Joni Erkkilä

Automaatiotestien määrittely ja arviointi verkkokauppakehityksessä

Opinnäytetyö

Tradenomi (YAMK), Tietojenkäsittely ja liiketoimintaosaaminen

Kevät 2021

Tiivistelmä

Tekijä: Erkkilä Joni

Työn nimi: Automaatiotestien määrittely ja arviointi verkkokauppakehityksessä

Tutkintonimike: Tradenomi (YAMK), tietojenkäsittely ja liiketoimintaosaaminen

Asiasanat: Automaatiotestaus, verkkokauppa, testaus

Opinnäytetyön tarkoitus on tutkia automaatiotestausta ja sen käyttöönoton mahdollisuutta opinnäytetyön toimeksiantajan työympäristössä. Toimeksi antajana toimii LianaTechonologies yritys, joka toimittaa viestintä ratkaisuja omille asiakkailleen.

Toimeksiantajalla on useita omia tuotteita, joilla voidaan tarjota viestintäratkaisuja. Yksi näistä tuotteista on verkkokauppa-alusta. Automaation kautta toteutettava testaus selkeyttäisi kehitysprosesseja ja parantaisi työn laatua. Opinnäytetyön tekijä työskentelee toimeksiantajan verkkokauppa osastolla kehittäjänä.

Teoriaviitekehys on muodostettu ohjelmiston kehittämisen, ohjelmiston ylläpidon, sekä projektitoiminnan ympärille. Ohjelmiston kehittämisen ja ylläpidon käsitteitä lähestytään pääosin testaamisen näkökulmasta. Projektitoiminnan kautta esitellään ketterän kehityksen menetelmät ja nämä menetelmät linkitetään testaamiseen.

Opinnäytetyön tutkimusosa on tyypiltään kvalitatiivinen. Tutkimuksen strategiana toimii tapaustutkimus. Strategiaa toteutetaan havainnointi menetelmien muodossa. Teoriakatsauksen aikana etsittiin tarvittavaa tietoa, jonka perusteella pystyttiin luomaan lopullinen tuotos.

Opinnäytetyön tuloksena syntyi tekniset määritelmät ja aika-arvio toteutukselle. Näiden kahden ominaisuuden perusteella projektia voidaan lähteä viemään eteenpäin. Samalla syntyi tekniset rajaukset, mitä tulee ottaa huomioon, kun työtä aletaan viemään eteenpäin. Tuloksen perusteella organisaation sisällä voidaan pohtia projektin jatkoa. Onko esitetty tulos ja sen rajaukset, jotain mistä organisaatio hyötyy lopullisesti.

Abstract

Author: Erkkilä Joni

Title of the Publication: Review of automated testing enviroment in ecommerce development

Degree Title: Master of Business Administration

Keywords: Automated testing, e-commerce, testing

The purpose on this thesis is to explore term automated testing and how this term can be made into reality within clients working environment. The client for this thesis is communications company Liana Technologies.

The client has multiple products of its own which it uses to provide communications solutions to its own customers. One of these products is e-commerce content management system. Testing done with automation would make development processes clearer and improve the quality of work. Author of this thesis work for the client as e-commerce developer.

Theoretical framework for thesis consists of software development, software maintenance also project management. Point of view for software development and maintenance is testing. Agile development methods are introduced with project management and term is linked into testing.

Qualative research method is used for during thesis and strategy for research is case study. Methods for the actual research consist of observation. Theoretical framework is used in research stage to develop the outcome for the research.

Outcome is tehnical specifations and estimation of required work time for actual work which is processed after the thesis is done. By the time of writing of thesis it is unclear whether this outcome is what would actually benefit the organisation.

Sisällys

Symboliluettelo

Kanban	Projektinhallinta työkalu, työtehtävä etenee taulun vasemmasta laidasta taulun oikeaan laitaan.
Scrum	Ketterän kehityksen työkalu. Voidaan linkittää Kanban tauluun.
Sprint	Scum jakautuu lyhyihin ajanjaksoihin. Yleensä ajanjakson pituus on kaksi viikkoa, mutta pituus sovitaan organisaatiossa tai kehitystiimissä.
Verkkokauppa	Sähköinen kaupankäynti alusta. Koostuu teknologia kerroksista ja palvelukanavista.

1 Johdanto

Opinnäytetyössä perehdytään verkkokehityksen maailmaan. Erityisesti pureudutaan automaatiotestaamiseen, verkkokauppa kehitykseen ja verkkokaupan ulkoasujen maailmaan. Tarkoitus on ideoida, kuinka olemassa olevaan ympäristöön olisi mahdollista liittää automaatiotestauksen mahdollisuus.

Tavoite tällä opinnäytetyöllä on luoda käyttäjätarina kohdeorganisaation projektinhallintaan. Tämä tarina sisältää määritykset, mitä kehittäjän tulee tehdä, että hän voi suorittaa työnsä. Tarinaan on myös liitetty aika-arvio, kuinka kauan suunnilleen työn toteuttamiseen menee.

Testien automatisointi on pyörinyt pitkään organisaatiossa idean tasolla. Puuttuvan aika-arvion ja teknisten määritysten takia, idean edistäminen on jäänyt toteuttamatta. Aihe valikoitu organisaatiossa käytyjen keskustelujen perusteella. Keskustelujen perusteella löydettiin kaksi aihetta, joista valitsin tämän.

Tutkimuskysymykset tälle opinnäytetyölle ovat:

- Miten olemassa olevaan ympäristöön voidaan lisätä automaattinen testausympäristö?
- Mitä testejä on mahdollista ja järkevää testata automaation kautta?
- Kuinka kauan automaattisen testausympäristön toteuttamiseen menee suunnilleen?

Tutkimusstrategiana käytetään tapaustutkimusta, sillä sen metodit vastaavat parhaiten tämän kaltaisen työn tarpeita. Menetelmä, joita strategian sisällä sovelletaan, on pääosin havainnointi. Havainnointi on tavallinen tapa työskennellä, kun lähdetään pohtimaan teknisiä määrityksiä ja perustelemaan aika-arvioita. Uuteen aiheeseen tulee perehtyä, tämän jälkeen voidaan reflektoida uutta tietoa asetettua ympäristöä ja asetettuja kysymyksiä vasten.

Työn osuus alkoi perehtymällä teoriaan. Teoria katsauksen aikana erityisesti pyrin tarkastelemaan mahdollisia sudenkuoppia, vastauksia kysymyksiin, minkälaisia lähestymisiä tulisi välttää. Näiden perusteella on mahdollista ideoida, miten toteutuksen voisi toteuttaa.

Työn toimeksiantajan toimii LianaTechologies, suomalainen viestintä alan yritys perustettu vuonna 2005 (LianaTechologies n.d). Tämä yritys tarjoaa viestintä ratkaisuja asiakkailleen käyttämällä omia tuotteitaan, joista yksi on verkkokauppa-alusta. Jokaisella näistä tuotteista on oma

kehitystiiminsä, joka vastaa tuotteen viemisestä eteenpäin ja ylläpidosta. Opinnäytetyön teoria katsauksen aikana saatua tietoa katselmoidaan ainoastaan verkkokauppa kehitystiimin kautta.

Opinnäytetyön aikana avataan, kuinka kehittämistyötä tehdään organisaation sisällä. Tämä auttaa valottamaan tilannetta ja työskentelytapoja, joita organisaatio käyttää. Automaation kautta suoritettavat testit vaikuttavat näihin menetelmiin.

Työn lopullista tulosta pohditaan viimeisessä kappaleessa. Tämä kappale tarjoaa vastaukset asetettuihin tutkimuskysymyksiin. Lopullinen pohdinta arvio työn lopullisuutta ja tulevaisuutta opinnäytetyön jälkeen.

2 Testaamisen yleinen teoria

Ohjelmistojen kehitykseen kuuluu uusien toiminnallisuuksien luominen, mutta myös vanhojen toiminnallisuuksien ylläpitäminen ja kehittäminen eteenpäin. Tämä voidaan rinnastaa hyvin perinteiseen insinöörimäiseen työhön, jossa rakennetaan fyysisesti uutta ja ylläpidetään vanhaa. Itse asiassa testaaminen terminä kuulostaa insinööritieteeltä muun muassa kemian tekniikan, koneenrakennuksen, maa- ja vesirakentamisen sekä sähkötekniikan alalta (Mili & Tchier 2015, 3).

Testaaminen on prosessina erittäin monimutkainen. Etenkin kun puhutaan ohjelmistokehityksestä. Jopa yksinkertaisella sovelluksella voi olla satoja, jopa tuhansia mahdollisia kirjoitus – ja luku kombinaatioita (Myers, Badgett. & Sandler 2012, 5).

Myers (2012) tarkoittaa, yksinkertainen lomake voidaan täyttää monella tapaa. Ensimmäisestä kentästä viimeiseen kenttään, tai viimeisestä kentästä ensimmäiseen. Tai täysin satunnaisessa järjestyksessä. Lomakkeella voi olla pakollisia kenttiä, tai tietyn kentän pakollisuus voi riippua toisesta kentästä, joka on pakollinen tai vapaaehtoinen.

Kun puhutaan verkkokehityksestä ja testaamisesta, käsite, joka toistuu usein, on bugi. Bugista käsitteenä on muodostunut yleinen kuvaus, joka kuvaa kaiken. Tämän käsitteen tarkoitus on tärkeä avata ja mitä muita käsitteitä se pitää tahattomasti pitää allaan. Koska ohjelmistotuotteet vaativat paljon suunnittelua, ohjelmistotuotteet ovat virhealttiimpia, kuin työskentely muilla tekniikan aloilla (Mili ym 2015, 10).

2.1 Testaaminen

Ohjelmien ja järjestelmien testaaminen on välttämätöntä, jotta voidaan välttää loppukäyttäjälle näkyvät virheet, jotka aiheuttavat negatiivista näkyvyyttä organisaatiolle. Bugit ja hallitsemattomat virheet aiheuttavat huonoa julkisuutta organisaatioille, joihin ne yhdistyvät. Testaaminen on tärkeä osa työtä, jossa kehitetään tuotetta eteenpäin. (Homés 2013, 1.)

2.1.1 Määritelmä

Terminä testaaminen on itsestään selvä. On vaikea kuvitella työtä, jonka työnkulusta puuttuu testaaminen. Se toteutetaan käytännössä aina samalla tavalla, testaaja tekee tietyn toiminnon tuotteen parissa ja odottaen tuotteelta tiettyä lopputulosta. Hän istuu tuolille odottaen, että se kannattaa hänen painonsa. Testaaminen on joukko toimintoja, joiden tarkoituksena on tunnistaa ohjelmiston tai järjestelmän ongelmia ja arvioida sen laatutaso käyttäjien tyytyväisyyden saavuttamiseksi. (Homés 2013, 8.)

Testaaminen on prosessi, jossa ohjelmaa suoritetaan tarkoituksena löytää virheitä. Osuva määritelmä, tätä kehittäjä tekee, kun hän testaa sovellusta. Hän suorittaa osaa ohjelmasta ja samalla tarkastelee, suoriutuuko ohjelma annettujen määrityksien mukaisesti. Virhe, tässä määrityksessä, on toiminto, joka aiheuttaa poikkeavaa toimintaa odotetusta lopputuloksesta. Virheen määritelmä on jo esitelty ja se on helppo sovittaa testaamisen määritelmän parametreihin. (Myers ym 2015, 6.)

Myers (2015) huomauttaa, että testaamisen määritelmä on harhaanjohtava. Ihmisillä on tapana keskittyä tavoitteeseensa, tavoitteen määritteleminen ihmiselle on tärkeää psykologisella tasolla. Mikäli tavoitteemme on todistaa, että ohjelmamme on virheetön, tällöin pyrimme alitajuntaisesti todistamaan, että ohjelmamme on virheetön. (Myers ym 2015, 6.)

2.1.2 Päämäärä

Testaamisella on kaksi tavoitetta, joihin sillä pyritään. Nämä tavoitteet jaetaan usein laadunvalvontaan ja laaduntarkistukseen. Laadunvalvonta on keskittynyt vikojen tunnistamiseen, laaduntarkistus pyrkii estämään niitä. Laadunvalvonta on tuotekeskeistä ja pyrkii varmistumaan, että tulokset ovat odotusten mukaisia. Toisaalta laadunvalvonta on keskittynyt enemmän prosesseihin, jotka takaavat, että laatu on sisäänrakennettu. (Garcia 2018, 13.)

Mikäli asiaa tarkastellaan oikeasta näkökulmasta, päämääränä on tuottaa arvoa. Laadunvalvonta ja laadun tarkistaminen pyrkivät juuri tähän. Tästä syystä testaaminen käytännössä kääntyy päälaelleen. Testaustilanne, joka löytää virheen, on onnistunut. Tarkemmin ajateltuna se on onnistunut, sillä se on osoittanut olevansa arvokas investointi. Epäonnistunut testaustilanne on se, joka saa ohjelman tuottamaan oikean tuloksen ilman virheitä. (Myers ym 2015, 7.)

Myers (2015) tarkemmin perustelee epäonnistuneen testin määritelmää vertaamalla, sitä lääkärikäyntiin. Epäonnistuneen testin määritelmään päädytään, koska olemme aloittaneet testaus prosessin. Prosessi on aloitettu, koska oletamme, että ohjelmassa on vika ja haluamme toistaa tämän vian. Tätä samaa ajatuksen juoksua käytetään, kun menemme lääkäriin. Mikäli emme löydä virheitä, on testaus epäonnistunut jossain vaiheessa.

2.1.3 Osana projektia

Projektissa määrittely ja toteutus ovat selkeitä osia projektissa, testaaminen on osa projektia, joka tapahtuu koko ajan. Testaaminen tapahtuu koko ajan projektin taustalla. Testausta on ja sen tulisi olla läsnä koko ohjelmiston elinkaaren ajan suunnittelun alusta aivan huollon loppuun ja koko ohjelmiston käytön ajan. (Homés 2013, 5.)

Testaajan tulee testata, että ohjelma pystyy hallitsemaan virhetilanteita. Verkkokauppa esimerkkiin peilaten, mikäli yritetään ostaa tuotetta, joka on loppuunmyyty. Testi on epäonnistunut tässä tilanteessa, mikäli sovellus antaa käyttäjän ostaa tuotteen. Testi on myös epäonnistunut, mikäli ohjelma kaatuu. Testi on onnistunut, jos sovellus kertoo käyttäjälle, että tuotteen varastosaldo on loppunut. Määritykset ohjaavat, miten jokainen tapaus hoidetaan. Nyt kun tiedämme, mikä on oikea ohjelma, voimme helposti määritellä, mikä on ohjelman vika: se on mikä tahansa käyttäytymismalli, joka väärentää tai poikkeaa annetuista määrityksistä (Mili ym 2015, 101).

Mikäli kehittäjä on ainoa henkilö, joka testaa ohjelmistoa, se johtaa lisääntyneeseen puolueellisuuteen, jossa tekijä toimii samanaikaisesti, sekä tuomarina että valitsijahenkilönä (Homés 2013, 212).

Homés (2013) puhuu omassa teoksessaan ammattitestaajien käyttöön ottamisesta organisaatiossa. Tässä raportissa pohditaan automaatio testauksen käyttöönottoa ja sen hyviä ja huonoja puolia. Lopputuloksena molemmissa käyttötarkoituksissa on vähentää kehittäjän roolia varsinaisessa testauksessa.

2.1.4 Haasteet

Testaamisella on kaksi tärkeää seurausta: (1) Et voi testata ohjelmaa ja olla varma, että se toimii virheettömästi; ja (2) kustannustehokkuus on testaamiselle keskeinen osa. Tyhjentävä testaus on

mahdotonta, tavoitteen tulisi olla maksimoida testausinvestoinnin tuotto maksimoimalla lopullisen määrän testitapausten löytämien virheiden määrä. (Myers ym 2011, 10.)

