LAUREA
AMMATTIKORKEAKOULU
Yhdessä enemmän

Testaus osana Zalandon Fleek-mobiilisovelluksen tuotekehitystä

Koskela, Minna

2016 Laurea

Laurea-ammattikorkeakoulu

Testaus osana Zalandon Fleek-mobiilisovelluksen tuotekehitystä

Minna Koskela
Tietojenkäsittely
Opinnäytetyö
Marraskuu, 2016

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittelyn koulutusohjelma
Tradenomi (AMK)

Minna Koskela

Testaus osana Zalandon Fleek-mobiilisovelluksen tuotekehitystä

Vuosi 2016 Sivumäärä 42

Opinnäytetyön aiheena on Zalandon Fleek-mobiilisovelluksen ostotapahtuman sekä sen eri toimintojen testaussuunnitelma. Fleek-sovellus yhdistää sosiaalisen median ja verkko-ostokset sekä toimii verkkoalustana muotimerkeille ja -brändeille omien tuotteidensa myyntiin ja markkinointiin.

Testaussuunnitelma sisältää Fleek-sovelluksen bisneskriittisten osien, kuten sisäänkirjautumisen ja ostotapahtumien, testauksen. Opinnäytetyö ottaa huomioon laadunvarmistuksen kokonaisuutena, mutta keskittyy testauksen automatisointiin. Teoriaosiossa perehdytään testauksen osuuteen ohjelmiston kehitysprojekteissa, testausprosessiin sekä testauksen menetelmiin – automaatiotestausta ja mobiilisovellusten testausta unohtamatta.

Testaussuunnitelman luomisessa käytettiin mallinnusta apuna Fleek-sovelluksen ostotapahtuman toiminnoista. Testaussuunnitelma sisältää 32 testitapausta sekä testauksen aloitus- ja lopetusehdot. Koska laadunvarmistus on muutakin kuin vain ohjelmiston testaamista, opinnäytetyössä otetaan kantaa myös testausprosessiin. Näitä huomioita kerättiin haastattelemalla Fleek-sovelluksen iOS-version kehittäjiä ja muita saman sovelluksen parissa työskenteleviä henkilöitä.

Jatkossa testaussuunnitelmaa voidaan käyttää testauksen tukena uusien ohjelmistoversioiden aloitus- tai regressiotestauksessa. Testaussuunnitelmaa on myös jatkossa kehitettävä ja uudistettava, mikäli ostotapahtumaan implementoidaan uusia ominaisuuksia tai käyttöliittymässä tehdään mittavia muutoksia.

Asiasanat: Mobiilisovellukset, Automaatiotestaus, Testausprosessi

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bachelor's Thesis

Minna Koskela

Testing as a Part of the Development of Zalando's Mobile Application Fleek

Year 2016 Pages 42

The main subject of this thesis is to create a test plan for Zalando's mobile application Fleek. The main focus is on the purchases and different functionalities around it. The Fleek application combines social media with online shopping and serves as a platform for fashion brands to sell and market their products.

This test plan consists of the business critical functions in Fleek application, such as login and purchase of the products. The thesis covers all the aspects of quality assurance, but focuses on the test automation. The theoretical section of the thesis is about testing as a part of a software development project, the processes and methods of the testing including automation and mobile software testing.

Modeling was used as an aid when the test plan for the purchase and its functions in Fleek application was created. The test plan consists of 32 test cases, including the start and exit criteria for the tests. As quality assurance is much more than merely testing, this thesis is also about the test process. The material for this was acquired by interviewing the iOS developers of Fleek application and other employees working with it.

The test plan can be used for smoke and regression testing on new builds. The test plan must also be refined after implementing new functionalities into the purchase process or after major changes in the user interface.

Avainsanat: Mobile application, Test automation, Test process

Sisältö

1 Johdanto

Tämän työn aiheena on mobiilisovelluksen testaus. Opinnäytetyön tuloksena syntyy toteutuskelpoinen ja kattava testaussuunnitelma Zalandon Fleek-sovelluksen ostotapahtuman toiminnoista. Fleek yhdistää sosiaalisen median sekä verkko-ostokset ja tarjoaa muotitaloille ja -merkeille teknisen alustan omien tuotteidensa myyntiin ja markkinointiin.

Zalandon tuotekehitys työskentelee Radical Agility -menetelmällä. Menetelmän perusteita tarkastellaan tarkemmin kappaleessa 6. Pohjimmaisena ideana menetelmässä on kehittäjien itsenäinen ja ketterä lähestymistapa päätöksentekoon. Ketterien menetelmien mukaisesti erillisiä testausasiantuntijoita ei käytetä, vaan kehittäjät itse toimivat myös testaajina. Menetelmän etuna on muun muassa kehittäjän vastuu oman koodin virheettömyydestä, kun vastuu laadusta ei ole enää erillisellä testausorganisaatiolla. Perimmäisenä ajatuksena menetelmän käyttämiselle on kuitenkin teknologiaorganisaation kehittämien tuotteiden laadun parantaminen.

Testaus on tärkeä osa ohjelmiston kehitysprojektia ja tarpeellista vielä ohjelmiston käyttöönoton jälkeen, mikäli ohjelmistoon tulee muutoksia. Mitä aikaisemmin testaus otetaan mukaan ohjelmistokehitykseen, sen parempi. Esimerkiksi vian korjaaminen järjestelmää suunniteltaessa maksaa vain 1-2 prosenttia siitä, mitä se maksaisi järjestelmän julkaisun jälkeen. (Hass 2008, 2; Kasurinen 2013, 11–12.)

Testaus on lähes poikkeuksetta ohjelmistoprojektin kallein työvaihe, sillä arvioilta noin 50 prosenttia ohjelmistoprojektin kokonaiskulusta menee testaukseen tai siihen liittyviin toimenpiteisiin. Laadunvarmistus on myös lähes mille tahansa yritykselle taloudellisesti tärkein työvaihe. Testausta ja tuotteen kannattavuutta vertailtaessa näiden kahden väliltä löytyy selvä yhteys: huolellisesti testatuista tuotteista saa paremman katteen kuin huolimattomasti testatuista tuotteista. (Kasurinen 2013, 11–12.)

2 Tavoite

Käyttöliittymän testaussuunnitelmalle on havaittu olevan tarvetta, sillä testauksen suunnittelu on ollut vaillinaista, eikä bisneskriittisen toimintojen, kuten esimerkiksi sisäänkirjautumisen ja ostosten maksamisen testauksen automatisoitiin ole ollut tarpeeksi resursseja. Lisäksi haasteita automaation suunnittelulle ja kehittämiselle on tuonut sovelluksen uusien versioiden julkaisujen nopea sykli. Tämän testaussuunnitelman tavoitteena on nopeuttaa ja helpottaa sovelluksen testausta sekä vapauttaa kehittäjiä muihin tehtäviin.

Suunnitelma sisältää sovelluksen käyttöliittymän testauksen sovelluksen tärkeimpien toiminnallisuuksien osalta, joita ovat rekisteröityminen, kirjautuminen sekä ostosten maksaminen. Tulevaisuudessa suunnitelmaa voi käyttää tulevien versioiden regressio- ja aloitustestauksen suunnittelussa ja toteutuksessa. Regressiotestauksen helpottamiseksi käyttöliittymän testaus suoritetaan mahdollisimman pitkälti automaatiolla, mutta koska kaikkea ei ole järkevää testata tällä tavoin, suunnitelma käsittää myös manuaalisen testauksen.

Lisäksi tässä opinnäytetyössä otetaan kantaa Fleek-sovelluksen testausprosessiin iOS-käyttöjärjestelmäversion kehitystiimissä. Koska Zalandon käyttämä Radical Agility – menetelmä antaa kehittäjille vapaat kädet tuotteidensa kehittämiseen ja testaamiseen, on prosessin hyötyjä ja tarpeita syytä tarkastella säännöllisin väliajoin. Testausprosessin tarkastelun tavoitteena on tuotteen laadun parantaminen prosessia mahdollisesti parantamalla.

Tämä suunnitelma ei ota kantaa organisaation testausstrategiaan eikä projektitasoisen testaussuunnitelman testauksen laajuuteen tai riskeihin. Suunnitelma ei kata myöskään resursseja, kuten testaajien määrää eikä heidän tehtäviä tai vastuita, testausympäristöä tai aikataulua. Tässä suunnitelmassa ei myöskään oteta huomioon mobiilituotteiden käytettävyystestausta.

3 Menetelmät

Opinnäytetyössä käytetään konstruktiivista tutkimusotetta ja tiedonkeruuseen puolistrukturoitua haastattelua. Ojasalon, Moilasen ja Ritalahden mukaan konstruktiivisen tutkimuksen tavoitteena on luoda konkreettinen tuotos, kuten tässä tapauksessa testaussuunnitelma. Konkreettinen tuotos on käytännön ongelmaan teoreettisesti perusteltu ja uudenlainen ratkaisu, joka vielä osoitetaan toimivaksi. Konkreettiseen kehittämisenkohteen tiedonhankintamenetelmät voivat olla kirjavia, sillä tarvittavaa tietoa kannattaa hankkia monella eri tavalla. (Ojasalo ym. 2009, 65–66.)

Mobiilisovelluksen käyttöliittymän testaussuunnitelmaa varten tarvittava aineisto hankitaan kirjallisuudesta sekä haastatteluin. Haastatteluihin valitut henkilöt ovat työskennelleet testattavana olevan mobiilisovelluksen parissa. Testaussuunnitelman menetelmäksi valikoitui mallipohjainen testaus, jota avataan tarkemmin kappaleessa 3.2.

3.1 Haastattelu

Yleisesti haastattelumenetelmiä on useita erilaisia: täysin strukturoitu haastattelu, puolistrukturoitu haastattelu tai täysin avoin haastattelu. Lisäksi haastattelun muotoja on useita; haastattelu voidaan toteuttaa joko yksilö-, pari- tai ryhmähaastatteluna. Näitä eri haastattelumuotoja voi käyttää myös toisiaan täydentävinä. Haastattelumenetelmän ja -muodon valinta riippuu siitä, millaista tietoa kehittämistyöhön tarvitaan, keitä haastateltavat henkilöt ovat ja mikä on tutkimuksen aihe. (Ojasalo ym. 2009, 95–97; Hirsjärvi, Remes, Sajavaara, Sinivuori 2009, 210–211.)

Strukturoidussa haastattelussa eli lomakehaastattelussa kysymykset on muotoiltu valmiiksi ja niiden järjestys on täysin ennalta määrätty. Puolistrukturoidussa haastattelussa eli teemahaastattelussa haastattelun aihepiiri on tiedossa, mutta kysymysten tarkka asettelu ja järjestys puuttuvat. Täysin avoin haastattelu on keskustelun kaltainen, jossa haastattelija ja haastateltava keskustelevat yleisesti aiheesta tai ongelmasta aktiivisesti ja tasavertaisesti. Käsiteltävä aihe tai ongelma voi myös muuttua avoimen haastattelun aikana. (Ojasalo ym. 2009, 97; Hirsjärvi ym. 2009, 208–209.)

Tähän opinnäytetyöhön parhaiten sopii puolistrukturoitu haastattelu, sillä haastattelun aikana saattaa olla tarvetta kysyä tarkentavia kysymyksiä. Haastatteluun on valittu eri työnkuvan omaavia henkilöitä Fleek-sovelluksen iOS-version kehitystiimistä. Haastattelun tarkoituksena on saada selville mobiilisovelluksen parissa työskenteleviltä henkilöiltä konkreettisia tietoja ja mielipiteitä nykyisistä kehitys- ja testausprosessien eduista sekä haitoista. Lisäksi haastatteluilla on tarkoitus saada selville kehittäjien motiivit tuotteen testaamiselle sekä mielipide omalle työlle testauksen näkökulmasta. Haastattelukysymykset löytyvät liitteestä 3.

3.2 Mallipohjainen testaus

Mallipohjaisella testauksella (model-based testing) tarkoitetaan testauksen lähestymis- ja suunnittelutapaa, jossa testitapaukset luodaan ja suoritettavat testit valitaan siten, että ne tarkastavat onko järjestelmä toteutettu suunnitelmassa määritellyn mallin mukaiseksi. Järjestelmän malli voidaan kuvata esimerkiksi UML-työkaluja (Unified Modeling Language) käyttäen. Mallipohjaisen testauksen työkaluilla voidaan kuvata hyvinkin yksityiskohtaisesti ja kaavamaisesti järjestelmän toiminnot, joista testitapaukset kootaan. Järjestelmä ja malli vastaavat toisiaan, kun toimivat testitapaukset toimivat oikein järjestelmässä. (Kasurinen 2013, 75.)

