jamk.fi

Automaatiotestaus ketterässä kehi-tyksessä

Yli-Tuina Oskari

Opinnäytetyö
Marraskuu 2019
Liiketalouden ala
Tradenomi (AMK), tietojenkäsittelyn tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Yli-Tuina, Oskari	Opinnäytetyö, AMK	Marraskuu 2019
	Sivumäärä 41	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Automaatiotestaus ketterässä kehityksessä

Tutkinto-ohjelma
Tietojenkäsittelyn tutkinto-ohjelma

Työn ohjaaja(t)
Kiviaho, Niko

Toimeksiantaja(t)
Kansaneläkelaitos ICT-palvelujen tulosyksikkö

Tiivistelmä

Yhä yleistyvän ketterän kehityksen myötä on tärkeää kerätä tietoa siitä, miten ketteryyttä tulisi oikeaoppisesti käyttää. Oikein käytettynä ketterä kehitys on erittäin toimiva tapa tehdä ohjelmistokehitystä, kun taas väärin käytettynä se saattaa olla jopa haitallista kehitystiimille. Laadunhallinta on myös isossa osassa ketterää kehitystä ja se on myös aliarvostettua, minkä vuoksi myös siitä on hyvä puhua ketteryyden yhteydessä.

Tutkimuksen tavoitteena oli selvittää, mitkä menetelmät tiimin ketterässä kehityksessä toimivat hyvin ja missä kohdissa toimintatapoja olisi mahdollista parantaa siten, että saataisiin parannettua tiimin laadunhallintaa sekä ketteryyttä.

Opinnäytetyön tuloksena valmistui kehitysehdotus siitä, miten juuri ketterään toimintamalliin siirtynyt kehitystiimi voisi parantaa toimintaansa ja saada ketteryydestä mahdollisimman paljon hyötyä tekemiseensä. Tiimin dokumentaatioon sekä tarkasteluun perustuvan tiedon perusteella tiimille tehtiin kehitysehdotukset siitä, miten heidän tiiminsä voisi parantaa toimintaa.

Johtopäätöksissä saatiin selville, miten ehdotetut muutokset olivat vaikuttaneet tiimin toimintaan ja mitä lisäarvoa ne olivat tuoneet tiimille. Saatuja tuloksia tarkastelemalla myös muut samanlaisessa tilanteessa olevat tiimit voisivat kehittää toimintaansa hyödyntämällä opinnäytetyön tuloksena saatuja kehitysehdotuksia.

Avainsanat (asiasanat)
Testaus, automaatiotestaus, laadunhallinta, Scrum, ketterä kehittäminen

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Yli-Tuina, Oskari	Bachelor's thesis	November 2019
		Language of publication: Finnish
	Number of pages: 41	Permission for web publication: x

Title of publication
Automation testing in agile development

Degree programme
Business Information Technology

Supervisor(s)
Kiviaho, Niko

Assigned by
The Social Insurance Institution of Finland

Summary

While agile software development is increasing, it is important to gather information how to be agile in proper way. When used properly, agile development is a very effective way to do software development, whereas when misused it might be adverse to the development team. Quality management is also an important part of agile development and is often undervalued, therefore it must be considered at the same time.

The aim of the study was to find out which methods of team agile development work well and where improvements could be made to achieve better team quality management and agility.

As a result of this thesis, a proposal was prepared on how the development team which had just adopted an agile operating model could gain the maximum benefit from its agility. Based on the team's documentation and review, suggestions are made to the team how they could improve performance.

The conclusions revealed how the proposed changes had affected the team's operations and the added value they had brought to the team. Based on the findings, other teams in a similar situation could develop their operation by utilizing the suggested improvements resulting from the thesis.

Keywords/tags (subjects)
Testing, Automation Testing, Quality Management, Scrum, Agile Development

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

Ketteryys ja ketterät toimintamallit ovat jatkuvasti yleistyvä tapa tehdä ohjelmistokehitystä. Nykyään ohjelmistot alkavat olemaan niin suuria, että ketterä toimintamalli alkaa olemaan ainut vaihtoehto toteuttaa niitä järkevästi. Kehittämisen lisäksi myös laadunhallinta on tehtävä ketterästi, jotta tuotokset säilyttävät vaaditun tason testauksen osalla.

Opinnäytetyössä tarkastellaan vasta kolme kuukautta ketterään kehitystä tehneen tiimin Scrumin toimintaa sekä ketterää testausta. Tiimin dokumentaation perusteella pyritään selvittämään, missä asioissa tiimi voisi parantaa toimintaansa saavuttaakseen paremman Scrumin toiminnan sekä laadunhallinnan tason.

Tutkimusasetelmassa käydään lävitse toimeksiantaja sekä selvitetään tutkimuksen tavoitteet, lähtötilanne ja toimeksiantajan tutkimuksesta saama hyöty. Myös tutkimuksen toteutus, menetelmät ja tutkimuskysymykset käydään läpi. Teoriaosuudessa perehdytään testaukseen, Scrumiin sekä ketterään testaukseen. Tämän jälkeen kerätään tutkimustulokset, joita verrataan lähtötilanteeseen. Tästä saadaan selville tiimin kehityksen tarpeessa olevat kohteet, joille tehdään kehitysehdotukset. Johtopäätöksissä tarkastellaan tutkimuksen vastaamista tutkimuskysymyksiin ja lopuksi pohditaan, miten opinnäytetyön toteutus onnistui.

2 Tutkimusasetelma

2.1 Toimeksiantaja

Toimeksiantajana opinnäytetyössä toimii Kansaneläkelaitoksen IT-palvelujen tulosyksikkö. Kelan toiminta-ajatus on: “*Me rakennamme hyvinvoinnin IT-järjestelmät*” ja sen tärkein tehtävä on kehittää ja ylläpitää suomen sosiaaliturvan mahdollistavia palveluja. Kelan IT-palveluissa työskentelee noin 700 henkilöä, joista lähes kaikki toimivat joko Helsingissä tai Jyväskylässä. (IT-Palvelut 2019)

IT-Palvelujen tulosyksikkö vastaa palveluiden tuottamisesta Kelan toiminnallisten yksiköiden sekä tulosyksiköiden tekemien sopimuksien mukaisille toimijoille sekä Kelalle. Tuotetut palvelut ovat tuotanto-, ylläpito- sekä IT-kehittämispalveluita. (Tulosyksiköt 2019)

2.2 Kehitystyön tavoite

Toimeksiantajalla on kehitystiimi, joka on juuri siirtynyt ketterään toimintamenetelmään. Tiimi on ketterään kehitykseen siirtymisen myötä ottanut käyttöön uusia toimintatapoja kuten automaatiotestauksen ja Scrumin. Opinnäytetyön tarkoituksena on saada lisättyä tiimin ymmärrystä ja tietoisuutta siitä, miten testausta, automaatiotestaamista sekä ketterän toimintamallin kehystä Scrumia käytetään tehokkaasti ketterässä kehityksessä. Tarkoituksena on kehittää tiimin laadunhallintaa, ketterää toimintamallia, Scrumia sekä automaatiotestausta ketterässä kehityksessä.

Opinnäytetyön teoriaosuudessa on tavoitteena selvittää, mitä eri testausmenetelmiä on olemassa ja miten niitä voidaan hyödyntää erilaisissa testaustilanteissa. Lisäksi tutustutaan Scrumin toimintaperiaatteisiin ja käytäntöihin sekä määritellä Scrumiin sisältyvät tapahtumat ja kokoukset. Scrumin tapahtumat käydään tarkemmin läpi ja perehdytään niiden tavoitteisiin ja tarkoituksiin. Tämän jälkeen tutustutaan vielä testauksen ja ketterän kehityksen tehokkaaseen yhteiskäyttöön. Kehitystyön jälkeen tiimillä pitäisi olla parempi käsitys sekä osaaminen siitä, mitä ketterä kehittäminen on laadunhallinnan, Scrumin sekä näiden yhteistoiminnan osalta.

2.3 Tutkimusmenetelmä

Opinnäytetyössä tavoitteena on saada positiivista muutosta tiimin laadunhallintaan sekä ketterään kehityksen. Opinnäytetyö toteutetaan kehittämistutkimuksena, koska sen tuloksena syntyy tuotos, jonka avulla voidaan parantaa tiimin toimintaa. (Kananen 2012, 19).

Kehittämistutkimuksessa luodaan viitekehys eli perehdytään tutkittavaan aiheeseen keräämällä aiheesta tietoa kirjallisuudesta sekä sähköisistä tietolähteistä. Tietoa voidaan kerätä raporteista, malleista, teorioista sekä tutkimuksista, jotka sivuavat tutkittavaa aihetta. Tämä auttaa ymmärtämään ongelman sekä lisää ymmärrystä tutkittavaa aihetta kohtaan. (mts. 47-48)

Opinnäytetyössä käytetään perinteistä tutkimusmenetelmää eli ensin tehdään kenttätyö, jonka jälkeen on kirjoitusvaihe (mts. 48). Kenttätyön aikana tietoa kerätään tutustumalla tiimin dokumentaatioon.

2.4 Tutkimuskysymykset

Kysymykset on johdettu tiimin tilanteen mukaan ja niiden avulla pyritään kehittämään tiimin ketterää työskentelyä. Opinnäytetyö pyrkii vastaamaan seuraaviin tutkimuskysymyksiin;

- Miten tiimin testausta voitaisiin parantaa?
- Miten tiimi pystyisi hyödyntämään Scrumia mahdollisimman tehokkaasti ketterässä kehityksessä?
- Miten testaus, automaatiotestaus ja Scrum toimivat yhdessä?

2.5 Tutkimuksen toteutus

Tutkimuksen alussa kartoitetaan tiimin lähtötilanne tutustumalla tiimiin ja lukemalla tiimin dokumentaatiota. Kun on ymmärretty tiimin lähtötilanne laadunhallinnan sekä Scrumin suhteen ja selvitetty mitä on tarve kehittää, lähdetään keräämään teoriatietoa tutkimuskysymysten määrittämistä aihealueista.

Teoriatietoa kerätään aiheista testaus sekä automaatiotestaus, scrum ketteränä toimintamallina sekä automaatiotestauksen ja scrumin tehokas toiminta yhdessä. Kun teoriatieto on kerätty, lähdetään vertaamaan sitä tiimin dokumentaatioon, josta saadaan tutkimustulokset. Tutkimustuloksissa käydään lävitse tiimin ongelmat tai muutosta kaipaavat kohteet. Ongelma- sekä muutoskohteille pyritään antamaan teoriatiedon perusteella mahdollinen syy ongelmalle sekä parannus- tai muutosehdotus.

Johtopäätösosiossa tarkastellaan tutkimustuloksia. Tutkimustuloksia verrataan alkuperäisiin tutkimuskysymyksiin ja tarkastellaan, onnistuttiinko tutkimuksella vastaamaan niihin. Lopuksi pohdinta osiossa pohditaan, miten tavoitteissa onnistuttiin, olivatko tulokset odotettuja, miten tuloksia voi hyödyntää sekä miten jatkokehitystä voisi toteuttaa.

3 Testaus

3.1 Testaus ohjelmistokehityksessä

Testauksen tärkein tehtävä on varmistaa tuotteen toimivuus, oikeellisuus ja laatu (Maynard n.d). Ohjelmistokehityksessä testauksella tarkistetaan, ovatko varsinainen tulos sekä odotettu tulos toisiaan vastaavia. Sillä pyritään myös takaamaan, että testattavasta kohteesta ei löydy ongelmia tai muita kohtia, jotka eivät vastaa alkuperäistä suunnitelmaa. Testauksella pyritään myös karsimaan koodissa tai ohjelmistossa olevat viat tai poikkeamat. Sen lisäksi testaus on myös hyvä tapa löytää mahdolliset virheet tai puuttuvat kohdat, jotka eivät ole vaatimusten mukaisia. (What is Software Testing? n.d.)

Testausta tehdään, jotta saataisiin aikaan toimivia kokonaisuuksia mahdollisimman pienellä määrällä ongelmia. Tämä säästää organisaation aikaa ja rahaa, varmistaa oh-

jelman toiminnan sekä pyrkii varmistamaan mahdollisimman vähän virheitä tai ongelmia sisältävän tuotteen käyttäjille. (Maynard n.d). Yleensä ongelmat tai virheet aiheuttavat vain vähän harmistusta käyttäjässä tai saattavat jopa olla huvittavia. Tästä syystä testauksen arvoa saatetaan vähätellä ja pitää merkityksettömänä, mutta todellisuudessa ongelmat voivat olla todella kalliita tai jopa vaarallisia. Historiassa on esimerkkitapauksia siitä mitä voi aiheutua, jos testaukseen ei panosteta. Alla kuvattuna muutamia tapauksia maailmalta. (What is Software Testing? n.d.)

