Haaga-Helia
ammattikorkeakoulu Oy

Testausautomaation käyttöönotto uudessa ohjelmisto-hankkeessa

Kalle Mujunen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2020

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

20.11.2020

Tekijä(t)
Kalle Mujunen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Testausautomaation käyttöönotto uudessa ohjelmistohankkeessa

Sivu- ja liitesivumäärä
40

Tämä opinnäytetyö käsittelee automaatiotestausta tilanteessa, jossa yritys tai organisaatio ottaa käyttöön testausautomaation uuden, ketterillä menetelmillä kehitetyn järjestelmän käyttöönoton yhteydessä. Opinnäytetyö toteutettiin kirjallisiin lähteisiin tukeutumalla. Tavoitteena oli selvittää, miten varmistetaan testausautomaation käyttöönottoa projektitasolla, mitkä ketterien sovellustiimien ominaisuudet ja taidot edistävät testausautomaation käyttöönottoa, ja mikä on testaajien ja testaamisen merkitys ketterässä sovelluskehityshankkeessa.

Testaaminen tarkoittaa vikojen etsimistä, toiminnallisuuden ja ominaisuuksien testaamista ja ohjelman tutkimista sen tarkoituksenmukaisuuden näkökulmasta. Testaaminen perustuu suunnitelmallisuuteen, tavoitteellisuuteen ja ennalta määriteltyihin tuloksiin. Testaamisen tulokset raportoidaan ja näiden raporttien pohjalta suunnitellaan tarvittavat muutostoimet.

Työssä selvitettiin testausautomaatioon liittyviä hyötyjä ja haittoja. Testausautomaatioon liittyvien hyötyjen saaminen edellytti automatisoituun testaukseen liittyvien rajoitteiden tunnistamista ja automaattitestauksen suunnittelua, sekä automaattitestaukseen valittavien testitapausten ja -skriptien määrittämistä ja kirjoittamista. Testausautomaatio ei ole vastaus kaikkeen testaukseen automatisoitujen testitapausten ylläpitoon liittyvän työmäärän vuoksi.

Ketterät menetelmät painottavat yksilöitä ja vuorovaikutusta, asiakasyhteistyötä, toimivan ohjelman julkaisua ja muutokseen reagointia. Ketteryydellä tarkoitetaan myös kykyä luoda muutosta muutokseen reagoinnin lisäksi. Tietojärjestelmäkehityksessä tämä tarkoittaa iteratiivista, sykleittäin etenevää kehitysmallia, jonka aikana jokainen kehitetty ohjelman moduuli testattiin iteraation aikana ennen julkaisemista.

Kuvitteellisessa esimerkkitapauksessa asiakasorganisaatio otti käyttöön testausautomaation kehittäessään uutta järjestelmää tarpeisiinsa. Testausautomaation onnistuneeseen käyttöönottoon liittyi vaatimus siitä ymmärryksestä, että testausautomaatio on itsessään ohjelmisto, ja tämä tarkoitti kohdeorganisaation näkökulmasta kahta samanaikaista ohjelmistoprojektia ja niiden tarpeiden määrittelyä.

Johtopäätös oli, että hankkeen onnistuminen edellytti kaikilta osapuolilta sitoutumista, toimivaa vuorovaikutusta ja asiantuntemusta. Testausautomaation onnistunut käyttö vaati ammatillista osaamista, joka sisältää kykyä ohjelmoida, suunnitella, ylläpitää ja testata testiskriptejä. Tämän lisäksi automaatiotestaukseen määritellyt testitapaukset pitää osata tunnistaa ja määritellä oikein kokonaisuuden hallitsemisen varmistamiseksi.

Työn aikana nousi esiin jatkotutkimuksen aihe, joka koskee tietojärjestelmähankkeiden sisäistä kommunikaatiota, sen laatua ja niiden vaikutusta hankkeen sujuvuuteen.

Asiasanat
automaatio, automaatiojärjestelmät, ketterät menetelmät, testaus

Sisällys

1 Johdanto

Testaaminen on välttämätön osa sovellusten kehittämistä. Vain testaamalla voidaan varmistaa, että ohjelma tekee toivotut asiat ja vieläpä siten, kuin sen halutaan ne tekevän. Testaamisella voidaan myös vähentää ohjelmassa olevia virheitä niin, että ohjelma tuottaa aiottua hyötyä ja samalla vähennetään merkittävästi ohjelman toimimattomuudesta aiheutuvia taloudellisia ja toiminnallisia haittoja. Testaamiseen ei pitäisi suhtautua pakollisena työvaiheena, jossa tärkeintä on saada testitapaus menemään läpi.

Testaamisella etsitään ohjelmasta virheitä, mutta samalla testaamisen avulla selvitetään, että ohjelma toimii, kuten vaatimusmäärittelyssä on esitetty. Testaaminen voi tässä vaiheessa myös tuoda esiin sen, että vaatimusmäärittely ei vastaakaan todellisia toiveita, ja vaikka ohjelma toimisi määrittelyn näkökulmasta oikein, käyttäjän näkökulmasta ohjelma saattaa toimia väärin. Lisäksi testauksen yhteydessä varmistutaan siitä, että ohjelma tekee sen, mitä loppukäyttäjä haluaa. Ohjelma, joka toimii testauksen aikana moitteettomasti, ei vastaa tarkoitustaan, mikäli ohjelma ei vastaa käyttötarkoitustaan.

Tietojenkäsittely ja automatisointi ovat tulleet arkipäiväiseksi osaksi elämää kaikilla aloilla. Ilman tietojenkäsittelyä, siihen liittyviä laitteistoja ja ohjelmistoja nykyaikainen yhteiskunta ei enää toimi. Tietojärjestelmiin pätevät samat lainalaisuudet kuin niihin laitteistoihin, joilla niitä käytetään. Niiden käytöllä on elinkaari, jonka jälkeen ne muuttuvat auttamatta vanhentuneiksi. Syitä tähän on monta. Käyttöjärjestelmien kehittyessä uudemmat järjestelmät eivät välttämättä tue vanhoja sovelluksia ilman kallista, hankalasti ylläpidettävää tukea. Vanhat järjestelmät eivät yksinkertaisesti palvele muuttuneita tarpeita, minkä lisäksi monet vanhoista järjestelmistä on ohjelmoitu kielillä, joiden osaajia ei välttämättä enää ole.

Testauksen toteuttaminen manuaalisesti on aikaa vievää työtä ja sitoo sovelluskehitykseen osallistuvaa henkilöstöä testaukseen, kun he usein voisivat tehdä muuta työtä. Automaatio on tullut testaukseen nopeuttamaan sitä ja avustamaan testaajia tehtävissä, jotka toistuvat rutiininomaisina ja joiden toteuttaminen manuaalisesti ei ole mielekästä. Automaatio on oikein käytettynä nopeampi, tarkempi ja virheettömämpi, toimii kustannustehokkaammin ja kaikkina vuorokauden aikoina. Koska testausautomaation avulla ei voi testata kaikkea, automaatio vapauttaa testaajat tehtäviin, jotka kannattaa toteuttaa yhä manuaalisesti.

Uudemmat järjestelmät ohjelmoidaan ketterämmiksi, modulaarisiksi ja niitä on mahdollista käyttää verkon yli. Ohjelmistokielet ja laitteistovaatimukset muuttuvat, ja tämän lisäksi näihin järjestelmiin liitetään usein laitteistoja ja ohjelmistoja, jotka eivät alkuperäiseen kokoonpanoon kuulu, mutta joiden toiminta on ohjelmistotalon asiakkaalle tämän ydintoimin-

nan vuoksi välttämätöntä. Tämä luo tarpeen päivittää ohjelmistoja, rajapintoja ja laitteistoja vastaamaan uusia tarpeita. Tässä yhteydessä päivittyvät usein myös toiminnan prosessit, sillä ne kuvataan juuri ohjelmiston vaatimuksia arvioitaessa.

Tämän opinnäytetyön yhteydessä kuvataan teoreettinen viitekehys, johon tässä yhteydessä kuuluvat testaaminen, testausautomaatio ja ketterät menetelmät. Teoreettisessa viitekehyksessä kuvattu tausta liitetään yhteen empiirisessä osassa. Tässä vaiheessa esimerkkitapausta tarkastellaan teoreettisen viitekehyksen eri näkökulmista.

Tässä päättötyössä tarkastellaan kuvitteellista tapausta, jossa ketterillä menetelmillä kehitettyä, asiakaspalvelukäyttöön tarkoitettua ohjelmistoa testataan testausautomaatiota käyttäen sen käyttöönottovaiheen aikana. Esimerkkitapauksessa organisaatio ei ole käyttänyt testausautomaatiota aiemmin. Ohjelmiston käyttäjät ovat pääasiassa juuri asiakaspalveluorganisaation henkilöstöä. Tässä päättötyössä ei oteta kantaa siihen, onko kyseinen organisaatio julkishallinnon organisaatio, tai liike- tai rahoitustoimintaa harjoittava yritys.

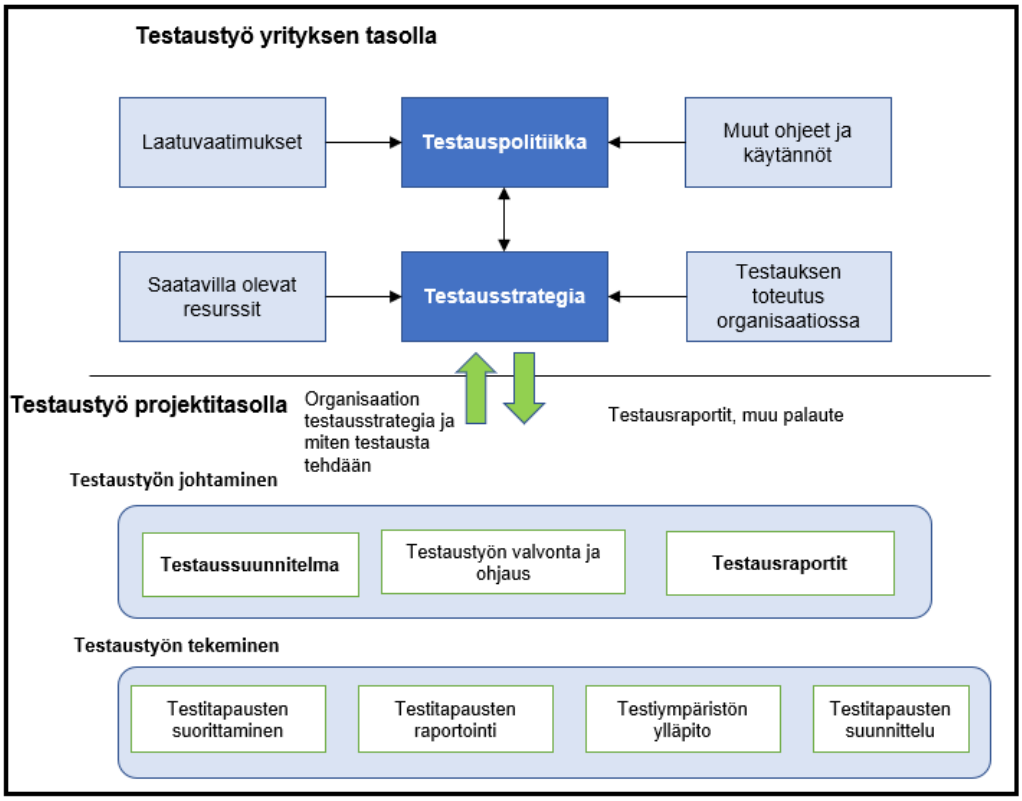
Tavoitteena on selvittää:

- Millä tavoin varmistetaan testausautomaation käyttöönoton onnistuminen hankkeessa projektitasolla?
    - ohjelmistohankkeen vaatimusten määrittelyn onnistumisen edellytykset
    - testausautomaation vaatimusten määrittelyn onnistumisen edellytykset
- Mitkä ketterän sovellustiimin ominaisuudet edistävät testausautomaation käytön onnistumista?
    - sovellustiimin tiedot ja taidot
    - sovellustiimin toimintaperiaatteet
- Mikä on testaamisen ja testaajien merkitys ketterässä sovelluskehityshankkeessa?
    - ketterien menetelmien vaikutus automaattitestauksen prosessiin

2 Testaaminen

Testaaminen on suunniteltu ja valmisteltu prosessi, jossa on ennalta määritelty, odotettu lopputulos. Lopputulos on se ehto, jonka täyttyminen tarkoittaa testauksen onnistunutta läpäisyä. Testaaminen voi olla sekä dynaamista, mikä tarkoittaa koodin ajamista, tai staattista, joka tarkoittaa ohjelman tutkimista dokumenttien ja lähdekoodin perusteella ilman ohjelman suorittamista. Testaamisen tarkoituksena on virheiden etsiminen ja ohjelman toiminnallisuuden ja käytettävyyden tutkiminen. Testausprosessi arvioidaan aina, ja tähän kuuluu myös vaatimusten ja määritysten tutkiminen. (Graham, Veenendaal, Evans & Black 2008, 12-13.)

(Kuvassa 1 kuvattu ISO/IEC 29119 -standardin mukaan (2013) tehty testauksen toimintamalli ei ota kantaa varsinaiseen testaustapaan tai ohjelmistokehityksen menetelmiin. Testauksen toimintamalli toimii yhtä pätevästi niin ketterillä menetelmillä kuin vesiputousmallinkin avulla kehitetyllä ohjelmistolla, ja toisaalta malli toimii yhtä hyvin myös täysin manuaalisessa testauksessa kuin osin automatisoidussakin testauksessa. Standardi on kansainvälinen ja soveltuu testaamisen tarkasteluun eri näkökulmista. Yrityksille standardi on hyödyllinen siksi, että sen perusteella on mahdollista tehdä auditointeja tai hankkia sertifikaatteja. (Kasurinen 2013, 82.)



Kuva 1. Yleinen testauksen toimintamalli ISO/IEC 29119 -mallia mukaillen (Kasurinen 2013, 82)

Testaamisessa yleisellä tasolla tukeudutaan organisaation testauspolitiikkaan ja testausstrategiaan. Näiden hyväksymisestä vastaa hankkeen ylin johto. Usein ylin johto myös tuottaa nämä dokumentit ja niiden tehtävänä on toimia ohjaavina periaatteina organisaation testaustyössä. (Ahmed 2018, 113; Kasurinen 2013, 82-83.)

Projektitasolla testauksella on kaksi kerrosta, johtaminen ja itse testauksen toteuttaminen. Testaus perustuu testaussuunnitelmiin ja johtaminen on varsinaisen testaustyön valvontaa ja ohjausta. Testauksen johtamiseen kuuluu myös testaustyön raportointi yritystason johdolle. Varsinaiseen testaustyön tekemiseen kuuluu testitapausten suorittamisen lisäksi toteutuksen raportointi testaustyön johdolle, testiympäristöjen ylläpito ja testitapausten suunnittelu. (Ahmed 2018, 113; Kasurinen 2013, 82-84.)

Helsingin Yliopiston tietojenkäsittelytieteen luennoitsija Antti-Pekka Tuovinen määritteli luennollaan testaamisen niin, että sillä tarkoitetaan ohjelman suorittamista, jotta ohjelman toimintaa tai käyttäytymistä voidaan verrata ohjelmalle asetettuihin vaatimuksiin. Tuovisen mukaan testaamisen tarkoituksena on vikojen löytäminen, laadun mittaaminen, luottamuksen lisääminen ja dokumentaation analysointi vikojen ennaltaehkäisemiseksi (Tuovinen 12.3.2013).

Kasurinen puolestaan määrittelee testaamisen niin, että kyse on työstä, jolla varmistetaan, että ohjelmistosta tulee sellainen, kuin on toivottu. Samalla varmistutaan siitä, että ohjelman ominaisuudet toimivat toivotulla tavalla (Kasurinen 2013, 10).

Teoksessaan Foundations of Software Testing Graham, Veenendaal, Evans ja Black esittävät seitsemän testauksen periaatetta (Graham, ym. 2008, 19). Kasurinen esittää samat periaatteet kirjassaan Ohjelmistotestauksen käsikirja (Kasurinen 2013, 38). Periaatteet ovat kuvattuina alla lyhyesti.

> Periaate 1. Testaaminen näyttää ohjelmassa olevat virheet, mutta ei voi todistaa, että ohjelma on virheetön. (Graham ym. 2008, 19.)
>
> Periaate 2. Täydellinen testaaminen on mahdotonta. Testaamisen painopisteiden tulisi perustua riskianalyysiin ja ohjelman ensisijaisiin ja kriittisimpiin käyttökohteisiin. (Graham ym. 2008, 19.)
>
> Periaate 3. Varhainen testaaminen. Testaamisen tulisi alkaa mahdollisimman varhain ohjelmistokehityksen aikana ja sen pitäisi keskittyä määriteltyihin tavoitteisiin. (Graham ym. 2008, 19.)