ISTQB:n (International Software Testing Qualifications Board) mukaan mallipohjaisen testauksen suurin hyöty on sen vaikuttavuudessa – mallit ovat yleensä helppoja tulkita ja niillä voidaan edistää kommunikaatiota ja ymmärrystä järjestelmän toiminnoista sidosryhmille. Toinen suuri hyöty on tehokkuudessa – samasta mallista voidaan tehdä useita eri testipaketteja ja sillä voidaan kattaa useita eri testaustasoja ja –tyyppejä eri kriteereitä käyttäen. Mallipohjaisella testauksella ei kuitenkaan yksinään kannata suunnitella koko järjestelmän testaamista, vaan käyttää sen tuomaa hyötyä lisänä. (ISTQB 2015, 8-9.)

Tässä työssä mallipohjaista testausta käytetään käyttöliittymän eri tilojen testauksen suunnittelussa ja testitapauksen luomisessa. Pääpaino testitapauksissa on onnistuneissa ostotapahtumassa tapahtuneissa siirtymissä ja syötteissä. Mallipohjaisen testauksen pohjana käytetty malli on liitteessä 2.

4 Zalando

Zalandon ovat perustaneet Robert Gentz ja David Schneider vuonna 2008 Saksassa. Heidän liiketoimintansa perusajatuksena oli tarjota erityylisiä ja -kokoisia kenkiä kuluttajalle riippumatta hänen sijainnistaan. Nykyään Zalando toimittaa tuotteita – vaatteita, urheiluvälineitä ja kenkiä - jopa 15 maahan ja on yksi Euroopan johtavista muotiin keskittyneistä verkkokaupoista. Statistan vuonna 2015 tekemän tutkimuksen mukaan Zalando oli kolmanneksi suurin verkkokauppa kotimaassaan Saksassa yli miljardin euron myynnillä. (Zalando 2016a; Ecommerce News 2016.)

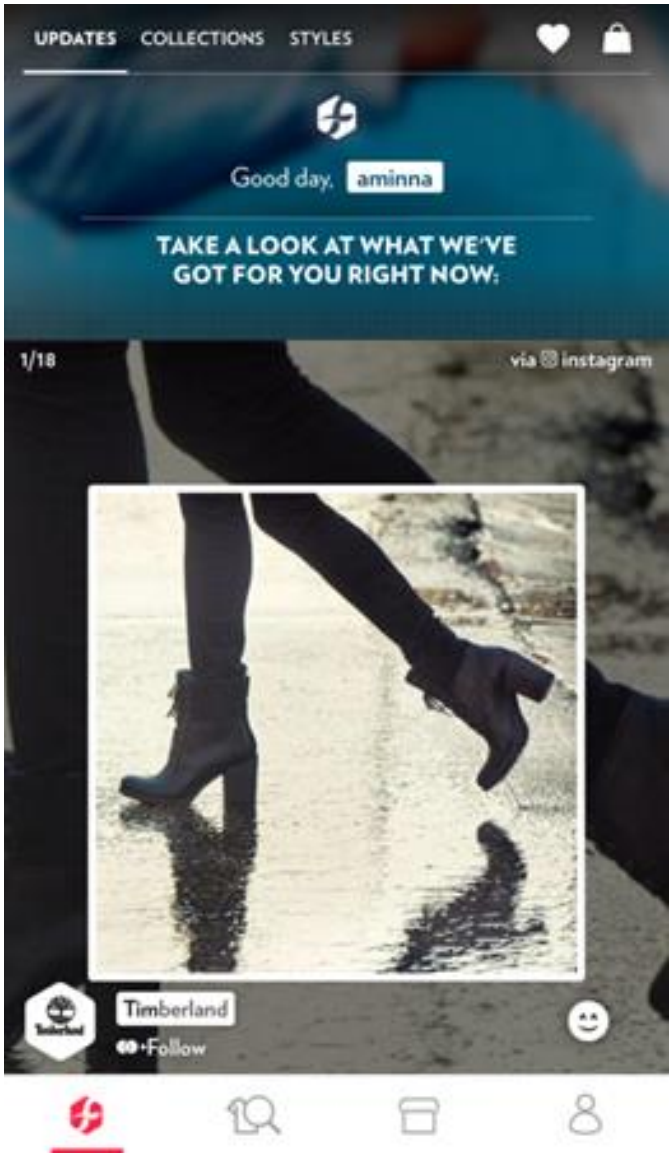
Vuonna 2008 Zalando käytti verkkokaupassaan standardoitua järjestelmää, joka pian osoittautui riittämättömäksi teknologiseksi ratkaisuksi. Zalando alkoikin kehittää omaa teknologiaansa vastatakseen kuluttajien sekä muotitalojen odotuksiin ja pysyäkseen mukana nopeasti vaihtuvaan valikoiman ja yrityksen kasvun kanssa. Nykyään Zalandon itsensä kehittämä alusta on kaupanteon ydin – se kattaa kaiken kaupan toimituksesta itse verkkokauppaan ja Zalandon teknologinen osasto on sen kantava voima. (Zalando 2016b.)

Zalandolla on toimipisteitä kolmessa eri paikassa: Saksassa, Irlannin Dublinissa sekä Helsingissä. Useiden vaihtoehtojen tutkimisen jälkeen Helsingin toimipisteen avaamiseen päädyttiin Suomessa olevan vahvan insinööriosaaminen vuoksi. Helsingin teknologiakeskus keskittyy erityisesti Zalandon uuden mobiilialustan kehittämiseen. (Lappalainen 2015).

4.1 Fleek

Fleek on muotiin erikoistunut applikaatio, josta kuluttaja voi ostaa muun muassa kenkiä, vaatteita, urheiluvälineitä ja asusteita. Kuvassa 1 on applikaation etusivu. Tähän mennessä Fleek on julkaistu Saksassa ja Itävallassa (Fleek 2016).

Zalandon tuotevastaava Daake (2015) kertoo Fleek-applikaation kehityksen alkaneen kuluttajien tuottaman internetissä olevan sisällön (User-generated content) tutkimisella. Moens, Li ja Chua (2014, 7) määrittelevät, että kyseiseen menetelmään kuuluu esimerkiksi blogien, keskustelufoorumien, chattien, videoiden tai sosiaalisen median sisällön analysointi. Daake (2015) jatkaa kertomalla kuluttajien palautteista: he mielellään tekevät ostopäätöksiä sekä hakevat inspiraatiota ostoksilleen Instagramissa julkaistujen kuvien perusteella ja turhautuvat, mikäli haluttua tuotetta ei voi ostaa suoraan. Näistä syistä Zalando päätti tehdä Instagramissa julkaistuista kuvista helposti ostettavia. Instagram onkin täydellinen lähde muodista kiinnostuneille jopa yli 300 miljoonalla päivittäisellä käyttäjällään (Lee 2016).



Kuvio 1: Fleek-applikaation etusivu (Fleek 2016)

Sovelluksen on tarkoitus palvella asiakkaita usealla eri tavalla. Kuluttaja-asiakkaita palvellaan tarjoamalla heille yksilöityjä ja räätälöityjä suosituksia ja mieltymyksiä heidän kiinnostuksiensa perusteella. Brändeille ja muille kumppaneille tarjotaan kohdennettua asiakaskuntaa ja heille räätälöityä markkinointia. Lisäksi kumppaneille tarjotaan alustaa, jossa myydä omia tuotteita ilman omia verkkosivuja. (Zalando 2015.)

4.2 Fleek ja laadunvarmistus

Mobiiliverkkokaupan laatuun panostaminen on kannattavaa – RetailMeNot-yrityksen teettämän tutkimuksen mukaan mobiililaitteilla, johon tässä tutkimuksessa lasketaan matkapuhelimet, tabletit sekä muut mobiililaitteet, tehtyjen ostojen summa oli Euroopassa kasvanut vuodesta 2014 vuoteen 2015 81 prosenttia. Saksassa kasvua oli ollut jopa 100 prosenttia. Mobiilisti tehdyissä verkkokauppaostoksissa kasvu oli ollut huomattavasti suurempaa kuin tietokoneella

tehdyissä ostoksissa, joissa kasvua oli ollut Euroopassa viisi prosenttia ja Saksassa lähes kuusi prosenttia. Toisaalta tutkimuksen mukaan vuonna 2015 tietokone oli silti ollut suositumpi laite ostoksien tekemiseen, sillä Euroopassa 77 prosenttia ja Saksassa 72 prosenttia verkkokauppaostoksista tehtiin tietokoneella. (Retail Research 2016.)

Vaikka Fleek on saanut asiakkaiden keskuudessa hyvät arvostelut – nykyiselle versiolle on annettu iTunesin (2016) mukaan viisi tähteä - haastatteluiden perusteella sovelluksen kehittäjät arvioivat sovelluksen laadun keskimäärin kolmen tähden arvoiseksi. Perusteluina oli lähes poikkeuksetta kokonaisuuden sekavuus sekä se, että sovellus ei ole kaikin puolin yhdenmukainen. Tämä johtunee osittain siitä, että Fleek-sovellusta on rakennettu toisen osapuolen tekemän koodin päälle, jonka vuoksi teknistä velkaa on kertynyt. Lisäksi haastatteluissa kävi ilmi, että sovelluksen suunnittelu on elänyt matkan varrella, jolloin toteutuskaan ei vastaa enää alkuperäisiä suunnitelmia.

5 Ohjelmistotestaus

Opinnäytetyön tietoperustaa käsitellään tässä luvussa. Tietoperusta rakentuu ohjelmistotestauksen perusteista sekä yleisestä ohjelmiston kehitysprosessista testauksen näkökulmasta. Lisäksi opinnäytetyö perehtyy testausautomaatioon ja mobiilisovellusten testaamiseen.

5.1 Ohjelmistotestauksen perusteita

Myersin, Badgettin & Sandlerin mukaan ohjelmistotestaus määritellään usein toiminnaksi, jolla varmistetaan ohjelmiston toimivuus tai vikojen puuttuminen. Pelkästään toimivuuden ja oikeellisuuden testaaminen ei välttämättä kuitenkaan lisää ohjelmiston luotettavuutta tai laatua – testaaja voi pyrkiä tätä tavoitetta kohden ja jättää alitajuisesti tärkeitä kohtia testaamatta. Siksi parempi lähestymistapa ohjelmistotestaukseen on olettaa, että ohjelmisto sisältää vikoja ja testaajan tehtävä on löytää ne. Toisaalta Spillner Linz, Schaefer täydentävät, että ohjelmistotestauksella voidaan silti saavuttaa haluttu lopputulos, vaikka ohjelmistosta ei löytyisi yhtään tai hyvin vähän vikoja. (Myers ym. 2012, 5-6; Spillner ym. 2014, 10.)

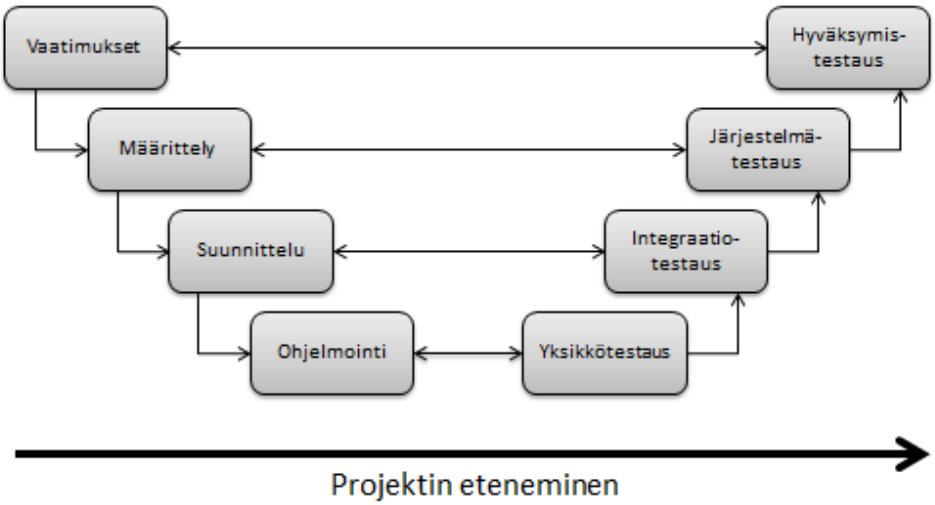
Yleisesti virheistä tai vioista puhuttaessa ne ovat lähes toistensa synonyymejä, mutta ohjelmistotestauksessa termeillä on viralliset määritelmät. Ihmisen aiheuttama virhe tai erehdys (mistake) aiheuttaa vian eli bugin (fault, bug) ohjelmistoon, sen lähdekoodiin tai dokumentaatioon. Vikoja voi ilmetä ohjelmistossa sen kehittämisestä tai siihen tehdyistä muutoksista asti. Häiriöksi (failure) kutsutaan tilannetta, jossa ohjelma toimii eri tavoin kuin sen pitäisi. Häiriöitä voi aiheuttaa mikä tahansa asia, kuten muut järjestelmän osat, käytetty laitteisto tai vaikka huonepöly. (Spillner ym. 2014, 7; Kasurinen 2013, 50.)