Tämä aiheuttaa eräänlaista sokeutta, on vaikea nähdä metsää puilta. Tai ajatella sovellusta ja sen logiikkaa loppukäyttäjän silmin. Tätä dilemmaa tarkastellaan tarkemmin, kun puhutaan valkoisesta – ja mustasta laatikosta. Kehittäjä on ensimmäinen henkilö, joka näkee ja testaa ohjelmistoa. Olemme kaikki enemmän tai vähemmän sokeita omiin puutteisiimme ja omille virheillemme: kun luemme kirjoittamamme, on mahdollista, että havaitsemme useita oikeinkirjoitusvirheitä, mutta aiottu lukijamme tunnistaa useita muita virheitä. (Homés 2013, 22.)

On jo mainittu, kuinka ohjelmistokehityksessä sovelluksen käyttötavat kasvavat suurin lukemiin. Tämä vaikeuttaa testaamista prosessina. Testatessa edes pientä laskinta, "kaiken testaaminen" tarkoittaisi kaikkien syöttöarvojen, toimintojen, kokoonpanojen (laitteisto ja ohjelmisto) yhdistelmien kokeilemista, mikä johtaisi melkein rajattomaan määrään testitapauksia, näiden testitapausten suunnittelu ja toteutus olisi absurdia (Homés 2013, 11).

Tämä sama ongelma on vastassa automatisoitujen testien kanssa. Testit tulee suunnitella ja toteuttaa, mutta kaikkia mahdollisia kombinaatioita on erittäin vaikea ottaa huomioon. Siksi tähän kuuluu suunniteltavien ja suoritettavien testien valinta. Tämä on riskienhallintaa. (Homés 2013, 11.)

2.1.5 Testaamisen vaiheet

Testaaminen vaatii suunnittelua, työtä ja toimintaa. On tärkeää, että testaamisen tarkoitus on selkeä ja prosessi toteutetaan kontrolloidusti. Annetun tiedon pohjalta on vaikeaa tehdä suunnitelmia. Homés (2013) listaa suunnitellulle testaamiselle viisi vaihetta; suunnittelu, analyysi, suoritus, arviointi ja päätös. Testaamisen vaiheet on kuvitettu kuvassa 1.

Suunnittelu on vaihe, joka on mukana kaikissa ihmisen elämän prosesseissa. Vaihe on tietenkin vapaaehtoinen, mutta tulisi silti toteuttaa aina selkeän prosessin nimissä. Suunnitteluun kuuluu tehtävien organisointi ja koordinointi muiden sidosryhmien, kuten kehitystiimien, tukitiimien, käyttäjien edustajien, johdon, asiakkaiden jne kanssa (Homés 2013, 15).

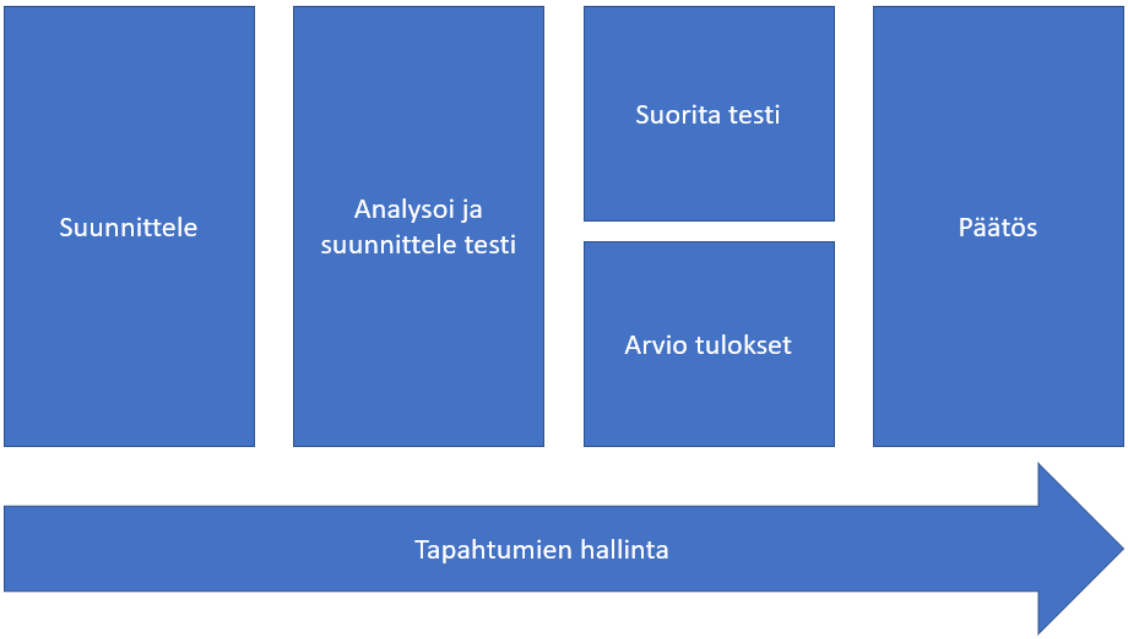
Kuten on jo määritelty aiemmin testaamisen luonteesta. Kyseessä on odotusarvon asettaminen, toiminnon toteuttaminen ja sitten saadun arvon vertaamista saatuun tulokseen. Analyysi antaa

mahdollisuuden määritellä testin tavoitteet, priorisoida ne ja arvioida pohjan ja tavoitteiden testattavuus (Homés 2013, 16). Analyysin tuloksia voidaan sitten verrata bugin tasoihin ja jatkaa toimia näiden tulosten perusteella.

Suorittaminen on nimensä perusteella hyvin selkeä vaihe. Testi on suunniteltu ja analysoitu, joten seuraavaksi on aika suorittaa testi. Suorittaminen on testiolosuhteiden muuntaminen testitapauksiksi ja testausmenettelyiksi, erityiset testitiedot ja tarkat odotetut tulokset (Homés 2013, 19).

Suorittaminen antaa tulokset testaajalle. On tärkeää arvioida tulokset. Mikäli metsästettiin bugia, saatiinko tilanne toistettua, jossa bugi ilmenee. Myös testin laadun arviointi on tärkeää, suoritettiin testi suunnitelulla tavalla. Analyysissa arvioidaan testikohdetta suhteessa suunniteluun, määrittelyyn, tavoitteisiin ja kriteereihin. Tämä arviointi suoritetaan testin suorittamisen aikana ja määrittää onko tarpeen muiden testiprosessien toteutus. (Homés 2013, 20.)

Päätös vaiheessa peilataan takaisin suunnittelu vaiheeseen, jossa kaikkia osapuolisia on informoitu tilanteesta. Nyt testin tulokset on tuotava heidän tietoonsa, jotta voidaan kulkea eteenpäin uuden tiedon valossa (Homés 2013, 21).



Kuva 1. Suunniteltu testausprosessi.

Esitetty testaamisen kulku toimii kaikissa testaamisen muodoissa. Musta – ja valkoinen laatikko, konseptit, joista puhutaan enemmän seuraavassa kappaleessa. Mutta myös manuaalisessa testaamisessa, kuin myös automaation kautta suoritetussa.

2.2 Verkkokauppa

Verkkokauppa on osa sähköistä kaupankäyntiä ja termi kuvaa erityisesti sellaista verkossa tapahtuvaa kauppatapahtumaa, jossa ostaja on ihminen (Hallavo 2013). Verkkokauppa on luonteeltaan teknologinen kerros, joka yhdistää yrityksen perustietojärjestelmät ja -prosessit useisiin palvelukanaviin verkossa. Verkkokauppa mahdollistaa verkon eri palvelukanavien tarjoamisen ja niiden ketterän kehittämisen. (Hallavo 2013.)

Verkkokaupan toiminta perustuu kanavoihin, joita tulkitsemalla voidaan luoda eheä kanavien kokonaisuus (Hallavo 2013). Hallavo (2013) esittää, että verkkokaupassa on monta teknologia tasoa (kuva 2). Korkein taso on käyttöliittymät ja kanavat, joka on taso, jota loppukäyttäjä käyttää laitteellaan. Seuraava taso on kaupan toiminnallisuus, joka on varsinainen verkkokauppa-alusta, joka tarjoaa tiedon käyttäjän laitteella. Seuraavaksi tulee tilaukset ja varastonhallinta taso, sekä tuotehallinta taso. Nämä tasot toimivat kaupan toiminnallisuuden alla tarkoituksena esittää tietoa kauppiaalle tuotteista, jota on hän syöttänyt alustaan ja tilauksista, joita käyttöliittymä tason kautta on tehty. Viimeinen on integraatio taso, jossa verkkokauppa kommunikoi muiden järjestelmien kanssa.

Kuva 2. Verkkokaupan teknologia kerrokset.

Käyttöliittymä ja kanavat tasoon on myös mahdollista liittää verkkokaupalle ulkoisia kanavia. Näitä ulkoisia kanavia voivat olla sosiaalisen median integraatiot. (Hallavo 2013.)

Kerrosmallin kautta organisaation on mahdollista arvioida, miten sen infrastruktuuri ja sen sisäiset prosessit pystyvät tukemaan verkkokaupan tuomaa moni kanavanaisuutta, tai mitä ratkaisuja verkkokauppa-alustan tulisi pystyä tarjoamaan. On mahdollista, että yllä kuvatut (kuva 2) toiminnot ovat jo olemassa organisaatiossa, mutta vain tiettyä tarkoitusta varten, kuten myymäläketjun ohjaamista varten. (Hallavo 2013.)

Hallavo (2013) listaa verkkokaupan toimintoja: ostoskorin saatavuustarkistukset, tilausten jakaminen eri toimituksiin ja toimituskohtainen ohjaaminen, varastosaldojen hallinta ja esittäminen, tavarantoimittajien tai tukkureiden toimituslogistiikan ohjaaminen, tilauksen statuksen viestiminen asiakkaalle, myymälätoimitusten ja -noutojen hallinta, palautuslogistiikka ja rahojen palautukset ja raportointi taloushallintoon. Nämä toiminnot muodostavat kanavia. Kun useita toimintoja liitetään yhteen päästään lähemmäksi monikanavaisuutta.

2.3 Projektinhallinta scrum mallin mukaan

Scrum on iteratiivinen, mutta se myös kannustaa työryhmän jäseniä oppimaan uutta ja käyttämään tätä uutta opittua tietoa hyväkseen. Scrumin vastakohta on vesiputous malli, jossa edetään peräkkäisessä järjestyksessä. (Pries & Quigley 2010, 1.)

Vesiputous mallissa toimitaan vaiheittain, vaiheen on valmistuttava ennen seuraavan vaiheeseen siirtymistä. Todellisuudessa tämä malli toimii hyvin harvoin projekteissa, sillä on mahdollista, että tilanteet muuttuvat, jotka vaikuttavat projektin määrityksiin. (Pries ym 2010, 15.)

Scrum malli sallii huomattavasti enemmän hengitys tilaa projektin aikana. Tyypillinen suunnittelu tapahtuu viikkojen päähän ja suunnitellut toimet ovat selkeitä ja saavutettavissa. (Pries ym 2010, 16.)

Kun projektia viedään eteenpäin scrum-mallin mukaan, käyttäjätarinat ovat menetelmä vaatimusten selittämiseksi. Käyttäjätarinat hyötyvät siitä, että ne ovat ytimekkäitä ja asuvat usein hakemistokortin rajoissa. (Pries ym 2010, 18.) Vaatimukset on kirjattu hakemistokorttiin (Pries ym 2010, 18).

Määrittely on kuvaus vaadituista ominaisuuksista, jotka tuotteen on täytettävä. Määrittely johdetaan yleensä tunnistamalla kaikki ohjelmistotuotteen sidosryhmät, asettamalla vaatimukset, jotka heidän odotetaan täyttävän, muotoilemalla ja yhdistämällä nämä vaatimukset ja kokoamalla ne yhtenäiseksi asiakirjaksi. (Mili ym 2015, 38.)

Milin (2015) määrittelyn tulee täyttää kaksi ominaisuutta, jotka ovat muodollisuus ja abstraktio. Muodollisuudella viitataan riittävään kuvaukseen toiminnallisuudesta, mikä kertoo, minkälaista toiminnallisuutta vaaditaan. Abstraktiolla tulee vastata kysymyksiin, mitkä vaatimukset tulee täyttää.

Käyttäjätarinan kirjoittamisen perusta keskittyy pieniin ja ytimekkäisiin lauseisiin. Lauseet on muotoiltu pieneksi toiminnalliseksi lisäykseksi, joka voidaan toimittaa yhden kehityksen iteraation sisällä. Kun tarina näyttää liian suurelta tulisi kirjoittajalle mahdollistaa aikaa, jonka tarinan voi pilkkoa vielä pienemmiksi osiksi. (Hughes 2013, 117.)

Scrum-mallissa työaika, jota kutsutaan sprintiksi, ja määrä jaetaan pienempiin helposti hallitaviin osiin. Tiimillä ja projektipäälliköllä on päivittäistä ajatusten vaihtoa, minkä tarkoitus on vähentää

työtä hidastavia ongelmia. Projektipäällikkö ja tiimi ovat keskittyneet tiettyihin sprintille lueteltuihin tehtäviin ja yhteistyö on laaja-alaista, jotta varmistetaan tehtävien onnistuminen sprintin aikana, johon joukkue on sitoutunut, mikä lisää ajan laatua. käytettyjä tietoja. (Pries ym 2010, 57.) Tehtävien suunnittelu ja arviointi on tärkeä osa scrum-mallia. (Pries 2010, 22).

Scrum-malli ohjaa ainoastaan ajankäyttöä projektien etenemisessä, mutta mallista puuttuu mahdollisuus visualisoida ajankäyttöä ja ajankäytön suhde määrättyihin tehtäviin. Tämän voidaan lisätä scrum-malliin lisäämällä kanban-malli, jolloin saadaan ”scrumban”. (Hughes 2013, 284.) Kanbanin tarkoitus on ilmoittaa, mikä on valmiina työtä varten. (Pries 2010, 22).

2.3.1 Ketterä kehittäminen ja ketterä testaaminen

Ketterä kehitys suosii iteratiivista ja inkrementaalisesti etenevää kehitystä luopumalla laadusta, joka korostaa asiakkaan roolin merkitystä projektissa. Perinteisillä ohjelmistokehitystavoilla on tapana minimoida asiakkaan roolia. Vaikka ketterät menetelmät prosessien joustavuuteen, pääpaino on asiakastyytyväisyydessä. (Myers ym 2012, 175–176.)

Mitä tahansa kehitysmenetelmää voidaan kutsua ketteräksi. Kunhan menetelmästä voidaan tunnistaa kolme tunnistavaa tekijää: Asiakkaalla on merkittävä rooli kehityksessä, testaamisella on merkittävä rooli ja kehitysjaksot ovat lyhyitä, sekä iteratiivisia. (Myers ym 2012, 176.)

Ketterää testaaminen voidaan nähdä ryhmätyönä, jossa kaikki ovat mukana prosessin suunnittelussa, toteutuksessa ja testaussuunnitelman luonnissa, sekä toteuttamisessa. Asiakkaat osallistuvat testien määrittelyyn määrittelemällä käyttötapaukset ja ohjelman määritteet, sekä antavat hyväksynnän toteutukselle. (Myers ym 2012, 178.)

Koska ketterissä metodeissa, myös ketterässä testaamisessa toimitaan nopeasti. Toimintaan myös osallistuu useita toimijoita. On tehokas kommunikaatio kaikkien osapuolten välillä erittäin tärkeää (Myers ym 2015, 178).