Periaate 4. Virheiden kerääntyminen. Ennen julkaisua tehdyn testauksen aikana suuri osa virheistä löytyy pienestä joukosta ohjelman osia tai moduuleja. (Graham ym. 2008, 19.)

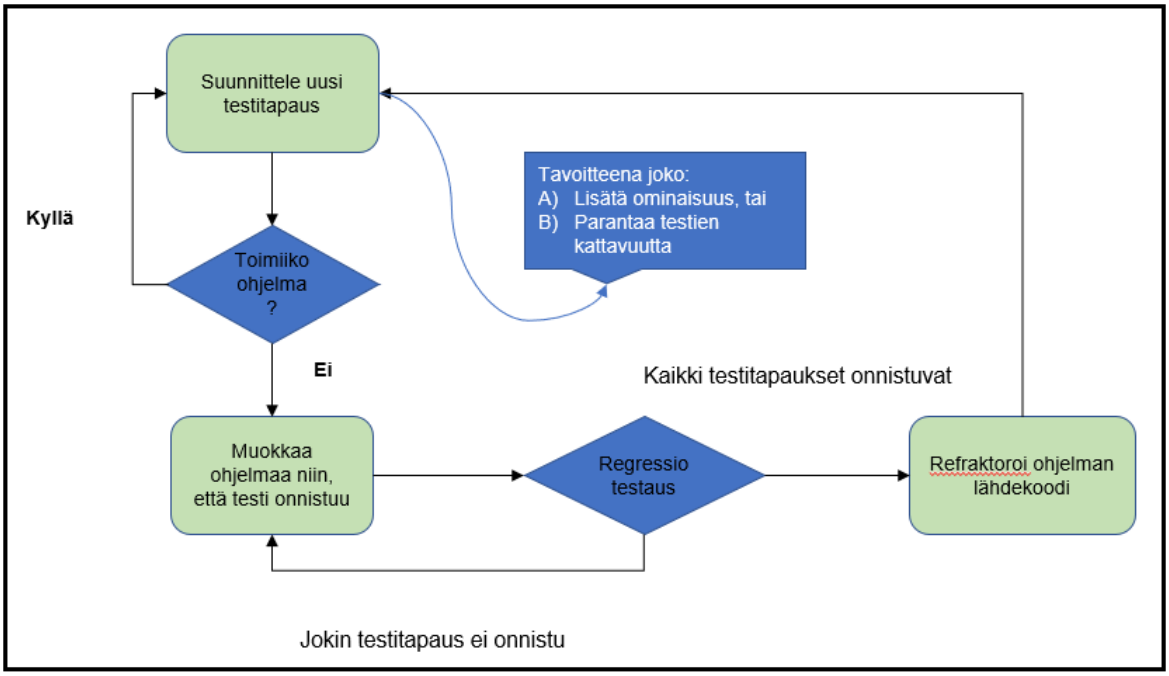
Periaate 5. Pesticide Paradox tai ”hyönteismyrkkyparadoksi” (Kasurinen 2013, 93). Tämä tarkoittaa käsitteenä sitä, että mitä enemmän samaa testiä ajetaan jollekin ohjelman osalle, sitä vähemmän virheitä testillä löydetään. Tämän välttämiseksi testiä pitäisi arvioida, uudistaa ja muuttaa säännöllisin väliajoin. (Graham ym. 2008, 19.)

Periaate 6. Testaaminen riippuu kontekstista. Tämä tarkoittaa, että testaaminen toteutetaan eri tavoin, kun kyse on esimerkiksi korkean tietoturvavaatimustason ohjelmasta tai verkkokaupan sivustosta, joka esittelee verkkokaupan tuotekuvastoa. (Graham ym. 2008, 19.)

Periaate 7. Virheiden puuttumisen harhakuva. Vaikka ohjelmasta löydettäisiin ja korjattaisiin kaikki virheet, tämä ei tee ohjelmasta käytettävää loppukäyttäjän näkökulmasta, mikäli ohjelma ei sovellu aiottuun käyttötarkoitukseensa. (Graham ym. 2008, 19.)

2.1 Testiohjattu sovelluskehitys

Testaaminen voi toimia myös sovelluskehityksen lähtökohtana ja tässä tapauksessa puhutaan testiohjatusta sovelluskehityksestä (Kuva 2).



Kuva 2. Testiohjattu ohjelmistokehitys Kasurista mukaillen (Kasurinen 2013, 121)

Testiohjattu sovelluskehitys tarkoittaa sitä, että ensimmäiseksi tehdään testitapaus. Seuraavaksi ohjelmaa kirjoitetaan sen verran, että ohjelma kykenee suorittamaan ennalta kirjoitetun testitapauksen onnistuneesti (Cockburn 2007, 275; ITSQB 2014, 29; Kasurinen 2013, 120; Katara, Vuorinen & Jääskeläinen 1.9.2016). Katara, Vuorinen ja Jääskeläinen Tampereen yliopiston tietotekniikan laitokselta esittävät luentomateriaalissaan, että testiohjatussa ohjelmistosuunnittelussa yksikkötestauksesta muodostuu rutiinia, ja testien läpäisy kertoo samalla koodauksen edistymisestä (Katara ym. 1.9.2016).

Haikala ja Mikkonen kuvaavat testiohjatun ohjelmistosuunnittelun sykliä kirjassaan Ohjelmistotuotannon käytännöt listaamalla viisi vaihetta. Ensimmäiseksi lisätään heidän mukaansa uutta ominaisuutta koskeva testitapaus olemassa olevien testitapausten joukkoon. Ajatuksena tässä on, että sovelluksen kehittäjä ymmärtäisi, mitä ominaisuuden pitäisi tehdä. (Haikala & Mikkonen 2011, 215; ITSQB 2014, 29; Katara ym. 1.9.2016.)

Tämän jälkeen kaikki testitapaukset suoritetaan ja tarkastetaan, epäonnistuuko jokin testitapauksista. Jos tässä vaiheessa uuden testitapauksen suoritus onnistuu, on se kuitenkin merkki siitä, että joko testitapauksessa, testausjärjestelyissä tai itse järjestelmässä on vikaa. Testitapauksen tulisi epäonnistua tässä vaiheessa, koska vaadittavaa ohjelmakoodia ei ole vielä kirjoitettu. (Haikala & Mikkonen 2011, 215; ITSQB 2014, 29; Katara ym. 1.9.2016.)

Uuden testitapauksen kuvaama ominaisuus rakennetaan seuraavaksi. Koodia kirjoitetaan vain sen verran, että testitapaus menee läpi (Cockburn 2007, 275; Haikala & Mikkonen 2011, 215; ITSQB 2014, 29; Katara ym. 1.9.2016).

Testitapaukset testataan uudelleen ja testitapausten onnistuminen tarkistetaan. Etuna tässä menettelyssä on se, että sovelluksen kehittäjä tietää, että uudet ominaisuudet eivät ole vioittaneet aiemmin rakennettuja. Tämä johtuu siitä, että kaikki testitapaukset ajetaan aina. (Haikala & Mikkonen 2011, 215; ITSQB 2014, 29; Katara ym. 1.9.2016.)

Lopuksi ohjelmaa refaktoroidaan, mikä tarpeellista. Refaktorointi tarkoittaa ohjelman siistimistä (Katara ym. 1.9.2016). Haikala ja Mikkonen tarkoittavat tässä yhteydessä myös laajempaa ohjelman muokkaamista ja parannusten tekemistä ohjelman rakenteeseen ja arkkitehtuuriin. (Haikala & Mikkonen 2011, 215-216; ITSQB 2014, 29.)

Bertrand Meyer kritisoi kirjassaan Agile! The Good, Hype and the Ugly testivetoista kehitystä toteamalla, että testivetoinen kehitys aiheuttaisi näkökentän kaventumisen, minkä vuoksi ohjelmistokehitysyhtiöt eivät ole ottaneet menetelmää laajamittaisesti sellaisenaan käyttöön (Meyer 2014, 151).

2.2 Muutosten testaaminen

Muutosten testaaminen voidaan jakaa kahteen luokkaan, vahvistustestaukseen ja regressiotestaukseen. Vahvistustestaus tulee kyseeseen, jos alkuperäisen testauksen yhteydessä on havaittu toiminnallinen virhe, joka on korjattu. Vahvistustestauksella pyritään todentamaan, että vika todella on saatu korjattua ja tähän tavoitteeseen päästään toteuttamalla testaus täysin samalla tavoin, kuin ensimmäisellä testauskerralla. (Graham ym. 2008, 49.)

Regressiotestaus on vahvistustestauksen tapaan samanlainen testaus, että myös tässä testataan jotakin, joka on jo testattu aiemmin (Graham ym. 2008, 49). Kasurisen mukaan termi (Kasurinen 2013, 54) itsessään tarkoittaa uudelleentestaamista, eikä kyse ole varsinaisesti uudesta testaustavasta. Regressiotestauksen tarkoitus on, että sen avulla varmistetaan ohjelman luotettava toiminta tilanteessa, missä ohjelman jotakin komponenttia on muokattu, tai ohjelmaan on lisätty uusia osia. Regressiotestaus eroaa vahvistustestauksesta siinä, että regressiotestauksessa käytettävä testiketju on tavallisesti läpäissyt aiemman testauksen. Regressiotestaus perustuu siihen, että virheet ohjelmakoodissa löytyvät uusista komponenteista, tai niitä käyttävistä toiminnoista. (Graham ym. 2008, 49-50; Kasurinen 2013, 54-55.)

Sovelluksen iteratiivinen kehitys saa aikaan sen, että jokainen regressio on edellistä suurempi. Muutoksia aiheuttavat koodin siistiminen, eli refaktorointi, uusien ominaisuuksien lisääminen ja niiden muokkaaminen ja koodirivien poistaminen (ITSQB 2014, 24). Ominaisuudet voivat muuttua myös sprinttien välisissä katselmoinneissa ja niiden yhteydessä tehdyissä suunnitelmissa. Tästä syystä testitapausten pitäminen ajan tasalla, vanhentuneiden poistaminen ja uusien suunnitteleminen ovat tärkeää ketterässä sovelluskehityksessä toimivan ohjelmiston tuottamiseksi. Ketterien menetelmien puolestapuhujat painottavat tässä testausautomaation merkitystä. (Graham ym. 2008, 177; ITSQB 2014, 24; Kasurinen 2013, 61; Katara ym. 1.9.2016.)

2.3 Testauksen suunnittelu

Hyväksymistestaamiselle ja sen määritelmälle on kirjassa Guide to Software Development. Designing and Managing the Life Cycle (Langer 2012, 262) suositus, jonka mukaan hyväksymistestaaminen pitäisi määritelläkin testaussuunnitelman toteuttamiseksi. Alistair Cockburn puolestaan esittelee kirjassaan Agile Software Development (Cockburn 2007, 275) lyhyesti testivetoisen ohjelmistosuunnittelun käsitteen. Ajatuksena on, että testaussuunnitelma on jo ohjelmoijilla tiedossa ennen koodaamisen aloittamista, ja ohjelmakoodia kirjoitetaan pienin mahdollinen määrä, jotta suunniteltu testitapaus saadaan läpäistyä.

Kumpikin kirjoittajista ilmaisee, että testaussuunnitelma tulee olla olemassa jo sovelluksen tai ohjelmiston kehityskaaren alkuvaiheessa.

Grahamin ja kumppaneiden mukaan testaussuunnitelma on dokumentti, johon on kirjattu muun muassa laajuus, resurssit, testaustapa, testausaikataulu ja testattavat ominaisuudet, mutta myös esimerkiksi testausympäristö ja sen ominaisuudet, tässä vain muutama esimerkki mainitakseni. Testaussuunnitelmalla on heidän mukaansa kolme tarkoitusta, jotka ovat ajattelun ja ajattelutavan ohjaaminen, kommunikointikeinona toimiminen hankkeen omistajien, käyttäjien ja kehittäjien välillä, sekä muutoksenhallinnan apuvälineenä toimiminen. (Graham ym. 2008, 133.)

Kasurinen esittää Ohjelmistotestauksen käsikirjassa, että testaussuunnitelma on projektitason suunnitelma, joka perustuu testausta suorittavan yrityksen testausstrategiaan. Testaussuunnitelma määrittää, mitä ohjelmasta testataan, missä vaiheessa projektia testaus toteutetaan ja millaisia menetelmiä käyttäen (Kasurinen 2013, 91-92). Testausstrategia puolestaan on hanketason dokumentti, joka määrittää yleisellä tasolla, miten organisaatiossa testausta tehdään (Graham ym. 2008, 140-142; Kasurinen 2013, 87-89).

Testitapaukset ovat keino testata ohjelmaa testaussuunnitelman mukaisesti. Testitapaus on malli, joka kuvaa yhtä testattavassa ohjelmassa toteutettavaa tapahtumaa (Kasurinen 2013, 93). Testitapaukseen kuvataan kaikki vaadittavat askeleet ja ehdot, jotka vaaditaan testitapauksen toteuttamiseksi (Kasurinen 2013, 94). Kasurinen jatkaa edelleen, että ensimmäiset testitapaukset tehdään vaatimusmäärittelyn antaman tiedon perusteella (Kasurinen 2013, 94).

Esimerkkinä tämä voi tarkoittaa seuraavaa: testillä testataan asiakkaan kirjautumista OmaKanta -järjestelmään käyttäen pankkitunnuksia. Testin esiehdot vaativat, että testauksessa on käytettävissä verkkopankkitunnuksia simuloiva kirjautumismahdollisuus, joka vastaa aitoudeltaan oikeita pankkitunnuksia ja OmaKanta-integraatio. Tämän lisäksi tulee olla testikäyttäjä, jolla on tietoja OmaKanta -järjestelmän testiympäristössä. Testi onnistuu, jos testaaja onnistuu kirjautumaan tietokoneeltaan OmaKanta -järjestelmään pankkitunnuksia käyttäen ja pääsee tarkastelemaan testiympäristöön luotuja tietoja testikäyttäjästä.

Testitapaukseen kirjataan, mitä testillä testataan, mitä ehtoja testin onnistuneeseen aloittamiseen liittyy, miten ohjelman pitäisi käyttäytyä saatuaan pyydetyn syötteen, testitapauksen suunnittelija, ohjelman osat, joihin testitapaus vaikuttaa (Kasurinen 2013, 94). Testitapaukseen kirjataan myös se, milloin testitapaus epäonnistuu tai testin vieminen loppuun keskeytyy, milloin testiä voidaan pitää hyväksytysti läpäistynä, kenellä on oikeus

tehdä muutoksia testitapaukseen, mahdolliset riippuvuudet muista testitapauksista, mikä on testitapauksen odotettu lopputulos, sekä mahdolliset rajoitteet. (Kasurinen 2013, 95.)

Kasurisen mukaan testitapausten suunnittelussa käytetään vaatimusmäärittelyn ja arkkitehtuurin malleja testattavasta järjestelmästä, ja että ensimmäiset testitapaukset luodaan näiden pohjalta (Gupta 2016; Kasurinen 2013, 93). Ketterillä menetelmillä toteutettavassa ohjelmistotuotannossa testitapausten suunnittelu tapahtuu yhteistyössä kehittäjien, testaajien ja asiakasorganisaation edustajien kanssa. Tämä tapahtuu käyttäjätarinoita luomalla sen jälkeen, kun ohjelmiston iteraatiolle määritellyt vaatimukset on kirjattu (ITSQB 2014, 13; Gupta 2016).