5.2 Testaus osana ohjelmistotuotantoa

Erilaiset testaustehtävät liittyvät järjestelmän kehitysprojektiin ja erilaisiin kehitysprojektimalleihin liittyy erilaisia lähestymistapoja testaukseen. Yksi varhaisimpia ohjelmiston kehityksen prosessimalleja on vesiputousmalli, joka on idealtaan hyvin yksinkertainen: kun edellinen työvaihe on päättynyt, seuraava voi alkaa. Testauksen näkökulmasta tällainen kehitysmalli on erittäin epäkäytännöllinen, sillä mallin mukaan testaus on vain viimeinen vaihe ennen ohjelmiston julkaisua eikä koko ohjelmiston kehityksen aikainen tehtävä. (Spillner ym. 2014, 17–18.)

Kasurisen (2013, 51) mukailemassa ohjelmiston prosessimallissa, jota kutsutaan yleiseksi V-malliksi (kuvio 2), testausta tehdään jokaisessa järjestelmän kehityksen vaiheessa. Hassin (2008, 4) mukaan yleinen V-malli on vesiputousmallin laajennus, jossa testaus on enemmän kuin vain viimeinen tarkastus. V-mallin vasemmalla puolella ovat ohjelmiston kehittämiseen ja tuottamiseen liittyvät vaiheet, joiden perusteella konkreettinen ohjelmisto tai järjestelmä

syntyy. Mallin oikealla puolella ovat testaustasot, joilla varmistetaan, että valmis tuote toimii ja on suunnittelun mukainen. V-mallissa esitettyjä testaustasoja käydään läpi myöhemmin.



Kuvio 2: V-mallissa yksikkötestaus, integraatiotestaus ja järjestelmätestaus johtavat hyväksymistestaukseen, jonka jälkeen ohjelma on valmis käyttöönottoon

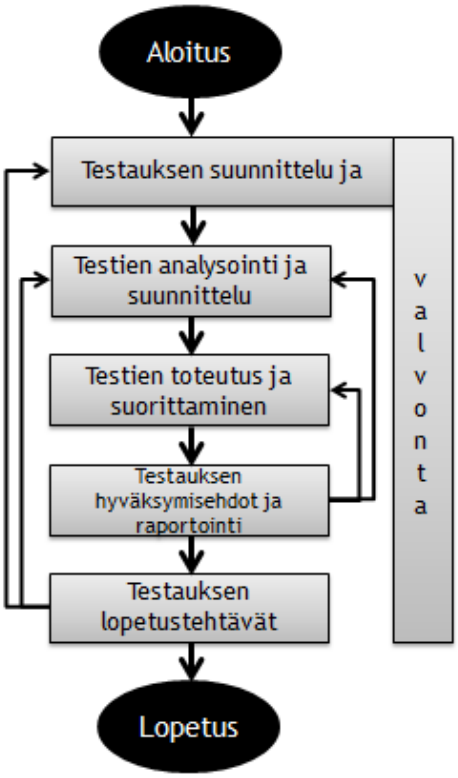
Hassin mukaan (2008, 7) suunnitelmalähtöisten prosessimallien, kuten edellä mainitun vesiputousmallin, rinnalle on noussut viime vuosina ketteriä menetelmiä (agile methods). Kasurinen (2013, 27) arvelee, että ketterä menetelmä terminä voi viitata suunnitelmalähtöisten ohjelmistoprosessien olevan raskaita ja hitaita byrokratian tai hallinnon takia. Ero näiden kahden menetelmän välillä on Kasurisen (2013, 27) mukaan seuraava: siinä missä suunnitelmalähtöiset menetelmät pyrkivät varautumaan mahdollisimman hyvin mahdollisiin ongelmatilanteisiin, pyrkivät ketterät menetelmät taasen reagoimaan mahdollisimman nopeasti esille tuleviin ongelmatilanteisiin.

Erilaisia ketteriä menetelmiä on useita, mutta Myers ym. (2012, 176) luettelevat menetelmille kolme yhteistä nimittäjää: asiakkaan läsnäolo ja osallistuminen, merkittävässä roolissa oleva testaus sekä lyhyet iteratiiviset kehityssyklit. Myers ym. (2012, 178) mukaan ketterässä testaamisessa kaikki ohjelmiston kehitysprojektiin osallistuvat ovat mukana testausprosessissa – asiakkaat ovat mukana hyväksymistestauksessa käyttötapauksia luoden ja kehittäjät tekevät testaajien kanssa yhteistyötä järjestelmän toiminnallisuuksien testaamisessa automaattisesti. Myers ym. (2012, 178) korostavat, että ketterissä projekteissa testaus vaatii kaikilta sitoutumista, kommunikointia ja yhteistyötä. Lisäksi ketterissä projekteissa, joissa järjestelmä kehittyy nopeaan tahtiin, kattava regressiotestaus on ehdoton vaatimus jokaisella iteraatiokierroksella (Hass 2008, 7).

5.3 Testauksen perusprosessi

Testaus on kuitenkin prosessi, kuten mikä tahansa muukin ohjelmiston kehityksen aikainen toiminta. V-mallin tapa esittää testausprosessia ei kuitenkaan ole kaikkein hedelmällisin tapa kuvata kuinka prosessi toimii ohjelmistoprojektissa. Jotta testauksen aikaiset tehtävät voidaan sulauttaa ohjelmistoprojektiin, tarvitaan yksinkertaisempi kuvaus eri testaustehtävistä, kuten Spillnerin ym. esittelemä kuvio 3 osoittaa. Tässä yleisessä testausprosessissa tehtävät on lajiteltu seuraavasti: testauksen suunnittelu ja valvonta, testien analysointi ja suunnittelu, testien toteutus ja suorittaminen, testauksen hyväksymisehdot ja raportointi sekä testauksen lopetustehtävät. (Spillner ym. 2014, 19.)

Prosessia voi soveltaa lähes mihin tahansa testaustapaan tai -tekniikkaan, eikä testauksen aikaisten aktiviteettien tarvitse noudattaa tiukasti samaa kaavaa. Kuten kuviossa 3 esitetään, voidaan testauksen hyväksymisehtoja arvioidessa huomata esimerkiksi testeissä epäloogisuutta, jonka jälkeen täytyy palata takaisin testien suunnitteluun. Tämän vuoksi testaustoimia ei ole syytä suorittaa tiukassa peräkkäisessä järjestyksessä eikä sitä tule ottaa tieteellisenä totuutena. Kyseessä on malli, jota tarvitsee räätälöidä yrityksen tai projektin tarpeiden mukaan. (Spillner 2014, 19; Hass 2008, 35–37.)



Kuvio 3: Yleinen testausprosessi

Testausprosessi alkaa ohjelmiston kehitysprojektin alussa testauksen suunnittelulla. Testauksen suunnittelun tärkein vaihe on testausstrategian määrittely. Strategiassa päätetään riskiarvioinnin perusteella mitä testataan ja missä järjestyksessä - ohjelmiston kriittisille osille tulee antaa enemmän huomiota. Strategian tavoitteena on jakaa järjestelmän osille optimaalisesti testausresursseja. Tämän lisäksi suunnitteluvaiheessa tehdään kirjallinen testaussuunnitelma, josta löytyvät testauksen tehtävä ja tavoite sekä resurssien jako. Resursseilla tarkoitetaan testaajien määrää ja tehtäviä, aikataulua, kalustoa sekä apuvälineitä. Vaikka nämä asiat päätetään projektin alussa, testauksen kulkua tulee valvoa koko testausprosessin ajan ja tehdä suunnitelmaan tarvittavia muutoksia. (Spillner ym. 2014, 19–20.)

Testien analysointi ja suunnittelu aloitetaan testaussuunnitelman arvioinnilla. Tarkoituksena on muuttaa testauksen tavoitteet konkreettisiksi testitapauksiksi käyttämällä apuna esimerkiksi ohjelmiston määrittelyitä ja vaatimuksia tai riskianalyysin tuloksia. Kun testitapaukset on viimeistelty ja laadittu, pystytetään vaadittava testiympäristö ja suoritetaan testit tärkeimmistä testeistä alkaen. Mikäli poikkeuksia havaitaan, ne kirjataan ylös. Vikojen korjaamisen jälkeen testaustehtävät toistetaan uudelleen – niin ne testit, jotka epäonnistuivat, kuin myös ne testit, joihin korjaus on voinut vaikuttaa. (Spillner ym. 2014, 22, 26–27.)

Testauksen hyväksymisehtojen arviointi on toimenpide, jolloin testien tuloksia arvioidaan suunnittelussa määriteltyjä tavoitteita vasten. Mikäli testauksen lopetusehdot eivät täyty, voidaan testejä lisätä tai määriteltyjä hyväksymisehtoja muuttaa. Mutta mikäli hyväksymisehdot ovat täyttyneet, kirjoitetaan testauksen kulusta yhteenvetoraportti eri osapuolille, kuten projektipäällikölle, testipäällikölle tai jopa asiakkaalle. Alemmilla testaustasoilla, kuten yksikkötestaustasolla, vähemmän formaali raportti voi olla riittävä. Testauksen lopetustehtäviin kuuluu testauksen aikaisten tietojen, kokemusten kerääminen ja analysoiminen. Näitä tietoja voidaan käyttää hyväksi muissa projekteissa testauksen tason parantamiseksi. (Spillner ym. 2014, 28, 30–31.)

5.4 Testaustasot

Perinteisessä mielessä voidaan sanoa, että ohjelmiston tai järjestelmän testaus on eri kehitysvaiheissa tehtävä mekaaninen työsuoritus. Kuten V-mallissa esitettiin, ohjelmiston kehitysprojektissa on useita kehitystasoja, joita seuraa useita eri testaustasoja. Eri testaustasoissa voi myös olla useita eri testausmenetelmiä, eikä niihin oteta tässä kantaa. (Kasurinen 2013, 50–51.)

Yksikkötestaus (unit testing, component testing) tarkoittaa yhden yksittäisen yksikön, moduulin tai komponentin testaamista toteutuksen yhteydessä. Yksikkötestaus perustuu joko yksikön vaatimuksiin ja määrittelyihin. Yksiköt testataan yleensä eristettynä muusta järjestelmästä, jotta komponentin toiminta voidaan varmentaa eikä testi epäonnistu mahdollisten

ulkoisen vaikutusten vuoksi. Monesti kuitenkin yksiköillä tai komponenteilla on vuorovaikutusta keskenään ja näissä tapauksissa testausta voidaan helpottaa rakentamalla testikomponentteja, joiden tehtävä on varsinaisen järjestelmän komponenttien välisen liikenteen simulointi. (Spillner ym. 2014, 42–43; Kasurinen 2013, 52.)

Integrointitestauksessa (integration testing) uusi yksikkö, moduuli tai komponentti on liitetty osaksi aiemmin testattuja ja toimiviksi todettuja komponentteja. Tarkoituksena on saada koko järjestelmä toimimaan yhtenä toimintakykyisenä kokonaisuutena. Integrointitestaus on yleensä sitä vaikeampaa, mitä monimutkaisemmaksi ohjelmisto kasvaa, joten integrointitestaus kannattaa suorittaa joka kerta, kun ohjelmistoon implementoidaan uusia komponentteja – missään tapauksessa integrointitestausta ei kannata jättää siihen pisteeseen asti, kunnes ohjelmiston kaikki elementit on saatu valmiiksi. (Spillner ym. 2014, 50–55.)

