

Opinnäytetyö

Teknologiaosaamisen johtaminen

YTEJOS19

2022

Pia Savolainen

# TESTAUS KETTERÄSSÄ OHJELMISTOKEHITYKSESSÄ

OPINNÄYTETYÖ (YAMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Insinööri (ylempi AMK), Teknologiaosaamisen johtaminen

2022 | 72 sivua

Pia Savolainen

## TESTAUS KETTERÄSSÄ OHJELMISTOKEHITYKSESSÄ

Opinnäytetyössä pyrittiin tunnistamaan haasteita, joita ketterä kehitys tuo testaukselle. Ongelmien tunnistamisen jälkeen perehdyttiin alan kirjallisuuteen ja yritettiin sieltä löytää keinoja vastata haasteisiin.

Työssä sivutaan kahta ketterän kehittämisen viitekehystä ja käsitellään ketterän kehittämisen hyötyjä, periaatteita ja käytäntöjä. Ketterään testaukseen perehdytään menetelmien, testitasojen ja vaiheiden kautta. Lisäksi työssä käydään lävitse testiautomaation osuutta, testauksen kehittämistä ja johtamista.

Johtopäätöksissä esiin nousee testilähtöinen kehittäminen ja testiautomaation tärkeys. Ketterässä kehityksessä ohjelma rakentuu osissa ja jokainen tuotantoon vienti kasvattaa ohjelmaa. Kehittäjät ja testaajat tarvitsevat ketterässä testauksessa testiautomaatio osaamista ja heidän tulee sitoutua sen tekemiseen. Testiautomaation avulla pystytään varmistamaan ohjelmiston toimivuus muutosten jälkeen. Ilman testiautomaatiota ja testilähtöistä kehittämistä regressiotestauksen käytettävä aika kasvaisi liian suureksi ja veisi aikaa kehitystyöltä.

AVAINSANAT:

testilähtöinen kehittäminen, testiautomaatio, ketterä testaaminen

MASTER'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Master's Degree Programme in Technological Competence Management

2022 | 72 pages

Pia Savolainen

## TESTING IN AGILE SOFTWARE DEVELOPMENT

The first step was to identify the challenges that agile development brings to testing. After identifying the problems, the literature in the field was reviewed and attempts were made to find ways to meet the challenges.

The thesis presents two frameworks for agile development and discusses the benefits, principles and practices of agile development. Agile testing is introduced through methods, test levels, and phases. In addition, the work covers the role of test automation, the development and management of testing.

The conclusions highlight test-based development and the importance of test automation. In agile development, the program is built in parts and each production delivery increases size of the program. Developers and testers in agile testing need test automation skills and should commit to doing that. Test automation makes it possible to verify that the software works after changes. Without test automation and test-driven development, the time spent on regression testing would become too large and take time out of development work.

### KEYWORDS:

test-driven development, test automation, agile testing

# SISÄLTÖ

<b>KÄYTETYT LYHENTEET</b>	<b>5</b>
<b>1 JOHDANTO</b>	<b>1</b>
<b>2 KETTERÄ OHJELMISOKEHITYS</b>	<b>2</b>
2.1 Ketterän ohjelmistokehityksen manifesti	3
2.2 Ketterän kehityksen periaatteet	4
2.3 Ketterän kehittämisen käytännöt	5
<b>3 KETTERÄN KEHITYKSEN ORGANISOITUMINEN</b>	<b>10</b>
3.1 Tuoteomistaja	10
3.2 Scrum master	10
3.3 Kehitystiimi	11
3.4 Järjestelmätiimi	11
3.5 RTE (Release Train Engineer)	11
<b>4 KETTERÄN KEHITTÄMISEN TUOTOKSET</b>	<b>12</b>
4.1 Tuotteen kehitysjono	12
4.2 Sprintin kehitysjono	13
4.3 Inkrementti	13
4.4 Valmiin määritelmät	13
<b>5 KÄYTÄNNÖN MALLIT KETTERÄÄN KEHITTÄMISEN</b>	<b>15</b>
5.1 Scrum	15
5.2 SAFe	17
<b>6 KEHITYKSEN DOKUMENTAATIO</b>	<b>21</b>
6.1 Aihio (Epic)	21
6.2 Ominaisuus (Feature)	21
6.3 Kyvykkyys (Capability)	22
6.4 Käyttäjätarina (Story)	22
6.5 Hyväksymiskriteerit	23
6.6 Ei-toiminnalliset vaatimukset	24
<b>7 TESTAUKSEN DOKUMENTAATIO</b>	<b>25</b>

7.1 Testauspolitiikka	26
7.2 Testausstrategia	26
7.3 Testisuunnitelma	26
7.4 Testitapaus ja testiohje	27
7.5 Virheilmoitus	27
7.6 Muutospyyntö	28
7.7 Kehitysidea	28
7.8 Testauksen väliraportti	28
7.9 Testauksen loppuraportti	29
<b>8 PERINTEISEN JA KETTERÄN TESTAUKSEN EROT</b>	<b>30</b>
<b>9 TESTAUKSEN VAIHEET JA PROSESSIT</b>	<b>33</b>
9.1 Ennen sprinttien aloitusta	33
9.2 Testauksen suunnittelu	34
9.3 Testitapausten suunnittelu	34
9.4 Testauksen suoritus	35
9.5 Testauksen lopetus sprintissä tai iteraatiossa	35
9.6 Testauksen lopetus	36
9.7 Testaukseen liittyvät prosessit	36
<b>10 TESTAUKSEN TASOT</b>	<b>37</b>
10.1 Testipyramidi	37
10.2 Yksikkötestaus	38
10.3 Integraatiotestaus	38
10.4 Järjestelmätestaus	38
10.5 Järjestelmien välinen integraatiotestaus	39
10.6 Hyväksymistestaus	39
<b>11 KETTERÄSSÄ TESTAUKSESSA KÄYTETTÄVÄT MENETELMÄT</b>	<b>40</b>
11.1 Testauksen kvartaalit	40
11.2 Riskilähtöinen testaus	42
11.3 TDD (Test Driven Development)	44
11.4 BDD (Behaviour Driven Development)	46
11.5 ATDD (Acceptance Test Driven Development)	46
11.6 Tutkiva testaus	47

<b>12 TESTIAUTOMAATIO</b>	<b>50</b>
12.1 Jatkuva integraatio	51
<b>13 KETTERÄN TESTAUKSEN METRIIKAT</b>	<b>52</b>
13.1 Projektin metriikat	52
13.2 Testauksen metriikat	54
<b>14 KETTERÄN TESTAAJAN TARVITSEMAT TAIDOT</b>	<b>56</b>
14.1 Testaajan rooli ketterässä kehityksessä	56
<b>15 TESTAUKSEN KEHITTÄMINEN</b>	<b>57</b>
15.1 Organisaatiotasoinen kehittäminen	57
15.2 Projektitasoinen kehittäminen	59
<b>16 TESTAUKSEN JOHTAMINEN</b>	<b>60</b>
<b>17 ANALYYSI</b>	<b>63</b>
17.1 Testauksen parhaat käytännöt	63
17.2 Testiautomaatio	64
17.3 Riskit ja vaaranpaikat	65
17.4 Testauksen kehittäminen ja johtaminen	66
<b>18 JOHTOPÄÄTÖKSET</b>	<b>68</b>
<b>LÄHTEET</b>	<b>70</b>

## KUVIOT

- Kuvio 1. Ketteryys
- Kuvio 2. PI- suunnittelun agenda
- Kuvio 3. Ketterän kehittämisen tuotokset
- Kuvio 4. Scrum roolit, aktiviteetit ja tuotokset
- Kuvio 5. Scrum käytännöt

Kuvio 6. SAFen tavoitteet  
Kuvio 7. Empatiakartta  
Kuvio 8. Testauksen dokumentaatio  
Kuvio 9. Vesiputousmalli  
Kuvio 10. Ketterä testaus  
Kuvio 11. Testauksen vaiheet  
Kuvio 12. Testipyramidi  
Kuvio 13. Testauksen kvartaalit  
Kuvio 14. TDD käytäntö  
Kuvio 15. Tutkiva testaus  
Kuvio 16. Esimerkki tutkivan testauksen testiohjeesta  
Kuvio 17. SAFen PI ja iteraatio metriikat  
Kuvio 18. Etäjohtamisen kulmakivet

## TAULUKOT

Taulukko 1. SAFen arvioinnit  
Taulukko 2. TMMi tasot

## KÄYTETYT LYHENTEET

ATDD	Acceptance Test Driven Development. Hyväksymistestilähtöinen kehitys.
BDD	Behavioural Driven Development. Käyttäytymislähtöinen kehitys.
CI/CD	Jatkuva integrointi (CI) ja jatkuva jakelu (CD).
CoP	Communities of Practise. Osaamisyhteisöt SAFe viitekehyksessä.
CTP	Critical Testing Processes. Testausprosessin kehittämismalli.
DoD	Definition of Done. Toiminnallisuus on valmis julkaistavaksi.
DoR	Definition of Ready. Toiminnallisuus on valmis kehitettäväksi.
FMEA	Failure Mode and Effect Analysis. Vika- ja vaikutusanalyysi.
FTA	Fault Tree Analysis. Vikapuuanalyysi.
IP	Innovation and planning iteration. Innovaatio- ja suunnitteluiteraatio.
LACE	Lean-Agile Center of Excellence. Kehitystä ohjaava toiminto.
MVP	Minimum viable product. Pienin mahdollinen arvoa tuottava toiminallisuus
PI	Iteration planning. Iteraation suunnittelu SAFe viitekehyksessä.
PRAM	Pragmatic Risk Analysis and Management. Riskienhallinnan käytäntö.
PRisMa	Product Risk Management. Riskipohjainen testausmenetelmä.
RTE	Release Train Engineer. Julkaisujunan päällikkö SAFe viitekehyksessä.
SAFe	Skaalautuva ketterän kehittämisen viitekehys.
Scrum	Ketterän kehittämisen viitekehys.
SST	Systematic Software Testing. Testauksen kehittämismalli.
STEP	Systematic Test and Evaluation Process. Testauksen kehittämismalli.
TDD	Test Driven Development. Testilähtöinen kehitys.



TMMi	Test Maturity Model Integration. Testauksen kehittämismalli.
TPI NEXT	Test Process Improvement Next. Testauksen kehittämismalli.
QFC	Quality Function Deployment. Riskien hallintamenetelmä.

# 1 JOHDANTO

Opinnäytetyössä tutkitaan testausta ketterässä ohjelmistokehityksessä. Ketterät menetelmät rajataan Scrum ja SAFe ja viitekehyksiin. Rajaukseen on vaikuttanut menetelmien käytön laajuus ja tämän vaikutus testaukseen.

Tutkimusaineistona käytetään alan kirjallisuutta ja artikkeleita. Aineistoa käsitellään deduktiivisesti sisällönanalyysi menetelmällä. Työssä käsitellään ketterän kehittämisen viitekehykset johdatuksena ketterään testaukseen. Viitekehysten läpikäynnillä pyritään avaamaan ympäristöä missä ketterää testausta tehdään. Työssä käsitellään kehityksessä ja testauksessa tuotettavaa dokumentaatiota. Perinteisen ja ketterän testauksen erot käydään lävitse ja perehdytään ketterässä testauksessa käytettäviin tekniikoihin, testauksen vaiheisiin ja tasoihin. Testiautomaation osuutta ketterään testaukseen käsitellään kirjallisuuden sekä analyysin kautta. Testauksessa käytettäviä metriikoita käsitellään testauksen tehokkuuden ja kehityksen edistymisen seurannan kautta. Testauksen organisoitumiseen, kehittämiseen ja johtamiseen liittyviä haasteita ja hyötyjä pyritään tunnistamaan. Testaajan roolia ketterässä kehitystiimissä ja rooliin liittyviä osaamistarpeita käydään lävitse kirjallisuuden kautta. Analyysissä mietitään testauksen parhaita käytäntöjä ja tunnistetaan vaaran paikkoja. Johtopäätöksissä pyritään vastaamaan tutkimuskysymykseen käytännöstä saatujen kokemusten pohjalta.

Tutkimuskysymyksenä: Miten testausta tulee tehdä ketterässä ohjelmistokehityksessä?

## 2 KETTERÄ OHJELMISOKEHITYS

Ketterän kehittämisen juuret ovat iteratiivisessa kehityksessä, jossa asiakas on mukana varhaisessa vaiheessa. Asiakkaalla tai loppukäyttäjällä on mahdollisuus ehdottaa parannuksia ja muutoksia vaatimuksiin sekä antaa palautetta kehitystiimille. Ketterille menetelmille on ominaista yhteistyön korostaminen, usein tehdyt toimitukset ja kyky vastata liiketoiminnan muuttuviin tarpeisiin. (Saleem ym. 2014.)

Ihmisläheisyys, arvon tuottaminen, joustavuus, sitoutuminen, läpinäkyvyys ja muutoksen hyväksyminen ovat ketterän kehityksen arvoja (Ripley & Miller 2020, 17-18). Ketterässä kehityksessä kannustetaan ongelman ratkaisuun tiiminä. Kehitystiimi jakaa vastuun toteutettavan ohjelmiston laadusta. (Baumgartner ym. 2021, 45.)

Käytettävissä oleva aika ja budjetti määräävät toiminnallisuuden laajuuden. Toteutettava toiminnallisuus, luodaan kehitysprojektin aikana tuoteomistajan prioriteettien mukaisesti. Ketterä kehitys eroaa kehitystyötä ostettaessa verrattuna vesiputousmallilla kehitettäessä. Vesiputousmallissa kehitystyön hinta on usein kiinteä ja ohjelmiston valmistumiselle on aikataulu. Ketterässä kehittämisessä, jokaisen sprintin jälkeen tavoitteena on toimiva, arvoa tuottava ja tuotantoon vietävä toiminnallisuus. Teoriassa ohjelmisto voisi valmistua sprintin tai PI:n jälkeen. (Baumgartner ym. 2021, 10 -11.)

