Ennen testien aloittamista täytyy testikehyksen (test harness) olla valmiina. Se sisältää ajurit (drivers) ja simulaattorit (simulators), jotka tarvitaan testitapauksien ajamiseksi. Myös testikehyksen viat voivat aiheuttaa häiriöitä. Siksi testikehyksen toiminta täytyy varmistaa. Testaaminen voi alkaa heti testattavien alijärjestelmien valmistuttua, kun kaikki valmistelutyöt ovat valmiit. (Spillner ym. 2014, 26.) Ajuri on komponentin korvaava testaustyökalu ja ohjelmistokomponentti, jolla kontrolloidaan tai kutsutaan testattavaa komponenttia tai järjestelmää (ISTQB 2007, 15). Simulaattori on toisen määrätyn järjestelmän tavoin käyttäytyvä testauksessa käytettävä laite, ohjelma tai järjestelmä (ISTQB 2007, 38).

Testauksen aluksi testattavasta osasta tarkistetaan, että se on kokonainen. Se asennetaan testiympäristöön ja tarkistetaan, että se käynnistyy ja tekee perustoimintonsa, siis tehdään sille aloitustesti (smoke test). Häiriöiden ilmetessä ei testausta kannata jatkaa, vaan komponentti pitää ensin korjata. (Spillner ym. 2014, 26.)

Testeistä täytyy kirjoittaa tarkka loki. Se sisältää tiedot siitä, mitkä testit on ajettu ja millä tuloksilla (hyväksytty/häiriö), kuka testin teki ja milloin. Informaatio täytyy ylläpitää niin, että testi on helppo toistaa myöhemmin samalla syötteellä ja ehdoilla. Lokiin tulisi kirjoittaa myös testikattavuus, jota pitäisi mitata. (Spillner ym. 2014, 26.)

Jos eroja odotettujen ja todellisten tulosten välillä löytyy testeissä, täytyy päättää onko kyseessä häiriö. Häiriöt täytyy dokumentoida. Häiriöstä täytyy analysoida karkeasti, mikä on siihen syynä. Häiriön syynä voi olla myös virheellinen tai epätarkka testin määrittely tai testitapaus, tai virheellinen testin ajo. Häiriön vakavuuden mukaan tehdään päätös vikojen korjaamisen priorisoinnista. Vikojen tultua korjattua, tarkistetaan regressiotestauksella, että ne on todella korjattu eikä uusia vikoja ole syntynyt. (Spillner ym. 2014, 26 – 27.)

Riskipohjainen testaus (risk-based testing) on seuraavanlaista. Kaikkien määriteltyjen testien tekemiseen ei välttämättä löydy aikaa ohjelmistoprojektissa. Siksi testitapaukset täytyy priorisoida. Tärkeimmät testit tehdään ensin siirtyen vähitellen vähemmän tärkeisiin. Näin mahdollisimman moni kriittinen vika on löytynyt, jos aika loppuu kesken. (Spillner ym. 2014, 27.)

Lopetuskriteerien arvioinnissa ja raportoinnissa testauksen kohdetta arvioidaan suunnittelussa määrättyjä lopetuskriteerejä vasten. Jos kriteerit on saavutettu, testaaminen voidaan lopettaa. Mahdollisesti päätetään, että lisätestitapauksia täytyy ajaa. Voidaan myös todeta kriteerien olleen liian kovia. (Spillner ym. 2014, 28.)

Vähintään yhden lopetuskriteerin jäädessä täyttämättä kaikkien testien tultua ajetuksi, tarvitaan lisää testejä. Uusien testitapausten pitäisi kattaa täyttämätön lopetuskriteeri paremmin. Voidaan todeta, että ei kannata tehdä lisätyötä lopetuskriteerin täyttämiseksi, jolloin lisätestaaminen peruutetaan. (Spillner ym. 2014, 28.)

Joskus täytyy muuttaa testisuunnitelmaa koska tarvitaan lisäresursseja. Toisinaan täytyy parantaa testimääritelmiä, jotta lopetuskriteerit voidaan täyttää. Kattavuuskriteerien lisäksi voidaan käyttää muita kriteerejä, jotta voidaan määritellä testauksen lopettamisen hetki, kuten häiriötiheys (failure rate). (Spillner ym. 2014, 29.)

Käytännössä testauksen lopun ajankohdan määräävät usein aika ja kustannukset. Testin lopuksi pitäisi kirjoittaa yhteenveto sidosryhmille, joita ovat projektipäällikkö, testauspäällikkö ja mahdollisesti asiakas. (Spillner ym. 2014, 30.)

Testauksen lopetustoimien vaiheessa analysoidaan projektista saatu kokemus ja arkistoidaan se seuraaville projekteille. Esimerkiksi seuraavia tietoja pitäisi kerätä. Milloin järjestelmä julkaistiin? Milloin testaus lopetettiin? Milloin virstanpylväs saavutettiin tai ylläpitojulkaisu tehtiin? Mitkä suunnitellut tulokset saavutettiin ja milloin? Mitä odottamattomia tapahtumia tuli? Syyt ja kuinka ne kohdattiin. Onko jäljellä avoimia ongelmia ja muutospyyntöjä? Miksi niitä ei ole toteutettu? Jatkuvaa kehitystä voidaan saada aikaan, jos havaitut parannusmahdollisuudet otetaan huomioon tulevissa projekteissa. (Spillner ym. 2014, 30 – 31.)

Testauksen materiaalit (testware) kannattaa säilyttää tulevaa varten. Tähän kuuluvat testitapaukset, lokit, infrastruktuuri, työkalut jne. Ohjelmistojen käytön aikana löytyy häiriöitä, joita ei löytynyt testauksessa ja asiakkaat pyytävät ohjelmistoon muutoksia. Siksi ohjelmistoa muutetaan ja silloin sitä pitää testata uudelleen. Ylläpitoa helpottaa, jos testauksen materiaalit ovat saatavilla. (Spillner ym. 2014, 31.)

Testaamisen jättämistä kehittäjien tehtäväksi ei suositella, koska omia virheitä ei huomata tarpeeksi helposti. Siksi on parempi käyttää erillisiä testaajia. Itsenäiset testaajat ovat harhattomia ja löytävät siksi enemmän ja erilaisia vikoja, kuin kehittäjät. Itsenäinen testaaja voi verifioida kehittäjien järjestelmän määrittelyn ja toteutuksen aikana tekemiä mahdollisesti implisiittisiä olettamuksia. Liiallinen eristyneisyys voi haitata kommunikaatiota testaajien ja kehittäjien välillä. Resurssien puutteessa itsenäisestä testauksesta voi tulla pullonkaula. (Spillner ym. 2014, 169.)

Testauksen itsenäisyydelle on viisi mahdollisuutta (Spillner ym. 2014, 170):

1. Kehitystiimi voi vastata testauksesta, mutta kehittäjät testaavat toistensa ohjelmat, eli kehittäjä testaa kollegansa ohjelman.
2. Testaajat voivat kuulua kehitystiimiin ja tehdä kaiken testaustyön.
3. Projektitiimin sisällä voi olla ainakin yksi testaukselle omistettu tiimi.
4. Itsenäisiä testausspesialisteja voidaan käyttää tiettyihin testaustehtäviin, kuten suorituskykytestaus, käytettävyystestaus, turvallisuustestaus tai näyttämään standardien ja säännösten noudattaminen.
5. Erillinen organisaatio, kuten testausosasto, ulkopuolinen testauslaitos, tai testilaboratorio voi hoitaa testauksen tai tärkeitä osia siitä, kuten järjestelmätestauksen.

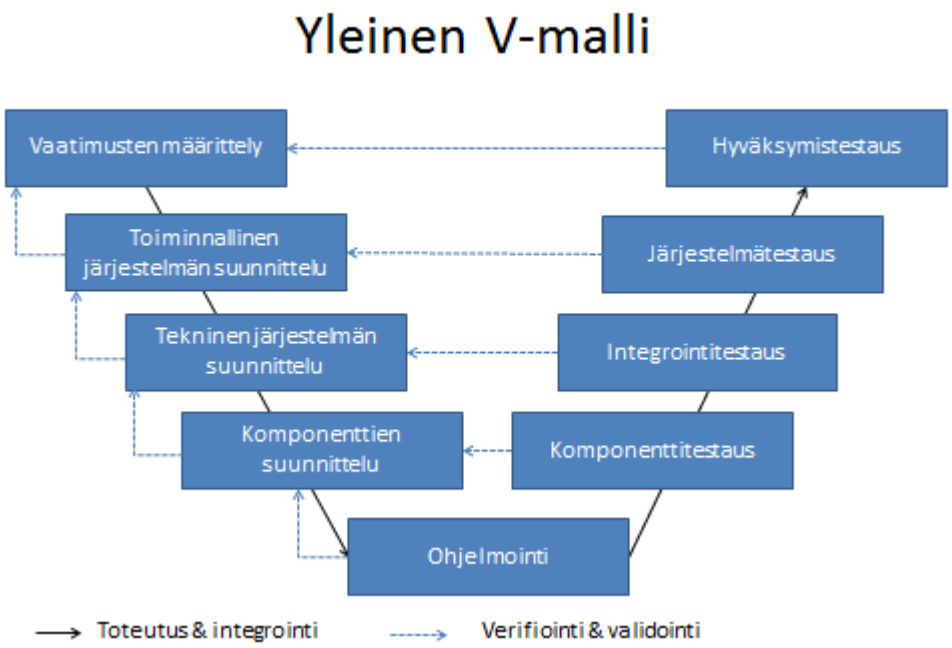
Komponenttitestauksessa testauksen pitäisi olla lähellä kehitystä. Pitäisi käyttää joko mallia 1 tai 2. Malli 2 on hyödyllinen, jos riittävä määrä testaajia on saatavilla. (Spillner ym. 2014, 170.)

Integrointitestauksessa, jos sama tiimi, joka kehittää komponentit, myös tekee integroinnin ja testauksen, voidaan testaus organisoida samoin kuin komponenttitestauksessa. Jos integroidaan useamman tiimin tekemiä komponentteja, pitäisi olla vastuussa sekoitettu integrointitiimi, jossa on mukana edustajia kehitystiimeistä tai itsenäinen integrointitiimi. Kehitysprojektin koosta ja komponenttien määrästä riippuen pitäisi harkita malleja 3, 4 ja 5. (Spillner ym. 2014, 171.)

Järjestelmätestauksessa katsotaan tuotetta asiakkaan ja käyttäjän näkökulmasta. Siitä syystä itsenäisen testaustiimin käyttäminen on elintärkeää. Siis vain mallit 3, 4 ja 5 ovat mahdollisia. (Spillner ym. 2014, 171.)

3.3 Testaustasot, riskiohjattu testaus ja testaussuunnittelu

Spillner ym. (2014, 40) antavat yleisen V-mallin testaukselle. Kuvio 1 kuvaa sitä. Malli on ohjelmistokehityksen vesiputousmallin laajennus. Lisäksi perinteiseen V-malliin verrattuna siinä on yksi uusi osa, ohjelmointi.



Kuvio 1. Yleinen V-malli testaukselle. (Spillner ym. 2014, 40)

Yleisessä V-mallissa vaatimusten määrittelyssä (requirements definition) kerätään, määritellään ja vahvistetaan asiakkaan tarpeet ja vaatimukset tehtävälle järjestelmälle. Määritellään siis järjestelmän tarkoitus ja halutut ominaisuudet. Toiminnallinen järjestelmän suunnittelu (functional system design) on seuraava vaihe. Siinä määritellään toiminnot ja dialogit uudelle järjestelmälle edellisessä vaiheessa luotujen vaatimusten pohjalta. Teknisessä järjestelmän suunnittelussa (technical system design) luodaan järjestelmän toteutus. Siinä suunnitellaan rajapinnat järjestelmän ympäristöön ja pienemmät ymmärrettävät alajärjestelmät, eli järjestelmän arkkitehtuuri. Jokainen alajärjestelmä määritellään komponenttien suunnittelussa. Sille määrätään tehtävä, käyttäytyminen, sisäinen rakenne ja rajapinnat muihin alajärjestelmiin. Ohjelmointi on kuvassa alimmainen vaihe. Ohjelmoinnissa luodaan jokainen komponentti (moduli, yksikkö, luokka) ohjelmointikielellä. (Spillner ym. 2014, 40.)

Ensimmäinen vaihe testauksen puolella on komponenttitestaus (component test). Siinä varmistetaan, että jokainen komponentti täyttää määrittelynsä. (Spillner ym. 2014, 41.) Komponentit voivat olla yksittäisiä aliohjelmia, alirutiineja, luokkia tai proseduureja (Myers ym. 2012, 85). Toisena vaiheena testauksessa on integrointitestaus. Siinä varmistetaan, että komponenttijoukot vaikuttavat toisiinsa, kuten teknisessä järjestelmän suunnittelussa on määrätty. Järjestelmätestaus on kolmas vaihe. Siinä varmistetaan, että järjestelmä kokonaisuutena täyttää määrätyt vaatimukset. Neljäs ja viimeinen testausvaihe on hyväksymistestaus. Siinä tarkistetaan, täyttääkö järjestelmä asiakkaan sopimuksessa määrätyt vaatimukset ja/tai täyttääkö järjestelmä käyttäjän tarpeet ja odotukset. (Spillner ym. 2014, 41.)