- *In April 2015, Bloomberg terminal in London crashed due to software glitch affected more than 300,000 traders on financial markets. It forced the government to postpone a 3bn pound debt sale.*
- *Nissan cars have to recall over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.*
- *Starbucks was forced to close about 60 percent of stores in the U.S and Canada due to software failure in its POS system. At one point store served coffee for free as they unable to process the transaction.*
- *Some of the Amazon's third party retailers saw their product price is reduced to 1p due to a software glitch. They were left with heavy losses.*
- *Vulnerability in Window 10. This bug enables users to escape from security sandboxes through a flaw in the win32k system.*
- *In 2015 fighter plane F-35 fell victim to a software bug, making it unable to detect targets correctly.*
- *China Airlines Airbus A300 crashed due to a software bug on April 26, 1994, killing 264 innocent live*
- *In 1985, Canada's Therac-25 radiation therapy machine malfunctioned due to software bug and delivered lethal radiation doses to patients, leaving 3 people dead and critically injuring 3 others.*
- *In April of 1999, a software bug caused the failure of a $1.2 billion military satellite launch, the costliest accident in history*
- *In may of 1996, a software bug caused the bank accounts of 823 customers of a major U.S. bank to be credited with 920 million US dollars.*

(What is Software Testing? n.d.)

3.2 Testausmenetelmät

Testausta voidaan tehdä monella eri tavalla ja menetelmällä, joiden avulla voidaan varmistaa, että tehty koodi toimii halutulla tavalla (Pittet n.d). Testausta voi suorittaa manuaalisella testauksella, jossa testaus suoritetaan manuaalisesti jonkun henkilön toimesta. Toinen tapa on käyttää automaattista testaamista, jossa automaatiotestaustyökalut suorittavat testaamisen. Testausmenetelmät voidaan jakaa pääsääntöisesti kahteen alakategoriaan: Musta- ja Valkolaatikko testaus. (What is Software Testing? n.d.)

Manuaalitestaus

Manuaalisessa testauksessa testaaja käyttää tuotetta manuaalisesti ja varmistaa, että se käyttäytyy odotetusti (Maynard n.d). Manuaalitestaus mielletään yleensä tylsäksi ja jopa turhaksi, mutta projektin täydellinen automaatiotestaus ei ole mahdollista, jonka vuoksi manuaalitestausta on myös tehtävä. Manuaalitestauksessa testaaja suorittaa testit manuaalisesti ilman minkään automaatiotyökalun apua. Tämä on erittäin kallista, koska se vaatii työntekijän suorittamaan testit ja on altis inhimillisille virheille, koska testaaja voi unohtaa kohtia tai tehdä muita virheitä. (Pittet n.d). Vaikka käsin manuaalitestaus on kaikista alkeellisin tapa testata, silti sen avulla pystytään löytämään virheitä, joita ei automaatiolla löydettäisi. Manuaalitestaus on työläydestään huolimatta pakollista myös, jotta voidaan varmistua automaation soveltuvuudesta testattavaan kohteeseen. (Manual Testing Tutorial for Beginners n.d.)

Manuaalisen testauksen tavoitteena on varmistaa, että sovellus toimii oikein eli on määriteltyjen vaatimusten mukainen eikä siinä ole virheitä (Manual Testing Tutorial for Beginners n.d). Manuaalitestaus on myös vaatimus sille, että testattavaan kohteeseen voidaan tehdä automaattinen testaus. Mitään kohdetta ei voida automatisoida ennen manuaalista testaamista. (AUTOMATION TESTING Tutorial n.d.)

Automaatiotestaus

Automaatiotestaus on nykyään jo lähes kaikissa ohjelmistoprojekteissa käytössä oleva testaustapa, jonka tarkoituksena on tarkastaa ja validoida ohjelmisto. Auto-

maation arvo ymmärretään parhaiten, kun mietitään mitä testaaminen oli ennen automaation yleistymistä. Manuaalisen testauksen ollessa vielä normi, testaustiimi kehitti testisuunnitelmia ja tarkastuslistoja, joita sitten manuaalisesti suoritettiin aina kun projektiin tuli muutoksia, minkä jälkeen ne palautettiin takaisin suunniteltavaksi uudelleen. Tämä prosessi oli hidas, kallis ja virhealtis, minkä vuoksi kehitettiin automaatiotestaus. (Rehkopf n.d.)

Usein kehitysprojekteissa on kohteita, jotka vaativat testaamista useissa eri tilanteissa ja monta kertaa kehityksen aikana. Tähän automaatiotestaus on parhaimmillaan, koska sen voi laittaa tekemään saman testin samaan kohteeseen uudelleen ja uudelleen. Automaatio ei unohda, eikä se myöskään tee inhimillisiä virheitä, vaan suorittaa testin samalla tavalla joka kerta. Tällä tavoin pystytään vähentämään manuaalitestausta kohteissa, johon pystyy tekemään automaatiotestauksen. On kuitenkin hyvä muistaa, että automaatiotestauksen ei ole tarkoitus korvata tai poistaa manuaalitestausta vaan vähentää sen tarvetta sopivissa paikoissa. Manuaalitestausta ei silti saa unohtaa. (AUTOMATION TESTING Tutorial n.d.)0

Vaikka automaatiotestaus vaatiikin aikaa ja rahaa toimiakseen, maksaa se itsensä myös takaisin niin rahallisesti kuin laadullisesti. Manuaalitestaus koko projektin ajan, kaikkiin sen osiin kaikissa tilanteissa vie runsaasti aikaa eli on kallista. Inhimillisyys on myös haitta manuaalitestauksessa, koska ihminen on erehtyväinen ja alkaa väsymään testatessa, mikä voi aiheuttaa huolimattomuutta, unohduksia tai muita inhimillisiä virheitä, kun taas automaatio pystyy toistamaan testin samalla tavalla rajattoman määrän kertoja. (AUTOMATION TESTING Tutorial n.d.)

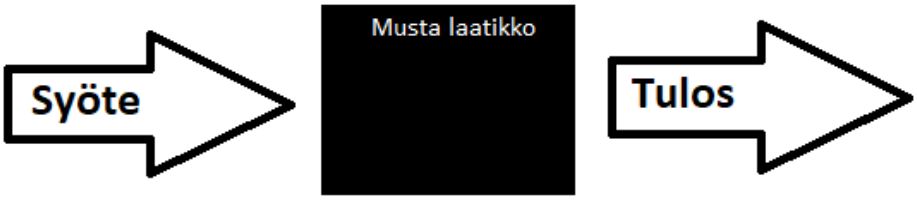
Automaatiotestaus ei enää tietyn vaiheen jälkeen vaadi ihmistä toimiakseen, vaan se pystyy suorittamaan itseään itsenäisesti esimerkiksi yöllä. Automaation avulla testit pystytään myös suorittamaan nopeammin kuin manuaalisesti. Testattavien kohteiden määrää on helppo kasvattaa automaatiotestauksessa, koska se ei unohda edellisiä testauskohteita. Näin ollen siitä on helppo tehdä todella kattava. (AUTOMATION TESTING Tutorial n.d.)

Automaatiotestausta on hyödyllistä käyttää silloin, kun kyseessä on jokin tärkeä kohde, jonka toimivuus olisi hyvä testata säännöllisesti, tai kohde pitää testata usein ja toistuvasti. Myös vaikeasti testattavat kohteet on hyvä testata automaattisesti, jos automaatio on mahdollista tehdä kohteeseen. Muita hyviä paikkoja automaatiotestaukselle on kohteet, joita on hidasta testata manuaalisesti, tai kohteen testaaminen on jostain muusta syystä ikävää. (AUTOMATION TESTING Tutorial n.d.)

On kuitenkin hyvä muistaa, että jokainen kohde on syytä testata manuaalisesti vähintään kerran ennen automaatiotestauksen tekemistä. Tällä pystytään varmistumaan, että kohde on mahdollista automatisoida. On myös hyvä joskus suorittaa manuaalitestausta kohteille, joihin on tehty automaatiotestaus. Kohteissa, jotka ovat jatkuvan muutoksen alaisina, on hyvä pohtia automaatiotestauksen kannattavuutta. Jos automaatiotestin ylläpitoon menee enemmän aikaa kuin manuaalitestaukseen, voi olla viisaampaa käyttää manuaalitestausta. Vaikka automaatiotestityökaluilla pystyy tekemään hyvin mukautuvia testejä, voi silti osa kohteista olla liian tapauskohtaisia testiautomaation käsiin. (AUTOMATION TESTING Tutorial n.d.)

Musta laatikko -testaus

Musta laatikko -testauksessa, samoin kuin valkoinen laatikko –testauksessa laatikko kuvaa testattavaa järjestelmää ja väri kuvaa sitä, mitä testaaja näkee järjestelmästä. Musta laatikko –testauksessa testaaja ei näe koodirakennetta tai toteutuksen yksityiskohtia. Tämän vuoksi testaus tapahtuu pelkästään ohjelmistovaatimusten sekä -määritysten pohjalta (ks. Kuvio 1). Musta laatikko -testauksella voi testata minkä tahansa sovelluksen, koska se ei vaadi koodin tuntemista tai siihen tutustumista ja se on myös tästä syystä nopea tapa testata. (What is BLACK Box Testing? n.d.).

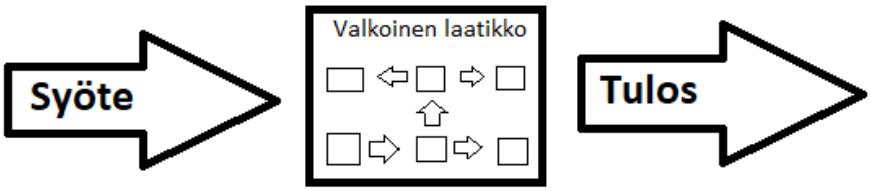


Kuva 1. Musta laatikko -testaus

Musta laatikko –testausta tehdään valitsemalla oikeanlaiset sekä vääränlaiset syötteet ja määrittelemällä odotetut tulokset valituille syötteille. Tämän jälkeen testit suoritetaan, ja syntyneitä tuloksia verrataan odotettuihin tuloksiin. Musta laatikko –testauksen tarkoituksena on validoida toiminnalliset vaatimukset sekä testata järjestelmän käyttäytymistä ja moduulien välistä kommunikaatiota. (What is BLACK Box Testing? n.d.)

Valkoinen laatikko –testaus

Valko laatikko –testauksessa testataan ohjelmiston sisäisiä rakenteita, suunnittelua sekä koodia. Toisin kun musta laatikko –testauksessa, koodin on oltava testaajan nähtävillä (ks. Kuvio 2). Tämän testaustyylin avulla pyritään kehittämään tuotteen suunnittelua ja käytettävyyttä sekä parantamaan turvallisuutta. (What is WHITE Box Testing? n.d.)