Paradoksi on, että ketterä kehitys tuo omat haasteensa testaajille. Nopea kehitystahti, muuttuvat toiminnallisuudet ja kiinteä aikataulu asettaa ketterässä kehityksessä lisäpaineita testaajille, minkä vuoksi testaus on saatava tehokkaaksi ja toimintaa on kehitettävä jatkuvasti. (Watkins & Mills 2011, 118.) Haasteita testaukseen voi tuoda myös kevyt vaatimusmäärittely, kasvava regressiotestauksen tarve, epäjohdonmukainen tai riittämätön yksikkötestaus (Black 2014, 20).

Ketterän kehityksen hyötyjä on isojen ohjelmistoprojektien pilkkominen pienemmiksi ja helpommin toteutettaviksi inkrementeiksi. Kehitystiimin saama palaute asiakkaalta tai tuoteomistajalta ohjelmiston soveltuvuudesta käyttötarkoitukseensa sprintin tai iteraation jälkeen auttaa tiimiä ymmärtämään loppukäyttäjää ja ohjelmiston liiketoiminta tarkoitusta. Testauksen varhainen osallistuminen ohjelmistokehitykseen. Käyttötapausten kirjoittaminen selkeällä ja yksinkertaisella kielellä, joka on ymmärrettävää sekä liiketoiminnalle että tekniikalle. Ketterä kehitys mahdollistaa asiakkaan pyyntöihin ja muutoksiin nopean reagoinnin. Päivittäiset palaverit tiimin ja tuoteomistajan välillä mahdollistavat

ajantasaisen viestinnän. Jatkuvan kehittyminen on yksi isoimmista ketterän kehityksen hyödyistä. (Watkins & Mills 2011, 117.) Kuvio 1 kuvaa miten ketteryyden käsitteet linkittyvät toisiinsa.



Kuvio 1. Ketteryys

## 2.1 Ketterän ohjelmistokehityksen manifesti

Manifesti syntyi talvella 2001 laskettelumajassa Utahissa järjestetyssä ohjelmisto kehittäjien tapaamisessa, jossa he miettivät miten parantaa ohjelmistokehitystä (Cloudt 2021, 37; Baumgartner ym. 2021, 8).

**Yksilöitä ja kanssakäymistä** enemmän kuin menetelmiä ja työkaluja

**Toimivaa ohjelmistoa** enemmän kuin kattavaa dokumentaatiota

**Asiakasyhteistyötä** enemmän kuin sopimusneuvotteluja

**Muutokseen reagoimista** enemmän kuin pitäytymistä suunnitelmassa

Manifestin kirjoittajat antavat arvoa oikean puolen näkökulmille, mutta arvostavat vasemmalla puolella olevia asioita enemmän (Baumgartner ym. 2021, 8). Ketterän kehityksen manifesti on muuttanut tapaa, jolla organisaatiot suunnittelevat, kehittävät, testaavat ja julkaisevat ohjelmistoja (Saleem ym. 2014).

## 2.2 Ketterän kehityksen periaatteet

1. Tärkein tavoite on tyytyväinen asiakas. Asiakkaalle toimitetaan arvoa tuottavaa toiminnallisuutta säännöllisesti.
2. Muuttuvat vaatimukset ovat tervetulleita kehityksen kaikissa vaiheissa. Tällä pyritään kilpailukyvyn parantamiseen muuttuneissa olosuhteissa.
3. Nopea ja säännöllinen kehitystahti, jonka tähtää nopeaan toiminnallisuuksien tuotantoon vientiin.
4. Liiketoiminnan edustajien ja kehitystiimin jatkuva ja päivittäinen yhteistyö koko kehityksen ajan.
5. Kehitystiimi tulisi rakentua motivoituneiden henkilöiden ympärille. Tiimille tulee antaa tukea ja luottamus työn tekemiseen.
6. Tehokkain ja toimivin tapa kommunikoida on kasvokkain käytävä keskustelu.
7. Toimiva ohjelmisto on edistymisen ensisijainen mittari.
8. Ketterät menetelmät edistävät kestävästä kehitystä ja jatkuvaa kehittymistä.
9. Teknisen laadun ja ohjelmiston arkkitehtuurin huomioiminen edesauttaa ketteryyttä.
10. Yksinkertaisuus on tärkeää ja oleellista. Tarpeettoman työn määrää tulisi vähentää.
11. Parhaat käytännöt, arkkitehtuuri ja suunnitelmat luodaan itseorganisoituvissa kehitystiimissä.
12. Tiimin säännöllinen itsearviointi, parantaa tehokkuutta ja kehittää toimintaa jatkuvasti paremmaksi. (Baumgartner ym. 2021, 8; ISTQB (2015b), 9; Black ym. 2017, 291.)

### 2.3 Ketterän kehittämisen käytännöt

Ketterän kehittämisen käytännöt poikkeavat eri viitekehyksissä toisistaan. Usein ketterän kehittämisen valitsevat organisaatiot muuttavat käytäntöjä omaan organisaatioon sopiviksi.

#### **Tuotteen kehitysjonon läpikäynti**

Tuotteen kehitysjonon läpikäynnille ei ole määritelty kiinteää aikataulua Scrum viitekehyksissä. Läpikäynti tulisi kuitenkin järjestää säännöllisesti ja varata sille riittävästi aikaa. Kehitystiimi tulee saada läpikäytävät vaatimukset hyvissä ajoin tietoonsa ja heidän tulee perehtyä toiminnallisuuksiin ennen läpikäyntiä. Tuoteomistaja on ennen palaverin alkua priorisoinut toiminnallisuudet. Läpikäynnissä tuoteomistaja esittelee tiimille ne toiminnallisuudet, jotka hän arvioi tulevan toteutettavaksi seuraavassa sprintissä. Kehitystiimi kysyy tarvittaessa tarkennuksia vaatimuksiin, jolloin työmäärän arviointi helpottuu ja tiimi pystyy paremmin sitoutumaan vaatimuksiin. Läpikäynnissä voidaan myös palauttaa toiminnallisuuksia takaisin vaatimusmäärittelyyn tai pilkkoa määrittelyjä pienempiin osiin. (Rubin 2013, 104-108; Schwaber & West 2020, 101-104.)

#### **Sprintin suunnittelu palaveri**

Ennen palaverin alkua scrum master on laskenut kehitystiimin kapasiteetin tulevalle sprintille. Tuoteomistaja on asettanut tuotteen kehitysjonolle liiketoiminta arvon ja priorisoinut kehitysjonon. Ennen työmäärän arviointia ja toiminnallisuuksien valintaa, tiimille järjestetään mahdollisuus keskustella tuoteomistajan kanssa vaatimuksia ja kysyä niihin tarvittaessa tarkennuksia. Tiimi arvioi vaatimusten työmäärät esimerkiksi pokeri pelin avulla, jossa kaikilla tiimiläisillä on oma korttipakka. Työmäärä arviot näytetään samanaikaisesti, isoimman ja pienimmän arvion antaneet perustelevat kantansa ja arvonta tehdään uudestaan. Palaverissa kehitystiimi päättää tuoteomistajan priorisoimasta tuotteen kehitysjonosta, kuinka monta toiminnallisuutta he pystyvät seuraavassa sprintissä toteuttamaan ja sitoutuvat tavoitteeseen. Palaverin lopputuloksena syntyy sprintin kehitysjono. Valinnan jälkeen tiimi jakaa toiminnallisuudet pienemmiksi palasiksi, eli tehtäviksi. Tavoitteena pilkkomisessa on, että tehtävä valmistuisi päivässä, jolloin sprintin edistymisen seuranta helpottuu. (Rubin 2013, 21-22, 131-133.)

## Daily

Lyhyt noin 15 minuutin päivittäinen palaveri, jossa seurataan sprintin tai iteraation edistymistä. Tiimin jäsenet yksi kerrallaan kertovat mitä saivat aikaiseksi eilen, mitä ottavat työnalle tänään ja onko jokin asia mikä hidastaa tai estää työn edistymistä. Scrum master kirjaa ylös kaikki työtä estävät tai hidastavat asiat ja palaverin jälkeen poistaa työn edistymistä haittaavat esteet. Jos esteitä ei saada poistetuksi, niin scrum master keskustelee tilanteesta ja vaihtoehtoista tuoteomistajan kanssa. (Rubin 2013, 23-25.)

## Demo

Scrum master avaa tilaisuuden ja kertoo sprintin tai iteraation tavoitteet. Palaveri on avoin asiasta kiinnostuneille. Tarvittaville sidosryhmille kannattaa lähettää kutsu palaveriin. Tiimiläiset esittelevät toteuttamansa toiminnallisuudet ja saavat palautetta toiminnallisuuksien sopivuudesta käyttötarkoitukseen sidosryhmiltä tai tuoteomistajalta. Tuoteomistaja hyväksyy tai hylkää sprintissä tai iteraatiossa toteutetun toiminnallisuudesta. Scrum master dokumentoi palautteen sekä toiminnallisuuksien hyväksymiset ja hylkäykset. (Rubin 2013, 363-373.)

## Retrospektiivi

Scrum master avaa palaverin käymällä läpi sprintin tavoitteet ja mittarit. Ja avaa keskustelun tavoitteena on, että tiimi keskustelee avoimesti ja toisiaan arvostavasti sprintin tahtumista. Keskustelussa käydään läpi mikä sprintissä meni hyvin, missä olisi parannettavaa, mitä käytäntöjä kannattaa jatkaa ja mitä ei kannata jatkaa. Palaverissa sana vapaa, kaikki saavat puheenvuoron. Valittava kehityskohde voidaan esimerkiksi valita seuraavalla tavalla. Tiimin jäsenet kirjoittavat kehitysideoita postit-lapuille, jonka jälkeen ne kiinnitetään seinälle. Scrum master yhdistää samat ideat yhteen. Tiimi äänestää valittavat kehitysidean, jonka otetaan käytäntöön seuraavassa sprintissä (Rubin, 2013, 375 - 382).

Retrospektiivissä voidaan tehdä parannuksia testauksen tehokkuuteen, testauksen tuotavuuteen, testitapausten laatuun. Vikojen juurisyyanalyysit voivat johtaa testauksen ja kehityksen parannuksiin. Retrospektiivissä voidaan löytää parannuksia esimerkiksi prosesseihin, organisoitumiseen, työtapoihin tai työkaluihin. Säännöllisesti pidetyt retrospektiivit ovat tärkeitä itseorganisoitumiselle sekä jatkuvalla kehitykselle. Jatkuva kehittyminen koskee sekä kehitystä että testausta. (ISEB (2015b), 14.)

### **Scrum of Scrum – palaveri**

Palaveri pidetään yleensä viikoittain. Palaveri on käytössä SAlFe viitekehyksessä. Palaverin tarkoituksena on tunnistaa riippuvuuksia ja tuoda näkyvyyttä tiimien kohtaamista haasteista ja esteistä. Palaveria fasilitoi RTE ja palaveri on tarkoitettu junan tiimien scrum mastereille. Palaveri on aikarajoitettu, mutta asioita voidaan tarvittaessa käsitellä palaverin jälkeen asiasta kiinnostuneiden henkilöiden kesken. (Leffingwell ym. 2018, 332.)

### **PO sync – palaveri**

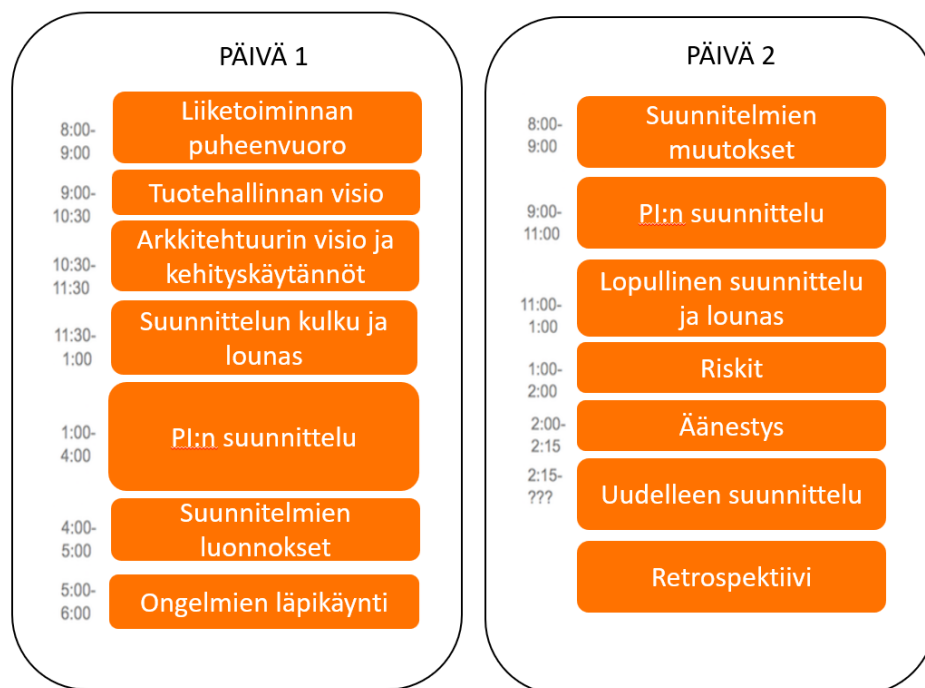
Palaverissa tuoteomistajat kokoontuvat jakamaan tietoa tiimien edistymisestä PI-tavoitteiden suhteen. Palaverissa voi olla mukana liiketoiminnan edustajia. Samoin kuin scrum of scrum-palaveri myös tämä palaveri mahdollistaa keskustelun ongelmista, tiimien riippuvuuksista ja muuttuneista tavoitteista. RTE fasilitoi myös tätä palaveria. Palaveri on aikarajoitettu, mutta palaverin jälkeen asioiden käsittelyä voidaan jatkaa pienemmällä joukolla. (Leffingwell ym. 2018, 332.)