Testaajan täytyy työssään tehdä kuvassa esitettyä verifiointia ja validointia. Validoinnissa tarkistetaan, että on kehitetty tuote, joka vastaa vaatimuksia juuri sillä abstraktion tasolla, jota tarkastellaan. Verifioinnissa tarkastetaan, että tietyn kehitystason tuotos on määrittelyihinsä verrattuna oikea. Verifioinnissa tarkastellaan, ovatko määrittelyt tulleet oikein toteutettua ja täyttääkö tuote määrittelynsä, mutta ei sitä, sopiiko tuote sen tarkoitettuun käyttöön. Käytännössä jokainen testi sisältää sekä validointia että verifiointia. Korkeammilla testaustasoilla validoinnin osuus kasvaa. (Spillner ym. 2014, 41 – 42.)

V-mallista voi saada käsityksen, että testaus alkaisi suhteellisen myöhään järjestelmän toteuttamisen jälkeen. Niin ei kuitenkaan ole. Mallin oikean haaran tasot pitäisi tulkita testauksen toteuttamisen tasoiksi. Testauksen valmistelu, (testisuunnittelu, testauksen analyysi ja suunnittelu) alkavat aikaisemmin ja tehdään rinnakkaisesti vasemman haaran kehitysvaiheiden kanssa. (Spillner ym. 2014, 42.)

Testauksen eri vaiheilla on erilaiset tavoitteet. Siitä syystä testauksen eri vaiheissa tarvitaan erilaisia menetelmiä ja erilaisia työkaluja sekä henkilökuntaa, joilla on monia erilaisia tietoja ja taitoja. (Spillner ym. 2014, 42.)

Ohjelmoijalla on tietenkin pääsy tekemäänsä ohjelmakoodiin, kuten myös yksikkötestin mahdollisesti tekevällä toisella testaajalla. Siksi siinä voi käyttää lasilaatikkotestausmenetelmiä. (Spillner ym. 2014, 48.)

Mette Jonassen Hass (2008, 9-10) käyttää yksikkötestauksesta nimeä komponenttitestaus (component testing), samoin Spillner ym. (2014, 41 – 49). Myers ym. (2012, 85 -111) kutsuvat sitä moduulitestaukseksi (module testing). Yleisessä komponenttitestaussuunnitelmassa pitäisi määritellä, missä järjestyksessä komponentit testataan. Jos se tehdään tarpeeksi aikaisin, kriittisimmät komponentit voidaan testata ensin. Testauksen kohde voi olla koostettu useammasta muusta komponentista. Tällöinkään ei testata komponenttien välistä vuorovaikutusta vaan komponentin sisäistä toimintaa. (Spillner ym. 2014, 43.)

Komponentin eristämisessä täytyy käyttää erillistä ajuria komponentin käyttämiseksi. Järjestelmän muiden osien simuloimiseksi käytetään tynkiä (stub) (Mette Jonassen Hass 2008, 11). Testiympäristön luomiseksi tarvitaan ohjelmointitaitoja ja tietoa testattavasta komponentista. Siksi kehittäjät yleensä testaavat oman tuotteensa. (Spillner ym. 2014, 45.) Komponentin testaus voi alkaa, kun kehittäjä sanoo sen olevan valmis eli täyttävän aloituskriteerit, eli se vähintään kääntyy (Mette Jonassen Hass 2008, 11).

Komponenttitestauksessa on tiettyjä tavoitteita. Ensinnäkin testataan, että testin kohde täyttää määrittelynsä oikein ja kokonaan eli tehdään toiminnallinen testaus. Siinä pitäisi myös testata kohteen robustisuus, eli että se toimii myös yllättävillä syötteillä. Kolmanneksi pitäisi testata kohteen tehokkuus, etteivät tehokkuusongelmat yllätä järjestelmätestauksessa tai hyväksymistestauksessa, jolloin niiden korjaaminen on jo kalliimpaa. Neljänneksi komponenttitestauksessa voidaan testata ylläpidettävyyttä, eli koodin rakennetta, modulaarisuutta, kommentointia, standardienmukaisuutta, ymmärrettävyyttä ja dokumentointia. (Spillner ym. 2014, 46 – 48.) Jokaisesta komponentista pitäisi kirjoittaa hyvin lyhyt raportti (Mette Jonassen Hass 2008, 11).

Yksikkötestauksen jälkeinen työvaihe on integrointitestaus (integration testing), jossa on tarkoitus saada järjestelmä toimimaan kokonaisuutena ja siksi liitetään osia yhteen inkrementaalisesti. Siinä uusi komponentti liitetään osaksi testattua kokonaisuutta. Uuden osan sisältäessä vielä olemassa olemattomia kytkentöjä muualle ohjelmaan, sitä varten luodaan tynkiä (stub) eli sijaiskomponentteja, joiden avulla järjestelmä saadaan toimimaan, vaikka siinä ei vielä olekaan kaikkia toimintoja. Tavoite on kokeilla järjestelmän osien toimivan yhdessä, ei enää yksittäisiä komponentteja. Testitapaukset ovat laajempia, kuin yksikkötestauksessa, mutta eivät vielä koko järjestelmää koskevia. Integrointitestauksessa lisätään todistetusti toimivaan kokonaisuuteen aina yksi osa lisää ja tarkastetaan kokonaisuuden yhä toimivan. (Kasurinen 2013, 54.)

Integrointitestauksessa testataan komponenttien välisiä rajapintoja. Oikea tietokantojen käyttö ja muiden infrastruktuurin komponenttien käyttö täytyy myös testata. Integrointitestausta varten pitää rakentaa testiympäristö. Siihen tarvitaan mm. testiajureita, jotka voivat olla valmiina komponenttitestauksesta. Tarvitaan myös ns. monitoreita, jotka ovat komponenttien välistä datasiirtoa lukevia ja tallentavia ohjelmia. Monitoreita standardiprotokollille on kaupallisesti saatavilla. Projektikohtaiset kehitettävän ohjelmiston sisäistä datasiirtoa tutkivat monitorit täytyy ohjelmoida. (Spillner ym. 2014, 52 -53.)

Vaikeampia löytää ovat ongelmat, jotka liittyvät komponenttien ajamiseen. Ne voidaan löytää vain dynaamisella testaamisella. Ne ovat ongelmia datan siirrossa tai kommunikaatiossa kahden komponentin välillä. Myös ei-toiminnallisia testejä voidaan ajaa integraatiotestauksessa. Näin voidaan testata luotettavuutta, suorituskykyä ja kapasiteettia. (Spillner ym. 2014, 53 – 54.)

Integrointitestausta varten pitäisi tuottaa integrointitestaussuunnitelma, jossa mm. määrätään, missä järjestyksessä testaus suoritetaan. Testausjärjestykseen on neljä vaihtoehtoista strategiaa, jotka ovat ylhäältä alas (top down), alhaalta ylös (bottom up), funktionaalinen integraatio (functional integration) ja iso pamaus (big bang). (Mette Jonassen Hass 2008, 12.)

Spillner ym. (2014, 57) listaavat lisäksi ad hoc –integraation ja selkärankaintegraation (backbone integration).

Ylhäältä alas –strategiassa ohjelman hierarkiassa ylimpänä olevat rajapinnat testataan ensin siirtyen alaspäin. Pääohjelma toimii ajurina. Tällä tavalla saadaan nopeasti luotua kuori. Haittapuolena on, että tarvitaan suuri määrä tynkiä. (Mette Jonassen Hass 2008, 12.)

Alhaalta ylös –integroinnissa testataan ensin alimman tason rajapinnat. Ylemmän tason komponentit korvataan ajureilla, joten niitä tarvitaan paljon. Tässä voidaan integroida aikaisin laitteistoon, jos se on tarpeen. (Mette Jonassen Hass 2008, 13.)

Funktionaalisessa integroinnissa osat integroidaan toimintoalueen mukaan. Tämä toimii kuin vertikaalisesti jaettu ylhäältä alas –strategia. Näin on mahdollista toteuttaa nopeasti toiminnalliset alueet. (Mette Jonassen Hass 2008, 13.)

Pienempiä ohjelmia voidaan testata laittamalla kaikki osat yhteen kerralla, jolloin puhutaan ”iso pamaus” –integroinnista (big bang) (Bourque & Fairley 2014, 86). Tässä on hyvin vaikeaa löytää vikoja rajapinnoista, kun kaikki ovat käytössä yhtä aikaa (Mette Jonassen Hass 2008, 13). Kaikki häiriöt tapahtuvat samaan aikaan (Spillner ym. 2014, 57 – 58).

Ad hoc –integraatiossa komponentit integroidaan siinä järjestyksessä, kuin ne valmistuvat. Tämä säästää aikaa, koska kaikki komponentit integroidaan testiympäristöön niin nopeasti kuin mahdollista. Haittana on, että tarvitaan tynkiä ja testiajureita. (Spillner ym. 2014, 57.)

Selkärankaintegraatiossa rakennetaan luuranko tai selkäranka ja komponentit integroidaan siihen yksi kerrallaan. Komponentit voidaan integroida missä järjestyksessä vain. Haittana vaaditaan mahdollisesti työvaltainen luuranko tai selkäranka. (Spillner ym. 2014, 57.)

Integraatiotestaus voidaan tehdä välittömästi sen jälkeen, kun integroitavat yksiköt ovat valmiit. Kaikkien komponenttien valmistumista ei tarvitse odottaa ennen kuin integrointitestaus voi alkaa. (Mette Jonassen Hass 2008, 13.)

Integraatiotestaus lopetetaan, kun jokaiselle rajapinnalle määritellyt lopetuskriteerit ovat täyttyneet. Jokaiselle testatulle rajapinnalle pitäisi tehdä lyhyt testausraportti. Integrointitestauksesta pitäisi tuottaa lopuksi yhteenvetoraportti. (Mette Jonassen Hass 2008, 13 – 14.)

Integrointitestauksen loppuun saattamisen jälkeen tulee järjestelmätestaus (system testing). Siinä tarkoitus on löytää vikoja järjestelmän ominaisuuksista. (Mette Jonassen Hass 2008, 14.) Yksikkötestatut komponentit on koottu toimivaksi kokonaisuudeksi integrointitestauksessa.

Sitten järjestelmätestauksessa testataan kokonaista järjestelmää. Siinä ei ole enää mukana integrointitestauksen tynkiä. Tavoite on varmistaa järjestelmän toteuttavan kaikki sen tavoitteet ja toimivan kokonaisuutena. Siinä voidaan käyttää mustalaatikko- tai lasilaatikkotestausta, käyttäjätestausta, kuormitustestausta, tutkivaa testausta tai mitä tahansa muuta toiminnallista kokonaisuutta tarkastelevaa menetelmää. Testaus suoritetaan testiympäristössä, ei vielä lopullisessa kohdeympäristössä. (Kasurinen 2013, 56-57.) Järjestelmätestausta voidaan käyttää ei-toiminnalliseen vaatimusten testaukseen, kuten turvallisuuden, nopeuden, tarkkuuden ja luotettavuuden testaukseen. Liittymiä ulkoisiin sovelluksiin, palveluihin, laitteistoihin ja käyttöympäristöihin voidaan testata. (Bourque & Fairley 2014, 86.)

Järjestelmää testataan tuotantoympäristöön nähden mahdollisimman samanlaisessa ympäristössä. Ympäristöön pitäisi asentaa ajurien ja tynkien sijaan samat laitteistot ja ohjelmistot, jotka ovat tuotantoympäristössä. (Spillner ym. 2014, 59 – 60.)

Tarkempi testin tavoite on tarkastaa täyttääkö kokonainen järjestelmä määritellyt toiminnalliset ja ei-toiminnalliset vaatimukset ja kuinka hyvin. Häiriöt väärässä, epätäydellisessä tai yhtäpitämättömässä vaatimusten toteutuksessa pitäisi löytää. Jopa dokumentoimattomat ja unohtuneet vaatimukset pitäisi löytää. Järjestelmän lisäksi tässä vaiheessa testataan käyttäjien dokumentaatio, kuten manuaalit, opetusmateriaalit yms. Datan laatua täytyy yhä useammin testata tietokantaa tai suurta datamäärää käyttävissä järjestelmissä. Datasta tarkistetaan, että se on yhdenmukaista, kokonaista ja ajantasaista. Jos vaatimuksia ei ole kirjoitettu, ne saattavat löytyä käyttäjien tai asiakkaan pääkopasta. Testaajien pitää sitten saada tietoa vaaditusta käyttäytymisestä. Yksi mahdollinen tekniikka yrittää selviytyä on tutkiva testaus (exploratory testing). (Spillner ym. 2014, 59 - 60.)