Kuva 2. Valkoinen laatikko -testaus

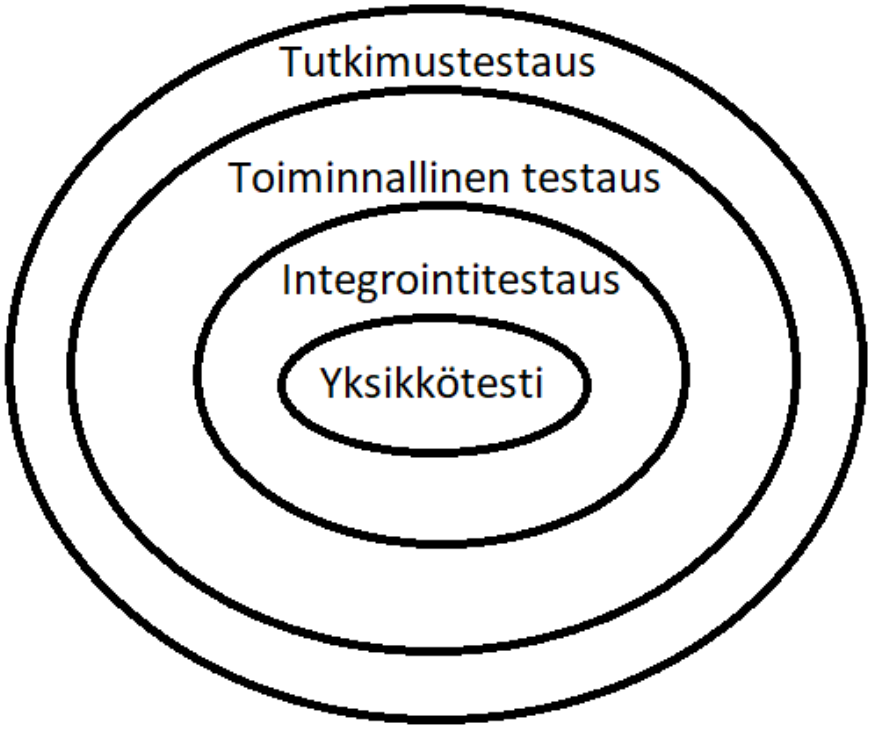
Valko laatikko -testauksella pyritään varmistamaan, että jokainen lauseke ja objekti toimii tietyllä arvolla. Koska koodi on nähtävillä, pystytään syötteiksi asettamaan haluttuja arvoja, mikä tekee testauksesta perusteellisempaa, koska kaikki koodipolut pystytään testaamaan. Koodista tulee myös optimoidumpaa, kun piilossa olevatkin

virheet pystytään löytämään. Valko laatikko –testauksessa testitapaukset on myös helppo automatisoida tarkan määrittelyn vuoksi. (What is WHITE Box Testing? n.d.)

Isommissa sekä monimutkaisissa kokonaisuuksissa valkoinen laatikko –testaus saattaa muodostua kalliiksi sekä monimutkaiseksi. Kun sovelluksen koko kasvaa tai monimutkaistuu, myös testattavien kohteiden määrä lisääntyy. Suuret ohjelmasovellukset vaativat todella paljon aikaa riittävän testien kattavuuden saavuttamiseksi sekä vaatii testaajalta osaamista, jotta hän pystyy testaaman koko monimutkaisen kokonaisuuden. (What is WHITE Box Testing? n.d.)

3.3 Testauksen tasot ja tyylit

Ohjelmistotestaus voidaan määritellä perustasoihin, joista kukin tutkii ohjelmistoa omalla ainutlaatuisella tavallaan. Tasoja on neljä, ja niitä voi kuvata ympyrällä, koska ne kuvaavat testattavan kohteen syvyyttä ohjelmistossa (ks.kuvio 5). Sen lisäksi on myös kolme muuta testaustyyliä. (Maynard n.d).



Kuva 3. Testauksen tasot

Yksikkötestaus

Yksikkötestaus on testauksen perustaso, jossa tarkastetaan, että syötteestä tulee odotettu tulos yksittäisessä kooditapauksessa eli komponentissa. Jos hypoteettisessa funktiossa “x = y” syöte “x”:n odotettu tulos on “y” ja yksikkötesti palauttaa “y”, on yksikkö testi läpäisty. Jos taas testi palauttaa “z” on testi epäonnistunut. (Maynard n.d).

Integrointitestaus

Ohjelmistojärjestelmä koostuu komponenteista. Jokaisen komponentin on toimittava luotettavasti yksinään, mikä testataan yksikkötestauksella, mutta kaikkien komponenttien pitää myös toimia yhdessä suunnitellusti (Linz 5.1.) Integraatiotestaamisen päätarkoituksena on testata, että yksiköt toimivat loogisesti ja oikein myös silloin, kun ne ovat yhteydessä toisiin yksiköihin (Integration Testing n.d).

Toiminnallinen testaus

Tämän testauksen tarkoituksena on simuloida ne testitapaukset, jotka vastaavat käyttäjäkokemusta. Niillä pyritään kuvaamaan ihmisen käyttäytymistä, esimerkiksi: Avaa linkki, paina elementtiä ja täytä lomake. (Maynard n.d).

Tutkimustestaus

Tutkimustestauksessa testaaja tekee testattavassa kohteessa jotain, mitä normaali käyttäjä voisi tehdä. Tällä tavoin pyritään löytämään virheet, jotka tulisivat käyttäjille vastaan tuotetta käytettäessä. Tavoitteena on löytää virheet, joten tuotteen käyttötapaa on hyvä muuttaa, sekä tehdä odottamattomia asioita testatessa tuotetta. (Maynard n.d).

Hyväksymistestaus

Nämä viimeisimpänä suoritettavat testaukset varmistavat sen, että järjestelmä täyttää liiketoiminnan vaatimukset. Niillä pyritään toistamaan koko käyttäjän käyttäytyminen sovelluksessa. (Pittet n.d.)

Suorituskykytestaus

Näillä testeillä pyritään varmistamaan järjestelmän toiminta myös silloin, kun se on tavallista suuremman kuormituksen alla. Suorituskykytestit ovat ei-toiminnallisia testejä, joilla voidaan tarkkailla esimerkiksi vasteaikoja suurella määrällä pyyntöjä, tai miten järjestelmä kestää valtavan määrän dataa. (Pittet n.d.)

Savutesti

Savutestien on tarkoitus testata järjestelmän kaikki kriittiset toiminnot nopeasti ja tehokkaasti. Niillä pyritään varmistamaan, että järjestelmän tärkeimmät ominaisuudet ovat toiminnassa. Niillä voidaan tarkistaa, että sovellus toimii esimerkiksi pystytyksen jälkeen tai ympäristön vaihdon jälkeen. (Pittet n.d.)

4 Ketterän kehityksen toimintamalli

4.1 Scrum

Scrum on käytetyin ketterän kehityksen toimintamalli (Ghahrai, 2018; Lamelas, 2018). Lyhyesti Scrum on kehys, jonka puitteissa ihmiset voivat ratkaista monimutkaisia ja adaptiivisia ongelmia, tuottaen samalla tuotetta luovasti ja tuottavasti korkeimmalla mahdollisella arvolla (A Better Way Of Building Products, n.d).

Scrum on kehys, joka auttaa tiimiä työskentelemään yhdessä tehokkaasti. Se rohkaisee tiimiä oppimaan ja kehittymään kokemusten kautta. Vaikka Scrumista usein puhutaan vain ohjelmistokehityksen yhteydessä, sen periaatteet ja opit sopivat kaikenlaiseen ryhmätyöhön, mikä tekeekin siitä niin suositun. Scrum on ketterä projektinhallinta työkalu, joka pitää sisällään työkaluja, palavereita ja kokouksia, jotka auttavat tiimiä onnistumaan tavoitteissaan ja organisoimaan työn tekemistä. (Drumond n.d.)

Toisinaan ihmiset ajattelevat scrumin ja ketterän kehityksen samana. Tämä johtuu siitä, että scrumin perusajatus on pyrkiä jatkuvasti parempaan, mikä on myös kaiken ketterän tekemisen perusperiaate. Scrum ja ketteryys eivät kuitenkaan ole sama asia, koska Scrum on kehys minkä avulla pystytään saavuttamaan tavoitteita ja ketteryys

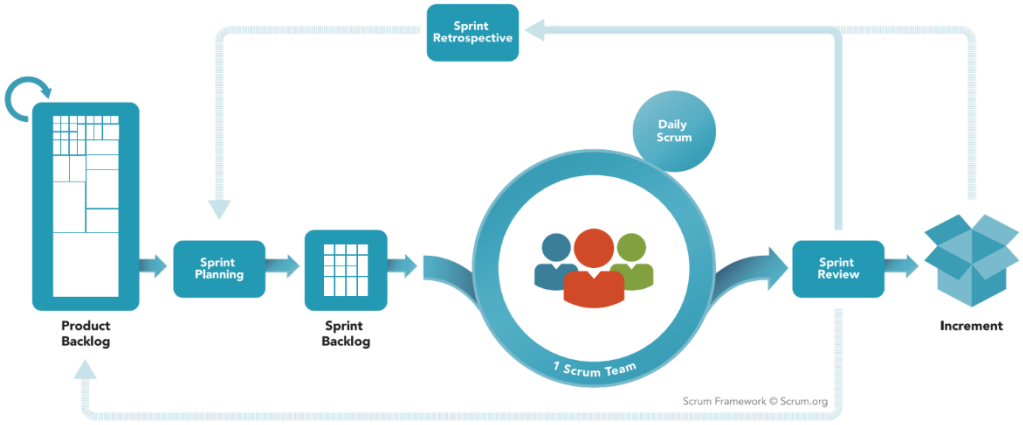
sen sijaan on tapa ajatella. Scrum on työkalu, jonka avulla ketteryyttä voidaan tuoda jokapäiväiseen työhön ja jonka seurauksena ajatusmallia pystytään ohjaamaan haluttuun suuntaan. Tiimi ei voi vain alkaa ketteräksi, koska ihmiset eivät voi vain vaihtaa tapaansa ajatella, varsinkaan tuntematta ketterän kehityksen periaatteita. Ketteryyteen on ennemminkin pyrittävä. (Drumond n.d.)

Scrum siis perustuu jatkuvaan oppimiseen ja sopeutumiseen. Se on rakennettu siten, että se mahdollistaa muuttuvat olosuhteet ja vaatimukset. Lyhyet sprintit sekä julkaisusyklit mahdollistavat uudelleen priorisoinnin ja mukautumisen, mikä auttaa tiimiä kehittymään ja parantamaan toimintaa. Myös Scrumin joustavuus tekee siitä suositun, sillä jokainen tiimi saa kehyksen sisässä muovata itselleen toimivan kokonaisuuden, mikä mahdollistaa Scrumin toimivuuden lähes kaikissa tapauksissa. (Drumond n.d.)

Scrumin tärkeimpiä ja tunnetuimpia toimintoja ovat kaikki tiimin säännöllisesti suorittamat kokoukset, tapahtumat ja seremoniat. Joitakin tiimejä tämä ärsyttää ja joidenkin mielestä ne ovat turhia, mikä kertoo siitä, että niitä ei käytetä oikein. Ne ovat kuitenkin tärkeä osa Scrumia. Hyvä tapa on kokeilla kaikkia kokouksia ja seremonioita esimerkiksi yhden sprintin verran, jonka jälkeen palautteen perusteella voi muovata niistä omalla tiimilleen sopivan kokonaisuuden. (Drumond n.d.)

4.2 Scrumin toimintaperiaate

Scrum koostuu tiimistä, artefakteista sekä tapahtumista. Kaiken pohjana on julkaisusykli, joka sisältää 2-5 sprinttiä (Release Planning. n.d). Sprintit ovat noin 2-3 viikon mittaisia jaksoja, jolloin Scrum-tiimi pyrkii saamaan aikaiseksi jotain valmista. Yhteen sprinttiin sisältyy artefaktit Product backlog, sprint backlog ja Increment sekä tapahtumat ja seremoniat sprint plannig, dailyt ja weeklyt, sprint rewiew ja sprint retrospective (Ks. Kuvio 4).



Kuva 4. Sprintti Scrumissa

Scrum artefaktit

Scrumissa on kolme artefaktia. Artefakti termi tarkoittaa esinettä tai asiaa, jonka ihminen on valmistanut, eli artefaktit ovat jotain, jonka tiimi on tehnyt (Green 2017). Tässä tapauksessa ne ovat työkaluja. Scrumissa nämä kolme artefaktia ovat product backlog, sprint backlog ja Increment. Product backlog on pääluettelo töistä, eli dynaaminen lista, jota tuoteomistaja ylläpitää, se on tiimin “To Do” lista. Siellä pidetään kaikki tunnetut vaatimukset, jotka pitää toteuttaa projektin onnistumiseksi (Linz 2.1). Monimutkaisessa ohjelmistokehitysprojektissa ei voi olla vain pitkää luetteloa kaikista tehtävistä, vaan tiimin on päätettävä mitkä tehtävät tehdään milloinkin (Linz 2.1). Sprint backlog on tehtävälista, jonka tuoteomistaja on valinnut ja tiimi on hyväksynyt toteutettavaksi kyseisellä sprintillä. Sprint backlog voi olla joustava ja kehittyä sprintin aikana, mutta kaikki sillä olevat tehtävät tulisi saada valmiiksi sprintin loppuun mennessä. Increment on sprintin tavoite. Tuotteen on tarkoitus kasvaa jokaisessa iteraatiossa, minkä vuoksi jokaisen sprintin tulisi tuottaa jotain julkaisukelpoista, jota kutsutaan incrementiksi (Linz 2.1). Vaikka joka sprintillä ei tulekaan mitään julkaisukelpoista valmiiksi, tulisi jokaisella sprintillä olla tavoitteena joku, vaikka pienikin, viimeistelty osa jotain mahdollisesti suurempaa kokonaisuutta. (Drumond n.d.)