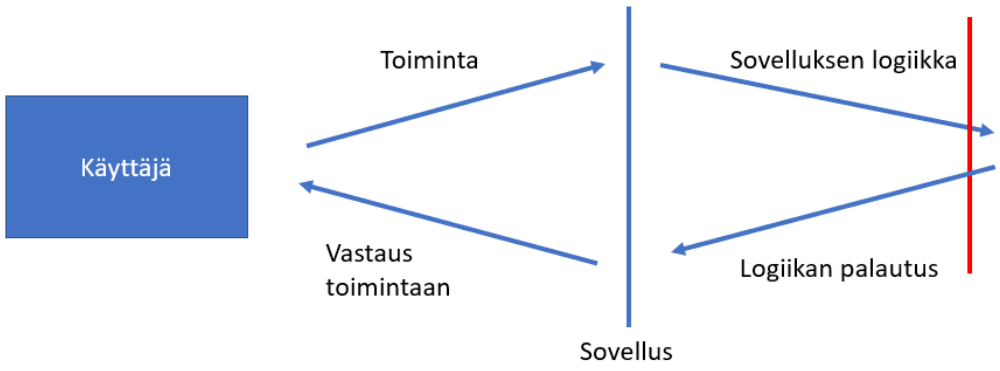
Ketterä kehittäminen koostuu usein pienestä kehittäjäryhmästä, jotka toimivat myös testaajina. Suuremmissa, enemmän resursseja sisältävissä projekteissa voi olla yksittäinen testaaja tai testausryhmä. Näissä tilanteissa testaaja voi vaikuttaa negatiiviselta, jopa syyttelevältä henkilöltä. Testaajan tehtävänä on viedä projektia eteenpäin antamalla palautetta ohjelmiston laadusta, jotta kehittäjät voivat toteuttaa virhekorjauksia ja tehdä vaatimusten muutoksia ja yleisiä parannuksia. (Myers ym 2015, 179.)

Hyväksyntätestien tarkoituksena on saavuttaa riittävä luottamus järjestelmään riippumatta toiminnallisesta näkökulmasta. Virheiden etsiminen on vain sivutuote, päätavoite on vahvistaa luottamusta. Jos havaitaan liian monta vikaa, luottamus ohjelmistoon tai järjestelmään vähenee ja asiakas suhtautuu skeptisemmin ohjelmiston tuleviin toimituksiin. (Homés 2012, 64.)

Hyväksyntätestin suorittavat käyttäjät tai asiakas, yleensä riippumattomana kehitystiimistä. Myös muut sidosryhmät voivat osallistua tähän testitasoon. Hyväksyntä testi suoritetaan yleensä mustan laatikon muodossa. (Homés 2012, 64.)

2.4 Bugi

Bugia käsitteenä voidaan määritellä, kun odotettu toiminnallisuus ja toteutunut toiminnallisuus poikkeavat toisistaan (Gupta 2019). Käyttäjä tekee toiminnallisuuden, jota esitetään poispäin menevällä nuolella (kuva 3). Sovellus ottaa tästä toiminnosta kiinni, pystyviiva esittää sovellusta, joka alkaa suorittamaan omia toimintojaan. Mikäli kaikki menee oikein, logiikka palauttaa sovellukselle tiedon, joka viedään käyttäjän tietoisuuteen. Mutta bugi, tässä kuviossa punainen pystyviiva, katkaisee kulun toiminnan kulun.



Kuva 3. Sovelluksen logiikka kuvitettuna, punainen viiva esittää bugia.

Guptan määritystä voidaan verrata toiseen lähteeseen ja samankaltaisuuksia on havaittavissa. Bugi voi olla virhe, erehdys, puute tai vika, joka aiheuttaa poikkeavuuden odotetusta lopputuloksesta (Technopedia 2017).

2.4.1 Bugin tasot

BrowserStack listaa kolme tasoa bugille: vähäinen, normaali ja korkea (BrowserStack 2020). Tasolle määritetty nimi kertoo tason luonteesta paljon. Tason nimi viittaa, kuinka nopeasti ongelma tulisi pystyä ratkaisemaan (BrowserStack 2020).

Bugilla on myös vakavuusaste, jotka on määritelty. Matala, vähäinen, vakava, kriittinen (BrowserStack 2020). Nimet jälleen kerran kertovat paljon vakavuuden tasosta. On huomion arvoista, että nämä kaksi esiteltyä tasoa risteävät toistensa kanssa. Kriittinen bugi voi olla vähäisellä tasolla ja toisinpäin. Verkkokauppa esimerkkiin sovellettuna, bugi on kriittinen, mikäli se vaikuttaa ostamiseen. Mutta sen taso on matala, mikäli sitä on vaikea saada toistumaan ympäristössä. Mikäli ongelma on vaikea toistaa, loppukäyttäjät harvemmin törmäävät siihen. Esimerkki on luonnollisesti karkea ja teoreettinen, käytännössä kriittisen tason bugi, vaikka se olisi vaikea toistaa, tulisi ratkaista nopeasti, jollain tasolla.

2.5 Virhe

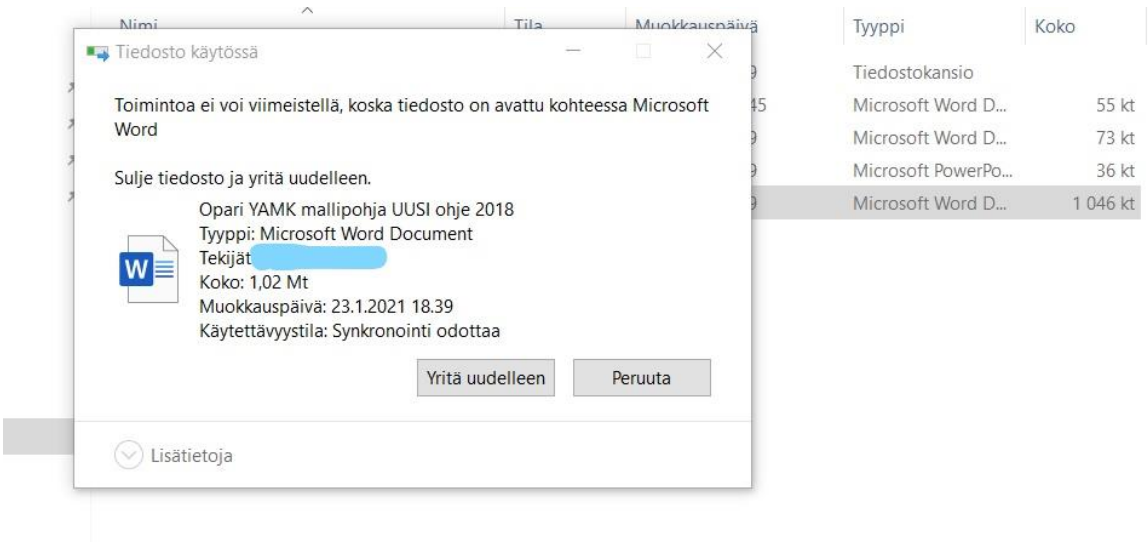
Monet käsitteet kuvaavat vääränlaista toiminnallisuutta: bugi, virhe, epäonnistuminen, puute, vika, erehdys jne. Näitä käsitteitä monesti käsitellään yhdenvertaisina, mikä voi johtaa väärin käsityksiin. (Homés 2013, 4.) Käsitteistä kun puhutaan, niin paljon päällekkäisyyksiä on havaittavissa.

Homés (2013) mainitsee korkeammalla painoarvolla näistä käsitteistä: virheet, vika ja epäonnistuminen. Virheet syntyvät ihmisen toimista, näin ollen vian juuri ja epäonnistumisen juuri piilee viassa. (Homés 2013, 4.) Käsitteet liikkuvat todella lähellä toisiaan. Aiemmassa kappaleessa viitatussa artikkelissa bugi ja vika oli lokeroitu samalle tasolle.

Ihmisen toiminta on loppupeleissä syy, mikä johtaa näihin tilanteisiin. Vikoja voi esiintyä ohjelmistoissa, mutta ne voivat esiintyä myös ohjelmiston dokumentoinnissa. Suuri määrä ohjelmistoongelmia johtuu vaatimuksista tai määrityksistä, jotka voivat olla epäselviä, jopa epäjohdonmukaisia tai yhteensopimattomia. (Homés 2013, 4.)

Kuva 4 esittää virhetilannetta, jossa alusta ennakoi käyttäjän tekemän virheen ja esti sen. Käyttäjä yritti poistaa tiedoston, mikä hänellä oli auki samalla hetkellä Word ohjelmassa. Tiedoston poisto

estettiin ja käyttäjälle annettiin ilmoitus, joka kertoo mitä oltiin tekemässä ja miksi näin tapahtui. Toisin sanoen käyttäjälle tarjottiin virheilmoitus.



Kuva 4. Käyttäjä yrittää poistaa tiedostoa, joka hänellä on auki ohjelmassa.

Sovelluksen sisällä on mahdollista ennakoida tilanteet, jotka aiheuttavat virheen ja luoda ratkaisu näille tilanteille. Virheenhallinta on prosessi, joka koostuu sovellusvirheiden, ohjelmointivirheiden tai tiedonsiirtovirheiden ennakoinnista, havaitsemisesta ja ratkaisemisesta (Technopedia n.d).

Toimivassa ohjelmassa tapahtuu virheitä, sillä ohjelmat suunnitellaan ihmisten käyttöön ja ihmisillä on tapana tehdä virheitä. Käyttäjäkokemuksen kannalta on tärkeää, että näihin virheisiin on varauduttu. Käyttäjän kokemus olisi ollut täysin pilalla, mikäli hän olisi pystynyt poistamaan opinnäytetyönsä. Sillä hetkellä, kun se on hänellä auki ohjelmassa ja hän tuottaa uutta sisältöä siihen. Inhimillisten virheiden - ja siten ohjelmistoon tuotujen vikojen määrän - vähentämiseksi voidaan toteuttaa koulutustoimia tai asettaa tiukempia prosesseja (Homés 2012, 4).

3 Automaatiotestaaminen

Automatisoitu testaaminen on sama asia, kuin manuaalinen testaaminen. Se suoritetaan vain automaattisesti. Perinteisesti ohjelmiston testaamistekniikat voidaan jakaa, joko mustan laatikoon testaamiseen (black-box testing), tai valkoisen laatikon testaamiseen (white-box testing) (Nindra & Dondeti 2012, 29).

Perehdytään näihin kahteen tekniikkaan testauksen automaation kautta. On kuitenkin huomion arvoista, että mainittuja tekniikoita voidaan soveltaa myös manuaalisessa testauksessa. Musta laatikko erityisesti suosii manuaalista suorittamista. Musta laatikko hakee selkeää käyttäjäkokemusta ja dokumentaatiota, välittämättä laatikon sisäisistä toimista (Homés 2012, 144).

Testit ensin metodi on yksinkertaistetusti menetelmä, jossa testi toteutetaan ennen varinaista toteutusta. Normaalisi ohjelmointi nähdään erittäin ratkaisukeskeisenä alana, jossa toteutus tulee ennen testausta. (Garcia 2018, 8.)

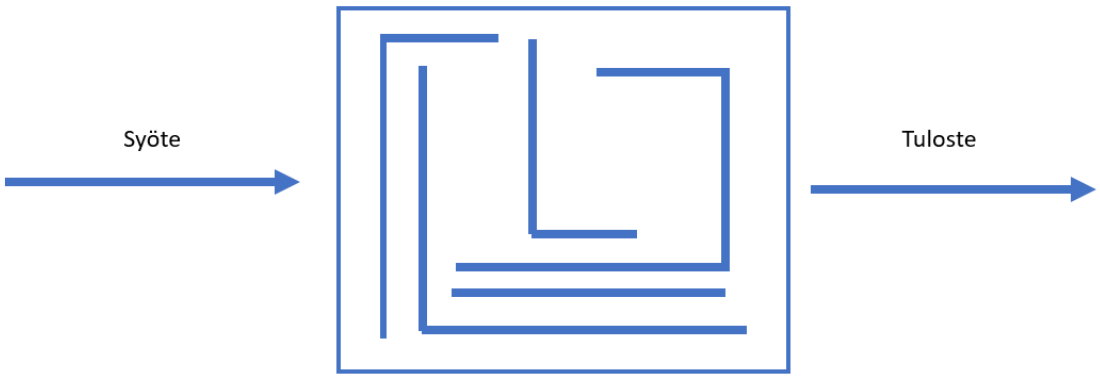
Testit ensin -ohjelmointitapa kannustaa rakentamaan ohjelman logiikan yksittäisinä yksikköinä, puhutaan yksikkötestaamisesta. Yksikkötestaus on käytäntö, joka pakottaa meidät testaamaan pieniä, yksittäisiä ja eristettyjä koodiyksiköitä. Yksikön muodostaa yleensä menetelmä, luokka tai mahdollisesti kokonainen sovellus. (Garcia 2018, 85.)

3.1 Valkoinen laatikko

Valkoisen laatikon testaaja, yleensä kehittäjä, tietää miltä koodi näyttää ja suorittaa testitilanteen suorittamalla ohjelmaa tietyillä parametreillä. Valkoisessa laatikossa testaaja katsoo testattavan ohjelmiston sisälle ja käyttää kyseistä tietoa testausprosessin osana (Garcia 2018, 11).

Toinen keskeinen piirre valkoisessa laatikossa, joka myös erottaa sitä mustasta laatikosta. Valkoisen laatikon testaamista käytetään pääosin, kun etsitään loogisia virheitä koodista (Nindra ym 2012, 38).

Kuvassa 5 ohjelma on esitetty valkoisen laatikon muodossa. Viivat laatikon sisällä kuvasta ohjelman logiikkaa, joka suoritetaan, kun käyttäjä antaa syötteen laatikolle. Lopulta laatikko antaa tulosteen käyttäjälle.



Kuva 5. Valkoinen laatikko, viivat laatikon sisällä symbolisoivat ohjelman logiikkaa.

Tämä tekniikka painottaa vahvasti teknistä puolta ohjelmassa. Tekniikan vahvuus on sen läpinäkyvyys. Ohjelman logiikka on nähtävillä. Tämän ansiosta testin suunnittelu ja toteutus on selkeää. Garcia (2018) korostaa, että valkoinen laatikko vaatii ymmärrystä sisäisistä prosesseista ja teknistä osaamista.

Myers (2012) on todennut, että ihmisillä on alitajuinen tarve päästä oletettuun tulokseen. Tämä tietenkin voi johtaa testin vääristymään, kun kaikki on nähtävillä, niin testaaja voi alitajuisesti suorittaa logiikan testin lopputulos silmissään. Hän pystyy antamaan vain ne syötteet, tai muokkaamaan logiikan, näin saaden sen lopputuloksen, mitä hän haluaa. Selkeä haittapuoli tässä tavassa on se, että tarvitaan pääsy koodiin ja ymmärrys koodista (Garcia 2018, 13).

Garcia (2018) korostaa, että valkoisen laatikon testaamisen metodia käytetään melkein aina automaation kautta ja testaaminen toteutetaan yksikkötestaamisena (Garcia 2018, 13). Yksikkötestaaminen on tapa lähestyä ohjelmointia ja testausta.

Valkoisen laatikon vahvuuksia ovat muun muassa: se on tehokas tapa löytää virheitä, sen kautta voidaan löytää piilotettuja virheitä ja se kannustaa katselmointi käytäntöä (Garcia 2018, 12–13). Metodilla on myös heikkoutensa. Valkoinen laatikolta puuttuu ymmärrys määrityksistä, se hajoaa herkästi jatkokehitysten aikana ja se vaatii pääsyn koodiin, sekä ymmärryksen koodista (Garcia 2018, 13).

Valkoisen laatikon testaaminen voidaan toteuttaa kahdella tavalla: staattisesti tai rakenteellisesti. (Nindra 2012, 38.) Seuraavissa luvuissa perehdytään näihin kahteen tekniikkaan tarkemmin.

3.1.1 Staattinen valkoinen laatikko

Staattinen valkoisen laatikon testaaminen käsittää ainoastaan tuotteen lähdekoodin, sulkien binaarien ja muiden suoritettavien testauksen pois ja testaus suoritetaan ennen kuin koodi ajetaan. Staattiseen testaukseen osallistuvat ainoastaan tietyt henkilöt, jotka tarkastelevat koodin laatua. (Nindra ym 2012, 38.)

Käytännössä staattinen testaaminen voidaan nähdä koodin katselmointeina, tässä käytännössä ryhmä teknisiä henkilöitä käy koodin läpi (Nindra 2012, 39). Ryhmä voi tässä kontekstissa olla myös yksittäinen henkilö. Katselmointikäytäntö riippuu siitä, miten organisaatio taustalla haluaa hoitaa katselmointiprosessin. Yleisesti katselmoinnin suorittaa yksittäinen henkilö, joka hyväksyy tai pyytä parannuksia toteutukseen. Tämä henkilö voi kuitenkin pyytää muita organisaation jäseniä myös katselmoimaan samaa ja antamaan oman hyväksyntänsä, ennen kuin itse hyväksyy tuotoksen. Koska olemme kaikki suhteellisen sokeita omiin vikoihimme, on tärkeää, että muut ihmiset katsovat suunniteltavaa ohjelmistoa. Testin riippumattomuus kehityksestä mahdollistaa riippumattoman arvioinnin ja siten rajoitetun häiriöntestin ja suunnittelun ristiriitaisten tavoitteiden välillä. (Homés 2013, 23.)