Järjestelmätestaus (system testing) tarkoittaa koko ohjelmiston tai järjestelmän testausta sen jälkeen, kun kaikki ohjelmiston osat on koottu yhdeksi toimivaksi kokonaisuudeksi ilman testikomponentteja. Järjestelmätestauksessa tarkoituksena on löytää mahdolliset viimeiset kriittiset viat. Toisaalta järjestelmätestauksessa voidaan ohjelmiston toimintoja peilata järjestelmän määrityksiin tulevien käyttäjien tai asiakkaiden näkökulmasta ja testattavia kohteita voivat olla myös esimerkiksi ohjelmiston dokumentaatiot ja käyttöohjeet. (Kasurinen 2013, 56; Spillner ym. 2014, 58–59.)

Hyväksymistestaus (acceptance testing) on perinteisen testausprosessin viimeinen työvaihe. Tavoitteena ei enää ole uusien vikojen löytäminen, vaan järjestelmä riittävän korkean laadun ja kyvykkyyden osoittaminen. Hyväksymistestauksessa varmistetaan, että ohjelmisto vastaa asiakkaan antamia vaatimuksia ja yleensä asiakas tai loppukäyttäjät ovat myös mukana testauksessa. Toisin kuin järjestelmätestauksessa, hyväksymistestausvaiheessa on hyvin tavallista, että järjestelmä testataan sen kohdeympäristössä. (Hass 2008, 15; Kasurinen 2013, 57.)

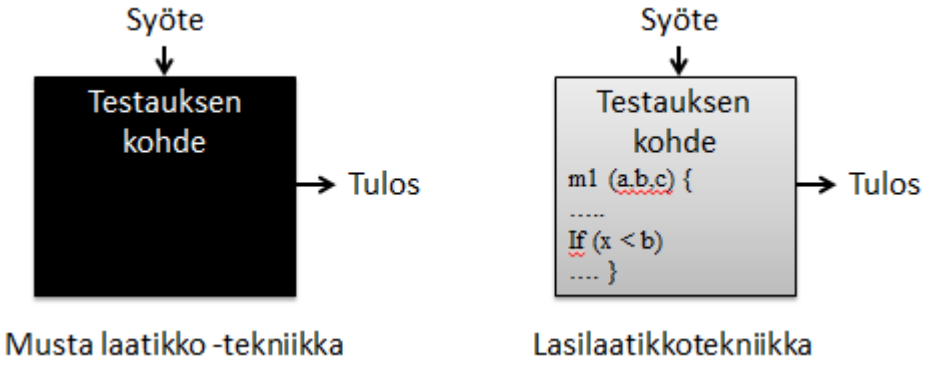
5.5 Testaustekniikat, -tyypit ja menetelmät

Järjestelmän kehitysvaiheessa testaus noudattaa usein suurin piirtein edellisessä kappaleessa esitettyjä testaustasoja. Näiden testaustasojen sisällä testausta voi tehdä kuitenkin erilaisin menetelmin ja tekniikoin. Näitä erilaisia tapoja ja niiden määritelmiä on useita, mutta tässä kappaleessa käydään läpi niistä yleisimmät.

5.5.1 Mustalaatikko-, lasilaatikko- ja harmaalaatikkotestaus

Testaustekniikat jaetaan perinteisesti kahteen pääryhmään: mustalaatikko- ja lasilaatikkotekniikoihin. **Mustalaatikkotekniikoissa** (black-box testing), joita voidaan kutsua myös määrittelypohjaisiksi tekniikoiksi, laadittavat testit pohjautuvat ohjelmiston määrittelyihin, vaa-

timuksiin tai jopa testaajan omaan intuitioon ja kokemukseen. Kuten Spillnerin ym. (2014, 109) kuviossa (kuvio 4) esitetään, **lasilaatikkotekniikoissa** (white-box testing), joita voidaan kutsua myös rakenteellisiksi tai rakenteeseen perustuviksi tekniikoiksi, testit perustuvat komponentin tai järjestelmän sisäiseen rakenteeseen. (Spillner ym. 2014, 109–110, 145.)



Kuvio 4: Mustalaatikkotekniikoiden ja lasilaatikkotekniikoiden erot

Mustalaatikkotestauksessa testaaja ei siis tiedä mitä ohjelman sisällä tapahtuu. Sen sijaan hän keskittyy testauksessa ohjelman toimintoihin antamalla sille erilaisia syötteitä ja vertaamalla niitä vaatimuksiin ja määritelmiin. Mustalaatikkotestaus onkin hyvin yksinkertainen menetelmä missä tahansa testauksen työvaiheessa. Tästä yksinkertaisesta luonteesta johtuen mustalaatikkotestejä on kannattavaa automatisoida regressiotestauksen ja toistuvan työn vähentämiseksi. (Myers ym. 2012, 9; Kasurinen 2013, 66.)

Lasilaatikkotestauksen lähtökohtana on, että testaaja pääsee käsiksi ohjelman koodiin tarkastellakseen sen sisäistä toimintaa. Lasilaatikkotestausta voidaan soveltaa ohjelmiston kehitysvaiheen alkupäässä suoritettaviin yksikkötesteihin tai integrointitesteihin, sillä lasilaatikkomenetelmät soveltuvat parhaiten yksinkertaisten koodinpätkien testaamiseen. Vaikka lasilaatikkotestaus on syvällisempää kuin mustalaatikkotestaus, se tarvitsee kuitenkin testaajalta ymmärrystä ohjelmoinnista ja järjestelmän toiminnasta aina lähdekooditasolle asti. Tämän lisäksi lasilaatikkotestaus ei ota kantaa esimerkiksi koodista puuttuviin ominaisuuksiin. Näistä syistä lasilaatikkotestaus soveltuu huonosti ainoaksi laadunvarmennustekniikaksi. (Spillner ym. 2014, 160; Kasurinen 2013, 67–68.)

Harmaalaatikkotestaus (Grey-box testing) on nimensäkin puolesta yhdistelmä mustalaatikko- ja lasilaatikkotestausta. Molemmista tekniikoista on harmaalaatikkotestaukseen yhdistetty parhaat puolet: mustalaatikkotestauksesta testitapaukset, jotka ovat luotu ohjelmiston vaatimusmäärittelyistä ja lasilaatikkotestauksesta testit, joissa tarkastellaan järjestelmän sisäistä toimintaa. Käytännössä tämä tarkoittaa mallikattavuutta, jossa kaikki vaatimukset on täytetty sekä koodikattavuutta, jossa koko lähdekoodi on tarkastettu. (Kasurinen 2013, 68.)

5.5.2 Toiminnallinen, ei-toiminnallinen ja rakenteellinen testaus

Testaustyypit voidaan jakaa neljään ryhmään – toiminnalliseen, ei-toiminnalliseen, rakenteelliseen ja muutokseen perustuvaan testaukseen. Koska eri testaustasoilla on erilaiset tavoitteet ja päämäärät, sopivat eri tasoille myös eri testaustyypit. Tässä kappaleessa esitellään kolme ensimmäistä testaustyyppiä, sillä muutokseen perustuva, toisin sanoen regressiotestaus, käsitellään myöhemmin. (Spillner ym. 2014, 69–70.)

Järjestelmän tai komponentin toiminnot ovat sitä, ”mitä” järjestelmä tekee. **Toiminnallinen testaus** (functional testing) tutkii ohjelmiston tai järjestelmän ulkoista käyttäytymistä mustalaatikkotekniikoin. Testit perustuvat siis järjestelmän toimintoihin ja ominaisuuksiin, jotka voidaan kuvata esimerkiksi vaatimusmäärittelyissä ja käyttötapauksissa tai ne voivat olla dokumentoimattomia. Toiminnallista testausta suoritetaan yleensä järjestelmätestaustasolla. (Spillner ym. 2014, 70–71.)

Ei-toiminnallisessa testauksessa (non-functional testing) testataan ”kuinka” järjestelmä toimii. Ei-toiminnallinen testaus tutkii ohjelmiston tai järjestelmän ulkoista käyttäytymistä ja piirteitä, joita voidaan mitata jollain asteikolla. Useat ei-toiminnalliset testit ovat luotu mustalaatikkotekniikoin ja niitä ovat muun muassa suorituskykytestaus, kuormitustestaus, rasitustestaus, käytettävyystestaus ja luotettavuustestaus. (Spillner ym. 2014, 72–74.)

Rakenteellista testausta (structure-based testing), jota suoritetaan lasilaatikkotekniikoin, voidaan tehdä kaikilla testaustasoilla, mutta yleensä sitä käytetään varsinkin yksikkö- ja integrointitestauksessa. Rakenteeseen perustuvia tekniikoita voidaan käyttää esimerkiksi koodikattavuuden mittaamiseen. Tällaista testausta on edullisinta ja helpointa suorittaa määrittelypohjaisten tekniikoiden jälkeen, jolloin rakenteeseen perustuvaa testausta voidaan käyttää halutun testikattavuuden saavuttamiseksi. (Spillner ym. 2014, 74; Hass 2008, 198.)

5.5.3 Muita menetelmiä

Kasurisen mukaan (2013, 70) joitain testauksen menetelmiä kannattaa käyttää vasta järjestelmälle, joka on lähes valmis tai testattavissa enimmillä tai kaikilla ominaisuuksilla. Tässä kappaleessa esitellään joitain tärkeimpiä menetelmiä. Nämä menetelmät sopivat käytettäväksi tässä opinnäytetyössä esitellyn suunnitelman lisäksi.

Aloitustestauksella (smoke testing) tarkastetaan, että ohjelmiston perusasiat toimivat. Tarkastelussa voi olla kohtia kuten ’lähteekö ohjelma käyntiin’ tai ’onnistuuko sisäänkirjautuminen’. Aloitustestauksella varmistetaan, että järjestelmän uusin versio toimii ennen varsinaisen testauksen aloitusta – mikäli tässä testauksessa esimerkiksi ilmenee, että ohjelmistoon

kirjautuminen ei onnistu, voidaan testaus keskeyttää ja jatkaa vanhalla versiolla tai odottaa uutta julkaisua. (Kasurinen 2013, 72; Kaner, Bach & Pettichord 2002, 161–162.)

Tutkivalla testauksella (exploratory testing) tarkoitetaan Halmeen (2004, 4-5) mukaan testaustapaa, jossa virheitä etsitään testaajan kokemuksen ja aikaisempien tulosten pohjalta. Tutkiva testaus eroaa muista testausmenetelmistä siten, että testitapauksia ei ole suunniteltu etukäteen. Oleellista tutkivassa testauksessa on se, että testejä ei suoriteta joka kerta samalla tavalla, joten testien sekä testauksen muuttuminen joka kerta parantaa virheiden löytämismahdollisuuksia. Lähes jokainen testaaja on jossain vaiheessa käyttänyt tutkivaa testausta esimerkiksi tutkiessaan löytämäänsä vikaa tarkemmin. Vaikka Kasurisen mukaan tutkivaa testausta menetelmänä on käytetty kauan, on se saanut huonon maineen muun muassa siksi, että tehtyä testausta ei pysty helposti dokumentoimaan tai jäljittämään. Tutkiva testaus voi kuitenkin olla Halmeen mukaan testaajalle mielenkiintoinen ja motivoiva menetelmä haasteellisuutensa ja vapautensa vuoksi, sillä testaajan kyvyt ja taidot ovat tässä tärkeimmät yksittäiset testauksen onnistumiseen vaikuttavat tekijät – kunhan löydetyt viat koetaan onnistumisina, eikä takaiskuina. (Halme 2004, 4-5; Kasurinen 2013, 74.)

Regressiotestaus tarkoittaa Kasurisen (2013, 68–69) mukaan uudelleentestaamista – se ei siis ole yksittäinen työtaso eikä varsinainen menetelmä, kuten esimerkiksi tutkiva testaus tai harmaalaatikkotestaus. Hass (2008, 71) täydentää, että regressiotestauksessa jo kertaalleen testattuja ja toimivaksi todettuja toiminnallisuuksia testataan uudelleen, ettei uusia vikoja ole syntynyt eikä aiemmin korjattuja vikoja ole ilmaantunut järjestelmään tehtyjen muutosten seurauksena. Regressiotestausta tehdään aina uuden version julkaisun jälkeen ja sitä voidaan tehdä kaikilla testaustasoilla (Mathur 2008, 337).

5.6 Automaatiotestaus

Kun ohjelmistoa käytetään toisen ohjelmiston testaamiseen, sitä kutsutaan automaatiotestaukseksi. Sen tavoitteena on mahdollisimman monen mielikuvituksettoman, usein toistuvan ja ikävän testin automatisointi. Pohjimmaisena ajatuksena testiautomaation käytölle on ajan ja rahan säästäminen sekä tuotteen nopeamman kehittämisen mahdollistaminen. (Kaner ym. 2002, 103; Hass 2008, 361.)