Sprint planning

Sprint planning on sprintin aloitustapahtuma. Tarkoituksena tässä tapahtumassa on saada kaikille tiimin jäsenille selkeä käsitys, mitä kyseisellä sprintillä on tarkoitus tehdä ja miten se tehdään. Planningin jälkeen kaikille tulisi olla selvää, kuinka kauan tehtävien tekemiseen on aikaa, mitä tehtäviä tehdään ja mistä aloitetaan. Sprint planningissä on tärkeä kaikkien ymmärtää, että kaikki mitä sprintille suunnitellaan, on tarkoitus saada valmiiksi. Tässä on hyvä myös tiedostaa, että sen ei ole tarkoitus olla negatiivinen asia (kaikki on saatava valmiiksi), vaan tiimin tulisi olla motivoitunut, haastettu ja yhtä mieltä siitä, että kaikki pystytään saamaan valmiiksi. Planningillä voidaan aiheuttaa paljon motivaation heikkenemistä, jos tiimi kokee, että tehtävä on mahdoton toteuttaa. (West 2019.)

Sprint planning voidaan ajatella laatikkona, jonne laitetaan tavoite. Tavoite kerätään yleensä product backlogilta ja se sisältää asioita, jotka voisivat olla osa tulevaa sprinttiä. Myös edellisellä sprintillä keskeneräisiksi jääneet asiat on hyvä ottaa huomioon. Laatikosta tulee ulos suunniteltu tulos. Tuloksen on tarkoitus olla suunnitelma sprintillä tehtävistä asioista, sekä siitä miten ne tehdään. Eli selkeä sprintin backlogi. (West 2019.)

Tässä kuvitteellisessa laatikossa tuoteomistaja määrittelee, mitkä tehtävät tällä sprintillä olisi tarkoitus tehdä, ja esittää oman mielipiteensä, mitkä product backlogin kohteet tulisi tehdä sprintin tavoitteen saavuttamiseksi. Tuotantotiimin pitää ymmärtää miten he voivat toteuttaa suunnitellut tehtävät ja mitä voidaan tehdä tämän savuttamiseksi. Tuotantotiimin tehtävä on myös sanoa, jos tiimin mielestä tehtävät eivät ole toteutettavissa sprintin aikana. Sen jälkeen tuotantotiimi suunnittelee tehtävät työt, jotta sprintin tavoite saadaan saavutettua. Jos sprint planningistä puuttuu tuoteomistaja tai tuotantotiimi, on sprintti mahdotonta suunnitella. Hyvä sprint planning on neuvottelu tuoteomistajan ja tuotantotiimin välillä, jotta saadaan paras ymmärrys siitä, miten tehdylle työlle saadaan paras arvo sprintin tavoitteiden kannalta. (West 2019.)

Sprint planningissä, kuten muissakin Scrumin tapahtumissa, tulisi olla aikaraja. Sen keston tulisi maksimissaan olla noin 1-2 tuntia jokaista sprintillä olevaa viikkoa kohti

riippuen tiimin koosta. Tarkoituksena ei siis ole, että kaikki tämä aika käytettäisiin, vaan se on maksimipituus planningille, jotta saadaan pidettyä tapahtuma tehokkaana ja että planningissä käsitellään vain siihen kuuluvat asiat. (West 2019.)

Jos aikarajassa pysyminen tuottaa ongelmia tiimille tai jos tapahtuma tuntuu muuten sekavalta, voi tiimi ottaa käyttöön pre-planningin (suom. esisuunnittelu). Siinä tiimin tarpeesta riippuen tuoteomistaja tai tuoteomistajat, Scrum-master sekä tuotantotiimi kokoontuvat tekemään alustavan suunnitelman siitä, mitä kohteita seuraavalle sprintille tulisi ottaa. Tällöin kaikki pääsevät näkemään suunnitellut tehtävät ja voivat antaa niistä palautetta. Se helpottaa tuoteomistajaa tekemään päätökset seuraavan sprintin tehtävistä, ja kehitystiimi saa hetken aikaa miettiä rauhassa tulevia töitä, mikä helpottaa niiden pisteytystä. (Sprint Preplanning n.d.)

Kun käytetään Scrumia ja suunnitellaan sprinttiä, on hyvä muistaa, että tarkoitus olisi keskittyä tulokseen. Liiallinen suunnittelu ei ole tarpeen ja on yleensä hukkaan heitettyä, koska harvoin on tarpeeksi tietoa sprintin täydelliseen suunnitteluun. On hyvä antaa arviot työn vaatimasta ajasta, mutta ei kannata kuvitella tietävänsä enempää kuin tietää. Ei ole realistista kuvitella, että kaikki pystyisivät antamaan tarkat arviot työn viemästä ajasta. (West 2019.)

Daily

Daily on päivittäinen, vain 15 minuuttia kestävä kokous, jossa Scrum-tiimin jäsenet ja mieluusti vain tiimin jäsenet, kokoontuvat yhteen. Dailyssä jokainen tiimin jäsen vastaa kolmeen kysymykseen. Teoriassa tämä kuulostaa helpolta, mutta siinä on silti mahdollista epäonnistua. Monta asiaa voi mennä pieleen, mikä taas laskee dailyn arvoa. (Angeling 2018.)

Dailyssa tarkoituksena on, että jokainen osallistuja vastaa kolmeen kysymykseen:

- Mitä tein eilen?
- Mitä teen tänään?
- Mitkä asiat hidastavat tai estävät työskentelyäni?

Koska daily on hyvin lyhyt ja tehokas kokous, on tärkeää, että pyritään noudattamaan tiettyjä sääntöjä, jotta siitä saataisiin mahdollisimman hyödyllinen. Sen tehokkuutta saadaan helposti laskettua rikkomalla tiettyjä perusperiaatteita. Dailyn tulisi aina alkaa samaan aikaan (Lum 2016). Dailyn päivittäin samana pysyvä alkamisajankohta on tärkeä, jotta siitä muodostuu rutiini tiimille, sillä se auttaa tiimiläisiä tulemaan paikalle oikeaan aikaan. Ketään myöhästyjää ei pidä jäädä odottelemaan alussa, vaan kaikkien olisi syytä olla paikalla oikeaan aikaan, etteivät he tulisi kesken ja keskeyttäisi muita. On myös tärkeää, ettei tapaaminen veny, ainakaan paljoa yli 15 minuutin. Aikataulussa pysymisen kannalta tehokas keino on pitää kokous seisten, koska silloin ihmiset pyrkivät paremmin pitämään reipasta tahtia yllä (Lum 2016). Seisominen myös auttaa siihen, että ihmiset ovat vastaanottavampia, eli he kuuntelevat paremmin toistensa kertomukset. Daily ei ole hyödyllinen, jos kaikki keskittyvät vain omalla vuorollaan, eikä kellään ole yhtään enempää tietoa keskustelluista töistä tai tapahtumista dailyn lopussa kuin sen alkaessa. (Angeling 2018.)

Dailyssä olisi hyvä muistaa ottaa huomioon myös kaikki läsnäolijat. Eli kaikki aiheet olisi hyvä käsitellä siten, että jokainen paikalla oleva tiimiläinen ymmärtää käsiteltävän asian. Se ei ole paikka, jossa aletaan ratkaisemaan ongelmia tai suunnittelemaan uusia ideoita. Tarvittaessa dailyssä voidaan sopia uusi tapaaminen ilmenneille ongelmille tai haasteille, mutta ei tuhlata kaikkien osallistujien aikaa niiden puhumiseen dailyssä, vaan kutsutaan uuteen palaveriin vain asiaan liittyvät tiimiläiset. (Angeling 2018.) Tarvittaessa dailyssä voidaan myös pitää sprint backlogia esillä, jos koetaan, että se helpottaa osallistujia ymmärtämään asian mistä puhutaan. Näin tehtäessä pitää kuitenkin muistaa, että se ei ole paikka missä jaetaan uusia tehtäviä tai suunnitellaan sprinttiä uudelleen (Lum 2016).

Dailyn pitäminen saattaa kuulostaa haastavalta ja tuntua hyödyttömältä, jos sitä ei osata pitää oikein eikä ymmärretä syytä sen pitämiselle. Dailyn tarkoitus on parantaa tiimissä läpinäkyvyyttä, tarkastella sprinttiä ja tarvittaessa mukautua muuttuneeseen tilanteeseen. Läpinäkyvyys on tärkeää, koska ketterässä kehitystiimissä kaikkien on hyvä tietää, missä mennään ja mitä tapahtuu. Kaikkien osapuolten on syytä tietää ja ymmärtää sprintin tavoite ja sen vaiheet. Tämä auttaa tarkastelemaan sprinttiä ja

sitä, ollaanko sprintin tavoitteeseen pääsemässä. Ja mikäli kaikki ei ole mennyt suunnitellusti tai suunnitelma on ollut alun perin mahdoton toteuttaa, saadaan käsitys siitä, miten pitäisi mukautua, jotta sprintin tavoitteet saavutettaisiin. On kuitenkin hyvä muistaa, että daily on tiimiä varten, ja jokaisen tiimin on itse löydettävä tapa, miten se palvelee tiimiä parhaiten. (Angeling 2018.)

Weekly

Päivittäisen dailyn lisäksi on olemassa viikoittainen weekly. Se ei ole yleisesti niin laajassa käytössä, koska kaikki eivät näe tarvetta pitää sitä päivittäisten tapaamisten lisäksi. Joissakin projekteissa ajatellaan vieläkin, että aika, joka ei ole käytetty projektin tekemiseen, ei ole hyvin käytetty. Tästä ajattelutavasta olisi hyvä päästä eroon. (Parker 2017.)

Weekly on hyvä aika tiimille kokoontua yhteen loppuviikon aikana keskustelemaan tehdyistä töistä ja tarvittaessa uudelleen suunnittelemaan sprintin jäljellä olevaa aikaa ja tehtäviä. Weeklyn tavoitteena on kohottaa tiimin moraalia tarjoamalla halukkaille mahdollisuus päästä kertomaan ja näyttämään mitä on viikolla tehnyt ja miltä tekeminen on tuntunut. Myös sprintin tavoite on hyvä tarkistaa viikoittain. Weeklyssä tarkastellaan tiimin ajankäyttöä tehtyjen tehtävien osalta ja pohditaan, onko aikaa riittävästi, jotta kaikki suunnitellut työt saadaan tehtyä. Jos sprintin tilanne näyttää siltä, että kaikkia töitä ei saada tehtyä, voidaan jäljellä olevat työt priorisoida ja pienimmillä prioriteeteilla olevat työt jättää pois sprintiltä. Tilanne voi myös olla se, että aikaa on jäämässä yli, jolloin sprintille voidaan nostaa lisää tehtäviä. (Parker 2017.)

Sprint review

Jokaisen sprintin lopussa järjestetään review-tilaisuus, jota virheellisesti saatetaan kutsua demoksi, eli tapahtumaksi, jossa vain esitellään tehdyt työt. Se ei kuitenkaan ole ainoastaan demo. Reviewin kutsuminen demoksi aiheuttaa ihmisille vääränlaisen mielentilan tapahtuman suhteen sekä antaa virheellisen kuvan tapahtuman luonteesta, koska reviewin tärkein tarkoitus ei ole esittely, vaan palaute. Reviewissä on tarkoitus antaa palautetta jokaisen työpanoksesta projektille. Se on myös hyvä aika

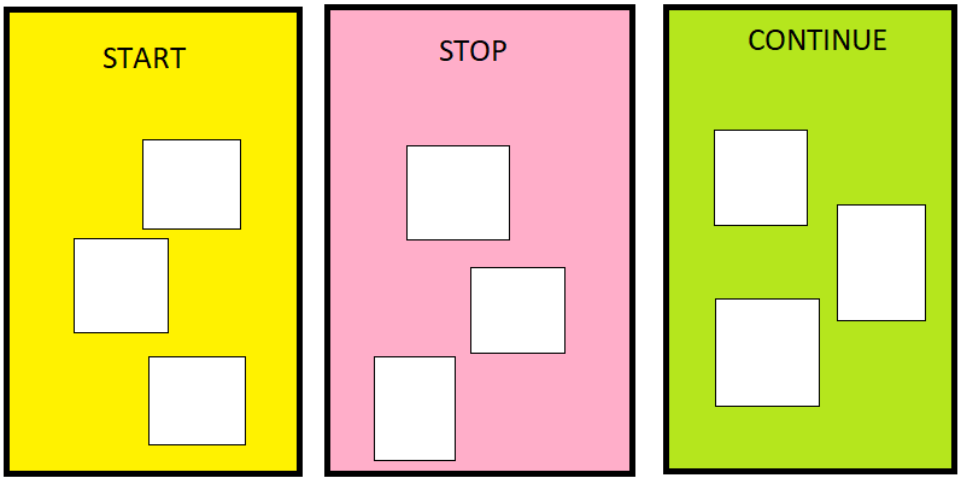
keskustella koko tiimin sekä tuotteen osakkaiden kanssa, mitkä tehtävät ovat tärkeimpiä tehdä seuraavaksi. Tehtävien priorisointi reviewissä helpottaa myös seuraavan sprintin suunnittelua. (Marten 2019.)