Kuvassa 6 on esitetty yksi ratkaisu hypoteettiselle tehtävälle, jossa tarkoituksena on lisätä kaksi numeroa yhteen. Tämä ratkaisu täyttää vaatimukset puutteellisesti. Jälkimmäiset kutsuvat palauttavat epävalidin tuloksen.

```
/**
 * Add two numbers together
 */
function doPlus(a, b) {
  return a + b;
}
var a = doPlus(1, 1);
var b = doPlus();
var c = doPlus('Hello', 'World');
```

Kuva 6. Määrityksen mukaan tulisi laskea kaksi numeroa yhteen.

Hómes (2013) listaa katselmoinnin päämääriä: tuote täyttää sille annetut määritykset, tuote täyttää sille annetut laatumääritykset. Muitakin päämääriä voidaan listata, mutta nämä kaksi ominaisuutta ovat opinnäytetyön puitteissa tärkeimmät. Kuvaan 5 palaten, suoritettu ohjelma täyttää määritykset. Mutta laatumääritykset se täyttää puutteellisesti.

Toinen osa staattista valkoista laatikkoa on, viralliset tarkastelut tai Fagan tarkasteluprosessi. Virallisen tarkastelun tarkoitus on löytää virheitä ohjelmasta (Nindra ym 2012, 39). Nämä metodit ovat paljon jäykempiä ja tarvitsevat paljon virallisemman prosessin. Toisin sanoen katselmointi tapahtuu enemmän tapaamisen muodossa.

On huomion arvoista, että ketterän kehityksen prosessin on pystyttävä reagoimaan nopeisiin muutoksiin projektissa, sekä laajuudessa että määrityksissä. Mutta kehityksen on myös pystyttävä vastaamaan annettuihin toimitus -ja laatuvaatimuksiin tavalla, joka on kustannustehokasta, vaatimatta suuria sankaruuden osoituksia kehittäjiltä. (Holcombe 2008, 2.) Katselmointi käytäntö on hyvä, sillä se parantaa varsinaisen toteutuksen laatua ja ennakoi ongelmia. Mutta se hidastaa projektin etenemistä.

3.2 Musta laatikko

Aiemmassa tekniikassa testaajalla on käsissään valkoinen laatikko, jonka sisään hän näkee. Nyt hänen käsissään on musta laatikko, jonka sisältö on mysteeri. Testaaja on tietoinen, mitä ohjelman tulee tehdä, mutta häneltä puuttuu tieto, miten ohjelma toteuttaa toiminnon (Garcia 2018, 11). Musta laatikko on toteutumisensa kannalta valkoisen laatikon vastakohta.

Samankaltainen kuvio (kuva 5) esitettiin, kun tarkasteltiin valkoista laatikkoa. Kuvio ja ohjelma kuvion sisällä toimii täysin samalla tavalla, kun tarkastellaan mustaa laatikkoa. Ohjelma nähdään tässä tilanteessa mustana laatikkona (kuva 7). Laatikon sisäisten toimien sijaan testausprosessi keskittyy tarkastelemaan tilanteita, joissa ohjelma toimii vastoin odotuksia. (Myers ym 2012, 8–9.)

Kuva 7. Musta laatikko.

Musta laatikko –testauksen suurin vahvuus on, että se painottaa määrityksiä. Toisin sanoen se luo kuilun kehittäjän ja loppukäyttäjän välille (Garcia 2018, 12). Kehittäjä on nähnyt laatikon sisälle, joten hänen voi olla vaikea asettua tähän mielentilaan. Prosessin voi suorittaa ilman teknistä osaamista (Garcia 2018, 12). Tästä syystä testaaja voi olla kuka vain organisaatiossa.

Mili (2015) puhuu, että testaamisen prosesseihin kuuluu validointivaihe. Vaiheen tarkoitus on varmistua, että tehty työ täyttää annetut vaatimukset. Validointi on helppo rinnastaa mustaan laatikkoon, sillä lopullisen hyväksynnän antaa asiakas. Henkilö, joka näkee syötteen ja tulosteen ja tietää mitä näiden kahden välillä tapahtuu.

Mustan laatikon testaaminen on käytännössä sama asia, kuin käyttäjäkokemuksen testaaminen. Laatikon testaajan tuli tarkkailla seuraavia ominaisuuksia, kun hän suorittaa testausprosessia.

1. Onko käyttöliittymä selkeä, ottaen huomioon tarkoitetun käyttäjäkunnan, jota halutaan palvella?

2. Onko käyttäjältä vaaditut syötteet selkeitä ja ymmärrettäviä?

3. Onko virheen hallinta selkeää ja helposti ymmärrettävissä?

4. Onko käyttöliittymä yhdenmukainen?

5. Mikäli syötteissä vaaditaan tarkkuutta, kuten verkkopankkijärjestelmissä, onko syötteissä tarpeeksi redundanssia?

6. Tarjoaako käyttöliittymä vain oleellisia vaihtoehtoja tarkoitetulle käyttäjälle? Vaihtoehtoja kuten linkkejä.

7. Tarjoaako käyttöliittymä suoraa interaktioita käyttäjän kanssa? Esimerkiksi tilanteessa, jossa hän syöttää arvoa syötekenttään.

8. Onko ohjelma helppokäyttöinen?

9. Rohkaiseeko käyttöliittymän käyttäjä toimimaan tarkasti?

10. Onko käyttäjäkokemus yhdenmukainen kaikkialla, toisin sanoen onko ohjelman käyttö helposti opittavissa?

11. Onko käyttäjäkokemus helposti navigoitavissa?

12. Täyttikö ohjelma sille annetut määritykset?

(Myers ym 2011, 144–145.)

3.3 Musta laatikko ja valkoinen laatikko -testaukset

Valkoinen laatikko lähestyy testaamista täysin eri kannalta, mitä musta laatikko. Tekniikat täydentävät toisiaan. On vahvasti suositeltavaa käyttää, sekä mustaa laatikkoa että valkoista laatikkoa varmistaakseen parhaan tasoisen testauksen laadun (Myers ym 2015, 83).

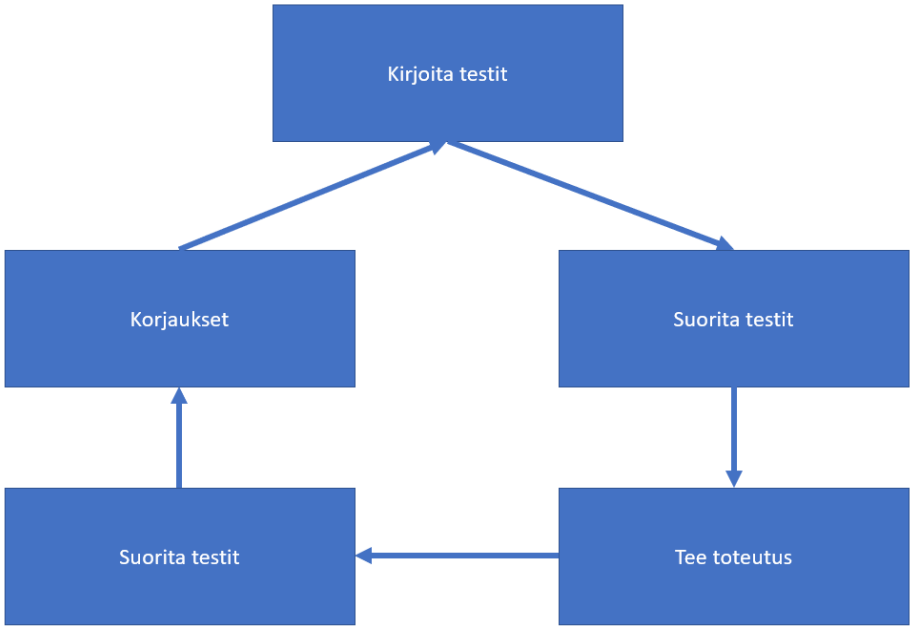
Testaajalla voi olla ymmärrystä ohjelman sisäisestä logiikasta, hän voi olla osallisena, kun sisäistä logiikkaa määritellään ja toteutetaan. On hyvin mahdollista, että hänellä on myös ymmärrystä ohjelman ulkoisesta logiikasta. Tällöin testaaminen on vaikea toteuttaa puhtaasti valkoisen – tai musta laatikon kautta, joten testaus voi kutsua “harmaaksi laatikoksi”. (Holmés 2012, 144.)

Puhtaasti tämän tekniikan toteuttaminen on vaikeaa, sillä projektiin osallistujien voi olla vaikea nähdä sovellus loppukäyttäjän silmin. He kaikki ovat osallistuneen määrittelyyn ja tämän kautta ovat osallisia tekniseen toteutukseen. Mustan laatikon suurin vahvuus on, että testaaminen tehdään täysin loppukäyttäjän näkökulmasta (Nindra ym 2012, 33).

3.4 Testit ensin -ohjelmointitapa

Ohjelmointia ajatellaan teknisenä lajina ja testaaminen tapahtuu samaan aikaan osana tätä lajia useiden eri toimijoiden toimesta. Koko prosessia voidaan lähestyä testit ensin -mentaliteetilla, joka on yksinkertaisesti tapa kirjoittaa testit ennen varsinaista toteutusta (Garcia 2018, 8). Testit ensin -ohjelmointitapa on helppo käsittää väärin. Toimintavan tarkoitus on kannustaa suunnittelemaan testit ennen varsinaista toteutusta. (Garcia 2018, 11.)

Luonnollisesti testit ensin -termi käytännössä tarkoittaa, että testit kirjoitetaan ohjelmalle ensin. Kuten kuvassa 8 on esitetty, tarkoitus on määritellä ohjelmalle ja sen tekijälle ensin, kuinka toiminto toimii ja vasta sitten tehdä toivottu toteutus.



Kuva 8. Testit ensin -prosessin kulku.

Ajatellaan sovelluskehitystä käytännössä. Projektipäällikkö ja asiakas tapaavat ja keskustelevat maallikkotermein, mitä tulisi kehittää. Tämä suulliseesti käyty keskustelu muodostaa määrityksen projektille, joka lopulta viedään kehittäjälle. Varsinaisen toteutuksen kannalta on paljon järkevämpää luoda testit ensin ohjelmalle. Ohjelma ensin ymmärtää miten toiminnon tulisi toimia, mutta samalla kehittäjä saa paremman ymmärryksen toiminnon laadusta. Tämän jälkeen hän ajaa testit annetuilla arvoilla, kunnes toiminto toimii. Sen jälkeen voidaan toteuttaa itse toiminto, kunnes se toimii ja viimeistellä projekti ajamalla testit vielä kerran. (Garcia 2018, 10.) Tässä toteutuksessa yhdistyvät, sekä valkoisen laatikon että mustan laatikon testaus.

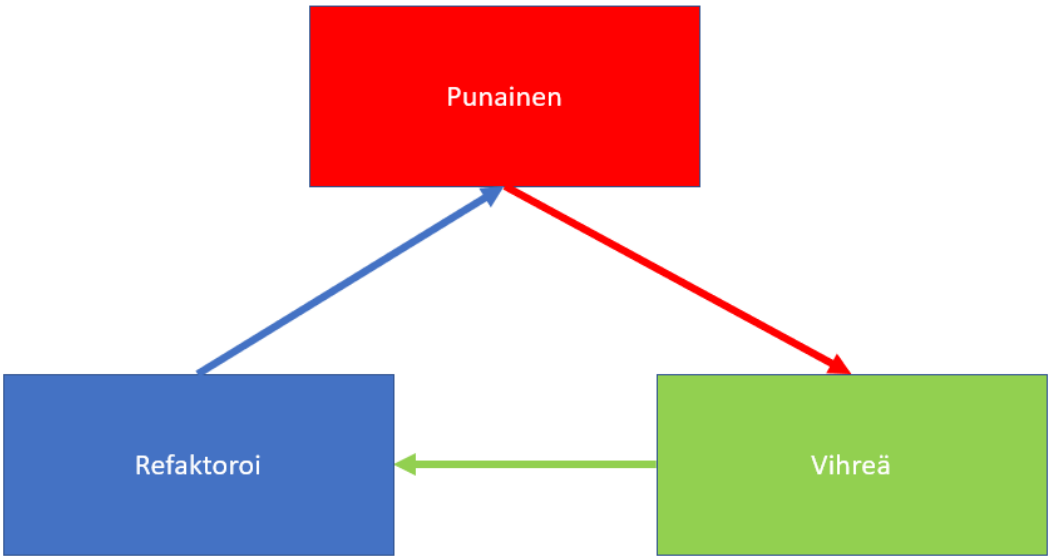
Suoritusten järjestys on erityisen tärkeä, kun puhutaan testit ensin -ohjelmointitavasta. Olemassa olevalle koodille on vaikea toteuttaa testejä. Jo olemassa olevan sovellukseen määritellyt testit ovat puolueellisia. Niillä on taipumus vahvistaa, mitä koodi tekee, ottamatta huomioon täyttyvätkö asiakkaan odotukset ja toimiiko koodi odotetusti. (Garcia 2018, 14.)

Kun testit kirjoitetaan ennen koodia, on koodin testaaminen kattavaa ja saadaan suurempi luottamus, että sovellus toimii, kuten pitää. Testit ensin -ohjelmointimetodilla voi tulla silti esiin ongelmia (Garcia 2018. 17.) Tämä myös säästää aikaa ja kustannuksia jatkokehityksiä ajatellen. Ominaisuudella on testit olemassa taustalla ja tästä osasta muiden kehittäjien on myös helppo päätellä, miten toiminnon tulee toimia. Toinen vaihtoehto olisi tutkia varsinaisen toteutuksen koodia ja pyrkiä sitä kautta päättelemään, miten toiminnon tulisi toimia. Tämä tapa on huomattavasti vaikeampi ja vie enemmän aikaa. Metodi nopeuttaa tuotantoon vientiä, mahdollistaa helpomman uudelleenkäsittelyn, auttaa luomaan paremman suunnittelun ja edistää kytkentää (Garcia 2018, 8).

Seniori -tason, sekä muut kokeneet kehittäjät tietävät panostaa ohjelmiston suunnittelemiseen riippumatta siitä käyttävätkö he testit ensin -ohjelmointitapaa. Testit ensin -metodi kannustaa kaikkia kehittäjiä, jopa aloittelevia, seuraamaan tiettyjä käytäntöjä, mikä tekee heidän koodistaan siistimpää ja luettavampaa. (Garcia 2018, 114.)

3.4.1 Punainen-vihreä-refaktoroi ohjelmointiprosessi

Punainen-vihreä-refaktoroi ohjelmointiprosessi on yksiselitteinen tapa kuvata testit ensin -ohjelmointitapaa. Kirjoita testit ensin ja kun kaikki testit läpäistään, suorita käyttöönotto. Jonka jälkeen vielä suorita testit vielä kerran. Tarkastellaan tarkemmin, miten käytännössä ohjelmointi tapahtuu. Prosessin kulku on helppo nähdä kuvan 9 esittämän kuvion kautta (testit ensin -prosessissa). Tarkastellaan kyseisen ohjelmointiprosessin kulkua alla esitetyn kuvion kautta.



Kuva 9. Punainen-vihreä-refaktoroi prosessin kulku.

Kaavion tulkitaan alkavan aina punaisesta laatikosta. Kehittäjä on aloittanut uuden kehitystehtävän. Joko täysin uuden toiminnallisuuden luominen, tai vanhan olemassa olevan laajentaminen. Punaisessa tilassa testien suorittaminen epäonnistuu. Koodin odotuksien ja varsinaisen toteutuksen välillä on eroja. (Garcia 2018, 59.)