### **PI-suunnittelu**

Sprinttien sijaan toteutusta suunnitellaan yhteistyössä junan kesken pidemmälle ajanjaksolle 8–12 viikkoa. Toki tuo ajanjakso jaetaan osiin, joita kutsutaan iteraatioiksi. Kehitys ja testaus tehdään iteraatioissa ja sprinteissä samalla tavalla. Tiimien yhteistä suunnittelupalaveria kutsutaan PI suunnittelu -palaveriksi. Viitekehysten mukaan tehtynä palaveri kestää kaksi päivää ja se tehdään kasvotusten tiimien kokoontuessa yhteen. Palaverin tavoitteena on asettaa tavoitteet, miettiä riskejä ja riippuvuuksia sekä tehdä



suunnitelmat tuleville iteraatioille. Palaverissa jaetaan myös liiketoiminnan, arkkitehtuurin ja tuotehallinnan näkemyksiä tiimeille. Palaverissa tiimiläiset äänestävät pystyvätkö he sitoutumaan suunnitelmaan. Suunnitelmia hiotaan, kunnes ne saadaan hyväksyttävästi valmiiksi. Suunnitelman kaikkiin toteutukseen valittuihin toiminnallisuuksiin kehitystiimin ei tarvitse sitoutua. Tämä otetaan huomioon laskettaessa tiimin suoriutumista PI:n lopuksi. Suunnitelmat toteutetaan tiimikohtaisilla iteraatioilla. Iteraation viimeistä sprinttiä kutsutaan IP-iteraatioksi. (Leffingwell ym. 2018, 361-378.) Kuvio 2 esittelee PI-suunnittelun kaksi päiväisen agendan.



Kuvio 2. PI- suunnittelun agenda (mukaillen Leffingwell ym. 2018, 364)

### IP-iteraatio

Innovaatio ja suunnittelu iteraatio järjestetään PI:n lopussa. Iteraatio pitää sisällään innovointia, uuden oppimista ja kokeilua. Iteraatioon ei suunnitella toteutettavia toiminnallisuuksia, mutta testausta voidaan jatkaa. Yksi SFAe arvoista on innovaatio, mutta ajan löytäminen ideoille ja muutokselle toimitusten keskellä on vaikea toteuttaa. Innovaatioita

ja kokeilua voi olla esimerkiksi teknisen infrastruktuurin ja muiden kehityksessä tai testauksessa tarvittavien työkalujen evaluointi. SAFe tunnistaa ihmisten olevan elinikäisiä oppijoita. Muutokset teknologiassa, menetelmissä ja käytännöissä ovat esimerkkejä tästä. Oppimista voidaan lisätä kouluttautumalla tai jakamalla tietoa junan sisällä sisäisillä koulutuksilla. Iteraatioissa pidetään PI:n aikana toteutettujen toiminnallisuuksien demo sidosryhmille, minkä jälkeen järjestetään retrospektiivinen ja ongelmanratkaisutyöpaja toiminnan kehittämiseen. IP-iteraatioissa suunnitellaan seuraavaa PI:tä ja tutustutaan mahdollisiin tuleviin toteutettaviin toiminnallisuuksiin. (Leffingwell ym. 2018, 335-339.)

### 3 KETTERÄN KEHITYKSEN ORGANISOITUMINEN

#### 3.1 Tuoteomistaja

Tuoteomistaja vastaa vaatimuksista tuoden niihin liiketoiminnan ja loppukäyttäjän näkökulman. Kommunikoi kehitystiimille toiminnallisuuksien riskeistä ja kertoo toiminnallisuuksien liiketoiminnan arvon kehitystiimille. Tuoteomistajan tulisi olla käytettävissä, jos kehitystiimi tarvitsee tarkennuksia vaatimuksiin. Tuoteomistaja päättää toiminallisuuksien prioriteetit ja pitää toiminnallisuudet prioriteetti järjestyksessä tuotteen kehitysjonolla. Tuoteomistaja hyväksyy tai hylkää sprintissä tai iteraatiossa toteutetut toiminnallisuudet. (Rubin 2013, 15-16, 367; Linz 2014, 12.)

Hyvän tuoteomistaja ominaisuuksia ovat saavutettavissa oleva, hänen tulee omata liiketoiminta osaamista ja olla hyvä viestijä. Tuoteomistajalla tulee olla valtuudet tehdä päätöksiä (Cohen 2010, 138- 39).

#### 3.2 Scrum master

Scrum master toimii kehitystiimin valmentajana ja fasilitoi palavereja. Hänen tehtäviinsä kuuluu kehitystiimin tukeminen ja tiimin perehdyttäminen ketterän kehittämisen arvoihin, periaatteisiin ja käytäntöihin. Scrum master vastaa siitä, että tiimillä on työrauha ja auttaa poistamaan työtä haittaavia esteitä. Scrum master vastaa sprintin kapasiteetin laskemisesta ja viestii sprintin tai iteraation edistymisestä tuoteomistajalle ja muille sidosryhmille. (Rubin 2013, 16, 123; Linz 2014, 12.) SAFessa kapasiteetin laskenta ulotetaan koko PI:lle.

Hyvän Scrum masterin ominaisuuksia ovat vastuullisuus, nöyryys, yhteistyökyky, sitoutuneisuus, vaikutusvaltaisuus ja omaa hyvän ammattitaidon (Cohen 2010, 125-128).

### 3.3 Kehitystiimi

Kehitystiimiin kuuluu kaikki ne henkilöt, joita tarvitaan haluttujen toiminallisuuksien suunnitteluun, toteutukseen ja testaukseen. Tiimi on itseohjautuva ja he jakavat vastuun toteutettavan toiminallisuuden laadusta. (Rubin 2013, 16; Linz 2014, 12).

### 3.4 Järjestelmätiimi

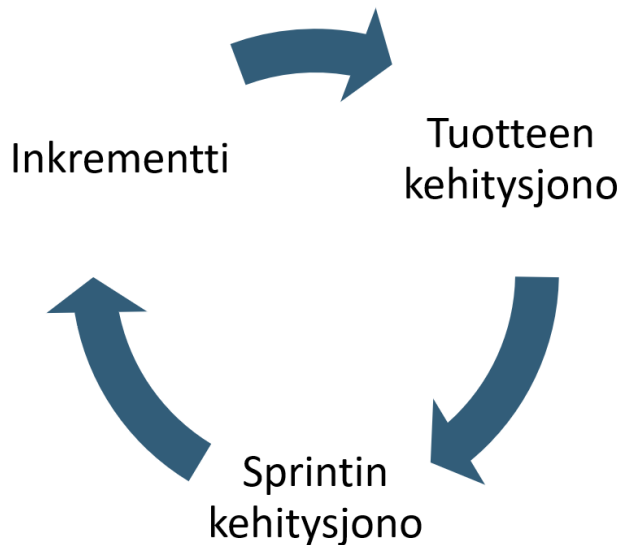
Järjestelmätiimi on käytössä SAFe viitekehyksessä ja se auttaa junan muita tiimejä hallinnoimalla testiympäristöjä ja testidataa, suorittamalla ei-toiminnallista testausta sekä päästä päähän testausta. Tiimi tukee järjestelmien välisessä integraatiotestauksessa ja testiautomaation käytössä. CI/CD-putken hallinnointi ja kehitys kuuluvat järjestelmätiimille. Tiimin tehtävä on auttaa demojen järjestelyissä. (Baumgartner ym. 2021, 60-63.)

### 3.5 RTE (Release Train Engineer)

RTE:n tehtävänä on ohjata junaa, hallita riskejä, edistää kehittymistä, toimia tiimien valmentajana ja auttaa tuottamaan arvoa. Hän myös fasilitoi junakohtaisia palavereita. Roolia voi verrata scrum masteriin niin, että RTE:n työ koskettaa junan kaikkia tiimejä. (Knaster & Leffingwell, 2020, 287-289.)

## 4 KETTERÄN KEHITTÄMISEN TUOTOKSET

Toteutettavat toiminnallisuudet ovat ensin tuoteomistajan hallinnoimassa tuotteen kehitysjonossa ja siirtyvät siitä eteenpäin, kunnes päätyvät tuotantoon. Kuvio 3 kuvaa toiminnallisuuden siirtymistä vaiheesta toiseen.



Kuvio 3. Ketterän kehittämisen tuotokset (mukaillen McGreal & Jocham 2018, 155)

### 4.1 Tuotteen kehitysjono

Tuotteen kehitysjono sisältää kaikki tiedossa olevat ohjelmiston vaatimukset prioriteetti järjestyksessä. Tuoteomistaja hallinnoi kehitysjonoa. Tuotteen kehitysjono muuttuu projektin edetessä, kun siihen lisätään kehitysideat sekä uudet ja muuttuneet toiminnallisuudet. Scrum master varmistaa, että vaatimukset on kirjoitettu sovitussa muodossa ja että ne sisältävät hyväksymiskriteerit toiminnallisille ja ei-toiminnallisille ominaisuuksille. (Linz 2014, 28-30.)

## 4.2 Sprintin kehitysjono

Sprintin kehitysjono muodostuu sprintin suunnittelu palaverissa. Kehitysjonoa täytetään kehitystiimin valitessa toteutettavia toiminnallisuuksia sprintille tuotteen kehitysjonon prioriteetti järjestyksessä, kunnes tiimin kapasiteetti täyttyy. (Linz, 2014, 28-30.)

SAFessa alustavat suunnitelmat iteraatioissa toteutettavista toiminnallisuuksista luodaan PI-suunnittelun aikana.

## 4.3 Inkrementti

Sprintissä tai iteraatiossa toteutettu toiminnallisuus, joka toteuttaa valmiin määritelmän. Inkrementti lisää ohjelmiston toiminnallisuutta, kunnes ohjelmisto on valmis. (Ribley & Miller 2020, 163-164.)

## 4.4 Valmiin määritelmät

Ketterä kehitys tuo mukanaan kaksi käsitettä. DoR määrittelee mitä pitää olla tehtynä ennen kuin toteutus voi alkaa ja DoD määrittelee asiat, jotka pitää toteutua ennen kuin toiminnallisuus voidaan todeta valmiiksi. (McGreal & Jocham 2018, 15, 89, 224–229.)

### **Valmis toteutettavaksi (DoR)**

Määritelmällä pyritään varmistamaan, että ennen toteutuksen alkua organisaation yhdessä määrittelemät vaatimukset täyttyvät. Vaatimuksilla pyritään varmistamaan kehitystiimin tai tiimien yhteinen ymmärrys toteutettavasta toiminnallisuudesta niin vaatimusmäärittelyn kuin aikataulunkin suhteen. Tiedossa oleva aikataulu puolestaan auttaa testaajia suunnittelemaan testauksen. Jos toiminnallisuuden toteuttamiseen tarvitaan useampaa tiimiä aikataulu ja riippuvuudet täytyy olla selvillä, ennen toteutuksen alkamista. DoR-vaatimuksissa on yleensä määritelty mitä vaatimusmäärittelyssä täytyy olla. Kehitys ja testaus hyötyvät laadukkaista vaatimuksista, työ nopeutuu, turha uudelleen määrittely ja määrittelyiden tarkentaminen vähenee tai mahdollisesti poistuu kokonaan. (McGreal & Jocham 2018, 224–229.)

### **Valmiin määritelmä (DoD)**

Valmiin määritelmän täyttyessä toiminnallisuudet ovat julkaisukelpoisia. Valmiin määritelmän käyttämisen on todettu vähentävän teknistä velkaa. Selkeät ja hyvin kommunikoidut vaatimukset lisäävät läpinäkyvyyttä. Valmiin määritelmässä otetaan kantaa muun muassa testaukseen, laatutavoitteisiin, lain ja säädösten tuomiin vaatimuksiin ja dokumentaatioon. McGreal ja Jochan esittelevät kirjassaan esimerkin valmiin määritelmästä, yksikkötestit on suoritettu hyväksytysti, koodi on katselmoitu ja toteuttaa koodille asetetut muoto määrittelyt, ei tiedossa olevia vireitä, koodi on viety versionhallintaan ohjeiden mukaisesti, rajapinnat on dokumentoitu, hyväksymistestit on suoritettu hyväksytysti, tuoteomistaja on antanut hyväksyntänsä, regressiotestit on suoritettu hyväksytysti, julkaisun tiedote on päivitetty, suorituskykytestit on suoritettu hyväksytysti, käyttöohje on päivitetty, tuelle on annettu päivitetty ohje ja tietoturvatestit on hyväksytysti suoritettu. (McGreal & Jocham 2018, 215-224.)