Hyväksymistestaus (acceptance testing) on V-mallin viimeinen vaihe. Päätavoitteena on osoittaa, että ohjelma täyttää vaatimusmäärittelyssä asetetut vaatimukset. Enää ei etsitä vikoja vaan sen odotetaan toimivan. Testauksen kohteena on koko järjestelmä, joka tarkastetaan virallisesti. Asiakas hyväksyy tuotteen valmistuneeksi. Onnistuneen hyväksymistestauksen jälkeen ohjelmisto siirtyy asiakkaan omaisuudeksi. Hyväksymistestauksessa järjestelmää käytetään sen kohdeympäristössä. Tämä vaihe voi sisältää tai voi olla sisältämättä ohjelman valmistajan. Installointitestauksessa (installation testing) tuote asennetaan asiakkaan järjestelmään kokeiltavaksi siinä. (Bourque & Fairley 2014, 86 – 87; Kasurinen 2013, 57; Mette Jonassen Hass 2008, 15.)

Käytetyt tekniikat ovat useimmin kokemukseen perustuvia, joilla tulevat käyttäjät kokeilevat tuotetta validoidakseen sen. Testaukseen saapuvat käyttäjät hyötyvät testauksesta, koska he saavat yksityiskohtaisen käsityksen tuotteesta. (Mette Jonassen Hass 2008, 15.)

Sopimushyväksymistestiä (contract acceptance test) sanotaan myös tehdashyväksymistestiksi (factory acceptance test). Sopimuksessa täytyy olla selkeät hyväksymiskriteerit ohjelmistolle. Testaustulokset täytyy rekisteröidä tarkasti todisteiksi siitä, mihin asiakkaan hyväksyntä perustuu. (Mette Jonassen Hass 2008, 15 - 16.) Käytännössä järjestelmän valmistaja on kokeillut kriteerejä jo järjestelmätestauksessa, joten hyväksymistestauksessa täytyy vain samat testit uudelleen ajamalla näyttää järjestelmän toimivan. Asiakkaan täytyy olla ollut mukana testitapauksien suunnittelussa tai ainakin kunnolla tarkastanut ne, koska valmistaja voi olla ymmärtänyt kriteerit väärin. Hyväksymistesti voi yllättäen epäonnistua, koska järjestelmätesti ja hyväksymistesti tehdään eri ympäristöissä. (Spillner ym. 2014, 62 – 63.)

Alfatestauksessa käyttäjien edustajat käyttävät järjestelmää kehittäjän tiloissa, mutta todellisen kaltaisessa järjestelmässä. Käyttäjätuki täytyy järjestää. Alfatestaus on kallista. Beetatestauksessa valitut tai vapaaehtoiset asiakkaat käyttävät tuotetta asiakkaan tiloissa. Tuotetta käytetään kuten tuotannossa (Mette Jonassen Hass 2008, 16.)

Neljäs mahdollinen testitapa on käyttäjän hyväksyntätestaus (user acceptance testing), jota suositellaan käytettäväksi, jos asiakas ja käyttäjä ovat eri tahot. Käyttäjän mielestä järjestelmä voi olla hankala käyttää vaikka se olisi kelvollinen toiminnallisesta näkökulmasta. Siksi pitäisi järjestää hyväksymistestaus jokaiselle käyttäjäryhmälle. (Spillner ym. 2014, 63 – 64.)

Viides testaustapa on käyttöön soveltuvuuden hyväksymistestaus (operational acceptance testing). Siinä hankitaan hyväksyntä järjestelmän hallinnoijilta. Siihen voivat kuulua varmistuskopionti ja tietojen palautus, katastrofista palautuminen, käyttäjien hallinnointi ja turvallisuushaavoittuvuuksien testaus. (Spillner ym. 2014, 64.)

Riskiohjatussa testauksessa käytetään tietoa projektin ja tuotteen riskeistä ja ohjataan testausta korkean riskin alueisiin. Riski määritellään kahden tekijän matemaattiseksi tuloksi. Tekijät ovat tappio tai vahinko, joka aiheutuu järjestelmän häiriöstä sekä sen todennäköisyys tai frekvenssi. Todennäköisyys riippuu siitä, kuinka tuotetta käytetään. Riskin tekijöitä tulee projektista (projektiriski) ja tuotteesta (tuoteriski). (Spillner ym. 2014, 186 – 187.)

Riskiohjattu testien priorisointi varmistaa, että riskaabelit tuotteen osat testataan intensiivisemmin ja aikaisemmin, kuin vähemmän riskaabelit. Vakavat ongelmat, jotka aiheuttavat paljon korjaustyötä ja vakavia viivästyksiä, löydetään aikaisin. (Spillner ym. 2014, 189.)

Ohjelmiston testaus on suuri suunnitelmallisuutta vaativa työ. Testauksen suunnittelu ja valmistelu pitäisi aloittaa aikaisin. Testisuunnitelmaan vaikuttavat organisaation testauspolitiikka, projektin tavoitteet, riskit ja rajoitteet sekä tuotteen kriittisyys. (Spillner ym. 2014, 174.)

Testauspäällikölle mahdollisia suunnitteluaktiviteetteja ovat seuraavat. Ensiksikin, hän osallistuu testauksen yleisen lähestymisen ja strategian määrittelyyn. Toisena on testiympäristöstä ja testiautomaatiosta päättäminen. Kolmantena tulee testaustasojen ja niiden interaktion määrittely sekä testausaktiviteettien integrointi muuhun projektiin. Neljäntenä on testitulosten arvioinnista päättäminen. Viidenneksi, hän voisi osallistua mittarien valitsemiseen testauksen valvontaan ja kontrollointiin sekä lopetuskriteerien määrittämiseen. Kuudentena on päättäminen siitä, kuinka paljon testausdokumentaatiota kirjoitetaan ja malleista päättäminen. Seitsemäntenä on testaussuunnitelman kirjoittaminen ja päättäminen siitä, kuka testaa mitä, milloin ja kuinka paljon. Kahdeksantena on testauksen työmäärän ja kustannusten arviointi sekä testaustehtävien uudelleenarviointi ja uudelleensuunnittelu myöhemmän testaustyön aikana. (Spillner ym. 2014, 175 – 176.)

Testaussuunnitelma tehdään kirjallisesti. Valmis pohja on saatavilla IEEE:n standardissa 829-1998. Sen pääkohdat ovat seuraavat: testisuunnitelman tunniste, esittely, testattavat nimikkeet, testattavat ominaisuudet, ominaisuudet, joita ei testata, lähestymistapa, testin lopetusehdot, keskeytyskriteerit (suspension criteria) ja uudelleenaloitusehdot (resumption requirements), testauksen tuotokset, testaustehtävät, ympäristön tarpeet, vastuut, henkilöstönhallinta ja valmennustarpeet, aikataulu, riski ja epävarmuudet sekä hyväksyntä. Testisuunnitelmaa täytyy ylläpitää koko kehitysprosessin ajan perustuen testausaktiviteettien palautteeseen ja muuttuviin riskeihin. (Spillner ym. 2014, 176 - 177.)

Suunnitelmallisuudesta huolimatta aika ja budjetti eivät välttämättä riitä kaikkien suunniteltujen testien tekemiseen. Silloin täytyy vähemmällä testimäärällä löytää mahdollisimman paljon kriittisiä vikoja. Siis testitapauksia täytyy priorisoida niin, että minkä tahansa testin päättyessä ennenaikaisesti, paras testaustulos on saavutettu. Tärkeimmät ongelmat löydetään aikaisin. Kriteerejä ovat toiminnon käyttötiheys, häiriön todennäköisyys, häiriön riski, häiriön näkyvyys ja vaatimusten tärkeysjärjestys. Komponentit, joiden häiriöillä on vakavat seuraukset järjestelmän kannalta pitäisi testata erityisen tarkasti. (Spillner ym. 2014, 177 - 178.)

Yksittäisten komponenttien ja järjestelmän osien monimutkaisuus voi aiheuttaa ohjelmointivirheitä. Toisaalta yksinkertaisina pidettyjen ohjelman osien tekemiseen ei ole välttämättä suhtauduttu tarpeeksi huolellisesti. Korkean projektiriskin omaavat viat pitäisi löytää aikaisin. Nämä ovat vikoja, joiden korjaaminen vaatisi huomattavaa työmäärää ja, jotka johtaisivat huomattaviin projektin viivästymisiin. Priorisoinnin kriteerit ja prioriteettiluokat tulevat testaussuunnitelmaan. Jokainen testitapaus pitäisi priorisoida kriteerien mukaan. Siten ongelmien tullessa voidaan päättää, mitkä testitapaukset jätetään pois. (Spillner ym. 2014, 178 – 179.)

Aloitus- ja lopetusehtojen määrittäminen on tärkeä osa testaussuunnittelua. Ne määrittävät, milloin testaus voidaan aloittaa ja lopettaa kokonaan tai testitasolla. Lopetusehdot varmistavat, että testausta ei lopeteta ennenaikaisesti esim. ajanpuutteen tai puuttuvien resurssien vuoksi. Ne myös estävät testauksen muodostumisen liian suureksi. Testauspäällikkö määrittelee projektikohtaiset lopetuskriteerit testisuunnitelmassa. Testien aikana kriteerejä mitataan säännöllisesti ja ne toimivat pohjana päätöksille testauksen ja projektin johdolle. (Spillner ym. 2014, 179 – 180.)

3.4 Toiminnallinen ja ei-toiminnallinen testaus

Toiminnalliseen testaukseen (functional testing) kuuluvat kaikki testit, joilla varmistetaan järjestelmän syöte/tuloste-käyttäytymistä. Toiminnallisten testien suunnitteluun käytetään mustalaatikkotestauksen menetelmiä. Testauksen lähdedokumenttina ovat toiminnalliset vaatimukset, jotka määrittelevät järjestelmän käyttäytymisen. (Spillner ym. 2014, 70.)

Ei-toiminnalliset (non-functional) vaatimukset eivät kuvaa järjestelmän toimintoja. Ne kuvaavat toiminnallisen käyttäytymisen tai järjestelmän piirteitä. Ne kuvaavat, kuinka hyvin järjestelmä tai sen osa toimii. Niiden toteutuksella on suuri vaikutus asiakkaan ja käyttäjän tyytyväisyyteen ja siihen, kuinka paljon he nauttivat tuotteen käytöstä. (Spillner ym. 2014, 72.) Näiden ominaisuuksien testaaminen on ei-toiminnallista testausta.

3.5 Testaustekniikat ja –tavat

Testausympäristö koostuu laitteistoista ja ohjelmistoista, joiden päällä testit ajetaan. Ympäristö täytyy määritellä ennen, kuin voidaan määritellä, kuinka testit ajetaan. Laitteistoihin kuuluvat tietokoneet, joilla testit tehdään, testituloksia tarkkailevat laitteet, laitteistojen simulointiohjelmat, joiden päällä ohjelmaa ajetaan ja laitteet, jotka vuorovaikuttavat ohjelman kanssa testissä. Ohjelmistoja ovat testattava ohjelma, simulointiohjelmat, testiä tarkkailevat ohjelmistot, tuloksia tulkitsevat ohjelmistot ja käyttöjärjestelmät. (Fournier 2009, 160.)

Testausmenetelmissä erotetaan staattinen ja dynaaminen testaaminen. Staattisessa testauksessa järjestelmää ei käytetä, vaan sitä tutkitaan esim. koodianalysaattoreilla, koodiarvioinneilla tai arkkitehtuurisuunnittelun näkökulmasta. Siinä pyritään poistamaan perustason tarkastuksilla ilmiselvät ongelmat tuotteesta. Dynaamisessa testauksessa testattavaa järjestelmää jo käytetään ja seurataan miten se reagoi annettuihin syötteisiin. Useimmat testausmenetelmät ovat dynaamisia. Staattisessa testauksessa löydetyt viat ovat halvempia korjata kuin dynaamisessa testauksessa löydetyt. (Bourque & Fairley 2014, 82; Kasurinen 2013, 65.)

Staattisissa testeissä erotetaan strukturoidut ryhmäarvioinnit ja staattinen analyysi. Strukturoidun ryhmäarvioinnin tekevät kokonaan ihmiset, staattiseen analyysiin käytetään analyysiohjelmaa. (Spillner ym. 2014, 79 – 95.)

Dynaamiseen testaukseen kuuluvat mustalaatikkotestaus ja lasilaatikkotestaus. Mustalaatikkotestauksessa (black box testing) annetaan ohjelmalle syötteitä ja katsotaan, kuinka ohjelma reagoi. Ohjelman sisäistä toimintaa ei tarkastella. Testaaja kirjoittaa testinsä perustuen vaatimuksiin. Menetelmää voidaan käyttää aina, kun on olemassa käynnistyvä ohjelma eli kaikissa V-mallin testausvaiheissa: yksikkö-, integrointi-, järjestelmä- ja hyväksymistestauksessa. Siinä kokeillaan käyttötapauksia, tiettyihin tehtäviin liittyviä toimintosarjoja. Käyttötapaukset kirjoitetaan sen pohjalta, mitä ohjelman pitäisi toimintosarjoissa tehdä. Siinä voidaan tarkastella virheellisten ja haitallisten syötteiden vaikutusta toimintaan, kuten ylisuurien lukujen tai SQL-komentojen. Mustalaatikkotestauksessa pyritään todistamaan testattavan toiminnallisuuden toimivan oikein odotettavissa tilanteissa. Siksi tulisi käyttää mahdollisimman monta kombinaatiota kaikista mahdollisista tapauksista. Ei-toivottujen lopputuloksien tapauksissa täytyy raportoida poikkeuksista ohjelmistokehittäjille, joiden tehtävä on korjata asia. (Fournier 2009, 9; Kasurinen 2013, 65-66.)