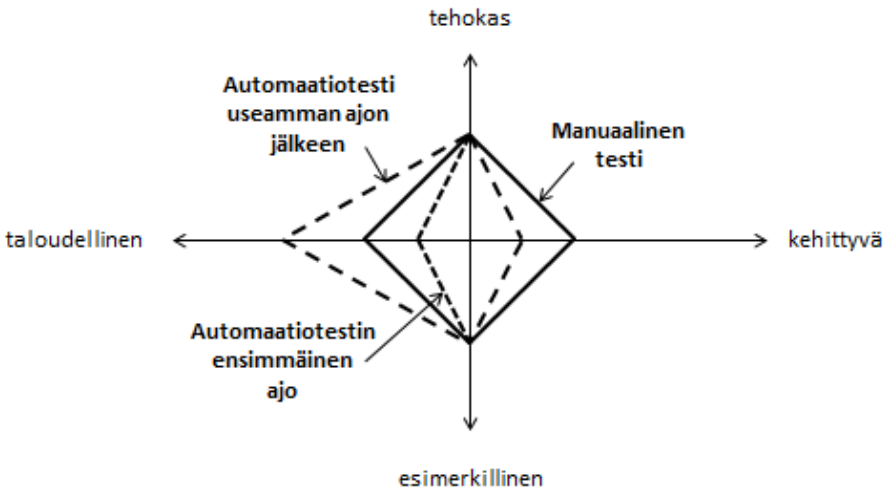
5.6.1 Automaatiotestauksen hyödyt

Nykyään järjestelmät ovat monimutkaisia ja sisältävät paljon erilaisia toimintoja. Niiden testaaminen voi sisältää tähtitieteellisen määrän mahdollisia testitapauksia, eikä testaajan aika aina riitä kuin testaamaan murto-osan niistä. Suurin osa automaation hyödystä saavutetaan sellaisen testausnopeuden ansiosta, mihin ihminen ei pysty. Nopeus on ohjelmistokehityksessä

valttia – mitä aikaisemmin vika on huomattu, sitä helpompi ja edullisempi se on korjata. (Fewster, Graham 1999, 4; Kaner ym. 2002, 104.)

Balasubramiam (2014) on listannut artikkelissaan automaattitestauksen hyötyjä, joista tärkeimmät liittyvät ajankäyttöön, nopeuteen ja luotettavuuteen. Ensiksi, järjestelmän automaatiotestaus voidaan hoitaa toimistoaikojen ulkopuolella, jolloin testaajalla jää aikaa muihin testauksen aikaisiin tehtäviin kuin manuaaliseen testaamiseen. Toiseksi, automaatiotestejä voidaan toistaa useita kertoja, kun taasen manuaaliset testit vaativat testaajan työpanoksen jokaisella suorituskerralla. Kolmanneksi Balasubramaniamin (2014) mukaan automaatiotesteillä löydettyjä vikoja ei tarvitse toistaa uudelleen, sillä automaatiotestaus on luotettavampaa kuin manuaalitestaus, mikäli testit ovat koodattu oikein. Näiden lisäksi Forsten (2016) lisää automaatiotestauksen hyödyksi toistettavuuden: samoja testejä voi ajaa jokaisen pienenkin muutoksen jälkeen, jolloin viat on mahdollista löytää helpommin, kun ainut muuttunut asia on ohjelmisto itse. Kasurinen (2013, 77–78) kuitenkin huomauttaa, että automaatiotestauksella ei niinkään etsitä uusia vikoja ohjelmistosta, vaan varmistetaan, että aiemmin toimineet osat toimivat vieläkin.

Hyvällä ja laadukkaalla testitapauksella on neljä ominaisuutta: tehokkuus, eli miten hyvin testi löytää vikoja; esimerkillisyys, eli kuinka monta testiä testi testaa samaan aikaan; taloudellisuus, eli miten paljon testitapauksen suorittaminen maksaa ja kehittyvyys, eli miten paljon kunnossapitoa testi vaatii. Fewsterin ja Grahamin esittämässä kuviossa (kuvio 5) näytetään nämä neljä ominaisuutta diagrammin muodossa. Testitapaus, joka suoritetaan manuaalisesti, on suoralla viivalla. Testitapaus, joka automatisoidaan ja ajetaan ensimmäistä kertaa, on hieman vähemmän taloudellinen, sillä sen luomiseen on käytetty aikaa. Kaikista taloudellisin on automaatiotesti, joka on ajettu useamman kerran. (Fewster & Graham 1999, 4-5.)



Kuvio 5: Testin ”hyvyys” voidaan määrittää neljällä ominaisuudella. Mitä pidemmällä ominaisuus on, sitä suurempi on vinoneliön pinta-ala ja sitä parempi on testi

Automaatiotestaus tulee erottaa manuaalisesta testauksesta jo testien suunnittelussa. Manuaalinen testien suorittaminen ja automaatiotestaus ovat kaksi hyvin eri asiaa – automaatiossa testi tekee joka kerta sen, mitä testiin on määritelty, mutta testaaja suorittaa manuaalisen testitapauksen joka kerta hieman eri tavalla. Automaatiotestauksen suunnittelu onkin tärkeää erottaa muun testauksen suunnittelusta, jotta testauksesta kokonaisuudessaan saadaan mahdollisimman suuri hyöty. (Kaner ym. 2002, 130,107.)

5.6.2 Automaatiotestauksen kohteet

Automaatiota kannattaa erityisesti suosia sellaisissa testaustyypeissä, joita ei ole mahdollista toteuttaa manuaalisesti. Kaner ym. (2002, 95–96) listaavat muutaman esimerkin, joissa testauksen automatisointi on lähes välttämätöntä:

- Kuormitustestauksessa (load testing) järjestelmää kuormitetaan usealla käyttäjällä samaan aikaan. Usean tuhannen samanaikaisen käyttäjän simulointi vaatii automatisointia
- Kestävyystestausta (endurance testing) käytetään esimerkiksi muistivuotojen, pinon ylivuotojen, laittomien muistiosoitusten ja muiden vastaavien virheiden löytämiseen käyttämällä järjestelmää viikkoja tai jopa kuukausia yhtäjaksoisesti
- Yhdistelmävirheet (combination errors) esiintyvät silloin, kun järjestelmän eri ominaisuudet ovat kanssakäymisessä keskenään. Automaatiolla voi luoda erilaisia ja monimutkaisia tilanteita, joissa järjestelmän eri ominaisuudet ovat käytössä.

Aloitustestausta suositellaan suoritettavaksi automaatiolla, jolloin ne voi ajaa kuka tahansa ja milloin tahansa. Aloitustestit voidaan ajastaa suoriutumaan aina version vaihtuessa, jolloin kehittäjät voivat aina tarkistaa oman muutoksensa vaikutuksen muuhun ohjelmistoon. Jos aloitustestit menevät läpi, voidaan olettaa, että versio toimii ja muun testauksen voi aloittaa. Tämä nopeuttaa kehittäjän työtä ja parantaa koodin laatua. (Kaner ym. 2002, 94–95.)

Erityisesti mobiiliympäristössä automaatiosta on käytännön hyötyä, sillä eri laitteiden ja käyttöjärjestelmien kombinaatiota on valtavan suuri määrä. Mobiilituotteita on usein tarvetta testata niin monella eri kombinaatiolla kuin on mahdollista, mutta manuaalisesti tämä on hankalaa ja työlästä. Automaattiset testitapaukset, joita voi suorittaa millä tahansa eri kombinaatiolla, säästävät testaajan aikaa ja vaivaa. (Rand 2014.)

Kaikkea ei kuitenkaan kannata testata automaatiolla, sillä aina löytyy jotain testejä, jotka on helpompi testata manuaalisesti. Näitä voivat esimerkiksi olla testit, joita suoritetaan hyvin harvoin tai testit, joissa vaaditaan visuaalista silmää, kuten fontin värin tai sivun esteettisen

asettelun tarkastelu. Lisäksi on testejä, jotka vaativat fyysistä vuorovaikutusta, kuten esimerkiksi laitteen kytkeminen päälle tai pois päältä. (Fewster & Graham 1999, 22–23.)

Suurin haaste mille tahansa organisaatiolle, joka haluaa lisätä automaatiotestauksen määrää, on tietää, miten pitkälle automaatio voidaan ja kannattaa viedä. Mikäli ohjelmisto on vielä epävakaa tai sen ominaisuudet kehittyvät ja muuttuvat, automaation lisääminen ei välttämättä ole tehokkain tapa testata tuotteen laatua. Kun on selvillä, mitä ohjelmistolta vaaditaan ja mitkä osat ohjelmistoa voivat muuttua uusien julkaisujen myötä, voidaan automaatiota kehittää tehokkaasti ja järkevästi. Lisähaasteita automaation tehokkaaseen käyttöön tuo sen ylläpidon raskaus ja resurssien tarve. (Balasubramaniam 2014.)

5.7 Mobiilisovellusten testaus

Mobiiliapplikaatiot voidaan jakaa kolmeen ryhmään: natiiviapplikaatioihin, hybrideihin ja selainpohjaisiin applikaatioihin. Natiiviapplikaatiot ovat käyttöjärjestelmän tukemilla ohjelmointikielillä koodattuja ja niiden tekemiseen on yleensä käytetty käyttöjärjestelmän virallisesti tukemia kehitysympäristöjä. Natiiviapplikaatiot ovat yleensä suorituskykyisimpiä ja käyttäjäystävällisimpiä. Hybridit ovat sekoitus selainpohjaisia sovelluksia ja natiiviapplikaatioita – niiden kehitykseen on esimerkiksi käytetty erilaisia työkaluja. Hybridit eivät vaadi suorituskyvyltään yhtä paljon kuin natiiviapplikaatiot, mutta vaativat natiiviapplikaatioiden tavoin käyttäjältä käyttöoikeuksia esimerkiksi laitteen kameraan, kalenteriin tai sijaintitietoihin. Selainpohjaisten applikaatioiden kehittämisessä käytetään usein standardoituja ohjelmointikieliä ja yleensä ne toimivat useammalla eri käyttöjärjestelmällä. Selainpohjaisilla applikaatioilla ei ole välttämättä käyttöoikeuksia kaikkiin laitteen toimintoihin. (Stangarone 2016; Korf & Oksman 2016.)

Kuten edellä on todettu, mille tahansa ohjelmistolle on useita testaustapoja ja –menetelmiä, joten myös mobiilisovellusten testauksessa on omia menetelmiä, joista Palani (2014, 23) erottelee kahdeksan erilaista: toiminnallinen, suorituskyky-, käytettävyys-, turvallisuus- ja yhteensopivuustestaus, joita käytetään myös tietokoneohjelmiston testaamisessa. Lisäksi Palani (2014, 23) luettelee mobiilisovellukselle ominaisia testaustyyppejä: mobiilisovellusten keskeytystestauksen (mobile interrupt testing), yhteentoimivuustestauksen (mobile interoperability testing) ja lokalisointitestauksen (mobile localization testing).

Knott kertoo artikkelissaan (2014), että keskeytystestauksessa mobiilisovelluksen toiminta keskeytetään normaaleilla mobiililaitteiden toiminnoilla, kuten puheluilla, tekstiviesteillä tai ilmoituksilla. Knottin (2014) mukaan myös eri näppäimien painelut kesken sovelluksen toiminnan katsotaan keskeytystestaukseksi. Yhteentoimivuustestauksella tarkoitetaan Álvarezin (2012) mukaan erilaisten mobiiliverkkojen toimintaa: sovelluksen toimintaa tarkaillaan ympäristössä, missä on useita yhteyksiä päällä samaan aikaan, jotka voivat vaikuttaa sovelluksen

toimintaan. Lokalisointitestauksessa Kenistonin (1997) mukaan tarkastetaan, että sovellus vastaa kohdemaan kulttuurillisia sekä teknisiä vaatimuksia. Esimerkiksi kielelliset vaatimukset voivat olla tällaisia, mutta muitakin kulttuurillisia näkemyseroja voi olla, joiden vuoksi sovellus ei välttämättä menesty kaikilla alueilla. (Keniston 1997.)