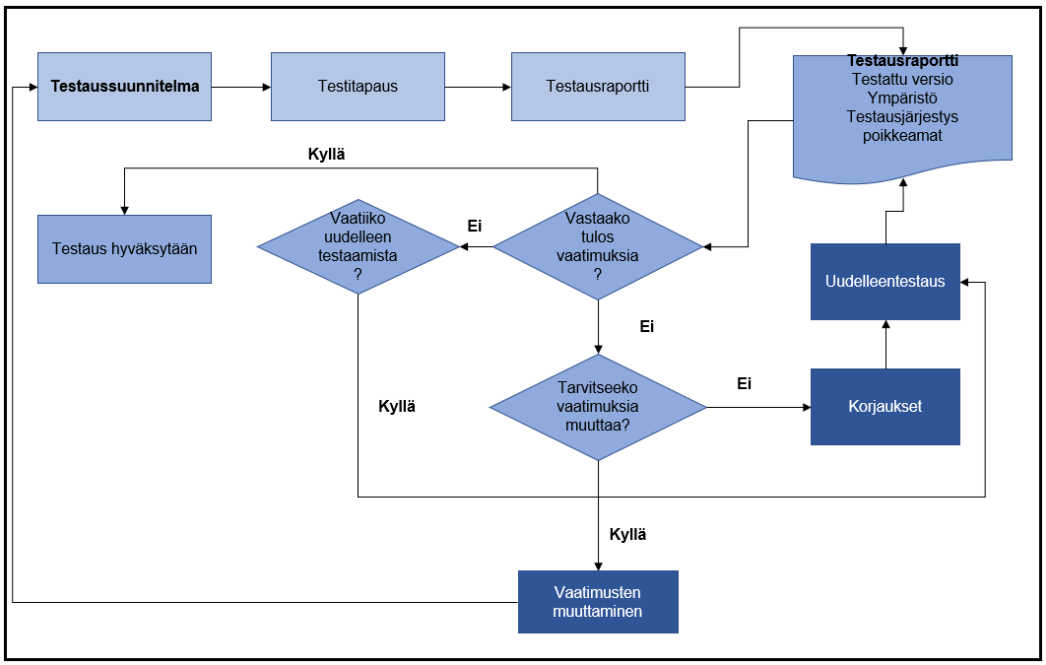
2.4 Testiympäristön luominen

Järjestelmän testaamista varten tarvitaan testausympäristö, jossa voidaan mallintaa kaikkia niitä ohjelmistolla suoritettavia toimia, joita tuotantoympäristössä ja lopullisen käytön yhteydessä tullaan tekemään. Testausympäristöön rakennetaan malli todellisista laitteistoista, palvelimista ja mahdollisista mobiili- ja päätelaitteista, instrumenteista, ohjelmistotyökaluista ja tiedonsiirtoon liittyvistä integraatioista, sekä käytettävästä datasta. Testausympäristön tulisi olla mahdollisimman tarkka kopio todellisesta tuotantoympäristöstä, jotta käyttöympäristön aiheuttamat riskit ohjelman toiminnalle voidaan minimoida. (Graham ym. 2008, 44, 136-137; Subramani 2020.)

Kataran ja kumppaneiden mukaan erot testaus- ja tuotantoympäristöjen välillä luovat tilanteen, että kaikkea ei testata. Asiaan vaikuttavat testausympäristön todellisuus kokoonpanon osalta ja testausdata. Onko testausdata luotu keinotekoisesti, vai käytetäänkö todellista dataa? Mitä muita ohjelmia käyttäjän laitteistossa on ja vastaavatko ne tuotantoympäristöä? Onko testausjärjestelmään integroituneet järjestelmät ja tietokannat simuloitu jollakin tavoin, vai käytetäänkö todellisia järjestelmiä ja tietokantoja? (Katara ym. 1.9.2016.)

2.5 Testin suorittaminen ja tulosten tarkasteleminen

Testauksen toteuttaminen on tavoitteellista ja suunnitelmallista toimintaa riippumatta testauksen tasosta. Testaus noudattaa testaussuunnitelmaa ja sen perusteella laadittuja testitapauksia. Testauksen aikana verrataan testauksen aikana syntynyttä todellista tulosta odotettuun testauksen tulokseen ja lopputulos raportoidaan. Testauksen kulku on kuvattu kuvassa 3.



Kuva 3. Testauksen kulku ja raportointi

Testauksen suorittamisesta tulee dokumentoida myös se, millä versiolla testaus toteutetaan, missä ympäristössä testaus tehdään ja missä järjestyksessä testitapaukset on toteutettu, mikäli testitapausten toteuttamisen järjestyksellä on testisalkun onnistumisen kannalta merkitystä. Odotettujen ja todellisten testitulosten väliset poikkeamat kirjataan testausraporttiin, joka analysoidaan. Tarpeen vaatiessa tehdään uudelleentestaus tai vahvistustestaus, mikäli testauksen aikana on todettu korjauksia vaativa poikkeama, ja tämän korjauksen jälkeen järjestelmää testataan uudelleen. Testauksen kohteena on yksittäinen komponentti, komponenttien yhdistelmä mukaan luettuna integraatiot ohjelmiston kanssa, tai koko järjestelmä. (Graham ym. 2008, 24-25; Kasurinen 2013, 47.)

Testauksen tuloksista tehtyjä raportteja tarkastellaan vertaamalla testaustuloksia läpäisyvaatimuksiin (Graham ym. 2008, 25). Läpäisyvaatimukset tarkoittavat Grahamin ja kumppaneiden mukaan hankkeen omistajien kanssa hyväksyttyjä ja valittuja yleisiä ja määriteltyjä ehtoja, joiden mukaan arvioidaan, onko testausprosessi valmis. Tämän tarkoituksena on varmistaa, että testausta ei lopeteta ennen kuin määritellyt tavoitteet on saavutettu. (Graham ym. 2008, 22.)

Testauksen aikana syntyneet tulokset kirjataan testausraporttiin. Testauksen aikana kirjattuja havaintoja verrataan kyseisen tason testauksen läpäisyvaatimuksiin ja sen perusteella arvioidaan, onko testaus läpäissyt sille asetetut tavoitteet ja vaatimukset. (Graham ym. 2008, 25.)

Hyväksymiskriteereiden arvioinnissa on Grahamin ja kumppaneiden mukaan otettava huomioon seuraavat tehtävät: testaussuunnitelmassa määriteltyjen hyväksymiskriteerei-

den vertaaminen testausraporttiin, lisätestauksen ja -testien tarpeen arviointi ja samalla hyväksymiskriteereiden muuttamisen tarpeen arviointi ja testauksen yhteenvetoraportin kirjoittaminen. (Graham ym. 2008, 25.)

Testausraportti on tiedosto, johon on kirjattu kronologisessa järjestyksessä toteutetut testit ja niistä kirjatut, riittävän yksityiskohtaiset tiedot (Graham ym. 2008, 24). Huolellisesti täytetystä testausraportista voidaan nähdä, mitä testejä on suoritettu, mitä niissä on ilmennyt ja mihin toimenpiteisiin havaintojen vuoksi on ryhdytty. Testausraporttiin kirjataan, mikäli testi on aiheuttanut korjaustoimenpiteitä ja vaatii tämän vuoksi uudelleen testaamisen, vahvistustestauksen. Tätä dokumenttia on käytännöllistä verrata testaustason hyväksymiskriteereihin ja tämän perusteella kyetään arvioimaan, onko testaus saavuttanut hyväksyttävän tason ja täyttyvätkö ohjelmalle asetetut vaatimukset. (Graham ym. 2008, 25.)

Syitä lisätestauksen tarpeelle tai lisätestien kirjoittamiselle ja toteuttamiselle Graham kumppaneineen mainitsee neljä. Testausraportin ja hyväksymiskriteerien vertailu voi osoittaa, että kaikkia suunniteltuja testejä ei ole esimerkiksi tehty. Toiseksi, vertailu voi osoittaa, että testauksella ei ole saavutettu sellaista kattavuutta, jota on testaussuunnitelmassa tavoiteltu, jolloin tarve lisätestaukselle ja uusille testitapauksille syntyy. Kolmanneksi, tarve lisätestaukselle voi syntyä siitä, että projektin riskit ovat syystä tai toisesta kasvaneet. (Graham ym. 2008, 25.)

Hyväksymiskriteerien muuttaminen voi tulla kyseeseen, jos kriteereitä täytyy laskea siksi, että liiketoiminnalliset riskit tai projektin riskit nousevat hankkeen tärkeydessä niin paljon, että projekti on tärkeää saada valmiiksi. Hyväksymiskriteerien muuttamisesta tulee kuitenkin sopia hankkeen omistajien kanssa. (Graham ym. 2008, 25.)

Loppuraportti sisältää tiedot siitä, mitä on testattu ja testauksesta saadut lopputulokset. Loppuraportti sisältää myös arvioinnin siitä, kuinka hyvin testitapaukset ovat soveltuneet hyväksymiskriteerien testaamiseen ja onko tulos ollut odotetun kaltainen. Testauksen loppuraportti kirjoitetaan hankkeen omistajille ja raportin tarkoituksena on taata, että päätöksentekovastuussa olevat saavat ohjelmiston testauksesta ja sen lopputuloksesta vaadittavat tiedot päätöksentekoa varten. (Graham ym. 2008, 25.)

2.6 Miksi testaaminen epäonnistuu?

Testaamisen epäonnistumiselle on useampia syitä. Valsta (Valsta 2016) esittää opintokalvoissaan kaksi syytä, jotka johtavat testauksen epäonnistumiseen. Nämä ovat planning- ja design -virheet, joista kumpikin jakaantuu vielä kahteen virheeseen. (Valsta 2016.)

Planning -virheitä ovat väärät tuoteominaisuudet ja tuoteominaisuudet, jotka eivät tuota kilpailuetua. Väärillä tuoteominaisuuksilla tarkoitetaan ominaisuuksia, joita kukaan ei ole toivonut, tai niiden kehittämiselle ei ole varattu tarpeeksi resursseja. Väärät tuoteominaisuudet voivat olla myös ominaisuuksia, jotka eivät tyydytä käyttäjän tarpeita. Nämä johtuvat tietämättömyydestä toimintaympäristöstä. (Valsta 2016.)

Design -virheet liittyvät ohjelman tekniseen suunnitteluun. Virheet voivat johtua siitä, että ohjelma on teknisesti huonosti suunniteltu, tai se ei täytä vaatimuksia, joita sille on asetettu. Design -virheeksi katsotaan myös korkea tuotteen hinta, joka ei vastaa asiakkaan tarpeita ja arvoja. (Valsta 2016.)

Katara, Vuori ja Jääskeläinen (Katara ym. 1.9.2016) esittävät, että virheitä syntyy ohjelmistokehityksessä viidessä vaiheessa. Nämä ovat vaatimusmäärittely, suunnittelu, koodaus, dokumentointi ja muokkaukset. Määrällisesti virheitä syntyy eniten heidän mukaansa suunnitteluvaiheessa, toiseksi eniten vaatimusmäärittelyn yhteydessä ja koodaaminen tulee vasta kolmantena. Vähiten virheitä syntyy dokumentoinnin yhteydessä. (Katara ym. 1.9.2016.)

Graham ja kumppanit (Graham ym. 2008, 5) mainitsevat puolestaan kolme syytä virheille ja nämä ovat yhteisiä Kataran ja kumppaneiden (Katara ym. 1.9.2016) nimeämille syille. Virhelähteet ovat virheet vaatimusmäärittelyssä, virheet suunnittelussa ja virheet ohjelman koodaamisen aikana. (Graham ym. 2008, 5.)

Katara ja muut (Katara ym. 1.9.2016) luettelevat puolestaan seitsemän syytä virheiden lähteille. Nämä ovat väärät käsitykset halutusta toiminnasta, ympäristötekijät, muutokset muussa ohjelmistossa, toteutusvirheet, huonolaatuiset moduulit, väärä käännösyksikkö ja kääntäjien viat.

Väärät käsitykset halutusta toiminnasta johtuvat informaatiosta. Tiedonlähteet voivat olla vääriä, tieto voi olla vanhaa, tai mahdollisesti tieto voidaan ymmärtää useammalla eri tavalla. Dokumentaatio voi olla puutteellista tai vanhentunutta, konteksti voi olla virheellinen ja lisäksi tieto voi muuttua matkan varrella. (Katara ym. 1.9.2016.)

Ympäristötekijöillä tarkoitetaan erilaisia laiteympäristöjä, ohjelmistoja, käyttäjien oikeuksia, muita ohjelmistoja ja päivityksiä. Näillä kaikilla on mahdollisuus vaikuttaa ohjelman toimintaan. (Katara ym. 1.9.2016.)

Muutokset muussa ohjelmistossa tarkoittaa muutoksia muihin moduuleihin tai pääohjelmistoon ja puutteellista muutoksenhallintaa. Muutokset jossakin toisessa ohjelman osassa

voivat aiheuttaa vaurioita, jotka estävät ohjelman toista osaa toimimasta oikein. (Katara ym. 1.9.2016.)

Toteutusvirheet pitävät sisällään virheitä koodauksessa, konfiguroinneissa ja algoritmeissa. Lisäksi jonkun toisen tekemät muutokset voivat vaikuttaa odottamattomasti ohjelman toimintaan. Toteutusvirheisiin kuuluvat myös ohjelman käyttämien resurssien virheet. (Katara ym. 1.9.2016.)

Huonolaatuiset moduulit ovat jonkin toisen osapuolen tuottamia ohjelmistomoduuleita ja nämä aiheuttavat yhteensopivuusongelmia testattavan ohjelman kanssa. Syynä voivat olla paitsi yhteensopivuuden puuttuminen, myös moduuleissa olevat viat. (Katara ym. 1.9.2016.)

Väärä käännösyksikkö liittyy versionhallintaan ja konfiguraation hallintaan. Käännösyksikkö ei tässä tapauksessa sovellu tarkoitukseen ohjelmakoodin kirjoittamisen näkökulmasta ja seurauksena on virhe, joka aiheuttaa testaamisen epäonnistumisen. (Katara ym. 1.9.2016.)

Vikoja voi olla myös kääntäjissä. Lähdekoodi itsessään voi olla virheetöntä, mutta mikäli kääntäjässä on vikaa, tuloksena on virheellinen konekoodi, joka aiheuttaa ohjelman virheellisen toiminnan. (Katara ym. 1.9.2016.)

3 Testausautomaatio

Testausautomaatiolla on useampia määritelmiä riippuen sovelluskehityksen edustajasta. Kirjassaan Implementing Automated Software Testing Dustin, Garret ja Gauf (Dustin, Garret & Gauf 2009, 34-36) mainitsevat, että testausautomaatio voidaan määritellä testivetoiseksi ohjelmistokehitykseksi, yksikkötestaukseksi, nauhoittavaksi testausohjelmistoksi, tai räätälöityjen testiketjujen kehittämiseksi ohjelmointikielillä, joista kirjoittajat mainitsevat seuraavat kielet: Pearl, Python ja Ruby. Kirjoittajat katsovat kaikkien noiden käsitteiden täyttävän testausautomaation käsitteen vaatimukset. (Dustin, Garret & Gauf 2009, 34-36.)

Cockburn ja myös Graham kumppaneineen pitävät testausautomaatiota sopivana työkaluna regressiotestaukseen, missä testitapaukset toistuvat samanlaisina ja varmistetaan, että ohjelmisto ei ole muuttunut uusien ominaisuuksien rakentamisen ja lisäämisen jälkeen (Cockburn 2007, 224; Graham ym. 2008, 49-50). Sovelluskehityksen nopeutuessa testausautomaatiota käytetään nykyään myös uuden ohjelmakoodin testaamisessa erityisesti ketterissä menetelmissä (Gupta 2016). Testausautomaation ajama testauskoodi kirjoitetaan samaan aikaan kuin ohjelmakoodi, jota sillä on tarkoitus testata (Gupta 2016).

Cockburn mainitsee myös, että ohjelmistoissa on osia, joiden automaattinen testaaminen on vaikeaa. Tällainen automaatiotestauksen työkaluilla vaikeasti testattava elementti on hänen mukaansa graafinen käyttöliittymä (Cockburn 2007, 224; Gupta 2016). Testausautomaatiolta puuttuu myös inhimillinen hahmotuskyky, jolla on merkitystä käyttöliittymän käytettävyyden arvioinnissa ja esimerkiksi värimaailman hahmottamisessa (Gupta 2016).

Testausautomaatio ei ole ratkaisu kaikkeen. Markkinoilla on useita järjestelmiä, joilla automaattitestausta voi tehdä, mutta niistä yksikään ei sovellu kaikilla ohjelmointikielillä, kaikissa käyttöjärjestelmissä tai -ympäristöissä toteutettavaan testaamiseen (Dustin, Garret & Gauf 2009, 112-113; Gupta 2016). Testausautomaatio auttaa testaajia työssään, mutta kaikkea testaamista ei voida automatisoida (Gupta 2016).

Tavallisimmat harhakäsitykset Dustinin, Garretin ja Gaufin mukaan ovat testausautomaation kyky luoda automaattisesti testaussuunnitelma, yhden testaustyökalun sopivuus kaikkeen testaamiseen, testaamiseen käytetyn energian ja työn väheneminen, testausaikataulun automaattinen lyheneminen, helppokäyttöisyys, kaikkien testien automatisointi, täydellinen testauksen kattavuus, testauksen nauhoituksen ja uudelleenajamisen rinnastaminen automaatiotestaukseksi, testauksen tavoitteen, siis vikojen löytämisen, unohtaminen ja keskittyminen järjestelmätestaukseen yksikkötestauksen sijasta. (Dustin, Garret & Gauf 2009, 111-120; Gupta 2016.)

3.1 Testausautomaation käytöstä saatavat hyödyt

Testausautomaation onnistunut käyttö laskee testaukseen tarvittavaa aikaa ja kustannuksia. Dustin, Garret ja Gauf toteavat, että vaikka testaukseen käytettyjen kustannuksien ei usein ajatella liittyvän sovellusten kehittämiseen, todellisuudessa testaukseen käytetty aika ja testauksen kustannukset ovat verrattavissa varsinaiseen kehitystyöhön liittyviin ajallisiin ja rahallisiin kustannuksiin (Dustin, Garret & Gauf 2009, 59-60). Gupta (Gupta 2016) mainitsee testausautomaation eduiksi luotettavuuden, toistettavuuden, nopeuden, uudelleen käytettävyyden, monipuolisuuden, kustannusten hallinnan ja ohjelmoitavuuden (Gupta 2016).