Mustalaatikkotestauksessa pyritään tuottamaan sellaiset testitapaukset, joilla on suurin mahdollisuus havaita eniten vikoja. Testitapauksen pitäisi vähentää tarvittavien toisten testitapauksien määrää enemmän, kuin yhdellä. Sen pitäisi myös kattaa suuri määrä muita mahdollisia testitapauksia eli kertoa jotakin vikojen olemassaolosta siinä olevien syötteiden edustamassa joukossa muita syötteitä. (Myers ym. 2012, 50)

Näihin vaatimuksiin pyritään vastaamaan ekvivalenssijakomenetelmällä (equivalence partitioning). Siinä jaetaan ohjelman syötealue joukkoon ekvivalenssiluokkia (equivalence class) niin, että luokkaa edustavan arvon voidaan odottaa olevan samanarvoinen kaikkiin muihin luokkaa edustaviin arvoihin verrattuna. Jos yksi testiarvo tuottaa häiriön, voidaan odottaa kaikkien muidenkin luokkaan kuuluvien arvojen tuottavan saman häiriön eli löytävän saman vian. Myös, jos häiriötä ei testitapauksella tule, niin myöskään muiden luokkaan kuuluvien testitapauksien ei pitäisi tuottaa häiriötä. (Myers ym. 2012, 50.)

Testitapaukset, jotka tutkivat rajatilanteita, löytävät muita enemmän vikoja. Rajaarvoanalyysi (boundary value analysis) perustuu tähän. Siinä haetaan ainakin yksi elementti ekvivalenssiluokan rajalta. Syötearvojen lisäksi testitapauksia haetaan myös kiinnittämällä huomiota tulosalueisiin. (Myers ym. 2012, 55.) Robustisuuden testaamisessa testitapaukset valitaan muuttujien arvojen rajojen ulkopuolelta, jotta tiedettäisiin, kuinka ohjelma käyttäytyy odottamattomien tai väärien syötteiden kohdalla (Bourque & Fairley 2014, 89).

Virheen arvauksessa (error guessing) testaaja käyttää intuitiotaan ja kokemustaan löytääkseen todennäköisiä vikoja ja kirjoittaa testitapauksia niiden paljastamiseksi. Se on eräänlainen ad-hoc –prosessi. Periaate on tehdä lista virheille alttiista tilanteista ja kirjoittaa sen pohjalta testitapauksia. (Myers ym. 2012, 81.) Tietolähteinä voidaan käyttää aikaisemmissa projekteissa ilmenneitä virheitä ja testaajan kokemusta (Bourque & Fairley 2014, 90).

Lasilaatikkotestauksessa eli valkolaatikkotestauksessa (glass box testing, white box testing) tarkastellaan järjestelmän sisäistä toimintaa testin aikana. Päinvastoin kuin mustalaatikkotestauksessa, tässä siis tunnetaan ohjelman lähdekoodi. Siinä testaaja pystyy jäljittämään löytämänsä virheet lähdekooditasolle asti. Siten testaajan pitäisi pystyä näyttämään, että vikaa ei ole tai, että tulos ei ole sattuma. Testaajan pitää tuntea järjestelmä niin hyvin, että ohjelman tarkastaminen lähdekooditasolta onnistuu ja järjestelmän logiikkaa niin hyvin, että hän voi varmuudella tietää toimiiko ohjelma oikein. (Kasurinen 2013, 67.) Lasilaatikkotestauksessa ollaan kiinnostuneita siitä, kuinka hyvin testitapaukset kattavat ohjelman logiikkaa eli lähdekoodia. Täydellinen kaikki polut kattava testaaminen ei ole realistinen tavoite testaukselle. (Myers ym. 2012, 42.)

Harmaa laatikko –testaus (grey box testing) on yhdistelmä mustalaatikko- ja lasilaatikkotestausta. Siinä käytetään mustalaatikkotestauksen vaatimusmäärittelystä tehtyjä testitapauksia ja lasilaatikkotestauksen koodin tarkastelua. Menetelmässä testataan mallikattavuus eli että kaikki vaatimukset on täytetty ja koodikattavuus eli lähdekoodin tarkastus. Se voi olla paras testausmenetelmä, jos järjestelmästä tunnetaan lähdekoodin tasolla vain osa. Näin voi olla, jos osa järjestelmästä on oman organisaation ulkopuolelta. (Kasurinen 2013, 68.)

Regressiotestaus ei ole varsinaisesti oma testauksen muotonsa, vaan tarkoittaa uudelleentestaamista. Regressiotestausta on, kun järjestelmän osaa muutetaan ja varmistetaan sen toimivan edelleen oikein. Siitä puhutaan myös silloin, kun on saavutettu osatavoite ja kehitysversiosta varmistetaan kaikkien toimintojen toiminta, myös aikaisemmassa versiossa korjattujen. Siinä varmistetaan, että korjattuja ongelmia ei enää ole eikä mitään uusia vikoja ole korjauksissa ilmaantunut. (Bourque & Fairley 2014, 87; Kasurinen 2013, 68-69.)

Joitakin testejä ei haluta tai voida tehdä ennen kuin tuote on lähes valmis tai kokeiltavissa kaikilla tai useimmilla ominaisuuksilla. Työvaiheita, joita käytetään projektin vaiheessa, jossa järjestelmä toimii mutta sitä ei vielä ole julkaistu tai luovutettu asiakkaalle ovat käytettävyystestaus, kuormitustestaus, suorituskykytestaus, aloitustestaus, alfa- ja betatestaus, tutkiva testaaminen, ad hoc –testaus ja mallipohjainen testaaminen. Käytettävyystestauksessa (usability testing) käsitellään järjestelmän käyttöliittymän toimivuutta ja intuitiivisuutta. Kuormitustestauksessa (load testing, stress testing) ohjelmaa käytetään suunnitellulla käyttäjämäärällä esim. virtuaalikäyttäjiä käyttämällä normaalia toimintaa simuloivia testitapauksia.

Suorituskykytestauksessa (performance testing) pyritään osoittamaan järjestelmän suoriutuvan tehtävistään määrittelyjen mukaisesti. (Kasurinen 2013, 70 – 72.)

Aloitustestissä (smoke test) selvitetään yksinkertaisilla peruskokeilla toimivatko perusasiat järjestelmässä. Siinä käydään läpi tarkastuslistaa, jossa on kohtia, joissa on yksinkertaisia perustoimintoja. Aloitustestissä varmistetaan tietty minimitaso perusasioiden toimimiselle, koska testaajien työaikaa ei kannata tuhlata ohjelmiin, joissa edes perusasiat eivät toimi. (Kasurinen 2013, 72.)

3.6 Havaintoraportin teko ja havaintojen hallinta

Testeissä havaitut häiriöt pitää hoitaa tarpeen mukaan nopeasti. Sitä varten tarvitaan menetelmä havaintojen (incident) kommunikointiin ja hallintaan. Havaintojen hallinta alkaa testilokin arvioinnilla testikierroksen tai testin lopettamisen yhteydessä. (Spillner ym. 2014, 192.)

Testilokista vertaillaan tuloksia odotettuihin tuloksiin. Automatisoiduista testeistä normaalisti työkalu tekee tämän vertailun heti. Jokainen merkittävä odottamaton havainto lokissa analysoidaan, koska se voi olla merkki viasta. Siitä täytyy saada selville syy. Syy voi olla poikkeama odotetusta käyttäytymisestä, väärin suunniteltu testitapaus, virheellinen testin automaatio tai virheellinen testin ajaminen. Virhe voi myös olla testaajan. (Spillner ym. 2014, 193.)

Jos syy on testauskohteessa, kirjoitetaan havaintoraportti (incident report). Havainto voi olla kaksoiskappale aikaisemmasta havainnosta, jolloin tarkistetaan sisältääkö se uutta tietoa, joka voisi tehdä helpommaksi ongelman syyn löytämisen. Havaintoa ei pitäisi kirjata kahdesti. Virheidenpoisto (debugging) on kehittäjien työtä. (Spillner ym. 2014, 193.)

Havaintoraportointia varten luodaan tietokanta, jonne kaikki havainnot ja häiriöt kirjataan. Raportit voivat koskea ongelmia testatuissa ohjelmissa tai puutteita määrittelyissä, käyttöoppaissa tai muissa dokumenteissa. Myös kehittäjät voivat kommentoida raportteja; he voivat esim. pyytää kommentteja tai selvennystä testaajalta tai torjua perusteettoman raportin. Korjaukset testauskohteisiin kirjataan myös tietokantaan, mikä auttaa testaajaa ymmärtämään korjauksen vaikutukset testatakseen kohdetta seuraavalla kierroksella. Testauspäällikkö ja projektipäällikkö saavat havaintotietokannasta käsityksen ongelmien määrästä, tilanteesta ja korjauksista. (Spillner ym. 2014, 193 – 194.)

Havaintoraporttien pitää noudattaa projektinlaajuista muotoa, jotta kommunikointi olisi sujuvaa ja tilastollinen analyysi olisi mahdollista. Testauspäällikön pitäisi määritellä malli esim. testaussuunnitelmassa. Taulukko 2 on esimerkki havaintoraportin mallista. Jos tietokantaa

käytetään hyväksymistestauksessa tai tuotetuessa, täytyy kerätä myös asiakastietoja. (Spillner ym. 2014, 194.)

Määrite	**Tarkoitus**
Tunnistus	
Tunnus/numero	Yksilöllinen tunnus/numero jokaiselle raportille
Testin kohde	Testin kohteen tunniste tai nimi
Versio	Testin kohteen version tunnus
Alusta	Tunnus testiympäristön laitteistolle ja ohjelmistolle, jossa ongelma esiintyy
Raportoija	Raportoivan testaajan tunnistus (mahdollisesti testaustaso)
Vastuullinen kehittäjä	Testin kohteesta vastuullisen kehittäjän tai tiimin nimi
Raportointipäivä	Päiväys ja mahdollisesti aika, jolloin ongelma havaittiin
Luokitus	
Tilanne	Tämän hetken tilanne (ja koko historia) raportin prosessoinnista
Vakavuus	Ongelman vakavuuden luokittelu
Prioriteetti	Korjauksen prioriteetin luokittelu
Vaatimus	Viittaus (asiakkaan) vaatimuksiin, jotka eivät täyty ongelman vuoksi
Ongelman lähde	Projektin vaihe, jossa ongelma syntyi (analyysi, suunnittelu, ohjelmointi); hyödyllinen suunnittelun prosessin parantamistoimille
Ongelman kuvaus	
Testitapaus	Testitapauksen kuvaus (nimi, numero) tai askeleet ongelman uudelleen tuottamiseen
Ongelman kuvaus	Ongelman tai häiriön kuvaus; odotettu vs. toteutunut havainto tuloksista tai käyttäytymisestä
Kommentit	Luettelo kommenteista raportille kehittäjiltä ja muulta henkilökunnalta
Vian korjaus	Kuvaus tehdyistä muutoksista vian korjaamiseksi
Viittaukset	Viittaus toisiin tähän liittyviin raportteihin

Taulukko 2. Havaintoraportin malli. (Spillner ym. 2015, 195).

Jokainen raportti pitää kirjoittaa niin, että vastaava kehittäjä voi tunnistaa ongelman pienimmällä mahdollisella vaivalla ja korjata sen syyn mahdollisimman nopeasti. Ongelmien uudelleen tuottaminen, syyn paikallistaminen ja vikojen korjaaminen ovat usein suunnittelematonta lisätyötä kehittäjille. Siksi testaajan täytyy usein ”myydä” raportti kehittäjille. Raportin pitäisikin olla helposti ymmärrettävä ja selkeä. (Spillner ym. 2014, 194 – 195.)

Ongelman vakavuus on tärkeä kriteeri ongelman hoitamiselle. Taulukko 3 on mahdollinen luokittelu ongelmien vakavuudelle. Ongelman vakavuus pitäisi määrätä järjestelmän käyttäjän näkökulmasta. (Spillner ym. 2014, 195 - 196).

Luokka	Kuvaus
Kohtalokas	Järjestelmä kaatuu, mahdollinen datan tuhoutuminen. Testauksen kohdetta ei voida julkaista tässä muodossa.
Hyvin vakava	Järjestelmä toimii virheellisesti. Vaatimuksia ei ole otettu huomioon tai on toteutettu väärin. Merkittävä häiriö monille sidosryhmille. Testin kohdetta voidaan käyttää vain vakavilla rajoituksilla (vaikea tai kallis kiertäminen).
Vakava	Toiminnallinen poikkeavuus tai rajoite (”normaali” häiriö). Vaatimus väärin tai vain osittain toteutettu. Merkittävä häiriö joillekin sidosryhmille. Testin kohdetta voidaan käyttää rajoituksilla.
Kohtuullinen	Pieni poikkeama. Kohtuullista haittaa vähälle määrälle sidosryhmiä. Järjestelmää voidaan käyttää ilman rajoituksia.
Lievä	Lievää haittaa pienelle määrälle sidosryhmiä. Järjestelmää voidaan käyttää ilman rajoituksia. Esimerkiksi kirjoitusvirheet tai väärä näytön muoto.