Automaatiotestauksen maksimointi tuo mobiilisovellusten testaukseen tehokkuutta ja vähentää testauksen kustannuksia pitkällä aikavälillä. Mobiilisovellusten automaatiotestaus myös lisää testikattavuutta, mikä vähentää vikojen sekä kustannuksen määrää ja sitä kautta nopeuttaa uusien versioiden julkaisua ja lisää asiakastyytyväisyyttä. Automaatiotestausta kannattaa käyttää mobiilisovellusten regressiotesteissä aina uuden version julkaisun jälkeen ja erityisesti silloin, kun käyttöjärjestelmän versio päivittyy. Tällöin havaitaan viat, jotka ovat syntyneet käyttöjärjestelmässä tapahtuneiden muutosten seurauksena. Mobiilisovellusten maailmassa kaikkea ei kuitenkaan kannata eikä edes pysty testaamaan automaatiolla. Esimerkiksi yksityiskohtien ja käyttökokemuksen testaus tai edellä mainitun keskeytystestauksen automatisointi on lähes mahdotonta, jolloin testauksessa suositellaan käytettäväksi oikeaa mobiililaitetta. Näistä edellä mainituista syistä johtuen testaus kannattaa suorittaa ensisijaisesti automaatiolla ja jättää manuaalinen testaus itse laitteella suoritettuna viimeiseksi työvaiheeksi testauksessa. (Narasimha 2013; Ryan 2016; Myers 2012, 222–223.)

6 Tuotekehitys Zalandolla

Tässä kappaleessa esitellään Zalandon tuotekehityksen yleinen lähestymistapa tuotannon prosesseihin. Lisäksi tässä kappaleessa esitellään Fleek-sovelluksen iOS-version kehittäjien omat prosessit ja periaatteet testauksen näkökulmasta. Lopuksi esitellään testauksessa käytettävä työkalu Xcode.

6.1 Tuotekehityksen menetelmät

Termi Radical Agility tarkoittaa Zalandon lähestymistapaa tuotekehityksen johtamiseen. Yksinkertaistettuna Radical Agility tarkoittaa hyvin ketterää tekemistä, jossa ryhmät itse vastaavat tuotteen suunnittelusta, kehityksestä, arkkitehtuurista sekä testauksesta. Menetelmä kehitettiin vastaamaan niihin haasteisiin, joita Zalando kohtasi nopeasti kasvavassa tuotekehityksessään. Radical Agility -menetelmän idea perustuu kolmeen pilariin, joista käytetään termejä Purpose, Autonomy ja Mastery. (Apple 2015; Bowman 2016.)

Termin Autonomy tarkka suomennos on itsehallinto. Zalandolla se tarkoittaa luottamusta ja itsenäistä päätöksentekoa. Kehitystiimit voivat keksiä ja kokeilla omia tapoja, tekniikoita sekä viitekehyksiä tarpeen mukaan. Termillä Mastery tarkoitetaan itsensä kehittämistä ja valmentamista ammatillisesti, joka motivoi ja inspiroi työntekijöitä työskentelemään yrityksen sekä oman uransa eteen. Termillä Purpose tarkoitetaan työn tarkoitusta ja päämäärää. Tarkoitus toimii kehyksenä päätöksille ja helpottaa keskittymistä olennaiseen mahdollisten ongelmien edessä. (Bowman 2016.)

6.2 Fleek-sovelluksen testaus

Ketterien menetelmien mukaisesti ohjelmiston kehittäjät toimivat myös testaajina ja hyväksymistestauksen suorittavat tuotteen käyttäjät antamalla palautetta tuotteen laadusta (Myers ym. 2012, 178–179). Fleek-sovelluksen iOS-version kehityksessä kehittäjät koodaavat kirjoittamansa toiminnon tai korjaamansa vian omaan haaraan versionhallinnassa, testaavat sen itse, jonka jälkeen vielä toinen kehittäjä testaa koodin toimivuuden. Tämän lisäksi koodi katselmoidaan ennen kuin ominaisuus tai korjaus lisätään päähaaraan. Näillä toimenpiteillä pyritään varmistamaan, että päähaarassa olevassa koodissa on vikoja mahdollisimman vähän ja tuote on jatkuvasti valmis julkaistavaksi. Erityisen tärkeää tämä on Apple Storen omien prosessien vuoksi, joiden takia uusien versioiden julkaisu saattaa kestää jopa päiviä. (Forsten 2016.)

Kehittäjien tekemien testien lisäksi ennen sovelluksen siirtämistä Apple Storen julkaisujonoon Fleek testataan vielä manuaalisesti. Tämä testausvaihe sisältää sekä tutkivaa testausta että testitapauksien suorittamista. Testattavia kohteita on muun muassa sisäänkirjautuminen ja sen eri variaatiot. (Lappalainen 2016.)

Fleek-sovelluksen iOS-version automaatiotestien luomisessa ja suorittamisessa käytetään käyttöjärjestelmän virallisesti tukemaa kehitysympäristöä Xcodea. Xcode on Applen kehittämä ohjelmointiympäristö Apple-tuotteiden käyttöjärjestelmille. Xcoden käyttö Fleek-sovelluksen testaamisessa on luontevaa ja helppoa, sillä Fleek-sovellusta kehitetään samalla työkalulla. (Apple 2016a; Forsten 2016.)

Xcode on monipuolinen testauksen työkalu. Uusien projektien aloittaminen ja konfigurointi on tehty automaattiseksi, jolloin testien luominen ja suorittaminen on nopeampaa. Xcodella voi lisäksi suorittaa useita eri testauksen menetelmiä: työkalulla voi esimerkiksi seurata ja testata sovelluksen suorituskykyä sekä luoda käyttöliittymän testejä nauhoittamalla käyttöliittymän tapahtumia ja toimintoja. (Apple 2016b.)

7 Tulokset

Tässä kappaleessa tarkastellaan tämän opinnäytetyön tutkimuksen tuloksia. Ne jaetaan kahteen osaan: testaussuunnitelmaan ja testausprosessin analysointiin.

7.1 Testaussuunnitelma

Kuten aikaisemmin tässä opinnäytetyössä on todettu, ohjelmistotestauksen yksi tarkoitus on tarjota tietoa tuotteen laadusta sidosryhmille, jotka voivat tämän tiedon perusteella tehdä liiketoiminnallisia päätöksiä. Tehtävänanto testaussuunnitelmalle oli selvä: Fleek-sovellus sisältää suuren määrän erilaisia toimintoja sekä toiminnallisuuksia, mutta suunnitelman pääpaino tulee olla sovelluksen bisneskriittisimmissä osissa, kuten sisäänkirjautumisissa sekä ostotapahtuman siirtymissä ja onnistuneissa ostotapahtumissa.

Vaikka kehittäjien toiveena oli automaation lisääminen, ottaa suunnitelma kuitenkin huomioon myös manuaalisen testauksen tarpeen. Suunnitelman tavoitteena on keskittyä keinoihin, joilla Fleek-sovelluksen laatua parannetaan. Tämän lähestymistavan vuoksi suunnitelmassa mukana on myös testejä, joita on vaikea toistaa automaatiolla. Kiteyttäen, opinnäytetyö keskittyy kokonaisvaltaisesti laadunvarmistukseen – eikä pelkästään luomaan automaatiota vailla kunnon tarkoitusta.

Mallipohjainen testaus aloitettiin mallin luomisella. Tämän pohjana käytettiin Zalandon vuokaaviota, joka teknisten yksityiskohtiensa vuoksi oli yksistään kehno malli tähän tarkoitukseen. Sen vuoksi teknisestä mallista on tehty tätä opinnäytetyötä varten yksinkertainen versio Fleek–sovelluksen ostotapahtumasta ja sen eri siirtymistä.

Testit ovat luotu testaussuunnitelmaan mustalaatikkotekniikalla ohjelman koodia tutkimatta. Testien luomisessa on siis käytetty toimintojen ja siirtymien odotettuja tuloksia. Suunnitelma sisältää kuusi pääteemaa: sisäänkirjautumisen, rekisteröitymisen, osoitteen lisäämisen ja/tai vaihtamisen, maksutavan lisäämisen ja/tai vaihtamisen, ostoskorin sisällön sekä itse ostotapahtuman. Testaussuunnitelma sisältää 32 testiä ja se löytyy liitteestä 1.

7.2 Testausprosessi

Laadukkaan ohjelmiston saavuttamiseksi tarvitaan testausta. Aina pelkkä testaus ei kuitenkaan riitä takaamaan laatua, vaan testauksen prosesseihin tulee myös kiinnittää huomiota. Jotta testaus toisi lisäarvoa ja laatua tuotteelle, otettiin tähän opinnäytetyöhön mukaan myös itse testauksen prosessin tarkastelu.

Testauksen prosessia ja yleistä laadunvarmistusta tarkasteltiin haastatteluin. Haastatteluihin osallistui viisi Fleek-sovelluksen iOS-kehittäjää. Vastaajat valikoituivat vapaaehtoisuuden sekä

eri työtehtävien perusteella. Haastattelukysymykset löytyvät liitteestä 3. Haastattelun avoimen luonteen vuoksi tarkentavia kysymyksiä oli usein tarvetta esittää. Haastattelujen kestossa oli vaihtelua, kymmenestä minuutista lähtien aina puoleen tuntiin asti.

Laatu ja laadukas ohjelmisto merkitsee kehittäjille useita asioita. Parhaiten laadun voisi kiteyttää seuraavasti: koodin toimivuus yhdessä hyvän käyttöliittymäsuunnittelun kanssa takaa käyttäjille sujuvan käyttökokemuksen, jolloin bisnes toimii. Statistiikan ja tilastojen mukaan Fleek-sovelluksen osalta näin onkin, mutta kehittäjien mielestä tuotteen laadussa on silti parantamisen varaa. Tämä oli merkittävä huomio, sillä kaikilla vastaajilla oli testausprosessi sekä oma kaksoisrooli kehittäjänä ja testaajana tiedossa. He myös kokivat, että voivat itse vaikuttaa tuotteen laatuun ainakin jossain määrin esimerkiksi kirjoittamalla ja suorittamalla testejä sekä auttamalla muita testaamaan omaa koodiaan.

Lähes kaikki vastaajat nimesivät automaatiotestauksen nykytilan yhdeksi haasteeksi testausprosessissa. Vastaajat kokivat, että automaatiotestejä ei ole tarpeeksi, niitä on työlästä ylläpitää, eivätkä kaikki testit testaa oikeita asioita – viimeisestä esimerkkinä yksikkötestit, jotka hakevat tietoja palvelimelta asti, jolloin testit epäonnistuvat, mikäli palvelimella on häiriöitä. Haastatteluissa nostettiin esiin myös toinen ongelma: kokonaisuuden testaaminen. Vaikka periaatteessa jokainen versio on aina julkaisukelpoinen, kehittäjillä ei silti ollut varmuutta siitä, rikkooko oma koodi päähaarassa jotain.

Vaikka testaussuunnitelma helpottaa sovelluskokonaisuuden testauksen nykytilaa, suunnitelma yksistään ei riitä – testaukseen tulee panostaa koko sovelluksen elinkaaren ajan kirjoittamalla uusia ja ylläpitämällä vanhoja testejä. Testaukseen panostaminen saattaa olla haasteellista kiireen sekä liiketoiminnan vaatimusten vuoksi, mutta se kannattaa asiakastyytyväisyyden ja liiketoiminnan ylläpitämiseksi. Automaatiotestauksen kehittämisessä kannattaa kuitenkin muistaa, että määrä ei korvaa laatua. Suuri määrä automaatiotestejä tarkoittaa raskaampaa ylläpitoa, mikä muun muassa lisää kustannuksia. Siksi testitapauksia tulee säännöllisesti katselmoida duplikaattien tai turhien testien poistamiseksi. Toisaalta automaatiotesteistä ei kuitenkaan kannata tehdä liian monimutkaisia, sillä pitkistä ja monimutkaisista testeistä on testin epäonnistuttua hankalaa paikantaa, missä kohtaa testi epäonnistui.

Myös itse testausprosessiin tulee kiinnittää huomiota. Esimerkiksi tutkivassa testauksessa löytynyt vika voi tarkoittaa, että automaatiotestejä tulee tarkentaa ja parantaa. Mikäli taas aloitustestauksessa löytyy vika, voi yksikkötestejä joutua tarkastelemaan.

8 Yhteenveto

Opinnäytetyön työstäminen alkoi alkuvuodesta 2016 tutustumalla Zalandon omiin prosesseihin ja mobiilisovellukseen sekä opinnäytetyön aiheen rajaamisella. Oleellista oli kehittää testausprosessia konkreettisesti ja kehittää testaussuunnitelma sovelluksen bisneskriittisille osille.