Testin on paitsi epäonnistuttava, on sen myös epäonnistuttava tietystä syystä. Tässä vaiheessa olemme edelleen punaisessa vaiheessa. Testit suoritettiin ja viimeinen epäonnistui. (Garcia 2018, 59.) Kuten kuvan 1 esiteltiin testauksen tulisi seurata tiettyä prosessia ja päätyä oletettuun tilaan.

Vihreään tilaan päästään, kun ohjelman logiikka on kirjoitettu. Kirjoituksen jälkeen kaikki testit on suoritettu onnistuneesti. Kaikki testit ovat läpäisseet ja sovellus käyttäytyy kuten odotamme sen käyttäytyvän (Garcia 2018, 60).

Refaktoroi on viimeinen vaihe. Sovellus toimii, kuten pitääkin, mutta tekninen toteutus on vaikea lukuista ja epäselvää. Refaktorointi on vapaaehtoinen vaihe, toteutuksen laadusta riippuen tämä vaihe voidaan ohittaa (Garcia 2018, 60).

Kuviosta 8 on hyvä huomata kehämäinen rakenne. Refaktorointi -vaiheen jälkeen testit tulisi suorittaa uudestaan. Mikäli testit palauttavat virheen, palataan punaiseen tilaan. Prosessin läpi käynti voi kestää muutamasta sekunnista muutamiin minuutteihin (Garcia 2018, 60).

3.4.2 Vaikutus luettavuuteen ja dokumentaatioon

Erittäin yleinen vitsi kehittäjien kesken: “Hyvä koodi dokumentoi itsensä”. Testit ensin -metodilla on mahdollista päästä askelta lähemmäs tätä itseään dokumentoivaa koodia. Dokumentointi on edelleen tärkeää. Sekä yksinkertaiset, että monimutkaiset toiminnot pitäisi dokumentoida koodin ulkopuolelle. Useimmissa tapauksissa on paljon helpompaa selvittää, mitä koodi tekee, katsomalla testejä kuin itse toteutusta. Garcian (2018) mukaan menetelmien tarkoitus, sekä toivottu toiminnallisuus voidaan päätellä tutkimalla testejä, jotka on sidottu toimintoon.

Testien kautta tehtävä dokumentaatio on monessa suhteessa parempi, kuin vanhanaikainen tekstin muodossa tehty dokumentaatio. Suurin ongelma perinteisessä dokumentoinnissa on, että se on vaikea pitää ajan tasalla pitkällä aikavälillä. Kehitystä tehdään iteroiden, joten muutoksia ja uusia toimintoja tulee ja ajan myötä dokumentaatio viittaa vanhaan tietoon. (Garcia 2018, 15.)

Koska testaaminen tapahtuu valkoisen laatikon ja mustan laatikon muodossa, on vanhanaikaista dokumentaatiota myös ylläpidettävä. Valkoisen laatikon testaaja pystyy hyödyntämään teknistä dokumentaatiota ja tulkitsemaan tarkoitetun toiminnallisuuden testien kautta. Mustan laatikon testaajille on tärkeää ajantasainen ja selkeä dokumentaatio (Garcia 2018, 16).

Kehittäjä tekee työnsä pienissä osissa ja aina päästyään vihreään tilaan, hänen tulisi kysyä itseltään, onko tekninen toteutus selkeä. Punainen-vihreä-refaktoroi -metodi suosii ominaisuuksien kehittämistä pienissä osissa, ottamalla käyttöön yhden testin kerralla, joka ensin epäonnistuu (Garcia 2018, 114).

3.5 Yksikkötestaaminen

Kirjoitetut kirjat sisältävät tekstiä ja teksti jaetaan loogisiin pienempiin kappaleisiin, jotta lukija ja kirjoittaja ymmärtää tarinan taustalla. Yksikkötestaaminen on sama asia, siinä otetaan yksittäinen osio ohjelmasta, tai joukko osioita, jotka muodostavat yksikön (Holcombe 2008, 216). Holcombe (2008) korostaa, että yksikkötestaamisen onnistuminen vaatii tarkkaan määritellyt yksiköt.

Yksikkötestaaminen on suositeltava tapa toimia ja tälle käytännölle voidaan listata kolme syytä. Ensimmäiseksi se helpottaa testien hallintaa, sillä testaaminen keskittyy pienempiin osiin ohjel-

maa. Toiseksi tekniikka helpottaa vian paikallistamista ja viimeisenä se mahdollistaa rinnakkaisajon. Rinnakkaisajossa testejä voidaan suorittaa samaan aikaan, mikä tehostaa koko prosessin suoritusta. (Myers ym 2008,85.)

Testaamisen päämärässä mainittiin, että onnistunut testitapaus on sellainen, joka löytää virheen. Tämä on yksikkötestaamisen tarkoitus, tarkoitus on todistaa täyttää moduuli sille annetut määritykset oikein (Myers ym 2008, 85).

Yksikkötestaaminen on vahvasti valkoiseen laatikkoon orientoitunut testauksen muoto. Yksi syy on, että sillä testataan laajempi kokonaisuuksia, kuten kokonaisia ohjelmia. Toinen syy ollen, että prosessien tarkoitus on nimenomaan löytää virheitä. (Myers ym 2008, 86.)

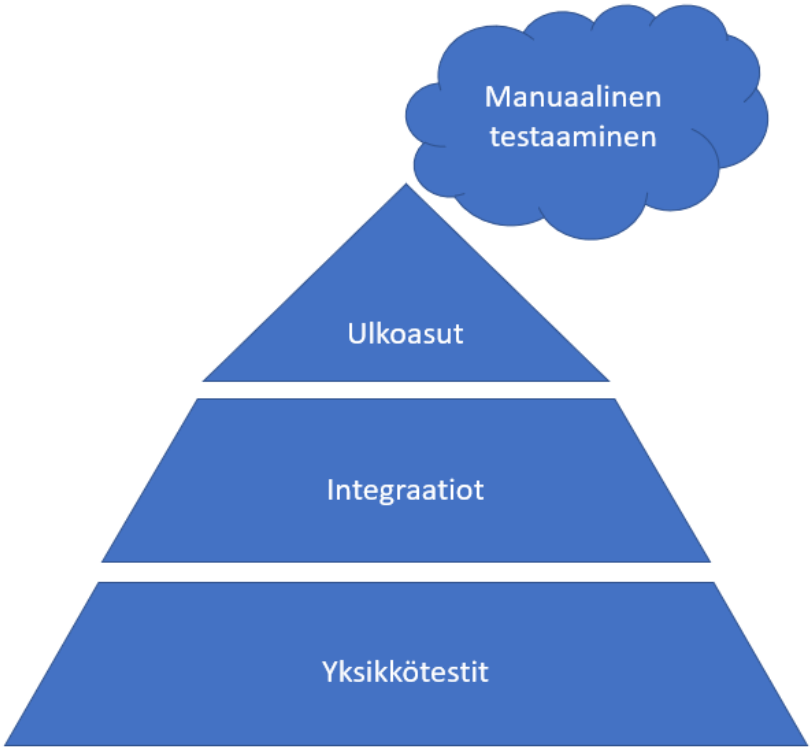
Yksikkötestaaminen kohdistuu pienempiin yksikköihin, ja nämä yksiköt muodostavat sitten suuremman kokonaisuuden. Huomion arvoista yksikkötestaamisesta on, yksikkötestaaminen on lisäys muihin testaustyyppeihin. Sen sijaan yksikkötestaus vähentää muun tyyppisten testien määrää. (Garcia 2018. 86.)

Suurin vahvuus yksikkötestaamisessa on sen kyky pakottaa kirjoittamaan koodia pienemmissä osissa. Yksikkötestaaminen on luoteeltaan helpompaa ja nopeampaa kirjoittaa, kuin muut testaamisen tyypit, mikä vähentää kustannuksia ja siihen käytettävää aikaa. Koska niiden kirjoittaminen ja suorittamiseen kuluva aika on vähäinen, mahdollistaa se ongelmien paikallistamisen nopeasti. (Garcia 2018, 86.)

Yleinen kuvio, jota käytetään visualisoimaan testausta, on pyramidi (kuva 10). Pyramidi pysyy pystyssä, vain koska alempi taso on paksumpi, kuin ylempi taso ja paksuuden määrittää yksikkötestien määrä (Garcia 2018, 88).

Yksikkötestaaminen ja testit ensin -ohjelmointitapa ovat itsenäiset entiteettinsä. Tavallisesti tarkasteltuna ne ovat melkein toistensa vastakohtia, sillä yksikkötestit kirjoitetaan käyttöönoton jälkeen ja testit ensin -ohjelmointitavassa testit kirjoitetaan ennen käyttöönottoa (Garcia 2018, 88.)

Nämä kaksi tapaa täydentävät toisiaan, samalla tavalla kuin musta laatikko täydentää valkoista laatikkoa. Testit ensin -ohjelmointitavan tarkoitus on lähestyä työn suunnittelua. Yksikkötesteillä varustetun testin ensin -tavan tapauksessa tämä soveltamisala on pieni osa ohjelmaa. Lisäksi yksikkötestien ohjaamana testit ensin -ohjelmointitapa ohjaa kehittäjää pitämään toteutuksensa mahdollisimman yksinkertaisena. (Garcia 2018, 88.) Yksinkertainen toteutus palaa takaisin laadukkaaseen dokumentaatioon ja ohjelmoijien vitsiin hyvästä koodista ja dokumentaatiosta.



Kuva 10. Testauspyramidi.

Yksikkötestaaminen ilman testit ensin -ohjelmointia testaa vain olemassa olevaa koodia (Garcia 2018, 88). Tämä sama ongelma, mikä valkoisella laatikolla on. Pystytään vain vastaamaan tekniseen toteutukseen. Testit ensin -tapa pakottaa meidät miettimään projektin vaatimuksia ja suunnitelmiamme, kirjoittamaan puhdasta koodia, joka toimii, luomaan suoritettavia testivaatimuksia ja refaktoroimaan turvallisesti ja usein (Garcia 2018, 89).

3.6 End-to-end testaaminen

Käsitteelle on vaikea keksiä suomennosta, joka säilyttäisi merkityksensä, päästä päähään testaaminen, on liian monitulkintainen. End-to-end termillä viitataan, koko ohjelman suorituksen testaamiseen aivan alusta loppuun (BrowserStack 2020).

Palataan takaisin kuvassa 11 esitettyyn testauspyramidiin. Voidaan kuvitella alaspäin menevä nuoli pyramidin vasemmalle puolelle ja ylöspäin menevä nuoli pyramidin oikealle puolelle. Näin koko ohjelma suoritetaan sen täydellä mitalla. Ulkoasut tasolta lähtee pyyntö, mikäli pyyntö vaatii integraation toisen järjestelmään ohjelma suorittaa sen ja lopulta ohjelma suorittaa pohjatason.

Pohjataso palauttaa tiedon integraatiotasolle, mikäli sitä tarvittiin ja lopulta päädyttään takaisin ulkoasuihin.

Garcia (2018) iteroi rekisteröinti lomakkeen toiminnallisuutta ja kuinka näiden testaaminen tulisi toteuttaa eri tavalla pyramidin (kuvio 9) alimmalla tasolla ja ylimmällä tasolla. Kattava testaus alustan tasolla on riittävä alimmalle tasolle, ylimmällä tasolla testiksi riittää, että lomake lähetetään oikeaan paikkaan. Tätä samaa logiikkaa on helppo soveltaa verkkokaupan toiminnallisuuksiin, joista yksi on rekisteröityminen.

End-to-end testaamisen automatisaatio on haastavaa. Sillä nämä toiminnot on sidottu ulkoasuihin, jotka suunnitellaan ihmisten käyttöön. Ihmisten käyttäytymistä on vaikea ennakoida tavalla, joka kattaisi kaikki mahdolliset tavat käyttää ohjelmaa. Kuten Myers (2011) puhui ohjelmistoista ja kuinka niiden käyttötarkoituksia voi olla satoja. Syy tähän on käyttäjässä. On naurettavaa olettaa, jokainen käyttäjä käyttäisi sovellusta samalla tavalla. Automaation kautta ajettu testi suorittaa itsensä aina samalla tavalla.

Automaation ideana on määritellä tapa, jolla käytetään sovellusta tietyillä parametreillä. End-to-end testaaminen vaatii inhimillistä väliintuloa, vaikka se olisi vain graafisen esityksen laadun arviointia ja annettujen teknisten vaatimusten täyttymisen arviointia. (Holcombe 2008, 232.)

4 Verkkokauppa ja sen testaus

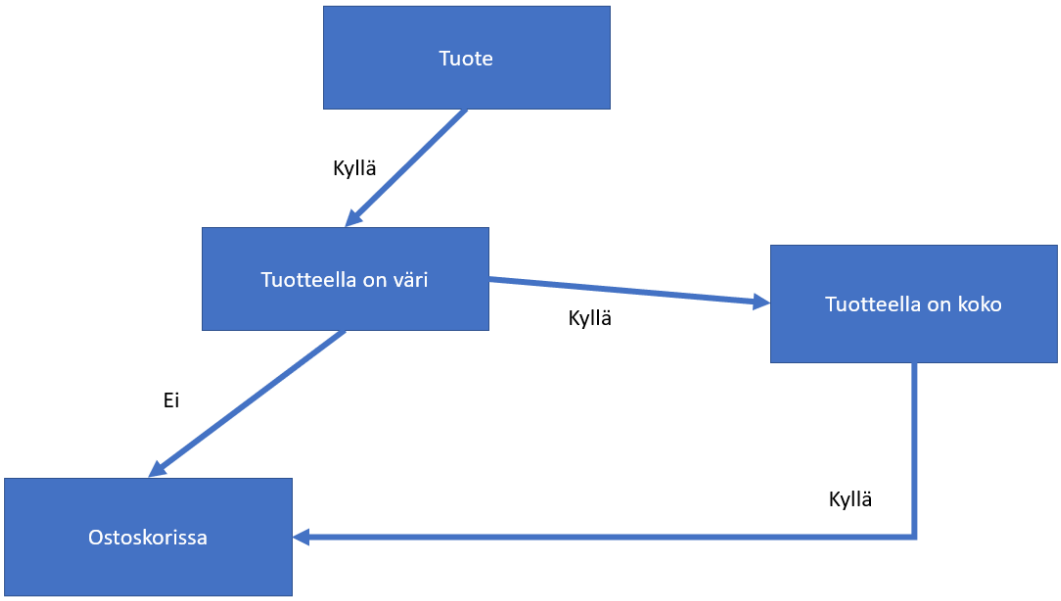
Verkkokauppa käsite on prosessoitu teoreettisella tasolla. Myös testaamista on lähestytty käsitteen kautta ja käsite on liitetty automaation. Tarkoituksena on tuoda esitelty teoria lähemmäs käytäntöä ja rajata siitä tarkemmin osa-alueet, jotka ovat oleellisia opinnäytetyölle.

4.1 Verkkokauppa toteutuksen kuvaus

Kuvan 2 kohdalla esiteltiin verkkokauppa ja sen tasot, joilla toimintaa tapahtuu. Opinnäytetyön tutkimuskysymyksiin ja tarkoitukseen peilaten tarvitsee ainoastaan keskittyä ylimpään tasoon. Käyttöliittymä ja kanavat tasoon. Testauspyramidista puhutettaessa viitataan ainoastaan sen korkeimpaan tasoon, sekä pyramidin päällä leijuvaan pilveen.

Testauksen suunnittelua varten tulee ottaa huomioon kaikki prosessit. On kuitenkin muistettava testauspyramidin rakenne (kuva 9). Testien suunnittelun tulee ottaa huomioon testit alemmalla tasolla, tällöin vältytään testaamasta samaa asiaa monta kertaa. Ulkoasu tasolla on järkevämpää lisätä testi, joka testaa lomakkeen, jonka kautta tuote voidaan lisätä ostoskoriin. Ovatko kaikki pakolliset kentät olemassa käytössä olevassa näkymässä?

Kuvion esittämällä tuotteella (kuva 11) voi olla väri- tai koko-ominaisuus. Kaavion koko räjähtäisi eksponentiaalisesti, mikäli tuotteella olisi vielä kolmas ominaisuus, esimerkiksi rinnanympärys. On erittäin vaikeaa ja aikaa syövää kirjoittaa testi, joka ottaa huomioon kaikki mahdolliset toiminnot ja toiminnot toiminnon sisällä. Joten on tärkeää tunnistaa ne osa-alueet, joihin halutaan tarkastella tarkemmin.