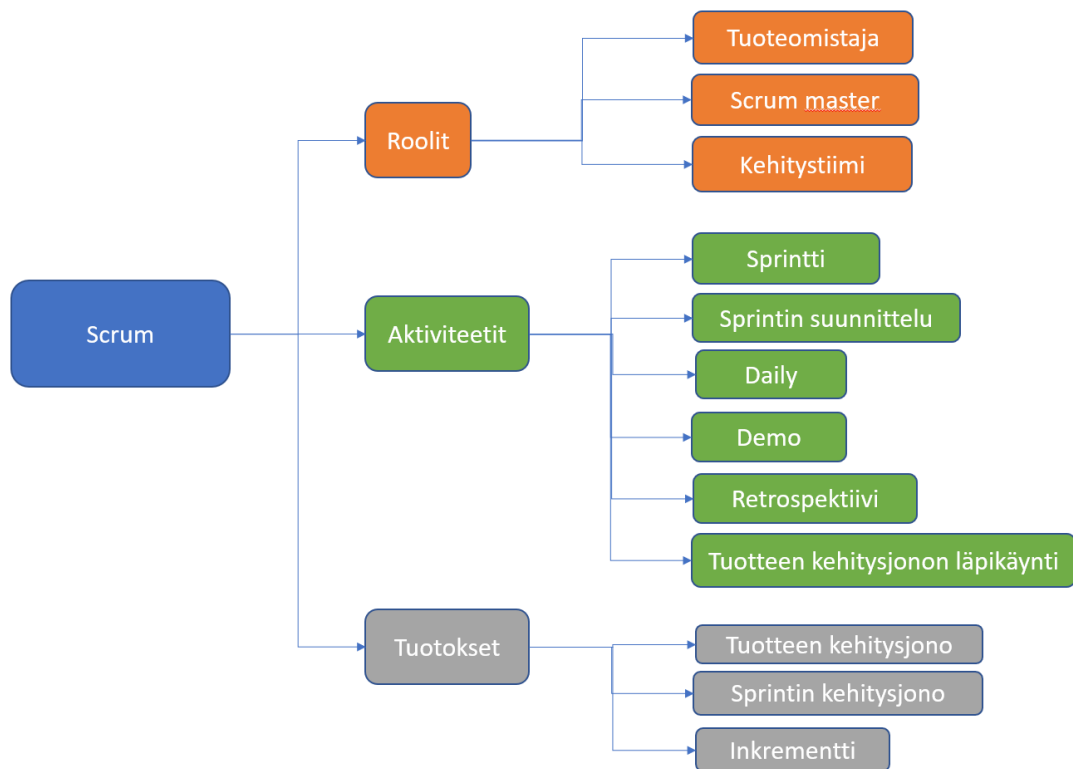
## 5 KÄYTÄNNÖN MALLIT KETTERÄÄN KEHITTÄMISEN

Ketterän kehityksen viitekehyksiä on useita, tässä työssä käydään lävitse niistä kaksi Scrum ja SAFe. Viitekehykset valittiin testauksen näkökulmasta SAFe:ssä on useita kehitystiimejä ja Scrum tiimejä on yleensä vain yksi.

### 5.1 Scrum

Scrum on yleisimmin käytetty ketterän kehittämisen viitekehys. Viitekehityksen esittelivät Ken Schwaber ja Jeff Sutherland OOPSLA konferenssissa vuonna 1995. Työtapa on iteratiivinen noudattaen ketterää manifestia. (Cloudt, 2021, 38.)

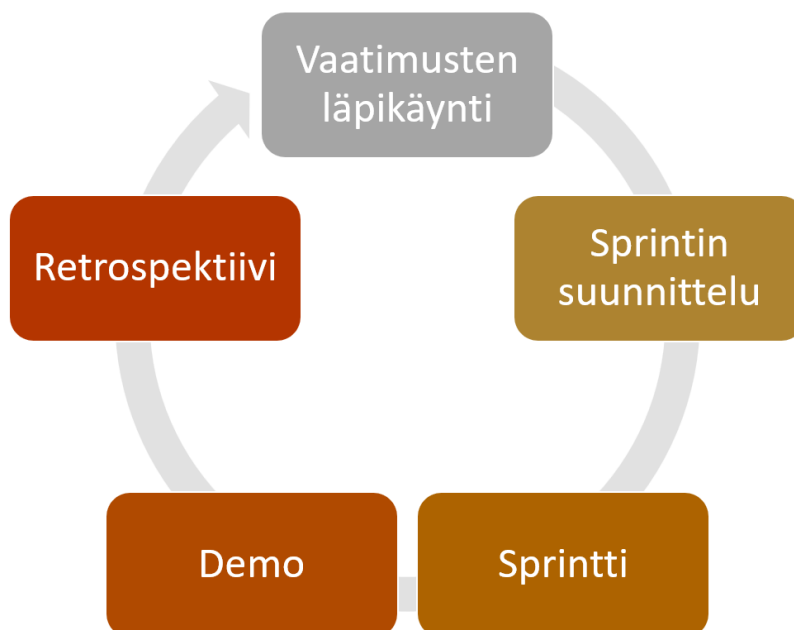
Scrumissa työ on nopeasyklistä, ajallisesti säännöllistä kehittämistä, jonka tavoitteena on jokaisen sprintin jälkeen viedä toimiva toiminnallisuus tuotantoon. Sprintin kesto on normaalisti 1–4 viikkoa. Toiminnallisuuden toteuttaa itseohjautuva ja tarvittavista kyvykkyyksistä koostuva kehitystiimi. (Rubins 2013, 1-2.) Kuviossa 4 selkeytetään Scrum viitekehityksen rooleja, aktiviteetteja ja tuotoksia.



Kuvio 4. Scrum roolit, aktiviteetit ja tuotokset (mukaillen Rubins 2013, 14)



Scrum itsessään ei ota kantaa testaamiseen, muuten kuin, että kehitystiimi on yhdessä vastuussa testauksesta. Ennen vaatimusten läpikäyntiä tuoteomistaja on priorisoinnut tuotteen kehitysjonon. Vaatimusten läpikäynnissä tarkennetaan vaatimuksia tarvittaessa ja palautetaan keskeneräiset vaatimukset takaisin määrittelyyn. Palaverissa tiimi arvioi tuotteen kehitysjonon vaatimusten työmäärän prioriteetti järjestyksessä ja valitsee toteutettavia ominaisuuksia, kunnes sprintin kapasiteetti on täynnä. Palaverin seurauksena syntyy sprintin kehitysjono. Sprintin aikana toiminnallisuudet toteutetaan ja testataan. Toteutuksen ja testauksen edistymistä seurataan päivittäin sprintin edetessä. Sprintin lopuksi pidetään demo, jossa toteutetut toiminnallisuudet esitellään tuoteomistajalle, joka hyväksyy tai hylkää yksittäiset toiminnallisuudet. Hyväksytyt toiminnallisuudet viedään tuotantoon. Ennen seuraavan sprintin alkua pidetään retrospektiivi, jossa käsitellään kuluvan sprintin asioita ja valitaan kehityskohde seuraavalle sprintille. Kuviossa 5 esitellään aktiviteettien järjestys.

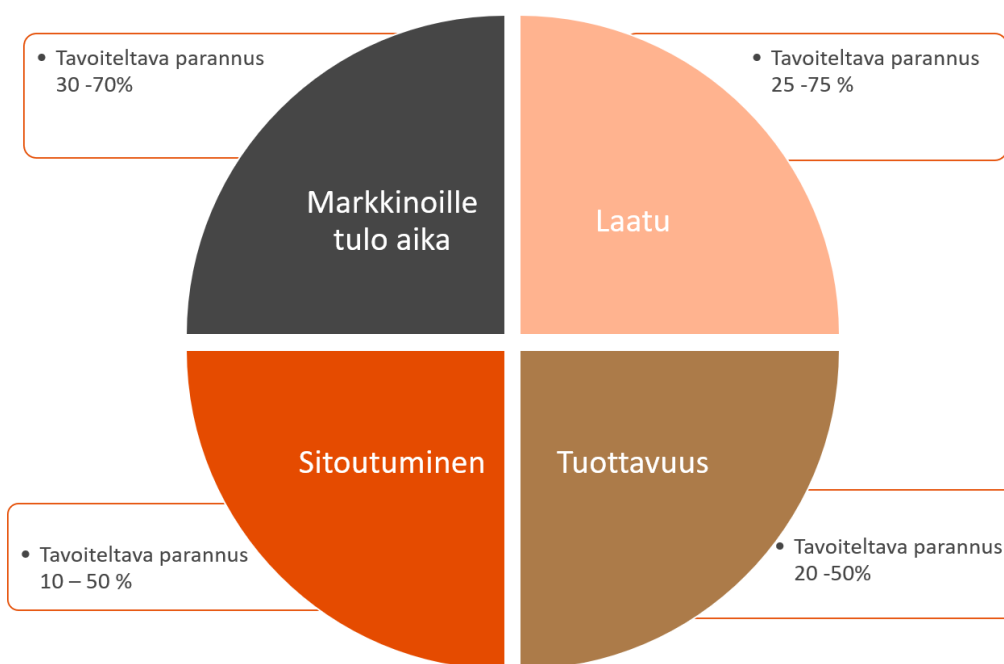


Kuvio 5. Scrum käytännöt

## 5.2 SAFe

SAFe tuo ketterään kehittämiseen skaalautuvuuden ja on konfiguroitavissa eri kokoiisiin projekteihin. Viitekehyksessä on neljä eri mallia toteuttaa ketterää kehitystä. Ero Scrumiin on siinä, että SAFessa monta tiimiä työskentelee yhdessä hyödyntäen ketteriä menetelmiä. Tiimit yhdistetään juna käsitteellä. Erittäin suurissa projekteissa voi juna olla useita.

SAFe auttaa yrityksiä menestymään toimittamalla innovatiivisia tuotteita ja palveluita nopeammin, ennakoitavammin ja laadukkaammin. Organisaatiot voivat mukauttaa käytännöt omiin liiketoimintatarpeisiinsa. (Knaster ja Leffingwell, 2020, 11.) Kuvio 6 kuvaa tavoitteet, joita voidaan tavoitella, kun SAFen valinta kehitys malliksi.



Kuvio 6. SAFe:n tavoitteet (mukaillen Leffingwell ym. 2018, 51)

Ennen PI-suunnittelua liiketoiminta on priorisoinnut toteutettavat ominaisuudet ja scrum master on neuvotellut tiimin kokoonpanon ja laskenut tiimin käytössä olevan kapasiteetin PI:n ajalle. PI:n aikana tiimit valitsevat toteutettavat ominaisuudet, tunnistavat riskit ja riippuvuudet toisiin tiimeihin. Tiimit määrittelevät tavoitteen PI:lle ja päättävät voivatko he

sitoutua kaikkiin toteutettaviksi valittuihin ominaisuuksiin. Tiimit miettivät miten voivat alentaa tiimikohtaisia riskejä ja raportoivat RTE:lle koko junaan koskevat riskit. Ensimmäisen päivän päätteeksi, johto yrittää pienentää junakohtaisia riskejä. Suunnittelua jatketaan seuraavana päivänä, kunnes tiimit saavat tehdyksi lopullisen suunnitelman PI:lle. Palaverissa käsitellään junakohtaiset riskit ja äänestetään PI:n onnistumisesta. Lopuksi järjestetään retrospektiivi. PI-suunnittelussa tehty suunnitelma toteutetaan iteraatioissa, jotka vastaavat sprinttejä. PI:n etenemisen ongelmia seurataan viikoittain tiimien scrum mastereiden kesken ja edistymistä tuoteomistajien ja liiketoiminnan edustajien kesken. Tuotantoon vietejä tehdään sovitun aikataulun mukaa, tämä voi olla esimerkiksi viikoittain. PI:n lopussa pidetään IP-iteraatio, jonka aikana kehitetään toimintaa ja pidetään junakohtainen demo, arvioidaan PI:n tavoitteiden täyttyminen tiimikohtaisesti ja kehitystiimit tekevät itsearviointit PI:n kulusta.

Knaster ja Leffingwell listaavat seitsemään ydinosaamista, joita tarvitaan kilpailuedun saavuttamiseksi ja ylläpitämiseksi. Näiden avulla organisaatio saavuttaa ketteryyden vastaamaan epävakaisiin markkinaolosuhteisiin, muuttuviin asiakkaiden tarpeisiin ja uusiin teknologioihin:

1. Lean-Agile-johtamisen kehittäminen ja soveltaminen. Tarkoituksena on ylläpitävät organisaatiomuutosta antaen yksilölle ja tiimeille mahdollisuuden saavuttaa paras mahdollinen tulos ja motivoitua yhteistyöstä.
2. Kehitystiimin tekninen ketteryys mukaan lukien sisäänrakennettu laatu, käyttäytymiseen perustuva kehitys ja testaus (BDD), tutkiva testaus ja testilähtöinen kehitys (TDD).
3. Jatkuva toimitusputki, joka tuottaa arvoa jatkuvana virtana. Tämä mahdollistetaan DevOps-mallilla.
4. Turvallisuus näkökulman huomioiminen verkko- ja kyberratkaisujen rakentamisessa ja ylläpidossa.
5. Vision ja strategian luominen sekä portfolion priorisointi.
6. Organisaation ketteryys.
7. Jatkuvan oppimisen kulttuuri, jossa kerää tietoa, kasvatetaan osaamista, parannetaan käytäntöjä ja toimintaa.

## Sisään rakennettu laatu

Leffingwell ym. (2018, 197-201) esittelee kirjassaan sisään rakennetun laadun. Junan tiimien tulisi jakaa sisäänrakennetun laadun tavoitteet ja periaatteet. DevOps-käytännöillä parannetaan suorituskykyä ja vähennetään virheitä.

### 1. Virtauksen saavuttaminen

Virtaus saavutetaan CI/CD-putken avulla. Tiimit määrittelevät ja pyrkivät suorittamaan testejä mahdollisimman aikaisin, testejä tulee suorittaa useilla testitasoilla. Kehitys tulisi tehdä käyttäen testiohjattua kehitystä. Vaatimusten tulee sisältää hyväksymiskriteerit, jolloin testauksessa voidaan käyttää myös BDD- tai ATDD tekniikkaa.