Opinnäytetyön toiminnallinen osuus alkoi sovelluksen tekniseen vuokaavioon sekä itse Fleek-sovellukseen tutustumisella. Vuokaaviosta luotiin malli, joka oli yksinkertainen versio teknisestä vuokaaviosta. Mallin työstäminen alkoi mallipohjaiseen testaukseen tutustumisella. Mallista pystyi suoraan luomaan testitapaukset eri toimintojen pohjalta. Testitapausten luomisessa otettiin huomioon vain sovelluksen odotetut tapahtumat eikä erilaisiin virhetilanteisiin tai –viesteihin otettu kantaa. Mallipohjaisessa testauksessa testitapausten luominen on myös mahdollista erilaisten työkalujen avulla, mutta niitä ei käytetty - uuden työkalun opettelemiseen kuluva aika olisi pois itse työstä, eikä sen käyttö olisi tuonut työhön merkittävää lisäarvoa.

Koska laadunvarmistus on muutakin kuin pelkästään testaussuunnitelma, otettiin opinnäytetyössä myös kantaa itse laadunvarmistuksen prosessiin sekä siihen liittyviin haasteisiin ja hyviin puoliin. Lisäksi haastatteluissa haluttiin selvittää millaiseksi vastaajat itse kokevat laadukkaan tuotteen ja mitä he itse pystyvät laadun eteen tekemään. Tämä koettiin varsin tärkeäksi, sillä Zalandolla laadunvarmistusta ei ole ulkoistettu erillisille testausasiantuntijoille, vaan kehittäjät ovat itse vastuussa sovelluksen laadusta. Työhön liittyviin kulttuurisiin tai sovittuihin tapoihin ei otettu kantaa, vaan testauksen tilaan keskityttiin kokonaisuudessaan.

Tulevaisuudessa testaussuunnitelmaa voidaan käyttää aloitustestauksessa ja regressiotestauksessa uusien versioiden julkaisun jälkeen. Koska suunnitelma keskittyy käyttöliittymässä tapahtuviin toimintoihin, on testejä syytä päivittää ja lisätä, kun uusia toimintoja syntyy. Testaussuunnitelmassa on mukana myös jokaisessa testitapauksessa sen priorisointi, mikä merkitsee sen suorittamisen ja päivittämisen tärkeyttä. Priorisointiluokka 1 tarkoittaa, että vähintään nämä testit tulee suorittaa jokaisen version yhteydessä. Tämä helpottaa testauksen keskittämistä sovelluksen tärkeimpiin osiin.

Vaikka Fleek-sovellus on tilastojen mukaan asiakkaiden mieleen, oli silti haastateltavien mukaan laadunvarmistuksessa kehitettävää. Erityisesti hyväksymistestien luomisessa ja ylläpitämisessä havaittiin olevan haasteita. Kuitenkin sovelluksen kehitysprosessissa havaittiin hyvänä asiana runko ja rutiinit, jolloin tietyt toimenpiteet tulevat varmasti tehtyä. Haasteena onkin luoda testauksen kehittämisestä rutiini sovelluksen kehittäjille.

Kaiken kaikkiaan opinnäytetyön kirjoittaminen ja testaussuunnitelman luominen oli mielenkiintoinen projekti, jossa sai käyttää omaa ammattitaitoaan hyväksi. Lisäksi projektissa pääsi tutustumaan erilaiseen työkulttuuriin ja –menetelmiin ketterän ohjelmistokehityksen maailmassa. Opinnäytetyö tulee auttamaan Zalandoa Fleek-sovelluksen laadunvarmistuksessa, kunhan testauksen ylläpidolle ja päivittämiselle kyetään antamaan tarpeeksi resursseja.

Lähteet

Painetut

Álvarez, R. F. 2012. Testing on a Real Handset vs. Testing on a Simulator – the big battle. Testing Experience. 19/2012, 42–44.

Balasubramaniam, V. 2014. Automation - The path to testing efficency. The European Software Tester. 4/2014, 26–29.

Fewster, M. & Graham, D. 1999. Software Test Automation – Effective use of test execution tools. Iso-Britannia: Addison-Wesley.

Hass, A. M. J. 2008. Guide to advanced software testing. Boston: Artech House.

Hirsjärvi, S., Remes, P. & Sajavaara, P. 2009. Tutki ja kirjoita. 15. painos. Helsinki: Tammi.

ISTQB. 2015. Certified Model-Based Tester: Foundation Level Syllabus. International Software Testing Qualifications Board.

Kaner, C., Bach, J. & Pettichord, B. 2002. Lessons learned in software testing – A context-driven approach. New York: Wiley.

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

Keniston, K. 1997. Software Localization: Notes on Technology and Culture. Massachusetts: Massachusetts Institute of Technology

Knott, D. 2014. How to Stress Test Your Mobile App. Testing Experience. 27/2014, 4-5.

Mathur, A. 2008. Foundations of Software Testing. India: Dorling Kindersley.

Moens, M.-F., Li J., & Chua T.-S. 2014. Mining user generated content. Boca Raton: CRC Press.

Myers, G., Badgett, T. & Sandler, C. 2012. The Art of Software testing. 3. painos. New Jersey: John Wiley & Sons.

Narasimha, R. K. B. 2013. Key Principles in Selecting the Right Automation Tools for Mobile Application Testing. Testing Experience. 23/2013, 18-22.

Ojasalo, K., Moilanen, T. & Ritalahti, J. 2010. Kehittämistyön menetelmät. Uudenlaista osaamista liiketoimintaan. Helsinki: WSOY.

Palani, N. 2014. Mobile Software Testing. Mumbai, India: Wordit Content Design & Editing Services Pvt.

Spillner, A., Linz, T. & Schaefer, H. 2014. Software Testing Foundations. 4. painos. Santa Barbara: Rocky Nook.

Rand, S. 2014. Is automation testing the future or fad. The European Software Tester. 3/2014, 19–21.

Sähköiset

Apple, L. 2015. So, you have heard about ”Radical Agility”…. Viitattu 25.10.2016. https://tech.zalando.com/blog/so-youve-heard-about-radical-agility...-video/

Apple. 2016a. Xcode 8. Viitattu 12.11.2016. https://developer.apple.com/xcode/

Apple. 2016b. About Testing with Xcode. Viitattu 12.11.2016. https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/01-introduction.html#//apple_ref/doc/uid/TP40014132-CH1-SW1

Bowman, E. 2016. Radical Agility 101: Study Notes. Viitattu 25.10.2016. https://tech.zalando.com/blog/radical-agility-study-notes/

Daake, M. 2015. How Zalando's App Makes Instagram Images Shoppable. Viitattu 24.9.2016. https://tech.zalando.com/blog/how-zalandos-app-makes-instagram-images-shoppable/

Ecommerce News. 2016. The biggest online stores in Germany 2015. Viitattu 25.9.2016. http://ecommercenews.eu/biggest-online-stores-germany-2015/

Fleek. 2016. FAQ. Viitattu 22.9.2016. https://www.fleek.de/en/faq.html

Halme, E. 2004. Tutkiva testaus hyväksymistestauksen menetelmänä. Viitattu 13.10.2016. http://www.soberit.hut.fi/T-76.5650/Spring_2004/Papers/E.Halme_76650_final.pdf

iTunes. 2016. Fleek - Social Fashion Shopping. Viitattu 13.10.2016. https://itunes.apple.com/de/app/fleek-social-fashion-shopping/id1065471885?mt=8

Korf, M. & Oksman, E. 2016. Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options. Viitattu 10.11.2016. https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

Lappalainen, E. 2015. Zalando avaa kesällä toimiston Helsinkiin: "Jos kaikki menee hyvin, muutamassa vuodessa 150–200 työntekijää". Viitattu 19.9.2016. http://www.talouselama.fi/uutiset/zalando-avaa-kesalla-toimiston-helsinkiin-jos-kaikki-menee-hyvin-muutamassa-vuodessa-150-200-tyontekijaa-3476960

Lee, D. 2016. Instagram users top 500 million. Viitattu 24.9.2016. http://www.bbc.com/news/technology-36584511

Retail Research. 2016. Mobile retailing 2015. Viitattu 26.9.2016. http://www.retailresearch.org/mobileretailing.php

Ryan, M. 2016. When to Use Automated Mobile App Testing vs. Manual App Testing. Viitattu 11.11.2016. http://mobilelabsinc.com/when-to-use-automated-mobile-app-testing-vs-manual-app-testing/

Stangarone, J. 2016. The Mobile App Comparison Chart: Hybrid vs. Native vs. Mobile Web. Viitattu 10.11.2016. http://www.mrc-productivity.com/blog/2016/06/the-mobile-app-comparison-chart-hybrid-vs-native-vs-mobile-web/

Zalando. 2015. EUROPE'S LEADING ONLINE FASHION DESTINATION. Viitattu 13.10.2016. https://corporate.zalando.com/sites/default/files/mediapool/02_cmd_tech_zalando_0.pdf
Zalando. 2016a. From a start-up to a grown up. Viitattu 19.9.2016. https://jobs.zalando.de/en/about/

Zalando. 2016b. Why technology? Viitattu 19.9.2016. https://corporate.zalando.com/en/why-technology-p#fc-87

Julkaisemattomat

Fleek. 2016. Kuvakaappaus. Otettu 22.9.2016.

Forsten, J. 2016. Haastattelu 27.9.2016. Zalando. Helsinki.

Lappalainen, P. 2016. Haastattelu 27.9.2016. Zalando. Helsinki.

Kuviot

Liitteet

Liite 1: Testaussuunnitelma

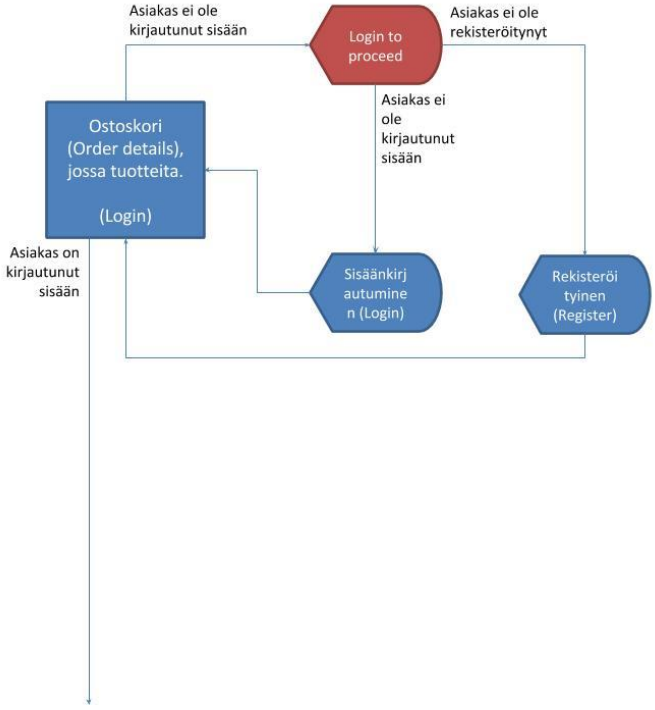
Yhteenveto: Tämä testaussuunnitelma on tarkoitettu Zalandon Fleek-mobiilisovelluksen uusille julkaisuille. Testit voidaan suorittaa manuaalisesti tai ne voidaan automatisoida. Testit, joiden otsikossa on merkintä P1, suoritetaan jokaisen uuden version jälkeen. Testit, joiden otsikossa on merkki A1, suositellaan automatisoivaksi ensimmäisenä. Esiehto: Jokaisen testin alussa ostoskorissa on tuotteita.	
Testauksen aloitusehdot: – Testiympäristö on saatavilla ja valmis – Testattava versio on ladattu testiympäristöön	
Testauksen lopetusehdot: – Vähintään P1 testit on suoritettu – Testauksessa on löytynyt testauksen lopettamisen kannalta kriittinen vika, joka on paikannettu.	
Testitapaukset	
1.	Otsikko: **Sisäänkirjautuminen – Onnistunut** *P1/A1*
	Odotetut tulokset: Sisäänkirjautuminen onnistuu omilla tunnuksillaan. Sisäänkirjautumisen jälkeen tuotteet näkyvät edelleen ostoskorissa.
2.	**Sisäänkirjautuminen – Onnistunut – Tuotteita jo ostoskorissa** *P1/A1*
	Sisäänkirjautuminen onnistuu omilla tunnuksillaan. Sisäänkirjautumisen jälkeen tuotteet näkyvät edelleen ostoskorissa ja asiakkaan aikaisemmin lisäämät tuotteet on lisätty ostoskoriin
3.	**Sisäänkirjautuminen – Virheelliset syötteet** *P3/A3*
	Asiakas ei pääse kirjautumaan sisään virheellisten syötteiden vuoksi (min 2 merkkiä)
4.	**Sisäänkirjautuminen – Ei syötteitä** *P2/A2*
	Sisäänkirjautuminen ei onnistu mikäli yksikin kenttä on tyhjä
5.	**Sisäänkirjautuminen – Brute force** *P1/A1*
	Sisäänkirjautuminen ei onnistu mikäli salasana on väärin
6.	**Sisäänkirjautuminen – Zalandon tunnuksilla** *P1/A2*
	Sisäänkirjautuminen onnistuu. Sisäänkirjautumisen jälkeen tuotteet näkyvät edelleen ostoskorissa.
7.	**Sisäänkirjautuminen – Facebookin tunnuksilla** *P1/A2*
	Sisäänkirjautuminen onnistuu. Sisäänkirjautumisen jälkeen tuotteet näkyvät edelleen ostoskorissa.
8.	**Sisäänkirjautuminen – Osto epäonnistunut** *P1/A2*
	Osto ei onnistu, sillä asiakas ei ole kirjautunut sisään