Luotettavuus tarkoittaa Guptan (Gupta 2016) mukaan sitä, että testausautomaatio toteuttaa testin aina samalla tavoin. Uudelleenkäytettävyydellä puolestaan tarkoitetaan sitä, että samaa testiä voidaan käyttää erilaisissa käyttöympäristöissä. Toistettavuuden avulla voidaan testata puolestaan sitä, kuinka testattava ohjelma käyttäytyy, kun samaa toimintoa toistetaan useita kertoja. Testausautomaatio on huomattavasti nopeampi, kuin vastaavaa testiä suorittava ihminen. Monipuolisuus mahdollistaa sen, että testitapaukset voidaan suunnitella sellaisiksi, että ne kattavat kaikki testattavan ohjelman ominaisuudet. Testausohjelmisto toimii tämän lisäksi ympäri vuorokauden viikonpäivästä riippumatta, mikä alentaa kustannuksia. Ohjelmoitavuus tarkoittaa sitä, että testattavaksi voidaan ohjelmoida mutkikkaitakin liiketoiminnallisia tapahtumasarjoja. (Gupta 2016.)

Testausautomaatio parantaa myös ohjelmiston laatua. Tämä johtuu siitä, että testausautomaation avulla voidaan ajaa useampia testejä (Dustin, Garret & Gauf 2009, 73-74; Gupta 2016). Käytännössä tämä tarkoittaa, että testitapaukset voidaan ajaa automaattisesti, testauksen kattavuutta saadaan nostettua ja testaukseen käytettyä aikaa voidaan lisätä, sillä automaattitestaus toimii myös yöllä ja viikonloppuisin (Dustin, Garret & Gauf 2009, 73-75; Gupta 2016).

Lisäksi testausautomaatio parantaa ohjelmiston laatua vapauttamalla testaajien aikaa enemmän huomiota vaativiin tehtäviin, eikä testausohjelmisto ikävysty tai väsy, mikä voisi aiheuttaa inhimillisistä syistä johtuvia virheitä. Automaattitestaus on toistettavissa täysin samanlaisena, minkä lisäksi suorituskykyyn liittyvä testaus on luotettavampaa, kun se tehdään testausautomaatiota käyttäen. (Dustin, Garret & Gauf 2009, 73-75; Gupta 2016.)

3.2 Testauksen automatisointi

Dustin, Garret ja Gauf listaavat kuusi tärkeää tekijää, jotka ovat heidän mukaansa onnistumisen edellytyksiä testauksen automatisoinnin onnistumiselle. Nämä tekijät ovat vaatimusten tunteminen, automaattitestauksen strategian kehittäminen, jatkuva edistymisen

seuraaminen ja tarvittavien muokkausten tekeminen, testausautomaation prosessien käyttöönotto ja oikeilla taidoilla varustettujen henkilöiden tuominen hankkeeseen (Dustin, Garret & Gauf 2009, 139).

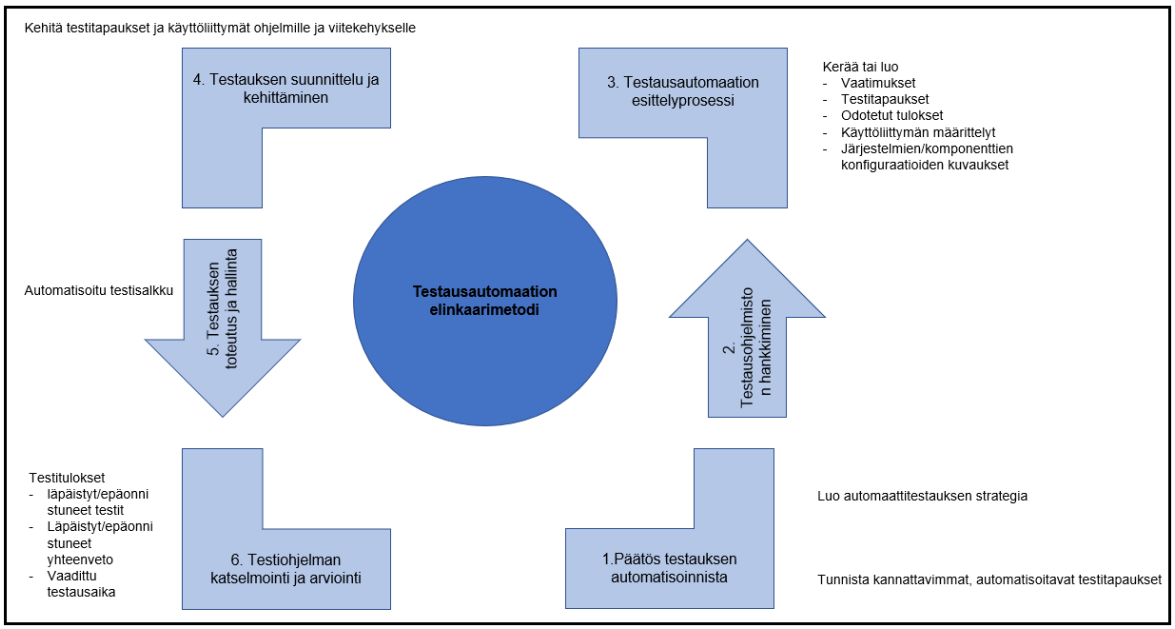
Vaatimusten tunteminen tarkoittaa testauksen automatisoinnin yhteydessä muutakin, kuin kehitettävän ohjelmiston vaatimusten tuntemista. Vaatimuksiin kuuluvat ohjelmiston suunnittelu, arkkitehtuurivaatimukset, projektin vaatimukset, tieto testausprosesseista, yleinen tieto hankkeesta ja testausautomaatiolle asetettavat vaatimukset (Dustin, Garret & Gauf 2009, 142-143). Tähän liittyy myös ymmärrys asiakkaan, siis ohjelmiston käyttäjän, liiketoiminnan prosesseista ja testauksen ja testausvaatimusten laajuus. (Dustin, Garret & Gauf 2009, 142-143.)

Automaattitestauksen strategian kehittäminen on mahdollista, kun vaatimukset on ensin määritelty (Dustin, Garret & Gauf 2009, 169-171). Strategiassa kuvataan testauksen laajuus, tavoitteet, testauksen viitekehys ja lähestymistapa, käytettävät testausvälineet, testiympäristö, henkilöstöä koskevat vaatimukset ja testauksen aikataulu (Dustin, Garret & Gauf 2009, 169-171). Automaattitestauksen strategia on dokumentti, jota tullaan käyttämään testauksen ohjaamisessa.

Automaattitestauksen viitekehyksen testaaminen on vaiheena seuraavaksi (Dustin, Garret & Gauf 2009, 210). Viitekehyksen testaamisen Dustin, Garret ja Gauf ovat jakaneet neljään osaan. Ensimmäiseksi tarkistetaan, että viitekehys vastaa määriteltyjä vaatimuksia ja ominaisuudet toimivat odotetulla tavalla (Dustin, Garret & Gauf 2009, 210). Tämän jälkeen tehdään vertaisarviointi suunnittelusta, testitapauksista ja kehityksestä (Dustin, Garret & Gauf 2009, 210). Seuraavaksi tarkastetaan ohjelmiston yksikkötestauksen kattavuus (Dustin, Garret & Gauf 2009, 210). Viimeisenä vaiheena on katselmointi asiakkaan kanssa (Dustin, Garret & Gauf 2009, 210).

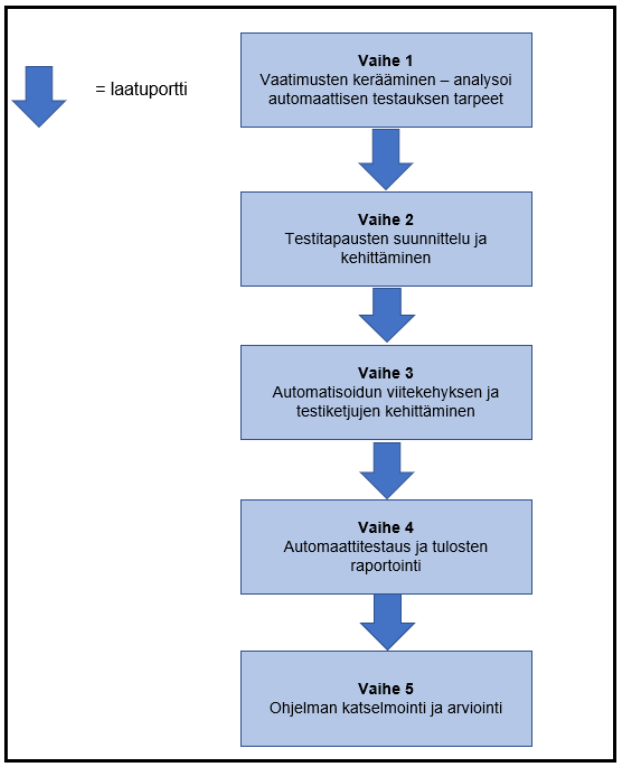
Testauksen edistymisen seuraamisella pyritään välttämään viivästyneet aikataulut ja hankkeen epäonnistuminen (Dustin, Garret & Gauf 2009, 230-233). Testauksen edistymisen seurannalla pyritään välttämään ohjelmaan jääviä vikoja, mittaamaan testauksen tehokkuutta ja lisäksi pitämään nämä kaksi asiaa hallinnassa. Seurannalla päästään käsiksi myös testauksessa ilmenneisiin vikoihin ja niiden alkuperäisiin syihin on tämän vuoksi mahdollista päästä käsiksi. (Dustin, Garret & Gauf 2009, 230-233.)

Dustin, Garret ja Gauf kuvaavat kirjassaan kuusivaiheisen testausautomaation elinkaarimetodin, joka on otettu käyttöön monessa alan yrityksessä (Kuva 4). Metodin ajatuksena on, että jokaisen vaiheen välissä on niin sanottu ”virtuaalinen portti”, jonka olemassaolo takaa sen, että seuraavaan vaiheeseen mennään vasta sitten, kun edellinen vaihe täyttää testausautomaatiolle asetetut vaatimukset. (Dustin, Garret & Gauf 2009, 255-257.)



Kuva 4. IDT:n muokkaamaa testausautomaation elinkaarimetodia mukaillen (Dustin, Garret & Gauf 2009, 257)

Varsinainen testausautomaatioon liittyvä prosessi on viisivaiheinen (Kuva 5). Jokaisen vaiheen välissä on aiemmin mainittu laatuportti ja nämä vaiheet ovat vaatimusten kerääminen, testitapausten suunnittelu ja kehittäminen, automatisoidun testauksen viitekehyksen ja testiketjujen kehittäminen, varsinainen testaus ja sen tulosten raportointi ja ohjelman katselmointi ja arviointi. (Dustin, Garret & Gauf 2009, 267.)



Kuva 5. Testausautomaation prosessi mukaillen (Dustin, Garret &Gauf 2009, 267)

3.3 Automatisoitavien testitapausten suositusominaisuudet

Gupta (Gupta 2016) määrittelee kirjassaan ominaisuudet automatisoitaville testitapauksille, ja näiden ominaisuuksien huomioiminen takaa hänen mukaansa sen, että testausautomaatiosta saadaan hyötyä (Gupta 2016). Nämä ominaisuudet on listattu alla.

Testitapausten tulee olla lyhyitä ja yksinkertaisia (Gupta 2016).

Toistettavuus. Tämä tarkoittaa testitapauksen ajamista useita kertoja peräkkäin ilman, että testaaja joutuu vaikuttamaan asiaan (Gupta 2016; Katara ym. 1.9.2016).

Lujuus. Tämä tarkoittaa, että testitapauksen ajaminen tuottaa samanlaisen lopputuloksen, vaikka testiympäristössä tapahtuisi muutoksia (Gupta 2016; Katara ym. 1.9.2016).

Riittävyys. Testitapausten pitäisi kyetä vahvistamaan kaikki testiskenaarioon liittyvät vaatimukset (Gupta 2016).

Kattavuus. Testitapausten pitäisi kattaa mahdollisuuksien mukaan kaikki tärkeimmät liiketoiminnan käyttötapaukset (Gupta 2016).

Selkeys. Kaikkien ilmaisujen pitää olla helposti ymmärrettävissä (Gupta 2016).

Tehokkuus. Tehokkuus tarkoittaa, että testitapaukset pitää kyetä ajamaan kohtuullisessa ajassa (Gupta 2016).

Itsenäisyys. Testitapaukset pitää kyetä ajamaan itsenäisesti, tai mikäli ne ovat osa testitapausten kokonaisuutta, ne pitää kyetä voida ajamaan missä tahansa järjestyksessä. Automatisoitujen testitapausten riippuvuutta toisistaan pitäisi välttää. (Gupta 2016.)

Virheiden käsittely. Mikäli testiä ajettaessa tulee odottamaton virhe, pitää testitapaus kyetä käsittelemään asianmukaisesti (Gupta 2016).

Testausraportit. Testausraporttien pitää kuvata tarkat tiedot ajetusta testitapauksesta (Gupta 2016).

Muokattavat kommentit. Jos testitapaus epäonnistuu, on testitapaukseen oltava mahdollisuus liittää muokattavia kommentteja niistä syistä, joiden vuoksi testitapaus on epäonnistunut. Tämä sisältää sanalliset kuvaukset ja kuvakaappaukset. (Gupta 2016.)

Ylläpidettävyys. Testitapausten tulee olla helppoja ymmärtää, muokata ja laajentaa (Gupta 2016).

Jäljitettävyys. Testitapaukset pitää kyetä jäljittämään niin manuaalisiin testitapauksiin, vaatimusmäärittelyihin, kuin virheisiinkin (Gupta 2016).

Automaatiotestauksella testattavalle ohjelmalle on myös omat vaatimuksensa, jotta testausautomaation käyttö on mielekästä (Katara ym. 1.9.2016). Katara ja muut (Katara ym. 1.9.2016) luettelevat listan ominaisuuksia, jotka mahdollistavat ohjelman testaamisen automaation avulla.

Ohjelman tulee olla yleiseltä rakenteeltaan selkeä ja yksinkertainen. Monimutkaista ja sekavaa rakennetta on vaikeaa, ellei mahdotonta testata automaation avulla (Katara ym. 1.9.2016).

Ohjelma pitää koodata rakenteisesti, modulaarisesti ja selkeästi hahmotettavalla tavalla. Koodin kannattaa olla helposti ymmärrettävissä, jotta testattavat ominaisuudet ovat paremmin tunnistettavissa. Suositus myös on, että resurssitietoja ei kirjoiteta ohjelmakoodiin, vaan ne kuvataan tiedostoissa, jotta testiversioiden tekeminen on mahdollista. (Katara ym. 1.9.2016.)

Ohjelman arkkitehtuurille asetettuja vaatimuksia on kaksi. Modulaarisuus, mikä mahdollistaa yksittäisten komponenttien testaamisen erikseen, ja mahdollisuus käyttöliittymän ohittamiseen ohjelmalogiikan testaamisen yhteydessä. (Katara ym. 1.9.2016.)

Ohjelman käyttöliittymän suunnittelun näkökulmasta vaatimuksia automaatiotestaukselle on useampia. Käyttöliittymä pitää suunnitella sellaisilla teknologioilla, että testausautomaation on mahdollista tunnistaa käyttöliittymäkomponentit ohjelmatietojen avulla. Lisäksi tietojen syöttäminen ja lukeminen käyttöliittymän kentistä API:n (Application Programming Interface) avulla pitäisi olla mahdollista. (Katara ym. 1.9.2016.)

Yhtenä vaatimuksena on myös tietokantojen standardointi. Datan hakeminen niistä on helpompaa. Lisäksi testidata pitäisi kyetä voida luomaan ilman sovellusohjelman käyttöä. (Katara ym. 1.9.2016.)

3.4 Testausautomaation puutteet

Korkeat aloituskustannukset. Testausautomaatio on ohjelmisto, joka pitää hankkia ja sen käyttö vaatii koulutusta. Hankinta- ja koulutuskustannukset ovat korkeat, minkä vuoksi testausautomaation käytön suunnittelussa on harkittava, kannattaako sen hankinta ja

käyttö. Hankkeet, joissa ohjelmiston suuri koko tuottaa suuren määrän testitapauksia, on mielekäs automatisoida. (Gupta 2016; Katara ym. 1.9.2016.)

Ylläpitokustannukset ovat korkeat. Testausautomaatio toimii vain, jos testiskriptejä päivitetään ja ylläpidetään ajan tasalle säännöllisesti ohjelman päivittyessä. Tämä vaatii automaatiotestaajan työtä ja työaikaa. Testiskriptien päivittämisen edellytyksenä on ohjelmaan tehtyjen muutosten huolellinen dokumentaatio. (Gupta 2016; Katara ym. 1.9.2016.)