Laadun sisään rakentaminen varmistaa, että säännölliset muutokset eivät aiheuta uusia virheitä. Tämä mahdollistaa nopean ja luotettavan toteutuksen. Testiohjattu kehittäminen koskee sekä toiminnallisia vaatimuksia että ei-toiminnallisia vaatimuksia. Nopean virtauksen tukemiseksi testit on suoritettava nopeasti. Tästä syystä ne tulisi automatisoida. Jatkuva integrointi edistää sisäänrakennettua laatua nopeuttaen palautetta. Yhteistyö järjestelmätiimin kanssa lisää kehitystiimin nopeutta ja laatua sekä varmistaakseen virtauksen.

### 2. Arkkitehtuuri ja suunnittelun laatu

Järjestelmän arkkitehtuuri ja suunnittelun laatu ratkaisevat viime kädessä, kuinka hyvin järjestelmä pystyy tukemaan nykyisiä ja tulevia liiketoiminnan tarpeita. Arkkitehtuurin tulisi helpottaa tulevien vaatimusten toteuttamista ja tukea tulevaisuuden liiketoiminnan tarpeita. Arkkitehtuuri ja suunnittelu vaikuttavat ohjelmiston testattavuuteen. Modulaariset ja komponenttipohjaiset arkkitehtuurit yksinkertaistavat testausta. Suunnittelun laatuun tulee kiinnittää huomiota. Huonolaatuista arkkitehtuuria on vaikea ymmärtää ja muokata, mikä yleensä johtaa hitaampaan kehitykseen, joka sisältää enemmän virheitä. Sopivan abstraktiotason löytäminen helpottaa toteutuksen ymmärtämistä ja muokkaamista. Suunnittelun tulee tukea näitä periaatteita ja tarjota keinot periaatteiden ymmärtämisen ja luettavuuden helpottamiseksi.

### 3. Koodin laatu

Uusien ominaisuuksien lisäämisen ja muutosten tekemisen nopeus ja helppous riippuu koodin laadusta. Yksikkötesteissä tulisi varmistaa, että jokaiselle toiminnallisuudelle on automatisoidut testit. Testit suoritetaan automaattisesti jokaisen muutoksen jälkeen, näiden avulla kehittäjät voivat tehdä nopeita muutoksia luottaen siihen, että muutos ei riko ohjelmistoa. Testit toimivat myös dokumentaationa ja helpottavat koodin ylläpitoa ja ymmärtämistä. Ohjelmiston parempi ymmärtäminen johtaa nopeampaan kehitykseen vähemmällä virheillä ja pienemmällä refaktorointi tarpeella.

Parityö kehityksessä luo ja ylläpitää laatua. Se lisää tietoa, ottaa huomioon eri kehittäjien näkökulmat ja tukee parhaita käytäntöjä. Tiimin osaaminen nousee, tiimin opissa toinen toisiltaan. Kollektiivinen omistajuus vähentää riippuvuuksia ja tukee nopeaa toimitusta, tällöin koodi ei ole yhden henkilön omistuksessa.

### 4. Järjestelmän laatu

Koodin ja suunnittelun laatu varmistavat, että järjestelmän toiminnallisuudet voidaan helposti ymmärtää ja muuttaa. Järjestelmän testaus varmistaa, että järjestelmät toimivat odotetulla tavalla. Jaettu ymmärrys järjestelmästä mahdollistaa nopean työnkulun. BDD tekniikka tukeutuu yhteistyöhön, jossa tuoteomistaja ja kehitystiimin jäsenet sopivat tarinan ja ominaisuuden tarkasta käyttäytymisestä. BDD-tekniikan käyttäminen auttaa kehittäjiä rakentamaan oikean toiminnallisuuden ensimmäisellä kerralla. Jatkuva integrointi tarjoaa kehittäjille nopean palautteen. CI/CD-putkessa testiautomaation osuus on suuri, mutta osa testeistä tulee suorittaa manuaalisesti tutkivana testauksena.

### 5. Julkaisun laatu

Julkaisulaadun avulla organisaatio voi mitata toiminnan tehokkuutta. Mitä nopeammin organisaatio julkaisee, sitä nopeammin se oppii ja sitä enemmän arvoa se tuottaa. Pienemmät muutokset mahdollistavat nopeammat, useammin tehtävät ja vähemmän riskialttiit julkaisut, mutta vaativat automaattisen CI/CD-putken laadun varmistamiseksi.

## 6 KEHITYKSEN DOKUMENTAATIO

Vaatimukset perustuvat käyttäjäryhmien tarpeiden ymmärtämiseen, liiketoiminta vaatimuksiin ja mahdollisesti myös kilpailija-analyysihin.

IEEE määrittelee hyvälle vaatimukselle seuraavat ominaisuudet oikeellisuus, yksiselitteinen, johdonmukainen, priorisoitu, testattavissa, muokattavissa ja jäljitettävissä (Lucassen ym. 2016).

### 6.1 Aihio (Epic)

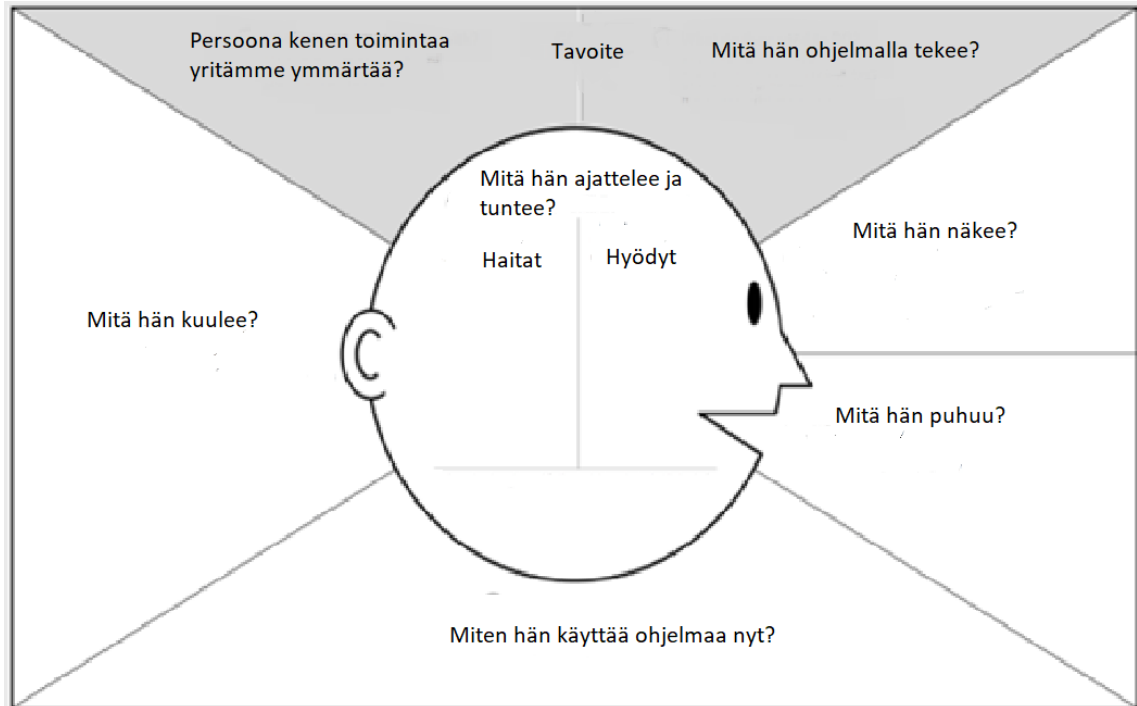
Aihio on kooltaan suuri ja sisältää useita liiketoiminta arvoja ja ominaisuuksia. Aihioita on kahden tyyllisiä liiketoiminnalle arvoa tuottavia ominaisuuksia ja arkkitehtuurisia teknistä arvoa lisääviä, kuten esimerkiksi teknologian vaihto. Ennen aihion toteutusta tulee se analysoida hyvin ja toteutus tulisi tehdä osissa hyödyntäen toteutamis-sykliä. Aihio kannattaa dokumentoida hyvin ja sen tulisi olla liiketoiminnan yhdessä hyväksymä ja määrittelemä. Toiminnallisuuden kannattavuutta voidaan arvioida toteuttamalla toiminnallisuudesta MVP (Minimum viable product). MVP on asiakas arvoa tuottava toiminnallisuus, mutta kooltaan pienin mahdollinen (Leffingwell ym. 2018, 609 - 614).

### 6.2 Ominaisuus (Feature)

Ominaisuus on usein osa aihiota. Määrittely tulee sisältää liiketoiminnalle tuotettavan arvon, liiketoiminta säännöt, data mallin, käyttöliittymän layoutin, esimerkit ja hyväksymiskriteerit. Vaatimuksia määriteltäessä voidaan hyödyntää käyttäjä tarinoita ja empatiakarttaa. Ei-toiminnalliset vaatimukset tulee myös määritellä, näitä ovat muun muassa turvallisuus, luotettavuus, suorituskyky, ylläpidettävyys, skaalattavuus ja käytettävyys. Ominaisuudet jaetaan käyttäjätarinoiksi, jotka toteutetaan, testataan ja esitellään sekä hyväksytään ennen julkaisua. (Leffingwell ym. 2018, 305 – 324.)

Empatiakartat auttavat tunnistamaan asiakkaita ja heidän tarpeitaan. Ne auttavat tiimejä kuvittelemaan, mitä loppukäyttäjä ajattelee, tuntee, kuulee ja näkee käyttäessään ohjelmistoa. Mitä enemmän tiimillä on tietoa asiakasta, sitä todennäköisemmin se pystyy

suunnittelemaan asiakasta tyydyttävän toiminnallisuuden. (Scaled Agile.) Kuvio 7 kuvaa yhden empatiakartan.



Kuvio 7. Empatiakartta (mukaillen Scaled Agile)

### 6.3 Kyvykkyys (Capability)

Kyvykkyudet lisäävät järjestelmän teknistä tehokkuutta tai mahdollistavat tulevaisuuden liiketoiminta tarpeita. Ne voivat olla arkkitehtuurisia tai infrastruktuurisia parannuksia. Arkkitehtuuriset parannukset voivat esimerkiksi korjata olemassa olevia ongelmia. Ratkaisuja on yleensä useita ja ennen valintaa vaihtoehdot tulee analysoida. (Leffingwell ym. 2018, 311 – 315.)

### 6.4 Käyttäjätarina (Story)

Käyttäjätarinaa määriteltäessä tulisi käydä keskustelu sidosryhmien välillä käyttäjätarinan yhteisen ymmärryksen takaamiseksi. Käyttäjätarinalla tulee olla hyväksymiskriteerit.

Hyvän käyttäjätarinan omaa seuraavat ominaisuudet, hyvin muotoiltu ja selvällä kielellä kirjoitettu pitäen sisällään roolin, toiminnallisuuden ja syyn toiminnalle, on toisista käyttäjätarinoista riippumaton, minimalistinen, johdonmukainen, ratkaisulähtöinen, yksiselitteinen, arvioitavissa ja ristiriidaton. Muotoa tärkeämpi on kuitenkin sisältö. (Lucassen ym. 2016).

Käyttäjätarinoiden kirjoittamiseen voidaan käyttää INVEST-tekniikkaa, joka mainitaan monissa lähteissä (McGreal & Jocham 2018, 198-199; Leffingwell 2011, 105 – 111; Pugh 2011, 55.)

- Itsenäinen, käyttäjätarinan tulee olla riippumaton muista tarinoista.
- Neuvoteltavissa, käyttäjätarinaa voidaan muuttaa, kunnes sitä aletaan toteuttaa.
- Arvoa tuottava, käyttäjätarina tulee tuottaa arvoa loppukäyttäjälle.
- Arvioitavissa oleva, käyttäjätarinan on oltava kooltaan sellainen, että sen toteutus pystytään arvioimaan.
- Pieni, käyttäjätarinan tulee olla sopivan kokoinen toteutettavaksi sprintissä tai iteraatiossa.
- Testattava, käyttäjätarinan tulee olla yksiselitteinen ja testattavissa.

## 6.5 Hyväksymiskriteerit

Hyväksymiskriteerit omistaa tuoteomistaja, mutta hänen tulee sitouttaa kehitystiimi ja tarpeelliset sidosryhmät niiden määrittelyyn. Hyväksymiskriteereillä varmistetaan, että toteutuksesta tulee määrittelyn mukainen, lisätään yhteistä näkemystä ja vähennetään käyttöönoton riskiä. Hyväksymiskriteerit voidaan kirjoittaa SMART-tekniikalla. Tekniikka pitää sisällään viisi ominaisuutta. Hyväksymiskriteerin tulisi olla täsmällinen, mitattava, saavutettavissa oleva. Tällä tarkoitetaan, että vaatimus pystytään toteutettaman nykyisellä tekniikalla ja henkilöstön osaamisella. Kriteerin tulisi myös olla relevantti eli liitettävissä tavoitteisiin ja aikasidonnainen. Hyväksymiskriteerin olisi hyvä alkaa esimerkiksi varmistaa, että tai testaa, että, näin niihin saadaan konkretiaa. (McGreal & Jocham 2018, 205-209.)