Taulukko 3. Häiriön vakavuus. (Spillner ym. 2014, 196)

Kun määritetään vian korjaamisen kiireellisyyttä, täytyy ottaa huomioon vaatimukset tuote- tai projektijohdolta (korjaamisen monimutkaisuus) ja vaatimukset tulevilta testiajoilta (estetyt testitapaukset). Kiireellisyyden määräämiseen tarvitaan vian prioriteetti eli korjauksen prioriteetti. Taulukko 4 on esimerkki mahdollisesta luokittelusta (Spillner ym. 2014, 196.).

Prioriteetti	Kuvaus
Välitön	Käyttäjän liiketoiminta tai työprosessi on estynyt tai testejä ei voida jatkaa. Ongelma tarvitsee välittömän tai, jos tarpeen, väliaikaisen korjauksen.
Seuraava kierros	Korjataan seuraavassa normaalissa julkaisussa tai seuraavan testikohteen version toimituksen yhteydessä.
Tilaisuuden tullen	Korjataan kun järjestelmän osat käydään läpi joka tapauksessa.
Avoin	Korjaamisen suunnittelua ei ole vielä tehty.

Taulukko 4. Vian prioriteetti. (Spillner ym. 2014, 196)

Testauspäällikön pitää tukea nopeaa vikojen korjausta ja parannettujen testauksen kohteiden toimittamista. Vikojen analyysiä ja korjausta täytyy tarkkailla jatkuvasti. Tähän tarkoitukseen käytetään tapahtuman statusta havaintoraporteille. Havaintoraportti käy läpi ennalta määrätyt tilat uudesta suljettuun. Taulukko 5 sisältää listan tiloista. (Spillner ym. 2014, 197.)

Status (asettaja)	Kuvaus
Uusi (testaaja)	Uusi raportti on kirjoitettu sisältäen kuvauksen ja luokittelun.
Avoin (testauspäällikkö)	Testauspäällikkö tarkistaa uusista raporteista säännöllisesti ymmärrettävyyden ja, että niissä on kaikki tarpeelliset yksityiskohdat. Tarpeen mukaan yksityiskohtia muokataan projektinlaajuisen yhtenevän arvioitavuuden varmistamiseksi. Kaksoiskappaleet ja hyödyttömät raportit hylätään. Raportti osoitetaan vastuulliselle kehittäjälle ja statukseksi asetetaan avoin.
Hylätty (testauspäällikkö)	Kaksoiskappaleet, selvästi väärät ja perusteettomat raportit hylätään, jos testauksen kohteessa ei ole vikaa tai pyyntöä muuttaa raporttia ei ole otettu huomioon.
Analyysi (kehittäjä)	Statukseksi asetetaan analyysi, kun vastuullinen kehittäjä aloittaa raportin käsittelyn. Analyysin tulos kirjataan kommenteiksi, esim. syy, mahdolliset toimet, arvioitu korjaamisen työmäärä yms.
Havainto (kehittäjä)	Tapahtumaa ei voida toistaa eikä eliminoida. Raportti on ratkaisematon kunnes lisäinformaatiota saadaan.
Korjaus (projektipäällikkö)	Projektipäällikkö päättää analyysin perusteella korjauksesta ja asettaa statukseksi korjaus. Vastuullinen kehittäjä tekee korjaukset ja dokumentoi ne kommenteissa.
Testi (kehittäjä)	Kun vastuullinen kehittäjä on mielestään korjannut ongelman, hän asettaa statukseksi testi. Ohjelmiston uusi versio identifioidaan.
Suljettu (testaaja)	Raportit, joiden status on testi, verifioidaan seuraavalla testauskierroksella. Silloin uusitaan ainakin ne testit, jotka paljastivat ongelman. Jos havaitaan, että ongelma on poistunut, testaaja päättää raportin historian asettamalla statukseksi suljettu.
Epäonnistunut (testaaja)	Jos uusitussa testissä havaitaan, että ongelmaa ei ole onnistuttu korjaamaan, asetetaan statukseksi epäonnistunut. Uusi analyysi on tarpeen.

Taulukko 5: Havaintoraportin statustilat. (Spillner ym. 2014, 198.)

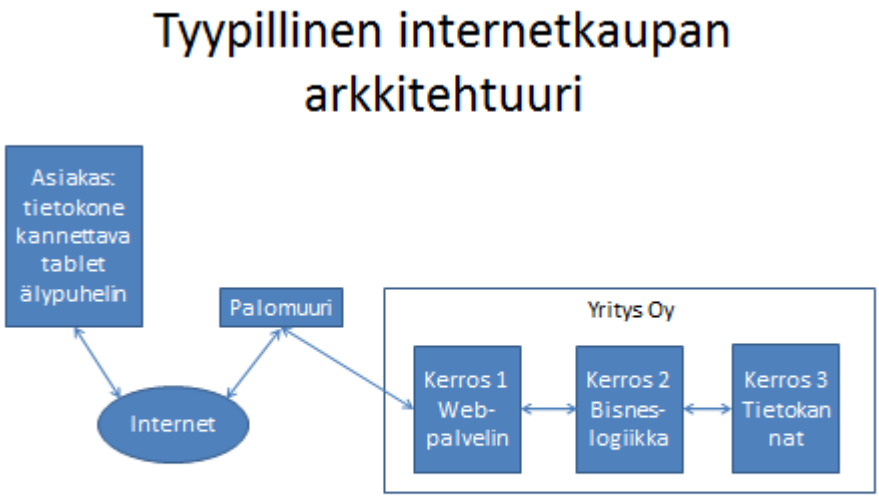
Monesti kehittäjien tekemät muutokset eivät ole korjauksia vaan toiminnallisia parannuksia. Erottelu havaintoraportin ja parannuspyynnön välillä on usein mielipidekysymys. Siksi tarvitaan elin niitä hylkäämään tai hyväksymään. Tämä elin, muutosraati (change control board), koostuu yleensä seuraavien sidosryhmien edustajista: tuotejohto, projektijohto, testausjohto ja asiakas. (Spillner ym. 2014, 199 – 200.)

3.7 Web-pohjaisten järjestelmien testaaminen

Nykyään internetkäyttäjillä on internetsivustoille korkeat vaatimukset. Jos sivusto ei lataudu nopeasti, vastaa toimiin heti ja, jos navigointi ei ole intuitiivista, käyttäjät menevät kilpailevalle sivustolle. Erityisesti tämä on tärkeää internetkaupassa. (Myers ym. 2012, 194.)

Internetpohjaisten sovellusten testaamisessa täytyy ymmärtää kolmekerroksista asiakas-palvelin–arkkitehtuuria (client-server architecture), jota käytetään tyypillisessä internetkaupan sovelluksessa. Tämä malli mahdollistaa sisällön muuttamisen yhdessä kerroksessa il-

man huolta kokonaisuuden rikkoutumisesta. Kuvio 2 on esitys asiakas-palvelin -arkkitehtuurista. (Myers ym. 2012, 195.)



Kuvio 2. Tyypillinen internetkaupan arkkitehtuuri. (Myers ym. 2012, 195)

Mallissa asiakas saattaa käyttää lähes millaista päätettä tahansa. Se voi olla esim. tietokone, älypuhelin, pelikonsoli, musiikkisoitin, jääkaappi tai auton internetselain. Internetselaimet eroavat dramaattisesti siinä, kuinka ne näyttävät sivuja. Internetsovelluksista täytyykin testata, kuinka ne toimivat eri selaimilla. Selaimet seuraavat julkaistuja standardeja, jotta ne käyttäytyisivät yhtenevästi. Kuitenkin niissä on omia parannuksia, jotka saavat ne toimimaan epäyhtenäisesti. (Myers ym. 2012, 195.)

Web-palvelin on ensimmäinen kerros kolmekerroksisessa arkkitehtuurissa. Internetsovelluksen ulkonäkö ja tuntuma tulee tästä kerroksesta. Toinen nimi sille on esityskerros (presentation layer). Siinä voidaan käyttää HTML-kieltä tai Common Gateway Interface (CGI) -skriptejä dynaamisen sivuston luomiseen. Todennäköisesti käytetään sekä staattisia että dynaamisia sivuja. (Myers ym. 2012, 195 – 196.)

Sovelluspalvelin sijaitsee kerros 2:ssa eli liiketoimintakerroksessa (business layer). Siinä ajetaan liiketoimintaprosessia mallintavaa ohjelmistoa. Toiminnallisuuksia kerroksessa ovat kaupankäynnin prosessointi, käyttäjien autentikointi ja datan validointi. (Myers ym. 2012, 196.)

Tietokannat sijaitsevat kolmannessa kerroksessa (data layer). Tyypillisesti käytetään relaatiotietokantoja. Kerros keskustelee kerroksen 2 kanssa. Joskus käytetään useampaa tietokantapalvelinta. Jotkut internetkaupat lisäävät autentikoinnin tänne. (Myers ym. 2012, 196.)

Internetsovelluksen testaukseen liittyy monia haasteita, sillä siihen liittyy elementtejä, joita ei voi kontrolloida ja useita itsenäisiä komponentteja. Testien tekemistä varten joudutaan tekemään oletuksia asiakkaista ja siitä, kuinka he käyttävät sivustoa. (Myers ym. 2012, 196.)

Mahdollinen käyttäjäkunta on suuri ja heterogeeninen. Heillä on erilaiset taidot. He käyttävät erilaisia internetselaimia, käyttöjärjestelmiä ja laitteita. Internetliittymillä on erilaisia nopeuksia. Liittymän kaistanleveydellä on yhä merkitystä, kun Internetin sisältö käy rikkaammaksi ja interaktiivisemmaksi. (Myers ym. 2012, 197.)

Testausympäristön pitäisi olla tuotantoympäristön kaltainen. Siinä pitäisi käyttää webpalvelimia, sovelluspalvelimia ja tietokantapalvelimia, kuten normaalikäytössä. Tarkkaan testaamiseen pitäisi käyttää aidon kaltaista verkkoympäristöä, missä olisi reitittimet, kytkimet ja palomuurit. (Myers ym. 2012, 197.)

Esimerkkejä esityskerroksen testaukseen kuuluvista seikoista on seuraavassa. Fonttien pitäisi olla samat eri selaimissa. Kaikkien linkkien pitäisi toimia. Grafiikan pitäisi olla oikeaa tarkkuutta ja kokoa. Oikoluetaan jokainen sivu. Kielioppi ja tyyli pitäisi tarkistaa. Varmistetaan, että kursori on oikeassa tekstilaatikossa. Oletusnapin pitäisi olla valittuna sivun latautuessa. Palautteen pitäisi olla käyttäjäystävällistä ja yhteneväistä interaktiivisissa toiminnoissa. Termien ja tyylin pitäisi olla yrityksen tai toimialan mukaista. (Myers ym. 2012, 198 – 199.)

Liiketoimintakerroksen testaamiseen kuuluu seuraavia asioita. Arvonlisävero ja toimituskustannukset pitää laskea oikein. Vastausajan ja suoritustehon pitää olla vaaditulla tasolla. Tapahtumien pitää mennä läpi oikein. Epäonnistuneet tapahtumat pitää käsitellä oikein. Tieto pitää kerätä oikein. (Myers ym. 2012, 198.)

Datakerroksen testaukseen kuuluu mm. seuraavia seikkoja. Tietokantaoperaatioiden pitää tapahtua tavoitteiden mukaan. Data täytyy tallentaa oikein ja tarkasti. Tietokanta täytyy pystyä palauttamaan varmistuksista. Testataan salaus ja turvallisuus varsinkin luottokorttitietojen ja asiakastietojen varalta. (Myers ym. 2012, 198.)

Testausstrategiassa kaiken perusta on määrittelydokumentti, joka kuvaa sivuston odotetun toiminnan ja suorituskyvyn. Ilman sitä ei sopivia testejä voida suunnitella. Testaukseen tulevat sekä itse kehitetyt että ostetut komponentit. Ne pitäisi integroida vasta, kun on varmis-

tettu, että ne toteuttavat määrittelyt. Ostettuihin komponentteihin käytetään mustalaatikkotestimenetelmiä. (Myers ym. 2012, 200 – 201.)

Jokaisella kerroksella on omat ominaisuutensa, jotka rohkaisevat testien erittelyyn. Erikseen kerrosten testaaminen tekee mahdolliseksi tunnistaa vikoja ja virheitä ennen kuin koko järjestelmän testaaminen alkaa. (Myers ym. 2012, 201 – 202.)

Kerroksien testaamiseen liittyvät omat asiansa. Esityskerroksen testaus koostuu vikojen löytämisestä graafisesta käyttöliittymästä. Se on tärkeää, jotta asiakas saisi mielikuvan laadukkaasta ja robustista sivustosta. Jos asiakkaat törmäävät käyttöliittymässä vikoihin, he eivät ehkä palaa. Kolme tärkeää aluetta testaukselle tässä kerroksessa on sisältötestaus, sivuston arkkitehtuurin testaus ja käyttäjäympäristön testaus. Sisältötestaukseen kuuluu estetiikka, fontit, värit, oikeinkirjoitus, sisällön tarkkuus ja oletusarvot. Sivuston arkkitehtuurin testaukseen kuuluu linkkien ja grafiikan testaus. Käyttäjäympäristön testaukseen kuuluu eri selainversioiden ja käyttöjärjestelmien testaus. Esityskerroksen testaukseen voidaan käyttää lasilaatikkotestimenetelmiä. (Myers ym. 2012, 203.)