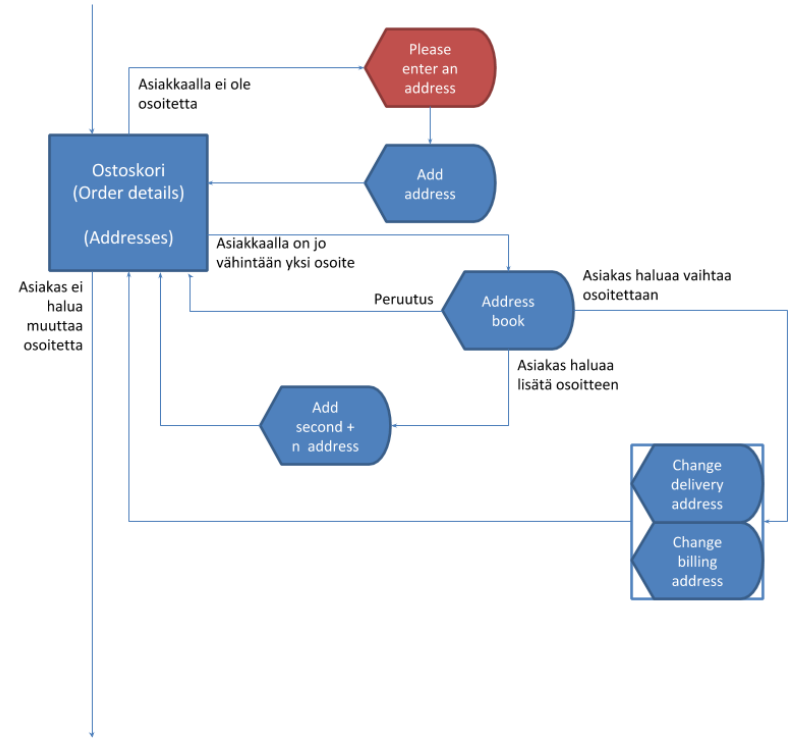
9.	**Rekisteröityminen – Onnistunut** *P1/A1*
	Asiakas pääsee rekisteröitymään onnistuneesti omilla tiedoillaan. . Rekisteröitymisen jälkeen tuotteet näkyvät edelleen ostoskorissa.
10.	**Rekisteröityminen – Virheelliset syötteet** *P3/A3*
	Asiakas ei pääse rekisteröitymään sisään virheellisten syötteiden vuoksi (min 2 merkkiä)
11.	**Rekisteröityminen – Ei syötteitä** *P2/A2*
	Rekisteröityminen ei onnistu mikäli yksikin kenttä on tyhjä
12.	**Rekisteröityminen – Sähköposti jo käytössä** *P1/A1*
	Rekisteröityminen ei onnistu, mikäli sähköpostiosoitteella on jo rekisteröidytty
13.	**Rekisteröityminen – Sukupuoli** *P2/A1*
	Rekisteröityminen ei onnistu, mikäli sukupuolta ei ole valittu
14.	**Osoite – Ei osoitetta – Uuden osoitteen lisääminen – Onnistunut** *P1/A1*
	Asiakkaan osoitteen antaminen ja tallentaminen onnistuu
15.	**Osoite – Ei osoitetta – Uuden osoitteen lisääminen – Ei syötteitä** *P3/A3*
	Osoitteen antaminen ei onnistu mikäli yksikin kenttä on tyhjä (pl. Additional information)
16.	**Osoite – Useampi osoite – Uuden osoitteen lisääminen – Onnistunut** *P2/A2*
	Asiakkaan osoitteen antaminen ja tallentaminen onnistuu
17.	**Osoite – Useampi osoite – Uuden osoitteen lisääminen – Ei syötteitä** *P3/A3*
	Osoitteen antaminen ei onnistu mikäli yksikin kenttä on tyhjä (pl. Additional information)
18.	**Osoite – Useampi osoite – Toimitusosoitteen muutos** *P2/A2*
	Toimitusosoitteen muuttaminen onnistuu
19.	**Osoite – Peruutus** *P3/A3*
	Osoitteen antaminen peruutetaan
20.	**Osoite – Osto epäonnistunut** *P1/A2*
	Osto ei onnistu, sillä asiakas ei ole antanut osoitettaan
21.	**Maksutapa – Uusi maksutapa – Maksutavan valinta – Lasku** *P2/A3*
	Laskun valinta onnistuu
22.	**Maksutapa – Uusi maksutapa – Maksutavan valinta – PayPal** *P2/A3*
	Paypalin valinta onnistuu
23.	**Maksutapa – Uusi maksutapa – Maksutavan valinta – Luottokortti** *P2/A3*

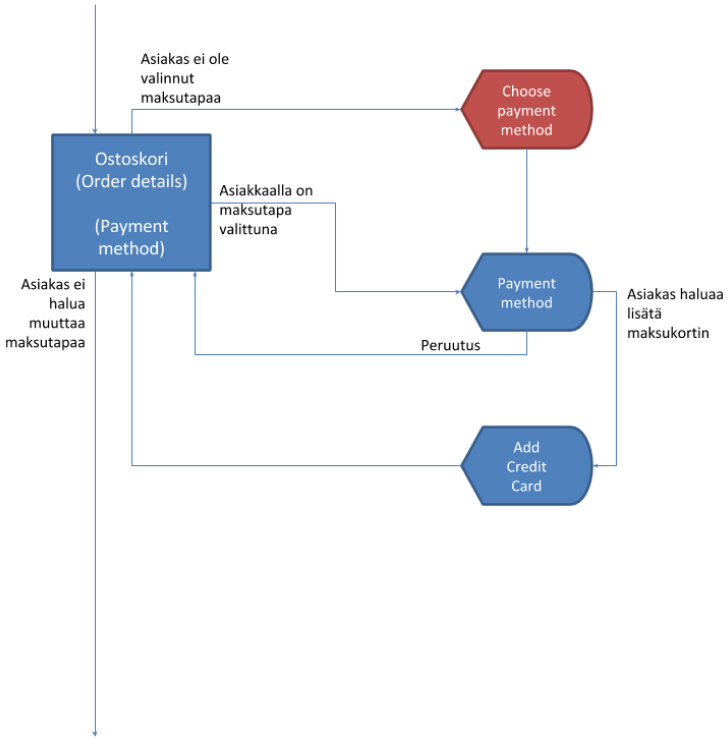
	Luottokortin valinta onnistuu
24.	**Maksutapa – Maksutavan valinta – Maksutavan vaihto – Onnistunut** *P3/A3*
	Maksutavan vaihto onnistuu
25.	**Maksutapa – Luottokortin lisääminen – Onnistunut** *P2/A2*
	Luottokortin lisääminen onnistuu
26.	**Maksutapa – Luottokortin lisääminen – Virheelliset syötteet** *P3/A3*
	Asiakas ei pääse lisäämään maksukorttia virheellisten syötteiden vuoksi
27.	**Maksutapa – Luottokortin lisääminen – Ei syötteitä** *P3/A3*
	Asiakas ei pääse lisäämään maksukorttia, mikäli yksikin kenttä on tyhjä
28.	**Maksutapa – Osto epäonnistunut** *P1/A2*
	Osto ei onnistu, sillä asiakas ei ole valinnut maksutapaa
29.	**Ostoskori – Tuotteen poistaminen/siirtäminen toivelistalle** *P1/P1*
	Tuotteen poistaminen/siirtäminen toivelistalle onnistuu. Tuote katoaa ostoskorista
30.	**Ostoskori – Tuotteen koon muuttaminen** *P3/A3*
	Tuotteen koon muuttaminen onnistuu ja ostoskori päivittyy
31.	**Ostoskori – Tuote on loppuunmyyty** *P3/A3*
	Osto ei onnistu
32.	**Osto – Onnistunut** *P1/P1*
	Kaikki tiedot on annettu

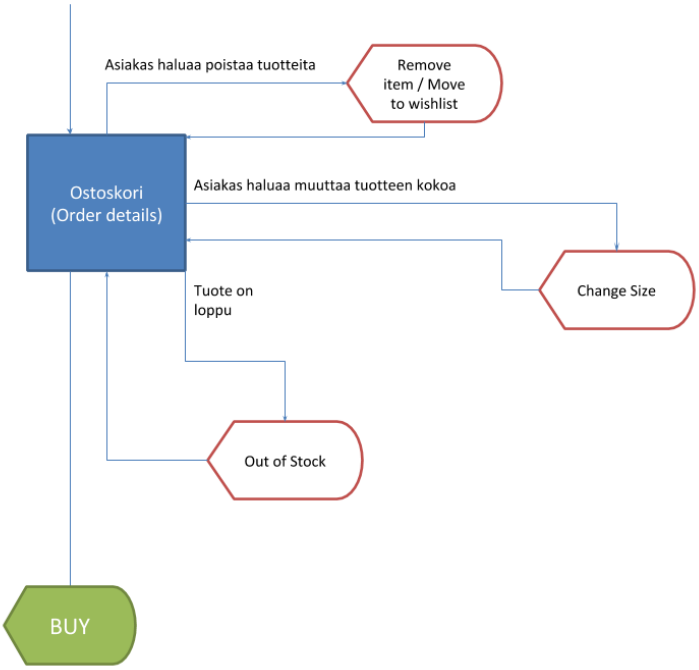
Liite 2: Malli mobiilisovelluksen ostotapahtuman toiminnoista

Malli on yksinkertaistettu versio teknisestä versiosta mobiilisovelluksen kassan toiminnasta. Kuviossa sinisellä pohjalla oleva Ostoskori-laatikko tarkoittaa ostoskorin pääsivua, josta löytyy erilaisia toimintoja. Muut siniset laatikot ovat sivuja, joissa on mahdollista muuttaa tai lisätä syötteitä. Punaisella pohjalla olevat laatikot tarkoittavat joko virheellistä tilannetta tai puuttuvia pakollisia syötteitä. Vihreä laatikko osoittaa tapahtuman onnistunutta päättämistä ja ostoskortista poistumista.




Please enter an address
Asiakkaalla ei ole osoitetta
Ostoskori (Order details)
(Addresses)
Add address
Asiakkaalla on jo vähintään yksi osoite
Asiakas ei halua muuttaa osoitetta
Peruutus
Address book
Asiakas haluaa vaihtaa osoitettaan
Asiakas haluaa lisätä osoitteen
Add second + n address
Change delivery address
Change billing address


Asiakas ei ole valinnut maksutapaa
Choose payment method
Ostoskori (Order details)
(Payment method)
Asiakkaalla on maksutapa valittuna
Asiakas ei halua muuttaa maksutapaa
Payment method
Asiakas haluaa lisätä maksukortin
Peruutus
Add Credit Card


Asiakas haluaa poistaa tuotteita
Remove item / Move to wishlist
Ostoskori (Order details)
Asiakas haluaa muuttaa tuotteen kokoa
Change Size
Tuote on loppu
Out of Stock
BUY

Liite 3: Haastattelukysymykset

Miten kuvailisit Fleek-sovelluksen kehitysprosessia?

Millaisia haasteita prosessissa on?

Mitä hyötyjä prosessissa on?

Kuka testaa Fleek-sovellusta ja miten?

Millaisia haasteista tällaisessa prosessissa on?

Millaisia hyötyjä tällaisessa prosessissa on?

Mitä laatu mielestäsi tarkoittaa?

Minkä numeron antaisit Fleek-sovelluksen laadulle, jos 1 on erittäin huonolaatuinen ja 5 erittänen hyvälaatuinen?

Voitko vaikuttaa sovelluksen laatuun ja miten?