Kuva 11. Karkea esimerkki, testi on suoritettu, kun jokainen yksikkö suoritetaan vähintään kerran

4.2 Verkkokauppa ja opinnäytetyö

Tämän opinnäyteyön tarkoitus on pohtia mahdollisuutta testaamisen automatisoinnille nykyisessä organisaatiossa ja sen kehitysympäristössä. Testaamista keskitytään tarkastelemaan ketterän kehityksen sisällä.

Tämän opinnäytetyön kannalta ja käytännön toteutuksen, johon tämä työ johtaa, tarkoitus on automatisoida testejä. Tämä luonnollisesti vähentäisi aikaa, jonka kehittäjä tarvitsee tehdäkseen työnsä. Tarkoitus on myös vähentää bugien määrä ja vaikeuttaa uusien bugien syntymistä, sekä parantaa virheen hallintaa.

Kun ohjelmalle annettaan määrityksiä on luonnollista keskittyä tilanteisiin, joissa prosessi onnistuu. Mutta on otettava huomioon tilanteet, joissa voi epäonnistua. Palataan esitettyihin määritelmiin bugille, avainsana näissä määritelmissä on poikkeavuus.

Esitetyssä verkkokauppa esimerkissä kyse on selkeästi bugista, mikäli odotettua lopputulos jää saavuttamatta. Mutta mikäli ohjelma kertoo käyttäjälle, miksi toivottu jäi saavuttamatta, olisi kyse virheestä. Tai paremmin sanottuna hallitusta virheestä.

5 Automaatiotestien määrittely ja arviointi verkkokauppakehityksessä

Tässä luvussa kuvataan kehittämistoiminnan ja tutkimuksen kohdetta. Aluksi tarkastellaan opinnäytetyön toimeksiantajaa. Kiinnostuksen kohteena on tapa, jolla projekteja suoritetaan organisaation sisällä. Projektivaiheita on paljon (ns. projektiputki on pitkä). Opinnäytetyön fokus on vaiheessa, jossa tehtävä tulee kehittäjän työlistalle. Lisäksi kuvataan opinnäytetyön tekijän rooli organisaatiossa ja esitellään kehittämistehtävän vaatimukset ja tavoitteet, mikä toimii opinnäytetyön tutkimusmenetelmien perusteluina. Lopuksi esitellään tutkimusmenetelmät ja kehittämistehtävä, jonka tavoitteena on tuottaa tekniset määritykset ja aika-arvio varsinaiselle työlle.

Toimeksiantaja on suomalainen yksityisellä puolella toimiva yritys, LianaTechnologies. Loppuasiakkaille pyritään tarjoamaan viestintäratkaisuja organisaation omien tuotteiden kautta. Organisaation sisällä on useita osastoja ja tuotteita. Tämä kehittämistehtävä rajataan verkkokauppaosastoon ja verkkokauppa-alustaan.

LianaTechnologies organisaatio rakentuu useista osastoista, joihin muun muassa kuuluu myynti-, kehitys- ja tukitiimi. Kehitystiimi on oleellisin tämän työn kannalta, sillä se työskentelee verkkokaupparatkaisujen parissa. Tämän opinnäytetyön tulokset hyödyttävät kehitystiimiä.

Projektien vetoon yrityksessä käytetään niin kutsuttua 70/30 sääntöä. Tämä tarkoittaa 70 % työhön käytetystä ajatusta tulisi käyttää määrittelyyn, joten varsinaiselle työlle jää 30 % täydestä ajasta. Luonnollisesti tämä on vain ajatusmalli, joka auttaa hahmottamaan työ ajan käyttöä. Tärkeää on kuitenkin huomioida, että kun tehtäväpyyntö tulee kehitysjonoon, sen määrittelyyn on käytetty paljon aikaa. Tällöin kehittäjällä mahdollisimman selkeä tehtävä edessään.

Opinnäytetyöntekijä on työskennellyt organisaatiossa vuodesta 2016 lähtien. Tätä edelsi puolen vuoden harjoittelujakso. Ennen nykyistä asemaa kirjoittaja on myös työskennellyt sähköpostimarkkinointi ja verkkosivujen kehitysprojektien parissa. Tällä hetkellä kirjoittaja kuuluu verkkokauppa -osastolle. Jokapäiväinen työskentely tapahtuu verkkokauppa-alustan parissa Full-Stack Developerin roolissa. Tämä rooli antaa näkemyksen siihen, ”mitä konepellin alla tapahtuu”. Mutta myös siihen, miten tuotetta käytetään rakentamaan loppuasiakkaalle tarkoitettu ja personoitu verkkokaupparatkaisu.

5.1 Kehittämistehtävä

Kehittämistehtävässä tutkitaan ja määritellään, kuinka automaatiotestaus voidaan toteuttaa ja ottaa käyttöön verkkokauppa-alustan ulkoasujen kehityspuolella. Kaikki nämä kehitystehtävät ovat asiakasprojekteja, mutta asiakasprojektit voivat vaatia myös itse verkkokauppa-alustan kehitystä.

5.1.1 Yrityksen toimintamalli

Jokaisella työntekijällä on tietty määrä tunteja käytössään yhden sprintin aikana. Tämä on kohdeyrityksessä kymmenen henkilötyöpäivää. Joten on erittäin tärkeää, että jokaiseen tehtävään, joka nostetaan sprintille, on liitetty aika-arvio. Aika-arviolla viitataan arvioituun aikaan, jonka tehtävän suorittaminen vaatii.

Automatisoitujen testien käyttöönotto on ollut pitkään toiveissa, käyttäjätarinalta on puuttunut aika-arvio, joten käyttäjätarinan nostaminen sprintille on vaikeaa. Kun aloitetaan uuden kehittäminen, työ alkaa tehtävänannolla, joka on käytännössä ns. työtiketti aika-arvioineen. Tämän kehittämistehtävän tulos johtaa uuteen työtikettiin.

Aika-arvioita tekeminen on jäänyt toteuttamatta, koska tehtävän luonne on sisäistä kehitystä. Tarkoittaen, että on puutunut asiakasprojekti, joka vaatisi tämän kaltaisen toiminnallisuuden, joten työn tärkeysaste on pysynyt matalana. Mikäli tälle tehtävälle olisi ollut asiakastarvetta, olisi varsinainen kehitys aloitettu aikaisemmin.

Organisaatiossa työskennellään scrumban toimintamallin mukaan. Sprintin kesto tiimissä, jossa työskentely tapahtuu, on kymmenen työpäivää. Juuri tästä syystä on erittäin tärkeää, että käyttäjätarinalla ja siihen liitetyillä tehtävillä on mahdollisimman tarkka aika-arvio vaaditusta työnmäärästä. Sprintin suunnittelu voidaan tällöin tehdä järkevästi.

5.1.2 Kehittämistehtävän rajaukset

Varsinainen automaatiotestauksen toteutus on rajattu tämän opinnäytetyön ulkopuolelle. Tavoitteena on määritellä vankka, ensimmäinen, tuotantoon sopiva versio projektille. Tätä versiota tulee olla mahdollista laajentaa tulevaisuudessa, kun sille nähdään tarvetta. Automatisoitujen

testien tulee ainoastaan tarkastella, miten verkkokauppa-alustan tekninen puoli toimii. Tästä kehittämistehtävästä rajataan pois verkkokaupan latausnopeus ja visuaalisen ilmeen testaaminen.

Kehittämisosiossa kerätään tietoa katselmoimalla nykyistä ympäristöä, jonka päälle projektit rakennetaan. Asemani ansiosta tämä ympäristö on jo tuttu, mutta havainnointi tapahtuu nyt eri silmin. Tämä stimuloi ajatuksia, siitä kuinka nykyinen ympäristö ja nykyiset prosessit toimivat ja kuinka sopivat yhteen tämän uuden idean kanssa.

Myös tiedonhaku ja siten useiden dokumentaatioiden lukeminen on iso osa. Tarkoitus on löytää sopiva työkalu, jonka kautta testi tapaukset ajetaan. Näillä työkaluilla on omat dokumentaationsa, jotka kertovat miten niitä käytetään. Näistä saa kuvan, miten tekninen toteutus tulee tapahtumaan.

5.2 Tutkimusstrategia ja tutkimus- ja kehittämismenetelmät

Strategiaa valittaessa tutkimukselle tulee tutkijan pohtia omaa asemaansa ja kohdettaan, sekä vertailla näitä kohteita kysymyksiin:

- Onko tavoitteena selittää tutkimuskohdetta vai kuvailla sitä?
- Millaiseen rooliin tutkija haluaa asettua tutkimuksessa. Haluaako hän olla aktiivinen tiedonkerääjä, minkälaista vuorovaikutusta hän haluaa tutkimuskohteensa kanssa, vai onko tarkoitus pysytellä tutkimuskohteen ulkopuolella?
- Onko tutkimuskohteen tarkoitus kuvata todellisuutta, vai tarjota yksi mahdollinen tulkinta?
- Onko mahdollista pyrkiä täydelliseen objektiivisuuteen?
- Onko tutkijalle tärkeää, että tutkimus on toistettavissa?

(Juuti & Puusa 2020.)

Tämän opinnäytetyön tutkimusstrategiaksi aiemmin esitettyjen pohdintojen perusteella on valittu tapaustutkimus. Olennainen osa tapaustutkimusta on, että tutkimus mielletään, sekä empiirisenä että teoreettisena (Juuti ym 2020). Tutkimusote, jonka pohjalta strategia suoritetaan, on

konstruktiivinen tutkimusote. Tarkoituksena on hankkia ajankohtainen kuva nykytilanteesta, syventyä teoriaan sen ympärillä ja laatia ratkaisuehdotus aikaisempien vaiheiden pohjalta.

Tapaustutkimus on syvällinen tutkimus useista näkökulmista tietyn projektin, politiikan, instituution, ohjelman tai järjestelmän monimutkaisuuteen ja ainutlaatuisuuteen tosielämän yhteydessä. Se perustuu tutkimukseen, sisältää erilaisia menetelmiä ja on näyttöön perustuvaa. (Simon 2009, 21.) Tämä määritys on yleisluontoinen, se pyrkii kattamaan kaikki mahdolliset tulevat ja olevat käyttötarkoitukset. Tapaustutkimusta käsitteenä voidaan myös määritellä IT alan näkökulmasta. Tässä kontekstissa tapaustutkimus on empiirinen tutkimus, joka hyödyntää useita todistuslähteitä nykyajan ohjelmistotekniikassa yhden, tai usean ilmiön tutkimiseksi sen yhteydessä tosielämään, varsinkin kun ilmiön ja kontekstin välistä rajaa on epäselvä (Host, Rainer & Runeson 2012, 12).

Tapaustutkimuksella on mahdollista luoda uutta teoriaa, tarkentaa olemassa olevaa teoriaa tai testata olemassa olevaa teoriaa (Juuti ym 2020). Tapaustutkimuksella on Simonin (2009) mukaan kaksi erilaista tapaa lähestyä tutkimustyötä; teoriajohtoinen ja teoriaa luova.

Teoriajohtoisella tarkoitetaan tapaustutkimista, tai jopa esimerkin antamista tietyn teoreettisen näkökulman kautta tai, kuten ohjelmassa arviointitapaustutkimus, jossa tutkitaan aluksi, mikä on ohjelman teoria, mitä se pyrkii saavuttamaan arvioinnin keskittämiseksi ja suunnittelemiseksi (Simon 2009, 21–22).

Tutkimusote on tämän työn puitteissa kvalitatiivinen. Tiedonhaulla kerätään tietoa eri toteutusvaihtoehdoista. Tähän aineistoon tulee perehtyä ja dokumentoida, jotta sen perusteella voidaan tehdä toteutus organisaatiolle. Toisin sanoen perehdytään arkistotietoihin. Käsitteenä tämä tarkoittaa arkistossa oleviin tietoihin perehtymistä (Host ym 2012, 57).

Tapaustutkimuksen aineistonhankinta menetelmissä havainnointi jää yleensä muiden menetelmien varjoon. Havainnointi menetelmä tarkoittaa systemaattista tiedon keräystä ja tieteelliseen työskentelyyn suuntautuvaa toimintaa, jossa kohdennamme aistejamme tarkemmin, kuin arjessa. (Juuti ym 2020.) Havainnointi on menetelmä, jota voi harjoittaa yhden tai useamman tutkijan toimesta (Juuti ym 2020).

Havainnointi menetelmässä on tärkeää pohtia etukäteen, mitä havainnoidaan, miten havainnoidaan ja mihin olisi hyvä kiinnittää huomiota. (Juuti ym 2020). Aineiston rajauksen on hyvä olla löyhä tutkimuksen alussa (Juuti ym 2020).

Tehokas lisäys havainnointi metodille on soveltaa "ajattele ääneen" -metodia, jossa tutkija kysyy toistuvasti kysymyksiä, kuten "Mikä on strategiasi?" ja "Mitä ajattelet?" (Host ym 2012, 56). Tapaustutkimuksessa on tärkeää ottaa huomioon eri roolien näkökulmat ja tutkia eroja esimerkiksi eri projektien ja tuotteiden välillä. (Host ym 2012, 49.)

5.2.1 Tutkimukseen valitut menetelmät

Tutkimusongelmana on pohtia teknistä toteutusta ja vaatimuksia automaatiotestaamiselle organisaation verkkokauppa kehitysryhmässä. Tutkimusongelmasta saadaan tutkimuskysymykset:

- Miten olemassa olevaan ympäristöön voidaan automaattinen testausympäristö?
- Mitä testejä on mahdollista ja järkevää testata automaation kautta?
- Kuinka kauan automaattisen testausympäristön toteuttamiseen menee suunnilleen?

Tutkija toimii tutkittavan ilmiön parissa. Tutkittavaan aihealueeseen on tarpeen perehtyä syvällisemmin organisaatiossa. Tutkimuksessa aineistonhankinta on toteutettu tiedonhankinnalla ja havainnoimalla.

Kerätyn tiedon pohjalta kasataan ehdotelmat, joiden pohjalta voidaan pohtia projektin suuntaan. Näiden ehdotelmien tueksi kasataan SWOT analyysit. SWOT on luonteeltaan yhteen vetävä synteesinomainen analyysi. Työkalun on tarkoitus tuottaa selkeä kokonaiskuva yrityksen tilanteesta strategisten valintojen tueksi. Hyvä SWOT-analyysi vaatii tuekseen lukuisia yrityksen resursseihin ja toimintaympäristöön liittyviä osa-analyysejä: jos organisaatiota ja sen toimintaympäristöltä puuttuu syvällinen tuntemus aiheesta, analyysin toteuttaminen on vaikeaa. (Vuorinen 2013, 88.)

6 Tutkimuksen toteutus ja tulokset

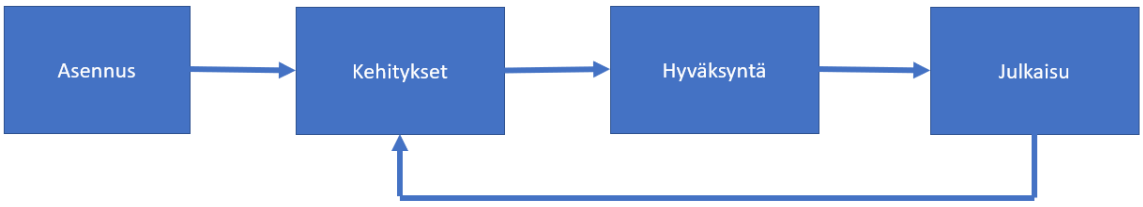
Tutkimus lähti liikkeelle lähtötilanteen selvittämisestä, joka muodostaa opinnäytetyön tutkimusosion. Tutkimusosiosta saatua tietoa verrattiin organisaation tilanteeseen ja nämä kaksi maailmaa pyrittiin sovittamaan.