Demossa kehittäjät näyttävät, mitä he ovat saaneet aikaan. Reviewissä taas tehtyjen töiden esittelyn lisäksi käydään lävitse myös Sprintin backlogin kohteet. Katsotaan siis, mitkä työt ovat valmistuneet ja mitkä eivät. Kerrotaan huolista ja onnistumisista, mitkä asiat menivät hyvin, mitkä huonosti ja miten niistä selvittiin. Tarkastellaan tuotteen tilaa, eli minkä töiden pitäisi valmistua milloinkin. Päätetään seuraavista toimista, mikä antaa myös paljon apua seuraavaan sprintin suunnitteluun. Käydään lävitse tuotteen muutokset ja niiden vaikutukset tuotteen käyttäjiin. Keskustellaan tulevista muutoksista ja siitä, mikä on niiden aikataulu, rahoitus ja mahdollinen kapasiteetti. (Marten 2019.)

Sprint retrospective

Olipa Scrum-tiimi kuinka hyvä tahansa, aina voi parantaa (Sprint Retrospective n.d). Retrospektiivin tarkoituksena on kehittää tiimiä. Se on sprintin lopussa järjestettävä tapahtuma, jonka pituudeksi on yleisesti määritelty maksimissaan noin yksi tunti jokaista sprintin viikkoa kohti. Tapahtuman pituutta voi soveltaa tiimin koon ja tarpeen mukaiseksi. Retrossa on tarkoitus käydä lävitse asiat mitkä onnistuivat, asiat mitä voidaan kehittää ja mitä tehdään, jotta ensi sprintistä tulisi parempi. (What is a Sprint Retrospective? N.d.)

Retrospectiivin pitämisessä on hyvä ottaa huomioon kaksi tärkeää asiaa. Ensinnäkin tapahtuman olisi hyvä olla aina erilainen. Jos noudatetaan aina samaa kaavaa, siitä muodostuu pitkäveteinen, eikä uusia ideoita enää synny samoista aiheista. Toiseksi, retrossa olisi hyvä olla jokin avustava kuva tai malli, jonka avulla pyritään helpottamaan tiimiläisten osallistumista. Yksi tällainen on “Start, Stop, Continue” -malli (ks. Kuvio 5). Siinä tiimiläiset voivat kertoa, minkä asioiden he haluaisivat alkavan, loppuvan tai jatkuvan tiimin toiminnassa. (Morales 2017.)



Kuva 5. Start, Stop, Continue

On myös hyvä muistaa, että retrospectiiviä ei pidetä vain, jotta voitaisiin parantaa tuotteen etenemistä ja tuloksen saavuttamista. Retrossa on myös tärkeää, että keskustellaan siitä, miten työn tekeminen olisi mukavampaa ja miten tiimin jäsenten välinen kommunikaatio sekä yhteispeli sujuu. (Sprint Retrospective n.d). Tiimi voi esimerkiksi pitää ensin tilaisuuden, jossa keskustellaan vapaammin omista tuntemuksistaan, jonka jälkeen vasta siirrytään puhumaan töistä ja edellisestä sprintistä. Myös tässä tilaisuudessa on suositeltavaa muuttaa kaavaa säännöllisesti ja käyttää avustavia malleja. Esimerkiksi yksi tällainen malli on “Glad, Sad, Mad”. Se vastaa toiminnaltaan “Start, Stop, Continue” mallia, mutta siinä käsitellään tiimiläisten tunnetiloja. On olemassa myös monia muita vastaavanlaisia malleja ja niitä voi keksiä myös itse tiimin tarpeen mukaan. (Morales 2017.)

5 Testaus ketterässä projektissa

5.1 Ketterä testaus

Vuonna 1970 Royce kuvasi ohjelmistokehitysmenetelmän, jota kutsutaan vesiputousmalliksi. Se on edelleen yleisessä käytössä ja on yksi tunnetuimmista menetelmistä. Siinä perusajatuksena on suorittaa yksi vaihe loppuun ennen seuraavaan vaiheeseen siirtymistä. Tästä syystä testauskin suoritettiin vasta lopuksi ja ainoastaan lopuksi, eikä sitä tehty muun prosessin aikana. (Tupper 2011.)

Ketterässä kehityksessä sen sijaan ei testata vasta sitten, kun kaikki on valmista, kuten vesiputousmallissa. Ketterässä kehityksessä testataan jatkuvasti. Scrum-tiimi tähtää yhtenä joukkueena kohti parasta mahdollista laatua. (What is Agile Testing? N.d.) Tästä syystä on tärkeää, että kehittäjät ja testaajat pystyvät tekemään tiivistä yhteistyötä. Kun testaajat pystyvät antamaan palautetta ja kertomaan ongelmista kehittäjille kehitysjakson aikana, muodostuu testauksen ja kehityksen jatkuva integrointi. (Agile Testing 2018.)

Nykyään ohjelmistokehitysprosessit alkavat jo olemaan niin monimutkaisia, että testausmenetelmienkin on kehityttävä, jotta niiden avulla pystyttäisiin testaamaan riittävän kattavasti (Agile Testing 2018). Monet tiimit, jotka yhä käyttävät perinteistä ohjelmistotuotantoprosessia kuten vesiputousmallia, huomaavat, että tuotteen koon kasvaessa myös testauksen määrä kasvaa huomattavasti suuremmassa tahdissa, mikä aiheuttaa ongelmia laadunvarmistuksen pitämisessä ajan tasalla. Tämä aiheuttaa tilanteen, jossa tuoteomistaja joutuu valitsemaan: siirretäänkö julkaisua vai karsitaanko testauksesta. Ja valitettavan usein päädytään valitsemaan näistä se väärä eli testauksen karsiminen. (Redigan n.d.)

Haasteet ketterässä testauksessa

Jatkuvat muutokset ja päivitykset ovat haaste laadunvarmistukselle, mutta tiedostamalla ongelmakohdat voidaan pyrkiä mahdollisimman hyvään lopputulokseen. Yksi yleisemmistä ongelmista ketterässä testauksessa on, että dokumentaatioon ei panosteta tarpeeksi, koska virheitä korjataan ketterästi. Pitää kuitenkin muistaa, että myös ketterät ohjelmistoprosessit vaativat dokumentaation (Linz 7.2). Myös uusien ominaisuuksien nopea käyttöönotto on ongelma ketterässä testauksessa, koska se vähentää testaukseen käytettävissä olevaa aikaa. Tämä johtaa myös siihen, että testisuunnitelmat tehdään liian nopealla aikataululla. Siispä vaikka kehitetään ja testataan ketterästi, tulisi testaajille siitä huolimatta antaa tarpeeksi aikaa testata kohteet sekä tehdä testisuunnitelmat. (What is Agile Testing? N.d.)

Nopea testaus ja DoR

Tärkein ominaisuus ketterässä testauksessa on antaa jatkuvaa palautetta ja tehdä kaikki testaus jatkuvaan palautteeseen tähdäten. Tästä syystä testausta on tehtävä

jatkuvasti, eikä vain vuoroin toteutuksen kanssa. Testiautomaatio on tässä tärkeässä osassa, mutta koska kaikkea testausta ei ole mahdollista automatisoida heti, tarvitaan nopeaa manuaalista testausta. Tämän testausmenetelmän tarkoituksena on testata ominaisuudet intuitiivisesti pelkästään tarinassa annettujen edellytysten perusteella. Nopea testaus mahdollistaa vähän aikaa vievän testauksen uusille ominaisuuksille. Tämä testaustekniikka vaatii taitoa testaajalta. (Linz 7.6)

DoR eli valmiin määritelmä (eng. Definiton of Ready) on tarkastuslista. Sen avulla voidaan varmistaa, että sprintillä oleva tarina on määritelty riittävän hyvin. Tämä tapahtuu tarkastelemalla sitä testaajan näkökulmasta. Mikäli testaaja pystyy luomaan testitapauksen ja hänelle on selvää, milloin testitulos on hyväksytty tai epäonnistunut, on tarina määritelty riittävän hyvin. Vaihtoehtoisesti tätä tapaa voidaan käyttää myös toisinpäin. Silloin tuoteomistaja ja testaaja työskentelevät parina. Tarinalle lisätään testitapaukset, jotka tuottavat puuttuvat tulokset. (mts. 7.6)

Jatkuva integraatio

Perinteisissä menetelmissä on tapana odottaa, kunnes kaikki ohjelmistokoodit on tuotettu ja viety kerralla testattavaksi. Tämä on ristiriidassa Scrumin perusajatuksen kanssa. Syntyy tarpeetonta viivästystä, jos sprintin alussa testiin vietyä kohdetta testataan vasta sprintin lopussa. Tämän vuoksi sprinteissä pitää olla käytössä jatkuva toteutuksen, testauksen, integrointitestauksen ja ongelmankorjauksen ketju. Eli jokainen koodinosa tulee asentaa heti valmistuttuaan testiympäristöön ja testata mahdollisimman pian. (Linz 5.5.)

5.2 Ketterä automaatiotestaus

Erikson (2012) toteaa artikkelissaan, että useimmat ketterästi työskentelevät tai ketteriä menetelmiä työnteossaan käyttävät, että testauksen automatisointi on välttämätöntä, mutta ymmärtävät myös, että sen tekemisessä on erittäin helppo epäonnistua. Jatkuvaa palautetta voidaan antaa vain silloin, kun testiautomaatiota on riittävästi (Linz 7.6). Testiautomaatiosta vaikeaa ketterässä kehityksessä tekee se, että kohde on jatkuvassa muutoksessa. Siitä huolimatta pitäisi pystyä varmistamaan, että

uudet ominaisuudet toimivat vanhojen kanssa (Shilpa 2019). Testiautomaation suunnittelijan kannalta haastavaa on testattavan kohteen jatkuvat muutokset ja niistä johtuvat ylläpitotyöt. (Erikson 2012.)

Automaatiotestauksen haasteet ketterässä kehityksessä

Yksi syy, miksi monet automaatiotestihankkeet epäonnistuvat ketterissä projekteissa, on niille asetetut kohtuuttomat odotukset. Koska automaatiotestaus on lähes välttämätöntä ja todella hyödyllistä ketterässä kehityksessä, aiheuttaa se joskus vääränlaisen kuvan, jossa kuvitellaan sen olevan helppo ratkaisu kaikkiin testausongelmiin. Todellisuudessa se ei kuitenkaan ole sitä. Testiautomaation toteuttaminen vie aikaa ja rahaa sekä vaatii taitoa testaajalta, eikä sillä silti pystytä korvaamaan testaajaa vaan tuetaan tätä. (Dijkstra 2019.)

Edelleen on laajalti sellainen kuvitelma, että testiautomaatio nappia painamalla testaa kaiken hetkessä. Testaajan tullessa kahden kuukauden työn jälkeen näyttämään automaatiotestiä, mikä ei vieläkään osaa testata kuin perusasioita, murskaa yleensä tämän kuvitelman. Valitettavan usein tällaisissa tapauksissa testaajasta, joka on tehnyt ahkerasti hyvää työtä, muodostuu silti syntipukki, joka saatetaan jopa lomauttaa. Tämän vuoksi olisi hyvä keskustella asiasta ja varmistaa, että kaikilla on sama ymmärrys siitä, mitä automaatiolta voidaan odottaa. (Dijkstra 2019.)