Testausautomaatiolla ei ole inhimillistä hahmotuskykyä. Testausautomaatio ei kykene arvioimaan käyttöliittymän ulkoasua ja selkeyttä, värimaailmaa ja luettavuutta samalla tavoin kuin ihminen. Tästä syystä testausautomaatio ei ole tarkoituksenmukainen väline graafisen käyttöliittymän testaamiseen. (Gupta 2016; Katara ym. 1.9.2016.)

Testausautomaatio ei sovellu käytettävyystestaukseen. Testausautomaatiolla voidaan kyllä testata, että ohjelma toimii määritellyllä tavalla. Sen sijaan se ei kykene vastaamaan kysymykseen, vastaako ohjelma käyttäjän tarpeisiin. (Gupta 2016.)

Testausautomaatio vaatii koulutettuja ja osaavia käyttäjiä toimiakseen halutulla tavalla. Se ei ole helppo käyttää, sillä automaatiotestaajilta odotetaan kykyä kirjoittaa automatisoituja testiskriptejä luotettavasti. (Gupta 2016; Katara ym. 1.9.2016.)

Testausautomaatiolla ei voida testata kaikkea. Testiskriptien ylläpitäminen vaatii työtä ja mikäli täydelliseen automatisointiin pyrittäisiin, loisi se mittavan testiskriptien tietokannan, jota pitäisi ylläpitää ja päivittää säännöllisesti. Tämä työ veisi aikaa huomattavasti ja vähentäisi testaamiseen käytettävää aikaa, minkä seurauksena testausautomaation käytöstä saatavat hyödyt menetettäisiin. (Gupta 2016; Katara ym. 1.9.2016.)

Lisäksi testausautomaation käyttö ei välttämättä lisää tehokkuutta. Tämä johtuu siitä, että testiskriptien kirjoittaminen vaatii aikaa ja ajoon käytetty aika suhteessa testiskriptin ylläpitoon vaadittaviin kustannuksiin voi olla kannattamaton. Harvoin ajettavia testejä ei tästä syystä kannata automatisoida. (Katara ym. 1.9.2016.)

Merkittävä haaste testausautomaation käytössä on myös se, että ohjelmisto pitää suunnitella ja tehdä niin, että testausautomaatio voi testata sen (Katara ym. 1.9.2016). Käytännössä tämä tarkoittaa, että testausautomaation pitää kyetä ohjaamaan testattavaa ohjelmaa ja lisäksi saamaan tiedon siitä, missä tilassa testattava ohjelma on. Lisäksi testausautomaation pitäisi kyetä hakemaan käyttöliittymärajapinnoista ja tietokannoista dataa. (Katara ym. 1.9.2016.)

Testausautomaatio on itsessään ohjelmisto, jonka käyttö vaatii ohjelmointia, dokumentointia ja suunnittelua, testausta ja ylläpitoa. Nämä kaikki vaativat taitoja. Lisäksi testausautomaatio pitää kyetä testaamaan. (Katara ym. 2016.)

4 Ketterät menetelmät

Alistair Cockburn määrittää ketteryyden tehokkuutena ja liikkuvuutena (Cockburn 2007, 222). Jim Highsmith puolestaan on määritellyt käsitteen kirjassaan Agile Project Management siten, että ketteryydellä tarkoitetaan kykyä sekä luoda muutosta, että vastata siihen voiton saavuttamiseksi muuttuvassa liiketoimintaympäristössä (Highsmith 2010, 13). Muutos voi Highsmithin mukaan olla yrityksen itsensä luomaa, mutta se voi johtua myös kilpailijoista tai asiakkaista (Highsmith 2010, 13).

Muutoksen tai jopa kaaoksen luominen tai niihin vastaaminen vaatii innovaatioita. Esimerkkeinä innovaatioista Highsmith mainitsee uudet tuotteet, tuotteiden muokkaaminen yhä vaativampaa ja kapeampaa kuluttajakuntaa varten, uusien myyntikanavien kehittäminen ja tuotekehitykseen käytettävän ajan pienentäminen. (Highsmith 2010, 13.)

Kasurinen kuvaa kirjassaan Ohjelmistotestauksen käsikirja (Kasurinen 2013, 25) ketteriä menetelmiä niin, että kaikki sellaiset työmenetelmät, jotka painottavat kommunikaatiota kaikkien kehitystyössä mukana olevien välillä, yhteistyötä asiakkaiden kanssa ja ohjelmistossa olevia toiminnallisuuksia enemmän kuin täydellistä ennakkosuunnittelua tai sopimuksella tehtyjä määrittelyjä, ovat ketteriä menetelmiä. (Kasurinen 2013, 25.)

Ketterien menetelmien ero perinteisiin ohjelmistokehityksen menetelmiin on määritelty myös niin, että jokaisen, lyhyen sprintin tuloksena syntyy asiakasorganisaatiolle valmista ohjelmaa, jolla on myös asiakkaalle arvoa tuottavia ominaisuuksia. Ketterässä kehityksessä ohjaavana ajatuksena on, ettei yksikään ominaisuus ole valmis, ennen kuin se on testattu järjestelmässä. (ITSQB 2014, 19.)

4.1 Tietojärjestelmähankkeista yleisesti

Riippumatta toteutustavasta uuden ohjelmiston käyttöön ottaminen tarkoittaa ohjelmistoa hankkivalle organisaatiolle aina toiminnan muutosta, joka on välttämätön tehokkaampien toimintatapojen käyttämiseksi. Samaan aikaan se koetaan kuitenkin uhkana, koska se tarkoittaa työtapojen muuttumista. Hankkeen onnistumisen takaamiseksi järjestelmää suunnittelevan ja hankkivan organisaation henkilöstö tulee ottaa mukaan hankkeeseen ja johdon tulee sitoutua siihen. (Williams 2014, 157- 160.)

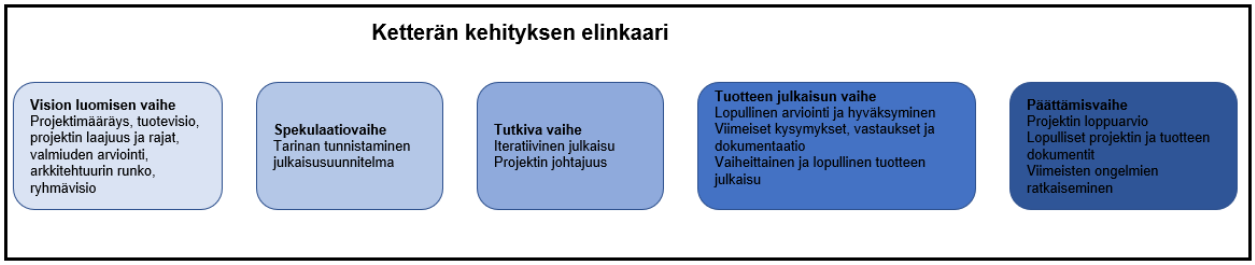
Uusien sovelluksien tuottaminen tapahtuu projekteina, mutta näiden projektien pituus ja laajuus riippuu siitä, kuinka suuresta tuotteesta on kysymys. Scot Berkun esittää kirjassaan Projektinhallinnan taito, että projektit noudattavat kolmasosien sääntöä riippumatta siitä, mitä metodia projektin hallinnassa käytetään. Nämä kolme osaa ovat suunnittelu, ohjelmointi ja testaus. (Berkun 2006, 33.)

Berkun esittää, että suurten projektien jakaminen pienempiin osaprojekteihin vähentää hankkeeseen kohdistuvia riskejä. Berkunin mukaan osaprojektien pienentäminen mahdollistaa paremman hallinnan ja lisää projektinhallinnan kykyä kohdata muutoksia. Nämä osaprojektit noudattavat kuitenkin aiemmin kuvattua kolmasosien sääntöä. (Berkun 2006, 35.)

Adrian Payne esittää kirjassaan Handbook of CRM: Achieving Excellence Through Customen Management huomion, että tietojärjestelmähanketta toteuttava organisaatio hyötyy siitä, että eri toimintayksiköiden väliset raja-aidat puretaan ja eri toiminnoista vastaavat, kehityshankkeeseen osallistuvat henkilöt tekevät yhteistyötä (Payne 2005, 353). Hän tarkoittaa tällä sitä, että nykyaikainen ohjelmisto palvelee useamman eri käyttäjäryhmän tarpeita ja näiden yhteen sovittamisessa tarvitaan yhteistyötä tietojärjestelmää hankkivan organisaation sisälläkin. (Payne 2005, 353.)

4.2 Ketterä projektinhallinta

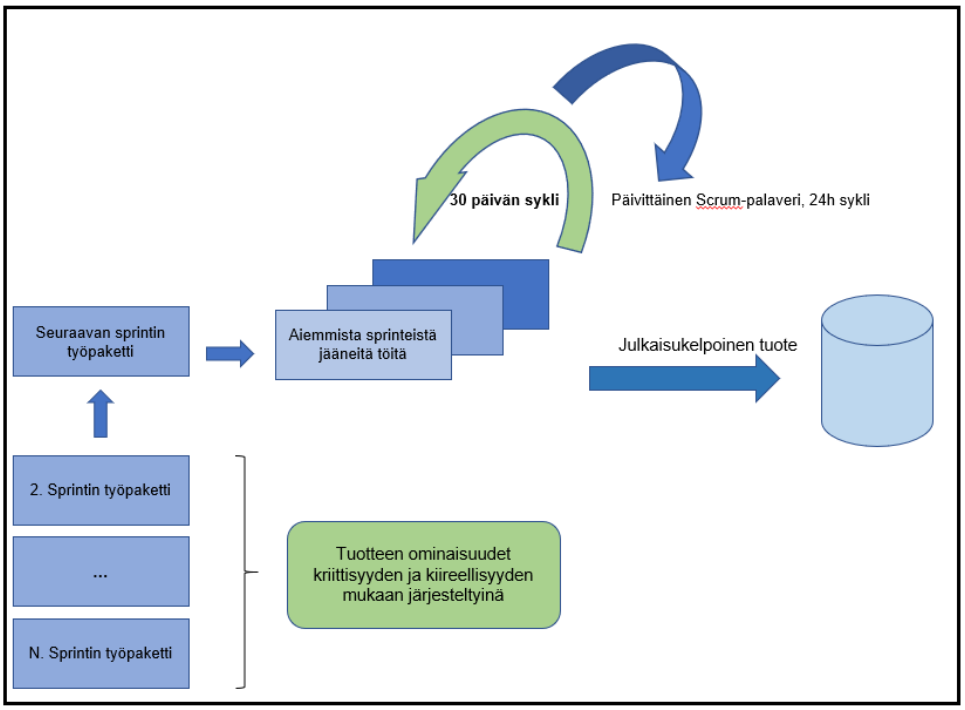
Highsmith jakaa ohjelmistoprojektin elinkaaren viiteen, kuvassa 6. kuvattuun vaiheeseen, jotka ovat vision luomisen vaihe, spekulaatiovaihe, tutkiva vaihe, tuotteen julkaisun vaihe ja päättämisvaihe. (Highsmith 2010, 88.)



Kuva 6. Ketterän kehityksen elinkaari (Highsmith 2010, 88)

Ketterän kehityksen elinkaarimallin viitekehyksen tutkiva vaihe on se viitekehyksen osa, jossa sovelluksen koodaaminen, testaaminen ja julkaiseminen tapahtuu. Tutkivan vaiheen alle kuuluvat projektinhallinta, iteraationhallinta ja tekniset käytännöt. Projektinhallinta jakaantuu Highsmithin mukaan pitkän aikavälin julkaisunhallintaan ja yhteistyöhön viiteryhmien kanssa. Iteraationhallinta puolestaan käsittää iteraatioiden suunnittelun ja hallinnan, sekä kehitystyöstä vastaavan ryhmän johtamista. (Highsmith 2010, 203.)

Ohjelmistokehityksen ketteriä menetelmiä on useita ja kuvassa 7. kuvattu Scrum on niistä yksi. Scrum-malli on julkaistu vuonna 1995, mutta ketteriä menetelmiä on käytetty jo huomattavasti aikaisemminkin. IBM on soveltanut ketteriä menetelmiä jo 1950-luvun lopulla (Kasurinen 2013, 25).



Kuva 7. Scrum-malli Kasurista mukaillen (Kasurinen 2013, 25)

Alistair Cockburn listaa kirjassaan useita ketteriä sovelluskehityksen malleja. Nämä mallit ovat Adaptive Software Development, XP, Scrum, Crystal, Feature Driven Development, Dynamic System Development Method ja ”pragmatic programming”. (Cockburn 2007, 369.)

”Ketterän ohjelmistokehityksen julistuksessa” (Haikala & Mikkonen 2011, 44-45; Highsmith 2010, 16; Kasurinen 2013, 25) ketterien menetelmien puolestapuhujat määrittävät periaatteet ja arvot, joita ketterät menetelmät noudattavat. Ketterän ohjelmistokehityksen julistus on seuraava:

> ”Me etsimme parempia keinoja ohjelmistojen kehittämiseen tekemällä sitä itse ja auttamalla siinä muita. Tässä työssämme olemme päätyneet arvostamaan
> **Yksilöitä ja vuorovaikutusta** enemmän kuin prosesseja ja työkaluja
> **Toimivaa sovellusta** enemmän kuin kokonaisvaltaista dokumentaatiota
> **Asiakasyhteistyötä** enemmän kuin sopimusneuvotteluita
> **Muutokseen reagoimista** enemmän kuin suunnitelman noudattamista
> Vaikka oikeallakin puolella on arvoa, me arvostamme vasemmalla olevia enemmän.”
> (Highsmith 2010, 16).

Cockburn selittää kirjassaan julistuksen neljää arvoa sovelluskehityksen näkökulmasta. Ensimmäinen periaate tarkoittaa sitä, että ketterän kehityksen näkökulmasta hyvä vuorovaikutus ilman dokumentointia on parempi vaihtoehto, kuin kattava dokumentointi ja vihamielinen vuorovaikutus. Cockburn perustelee tätä sillä, että ihmisten vuorovaikutus

tuo esiin ohjelmassa olevia puutteita ja tarpeita, oli kyse vanhan ohjelman korjaamisesta tai uuden kehittämisestä. (Cockburn 2007, 371.)

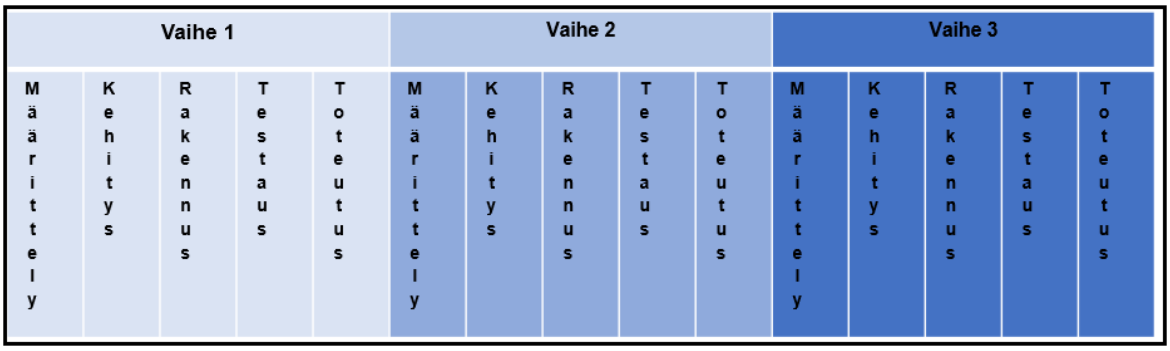
”Toimivaa sovellusta enemmän kuin kokonaisvaltaista dokumentaatiota” tarkoittaa puolestaan sitä, että paraskaan dokumentaatio ei merkitse mitään, jos itse ohjelma ei toimi. Määrittely- ja suunniteludokumentit ovat hyödyllisiä ohjelmiston kehittämisessä, mutta vasta ohjelman toteuttaminen kertoo, kuinka hyvin hanke oikeasti on onnistunut. (Cockburn 2007, 371.)

”Asiakasyhteistyötä enemmän kuin sopimusneuvotteluita” viittaa vuorovaikutukseen sovellusta kehittävän tiimin ja asiakkaan edustajien välillä. Ketterän kehityksen näkökulmasta asiakas ja asiakkaalle ohjelmistoa kehittävä sovellustiimi muodostavat yhden, keskenään vuorovaikutuksessa olevan tiimin, koska molempia tarvitaan onnistuneen ratkaisun tuottamiseksi. Yhteistyö mahdollistaa nopean vuorovaikutuksen ja päätöksenteon. (Cockburn 2007, 371-372.)