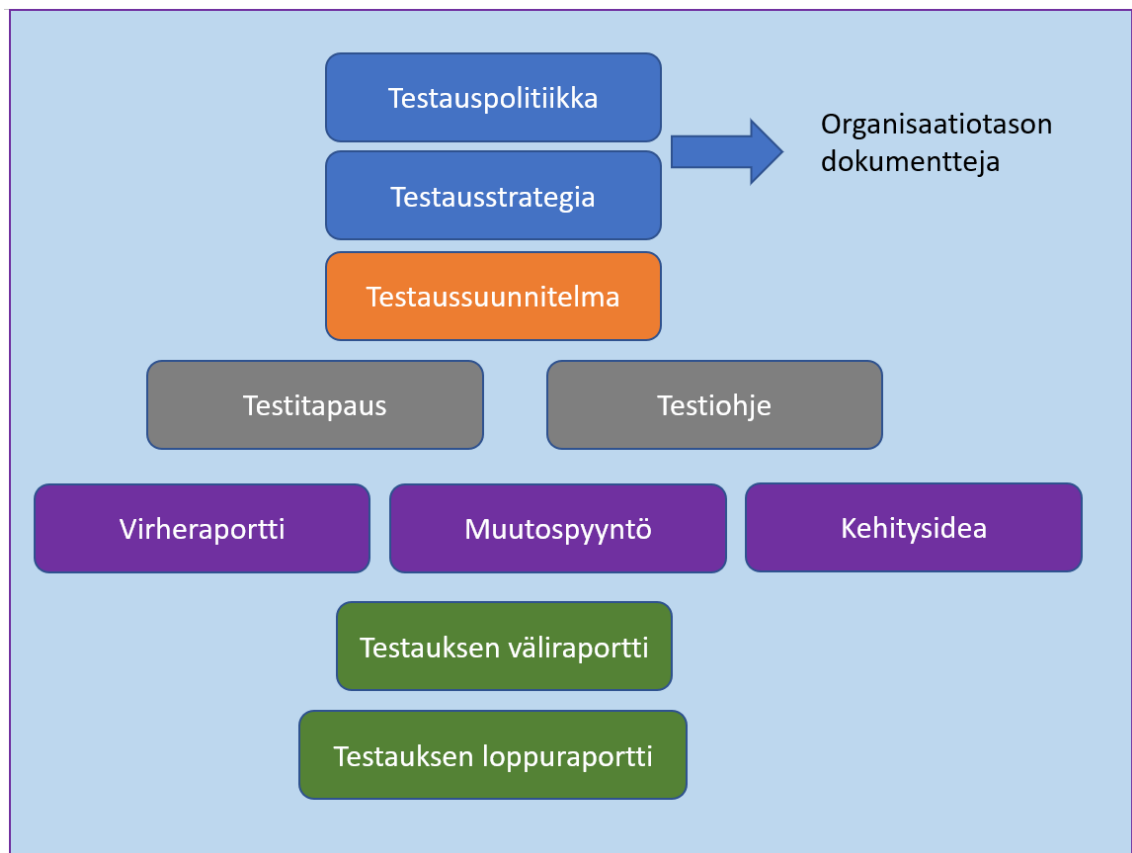


## 6.6 Ei-toiminnalliset vaatimukset

Ei-toiminnalliset vaatimukset määrittelevät ohjelman ominaisuuksia liittyen turvallisuuteen, luotettavuuteen, suorituskyykyyn, ylläpidettävyyteen, skaalautuvuuteen ja käytettävyyteen. Vaatimukset toimivat rajoitteina ohjelmiston suunnittelussa. Vaatimusten ymmärtäminen on tärkeää, sillä määrittelemällä vaatimukset liian tiukoiksi ohjelmistosta voi tulla liian kallis. Jättämällä vaatimukset määrittelemättä tai määrittelemällä ne liian heikoiksi ohjelmisto ei välttämättä ole riittävän suorituskyykyinen, turvallinen tai käytettävä aiottuun käyttöön. (Leffingwell ym. 2018, 317-322.)

## 7 TESTAUKSEN DOKUMENTAATIO

Dokumentaatiota tehtäessä on huomioitava lukijat ja heidän saamansa hyöty dokumentaatiosta. Turhaa dokumentaatiota ei kannata tehdä. Testauspolitiikka ja -strategia ovat organisaatiotason dokumentteja ja muut koskettavat projekteja. Kuvio 8 kuvaa testauksen dokumentaation tasoja.



Kuvio 8. Testauksen dokumentaatio

### 7.1 Testauspolitiikka

IEEE:n määritelmä testauspolitiikka kuvaa testauksen tarkoitusta, tavoitteita, periaatteita ja laajuutta organisaatiossa sekä kuinka testaus tukee organisaation yleistä liiketoimintastrategiaa ja missiota. (Reid 2017, 835.)

### 7.2 Testausstrategia

IEEE:n määritelmä testausstrategia määrittelee yleiset vaatimukset testaukselle, ja antaa yksityiskohtaista tietoa siitä, miten testaus tulee suorittaa. Organisaatiolla voi olla erilliset testausstrategiat perinteisille ja ketterille projekteille. (Reid 2017, 870.)

### 7.3 Testisuunnitelma

Testisuunnitelma on yksityiskohtainen kuvaus testauksen tavoitteista, laajuudesta, keinoista ja aikataulusta. Testisuunnitelman voi jakaa useammiksi testisuunnitelmaksi testustasojen tai erityyppisten testausten kesken. (Reid 2017, 927.)

IEEE 829 standardin mukaan testisuunnitelman tulisi sisältää seuraavat asiat (Reid 2017, 927-1101):

- Dokumentilla on yksilöivä tunnus, dokumentin tarkoitus on kuvattu ja suunnitelmassa on kuvaus testattavasta järjestelmästä.
- Testauksen tavoitteet, rajoitukset sekä riippuvuudet toisiin järjestelmiin, projekteihin tai tiimeihin. Testauksen taso, linkit vaatimuksiin jäljitettävyyden aikaansaamiseksi.
- Testauksen laajuus. Testauksen laajuudessa tulee ottaa kantaa myös asioihin, jotka rajautuvat testauksen ulkopuolelle. Laajuutta määritettäessä tulisi ottaa kantaa testauksen metodeihin ja miettiä käytetäänkö testauksessa testiautomaatiota.
- Testauksen prosessin kuvaaminen, testauksen lähestymistapa sekä käytettävät metriikat ja työkalut.
- Testauksen aloitus ja lopetus ehdot sekä testauksen keskeytys ja uudelleen jatkamisen ehdot.

- Testauksen löydösten määrittely sekä niille määritellyt vakavuus tasot. Prosessi kuvaukset ja käytännöt virheiden käsittelyyn, muutos pyyntöjen käsittelyyn ja kehitysideoille.
- Testaukseen ja laatuun liittyvät riskit, niiden vakavuus ja todennäköisyys.
- Testauksen vaiheet ja resurssointi sekä alustavat työmääräarviot, kulut ja aikataulu.
- Ohjeet testaajille ja mahdolliset testaajien koulutustarpeet.
- Testidatan erityistarpeet esimerkiksi salaus ja tarvittavat testiympäristöt. Testiympäristöistä vastaava tulee olla nimettynä.
- Raportointikäytännöt ja testauksen tuottamat dokumentit.
- Testauksen vastuulliset ja hyväksymiskäytännöt.

#### 7.4 Testitapaus ja testiohje

Testitapauksessa on vaihe vaiheelta kuvattuna, kuinka testi etenee. Vaihe pitää sisällään kuvauksen mitä tehdään ja mitä odotetaan lopputulokseksi sekä mahdollisen testidatan, jota vaiheessa tarvitaan.

Testiohje pitää sisällään suunnitelman testauksen sisällöstä. Testauksen edetessä suunnitelmaa voidaan muuttaa, mutta tällöin suunnitelman muutos tulee kirjata testiohjeeseen, josta testaukseen jälkeen muodostuu testiloki. Testiohje sisältää myös testaukseen käytettävän ajan. Testauksen jälkeen testiohjeeseen kirjataan testauksen aikana löydetty virheet ja muut huomiot.

#### 7.5 Virheilmoitus

Virheilmoituksen tarkoituksena on kuvata testauksessa havaittuja ongelmia. Virheet tulee kirjata niin, että ne kestävät aikaa, jos virheitä ei päästä korjaamaan heti. Ketterissä menetelmissä korjauksen toivotaan toteutuvan samassa sprintissä, missä se on löydetty. Organisaatioissa hallitaan virheitä yleensä työkalulla, myös monissa testauksenhallintatyökaluissa on tämä ominaisuus. (Reid 2017, 1235- 236.)

IEEE:n mukaan virheilmoituksessa tulee olla yksilöivä tunniste, konteksti, virheen kuvaus, joka sisältää vaiheet, miten virhe saadaan toistettua, tehdyt havainnot ja mahdolliset kuvankaappaukset ja lokitiedot virheestä, virheen vakavuus, mahdollisen riskin ja

virheen tilan, virheen raportioijan ja testiympäristön missä virhe on löydetty. (Reid 2017, 1235-1236.)

Virheilmoituksesta kannattaa tehdä niin hyvä, että kehittäjä ymmärtää heti mistä on kysymys. Lisätietojen kysyminen hidastaa korjausta. Tavoitteena kannatta pitää sitä, kuka tahansa testaajista pystyy uudelleen testaamaan vian korjauksen jälkeen virheilmoituksen perusteella. Virheilmoitus voi sisältää mahdollisen kiertotien virheelle. Tämä yleensä alentaa vakavuutta. Epäselvissä vaatimuksissa virheeseen voi liittää odotetun tuloksen selventämään virhettä.

## 7.6 Muutospyyntö

Muutospyyntöissä tulisi olla linkki tai tunniste vaatimukseen, jota halutaan muuttaa. Muutos voi koskea vaatimusta tai toteutusta, eli vaatimus itsessään voi olla oikein. Esimerkki tällaisesta virheestä voi olla toiminnallisuus, joka itsessään toimii, mutta suorituskyky kärsii. Muutospyyntöön pitää sisältää perustelut muutokselle ja mahdollinen ratkaisuehdotus, jos tällainen on tiedossa.

## 7.7 Kehitysidea

Kehitysidea voi koskettaa uutta toiminnallisuutta tai parannusta jo toteutettuun toiminnallisuuteen. Kehitysideassa tulee olla kuvaus uudesta toiminnallisuudesta ja perustelut sille miksi kehitysidea kannattaa toteuttaa.

## 7.8 Testauksen väliraportti

Testisuunnitelmassa on määritetty testauksen aikana seurattavat metriikat. Metriikat kannattaa valita niin, että ne voidaan tuottaa automaattisesti työkalun avulla. Metriikoiden avulla seurataan testauksen edistymistä ja raportoidaan tilanteesta sidosryhmille testisuunnitelmassa sovitun käytännön mukaisesti.

## 7.9 Testauksen loppuraportti

Testauksen loppuraportissa tehdään yhteenveto testatuista asioista ja saavutettiin testaukselle asetetut tavoitteet. Listataan asiat, jotka poikkesivat testisuunnitelmasta sekä listataan avoinna olevat virheet ja analysoidaan testauksen tulos. Raportti pitää sisällään hyväksynnän. (Burnstein 2003, 224.) Loppuraportissa olisi hyvä käsitellä projektista saatuja oppeja ja kehityskohteita sekä antaa suositus tuotantoon viennistä.

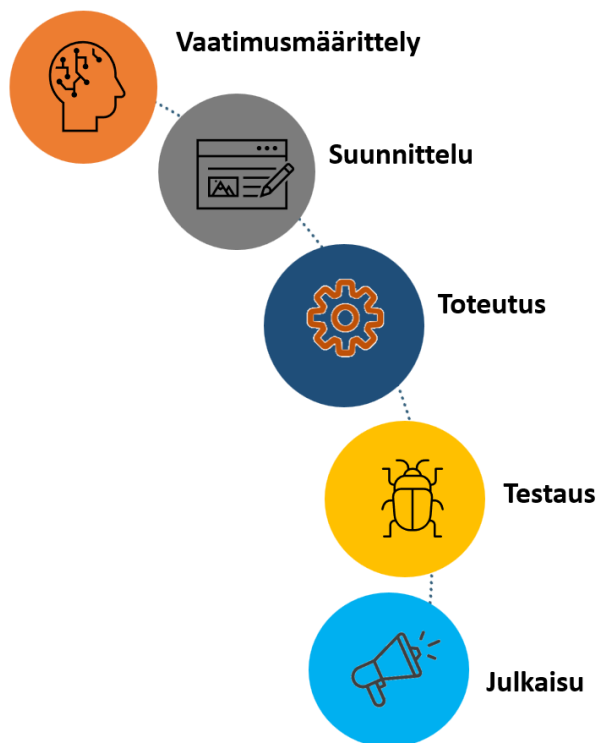
## 8 PERINTEISEN JA KETTERÄN TESTAUKSEN EROT

Itse testaus on samaa toiminta riippumatta siitä, onko se ketterää vai ei. Myöskään testauksen tavoitteet eivät muutu. Ketterä testaus tuo mukanaan omat haasteensa, mutta myös tiivistää yhteistyötä kehittäjien ja tuoteomistajan kanssa ja antaa testaajille mahdollisuuden olla mukana projektin alusta asti.

Testauksen tavoitteena on löytää virheitä, varmistaa että testattava ohjelmisto täyttää toiminnalliset ja ei-toiminnalliset vaatimukset. Vähentää ohjelmiston laatuun liittyvien riskien tasoa ja lisätä luottamusta ohjelman toimivuuteen. (Black ym. 2017, 5-6.)

Tavoitteisiin voi lisätä vielä seuraavat asiat, testaus varmistaa, että testattava ohjelmisto ja siihen liitetyt järjestelmät toimivat saumattomasti yhteen, tuottaa tietoa tuotannon viennin päätöksen tueksi, varmistaa ohjelmistossa olevan tiedon eheys, arvon tuotto sekä korkea tekninen laatu ja koettu laatu. Ohjelmisto tulee täyttää sopimuksiin, lakeihin, säädöksiin perustuvat vaatimukset. (ISTQB 2018, 12-13.)