Toinen merkittävä syy automaation epäonnistumiseen on sille annettu ajan puute. Ei pitäisi olla yllättävää, että ylläpidettävän ja tehokkaan automaation luominen vie aikaa ja vaatii panostusta. Silti automaatiotestaus on yleensä ensimmäinen asia, mistä lähdetään karsimaan ajan käydessä tiukaksi. Oikeastaan tämä ei välttämättä ole paha asia, koska tuote ja sen ominaisuudet tuovat arvon loppukäyttäjälle eivätkä automaatiotestit. Pitkällä aikavälillä tästä kuitenkin muodostuu ongelma, jos kehitys tapahtuu aina kiireessä ja testauksesta joudutaan karsimaan. (Dijkstra 2019.)

Automaatiotestauksen käyttö ketterässä kehityksessä

Testiautomaation tekeminen on hyvin tilanneriippuvaista, eikä kaikkia ohjeita voi soveltaa kaikissa tilanteissa. Ohjeet tiedostamalla ja niitä omaan projektiin soveltamalla

voidaan niillä kuitenkin edistää hyödyllisen automaation tekemistä ketterässä kehityksessä. Ketterässä automaatiotestauksessa on kuusi ohjetta, joita noudattamalla ja seuraamalla pystyy välttämään tavallisimmat ongelmat:

1. Kohtuulliset tavoitteet. Automaatiotestauksessa tärkeää on kohtuullisten tavoitteiden asettaminen. Kysymällä “miksi” -kysymyksiä on hyvä tapa asettaa odotukset. Miksi teemme tai tarvitsemme testiautomaatiota? Hyvä vastaus kysymykseen on: “Koska haluamme nopeaa palautetta”. Jos vastaus taas on: “Emme halua tehdä manuaalitestausta lainkaan”, tiedetään että odotukset ovat kohtuuttomia.

2. Käsittele testiautomaatiota tuotekehityksenä. Testiautomaation käyttöönotto projektissa on lähes yhtä suuri kuin ohjelmistokehitysprojektin tuominen projektiin, mikä kaikkien tulisi ymmärtää. Projektisuunnittelussa testiautomaatiolle on annettava aikaa ja resursseja. Teknisessä toteutuksessa pitää ottaa huomioon, että se on koodin kirjoittamista, joten on hyvä käyttää kehitysmalleja sekä -käytäntöjä.

3. Omat resurssit automaatiolle. Testiautomaatiossa voidaan menestyä, jos sen luomisesta ja ylläpidosta vastaavalla henkilöllä on aikaa ja osaamista tehdä se.

4. Lähtökohdan valinta. Kuten missä tahansa suuressa projektissa, myös automaatiossa voi olla vaikeaa päättää mistä aloittaa. Hyvä mahdollinen aloituskohta on helpot kohteet, jolloin automaation arvon saa näytettyä parhaiten. Toinen vaihtoehto on kriittiset kohteet, joihin liittyy suurin riski. Kokonaisvaltainen testiautomaatio ei ole hyvä kohde aloittaa.

5. Automaatio ja valmiin määritelmä. Ketterässä mallissa testiautomaatio olisi hyvä ottaa mukaan valmiin määritelmään (eng. DoR). Tässä on vaarana, että määritelmään tulee lause: “Kaiken tulisi olla automatisoitua”, koska joskus automaatio ei ole mahdollista tai järkevää toteuttaa. Sen sijaan määritelmässä voisi olla lause: “Automaatio on päivitetty vastaamaan muutoksia” tai “Testiautomaatio on luotu, jos se on koettu tarkoituksenmukaiseksi tai hyödylliseksi.”

6. Säädä ja opi. Testausautomaatio on ohjelmistokehitystoimintaa. Kuten muussakin ketterässä ohjelmistokehityksessä, myös automaatiossa voi soveltaa ketterää periaatetta eli nopea palaute ja arviointi sekä tehdessä oppiminen. (Dijkstra 2019.)

6 Tutkimustulokset

Tässä luvussa tutustutaan tiimin dokumentaatioon, jonka jälkeen verrataan tiimin alkutilannetta tutkimustuloksiin ja pyritään löytämään kohdat, joissa on mahdollista kehittää tiimin toimintaa sekä esitetään kehittämisehdotukset.

6.1 Testaus

Manuaalinen testaus

Tiimissä kaikki uudet tuotokset testataan oikeaoppisesti manuaalisesti. Manuaalitestauksessa käytetään vain yhtä testausmenetelmää: musta laatikko –testausta, eikä testauksessa käytetä muita testaustapoja. Manuaalitestauksessa ei myöskään käytetä kuin testauksen syvimpiä tasoja eli yksikkö ja integraatiotestausta, jotka keskittyvät pääosin vain uusiin tuotoksiin. Muita testaustyylejä ei ole käytössä, eikä ylempiä testaustasoja käytetä, joten aiemmin luotuja kohteita ei enää myöhemmin huomioida testauksessa.

Manuaalitestauksessa on tärkeää, että kaikki tuotokset testataan ensin manuaalisesti, mikä tiimissä tapahtuu oikein. Testausta voisi kuitenkin syventää ja tehdä kattavammaksi. Testaaja voisi käyttää myös valkoinen laatikko –testaamista, jolloin hän saisi paremman käsityksen testattavasta kohteesta. Tämä auttaisi ymmärtämään paremmin testattavaa kohdetta, jolloin testaaja voisi löytää myös inhimilliset virheet sekä tehdä kattavampia testejä kohteelle.

Uusien testityylien sekä testaustasojen käyttöönotto manuaalisessa testauksessa lisäisi testauksen kattavuutta. Esimerkiksi olisi myös hyvä tehdä välillä etsivää testausta, jotta vanhemmatkin kohteet tulisi käytyä manuaalisesti lävitse. Myös kriittiset

kohteet olisi hyvä testata aika-ajoin myös manuaalisesti, vaikka niillä olisikin kattava automaatio.

Automaatiotestaus

Automaatiotestaus ei ole automatisoitu, vaan se joudutaan ajamaan käsin, mikä lisää riskiä siihen, että ajo jää tekemättä. Automaatiotestaus on laaja, mutta se ei ole kattava. Automaatiotestit käsittelevät monia kohteita, mutta ei erilaisia mahdollisia tapahtumia kohteissa. Automaatiotestit eivät myöskään ota huomioon poikkeustapahtumia, mikä kertoo niiden riittämättömästä syvyydestä. Automaatiotestejä ei myöskään ole rakennettu riittävän tukeviksi, vaan ne kaatuvat välillä myös turhaan. Tämä tekee automaatiotesteistä epäluotettavia, mikä vähentää merkittävästi niiden hyödyllisyyttä.

Tiimillä on hyvä pohja lähteä rakentamaan automaatiotestausta, mutta nykyisessä tilassa se ei ole vielä riittävän kattava ja tukeva ollakseen hyödyllinen. Automaatiotestaus pitäisi saada automatisoitua, jolloin siitä tulisi säännöllistä, mikä nostaisi sen hyödyllisyyttä. Automaatiotestit pitäisi saada siihen tilaan, että ne eivät vaadi testaajalta mitään huomiota, ellei jokin ole mennyt pieleen, ja siinä tapauksessa testien pitäisi antaa selkeä ilmoitus testaajalle.

Automaatiosta olisi myös hyvä saada syvempi. Laaja, pinnallinenkin automaatio toki löytää virheitä mutta huomattavasti vähemmän ja yleensä vain sellaisia virheitä, jotka olisivat muutenkin löytyneet nopeasti. Sen sijaan laaja ja syvä, väsymätön ja unohtelematon automaatio pystyy löytämään ne virheet, mitkä jäisivät muuten löytymättä. Siitä olisi hyvä tehdä myös joustava, jotta turhat kaatumiset saataisiin minimiin. Tällöin virheet otettaisiin aina vakavasti ja automaatiotestaus olisi mahdollisimman hyödyllistä.

6.2 Scrum

Tiimissä on käytössä Scrum-kehys, jossa on jo joitakin Scrumin tapahtumia, jotka toimivat hyvin, mutta jotkut kohdat ovat vielä puutteellisia tai puuttuvat kokonaan. Artefaktien osalta sprint backlogia järjestellään pelkästään sprintin suunnittelussa, mikä

on liian harvoin. Incrementin määrittely jää yleensä hieman vajavaiseksi, minkä vuoksi tiimillä ei ole aina selkeää käsitystä siitä, mikä se on.

Tiimin suurin ongelma Scrumin tapahtumien ja seremonioiden osalta on tietämättömyys. Tiimi olisi halukas tekemään ketterästi, mutta kokemusta ja osaamista ketterässä kehityksessä ei ole vielä riittävästi. Koska ei ymmärretä kaikkien tapahtumien ja seremonioiden merkitystä, ovat ne jääneet vähälle käytölle, niissä ei ole osattu käsitellä oikeita asioita tai niitä ei ole pidetty ollenkaan. Tämä on heijastanut negatiivista vaikutusta koko Scrumin toimintaan.

Sprint planning

Sprintin suunnittelussa on ongelmana liian täydet sprint backlogit. Sprintin backlogille otetaan aivan liikaa tehtäviä, toivoen että ne ehdittäisiin tekemään. Se myös aihetta tiedottomuutta, koska liian täyttä sprintin backlogia on vaikeampi seurata eli on huomattavasti hankalampi nähdä omaa edistymistään tai mitä muut tiimiläiset ovat tehneet. Incrementti eli sprintin tavoite pyritään määrittelemään, mutta se saattaa olla niin iso, että sitä ei pystytä toteuttamaan. Sprint planningistä ei jää sellainen tunne työntekijälle, että määrätyt tehtävät ehditään tekemään, mikä aiheuttaa hämmennystä ja mahdollisesti stressiä tiimin työntekijöissä. Esisuunnittelu on myös puutteellista, jolloin tuoteomistajalla ei välttämättä ole realistista käsitystä suunniteltujen töiden koosta. Tämä näkyy myös siinä, että tehtävien pisteitä on vaikea arvioida, koska tehtävät saatetaan nähdä ensimmäistä kertaa sprintin suunnittelussa. Tämä johtaa siihen, että tehtävien työmäärän arvioiminen voi olla vaikeaa, koska niihin ei ole ehtinyt tutustua.

Sprint-suunnittelulle tulisi asettaa pisteraja. Pisteraja voidaan saada tarkastelemalla vanhoja sprinttejä ja katsoa paljonko olisi realistista pystyä saavuttamaan yhden sprintin aikana. Pisteraja lisäisi selkeyttä siihen, mitä kukin tiimin jäsen on tekemässä tällä sprintillä, sillä sprintin backlogi olisi selkeämpilukuinen, jos siellä olisi vain rajallinen määrä tehtäviä. Kun sprintin backlogilta pystyisi muiden töiden lisäksi myös selkeästi näkemään omat tehtävänsä, se toisi stressaavan tunteen sijaan haastavan tunteen siitä, että työntekijä pystyy suorittamaan omat tehtävänsä. Sprintin alussa ajatus siitä, että pystyy suorittamaan kaikki omat tehtävänsä, on enemmän motivoiva,

kuin ajatus loputtomasta työkuormasta. Kaikki suoritetut tehtävät myös lisäisivät onnistumisen tunnetta sprintin loppuun, mikä olisi myös hyvä motivaatiotekijä.

Sprint planningille voitaisiin pitää myös pre-planning. Tapahtumassa tuoteomistaja, Scrum-master ja kehitystiimi kävisivät seuraavan sprintin kohteet lävitse. Tämä mahdollistaisi sen, että kehittäjät voisivat antaa alustavaa palautetta tuoteomistajalle siitä, onko suunnitellut tehtävät realistisia toteuttaa seuraavalla sprintillä. Kehittäjät myös näkisivät seuraavan sprintin kohteita, jolloin heillä olisi hetki aikaa tutustua niihin ennen sprint planningiä.

Daily

Tämä päivittäinen palaveri toimii tiimillä hyvin. Se on kestolta 15 minuuttia ja se alkaa aina ajallaan samaan aikaan päivästä. Dailyssä käydään tehokkaasti kaikki kolme kysymystä lävitse jokaiselta osallistujalta. Välillä keskustelu karkaa liian syvälle aiheeseen, joka ei kosketa kaikkia osallistujia. Tällöin aikaa menee hukkaan niiltä osallistujilta, joita asia ei kosketa.