”Muutokseen reagoimista enemmän kuin suunnitelman noudattamista” tarkoittaa käytännössä kykyä nopeisiin muutoksiin. Ketterässä sovelluskehityksessä ei väheksytä suunnittelun merkitystä. Kyse on siitä, että ketterä sovelluskehitys mahdollistaa lyhyen aikajanan suunnitelmat, joiden aikana sovellustiimi tai tiimit rakentavat toimivan sovelluksen komponentin. Ajanjaksollisesti lyhyet iteraatiot mahdollistavat sen, että hankkeen omistajat voivat arvioida tärkeysjärjestystä kehitettävissä integraatioissa tai komponenteissa tarpeen vaatiessa uudelleen ja suunnitelmaa voidaan helposti päivittää. (Cockburn 2007, 372.)

Kuvassa 8. esitetty iteratiivinen sovelluskehityksen malli ei itsessään ole ketterä sovelluskehityksen malli. Ketterä sovelluskehitys perustuu iteraatioihin tai kehityssykleihin, joita voidaan kutsua myös sprinteiksi. Näiden iteraatioiden pituus voi vaihdella parista viikosta kolmeen tai neljään kuukauteen. Jokainen iteraatio muodostuu määrittelystä, kehityksestä, rakentamisesta, testaamisesta ja implementoinnista tai toteutuksesta. (Graham ym. 2008, 38-40; Kasurinen 2013, 25-26.)

Iteratiivinen sovelluskehityksen malli soveltuu myös sellaisiin tilanteisiin, joissa ei ole suurta hanketta takana, vaan vaatimuksia ja kehitystoiveita tulee kehitystiimille ajan kuluessa. Esimerkkinä Berkun mainitsee esimerkiksi verkkosivuston kehittämisen. Ne toteutetaan sitten sopivassa tilaisuudessa ja kokonaisuutta rakennetaan tällä tavoin paloittain. (Berkun 2006, 34.)



Kuva 8. Iteratiivinen sovelluskehityksen malli mukaillen (Graham ym. 2008, 38)

4.3 Ketterä sovelluskehitysprosessi

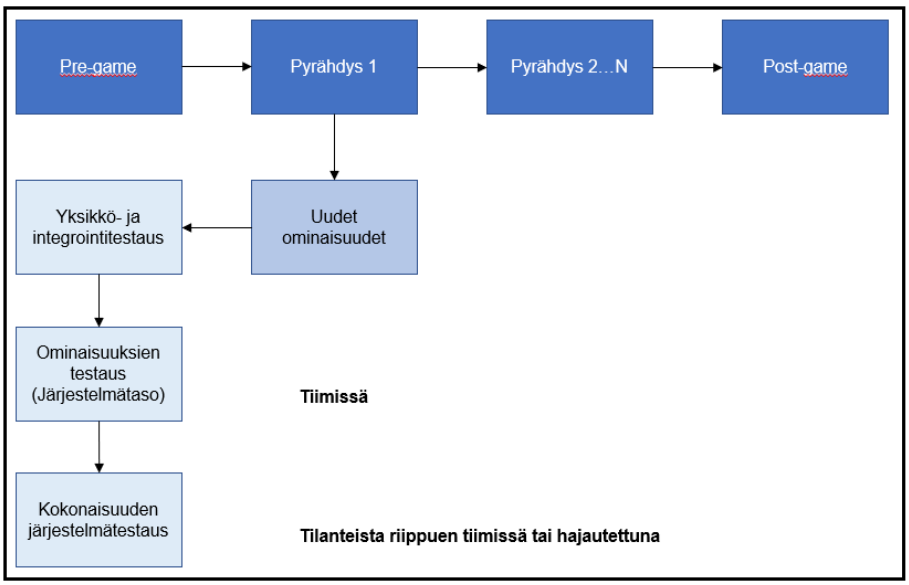
Alistair Cockburn esittää kirjassaan ketterien menetelmien käyttöön liittyvän toteamuksen. Hänen mukaansa kysymys siitä, voidaanko ketteriä menetelmiä käyttää, ei ole oikea. Oikea kysymys on, että kuinka säilyttää ketteryys vallitsevassa tilanteessa (Cockburn 2007, 222). Tällä hän viittaa siihen, että sovelluskehitystä tekevän tiimin koko vaikuttaa ketteryyteen. Mitä suurempi organisaatio hankkeessa on, sitä vaikeampaa ketteryyden säilyttämisestä tulee koko organisaation tasolla. (Cockburn 2007, 222; Highsmith 2010, 272-275).

Ketterä sovelluskehityksen prosessi perustuu lyhyisiin sprintteihin tai iteraatioihin, joiden tavoitteena on saada aikaan valmista, testattua ohjelmaa (ITSQB 2014, 19; Katara ym. 1.9.2016). Ketterässä sovelluskehityksessä testaajat ovat osa kehitystiimiä ja sovellusta rakentavien koodaajien tuotettua ominaisuuksia he ottavat ne testatakseen. Ensimmäiseksi tehdään yksikkötestaus, minkä jälkeen valmiit ohjelmiston osat integroidaan kokonaisuuteen ja tämän jälkeen tehdään järjestelmän testaaminen. (Katara ym. 1.9.2016.)

Meyer (Meyer 2014, 154) listaa joukon ketteriä käytäntöjä, jotka hän määrittelee erinomaisiksi ketterissä sovelluskehityksen hankkeissa. Ensimmäinen näistä on lyhyet iteraatiot. Toisena hän mainitsee jatkuvan integroinnin ja regressiotestausten testisalkun. Kolmas käytännöistä on niin sanottu suljetun ikkunan sääntö, joka tarkoittaa hänen mukaansa sitä, että kukaan hankkeessa oleva ei voi lisätä iteraatioon rakennettavaa aineistoa riippumatta henkilön statuksesta. Neljäs käytäntö on ehdottomat aikarajat, joista pidetään kiinni kaikissa olosuhteissa. Viides käytäntö on hankkeen omistajan rooli. Hänen tehtävänsä on määritellä, mitä rakennetaan ja missä vaiheessa. Kuudes käytännöistä on ketterien menetelmien perusajatus; toimivan ohjelman julkaiseminen. Seitsemäs hänen mainitsemistaan käytännöistä on tehtävätaulut, jotka tekevät hankkeen etenemisen näkyväksi. Kahdeksas käytäntö on kaikkien toimintojen testaaminen. (Meyer 2014, 154.)

4.4 Ketterä sovelluskehitys ja testaaminen

Ketterien menetelmien testausmallissa (Kuva 9) on kuvattu periaatteellisella tavalla, kuinka testaaminen ketterässä ohjelmistohankkeessa tapahtuu.

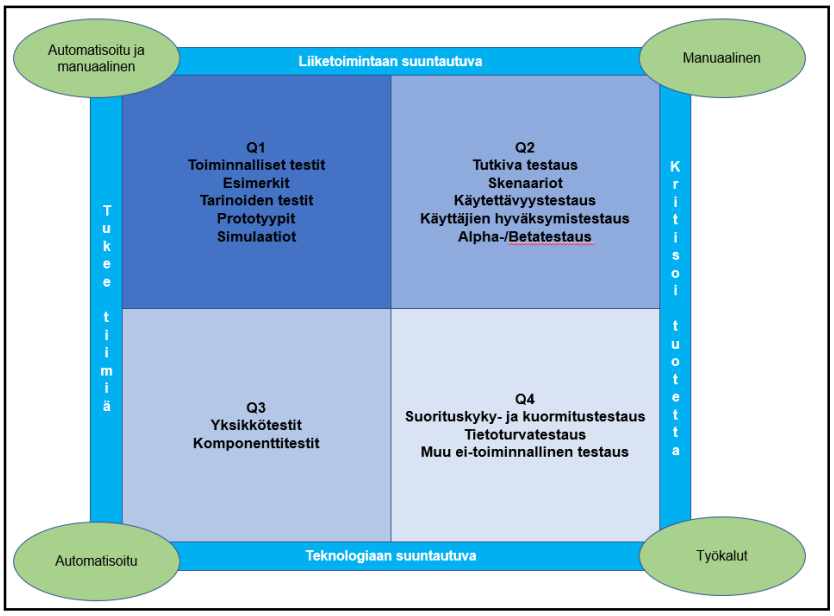


Kuva 9. Periaatteellinen ketterä testausmalli Katara ym. mukaillen (Katara ym. 1.9.2016)

Sprintin aikana rakennetut ominaisuudet testataan ensin yksikkö- ja integrointitestauksessa. Kun nämä testit on suoritettu onnistuneesti, siirrytään ominaisuuksien testaamiseen ja tämän vaiheen toteuttaa sovelluskehityksestä vastaava tiimi. Iteraation lopussa testataan järjestelmä kokonaisuudessaan siihen mennessä rakennettujen toiminnallisuuksien osalta. Tämä voidaan toteuttaa tiimissä tai hajautettuna. (Katara ym. 1.9.2016.)

Ketterien menetelmien testausmalli on kuvattu Kataran ja kumppaneiden aineistossa niin, että jokaisen pyrähdyksen tai sprintin alussa toteutettavat rakennustehtävät listataan työlistaan. Tämän jälkeen testaukselle luodaan omat tehtävät joko erikseen tai rakennettaviin toimintoihin liittyen. Testaus perustuu käyttäjätarinaan, jonka ei tarvitse olla raskaasti dokumentoitu, mutta tämä edellyttää hyvää kommunikaatiota tiimin sisällä. Toteutuksen tarkentuessa tarkentuu myös testauksen toteutus. Jos mahdollista, testaus pyritään automatisoimaan toistettavuuden helpottamiseksi. Tähän voidaan ottaa mukaan jo asiakkaan tai hankkeen omistajan edustajat. Pyrähdyksen aikana rakennettava toiminnallisuus on valmis vasta siinä vaiheessa, kun testaus on onnistunut. (Katara ym. 1.9.2016.)

Ketterän sovelluskehityksen testauksessa on käytössä myös nelikenttäinen malli (Kuva 10), joka ei sellaisenaan ole kaiken kattava, vaan tarkoitus on soveltaa sitä (Katara ym. 1.9.2016). Mallia voidaan käyttää kehitystiimin apuna tai tuotteen arvostelemiseen, toisaalta se voi olla myös teknologia- tai liiketoiminnallisesti suuntautunut. (Katara ym. 1.9.2016.)



Kuva 10. Ketterän testauksen nelikenttämalli Kataran ym. mukaillen (Katara ym. 1.9.2016)

Ketterän sovelluskehityksen ja ketterän testaamisen suhde on se, että yhdessä sprintissä pyritään rakentamaan toiminnallisuutta sen verran, että se ehditään myös testaamaan saman sprintin aikana (Katara ym. 1.9.2016). Käytännössä testaaminen alkaa heti, kun on jotakin valmista testattavaa, testaajat eivät siis odota koko sprintin julkaisun valmistumista (Graham ym. 2008, 38; Kasurinen 2013, 25-26; Katara ym. 1.9.2016).

Ketterän testauksen nelikenttämallin mukaan kvadrantissa yksi olevat testitapaukset olisivat testattavissa osin manuaalisesti, osin automaation avulla ja kvadrantissa kolme olevat yksikkö- ja komponenttitestit voitaisiin testata puhtaasti testausautomaatiota käyttäen. (Katara ym. 1.9.2016.)

Ketterässä testauksessa, joka hyödyntää testausautomaatiota, testausautomaation koodi kirjoitetaan samaan aikaan testattavan ohjelmakoodin kanssa (Gupta 2016). Tälle testaustavalle on tyypillistä se, että testien kehittämiseen käytettävä aika on lyhyempi, testien kehittäminen on samanaikaista itse ohjelmiston kehittämisen kanssa, testausautomaatio voidaan ottaa käyttöön suuremmassa osassa testitapauksia ja testaustiheyttä voidaan nostaa. (Gupta 2016.)

5 Testausautomaation käyttöönotto kohdeyrityksessä

Opinnäytetyön empiirisenä osana tulee olemaan kuvitteellinen käyttöönottotapaus, jossa ketterin menetelmin kehitetty ohjelmisto otetaan käyttöön asiakaspalvelua tuottavassa yrityksessä tai organisaatiossa. Testaus toteutetaan mahdollisuuksien mukaan testausautomaatiota käyttäen. Tutkittavan kohdeorganisaation sovelluskehityshankkeen sidosryhmiä ovat kohdeorganisaation eri toiminta- tai liiketoimintayksiköiden käyttäjät, niiden johto, organisaation hallinto ja organisaation asiakkaat, jotka voivat käyttää organisaation tarjoamaan palvelua omalta päätelaitteeltaan.

Hankkeen aikana eri sidosryhmien edustajat toimivat asiantuntijoina, jotka vastaavat vaatimusmäärittelystä, tietotarpeiden kokoamisesta, prosessikuvauksista ja ovat mukana käyttäjäkokemuksen suunnittelussa. Hankkeen omistajan edustajat ovat mukana myös suunnittelemassa testausta ja sen prioriteetteja.

Kohdeorganisaatio (jatkossa Organisaatio) aloitti asiakaspalvelijoiden käyttöön tarkoitetun tietojärjestelmän uusimisen palvelutoimintojen nykytilan kuvaamisella. Muuttuva toimintaympäristö ja muuttuvat tietotarpeet pakottavat tietojärjestelmän uusimiseen. Toisaalta taustalla on myös käyttöjärjestelmien uudistuminen ja tuen häviäminen vanhoilta käyttöjärjestelmiltä ja tähän liittyvä tietoturvariskien kasvaminen. Organisaation strategiaan kuuluu tietojärjestelmäuudistus, joka vastaa paremmin myös Euroopan tietosuoja-asetuksen (GDPR) esittämiin vaatimuksiin.

Organisaatio valitsi projektiryhmän omista sidosryhmistään. Projektiryhmään kuuluu edustajia kaikista järjestelmää käyttävistä ryhmistä, ja projektiryhmä työskentelee yhdessä sovellusta kehittävän ohjelmistotoimittajan kanssa. Merkittävä sidosryhmä tässä hankkeessa on Organisaation tietohallintopalveluiden edustaja. Tietohallinnon on tarkoituksena vastata ohjelmistotestauksesta tulevien ohjelmistopäivitysten yhteydessä Organisaation käyttöympäristössä.

Ohjelmistokokonaisuuden koko ja monimuotoisuus saivat hankkeen omistajan ja sidosryhmät harkitsemaan testauksen automatisointia ja sen mahdollisuuksia ohjelmiston kehittämisen ja myöhemmin ylläpidon aikana.

5.1 Ohjelmistohankkeen onnistumisen edellytykset

Hanke toteutetaan ketterillä menetelmillä ja jo alkuvaiheessa tehtiin päätös testauksen ottamiseksi mukaan päivittäiseen sovelluskehitystyöhön. Testaajat ovat mukana sovelluksen kehittämisessä suunnittelemalla testitapauksia samaan aikaan kehitettävän ohjelman kanssa vaatimusmäärittelyn pohjalta. (Kasurinen 2013, 20.)

Projektiryhmään valitut loppukäyttäjät edustavat ammattilaisia, jotka tuntevat Organisaation toimintaa ja ydintoimintojen prosesseja hyvin. He osaavat kuvata Organisaation nykytilan ja sen nykyiset prosessit, sekä kuvata halutun tavoitetilan. Tämä henkilöstö kuuluu Organisaation eri liiketoimintayksiköihin. (Williams 2014, 157- 160.)

Tavoitetilan kuvaamisen näkökulmasta loppukäyttäjien tulee kyetä määrittämään käsitteet ja vaatimukset yhteisesti sekaannusten ja päällekkäisyyksien välttämiseksi. Haasteena on, että eri käyttäjäryhmien käyttämä käsitteistö voi olla samasta asiasta hyvin erilainen, minkä lisäksi eri ammattiryhmillä on erilaiset tarpeet ohjelman käytölle. Mahdollisimman modulaarisen ja rakenteisen suunnittelun mahdollistamiseksi asiantuntijoina toimivien loppukäyttäjien tehtävänä on löytää yhteiset käyttökohteet ja tarpeet, mikä edellyttää toimivaa vuorovaikutusta eri ammattiryhmien kesken. (Payne 2005, 353.)

Tämän lisäksi vaatimusten määrittelyssä tulee ottaa huomioon jokaisen liiketoimintayksikön yksittäiset tarpeet. Niiden suunnittelussa tulee välttää päällekkäisyyttä käsitteiden ja toimintojen osalta muiden liiketoimintayksiköiden moduulien välillä. Haasteena on minimalistisen suunnittelun vaatimat taidot, joita Organisaation henkilöstöllä ei välttämättä ole. Tämän vuoksi ohjelmistotoimittajan edustajien ja Organisaation projektiryhmän jäsenten välillä tulee olla toimiva kommunikaatio, jossa kumpikin osapuoli kykenee ymmärtämään toistaan. (Haikala & Mikkonen 2011, 44-45; Highsmith 2010, 16; Kasurinen 2013, 25; Payne 2005, 353.)