Tutkimusstrategiana oli tapaustutkimus, joka toteutettiin havainnoinnin keinoin. Tässä opinnäytetyössä selvitetään, kuinka tämä kehitys voidaan toteuttaa. Selvityksessä tehdään laajasti tiedonhakua eri vaihtoehdoista perehtyen kirjallisuuteen ja toteutusvaihtoehtoihin. Tuotoksena on työohje aika-arvioineen yrityksen projektinhallintaan. Tämän perusteella yrityksessä voidaan arvioida tehtävälle sopiva tekijä. Itse tekijä sitten toteuttaa kehityksen, mutta tämä vaihe on rajattu pois opinnäytetyöstä.

6.1 Lähtötilanne

Ajatellaan tilannetta, jossa uusi asiakasprojekti aloitetaan. Kehittäjä saa tästä työtiketin, joka sisältää tarvittavat määritykset työtiketin suorittamiseen. Näitä voi myös ajatella tutkimuskysymyksinä ja työn suorittaminen on vastaus näihin kysymyksiin.

Kehittäjä ottaa pohjakaupan ja asettaa sen asiakkaan kaupaksi. Tämän jälkeen hän tekee vaaditut kehitystyöt. Tikettiin liittyvät tehtäviin voi kuulua laajempia projektikohtaisia räätälöintejä, nämä voivat vaatia lisäyksiä tai muutoksia pohjakaupan toiminnallisuuksiin. Tämä prosessi kokonaisuudessaan on esitetty kuviossa 12.



Kuva 12. Kehityksen prosessi.

Esimerkiksi pohjakauppa sallii käyttäjän lisätä niin monta tuote ostoskoriin, kuin hän haluaa. Mutta asiakkaan määrityksien mukaan on mahdollista, että käyttäjällä saa olla vain yksi tuote

kerralla ostoskorissa. Määritykset yleensä vaativat pohjakaupan graafisen ilmeen mukauttamista vastaamaan asiakkaan brändiä.

Pohjakauppa on oma projektinsa, jota kehitetään koko ajan. Kun pohjakauppa kopioidaan asiakkaan omaksi kaupaksi, on mahdotonta suoraan tietää sisältääkö pohjakauppa bugeja. Näiden olemassaoloa täytyy tarkkailla koko ajan. Testaaminen on prosessi, joka tapahtuu koko ajan projektin taustalla, joten näihin tilanteisiin on joka tapauksessa varauduttu.

On mahdollista, kuitenkin että bugi pääse yllättämään täysin kehittäjän, jolloin mennään niin sanottuun reagointitilaan. Näissä tilanteissa on aina pohdittava, onko bugi räätälöinnin tulos, vai pystyykö sen toistamaan pohjakaupassa samoissa oloissa.

6.2 Kehittäminen ja tulokset

Opinnäytetyön kehittämistehtävän tavoitteena on tuottaa kehitysehdotus automaatiotestien käyttöön verkkokauppakehityksessä. Konseptuaalisen idean rakentaminen alkoi tarkastelemalla olemassa olevaa pohjakauppaa. Katselmoinnit suuntautuivat teknisiin toteutuksiin, joita pohjakauppa pitää sisällään. Näiden katselmointien ideana oli löytää toimintoja, joihin automaatiotestit voitaisiin mahdollisesti toteuttaa. Katselmoinnit aloitettiin teoriaosuuden jälkeen. Tämän ansiosta oli mahdollista ottaa huomioon nykytilanne ja verrata sitä, kuinka toteutus tulisi tehdä teoreettisella tasolla.

Tarkoitus on ideoida ja arvioida, kuinka ympäristö voidaan toteuttaa. Ympäristöllä suoritettavat testit tarkastellaan toteutetaan myöhemmin. Pohjakaupasta täytyi löytää vähintään yksi toiminto, johon ajatustyö peilataan. Toiminto, jonka valitsin on tuotteen lisääminen ostoskoriin.

Suurin haaste, tämän toiminnon ja minkä tahansa muun toiminnon kanssa, on että kehittäjä voi tehdä mitä tahansa asiakasprojekti kohtaisia muutoksia, mihin tahansa kohtaa ohjelmaa. Käytännössä on mahdollista, että tämä johtaa tilanteisiin, jossa testi ajetaan pohjakaupassa ja se toimii oikein. Sama testi ajetaan asiakkaan kaupassa, projektin vaatimien kehityksien jälkeen ja testi epäonnistuu. Kuitenkin asiakkaan kauppa toimii oikein. Suurin ongelma on, että mikäli näin käy toistuvasti, menettää automaatiotestaus arvoansa ja lopulta automaatio testien suorittaminen jää pois testausprosessista, koska tulokset ovat epäluotettavia. Tämä on hyvä huomioida, kun toteutusta lähdetään viemään eteenpäin.

Ongelmana on, että asiakasprojekti vaatii räätälöintiä olemassa olevaan toimintoon. Ensimmäinen idea ratkaisemaan tämä ongelma olisi dokumentoida ne osat ohjelmasta, mitkä ovat riippuvaisia testeistä. Tällöin kehittäjällä olisi selkeä läpinäkyvyys, mitä osia hän saa muokata ja laajentaa ja mitkä osat ovat niin sanotusti suojeltuja. Tämä olisi nopea ratkaisu, mutta mahdollistaisi laajennuksien ja muokkauksien teon.

Toinen ja samalla työläämpi ratkaisu olisi muuttaa ehdotettu testausprojekti laajemmaksi. Tämä ehdotus käydään läpi seuraavissa kappaleissa. Sitä voisi kutsua verkkokaupan ytimeksi. Idea olisi, että perustettaisiin oma projekti, johon kehitetään kaikki toiminnot, jotka määritellään tärkeiksi. Näille toiminnoille tehdään toteutukset ja testit. Tämä estäisi täysin kehittäjää koskemasta toimintoihin.

6.2.1 Palvelun valinta

Automaatiotestit on ajettava testiympäristössä, jota tarjoaa usea palveluntarjoaja. Tähän opinnäytetyöhön valittiin kolme erilaista testiympäristöä.

- Cypress.io
- Nightwatch
- Puppeteer

Palvelut ovat käyttötarkoitukseltaan samoja ja toimivat samalla tavalla. Palvelut on kuvattu tarkemmin seuraavissa kappaleissa. Myös palvelun valinta prosessi on kuvattu.

Nämä palvelut toimivat ns. ajureina. Testit ovat jotain, jota organisaation täytyy itse määritellä ja kirjoittaa. Ajuri on osa, jonka kautta testi suoritetaan halutussa ympäristössä, selaimessa, ja näkymässä, joka on tietty sivu verkkosivustolta. Palvelu suorittaa testit ja raportoi tulokset.

Jokainen näistä palveluista on varteenotettava vaihtoehto. Näistä palveluja kartoitettiin tiedonhaku vaiheessa. Palveluja on lukuisia ja näistä valikoitiin kolme potentiaalisinta vaihtoehtoa niiden ominaisuuksien ja suositusten perusteella.

Nykyajan verkkoselaimet suorittavat niille kirjoitettua logiikkaa hyvin samalla tavalla. Hyvin todennäköisesti testit palauttavat saman tuloksen, riippumatta minkä selaimen kautta se ajettiin.

Käytin kuitenkin selaintukea pääkriteerinä, kun vertailin palveluja keskenään. Selaintuella tarkoitetaan, mihin selaimiin ajuri pystyy kohdistamaan testin.

Mikäli ympäristö pystyy suorittamaan testin vain valituissa selaimissa, pakottaa tämä kehittäjän työskentelemään näiden valittujen selainten kanssa. On myös todennäköisesti olemassa ääripää tilanteita, joissa tietty toteutettu logiikka toimii Google Chrome selaimessa ja epäonnistuu Mozilla Firefoxilla, joten on hyvä varautua näihin ääripään tilanteisiin.

Toinen vahva syy käyttää selaintukea kriteerinä on sen käytännöllisyys, sillä se on helposti hahmotettava. Toteutuksen ehdotelma on esitettävä tiimin muille jäsenille, joilla on enemmän tai vähemmän teknistä ymmärrystä. Selaintuki on jotain, jota teknisestä osaamista huolimatta on helppo hahmottaa.

Puppeteerin suurin heikkous on, että se pystyy tukemaan ainoastaan Google Chrome verkkoselainta. Tarkastelun pohjalta huomasin, että palveluun ollaan lisäämässä tukea eri verkkoselaimilla, mutta tämä työ on vielä kesken. Tarkastelen palveluja niiden nykyisten toiminallisuuksien valossa. (Taulukko 1.)

Vahvuudet	Heikkoudet
• Yksinkertainen asennus prosessi • Testit voidaan toteuttaa JavaScript ohjelmointikielellä	• Testit voidaan ainoastaan ajaa Google Chrome selaimessa
Mahdollisuudet	Uhat
• Selain tuki laajentumassa	• Perustuu avoimeen lähdekoodin, ajurin ylläpito tapahtuu ulkoisen osapuolen toimesta

Taulukko 1. Puppeteer SWOT.

Cypress on erittäin varteen otettava vaihtoehto tarkastelujen perusteella. Täydellinen selain tuki jää tällä palvelulla myös saavuttamatta, muun muassa Safari oli tuen ulkopuolella. Tämän puutteen kanssa on helpompi elää, kuin puuttuva Google Chrome tai FireFox tuki. Sillä Safari on selain, jota virallisesti voi käyttää vain Applen tuotteissa. Firefox on selain, jota kuka vain voi käyttää, mikäli hän näin haluaa. Suurimman hyödyn tästä palvelusta saa, mikäli kyseessä on varsinainen

sovellus, joka on kehitetty React, Vue, Angular sovelluskehyksen päälle. Pohjakauppa toimii ilman näitä sovelluskehyksiä. (Taulukko 2.)

Cypress:lla oli oma raportointi työkalu. Tämä tarjoaa saman tiedon lopulta, mitä Puppeteer tai Nightwatch mutta sen ulkoasu oli huomattavasti parempi. Parempi ulkoasu tarkoittaa parempaa tulkittavuutta ja luettavuutta. (Taulukko 2.)

Vahvuudet	Heikkoudet
• Yksinkertainen asennus prosessi • Testit voidaan toteuttaa JavaScript ohjelmointikielellä	• Rajallinen selain tuki.
Mahdollisuudet	Uhat
• BrowserStack tuki	• Testit voidaan ohjelmoida JavaScript:lla • Perustuu avoimeen lähdekoodin, ajurin ylläpito tapahtuu ulkoisen osapuolen toimesta.

Taulukko 2. Cypress SWOT.

Parhain vaihtoehto näistä kolmesta on Nightwatch palvelu. Tämä on hyvin samankaltainen, kuin Puppeteer. Nightwatch palvelun kautta on mahdollista suorittaa testejä, millä tahansa selaimella. Luonnollisesti rajoituksia on tässäkin, sillä Safari selaimen tuki rajoittuu ainoastaan Applen tuotteisiin, virallisesti ja sama pätee Microsoftin selaimeen. (Taulukko 3.)

Nightwatch palvelussa sain suoritettua testin kaikissa selainympäristöissä, mukaan lukien Safari ja tavallaan Microsoftin omalla selaimella. Tässä vaiheessa hyvä huomauttaa, että tietokone, jota käytin tutkimukseen on Macbook Pro. Tietokone, jolta puuttuu tämän selaimen tuki. Pystyin kuitenkin ajamaan testin käyttämällä BrowserStack palvelun emulaattoreja. Nightwatch mahdollistaa testien ajamisen omalla koneella, mutta myös BrowserStack emulaattori ympäristössä. (Taulukko 3.)

Vahvuudet	Heikkoudet
• Yksinkertainen asennusprosessi • Testit voidaan toteuttaa JavaScript ohjelmointikielellä • Laaja selaintuki	• Paljon asennustyötä. Selain pitää olla asennettuna koneeseen ja Nightwatch ajuri per selain.
Mahdollisuudet	Uhat
• BrowserStack tuki	• Perustuu avoimeen lähdekoodin, ajurin ylläpito tapahtuu ulkoisen osapuolen toimesta

Taulukko 3. Nightwatch SWOT.

SWOT analyyseissä on tarkoituksella pysytty erillään teknisen puolen vahvuuksista ja heikkouksista. Tähän puoleen perehtyminen vaatisi teknisen toteutuksen aloitus, jotta voidaan verrata, kuinka ne toimivat organisaation ympäristön puitteissa.

Tässä vaiheessa on järkevää huomauttaa, että BrowserStack dokumentaation mukaan (BrowserStack n.d) Cypress pystyy ajamaan testinsä myös BrowserStack palvelun kautta.

BrowserStack tarjoaa mahdollisuuden käyttää heidän verkkosivujen selain emulaattorien kautta. On siis mahdollista, että Windows käyttäjä voi testata toiminallisuuden käyttämällä Safari verkkoselaimen tiettyä versiota. Cypressin yhteyttä BrowserStack palveluun en erikseen testannut, mutta luotan kyseiseen dokumentaation ilman, että erikseen vahvistan heidän väittämänsä.

Käyttöönotto, testien kehittäminen ja testien ajaminen näiden kolmen palvelun välillä tapahtuu hyvin samalla tavalla. Kaikki kolme asennetaan samalla tavalla. Testien kehittäminen tapahtuu samalla tavalla, kun puhutaan Nightwatch ja Puppeteer palveluista, tavallisella JavaScript ohjelmointikielellä. Cypress käyttää JavaScript ohjelmointikieltä pohjanaan, mutta tämä tapahtuu normaalista syntaksista poikkeavalla tavalla.

Taulukko 4 esittää lopullisen yhteenvedon aiheesta. X-akselilla esitetään palvelu ja y-akselilla selain.

	Puppeteer	Cypress	Nightwatch
Chrome	x	x	x
FireFox		x	x
Opera	x	x	x
Edge			x
Safari			x

Taulukko 4. Yhteenveto.

6.2.2 Palvelun käyttöönotto

Palvelun valinnan jälkeen tuli vielä ratkaista, miten se voidaan ottaa käyttöön organisaation ympäristössä. Ongelmana tämä on vaikea ratkaista, sillä ympäristö, pohjakauppa, on jo olemassa, sekä ympäristön tulee mahdollisia asiakaskohtaisia räätälöintejä.

Niin kutsuttua, “ajattele ääneen” -metodia on vahvistamaan luottamusta saatuihin tuloksiin. Näistä saatuja tuloksia on reflektoitu omiin kokemuksiin ja ajatuksiin aiheesta. Teoriakatsauksen aikana saatua tietoa myös käytettiin hyväksi. Myös ajatuksien vertailu ja arviointi nykyisiin prosesseihin oli tärkeää. Tarkoitus on ollut vahvistaa luottamusta saadusta lopputuloksesta.

Näitä ”ajattele ääneen” -hetkiä on myös pidetty erittäin pintapuolisina. Eräänlainen vaara näissä oli, että ne hyvin nopeasti alkavat tuottaa vastauksia teknisiin kysymyksiin. Saatuja tuloksia on tutkittu hetkessä. Tuloksien dokumentointi on jätetty tekemättä, tulokset on tulkittu hetkessä ainoastaan vahvistamaan idean oikeellisuutta.

6.3 Toteutus

Esitellään opinnäytetyön lopullinen tuotos, josta voidaan johtaa vastaukset opinnäytetyön tutkimuskysymyksiin. Tarkastelu on ainoastaan objektiivinen esittely, miten tämä tapahtuu. Sen hyviä

ja huonoja puolia pohditaan pohdintaosuudessa. On myös mahdollista, että toteutukselle löydetään vaihtoehtoinen tapa, mutta esitelty tulos on opinnäytetyön lopputulos. Tämän työn perusteella lopputulos on luoda testit omaan projektiinsa ja suorittaa käyttöönotto projekteissa alimoduulin avulla**.**

Kuten on mainittu, asiakasprojektit kehitetään käyttämällä samaa kauppapohjaa. Pohjaa, johon on kehitetty kaikki toiminnot, joita halutaan tukea. Yksi vaihtoehto olisi tehdä automaatiotestaaminen suoraan osaksi tätä pohjakauppaa. Tässä lähestymisessä on kuitenkin kolme suoraa ongelmaa.