Daily toimii lähes oppikirjamaisesti. Jos keskustelu ajautuu ulos dailyn rajoista, voisi keskustelua kaipaavaan aiheeseen liittyvät osalliset pyrkiä sopimaan uuden ajan, jolloin aihe voitaisiin hoitaa kuluttamatta siihen aiheeseen kuulumattomien aikaa. Dailyjen pitäminen seisaaltaan voisi lisätä ihmisten vireystilaa, vastaanottokykyä ja se pitää palaverin tehokkaana.

Weekly

Tiimissä on käytössä weekly, eli viikoittainen palaveri, joka muistuttaa hieman pidennettyä dailyä. Weeklyssä käydään lävitse viikolla tehtyjä töitä, mutta ei pidetä demoja tai käydä lävitse sprint backlogia.

Weeklyssä on hyvä käydä lävitse viikolla tehdyt työt sekä niihin liittyviä ongelmia tai haasteita. Sen lisäksi weeklyyn voisi lisätä myös pienen tiimin sisäisen demon, jossa kaikki halukkaat saisivat näyttää viikolla tekemiään töitä. Tämä lisäisi ymmärrystä ja kohoittasi tiimihenkeä. Weeklyssä avataan sprintin backlogi, mutta sitä käytetään vain esitettävän työn näyttämiseen. Sprint backlogi pitäisi weeklyssä käydä kokonaan

lävitse ja varmistaa, että se on ajantasainen. Lisäksi tavoitteen tarkastelu ja mahdollinen uudelleen suunnittelu kuuluvat osaksi weeklyä. On hyvä tarkistaa viikoittain, mikä on sprintin tilanne ja ollaanko pääsemässä asetettuihin tavoitteisiin. Mikäli tilanne näyttää siltä, että tavoitteeseen ei olla pääsemässä, pohditaan, miten tavoitetta pitäisi mukauttaa, jotta siihen pystyttäisiin pääsemään.

Sprint review

Jokaisen sprintin lopussa tiimi pitää reviewin, joka muistuttaa demoa. Reviewiin tulee paikalle tiimi, asiakasedustajat sekä sidosryhmien henkilöt ja siellä esitellään demoamalla viikon aikana tehdyt työt. Palaveri ei sisällä backlogin läpikäyntiä tai tehtyjen töiden perusteellisempaa tarkastelua.

Review-tilaisuutta olisi tarpeellista pidentää tai jatkaa sen jälkeen reviewiä vielä tiimin kesken. Tässä pidennetyssä tilaisuudessa voitaisiin vapaammin puhua siitä, mitä on tehty ja käydä töitä syvemmällä tasolla lävitse. Esimerkiksi tuotekehittäjät voisivat näyttää tekemiään töitä kooditasolla, joita ei välttämättä ole pystynyt demoamaan visuaalisesti. Lisäksi sprintin backlogin voisi avata ja purkaa kaikki työt auki. Backlogilla voi olla töitä, jotka ovat jääneet kesken, mutta joista olisi silti hyvä keskustella. Myös työt, jotka ovat valmistuneet, mutta joita ei voinut demota, olisi hyvä käydä lävitse.

Tämä käytäntö mahdollistaisi paremmat mahdollisuudet palautteen antoon tiimin sisällä, mikä on erittäin tärkeä osa reviewiä. Palautteen anto tiimiläisille lisäisi tiimin ryhmähenkeä ja toisi onnistumisen tunnetta. Tarkempi töiden analysointi helpottaisi myös kaikkia tiimiläisiä ymmärtämään tehdyt asiat eli lisäisi läpinäkyvyyttä sekä auttaisi tulevaa sprintin suunnittelua, koska kaikilla olisi parempi käsitys siitä, mitä töitä edellisellä sprintillä on tehty.

Sprint retrospective

Retrossa käsitellään jo hyvin kuluneen sprintin ongelmakohtia ja onnistumisia, mutta siinä tulisi käsitellä myös tiimiläisten tuntemuksia. Retro pidetään oikeaoppisesti sprintin lopussa, mutta se on kestoltaan vain puolituntia, mikä on hieman lyhyt aika.

Siellä käsitellään tiimin kesken kulunut sprintti ja pohditaan yhdessä, kuinka se on sujunut. Retrossa ei juurikaan käsitellä muita asioita kuin edellistä sprinttiä ja sen tapahtumia.

Retro voisi olla ajallisesti pidempi, jotta siellä ehdittäisiin käsitellä muitakin asioita kuin ainoastaan kulunut sprintti ja siellä tapahtuneet asiat. Retrossa olisi tärkeää, että siellä kaikki pääsisivät jakamaan omia tuntemuksiaan niin yleisesti kuin sprinttiin liittyen, eikä se olisi pelkästään työasioihin liittyvä palaveri. Retron aikaa voisi pidentää kaksinkertaiseksi nykyisestä ja siinä voisi edelleen olla tapahtuma, jossa käsiteltäisiin sprintti ja siihen liittyvät asiat, mutta sen lisäksi siinä voisi olla jokin kevyempi keskustelu tunnetiloista. Tässä osiossa voisi olla jotain avustavia malleja kuten kuvia, musiikkia tai jotain muuta sellaista, jonka avulla voisi kuvata omaa tunnetilaansa esimerkiksi työmotivaation, työssäjaksamisen, työtaitojen tai vastaavien asioiden osalta.

6.3 Testaus ketterässä tiimissä

Ketterä testaus

Testaus ja ketterä kehitys toimii tiimissä monelta osin oikealla tavalla. Testausta tehdään jatkuvasti uusille kohteille. Testaajien ja kehittäjien välillä on hyvää yhteispeliä ja tieto liikkuu nopeasti, mikä mahdollistaa virheiden korjaamisen ketterästi. Tässä on myös riskinsä, sillä kaikista virheistä olisi myös hyvä pyrkiä tekemään dokumentaatio, ja tämä saattaa unohtua, jos virhe korjataan nopeasti.

Tiimissä on vielä hieman kehittämistä valmiin määritelmän tekemisessä. Tehtävien valmiin määritelmät ovat välillä liian suppeita tai puuttuvat kokonaan, jolloin testaaja ei pysty päättelemään tehtävästä, milloin se on valmis. Tämä aiheuttaa katkon, jolloin testaajan on selvitettävä mitä tehtävässä oli tarkoitus tehdä ja milloin se on valmis. Puutteelliset tehtävien kuvaukset myös poistavat mahdollisuuden nopeaan testaukseen, mikä johtaa katkoihin jatkuvassa integraatiossa.

Dokumentaation tekemiselle tiimissä tulisi sopia selkeä käytäntö, koska sitä on tehtävä myös ketterässä kehityksessä. Sprintin tehtävät tulisi myös kuvata paremmin ja

kirjoittaa valmiin määritelmät. Tällöin ylimääräisiä katkoja ei syntyisi jatkuvassa integraatiossa. Testaaja ja tuoteomistaja voisivat myös testata parityöskentelyä, jolloin he ensin kirjoittaisivat valmiin määritelmän, joka täydentäisi puuttuvat kohdat tehtävään.

Ketterä automaatiotestaus

Automaatiotestaus on otettu hyvin vastaan tiimissä. Kaikilla tiimiläisillä ja tuoteomistajilla oli riittävän realistinen käsitys siitä, mitä automaatiotestauksella pystytään saavuttamaan lyhyessä ajassa. Automaatiotestauksen merkitys on kuitenkin ymmärretty ja sille on annettu aikaa ja resursseja kehittyä.

Tiimin automaatiotestaus on lähtenyt kehittymään tehokkaasti ja se noudattaa lähes kaikkia automaatiotestauksen kuudesta ohjenuorasta. Automaatiotestaukselle on asetettu kohtuulliset tavoitteet. Sillä ei siis pyritä kattamaan kaikkea testaamista tai syrjäyttää manuaalitestausta, vaan parantamaan tiimin ketteryyttä ja jatkuvaa integraatiota. Yksi ohjenuora kuitenkin toteutuu vielä vähän heikosti: automaatiotestien lisääminen valmiin määritelmään. Tämä voi johtua siitä, että valmiin määrittely on muutenkin hieman vähällä käytöllä.

Tiimin automaatiotestauksessa ketterästä näkökulmasta on hyvä pohja, mutta se vaatii vielä paljon työtä. Testien tekeminen on aloitettu oikeaoppisesti helpoista sekä kriittisistä kohteista, mutta niissä olisi vielä paljon tekemistä riittävän kattavuuden saavuttamiseksi. Automaatiotestaus olisi myös hyvä lisätä valmiin määritelmään, kun valmiin määritelmä tiimissä saadaan kehittymään paremmalle tasolle.

7 Johtopäätökset

Tässä luvussa käsitellään opinnäytetyön tutkimuskysymyksiä ja sitä, miten tehty tutkimus onnistui vastaamaan asetettuihin tutkimuskysymyksiin. Johtopäätökset esitetään määriteltyjen tutkimuskysymysten mukaisessa järjestyksessä. Kysymysten alle on koottu vastaus siitä, miten tutkimus on vastannut asetettuun kysymykseen, ja mitä mahdollista lisäarvoa se on tuonut tiimille.

Miten tiimin testausta voitaisiin parantaa?

Tutkimuksen alkutilanteessa testausta toteutettiin suppeasti. Manuaalitestauksessa ei otettu huomioon kaikkia testauksen tasoja ja tyylejä. Automaatiotestaus ei ollut riittävän kattavaa, eikä siinä pohdittu riittävän tarkasti, mitä kohteita olisi hyvä automatisoida ja miten automaatio tulisi toteuttaa. Tutkimuksen ansiosta tiimin testaus on kuitenkin kehittynyt parempaan suuntaan, mikä on vaikuttanut positiivisesti tiimin tuotteiden laatuun.

Manuaalitestauksessa on otettu käyttöön myös valkoinen laatikko –testaus kriittisissä testauskohteissa sekä lisätty ylemmät testauksen tasot. Niiden avulla on onnistuttu parantamaan manuaalitestauksen tehokkuutta sekä kattavuutta, sillä testitapausten laajempi ymmärrys on auttanut tekemään testaamisesta kattavampaa. Lisäksi etsivällä testauksella on onnistuttu löytämään jo aiemmin tehtyihin kohteisiin jääneitä virheitä. Automaatiotestausta on saatu parannettua automaatiotestauksen vaatimusten ymmärryksen myötä. Perusteellisempi ymmärrys siitä, mitä automaatiotestauksella pystytään ja kannattaa testata, on auttanut testaajaa tekemään siitä kattavamman sekä syvemmän. Myös parempi käsitys automaatiotestauksen tarkoituksesta on saanut testaajan tekemään siitä joustavamman sekä automaattisen.

Miten tiimi pystyisi hyödyntämään Scrumia mahdollisimman tehokkaasti ketterässä kehityksessä?

Ennen tutkimusta tiimin Scrum tietämys ei ollut riittävällä tasolla. Tiimissä oli käytössä hyvä Scrum-pohja, mutta koska kaikkien tapahtumien ja palavereiden tarkoitusta ei ymmärretty täysin, jäi Scrum vajavaiseksi. Tämän vuoksi Scrumia käytettiin vain tavan vuoksi eikä se antanut tiimille lisäarvoa, niin kuin hyvän Scrumin tulisi antaa. Tutkimuksen ansioista tiimin Scrumia onnistuttiin parantamaan lisäämällä tiimiläisten tietoa eri tapahtumien merkityksestä. Tämän avulla on myös saatu muutettua tiimiläisten asennetta Scrumia kohtaan. Koska Scrumin tapahtumien hyödyllisyyttä saatiin parannettua ja tiimiläisten asennetta Scrumia kohtaan käännettyä positiiviseen päin, on Scrumin hyödyllisyys kasvanut.