Käyttäjäympäristön testaus tulee työläämmäksi, jos sivusto käyttää asiakaspään skriptien käsittelyä. Jokaisella internetselaimella on oma skriptikoneensa skriptien ajamiseksi. Erityistä huomiota selainyhteensopivuuteen pitäisi kiinnittää, jos käytetään jotakin seuraavista: ActiveX, JavaScrit, VBScript, javasovelmat, HTML5, Adobe Flash tai PHP. (Myers ym. 2012, 204.)

Liiketoimintakerroksen testauksessa virheitä haetaan suorituskyvystä, datahausta ja tapahtumien käsittelystä. Lasilaatikkotestausmenetelmiä käytetään itse kehitettyihin komponentteihin. Ostettuihin komponetteihin täytyy käyttää mustalaatikkomenetelmiä. Kolme aluetta täytyy testata: suorituskyky, datan validiteetti ja tapahtumat. (Myers ym. 2012, 205 – 206.)

Suorituskyvyn testaaminen on tärkeää, koska heikosti toimiva sivusto herättää käyttäjän mielessä kysymyksiä sen robustisuudesta ja usein kääntää hänet pois sivustolta. Määrittelyssä pitäisikin olla kohtansa vastausajoille ja suoritusteholle. Usein käytetään rasitustestausta, koska suuri määrä yhtäaikaisia kyselyjä saa sivuston usein hidastumaan mahdottomaksi käyttää. (Myers ym. 2012, 206 – 207.)

Tapahtumien testaus on tärkeää, koska verkkokaupan täytyy käsitellä tapahtumat oikein aina poikkeuksetta. Poikkeuksia ei sallita. Asiakkaat eivät siedä epäonnistuneita kauppoja. (Myers ym. 2012, 207 – 208.)

Tieto, jota sivustolla kerätään, on hyvin tärkeää. Kerättäviä tietoja voivat olla luottokorttinumerot, maksutiedot ja käyttäjäprofiilit. Tämän tiedon menettäminen voi olla katastrofaa-

lista yritykselle. Siksi sen suojelemisesta pitää huolehtia. Datakerroksen testaamisessa on kyse tietokantajärjestelmän testauksesta. Datakerroksesta pitäisi testata vastausnopeus, datan oikeellisuus sekä vikasietoisuus ja palautuvuus. (Myers ym. 2012, 208 – 209.)

Vastausnopeuden testaaminen on tärkeää, koska hidas verkkokauppa aiheuttaa tyytymättömiä ja luottamattomia asiakkaita. Testaamisessa pyritään löytämään tietokantaoperaatioita, jotka eivät vastaa suorituskykyvaatimuksia. Useimmat vastausaikatestit tehdään mustalaatikkomenetelmillä. Tähän työhön on saatavilla monia työkaluja. (Myers ym. 2012, 209.)

Datan oikeellisuuden testaamisessa yritetään löytää virheellistä dataa tietokannoista. Tämä eroaa datan validoinnista, joka testataan liiketoimintakerroksen testauksessa. Tässä yritetään löytää virheitä siitä, kuinka dataa tallennetaan. Datan tyyppi ja pituus voi aiheuttaa datan lyhentymistä tai tarkkuuden häviämistä. Päivämäärä- ja aikakentissä aikavyöhykeasiat ovat mukana. Tallennetaanko aika asiakkaan sijainnin mukaan vai web-palvelimen, sovelluspalvelimen vai tietokannan sijainnin mukaan? Merkistö voi vaikuttaa datan oikeellisuuteen. Pitäisi myös varmistaa viitetaulukoiden tarkkuus, joita käytetään tallentamaan arvonlisävero, postinumeroita ja aikavyöhykkeitä. Niistä pitää varmistaa paitsi oikeellisuus, myös ajantasaisuus. (Myers ym. 2012, 210.)

Vikasietoisuuden ja palautuvuuden testaus pitää tehdä, sillä relaatiotietokantaa käyttävän järjestelmän täytyy toimia lähes koko ajan. Yksi tavoite on maksimoida keskimääräinen aika häiriöiden välillä (MTBF) ja minimoida keskimääräinen palautuvuusaika (MTTR). Testauksessa yritetään alittaa MTBF ja ylittää MTTR. (Myers ym. 2012, 210 – 211.)

Edellä kerroin internetsovellusten testauksen näkemyksestä, joka löytyi kirjasta The Art of Software Testing, jonka ovat kirjoittaneet Myers ym. (2012). Tästä alkaa Perryn (2006) näkemys web-pohjaisten järjestelmien testauksesta.

Web-pohjaisiin järjestelmiin liittyy omat riskinsä. Testaus pitäisi suunnata niiden mukaan. Riskejä ovat turvallisuus, suorituskyky, oikeellisuus, yhteensopivuus, luotettavuus, datan muuttumattomuus, käytettävyys ja palautuvuus. (Perry 2006, 802-803.)

Turvallisuuteen liittyy seuraavia riskejä: ulkopuolinen tunkeutuminen järjestelmään, transaktioiden suojaus, virukset, pääsyn valvonta ja valtuutustasot. Internetkaupassa on tärkeää suojata transaktiot, jotta ulkopuoliset eivät pääse käyttämään siirrettäviä tietoja vääriin tarkoituksiin. Pääsyn valvonnalla pidetään huoli siitä, että vain valtuutetuilla käyttäjillä on pääsy sovelluksen tiettyihin osiin. Valtuutustasoja tarvitaan siihen, että vain tietyn valtuutustason omaavat henkilöt voivat tehdä tiettyjä transaktioita. (Perry 2006, 803.)

Järjestelmän suorituskyvyllä on ratkaiseva merkitys käyttäjien hyväksynnän saamisessa. Yleisin internetsovellusten suorituskykytesti on rasitustestaus. Sillä voidaan testata, toimiiko sovellus odotetuilla ja odotettua suuremmilla aktiviteettitasoilla. (Perry 2006, 803-804.)

Oikeellisuus sovelluksen toiminnassa on yksi tärkeimpiä huolen aiheita testauksessa. Nappien täytyy toimia oikein, kuten myös laskutoimitusten ja navigoinnin. Toiminnallinen oikeellisuus tarkoittaa, että sovellus toimii, kuten vaatimuksissa määrätään. (Perry 2006, 804.)

Sovelluksen täytyy olla yhteensopiva monenlaisten ympäristöjen kanssa. Tärkeimmät ympäristötekijät ovat käyttöjärjestelmät ja internetselaimet. Käyttöjärjestelmien tuki eri selaimille vaihtelee ja se vaikuttaa websovelluksen toimintaan käytetyssä selaimessa. (Perry 2006, 804.)

Luotettavuus on oma huolenaiheensa. Sovelluksen täytyy aina antaa oikeita tuloksia. Palvelimen ja järjestelmän saatavuus on yksi huolehdittava asia. (Perry 2006, 806.)

Websovelluksen täytyy pitää huoli siitä, että vain oikeaa dataa hyväksytään sisään validoinnin avulla. Datan täytyy myös pysyä oikeana, mihin tarvitaan varmistuskopiointia ja datan päivitystä vain kontrolloiduilla menetelmillä. (Perry 2006, 806.)

Käytettävyydestä täytyy pitää huoli, koska vaikeakäyttöisestä websovelluksesta siirrytään helposti kilpailevaan vaihtoehtoon. Siksi sovelluksen on oltava helppo käyttää ja ymmärtää. Sovelluksen tietojen täytyy olla helppoja tulkita ja käyttää. Navigoinnin täytyy olla selkeä ja toimia oikein. (Perry 2006, 806.)

Palautuvuus on erityisen tärkeää internetsovelluksissa, sillä ne ovat haavoittuvia katkoksille. Internetsovelluksissa ovat tärkeitä seuraavat asiat: kadonneet yhteydet aikakatkaisujen tai pudonneiden linjojen vuoksi, asiakasjärjestelmän kaatuminen ja palvelinjärjestelmän kaatuminen tai muu sovellusongelma. (Perry 2006, 807.)

Kun tarkistettavat riskit on valittu, täytyy valita testimenetelmät. Web-pohjaiset testityökalut on valittava niiden jälkeen. Testityökalujen valinnan jälkeen voidaan aloittaa testaaminen. Testaamisen tasoista ja menetelmistä on kerrottu aikaisemmin. Testauksesta pitäisi saada testaajilta raportti, jonka pitäisi sisältää vähintään seuraavat tiedot: lyhyt kuvaus testatusta järjestelmästä, testatut riskit, käytetyt ja käyttämättömät testityypit, käytetyt työkalut, toiminnallisuus ja rakenne, joka toimi testeissä oikein ja, joka ei toiminut oikein sekä testaajien mielipide järjestelmän riittävyydestä tuotantokäyttöön. (Perry 2006, 807 - 810.)

4 Tutkimusmenetelmät

Laadullista tutkimusta on kaikki, mikä ei käsittele numeroaineistoja ja, missä ei käytetä tilastollisia menetelmiä (Töttö 2000, 116). Sen perusteella tämä tutkimus on laadullinen. Tuomen ja Sarajärven (2009, 22) mukaan ”laadullinen tutkimus on empiiristä ja laadullisessa tutkimuksessa on kyse empiirisen analyysin tavasta tarkastella havaintoaineistoa ja argumentoida”. Tutkimusote pitäisi valita tutkimusongelma lähtökohtana. Tässä tutkimuksessa tutkimusongelma on seuraava. Minkälaisia havaintoja löydetään tuotannossa olevasta web-sovelluksesta toiminnallisella testauksella? Tämä tukee laadullista tutkimusotetta.

Uusitalon (1991, 50) mukaan aineiston ja tutkimusmenetelmän välillä on vuorovaikutussuhde. Tähän tutkimukseen on hankittu yhden intranetin testauksessa testaajien tekemiä dokumentteja. Siksi tutkimus tehdään kvalitatiivisena tapaustutkimuksena.

Mahdollisia tutkimusaineistoja kvalitatiivisessa tutkimuksessa on lähes rajattomasti. Yksi mahdollisuus on valmis aineisto. Erilaisia mahdollisia valmiita aineistoja on paljon erilaisia. (Eskola ja Suoranta 2000, 117 – 119.) Organisaatioiden dokumentit ovat yksi mahdollisuus. Niihin kuuluvat pöytäkirjat, joiden kaltaisia opiskelijoiden opintojaksolla E0023 Verkkopalvelun testaus kirjoittamat raportit ovat. (Uusitalo 1991, 94.) Raportteja oli rajallisesti, mikä helpotti opinnäytteen käytännön osuuden kirjoittamista. Tuomi ja Sarajärvi (2009, 84) sanovat, että yksityiset dokumentit ovat mahdollinen lähdeaineisto. Näihin kuuluvat myös opiskelijoiden raportit kurssilta. Kanasenkin (2008, 81) mukaan dokumentit, kirjallinen materiaali, sopii kvalitatiivisen tutkimuksen aineistoksi.

Eskola ja Suoranta (1998, 14) kertovat, että kvalitatiivisessa tutkimuksessa pyritään ymmärtämään jotakin toimintaa, kuten tässä tutkimuksessa web-pohjaisen sovelluksen toiminnallista testausta. Tutkimusaineiston analyysiin käytetään vähemmän strukturoituja menetelmiä, jotka kehittyvät analyysin edetessä (Uusitalo 1991, 53). Minun tutkimuksessani jouduin käyttämään menetelmiä, jotka kehittyivät työn edetessä.

Kvalitatiiviseen tutkimukseen liittyy anekdotalismin ongelma (problem of anecdotalism). Se tarkoittaa sitä, kuinka tukija vakuuttaa itsensä ja lukijansa siitä, että hänen löydöksensä perustuvat koko datan kriittiseen tarkasteluun eikä vain muutamaan valittuun esimerkkiin. (Silverman 2005, 211.)

Dokumentteja voidaan analysoida systemaattisesti ja objektiivisesti käyttämällä menettelytapana sisällönanalyysia. Siinä pyritään saamaan kuvaus tutkittavasta ilmiöstä tiivistetyssä ja

yleistetyssä muodossa. Se on tekstianalyysia. Siinä etsitään inhimillisiä merkityksiä. Sisällönanalyysissa dokumenttien sisältöä kuvataan sanallisesti. (Tuomi & Sarajärvi 2009, 103 – 106.)

Aineistolähtöinen analyysi lähtee yksittäisestä jatkuen yleiseen eli siinä luodaan teoreettisia käsitteitä aineistosta (Kananen 2008, 90). Aineistolähtöisessä aineiston analyysissa on kolme vaihetta: aineiston redusointi eli pelkistäminen, aineiston klusterointi eli ryhmittely ja abstrahointi eli teoreettisten käsitteiden luominen. Pelkistämisessä datasta karsitaan tutkimukselle epäolennainen pois (Tuomi & Sarajärvi 2009, 108 - 109).