Vesiputousmallissa vaatimukset kerätään ja määritellään vaatimusmäärittelyssä. Tämän jälkeen ohjelmisto suunnitellaan, toteutetaan ja testataan. Kun ohjelmisto on valmis, se otetaan käyttöön. Ohjelmiston toteutuksessa kuluu aikaa ja loppumetreillä voidaan huomata, ettei ohjelmisto ole toteutetussa muodossa käyttäjiä parhaalla mahdollisella tavalla hyödyttävä. Tämä saattaa johtaa muutoksiin, kunnes ohjelmisto lopulta täyttää käyttäjän odotukset. (Baumgartner ym. 2021, 6.) Kuvio 9 kertoo vesiputousmallin vaiheiden järjestyksen.



Kuvio 9. Vesiputousmalli

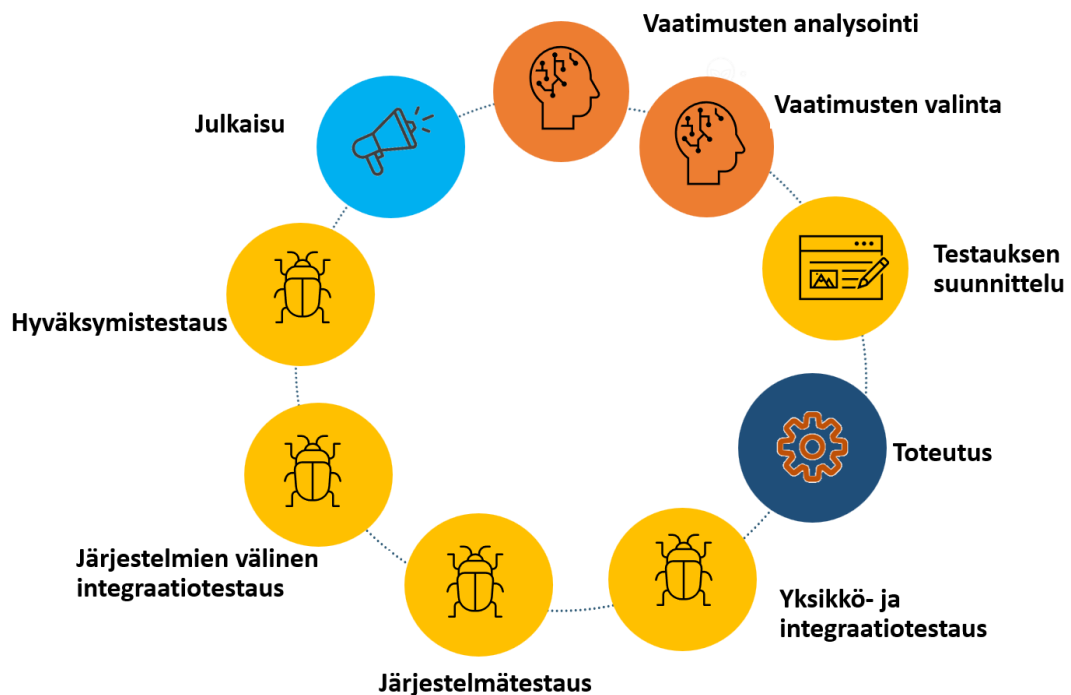
Ketterässä kehityksessä testaus tapahtuu nopeammassa syklissä ja muutoksia voi tulla paljon vaatimusten muuttuessa tai refaktoroinnin yhteydessä. Testiautomaation hyödyntäminen regressiotestauksessa korostuu. Kehitystiimi, johon testaajat kuuluvat ovat yhdessä vastuussa ohjelmiston testauksesta ja laadusta.

Erona perinteiseen vesiputousmallin testaus tapahtuu ketterässä testauksessa osissa koko projektin, eikä projektin viimeisenä vaiheena. Tämä mahdollistaa kehitystiimin saaman palautteen toiminnallisuuksista sprintin jälkeen. Vesiputousmallissa vaatimukset määritellään projektin alussa ja ovat muuttumattomia. Ketterässä kehityksessä vaatimukset voivat muuttua ennen toteutuksen aloitusta tai projektin aikana. Testaukseen tämä aiheuttaa useita regressiotestikierroksia. Vesiputousmallissa on yleensä erillinen tiimi ja testauspäälikkö suorittamassa testausta. Ketterässä testauksessa testaajat kuuluvat kehitystiimiin ja testauspäälikköä ei ole. Kehitystiimiin liittyminen lisää kommunikointia ja yhteistyötä kehittäjien kanssa ja testaajat ovat mukana koko



ohjelmistokehitysprosessin ajan, osallistuen vaatimusten analysointiin sekä työmäärän arviointiin. (Gregory & Crispin 2015, 51-56.)

Ketterässä testauksessa testausprosessi tulisi luoda joustavaksi ja tukemaan muuttuvia vaatimuksia. Joustavuus edellyttää jatkuvaa yhteistä ymmärrystä tavoitteista. Ketterässä testauksessa korostuu tehokkuus, jotta tämä saavutettaisiin, on testauksen suunnittelulle varattava aikaa. Tehokkuutta voidaan parantaa seuraamalla testausprosessin vaihtelua, jolloin parannusta vaativa vaihe saadaan näkyväksi. Erityisesti testauksen pullonkauloihin kannattaa kiinnittää huomioita ja yrittää poistaa niitä aiheuttavat esteet. Heikompi osa-alueita tulee parantaa ja säilyttää parhaat käytännöt. (Perry 2006, 817–833.) Kuvio 10 kuvaa ketterää kehitystä testauksen näkökulmasta.



Kuvio 10. Ketterä testaus

## 9 TESTAUKSEN VAIHEET JA PROSESSIT

Kuvio 11 kuvaa testauksen vaiheita projektin aloituksesta aina projektin lopetukseen asti. Kehällä kuvataan projektin aikana toistuvia vaiheita.



Kuvio 11. Testauksen vaiheet

### 9.1 Ennen sprinttien aloitusta

Ennen sprintin aloitusta tulee varmistaa, että vaatimusten tarkkuus tukee testausta. Valmiin määritelmät on kommunikoitu kehitystiimille ja ne ovat tiimin hyväksymät. Työkalut on valittu ja niiden käyttö on ohjeistettu. Tiimin resurssointi koon ja osaamisen suhteen on määritetty ja tiimi on saatu kokoon. Testiympäristöjen tarve on määritetty ja ne on asennettu ennen sprintin aloitusta. Testidatan saanti pitää määrittää ja päättää pitääkö testidatan sensitiiviset tiedot häivyttää. Julkaisuputki ja julkaisuun liittyvät käytännöt tulee määritellä. Tuotantoympäristö täytyy myös määritellä ja pystyttää. (Baumgartner ym. 2021, 121.)

Testauksen osalta valitaan testauksenhallintatyökalu, ohjeistetaan ja koulutetaan sen käyttö. Tehdään testisuunnitelma ja mietitään, pystytäänkö hyödyntämään testiautomaatiota ja määritellään mitattavat metriikat. Päätetään, tarvitaanko projektin aikana erillistä tietoturvatestausta, suorituskkytestausta tai käytettävyyden testausta tai mahdollisesti muita ei-toiminnallisen testauksen osa-alueita toteutettavasta ohjelmasta riippuen.

## 9.2 Testauksen suunnittelu

Testauksen suunnittelussa kartoitetaan tarvittava testidata, perehdytään vaatimuksiin ja tehdään tarkentavat kysymykset tuotteen kehitysjonon läpikäynti palaveriin. Päätetään testauksen laajuus ja testaus tekniikat sekä testiautomaation osuus, jos testiautomaatiota on mahdollista hyödyntää. Suunnittelun yhteydessä tulee tarkistaa ja ylläpitää aikaisemmin kirjoitetut testi ja varmistaa aikaisemmin tehdyn testiautomaation ylläpidon tarve ja testien ajautuminen.

## 9.3 Testitapausten suunnittelu

Testauksen suunnittelussa kannattaa hyödyntää positiivista ja negatiivista testausta. Nämä ovat toisiaan täydentäviä testaustekniikoita. Positiivisen testauksen tarkoituksena on varmistaa, että järjestelmä täyttää sille asetetut vaatimukset. Negatiivisessa testauksessa varmistetaan, että ohjelmisto ei tee sitä, mitä sen ei pitäisi tehdä, eikä väärin käytöstä aiheudu ongelmia. Lisäksi tulee tarkistaa, että virheilmoitukset ovat kuvaavia ja käyttäjää ohjaavia. Negatiivisessa testauksessa virheiden löytyminen on helpompaa. (Watkins & Mills 2011, 11-18.)

Testitapausten valintaan vaikuttaa testauksen kohde ja käytettävä tekniikka. Testiautomaatiota käytettäessä suositellaan perinteistä testitapausta sisältäen odotetun tuloksen tai BDD tekniikalla tehtyjä testitapausta. Hyväksymiskriteereihin perustuvaan ATDD tekniikkaan käyttö edellyttää, että hyväksymiskriteerit on määritelty. Tutkivatestaus suoritetaan manuaalisesti ja sen käyttö on yleistynyt ketterän testauksen ansiosta. Testitapaukset tulee linkittää vaatimuksiin jäljitettävyyden aikaansaamiseksi. Riskipohjaista lähestymistapaa kannattaa hyödyntää testitapausten priorisoimiseen.

Erittäin yksityiskohtaiset testitapaukset vievät aikaa pois testaukseen käytettävästä ajasta ja toisaalta liian ylimalkaiset testitapaukset lisäävät testaukseen käytettävää

aikaa. Testitapauksia kirjoittaessa on hyvä tunnistaa henkilöt, kenelle niitä tehdään ja valita tarkkuustaso tämän mukaan. Testitapausten tulee kuitenkin olla uudelleen käytettäviä.

#### 9.4 Testauksen suoritus

Testaus kannatta aloittaa sieltä mistä virheitä oletetaan löytyvän, jolloin virheen korjaukseen ja uudelleen testaukseen saadaan lisää aikaa ja näin vähennetään riskiä, että toiminnallisuus ei ehdi valmistua sprintin loppuun. Toki tähän vaikuttaa se missä järjestyksessä toiminnallisuudet ovat testattavissa. Testiautomaatio skriptien kirjoittaminen ja valmiiden testien siirtäminen regressiotesteihin kuuluu tähän vaiheeseen. Testauksen aikana voidaan joutua luomaan testidataa. Tulosten ja löydösten kirjaaminen tulee tehdä sovitun prosessin mukaisesti. Metriikaksi kannattaa valita mittarit, jotka pystytään tuottamaan automaattisesti. Tällöin testauksen sen hetkinen tila on nähtävissä kaikille kiinnostuneille. Näin saadaan lisättyä läpinäkyvyyttä projektin etenemiseen. Testauksen etenemistä seurataan mittareiden lisäksi päivittäisissä palavereissa.

Virheraattia voidaan hyödyntää tiedon jakamiseen ja virheen vakavuuden päivittämiseen. Virheraadissa virheen löytäjä esittelee virheen ja vastaa tarkentaviin kysymyksiin virheestä, jonka jälkeen virheen vakavuutta voidaan miettiä virheraadin kesken ja mahdollisesti päivittää sitä. Virheraadissa voidaan päättää virheen korjaamisen aikataulusta. Korjatut virheet tulee uudelleen testata ennen virheen sulkemista. Uudelleen testauksessa tulee huomioida vaikutukset myös regressiotestauksen osalta.

#### 9.5 Testauksen lopetus sprintissä tai iteraatiossa

Valmiiksi saadut toiminnallisuudet esitellään tuoteomistajalle demossa, jossa tämä hyväksyy tai hylkää ne (Ripley & Miller 2020, 175). Testaus toimittaa tuoteomistajalle väli-raportin testauksesta ja kommunikoi mahdolliset avoinna olevat virheet, jos niitä on. Kehitystiimi saa palautteen toteuttamastaan toiminnallisuudesta demossa. Demon jälkeen järjestetään retrospektiivi, jossa käsitellään kuluneen sprintin asioita ja mietitään miten toimintaa voisi parantaa tulevassa sprintissä tai iteraatiossa. Kehitysehdotukset voivat käsitellä mitä tahansa toiminnallisuutta sprintissä tai iteraatiossa myös testausta. Kehitysehdotusten tekemiseen ja valintaan osallistuvat kaikki kehitystiimin jäsenet. (Linz 2014, 158.)

## 9.6 Testauksen lopetus

Testauksen loputtua testauksesta kirjoitetaan loppuraportti. Testiympäristöt ajetaan alas ja testidata hävitetään. Automatisoidut testit siirretään tuelle tai taholle, joka ylläpitää ohjelmistoa ja jos ohjelmistoon on jäänyt auki olevia virheitä, niin näiden kommunikointi ja mahdollinen korjausaikataulu.