Tietoa lisäämällä lähes kaikkiin Scrumin tapahtumiin ja palavereihin saatiin tuotua uutta näkökulmaa. Osa tapahtumista toimi jo alussa oikein, kuten daily, jota ei tarvinnut juurikaan muuttaa, kun taas jotkut tapahtumat olivat kehityksen tarpeessa ja heijastivat myös negatiivista vaikutusta muihin palavereihin. Esimerkiksi sprint reviewin vajavaisuus teki kyseisestä palaverista vähemmän hyödyllisen kuin sen tulisi olla, mikä heijastui sprint planningiin negatiivisena vaikutuksena. Kun reviewiä saatiin parannettua, se vaikutti myös positiivisesti sprint planningiin. Sprint planningin kehityksen seurauksena myös weekly parani. Koska Scrum on kokonaisuus, jonka osiot vaikuttavat toisiinsa, paras tapa lisätä sen tehokkuutta on pyrkiä parantamaan jokaista osiota niin, että ne vaikuttavat positiivisesti toisiinsa. Tämä on merkittävä tekijä Scrumin tehokkaassa hyödyntämisessä.

Miten testaus, automaatiotestaus ja Scrum toimivat yhdessä?

Testauksen ja ketterän kehityksen välillä on tiimissä jo hyvä pohja, kunhan muistetaan tehdä dokumentaatiota oikein. Tutkimuksen myötä dokumentaation teko on parantunut tiimiläisten ymmärrettyä sen arvon paremmin. Valmiin määritelmän lisääminen tehtäviin sekä automaatiotestien lisääminen valmiin määritelmään toi tiimille ja tiimin testaajalle paremmat mahdollisuudet toteuttaa jatkuvaa integraatiota ilman keskeytyksiä. Tehtävien parempi määrittely mahdollisti myös nopean manuaalitestauksen kohteissa, joissa automaatio ei ole mahdollinen tai kannattava.

Automaatiotestauksen kattavuutta on lähdetty laajentamaan ja sen kannattavuus on parantunut tutkimuksen myötä, koska testaaja on saanut paremman ymmärryksen automaation merkityksestä. Automaatiotestaus on myös saanut säilytettyä oman aikansa ja resurssinsa, minkä vuoksi se on kehittynyt hyödyllisemmäksi ja alkanut maksamaan itseään takaisin.

8 Pohdinta

Opinnäytetyön tavoitteena oli parantaa tiimin toimintaa ja siinä onnistuttiin varsin hyvin. Tiimin testaus on parantunut alusta merkittävästi, mikä lisää tiimin tekemien

tuotteiden tai ohjelmien laatua. Molempia testaustyylejä eli manuaalista ja automaattista on laajennettu. Manuaalitestaukseen on tullut paljon lisää näkökulmia sekä toimintatapoja, ja automaatiotestaus on saanut lisää kattavuutta. Scrum-tietoisuus tiimissä on kasvanut, jonka myötä sitä osataan käyttää paremmin. Tämä on myös lisännyt tiimiläisten motivaatiota sekä ymmärrystä Scrumia kohtaan, koska kuten tutkimuksessa todettiin, väärinkäytettynä Scrum saattaa olla hämmentävä ja turhan tuntuinen, mutta oikein käytettynä tehokas ketterän kehityksen kehys. Myös testaus ketterässä kehityksessä on parantunut, kun on saatu lisää ymmärrystä mitä se tarkoittaa ja mitkä asiat siinä ovat merkityksellisiä.

Teoriatiedon kerääminen osoittautui opinnäytetyössä välillä haastavaksi. Scrum on kehitetty jo vuonna 1993 tai 1986, ja siihen on olemassa pohja, jonka mukaan se on tehty. Siitä huolimatta sen kehityksen ja helpon mukauttamisen takia ihmisillä on valtavasti erilaisia käsityksiä siitä, miten sitä käytätetään ja mikä se on, alkaen siitä, kuka sen kehitti (Krishnamurthy 2012). Tästä johtuen, tietoa on pyritty keräämään monista lähteistä, suosien mieluummin uusia kuin vanhoja lähteitä.

Saadut tutkimustulokset sekä parannusehdotukset ovat monelta osin yleistettävissä, mikäli tiimin tilanne on sama, eli se on vasta siirtynyt ketterään kehitykseen ja Scrumiin. Vaikka työ on tehty ajatellen yhtä tiimiä, on tiimillä olleet kehittämiskohteet yleisiä ongelmia Scrum-tiimeillä ja kehittämisehdotukset toimivat samalla tavalla muissakin tiimeissä, joilla on samoja ongelmia. Ketterä Scrum-tiimi, jolla on ongelmia tai halua kehittää omaa testaamistaan, Scrumia tai testausta ketterässä kehityksessä, voi käyttää työtä oppaana parantaakseen omaa toimintaansa.

Tutkimus suoritettiin vain yhdelle tiimille, suhteellisen lyhyessä ajassa. Tutkimusta voisi jatkaa havainnoimalla tiimin toimintaa tutkimuksen jälkeen ja katsoa miten tiimi jatkaa kehitystään, koska Scrumin oppimiseen voi mennä hyvinkin useita vuosia. Lisätutkimusta voisi tehdä myös ottamalla tutkittavasti jonkin toisen samassa tilanteessa olevan tiimin ja tarkastella onko se kohdannut samanlaisia ongelmia tai haasteita.

Tutkimustulosten perustuvat tiimin dokumentaation oikeellisuuteen, missä on voinut olla virheitä tai mitä ei ole päivitetty ajan tasalle. Tiedon luotettavuuden parantamiseksi tutkija olisi voinut tiimin dokumentaation tukemiseksi dokumentoida omaa tiimin havainnoimistaan. Tutkija teki havainnoivaa tutkimusta, mutta siitä ei ole tehty dokumentaatiota.

Tutkimuksen teoriatieto on kerätty pääosin sähköisistä lähteistä. Tutkimuksen aiheen vuoksi tutkija on suosinut mahdollisimman uutta tietoa, jonka vuoksi sähköisiä lähteitä on käytetty paljon tiedon keräämisessä. Vaikka tutkija pyrki olemaan kriittinen sähköisiä lähteitä valitessaan sekä ottamaan useita lähteitä, on sähköiset lähteet epäluotettavampia kuin painetut lähteet. Tämän vuoksi tutkija on pyrkinyt lisäämään teoriatiedon luotettavuuden lisäämiseksi tietoa painetuista lähteistä.

Tutkimuksen lopussa olisi voinut tehdä teemahaastattelun, jolloin johtopäätösten luotettavuutta olisi saanut parannettua. Nyt johtopäätökset perustuvat vain dokumentoimattomaan aineistoon.

Lähteet

Agile Testing. 2018. Artikkeli ReQtest www-sivulta. Viitattu 10.11.2019. https://reqtest.com/testing-blog/agile-testing-principles-methods-advantages/

Angeling, W. 2.9.2018. The Daily Scrum? Why daily? Viitattu 23.10.2019. https://medium.com/serious-scrum/the-daily-scrum-why-daily-91ca9b1dbd55

AUTOMATION TESTING Tutorial: What is, Process, Benefits & Tools. 2019. Artikkeli guru99 www-sivulta. Viitattu 16.10.2019. https://www.guru99.com/automation-testing.html

A Better Way Of Building Products. N.d. Artikkeli Scrum.org www-sivulta. Viitattu 9.11.2019. https://www.scrum.org/resources/what-is-scrum

Erikson, U. 2012. You can't work Agile without automated testing. Viitattu 10.11.2019. https://reqtest.com/agile-blog/you-cant-work-agile-without-automated-testing/

Ghahrai, A. 2018. Most Common Agile Development Methodologies. Viitattu 12.11.2019. https://www.testingexcellence.com/common-agile-development-methodologies/

Green, P. 2017. Scrum Artifacts. Viitattu 13.11.2019. https://www.scrumalliance.org/learn-about-scrum/scrum-elearning-series/scrum-artifacts

Integration Testing: What is, Types, Top Down & Bottom Up Example. N.d. Artikkeli guru99 www-sivulta. Viitattu 17.11.2019. https://www.guru99.com/integration-testing.html

IT-palvelut. 2019. Artikkeli Kela www-sivulla. Viitattu 14.11.2019. https://www.kela.fi/it-palvelut1

Kananen, J. 2012. Kehittämistutkimus opinnäytetyönä. Jyväskylän ammattikorkeakoulu.

Krishnamurthy, V. 2012. A Brief History of Scrum. Viitattu 18.11.2019. https://www.techwell.com/techwell-insights/2012/10/brief-history-scrum

Lamelas, A. 2018. Top 5 main Agile methodologies: advantages and disadvantages. Viitattu 12.11.2019. https://www.xpand-it.com/2018/10/11/top-5-agile-methodologies/

Linz, T. 2014. Testing in Scrum. Rocky Nook.

Lum, M. 2.3.2016. 5 Scrum Meeting Best Practices: Master the Daily Stand-Up. Viitattu 23.10.2019. https://sprint.ly/blog/scrum-meeting-best-practices/

Margaret, R. 2017. What is Eclipse (Eclipse Foundation)? Viitattu 12.8.2019. https://searchmicroservices.techtarget.com/definition/Eclipse

Marten, D. 2019. Why you should never call the Sprint Review the Sprint Demo. Viitattu 20.8.2019. https://medium.com/serious-scrum/why-you-should-never-call-the-sprint-review-the-sprint-demo-c6a3d4f9a44e

Manual Testing Tutorial for Beginners: Concepts, Types, Tool. 2019. Artikkeli guru99 www-sivulta. Viitattu 21.10.2019. https://www.guru99.com/manual-testing.html

Maynard, C. n.d. What is software testing? Viitattu 17.11. 2019. https://www.atlassian.com/continuous-delivery/software-testing

Morales, K.2017. 5 fun sprint retrospective ideas with templates. Viitattu 17.11.2019. https://www.atlassian.com/blog/jira-software/5-fun-sprint-retrospective-ideas-templates

Parker, L. 2017. The Weekly Scrum: What is it? Viitattu 9.8.2019. https://medium.com/@Leepish/the-weekly-scrum-what-is-it-7fdf631aa4eb

Pittet, S. n.d. The different types of software testing. Viitattu 17.11.2019. https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

Radigan, D. n.d. Engineering higher quality through agile testing practices. Viitattu 10.11.2019. https://www.atlassian.com/agile/software-development/testing

Rehkopf, M. n.d. What is automated testing? Viitattu 17.11. 2019. https://www.atlassian.com/continuous-delivery/software-testing/automated-testing

Release Planning. N.d. Artikkeli ScrumDesk www-sivulta. Viitattu 3.11.2019. https://www.scrumdesk.com/start/manual-for-scrumdesk-start/release-planning

Sprint Preplanning. N.d. Artikkeli ScrumDesk www-sivulta. Viitattu 12.11.2019. https://www.scrumdesk.com/start/manual-for-scrumdesk-start/sprint-preplanning/

Sprint Retrospective. N.d. Artikkeli Mountain goat software www-sivulta. Viitattu 28.10.2019. https://www.mountaingoatsoftware.com/agile/scrum/meetings/sprint-retrospective

Sprint Retrospective. N.d. Artikkeli Managing quickscrum www-sivulta. Viitattu 14.11.2019. https://www.quickscrum.com/ScrumGuide/182/sg-Sprint-Retrospective-Meeting

Tulosyksiköt. 2019. Artikkeli Kela www-sivulla. Viitattu 20.11.2019. https://www.kela.fi/tulosyksikot

Tupper, C. 2011. Learn more about Waterfall Method. Viitattu 10.11.2019. https://www.sciencedirect.com/topics/computer-science/waterfall-method

What is Software Testing? Introduction, Definition, Basics & Types. 2019. Artikkeli guru99 www-sivulta. Viitattu 21.10.2019. https://www.guru99.com/software-testing-introduction-importance.html

West, D. 2019. Sprint planning. Viitattu 6.11.2019. https://www.atlassian.com/agile/scrum/sprint-planning

What is a Sprint Retrospective?. N.d. Artikkeli Scrum.org www-sivulta. Viitattu 14.11.2019. https://www.scrum.org/resources/what-is-a-sprint-retrospective

What is Agile Testing?. N.d. Artikkeli Scrum.org www-sivulta. Viitattu 10.11.2019. https://www.scrum.org/resources/what-is-a-sprint-retrospective

What is BLACK Box Testing? Techniques, Example & Types. N.d. Artikkeli guru99 www-sivulta. Viitattu 16.11.2019. https://www.guru99.com/black-box-testing.html

What is White Box Testing? Techniques, Example & Types. N.d. Artikkeli guru99 www-sivulta. Viitattu 16.11.2019. https://www.guru99.com/white-box-testing.html

Yoni, F. 2016. Robot Framework Introduction. Viitattu 12.8.2019. https://blog.testproject.io/2016/11/22/robot-framework-introduction/