Minä tein tämän opinnäytetyöni enemmän tai vähemmän tässä järjestyksessä. Redusoinnissa otin analyysitaulukkoon mukaan vain kaikkein tärkeimmät tiedot opiskelijoiden raporteista. Klusteroinnin jätin pois omasta työstäni ellei testien jakaminen testaajaryhmiin käy tähän. Abstrahoinnissa muodostin uudet teoreettiset käsitteet eli havaintojen tyypit ja tavat havaintojen löytämiseksi.

Reliabiliteetti ja validiteetti ovat käsitteitä, joita käytetään kvantitatiivisessa tutkimuksessa, jollainen tämä tutkimus ei siis ole. Kvalitatiivisessa eli laadullisessa tutkimuksessa puhutaan enemmänkin analyysin arvioitavuudesta. Arvioitavuus tarkoittaa sitä, että lukija voi seurata tutkijan päättelyä. Kvalitatiivisessa tutkimuksessa käytetään myös käsitettä toistettavuus, joka tarkoittaa sitä, että ”tutkijan käyttämät luokittelu- ja tulkintasäännöt ovat yksiselitteiset ja että niitä noudatetaan johdonmukaisesti”. Tieteen tekemisessä kvalitatiivista tutkimusta käytetään teorian kehittelyyn. (Uusitalo 1991, 82.)

Reliabiliteetti viittaa yhdenmukaisuuteen, jolla eri tarkkailijat tai sama tarkkailija eri kerroilla liittävät asioita samaan kategoriaan. Tässä tulee kyseeseen se, miten eri tutkijat tarkastelisivat tutkimuksen tärkeimpiä tuloksia, eli luetteloja löydetyistä havaintojen tyypeistä ja tavoista niiden löytämiseksi. Joku toinen tutkija saattaisi nähdä asiat eri tavalla ja päätyä eri tuloksiin. Tämä on luonnollista laadullisissa tutkimuksissa yleensäkin. (Silverman 2005, 224.)

Validiteetti on toinen nimi totuudelle. Tutkimus ei ole validi, jos vain muutamia esimerkkejä raportoidaan, kriteerejä tai perusteita ei ole annettu joidenkin tapausten sisällyttämiseen ja toisten ei, tai materiaalin alkuperäistä muotoa ei ole saatavilla. (Silverman 2005, 224.)

Mäkelän mukaan laadullisen tutkimuksen luotettavuutta voidaan arvioida kolmella kriteereillä. Ensimmäinen on aineiston riittävyys. Toinen on analyysin kattavuus. Kolmas on analyysin arvioitavuus ja toistettavuus. (Mäkelä 1990, 47 – 48.)

Aineiston riittävyydellä tarkoitetaan kyllääntymistä eli saturaatiota. Kattavuudella tarkoitetaan sitä, että tulkintoja ei tehdä pelkästään aineiston satunnaisista osista. Arvioitavuus liit-

tyy tutkimusmateriaalin, eri vaiheiden ja tulkintojen todentaminen eli dokumentointi. Tarkkuus dokumentoinnissa tekee mahdolliseksi päätelmien ja ratkaisujen ulkopuolisen tarkastelun. Toistettavuus liittyy arvioitavuuteen siten, että riittävän tarkka tutkimusasetelman ja prosessien dokumentointi mahdollistaa toistettavuuden. (Kananen 2008, 125.)

5 Laurea Liven toiminnallisen testauksen toteutus

Laurea Live on järjestelmä, jonka kautta Laurea-ammattikorkeakoulun opiskelijat ja henkilökunta saavat tärkeää tietoa toiminnasta koulussa ja voivat esimerkiksi ilmoittautua kursseille ja tentteihin. Laurea Liven testaus järjestettiin Keravan toimipisteessä kurssin E0023 Verkkopalvelun testaus ja käyttöönotto osana kahtena peräkkäisenä vuonna järjestetyissä toteutuksissa. Tämä tapahtui vuosina 2013 ja 2014. Testaus oli opintojaksolla tehtävä, jonka perusteella opiskelijat saivat kurssin arvosanan. Testaus tehtiin ryhmätyönä.

Koululla järjestetyssä testissä Laurea Liveä testattiin toiminnallisuuden kannalta. Siis testattiin, että se toimii vaatimusten mukaan. Vaatimukset olisivat olleet käytössä, mutta ne luotiin testien yhteydessä ensimmäistä kertaa toimintoihin tutustuttaessa. Silloin katsottiin, kuinka toiminto toimii koulun tietokoneessa ja sen perusteella kirjoitettiin muistiin, kuinka se toimi siinä. Tähän liittyy virhemahdollisuus, koska opiskelijoilla oli valittavanaan koulun koneilla ainakin vuonna 2014 kaksi internetselainta, jotka olivat Internet Explorer ja Mozilla Firefox. Koululla opiskelijoiden käytössä oli HP-merkkiset 64-bittiset pöytäkoneet Windows 7 –käyttöjärjestelmällä.

Minulla oli käytettävissäni tutkimusaineisto, joka koostui vuosina 2013 ja 2014Laurea Livelle Laurean Keravan yksikössä tehdystä toiminnallisesta testauksesta kirjoitetuista raporteista. Ensimmäisessä iteraatiossa tutkin aineistoa saadakseni selville havainnot, joita opiskelijat järjestelmästä löysivät. Tämän lisäksi hain tapoja, joita opiskelijat työssään käyttivät. Kolmanneksi tukeuduin myös omaan muistiini kokemuksestani kurssin opiskelijana kevätlukukaudella 2014. Toisessa iteraatiossa tutkin, löysivätkö opiskelijat kahtena vuonna samoja havaintoja.

Tehtävässä opiskelijat tekivät lyhyet raportit kaikista tekemistään testitapauksista. Testin tulokseksi he merkitsivät joko OK, virheellinen tai ei voitu testata. OK tarkoittaa, että Laurea Live käyttäytyi testattaessa odotetusti. Virheellinen tarkoittaa odotetusta poikkeavaa käyttäytymistä. Ei voitu testata tarkoittaa sitä, että testiä ei jostakin syystä voitu suorittaa. Keräsin tiedot kaikista onnistuneista testeistä (testin tulos virheellinen) ja testeistä, joita ei voitu tehdä, yhteen Excel-taulukkoon, jonka nimesin analyysitaulukoksi.

Täytin tiedoston analyysitaulukko.xls testitapauksilla, joissa löytyi poikkeama Laurea Liven toiminnassa. Annoin testitapauksille yksilöivän tunnisteen. Lisäksi taulukossa on sarakkeet Toimenpiteet, Alkuehto, Järjestelmä, Internetselain, Odotettu lopputulos, Lopputulos, Havaitut poikkeamat, Havainnot, Vakavuusluokka, Havainnon tyyppi ja Menetelmä. Analyysitaulukossa on ryhmän tunnisteena numero ja vuosi (2013 tai 2014). Merkitsin taulukkoon muistiin Laurea Liven osan, johon ryhmä keskitti testinsä, jos sellaista oli. Havainnon tyypillä tarkoitan sitä, minkä tyyppinen löytynyt poikkeus on. Menetelmällä tarkoitan sitä, kuinka poikkeus on löytynyt.

6 Tutkimustulokset

Tehty testaus on hyväksymistestauksen kaltaista. Hyväksymistestauksessa käytetään vaatimuksia pohjana, jotta tiedettäisiin, kuinka järjestelmän pitäisi toimia. Ainakaan vuonna 2014 ei määrittelyjä opintojaksolla E0023 Verkkopalvelun testaus ja käyttöönotto välttämättä käytetty, vaan ohjelman oikea toiminta selvitettiin käyttämällä sitä koulun tietokoneilla ja merkitsemällä ohjelman käyttäytyminen muistiin. Tässä on virhemahdollisuus, koska koulun tietokoneilla oli kaksi internetselainta: Internet Explorer ja Mozilla Firefox. Laurea Liven toiminta tehtyjen testien mukaan riippuu käytetystä päätelaitteesta ja suuressa määrin käytetystä selaimesta. Internet Explorerilla testitulos saattoi olla erilainen, kuin Firefoxilla. Näin saatu malli Liven toiminnasta saattoi riippua selaimesta. Lisäksi ohjelma saattoi toimia virheellisesti myös koulun laitteilla selaimesta riippumatta.

Ensimmäisellä iteraatiolla tutkin, minkä tyyppisiä havaintoja opiskelijat löysivät Laurea Livestä vuosina 2013 ja 2014 sekä millä tavoilla ne löytyivät. Löytyneet havainnon tyypit ovat seuraavat. Havainnon tyyppi ”Linkki ei valikossa” selittää itsensä. ”Toiminto virheellinen” tarkoittaa sovelluksen käyttäytymistä virheellisesti. ”Kertakirjautuminen puuttuu” tarkoittaa sitä, että testaaja odotti siirtymiseen Laurea Livestä johonkin alijärjestelmään tapahtuvan niin, että siinä ei enää tarvitse kirjautua alijärjestelmään uudelleen, mutta uudelleen kirjautumista vaadittiin. ”Ominaisuus” tarkoittaa, että havainto on mielestäni sovelluksen ominaisuus. ”Sivuston ulkonäkö” tarkoittaa sivuston ulkonäköön vaikuttavaa häiriötä. ”Toiminto puuttuu” tarkoittaa, että testaajan hakemaa toimintoa ei ole olemassa. ”Kieliversio puutteellinen” tarkoittaa, että vieraalla kielellä tehty sivusto ei ole täydellinen eli siinä eivät kaikki sivut ole halutulla kielellä. ”Palaute puuttuu” tarkoittaa, että toiminto ehkä toimii, mutta se ei anna käyttäjälle palautetta onnistumisesta. ”Sovellus keskeneräinen” tarkoittaa, että toiminto löytyy sivulta, mutta se ei toimi. ”Käytettävyysongelma” tarkoittaa, että toimintoa on turhan vaikeaa käyttää tai vaikeaa löytää. ”Liveen kirjautuminen mahdotonta” tarkoittaa, että toimintoa ei päästy testaamaan, koska Liveen ei jostakin syystä voinut kirjautua. ”Laiteongelma” tarkoittaa käytettyyn laitteeseen liittyvää ongelmaa. ”Eteneminen pysähtyy” tarkoittaa, että Liveen on kirjauduttu, mutta toimintoon ei pääse. ”Valikko ei toimi näkymässä”

tarkoittaa mitä sanookin. ”Linkki ei toimi” tarkoittaa, että linkki on olemassa, mutta siitä ei pääse minnekään.

Seuraavaksi annan luokittelun sille, kuinka havainto on löydetty Laurea Livestä. ”Silmämääräinen tarkastus” tarkoittaa sitä, että havainto on paljastunut sivustoa katsomalla silmämääräisesti, eikä esimerkiksi jotakin linkkiä painamalla. ”Sovelluksen käyttö” tarkoittaa sitä, että havainto on löytynyt sovellusta käyttämällä, esimerkiksi toiminnon linkkiä painamalla. ”Selaimen vaihto” tarkoittaa, että havainto on löytynyt käyttämällä jotakin tiettyä internetselainta käyttämällä tietokonetta. ”Järjestelmän vaihto” tarkoittaa, että selainta ja laitteistoa on vaihdettu havainnon löytämiseksi. ”Tunnusten vaihto” tarkoittaa, että on testattu jotakin kohtaa kokeilemalla eri käyttäjätunnusta.

Toisella iteraatiolla tutkin löysivätkö opiskelijat testeissään kahtena vuonna samoja havaintoja. Niitä löytyi kaksi kappaletta. Oikopolku-linkki tentti-ilmoon toi onnistuneita testejä kahtena vuonna. Sekä vuoden 2013 ryhmä 1 että vuoden 2014 ryhmä 1 huomasivat tästä, että kertakirjautuminen ei toimi sen yhteydessä. Tentti-ilmoon ei ole voinut yhdistää kertakirjautumista, vaikka se on ohjelmoitu Laureassa (Salo 2015).

Sekä vuoden 2013 ryhmä 1 että vuoden 2014 ryhmä 1 huomasivat oikopolkuvalikosta Optima-linkin vievän uuteen kirjautumiseen. Kumpikin ryhmä havaitsi siis kertakirjautumisen puuttuvan tästäkin. Kertakirjautumisen tekeminen siihen on jäänyt muiden kehittämishankkeiden alle (Salo 2015).

7 Yhteenveto ja johtopäätökset

Tässä opinnäytetyössä on käsitelty kokonaista aineistoa koskien Laurea Liven testausta niinä kahtena vuonna, joina se järjestettiin. Tästä syystä kaksi ensimmäistä Silvermanin esittämää kriteeriä on täytetty eli ne eivät laske sen validiteettia.

Tässä tutkimuksessa ei varmaankaan ole koottu aineistoa niin paljon, että kyllääntyminen olisi tapahtunut. Aineistoa on kuitenkin koottu Laurea Liven testauksesta niin paljon kuin mahdollista eli sen määrää ei enää olisi voinut lisätä.