1) Testi tulisi suunnitella ja toteuttaa ennen toiminnon toteutusta, tällöin vältytään tilanteelta, jossa pyritään onnistumaan testissä testaamalla ratkaisua, testaamatta määrityksiä ja kuinka toteutus vastaa niihin. (Garcia 2018, 14). Automaatiotestauksen rakentaminen suoraan pohjakauppaa menisi juuri tätä ajatusmallia vastaan. Toiminnot on jo kehitetty ja nyt niille tulee rakentaa testit. Tämä sama malli todennäköisesti toistuisi uusien kehityksien parissa. Ensin määritellään ja rakennetaan toiminta ja vasta sen jälkeen pohditaan, onko testaaminen automaation kautta tarpeen ja toimitaan pohdintojen perusteella.

2) Toinen syy on, että jo olemassa oleville asiakkaille tämän tuen lisääminen olisi erittäin vaikeaa tällä tavalla. Käytännössä tämä tarkoittaisi, että kehittäjän tulisi asentaa koko ympäristö asiakkaan projektiin ja käsin kopioida testit pohjakaupasta. Sinänsä tämä on mahdollista toteuttaa, mutta tämä olisi erittäin aikaa vievää. Tämä myös vaatisi paljon käsin tehtävää kopiointia, mikä on virhealtis tapa tehdä toteutuksia.

3) Kolmas ongelma tässä lähestymisessä olisi seuraava, tämä on myös painavin syy. Uusi kauppaprojekti aloitetaan, jolloin pohjakauppa, mukaan lukien automaatiotestit, kopioidaan uuteen asennukseen. Kehittäjä tekee kehitystehtävänsä ja tämän jälkeen hän ajaa testit. Mikäli testi epäonnistuu hän joko korjaa bugin, joka aiheuttaa testin epäonnistumisen, tai hän muuttaa testiä, jotta testi onnistuu. Jälkimmäinen vaihtoehto olisi huono idea, sillä tämä johtaa siihen, että määritellyt toiminnot toimivat eri tavalla per asiakasprojekti.

Ehdotuksena onkin, että tästä tehdään oma versionhallinta projekti. Samaan tyyliin, kuin pohjakauppa on tällä hetkellä. Tähän uuteen projektiin rakennetaan testausympäristö, joka toteutetaan Nightwatch -palvelun avulla. Tämä projekti sisältäisi testausympäristön lisäksi myös testit, jotka suoritetaan sitten pohjakaupan projektissa tai asiakkaan projektissa.

Tästä projektista lopulta luodaan alimoduuli pohjakaupalle. Alimoduulin yleinen käyttötarkoitus on, että halutaan kehittää yhteen projektiin kaksi eri toimintoa, mutta nämä toiminnot halutaan pitää erillään toisistaan versionhallinnan näkökulmasta (Git n.d).

Tämä käytäntö mahdollistaa, että projektia voidaan lähestyä uuden projektin näkökulmasta. Tämän projektin ylläpito on riippumaton pohjakaupasta, jota kehitetään koko ajan. Myös vanhoihin kauppoihin olisi mahdollista suhteellisen helposta lisätä tämä toteutus.

Työarvio tälle toteutukselle olisi kolme työpäivää, mikä vastaa 22,5 tuntia. Tämä työarvio pitää sisällään uuden projektin pystyttämisen versionhallintaa, testausympäristön asentamisen siihen ja alimoduuli määritysten lisäämisen pohjakauppaan. Myös automaatiotestaus projektin dokumentointi kuuluu tähän työhön. Dokumentaatio vastaa kysymyksiin:

- Miten asennan projektiini
- Miten ajan testejä
- Miten päivitän uusimpaan versioon

Kaksi työpäivää tuntuu realistiselta työarviolta, mikä voidaan helposti saavuttaa, mikäli työ pystytään tekemään esteettä ja rauhassa. Kolmas työpäivä työarviossa toimii puskurina.

6.4 Testauksien laajuus

Kuinka laajasti tulisi automaatio testejä suorittaa? Laajalla testauksella tarkoitetaan, että jokainen toiminto, joka on kehitetty, tai tullaan kehittämään, saisi automaattisen testauksen taakseen.

Hyvä ohjesääntö on, mikäli toiminto on monimutkainen, sen testaaminen tulisi tapahtua manuaalisen työn kautta. Esimerkiksi ostoskorin testaaminen tulisi toteuttaa manuaalisella työllä. Automaatio testauksen oleellisin piirre on, että se suoritetaan aina samalla tavalla. Mutta on epärealistista olettaa, että käyttäjät käyttävät ja käyttäytyvät aina samalla tavalla. Monimutkainen toiminta antaa mahdollisuuksia käyttää toiminta eri tavoilla.

Hyvän käyttäjäkokemuksen testaamista on erittäin epäluotettava jättää automaation varaan, sillä käyttäjäkokemus on erittäin subjektiivinen. Vaikka testit palauttavat kaikkiin tilanteisiin onnistuneen tuloksen. Voi toiminnon käyttämäinen olla erittäin vaikeaa, tai jopa mahdotonta. Tämä on

selkä ristiriita, toiminto toimii oikein kaikissa määritetyissä tilanteissa, mutta sen käyttäminen on käyttäjälle vaikeaa. Toisin sanoen toiminnon toiminnassa on puutteita.

Toinen ohjesääntö tämän opinnäytetyön perusteella on: mikäli toiminto on tärkeä, lähtökohtaisesti se tulisi testata manuaalisen työn kautta, musta laatikko. Testaaminen tulee tehdä käyttäjä mielessä, mikäli automaation kautta testataan tärkeä toiminto tarkoittaa tämä, että mikäli käyttäjä käyttää niin sanotusti ”oikein” toimintoa, se toimii. Tässä tilanteessa jää huomioimatta tilanteet, joissa käyttäjä käyttää ”väärin” toimintoa, ennen kuin on liian myöhäistä. Tähän kriteeriin mahtuu tuotteen lisääminen ostoskoriin -toiminto.

Omien ajatuksieni pohjalta tuotteen lisääminen ostoskoriin on mahdollista testata automaation kautta, vaikka se on tärkeä toiminta, mutta se on samalla erittäin yksinkertainen. Testin takana tulee olla hyvät määritykset.

7 Pohdinta

Tämän opinnäytetyön tarkoituksena oli rajata kolmesta mahdollisesta vaihtoehtoisesta testausympäristöstä paras mahdollinen. Jokaisella näistä ympäristöistä on omat vahvuutensa ja heikkoutensa, luonnollisesti näitä käytetään rajauskriteereinä. Lopulta yksi näistä kolmesta täytti kriteerit.

Teknisen toteutuksen toteuttaminen oli myös alun perin ajatuksena toteuttaa tämän opinnäytetyön puitteissa, mutta tämä rajautui nopeasti pois. Suurimpana syynä, että tekninen toteutus, ohjelmointi, tuottaisi erittäin vähän sisältöä, mutta vie aikaa. Mikäli toteutuksen olisi pystynyt tekemään esimerkiksi verkkokyselyn muodossa, olisi tästä saanut valtavasti sisältöä opinnäytetyön raportointi osuuteen.

Teknisen toteutuksen toteuttaminen samalla, kun aihetta tutkii, on myös erittäin epätehokas tapa työskennellä. Työn aikana sain vastaukset riittävällä tasolla tutkimuskysymyksiin ja pystyin suunnittelemaan alustavasti toimivan toteutuksen toimivan toteutuksen tälle. Toteutuksen, jonka päälle voidaan rakentaa automatisoidut testaukset. Opin myös tarkastelemaan testausta prosessina uudella tavalla.

Vastaukset tärkeimpiin kysymyksiin löytyivät kirjallisuuskatselmoinnin aikana. Garcian (2018) teos osoittautui kaikkein parhaimmaksi. Suurin epävarmuus tekijä oli hahmottaa, kuinka testausympäristö tulisi liittää pohjakauppaan, tai asiakkaan kauppaan. Arvioidakseni opinnäytetyön toteutusta ja onnistumista, sanoisin työn vastaavaan tavoitetta.

Ennen toteutuksen aloitusta on vielä tarpeen käydä ideaa tarkemmin sisäisesti läpi yrityksessä. On mahdollista, että organisaation sisällä tästä toteutuksesta toivotaan erittäin laajaa. Kuten testauspyramidi osoitti (kuva 10), tämä on epärealistinen päämäärä. Tämä dialogi johtaakin rajauksiin, mitä tarkalleen aletaan testaamaan automaation kautta ja mitkä ovat näiden testauksien rajaukset.

7.1 Toteutuksen hyvät ja huonot puolet

Suurin vahvuus tälle valitulle toteutustavalle on, että se olisi olemassa täysin irrallaan pohjakaupasta, sekä muista kaupoista. Pohjakaupan kehitystä voidaan jatkaa, siitä missä se on tällä hetkellä. Toteutus olisi myös mahdollista ottaa käyttöön jo olemassa olevissa oikeissa kaupoissa. Vaikka on mahdollista, että toteutuksen voi ottaa käyttöön olemassa olevassa kaupoissa, tulee käyttöönotto tehdä vain harkinnan kautta. Projektien tarpeita tulee katsella tapauskohtaisesti.

Tämä johtuu puhtaasti siitä, koska on vaikea tietää mitä räätälöintejä kauppaan on voitu jo tehdä. Nämä räätälöinnit voivat mahdollisesti johtaa testien hajoamiseen. Näissä tilanteissa testit palauttavat väärän tuloksen. Toisin sanoen määritelty toiminto toimii väärin, mutta kauppa toimii oikein. Tilanteen voi ratkaista toimiminen punainen-vihreä-refaktoroi (kuva 10) toimintamallin mukaan. Tähän tilanteeseen tulisi ajautua ainoastaan, mikäli projekti saa siitä selvää hyötyä.

Toteutus on olemassa omana projektinaan, irrallaan pohjakaupasta. Tässä on mahdollisesti myös toteutuksen suurin heikkous. Pohjakauppa on projekti, jolla on määritelty kehitysprosessi ja määritelty kehittäjä. Toisin sanoen on mahdollista, että testausprojektin ylläpidon on pysyttävä yhteisessä rytmissä pohjakaupan kanssa. Jotta tältä ongelmalta vältytään, testausprojektin kehityksen tulee seurata pohjakaupan kehitysprosessia ja sillä täytyy olla oma määritelty kehittäjä. Luonnollisesti tämä kehittäjä voi olla sama, kuin pohjakaupan kehittäjä.

On kuitenkin huomioitava, että vain valitut toiminnot tarvitset automaatio testauksen. Pohjakaupan kehitykseen käytettävä määrittely muuttuu huomattavasti tiukemmaksi. Tiukempi määrittely vaihe tuo mukanaan muutoksen jo vakiintuneeseen kehitysprosessiin.

Vahvuudet	Heikkoudet
• Mahdollisuus kehittää irrallaan pohjakaupasta. Helpottaa huomattavasti ensimmäisen version toteutusta. • Mahdollista ottaa käyttöön jo olemassa olevissa projekteissa	• Käyttöönotto vanhoissa projekteissa voi vaatia ylimääräistä työtä • Räätälöinnit pohjakauppaa saavat testit epäonnistumaan.
Mahdollisuudet	Uhat

• Uusia testejä voidaan kehittää ja niitä voidaan ottaa käyttöön takautuvasti projekteissa. • Projektikohtaiset testit	• Mahdollisesti jää jälkeen pohjakaupan kehityksestä

Taulukko 5. Yhteenvedon SWOT.

7.2 Toteutuksen avoimet kysymykset

Opinnäytetyössä löydettiin ratkaisut asetettuun ongelmaan. Kysymykset, jotka ovat avoinna, liittyvät tekniseen toteutukseen ja projektin roolittamiseen.

Kysymys roolittamisesta viittaa siihen, kuka on henkilö, joka alkaa ylläpitämään tätä projektia. Vastaus tähän kysymykseen on erittäin tärkeä ja olisi suotavaa saada vastaus ennen varsinainen toteutuksen aloitusta. Tämän henkilön tulisi olla mukana pohjakaupan kehityksestä, jotta hän pysyy kärryillä toiminnoista, mitä sinne ollaan tekemässä.

Muut polttavat kysymykset, jotka jäivät avoimeksi, liittyvät itse testien tekniseen toteutukseen. Eli auki jääneet kysymykset ovat täysin uusia kysymyksiä, täysin riippumattomia opinnäytetyön tutkimuskysymyksistä. Suurin näistä ongelmista on räätälöinnin mahdollisuus. Pohjakauppa on suunniteltu, että sitä voidaan räätälöidä, mikäli projekti sitä vaatii. Kysymykseksi jää, kuinka toteuttaa testit, jotka toimivat oikein räätälöinneistä huolimatta. Tähän kysymykseen on vaikea antaa vastausta ilman määriteltyjä testitapauksia. Nämä testitapauksen määritellään opinnäytetyö prosessin ulkopuolella.

Jatkokehitys ideana olisi hyvä pohtia ja tutkia mahdollisuutta lisätä asiakasprojektikohtaisia testejä, jotka voidaan muiden testien lomassa. Tämän kaltaisesta toiminnosta olisi teoreettisesti ajateltuna hyötyä.

Lähteet

BrowserStack (N.d). *Run your Cypress tests on BrowserStack Automate | BrowserStack Docs*. Saatavilla 9.5.2021. https://www.browserstack.com/docs/automate/cypress

BrowserStack. (2020). *Bug Severity vs Priority in Testing.* Saatavilla 23.1.2021. https://www.browserstack.com/guide/bug-severity-vs-priority.

BrowserStack. (2020). *End To End Testing: A Detailed Guide.* Saatavilla 14.2.2021. https://www.browserstack.com/guide/end-to-end-testing

Garcia, A. (2018). *Test-Driven Java Development - Second Edition*. Packt Publishing.

Git (N.d) *Git – Submodules*. Saatavilla 1.5.2021. https://git-scm.com/book/en/v2/Git-Tools-Submodules

Gupta, N. (2019). *What is difference between Defect, Bug, Error*. Saatavilla 23.1.2021. https://medium.com/swlh/what-is-difference-between-defect-bug-error-b477e76b5502

Hallavo, J. (2013). *Verkkokaupan rautaisannos*. Talentum.

Holcombe, W. M. L. (2008). *Running an Agile Software Development Project*. John Wiley & Sons, Inc.

Homés, B. (2013). *Fundamentals of Software Testing*. London: Wiley.

Host, M., Rainer, A. & Runeson, P. (2012). *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley.

Hughes, R. (2013). *Agile Data Warehousing Project Management*. Morgan Kaufmann.

Juuti, P. & Puusa, A. (2020). *Laadullisen tutkimuksen näkökulmat ja menetelmät*. Gaudeamus.

LianaTechologies. (N.d). *Tietoa meistä*. Saatavilla 29.5.2021. https://www.lianatech.fi/tutustu/tietoa-meista/tietoa-meista.html

Mili, A. & Tchier, F. (2015). *Software Testing: Concepts and Operations*. Wiley.

Myers, G. J., Badgett, T. & Sandler, C. (2012). *Art of Software Testing*. John Wiley & Sons, Inc.

Nidhra, S & Dondeti, J. (2012). *BLACK BOX AND WHITE BOX TESTING TECHNIQUES –A LITERATURE REVIEW. International Journal of Embedded Systems and Applications (IJESA) Vol.2, No.2, June 2012. 29–50*. https://www.researchgate.net/profile/S-Nidhra/publication/276198111_Black_Box_and_White_Box_Testing_Techniques_-_A_Literature_Review/links/570e313f08ae2b772e46aa40/Black-Box-and-White-Box-Testing-Techniques-A-Literature-Review.pdf

Pries, K. H. & Quigley, J. M. (2010). *Scrum Project Management*. CRC Press.

Simons, H. (2009). *Case Study Research in Practice*. SAGE Publications Ltd.

Technopedia (N.d). *What is Error Handling – Definition from Technopedia*. Saatavilla 23.1.2021. https://www.techopedia.com/definition/16626/error-handling

Technopedia. (2017). *What is Software Bug? - Definition from Techopedia*. Saatavilla 23.1.2021. https://www.techopedia.com/definition/24864/software-bug-

Vuorinen, T. (2013). *Strategiakirja: 20 työkalua*. [Helsinki]: Talentum.