Yhteisen ymmärryksen löytymiseen on kaksi keinoa, jotka noudattavat molemmat ketterien menetelmien periaatetta. Asiakasyhteistyö ohjelmistoyrityksen ja Organisaation välillä takaa, että ohjelmiston kehittäminen tapahtuu yhteisten käsitysten mukaan ja asiakkaan tarpeita vastaavasti. Tiiviillä yhteistyöllä kyetään myös reagoimaan nopeammin muutostarpeisiin, mikäli ohjelmistokehityksen yhteydessä vaatimukset muuttuvat tai ilmenee mahdollinen toisistaan poikkeava ymmärrys. (Haikala & Mikkonen 2011, 44-45; Highsmith 2010, 16; Kasurinen 2013, 25.)

Toinen on toimivan kommunikaation takaaminen. Kommunikaation tulee toimia kahdella tasolla, Organisaation projektiryhmän sisällä, mutta toisaalta myös Organisaation ja ohjelmistoyrityksen välillä. Toimiva kommunikaatio Organisaation sisällä varmistaa sen, että jokainen loppukäyttäjäryhmä saa käyttöönsä toimivan ohjelman, eikä ohjelmaa rakenneta yhden tai kahden käyttäjäryhmän ehdoilla muiden toimintojen kustannuksella. Toisaalta kommunikaation tulee toimia myös ohjelmistotoimittajan suuntaan, sillä se takaa onnistuneen yhteistyön. (Haikala & Mikkonen 2011, 44-45; Highsmith 2010, 16; Kasurinen 2013, 25.)

Mitä toimivalla kommunikaatiolla sitten tarkoitetaan? Bath ja McKay (Bath & McKay 2008, 355-359) listaavat joukon ominaisuuksia, jotka liittyvät toimivaan kommunikaatioon. Ensinnäkin jokaisen keskusteluun osallisen tulisi tietää oma roolinsa ja paikkansa. Ohjelmistokehityksessä tarvitaan monenlaisia taitoja. Tämä ei kuitenkaan riitä, vaan tämän lisäksi oma asiantuntijuus tulisi kyetä tuomaan myös ymmärrettävästi esille niin suullisesti, kuin kirjallisestikin. Vuorovaikutuksen sävyllä on heidän mukaansa myös suuri merkitys, sillä aggressiivinen ja avoimen vihamielinen tapa keskustella vähentää halua vuorovaikutukseen ja tekee siitä pahimmassa tapauksessa mahdotonta. (Beth & McKay 2008, 355-359.)

5.2 Testausautomaation määrittelyn onnistumisen edellytykset

Testausautomaation tulo mukaan projektiin nostaa projektin loppukustannuksia. Testausohjelmisto hankittiin pilvipalveluna palveluntarjoajalta ja testauksesta vastaava henkilöstö koulutetaan ohjelmiston käyttöön testausautomaation toimittajan puolesta. Tehtävään valituilta toivottiin aikaisempaa kokemusta joko ohjelmoinnista, ylläpidosta, testaamisesta tai mieluiten kaikista näistä. Automatisoidusta testauksesta vastaavaan henkilöstöön etsittiin henkilöstöä ensisijaisesti Organisaation tietohallinnosta näiden lähtötasovaatimusten vuoksi. (Gupta 2016; Katara ym. 1.9.2016.)

Testausautomaation hankinnan yhteydessä tehdään tarkka selvitys siitä, mitä sen avulla voidaan tehdä ja miten. Tämän selvityksen avulla tuotetaan tietoa siitä, mitkä ovat käytettävän automaation realistiset käyttökohteet ja tällä pyritään välttämään myös virheelliset odotukset, joiden seurauksena testausautomaation käyttö saattaa epäonnistua. Samalla pyritään tunnistamaan kannattavimmat testitapaukset ja luodaan automaattitestaukselle strategia. (Dustin, Garret & Gauf 2009, 139, 142-143.)

Testausautomaation hankinnan yhteydessä tehtävänä on selvittää, mitä työkalulta odotetaan ja mihin sitä käytetään, mutta myös missä laajuudessa. Testaajien taidot suhteessa valittavaan testausautomaatioon tulee ottaa myös huomioon. Eri testausohjelmat käyttävät erilaisia skriptauskieliä ja automaatio on kyettävä päivittämään ja ylläpitämään. Ohjelmistosuunnittelun yhteydessä tehtävän testitapausten suunnittelun aikana on tunnistettava ne tilanteet, joissa testausautomaatiota voidaan käyttää. Näitä edellytyksiä ovat riippumattomuus muista testitapauksista, usein toistuva testaus, saman lopputuloksen saaminen ympäristöstä riippumatta ja testitapausten helppo ylläpidettävyys. Käytännössä tämä tarkoittaa, että testattavasta toiminnallisuudesta tiedetään, ettei se muutu jatkuvasti. (Cockburn 2007, 224; Graham ym. 2008, 49-50; Gupta 2016; Katara ym. 1.9.2016.)

Testausautomaation ottaminen käyttöön on tämän vuoksi sidoksissa ohjelman suunnitteluun. Ohjelman on oltava automaation avulla testattavissa. Samaan aikaan on hankkees-

sa oltava ymmärrys siitä, ettei kaikkea voi testata automaatiota käyttäen. Testausautomaatiolla testattavien testitapausten määritteleminen helpottaa testaustyökalun valintaa, mutta samaan aikaan projektissa työskentelevien testaussuunnittelijoiden tulee ymmärtää testaustyökalun vaikutus ohjelmiston suunnittelun näkökulmasta. Tämä näkökulma tulee voida tuoda esiin ja kuulluksi. (Gupta 2016; Katara ym. 1.9.2016.)

Testausautomaation käyttö voidaan aloittaa asteittain pilotoimalla se tiettyihin testitapauksiin ja lisäämällä automaatiotestauksella testattavia testiskriptejä myöhemmin sitä mukaa, kun testaajien taidot kasvavat ja käsitys testaustyökalun ominaisuuksista paranee. Onnistunut testausautomaation käyttöönotto edellyttää myös, että hankeryhmällä on todenmukainen käsitys siitä, mitä automaatiolla on mahdollista testata ja minkä testaaminen on mielekästä. Tämä tarkoittaa myös realistista käsitystä siitä, mitä testausautomaation hankinta- ja ylläpitokustannus on, mutta myös testiskriptien ylläpidon vaatimien kustannusten tiedostamista. Vain tällä tavoin testaustyökalu voidaan valita mahdollisimman kustannustehokkaalla tavalla. (Katara ym. 1.9.2016.)

5.3 Ketterän sovellustiimin tietojen ja taitojen merkitys testausautomaation käyttöönotossa

Ketterä kehityksen sovellustiimi, joka koostuu koodareista, loppukäyttäjiä edustavasta asiantuntijasta ja testaajista, on tiedoiltaan ja taidoiltaan sekoitus eri alojen osaamista. Tiimin toiminnan kannalta oleellista on, että jokainen sen toimija on asiantuntija omassa roolissaan. Tämän lisäksi ketterien menetelmien hallinta tulee osata ja ymmärtää. (Cockburn 2007, 371-372.)

Ohjelmoijat, jotka rakentavat valmista ohjelmaa annettujen määritysten perusteella, tulee omata paitsi jonkinlainen ymmärrys siitä, miksi ohjelman halutaan toimivan juuri tietyllä tavalla, myös kyetä tuottamaan ratkaisut, jotka ovat virheettömiä koodiltaan ja toimivat, kuten vaatimusmäärittelyssä on esitetty. (Kasurinen 2013, 20-21.)

Asiantuntijaroolissa olevien loppukäyttäjien tulee omata riittävä ymmärrys omasta työstään ja oman organisaation toiminnan prosesseista ja niiden tavoitetilasta. Tämä on välttämätöntä, jotta vaatimusmäärittelyt voidaan tuottaa oikein ohjelmaa kirjoittavien ohjelmoijien käyttöön. Vaatimusmäärittelyt ovat perusta myös samaan aikaan ohjelmakoodin kanssa tapahtuvalle testitapausten kirjoittamiselle.

Testaajat, jotka kirjoittavat testiskriptejä samaan aikaan kirjoitettavan ohjelmakoodin kanssa, tulee hallita riittävät ohjelmointitaidot ja testausohjelmiston hallinta, jotta testitapaukset voidaan kirjoittaa oikein ja testaus voidaan toteuttaa testauksen näkökulmasta onnistuneesti. Tämä tarkoittaa yhteistyötä paitsi ohjelmoijien, myös loppukäyttäjien kans-

sa, jotta testaajat ymmärtävät ohjelman toiminnan rakenteen ja periaatteet, mutta sen lisäksi myös käyttäjien odotukset siitä, kuinka sen tulisi toimia ja millaisia testituloksia testitapauksista odotetaan. (Cockburn 2007, 275; Haikala & Mikkonen 2011, 215; ITSQB 2014, 29; Katara ym. 1.9.2016.)

Ketterän sovelluskehityksen tiimin jäsenten työltä odotetaan tuloksellisuutta ja kykyä tuottaa toimivaa ja julkaisukelpoista ohjelmaa. Teknisten ja sisällöllisten taitojen lisäksi tämä tarkoittaa kykyä työn suunnitteluun ja hallintaan, jotta ketterän kehityksen tiimi pysyy sille asetetuissa aikatauluissa. Julkaisukelpoinen ohjelma on kuitenkin ketterän kehityksen periaatteiden mukaan testattu. (ITSQB 2014, 19.)

Tämän kaiken onnistumisen edellytyksenä on kaikkien jäsenten välinen toimiva kommunikaatio ja yhteinen ymmärrys siitä, millaisia toimintoja testausautomaatiolla voidaan testata. Lisäksi tiimin jäseniltä odotetaan kykyä suunnitella ohjelman komponentteja ja rakennetta niin, että ne ovat testattavissa automaation avulla. (Katara ym. 1.9.2016.)

5.4 Testausautomaation käytön onnistumisen perustana olevat ketterän sovellustiimin toimintaperiaatteet

Ketterä sovelluskehitys tukeutuu yhä enemmän testausautomaation käyttöön. Ketterä sovelluskehitys rakentuu sprinteistä, joilla on selkeä määräaika. Tänä aikana tiimi suunnittelee, kehittää, rakentaa ja testaa sille määritellyt ohjelman osat. Onnistuakseen tämä vaatii tiimiltä ja koko organisaatiolta muutamia periaatteita, joita kaikki noudattavat. (Haikala & Mikkonen 2011, 44-45; Highsmith 2010, 16; Kasurinen 2013, 25.)

Tiimin jäsenet ovat vastuullisia omista tehtävistään ja pitävät kiinni annetuista määräajoista. Tavoite on julkaisukelpoisen, testatun ohjelman tuottaminen sprintin aikana. Työn on oltava suunnitelmallista, ja vaikka ketterien menetelmien periaatteena onkin toimiva ohjelma dokumentaation sijasta, dokumentaation roolia ei voi silti vähätellä. Vaatimusmäärittelyjen ja muutosten tulee olla dokumentoituja. Ketterissä menetelmissä etuna on kyky reagoida nopeasti muutostarpeisiin, mutta muutosten jäljittäminen on erittäin haastavaa ilman dokumentaatiota. Tämä tehtävä muuttuu ylivoimaisemmaksi jokaisen toteutuneen sprintin myötä. (Katara ym. 1.9.2016.)

Ketteriä sovelluskehityksen periaatteita noudattavat organisaatiot eivät ole täysin vailla hierarkiaa, mutta ohjelmistokehityksen näkökulmasta tietystä tasavertaisuuden periaatteesta tulee silti pitää kiinni. Tämä tarkoittaa sitä, että sovituista sprinteistä pidetään kiinni, eikä yhdelläkään taholla tai tiimin jäsenellä ole oikeutta lisätä sprinttiin sen lukkoon lyömisen jälkeen mitään, riippumatta omasta asemastaan ohjelmistokehitystä tuottavassa tai hankkeen omistajana olevassa organisaatiossa. (Meyer 2014, 154.)

Sovelluskehitystiimin sisäisen vuorovaikutuksen tulee toimia. Tämä mahdollistaa avoimen kommunikaation ja keskustelevan ilmapiirin, jolloin muutostarpeista on helppoa ja kaikille sen jäsenille turvallista ilmoittaa nopeasti. Tämä mahdollistaa nopean reagoinnin ja muutostarpeisiin vastaamisen. Keskusteluyhteyden tulee toimia paitsi sovellustiimin sisällä, myös niiden välillä, ja tämän lisäksi projektin johdon suuntaan. Yhtenä edellytyksenä tälle on, että jokainen ammattiryhmän edustaja kunnioittaa toistaan oman alansa asiantuntijana. (Payne 2005, 353; Williams 2014, 157- 160.)

Testausautomaation käyttöä silmällä pitäen vuorovaikutus testaajien, viiteryhmien ja ohjelmoijien välillä on merkittävässä asemassa. Työskentelytavat voidaan valita niin, että testaaja on tietoinen kirjoitettavasta, automaatiotestaukseen valitusta ohjelman toiminnallisuudesta, jolloin samanaikainen työskentely on mahdollista. Parityöskentely testaajan ja ohjelmoijan välillä mahdollistaa paitsi testattavissa olevan ohjelman syntymistä, myös testiskriptien kirjoittamista annetussa määräajassa oikein. Parityöskentelystä saatavana etuna on myös se, että sisällön asiantuntijuudesta vastaava Organisaation edustaja kykenee kommunikoimaan samaan aikaan ohjelmoijan ja testaajan kanssa, jolloin informaatioketju lyhenee ja väärinymmärrysten riski pienenee. (Graham ym. 2008, 38; Kasurinen 2013, 25-26; Katara ym. 1.9.2016).

Sovellustiimillä tulee olla selkeä käsitys siitä, mitä automaation avulla testataan ja mikä hoidetaan manuaalisella testauksella. Onnistuakseen tämä edellyttää sitä, että testit on jaettu eri testisalkkuihin, jotka ajetaan manuaalisesti tai automaatiota käyttäen.

5.5 Automaattitestaus – testaamisen merkitys ketterässä sovelluskehityshankkeessa

Ketterän sovelluskehityksen ajatuksena on tuottaa valmista ohjelmaa, joka julkaistaan. Julkaisukelpoinen ohjelma on testattua, mutta ketterä sovelluskehitys tuottaa lisää toiminnallisuuksia edellisen rakennuksen päälle. Uuden julkaisun tulee olla sellainen, ettei se riko aikaisemmin kirjoitettua ohjelmakoodia. Tämän varmistamiseksi jokainen iteraatio testataan, mutta tämän lisäksi on testattava myös aiemmin rakennettu ohjelman osa. Käytännössä tämä tarkoittaa, että ohjelman kehittämisen aikana on suunniteltava ja rakennettava myös testitapaukset, jotta sprintin päättyessä voidaan todeta ohjelmakoodin toimivuus. (Gupta 2016; Katara ym. 1.9.2016.)

Jokainen sprintti tuottaa lisää komponentteja, toimintoja ja moduuleja aiemmin rakennettuun ohjelmaan. Nämä testataan ennen julkaisemista, mutta tästä jatkuvasta rakennustyöstä ja testitapausten kehittämisestä ja rakentamisesta seuraa se, että testitapausten määrä kasvaa jatkuvasti. Vaikka kaiken testaaminen ei ole mielekästä tai mahdollistakaan, sovelluskehitystiimin tulee kyetä varmistumaan siitä, että ohjelma on mahdollisim-

man virheetön. Testaaminen ei takaa virheettömyyttä, vaan auttaa löytämään virheet. Testausautomaatiolla päästään ihmistä suurempaan testausnopeuteen ja lisäksi testit voidaan ajaa öisin ja viikonloppuisin. (Dustin, Garret & Gauf 2009, 73-75; Gupta 2016; Katara ym. 1.9.2016.)

Testausautomaatio vaikuttaa myös tiedostavan ohjelmistosuunnittelun toimintaan. Saadakseen kattavamman ja tehokkaamman testauksen tiimi suunnittelee ja toteuttaa sprintissä toteutettavat moduulit niin, että ne voidaan ajaa sovitusti mahdollisimman suurelta osin testausautomaatioon turvautuen. (Katara ym. 1.9.2016.)

Testausautomaatiosta tulee ketterälle sovellustiimille kuitenkin taakka, mikäli automatisoitavia testitapauksia ei ole määritelty riittävän tarkasti ja mielekkäästi. Testiskriptien kirjoittaminen on ohjelmointia ja vie aikaa. Suuremmasta testausnopeudesta on hyötyä vain, jos automatisoitavien testitapausten manuaalinen ajo veisi ohjelmiston kehittäjiltä ja testaajilta kohtuuttoman paljon aikaa. Mikäli testitapauksen suorittaminen on manuaalisesti paljon nopeampaa, kuin siihen liittyvän automatisoidun testiskriptin kirjoittaminen, voi olla aiheellista kysyä, onko automatisointi mielekästä.