Aineiston kattavuus ei välttämättä ole paras mahdollinen, koska testausta Laurea Livestä ei tehty enempää. Suurempi määrä testejä suuremmasta osasta järjestelmää varmasti parantaisi kattavuutta. Toisaalta analyysi tässä tutkimuksessa on tehty koko aineistoa käsitellen, mikä parantaa kattavuutta.

Arvioitavuuden puolesta tässä tutkimuksessa on tehty dokumentointi tutkimuksen kulusta ja menetelmistä. Tutkimusmenetelmät on dokumentoitu osassa tutkimusmenetelmät.

Laurea Liven käyttö ja mahdollisuus käyttää sitä on hyvin riippuvaista käytetystä tietokonejärjestelmästä ja internetselaimesta. Windowsissakaan ei kaikilla selaimilla käyttö onnistu. Lisäksi erilaisissa päätelaitteissa Laurea Live ei välttämättä toimi.

Laurea Livessä on linkkejä Laurean ulkopuolisiin järjestelmiin. Niihin ei usein päästy sisään ilman erillistä kirjautumista, minkä testaajaryhmät katsoivat häiriöksi, mikä vaikuttaa aika rohkealta päättelyltä. Laurean tietokonejärjestelmien ylläpidolla ei välttämättä ole mahdollisuutta vaikuttaa näihin, ja siksi tarvitaan uusi kirjautuminen. Kertakirjautumisen puuttuminen kannattaisi raportoida häiriöksi vain, jos vaatimus sille on järjestelmän vaatimuksissa. Opiskelijat eivät kaikissa ryhmissä ainakaan vuonna 2014 käyttäneet niitä. Tämä onkin esimerkki siitä, kuinka testausta ei pitäisi tehdä. Testauksen pitäisi aina pohjautua vaatimuksiin.

Näinä kahtena vuonna, jona Laurea Liven testaus opintojaksolla E0023 Verkkopalvelun testaus ja käyttöönotto järjestettiin, löydettiin suuri määrä havaintoja. Testaus on kuitenkin ollut rajallista. Jos olisi testattu järjestelmä kokonaisuudessaan, olisi havaintoja varmasti löytynyt paljon suurempi määrä, koska järjestelmä on niin laaja ja tässä rajatussa testauksessa havaintoja löytyi niin paljon.

Samoja havaintoja ei tullut montaa eri vuosien kurssitoteutuksissa, koska opiskelijat pääasiassa keskittyivät testeissään eri alueisiin ohjelmasta. Yksi näistä havainnoista oli oikopolkulinkki tentti-ilmoon ja toinen oli oikopokujen Optima-linkki. Kummassakaan ei kertakirjautuminen toiminut kumpanakaan vuonna. Sen kehittäminen tentti-ilmoon olisi ollut hyvin työlästä ja sen kehittäminen Optimaan on jäänyt muiden kehittämishankkeiden vuoksi tekemättä.

Varsinaisena analyysipäätelmänä tuon esille kaksi kohtaa. Ensimmäinen on luettelo web-sovelluksen hyväksymistestauksessa löytyvistä havainnon tyypeistä. Toinen on samassa yhteydessä käytetyt tavat havaintojen paljastamiseksi.

Tämän opinnäytetyön analyysipäätelmän ensimmäisenä osana voin antaa listan löytämistäni havaintojen tyypeistä. Ne ovat samat, kuin taulukossa 6. Havaintojen tyypit ovat löytämieni havaintojen tyypittelyn tulos.

Havainnon tyyppi	Merkitys
Linkki ei valikossa	Haettua linkkiä ei löydy valikosta
Toiminto virheellinen	Sovellus käyttäytyy virheellisesti
Kertakirjautuminen puuttuu	Alijärjestelmään täytyy kirjautua erikseen
Sivuston ulkonäkö	Sivuston ulkonäköön vaikuttava häiriö
Toiminto puuttuu	Haettua toimintoa ei ole olemassa
Kieliversio puutteellinen	Vieraalla kielellä tehdyssä sivustossa eivät kaikki sivut ole halutulla kielellä
Palaute puuttuu	Toiminto ehkä toimii, mutta ei anna palautetta onnistumisesta
Sovellus keskeneräinen	Toiminto löytyy sivulta, mutta se ei toimi
Käytettävyysongelma	Toimintoa on turhan vaikeaa käyttää tai löytää
Järjestelmään kirjautuminen mahdotonta	Toimintoa ei päästy testaamaan, koska järjestelmään ei jostakin syystä voi kirjautua
Laiteongelma	Käytettyyn laitteeseen liittyvä ongelma
Eteneminen pysähtyy	Järjestelmään on kirjauduttu, mutta toimintoon ei pääse
Valikko ei toimi näkymässä	Valikon käyttäminen ei onnistu näkymässä
Linkki ei toimi	Linkki on olemassa, mutta siitä ei pääse minnekään

Taulukko 6. Havaintojen tyypit merkityksineen.

Kun käytetään testauksessa pohjana järjestelmän vaatimuksia, jäävät pois testitapaukset, joissa testataan järjestelmän ominaisuutta, joka arvostellaan häiriöksi. Siis havainnon tyyppi ”ominaisuus” jää pois. Kun tarkastellaan järjestelmien testaamista yleensä, täytyy havainnon tyypin nimi ”Liveen kirjautuminen mahdotonta” korvata uudella termillä ”järjestelmään kirjautuminen mahdotonta”. Havainnon tyyppi ”järjestelmään kirjautuminen mahdotonta” on aina yhteydessä siihen, että testiä ei voitu ajaa.

Löytämäni tyypit testaajien käyttämille tavoille havaintojen löytämiseksi ovat myös osa analyysipäätelmää tässä opinnäytetyössä. Taulukko 7 sisältää tavat merkityksineen.

Tapa	Merkitys
Silmämääräinen tarkastus	Havainto on paljastunut katsomalla sivustoa silmämääräisesti, eikä esimerkiksi jotakin linkkiä painamalla
Sovelluksen käyttö	Havainto on löytynyt sovellusta käyttämällä, esimerkiksi toiminnon linkkiä painamalla
Selaimen vaihto	Havainto on löytynyt käyttämällä jotakin tiettyä internetselainta
Järjestelmän vaihto	Selainta ja laitteistoa on vaihdettu havainnon löytämiseksi
Tunnusten vaihto	On testattu jotakin kohtaa kokeilemalla eri käyttäjätunnusta

Taulukko 7. Tavat havaintojen löytämiseksi merkityksineen.

Jos vaatimuksissa on määritelty sovellukseen jokin asia näkyville, niin sen luulisin aivan silmämääräisesti katsomalla löytyvän. Siksi silmämääräinen tarkastus paljastaa täysin puuttuvia toimintoja. Sovellusta käyttämällä voidaan tietenkin paljastaa sovelluksen virheellistä toimin-

taa. Selaimen vaihto toimii menetelmänä havaintojen tekemiseksi, koska internetsovellukset toimivat eri selaimissa niin eri tavoin tai eivät joissakin ollenkaan. Järjestelmän vaihto paljastaa eri laitteissa toimimattomia sovelluksen ominaisuuksia tai kokonaan toimimattomia internetsovelluksia. Tunnusten vaihto paljastaa eri käyttäjätunnuksilla toimimattomia osia internetsovelluksesta.

Kolmannessa tutkimuskysymyksessä kysyttiin, mitä seikkoja pitää ottaa huomioon websovelluksen toiminnallisen testauksen suunnittelussa. Kokoan tähän vastauksen teoriaosuudesta. Ensin täytyy tehdä testaussuunnitelma. Siihen kuuluvat testattavat ominaisuudet, ominaisuudet, joita ei testata, lähestymistapa, testin lopetusehdot, keskeytyskriteerit, uudelleenaloitusehdot, testauksen tuotokset, testaustehtävät, ympäristön tarpeet, vastuut, henkilöstönhallinta ja valmennustarpeet, aikataulu, riski ja epävarmuudet. Havaintoraportin malli täytyy suunnitella.

Testattavia seikkoja ovat eri käyttöjärjestelmät, internetselaimet ja päätelaitteet. Testausympäristö täytyy rakentaa aidon kaltaiseksi. Esityskerros, liiketoimintakerros ja datakerros täytyy testata kukin erikseen. Esityskerroksesta testataan fontit, linkit ja grafiikka sekä navigointi. Oikoluetaan sivusto. Varmistetaan kursorin sijainti ja oletusnapin valittuna olo. Liiketoimintakerroksesta tarkistetaan arvonlisäveron ja toimituskustannusten laskenta, tapahtumien läpimeno ja tiedon keräys. Datakerroksesta tarkistetaan tietokantaoperaatiot, datan tallennus, validointi, salaus ja turvallisuus.

8 Jatkokehitysehdotukset

Tämä tutkimus on tehty kouluympäristössä opiskelijoiden tekemästä web-sovelluksen hyväksymistestauksesta. Mielenkiintoista myöhemmin olisi tehdä samanlainen tutkimus todellisessa työelämässä web-sovelluksen toiminnallisessa testaamisessa. Siinä voisi olla eroja tämän tutkimuksen tuloksiin ja tulokset olisivat paremmin yleistettävissä.

Voitaisiin myös tutkia sovellustestausta testauksen eri vaiheissa. Tutkimus voisi keskittyä siis johonkin V-mallin muuhun vaiheeseen, eli yksikkötestaukseen, integrointitestaukseen tai järjestelmätestaukseen. Sieltä voisi tulla esille erilaisia havaintojen tyyppejä ja tapoja niiden paljastamiseksi.

Lähteet

Kirjat

Amman, P. & Offutt, J. 2008. Introduction to software testing. New York: Cambridge University Press.

Eskola, J. & Suoranta, J. 2001. Johdatus laadulliseen tutkimukseen. 5. painos. Tampere: Vastapaino.

Farrell-Vinay, P. 2008. Manage software testing. Boca Raton: Auerbach Publications.

Fournier, G. 2009. Essential software testing : a use-case approach. Boca Raton, Fla: CRC.

Kananen, J. 2008. Kvali: Kvalitatiivisen tutkimuksen teoria ja käytänteet. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Kasurinen, J. P. 2013. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo.

Mette Jonassen Hass, A. 2008. Guide to Advanced Software Testing. Norwood: Artech House.

Myers, G.J. & Badgett, T. & Sandler, C. 2012. The art of software testing. 3rd ed. Hoboken, N. J.: John Wiley & Sons.

Mäkelä, K. 1990. Kvalitatiivisen aineiston analyysi ja tulkinta. Helsinki: Gaudeamus.

Perry, W. E. 2006. Effective methods for software testing. 3rd ed. Indianapolis: Wiley.

Silverman, D. 2005. Doing qualitative research : a practical handbook. 2. ed. London: Sage Publications.

Spillner, A. & Linz, T. & Schaefer, H. 2014. Software testing foundations : a study guide for the certified tester exam : foundation level, ISTQB compliant. Santa Barbara, CA: Rocky Nook, Inc.

Tuomi, J. & Sarajärvi, A. 2009. Laadullinen tutkimus ja sisällönanalyysi. 9., uudistettu laitos. Helsinki: Tammi.

Töttö, P. 2000. Pirullisen positivismin paluu. Laadullisen ja määrällisen tarkastelua. Tampere: Vastapaino.

Uusitalo, H. 1991. Tiede, tutkimus ja tutkielma : johdatus tutkielman maailmaan. Porvoo: WSOY.

Yin, Robert K. 1989. Case Study Research. Design and Methods. Newbury Park, London, New Delhi: Sage Publications.

Artikkelit

Choudhary, Dinesh & Kumar, Vijay. 2011. Software testing. Journal of Computational Simulation and Modeling 1.1, 1 – 9.

Sähköiset lähteet

Bourque, P. & Fairley, R. E. 2014. SWEBOK V3.0, Guide to the Software Engineering Body of Knowledge. IEEE. Luettu 4.9.2014.
http://www.computer.org/portal/web/swebok/swebokv3

ISTQB. 2007. ISTQB:n testaussanasto. Luettu 14.4.2015.
http://www.fistb.fi/sites/fistb.ttlry.mearra.com/files/istqb_sanasto.pdf

Julkaisemattomat lähteet

Salo, M. 2015. Kysymys Laurea Liven kehittämisestä opinnäytetyötä varten. Sähköposti Jarmo Keinäselle 25.9.2015.

Kuviot

Kuvio 1. Yleinen V-malli testaukselle. (Spillner ym. 2014, 40) 19
Kuvio 2. Tyypillinen internetkaupan arkkitehtuuri. (Myers ym. 2012, 195) 33

Taulukot

Taulukko 1. Testitapausten kategoriointia. (Myers ym. 2012, 122)..... 14
Taulukko 2. Havaintoraportin malli. (Spillner ym. 2015, 195)..... 30
Taulukko 3. Häiriön vakavuus. (Spillner ym. 2014, 196)..... 31
Taulukko 4. Vian prioriteetti. (Spillner ym. 2014, 196)..... 31
Taulukko 5: Havaintoraportin statustilat. (Spillner ym. 2014, 198.) 32
Taulukko 6. Havaintojen tyypit merkityksineen..... 44
Taulukko 7. Tavat havaintojen löytämiseksi merkityksineen. 44