Asiaan luonnollisesti vaikuttaa myös se, kuinka usein testitapaus suoritetaan. Mikäli testitapaus ajetaan jokaisen iteraation yhteydessä myöhemmin regressiotestauksessa, voi olla kannattavaa automatisoida se, mikäli automatisointi on tehtävissä mielekkäällä työmäärällä suhteessa testausaikaan. (Katara ym. 1.9.2016.)

Testaus voi ohjata ohjelmiston suunnittelua testiohjatussa sovelluskehityksessä paljonkin, mutta kuten aiemmin jo mainittiin, myös testattavuus testausautomaation näkökulmasta on vaikuttamassa ohjelmointi- ja kehitystyöhön. Testaajat ja testaus eivät kuitenkaan tuo lisäarvoa sovelluskehitystyöhön, elleivät testaajat ole ammattitaitoisia ja hallitse työkalujensa käyttöä. Testauksen on oltava suunnitelmallista ja testitapausten tulee olla harkittuja ja kyetä testaamaan ohjelman ominaisuuksia monipuolisesti. Testauksesta ja sen tuloksista saatavat raportit tulee saada ohjelman kehityksen käyttöön heti testauksen jälkeen, koska nämä tiedot ovat palautetta siitä, miten ohjelmointityö ja ohjelman suunnittelu ovat onnistuneet. (Dustin, Garret & Gauf 2009, 169-171; Graham ym. 2008, 25; Gupta 2016; Katara ym. 1.9.2016.)

Testaamisen merkitys tulee ymmärtää Organisaatiossa oikein. Testaamista ei voi lähestyä siitä näkökulmasta, että asiat ovat menneet pieleen, jos testiketju ei läpäise virheen vuoksi testaamista. Testaamisen yksi tarkoituksista on juuri virheiden etsiminen ja mitä nopeammin niitä löydetään aikaisessa vaiheessa, sitä luotettavammin ohjelma toimii tuotantovaiheen aikana. (Graham ym. 2008, 12-13; Kasurinen 2013, 10; Tuovinen 12.3.2013.)

6 Pohdinta

Opinnäytetyön aiheen valinta itsessään oli prosessi, joka eli vuosi sitten syksyllä melkoisesti. Alkuperäinen aikomukseni oli kirjoittaa opinnäytetyö kokonaan toisesta aiheesta, mutta kesällä 2019 suorittamani Vaatimusmäärittelylähtöisen testauksen kurssi ja noiden oppien tuominen käytäntöön työtehtävissäni suuntasi kiinnostuksen aiheen uudelleen testaamiseen. Testausautomaatio kiinnosti siksi, että siitä ei itsellä ole aikaisempaa kokemusta, mutta testausautomaation lisääntyminen tarkoittaa sitä, että ala tarjoaa varmasti töitä myös tulevaisuudessa.

6.1 Opinnäytetyön tulosten tarkastelu

Testausautomaation ottaminen käyttöön uutena järjestelmänä ketterillä menetelmillä kehitetyn uuden ohjelmiston käyttöönoton yhteydessä on resursseja vaativa projekti. Kyse on taloudellisesta panostuksesta samanaikaisesti kahteen erilaiseen järjestelmään. Vaikka testausohjelmistoa ei lokalisoitaisi Organisaation omille palvelimille, sen käyttäminen vaatii verkon yli pilvipalveluna osaamista testausautomaation käytön vaatimien teknisten taitojen vuoksi. On perusteltua ajatella, että monet yritykset ja organisaatiot päätyvät ulkoistamaan automaattitestauksen yrityksille, jotka ovat tähän erikoistuneet.

Ohjelmistokehityshankkeessa tulee olla ymmärrys siitä, mitä testausautomaatio on, kuinka se toimii ja mitä sillä voi tehdä. Tämän lisäksi sovelluskehitystiimien tulee ymmärtää ohjelmiston testattavuus ja sen edellytykset testausautomaation näkökulmasta. Näiden ehtojen täyttyminen vaativat tiedon jakamista ja hyvää kommunikaatiota projektiin osallistuvien välillä.

Projektitiimillä ja koko hankeryhmällä tulee olla yhtenevä ja selkeä näkemys siitä, millaista ohjelmaa ollaan kehittämässä ja mitä siltä odotetaan. Vaatimusmääritysten pysyminen ajan tasalla auttaa ohjelmoijia ja testaajia tuottamaan oikeassa aikataulussa oikeanlaista koodia ja testiskriptiä. Hyvä kommunikaatio mahdollistaa sen, että muutoksiin kyetään reagoimaan nopeasti.

Yhteisen käsityksen muodostaminen hankittavasta ja halutusta ohjelmasta auttaa hankkimaan oikeanlaisen testausohjelman. Tämä tarkoittaa yhteneviä periaatteita siitä, mitä testitapauksia automaation avulla ajetaan. Vaatimusmäärittelyn tarkkuus ohjaa myös testiskriptien kirjoittamista, joten vaatimusten pitämiseen ajan tasalla ja niiden yhteiseen saatavuuteen tulee kiinnittää huomiota.

6.2 Johtopäätökset, sekä kehittämis- ja jatkotutkimusehdotukset

Testausautomaation ottaminen käyttöön uuden ohjelmistohankkeen yhteydessä on hanke, joka vaatii huolellista pohdintaa ja resurssien arviointia ja suuntaamista testaukseen. Testausautomaatiolle tulee olla oikea tarve, jotta sen käyttö on mielekästä. Testausautomaatiolla testattavien testitapausten salkkua pitää kyetä hallitsemaan ja tämä vaatii ammattitaitoa ja harkintaa. Testitapausten ylläpito ja päivittäminen on myös taloudellinen panostus, joten hankinta ja ylläpito edellyttävät kustannuksia suurempia hyötyjä, jotta se olisi oikeasti kannattavaa.

Mikäli testausautomaation käyttöön panostava organisaatio päätyy hankkimaan ja ylläpitämään automatisoitua testauspalvelua, se joutuu punnitsemaan erilaisia ratkaisuja, joissa painavat kustannukset, soveltuvuus oman, käytössä olevan ohjelmiston testaamiselle, ammattitaitovaatimukset ja resurssit. On varsin todennäköistä, että moni testausautomaation käyttöön panostava organisaatio hankkii palvelun ulkopuoliselta toimittajalta, joka on erikoistunut tarjoamaan testauspalvelua ja ylläpitämään testausautomaatiota.

Vaikka organisaatiolla on oma tietohallintopalvelunsa sisäisenä palveluyksikkönä, sen henkilöstössä ei ole välttämättä testausautomaation asiantuntijoita. Testiketjujen hallinta, suorittaminen ja ylläpito vaativat osaamista ja asiantuntemusta. On tarpeellista miettiä, kannattaako tietohallinnon edustajaa kouluttaa automaatiotestauksen asiantuntijaksi. Tämä on sekä ajallinen, että rahallinen panostus, josta ei saada täyttä hyötyä välittömästi.

Ohjelmistokehitykseen erikoistuneet yritykset ovat omaksuneet ketterien toimintatapojen kulttuurin, mutta uutta järjestelmää ketterillä menetelmillä hankkiva organisaatio voi olla kokonaan uudenlaisten toimintatapojen ja toimintakulttuurin äärellä. Vahvasti ylhäältä alas johdettu, byrokraattinen organisaatio voi kohdata vaikeuksia koettaessaan toimia ketterillä menetelmillä työskentelevän organisaation tai yrityksen kanssa. Toimintakulttuuri on toinen, viestintä on enemmän horisontaalista ja työntekijöiden ammattitaitoon ja kykyyn suunnitella omaa työtään pitäisi voida luottaa. Haasteita ei ole pelkästään organisaation johdolla. Asiantuntijarooliin siirtynyt työntekijä voi joutua opettelemaan oman työnsä johtamista ja suunnittelemista, kun työtä aiemmin on johdettu tiukasti ylhäältä käsin.

Ketteristä menetelmistä ja niiden käytöstä kohosi tutkimusongelmia, joista mielestäni voisi tehdä lisätutkimusta. Yksi tutkimuskohde on julkisen sektorin ja yksityisten yritysten vertailu ketterillä menetelmillä toteutetuissa tietojärjestelmähankkeissa. Tarkentaisin tutkimuksen kohdetta nimenomaan hankkeen sisäiseen kommunikaatioon, viestintään ja tiedon kulkuun ja niiden eroihin, sekä yhteistyöhön liittyvään viestintään hankkeen omistajan ja ohjelman toimittajan välillä.

6.3 Opinnäytetyöprosessin ja oman oppimisen arviointi

Opinnäytetyötä kirjoittaessa tuli varsin nopeasti selväksi se, että mitä enemmän asiaa opiskelee, sitä vähemmän tietää. Sen hahmottaminen, mikä oli tämän opinnäytetyön kannalta merkityksellistä, oli alkuun haastavaa. Tasapainoilu sen välillä, mitä työssä kannattaa käsitellä ja mitä jättää ulkopuolelle, ei aluksi ollut kovinkaan yksinkertaista. Vaara siitä, että työ paisuu hallitsemattoman kokoiseksi, oli todellinen. Tavoitteena ollut hyvä opinnäytetyö loi samaan aikaan toisenlaisen pulman: asiaa pitää lähestyä monipuolisesti ja kattavasti. Onnistuneen rajauksen löytäminen vaati runsaasti ajatustyötä ja prosessointia kirjoittamisen eri vaiheissa.

Opinnäytetyön yhteydessä kohdatut oppimisen haasteet osoittavat erittäin hyvin sen, että vaikka aiheeseen on perehtynyt tätä työtä tehdessä enemmän, kuin minkään yksittäisen kurssin yhteydessä olisin perehtynyt, ei tämän opinnäytetyön tekeminen tee valmista minusta ammattitaitoista ja valmista testaussuunnittelijaa testausautomaation käyttöön. Työ antaa paremmat edellytykset oppia asiaa, mutta edessä tulee olemaan pitkä matka, ennen kuin voi kutsua itseään alan ammattilaiseksi, asiantuntijasta puhumattakaan.

Olen ollut mukana tietojärjestelmähankkeessa kolme kertaa. Kahdella ensimmäisellä kerralla kouluttajana ja kolmannella kerralla sovelluskehittäjän roolissa. Kaikissa näissä tehtävissä testaaminen oli osa työtäni. Vasta viimeisessä hankkeessa ketterien menetelmien käsite tuli itselleni tutuksi, mutta käsitteen todelliseen avautumiseen tarvittiin tämän opinnäytetyön aloittaminen.

Vuorovaikutus ja yhteistyö olivat ketterien menetelmien periaatteista ne, jotka nousivat itselleni voimakkaimmin esiin, osittain omien heikkouksien tunnistamisen kautta. Vuorovaikutus on kykyä sanoa sanottavansa, mutta myös kuulla toista niin, että molemmat tulevat ymmärretyiksi. Ilman näitä ehtoja yhteistyö ei ole sujuvaa. Työskentelytavaksi voidaan valita ketteristä menetelmistä sprinttityöskentely, mutta mikäli tieto liikkuu raskaan byrokraattisen mallin mukaan, organisaatio ei ole vielä ketterä, kuten tätä työtä tehdessä tulin oppineeksi. Omana haasteenani tunnistin kyvyn ilmaista itseäni niin, että tulen myös kuulluksi, mutta toisaalta myös lunastaa näkyvä paikka työyhteisössä niin, että saan oikeaan aikaan oikean tiedon.

Organisaatioissa toimii yleensä useita eri tasoja, joilla on erilainen arvovalta, mikä kokemukseni mukaan voi vaikuttaa myös ohjelmistokehitykseen. Hankkeen omistajana olevassa organisaatiossa arvovaltaisemmat tahot voivat ajaa ohituskaistalla sprintteihin ylimääräistä työtä, mikäli hankkeen hallinta ei ole tässä asiassa tiukka ja vaadi hyviä perusteluja tällaiselle toiminnalle. Tämä ei yleensä vähennä sovelluskehitystiimin työtä, vaan

saattaa moninkertaistaa yhdessä sprintissä tehtävän työmäärän niin, että lopulta vain osa työtehtävistä saadaan valmiiksi ajoissa.

Haastavimpana prosessina tässä opinnäytetyössä on ollut empiirisen osan kirjoittaminen. Sen prosessointi ja kirjoittaminen ovat vaatineet useamman kirjoituskerran, lisää lukemista ja tähän vaiheeseen liittyy eniten epävarmuutta tuovia tekijöitä. Epävarmuutta luo se, onko toteutus tehty oikein ja halutulla tavalla. Tämä johtuu siitä, että hanketta ei ollut mahdollista toteuttaa oikeassa ympäristössä ja oikeassa yrityksessä.

Lähteet

Ahmed, S. 2018. Overview of Software Testing Standard ISO/IEC/IEEE 29119. International Journal of Computer Science and Network Security, 18, 2, s. 112-116. Luettavissa: https://www.researchgate.net/publication/323759544_Overview_of_Software_Testing_Standard_ISOIECIEEE_29119. Luettu: 22.9.2020.

Berkun, S. 2006. Projektinhallinnan taito. Gummerus Kirjapaino Oy. Jyväskylä.

Beth, G & McKay, J. 2008. The Software Test Engineer's Handbook: a study guide for the ITSQB test analyst and technical test analyst advanced level certificate. Rocky Nook Inc. 26 West Mission Street Ste 3, Santa Barbara, CA 93101.

Cockburn, A. 2007. Agile Software Development. The Cooperative Game. Second Edition. Pearson Education Inc. Rights and Contracts Department. 501 Boylston Street, Suite 900. Boston.

Dustin, E., Garret, T. & Gauf, B. 2009. Implementing Automated Software Testing, How to Save Time and Lower Costs While Raising Quality. Pearson Education Inc. Rights and Contracts Department. 501 Boylston Street, Suite 900. Boston.

Graham, D., van Veenendaa, E., Evans, I. & Black, R. 2008. Foundations of Software Testing. ISTQB Certification. Revised Edition. Cengage Learning EMEA. High Holborn House, 50-51 Bedford Row. London.

Gupta, R. 2016. Agile Automation and Unified Functional Testing. Pearson Education India. Luettavissa: https://learning.oreilly.com/library/view/agile-automation-and/9789332578944/. Luettu: 24.9.2020.

Highsmith, J. 2010. Agile project management: creating innovative products. Second Edition. Pearson Education Inc. Rights and Contracts Department. 501 Boylston Street, Suite 900. Boston.

ISTQB 2014. International Software Testing Qualification Board. Perustason sertifikaattisisällön laajennus. Ketterä testaaja. Käännösversio 2015. Luettavissa: http://www.fistb.fi/sites/fistb/files/liitteet/FND-Agile-Syllabus_FI.pdf. Luettu: 22.8.2020.

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

Katara, M., Vuori, M. & Jääskeläinen, A. 2016. Ohjelmistojen testaus. Luento. Tampereen teknillinen yliopisto, Tietotekniikan laitos. Luettavissa: http://www.cs.tut.fi/~tie21201/s2016/luennot/TIE-21201_2016.pdf. Luettu: 23.8.2020.

Langer, A. M. 2012. Guide to Software Development. Designing and Managing the Life Cycle. Springer-Verlag London Limited.

Meyer, B. 2014. Agile! The Good, the Hype and the Ugly. Springer International Publishing Switzerland. Zurich.

Payne, A. 2005. Handbook of CRM: Achieving Excellence Through Customer Management. Routledge.

Subramani, G. 2020. Types of Testing Environments. Luettavissa: https://www.testenvironmentmanagement.com/types-of-testing-environments/. Luettu: 28.8.2020.

Tirkkonen, K. 2014. CASE KELA: monimutkaisten ja laajojen järjestelmien suorituskyky- ja tietoturvatestaus. Luettavissa: http://www.cs.tut.fi/~tie21201/s2014/luennot/vierailuluennot/TTY_luento_SKT_ja_TTT_Kelassa%202014.11.24.pdf. Luettu: 31.8.2020.

Tuovinen, Antti-Pekka. 2013. Ohjelmistotestauksen perusteita I. Luento 1. Helsingin Yliopisto. Luettavissa: https://www.cs.helsinki.fi/u/aptuovin/testaus/Ohj_testaus_2013_1.pdf. Luettu: 10.6.2020.

Valsta, A. 2016. Vaatimusmääritys ja vaatimuslähtöinen testaus. Opintojakson johdantokalvot. Haaga-Helia ammattikorkeakoulu. Luettavissa: https://hhmoodle.haaga-helia.fi/pluginfile.php/1442657/mod_resource/content/1/Johdanto-swd8tn001.pdf. Luettu: 10.10.2020.

Williams, D.S. 2014. Connected CRM: Implementing a Data-Driven, Customer-Centric Business Strategy. John Wiley & Sons, Incorporated.