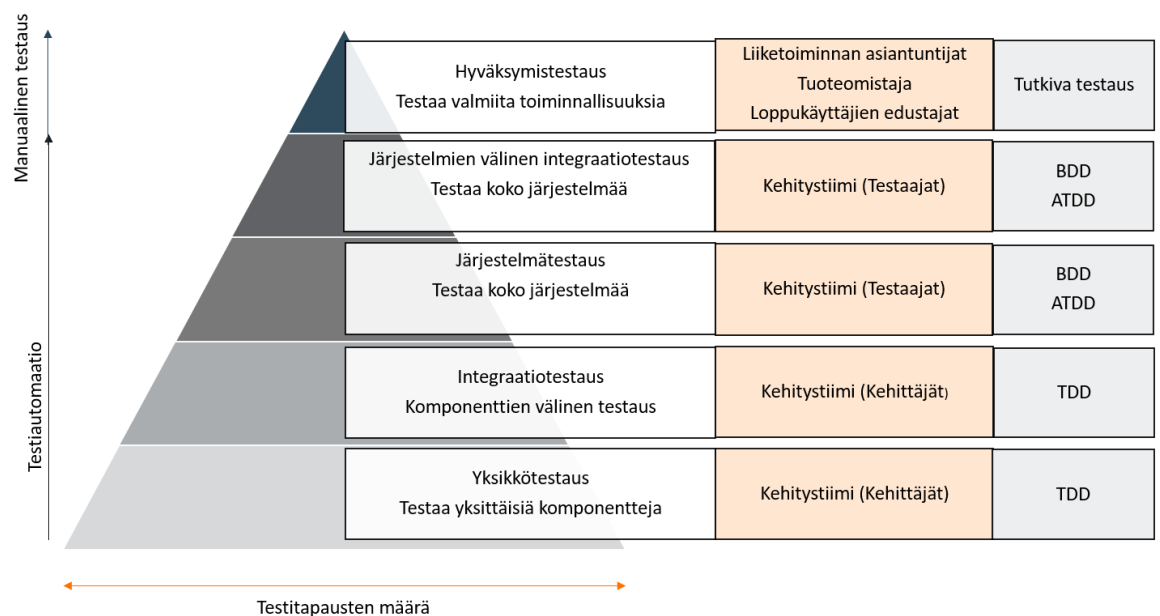
## 9.7 Testaukseen liittyvät prosessit

Testaukselle tulee määritellä prosessi. Testausprosessin lisäksi on hyvä määritellä prosessit virheiden käsittelyille, muutospyyntöjen hallinnalle sekä uusien kehitysideoiden hallinnalle. Jos ohjelmisto toteutetaan organisaation ulkopuolella, niin prosesseihin on hyvä mallintaa koko polku, myös miten niitä käsitellään organisaation ulkopuolella ja missä kohtaa vastuu siirtyy organisaatioiden välillä.

## 10 TESTAUKSEN TASOT

Testitasot eivät muutu ketterässä testauksessa. Toiminnallisuudet valmistuvat sprintissä eri aikaan, joten testitasot tapahtuvat päällekkäin eri toiminnallisuuksille. Kuvio 12 kuvaa testipyramidin. Testipyramidia on laajennettu lisäämällä siihen tasot, tekijät ja suositellut testausmenetelmät. Testipyramidin tarkoituksena on korostaa testiautomaation tarvetta ketterässä testauksessa.

### 10.1 Testipyramidi



Kuvio 12. Testipyramidi (mukaillen Cloudt 2021, 164)

Ohjelmistoa tulee testata eri tasoilla. Tyypilliset testitasot pyramidin pohjalta huipulle ovat yksikkö-, integraatio-, järjestelmätaso, järjestelmien välinen integraatio ja hyväksymistaso. Testipyramidi korostaa testitapausten suurempaa määrää alemmilla tasoilla ja ylemmille tasoille siirryttäessä testien määrän tulisi vähentyä. Testipyramidin ajatuksena on, että viat poistetaan mahdollisimman aikaisin. Luottamus ohjelman toimintaan kasvaa siirryttäessä huippua kohti. (ISTQB (2015 b), 30.)

## 10.2 Yksikkötestaus

Yksikkötestauksen tavoitteena on estää virheiden syntyminen, pienentää riskejä ja varmistaa, että ohjelmiston toiminnallinen ja ei-toiminnallinen käyttäytyminen vastaavat määritettyä komponenttien osalta (ISTQB 2018, 27). Yksikkötestauksen rinnalla tulisi käyttää koodin katselmointia. Yksikkötestien tuloksia verrataan komponenttien vaatimusmäärittelyyn. Yksikkötestauksessa virheen paikantaminen ja korjaus ovat tehokkaita, koska testattava yksikkö on selkeästi rajattu pieneen osaan ohjelmistoa. Erityisesti ketterässä kehityksessä ohjelmisto muuttuu jatkuvasti kehityksen tapahtuessa osissa ja testaukseen käytettävä aika on lyhyt. Tällöin automatisoidulla regressiotestauksella on tärkeä rooli varmistaa, että muutokset eivät ole rikkoneet jo olemassa olevaa toiminnallisuutta. Yksikkötestauksen tekee kehittäjä osana kehitystyötä. Testaus on lasilaatikkotestausta. (Watkins & Mills 2011, 48-54.)

## 10.3 Integraatiotestaus

Integraatiotestauksen tavoitteena on löytää virheitä ja estää niiden pääsy seuraavalle tasolle. Vaihe lisätä luottamusta rajapintojen laatuun ja pienentää riskejä laatuun liittyviä riskejä (ISTQB 2018, 27). Integraatiotestaus keskittyy komponenttien väliseen vuorovaikutukseen. Tässä kohtaa ohjelmiston eri osia aletaan sovittaa yhteen. Integraatiotestauksessa pyritään selvittämään, toimivatko ohjelmiston komponentit toistensa kanssa moitteettomasti, sujuuko tiedonsiirto virheettömästi, yhdistävätkö rajapinnat komponentit oikein ja onko toiminta vakaata. Vaikka yksittäiset komponentit toimisivatkin oikein, niin komponenttien yhteistoiminnassa saattaa ilmetä vikoja. Integraatiotestaus on yleensä kehittäjien vastuulla. Testaus voidaan tehdä mustalaatikko- tai lasilaatikkotestauksena. Mallit eroavat siinä, että lasilaatikkotestauksessa on näkyvyys koodiin. (Watkins & Mills 2011, 54-61.)

## 10.4 Järjestelmätestaus

Järjestelmätestauksessa testataan kokonaista järjestelmää. Testauksessa varmistetaan että, järjestelmä täyttää sille asetetut vaatimukset mukaan lukien ei-toiminnalliset vaatimukset. Järjestelmätestauksessa tutkitaan usein kokonaisia toimintoja testaten niitä päästä päähän.

Testiympäristön tulisi olla tuotannonkaltainen ja testidata vastaa tuotannossa olevaa dataa. Usein järjestelmätestaus tehdään kuitenkin tuotannosta eroavalla ympäristöllä johon tuen käytännön rajoitteista, jolloin testauksen tulosten luotettavuus vähenee. Järjestelmätestauksen on vastuussa koko kehitystiimi, mutta yleensä tämän vaiheen testaavat testaajat. Testaus on mustalaatikkotestausta. (Watkins & Mills 2011, 61-69.)

#### 10.5 Järjestelmien välinen integraatiotestaus

Järjestelmien välisessä integraatiotestauksessa keskitytään järjestelmien välisiin vuorovaikutuksiin ja rajapintoihin. Testauksessa otetaan huomioon myös vuorovaikutus ulkoisten organisaatioiden kanssa sekä niiden tarjoamat rajapinnat. Testauksen painopiste on järjestelmien yhteensopivuudessa ja toiminnan sujuvuudessa. Testauksessa hyödynnetään päästä päähän testausta. Testauksessa kannattaa käyttää negatiivista testausta varmistamaan järjestelmien toimivuus odottamattomien ongelmien varalta. Testaus on yleensä testaajien vastuulla. Testaus on mustalaatikkotestausta. (Watkins & Mills 2011, 69-75.)

#### 10.6 Hyväksymistestaus

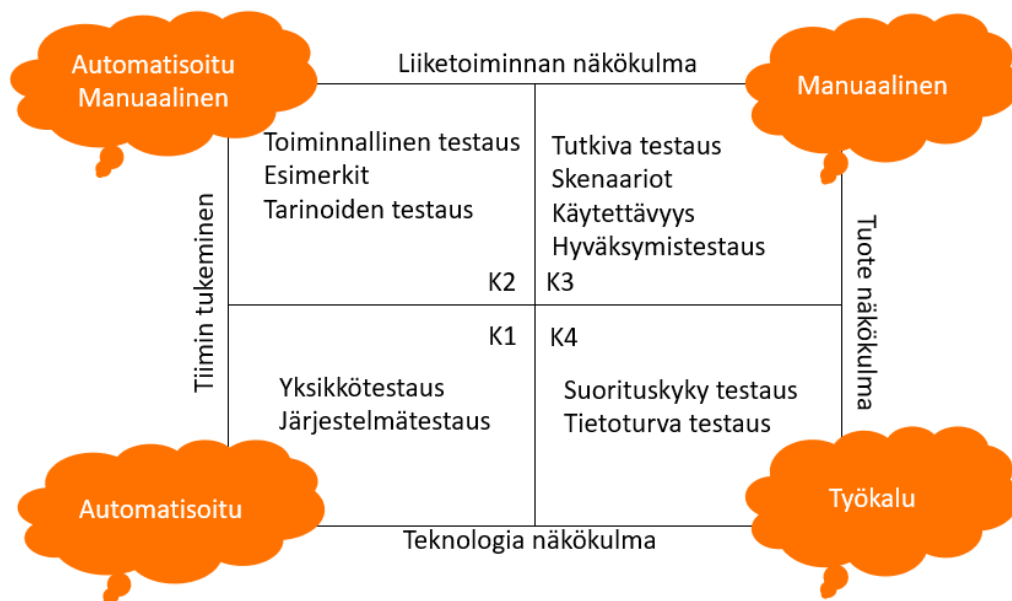
Hyväksymistestauksessa ei tulisi löytyä paljon vikoja. Viat pitäisi löytyä alemmilla testustasoilla ja tavoiteltavaa olisi, että viat on korjattu ja uudelleen testattu ennen hyväksymistestausta. Hyväksymistestaus tuottaa tietoa, jonka avulla arvioidaan, onko järjestelmä valmis julkaistavaksi ja soveltuuko se loppukäyttäjien käytettäväksi. Hyväksymistestauksen testitapaukset vastaavat loppukäyttäjien näkemystä ohjelmiston toiminnasta ja liiketoiminnalle tuotettavasta arvosta. Hyväksymistestaukseen tulisi suorittaa loppukäyttäjän edustaja. On tärkeää, että testaajalla on ymmärrys liiketoiminnasta. Hyväksymistestaus tulisi suorittaa tuotannon kaltaisessa testiympäristössä. Testaus on mustalaatikkotestausta. (Watkins ja Mills, 2011, 75-83). Hyväksymistestaus voi sisältää sääntelyihin tai lakeihin liittyvien vaatimusten tai käytössä olevien standardien täyttymisen varmistavia testejä. (ISTQB 2018, 32).



# 11 KETTERÄSSÄ TESTAUKSESSA KÄYTETTÄVÄT MENETELMÄT

## 11.1 Testauksen kvartaalit

Testauksen kvartaalit kuvaavat testiautomaation osuutta ketterässä testauksessa ja testivaiheiden jakautumista näkökulmien mukaan. Kuvio 13 avaa testivaiheiden jakautumisen.



Kuvio 13. Testauksen kvartaalit (mukaillen Gregory & Crispin 2015, 122)

### Ensimmäinen neljännes - Teknologia- ja tiimiä tukeva

Yksikkö- ja integraatiotestit, jotka ovat teknisesti suuntautuneita ja tiimiä tukevia. Menetelmäksi suositellaan TDD tekniikkaa, jossa testit kirjoitetaan ennen koodin luontia. Testit tulisi automatisoida ja näiden määrä pitäisi olla suuri, jolloin kehittäjät voivat luottavaisin mielin kehittää uutta ja tehdä muutoksia koodiin. Testiautomaatio antaa nopean palautteen kehittäjälle muutosten onnistumisesta. (Gregory & Crispin 2015, 122-123). Testien tulisi varmistaa, että yksittäisen toiminnallisuuden sekä komponentin toiminta on

vaatimusten mukainen ja yksittäiset toiminnallisuudet sekä komponentit voidaan integroida kokonaiseen järjestelmään. Määrällisesti näitä testejä tulisi olla eniten.

### **Toinen neljännes - Liiketoimintalähtöinen ja tiimiä tukeva**

Testit tukevat edelleen tiimiä, mutta korkeammalla tasolla. Testit auttavat varmistamaan, että kehitys tekee liiketoiminnan vaatimusten mukaista ohjelmistoa. Testit keskittyvät testaamaan haluttua toiminnallisuutta liiketoiminnan näkökulmasta. Testien lähteenä voivat toimia liiketoiminnan tekemät esimerkit ohjelmiston käyttötarkoituksesta. Testeistä olisi hyvä saada suurin osa automatisoiduksi nopean palautteen aikaan saamiseksi ja tukemaan regressiotestausta. (Gregory & Crispin 2015, 123-124.)

### **Kolmas neljännes - Liiketoimintalähtöinen ja tuote näkökulma**

Testit ovat liiketoimintalähtöisiä. Testit ovat manuaalisia ja tasoltaan hyväksymistestejä. Testaus menetelmäksi suositellaan tutkivaa testausta. Tämän vaiheen tarkoitus on selvittää vastaako ohjelmisto käyttäjien tarpeeseen. Testeihin on hyvä ottaa mukaan myös negatiivisia testejä ja pyrkiä miettimään mitä erinäisempiä tapoja käyttää ohjelmistoa. Tässä vaiheessa voidaan ohjelmalle tehdä käytettävyydestä. (Gregory & Crispin 2015, 125.)

### **Neljäs neljännes - Teknologialähtöinen ja tuote näkökulma**

Testit ovat teknologialähtöisiä ei-toiminnallisia testejä. Testien suorittamiseen tarvitaan usein tehtävään suunniteltuja työkaluja. Testien suunnittelu ja suoritus vaatii testausasiantuntijalta perehtyneisyyttä tiettyyn testityyppiin. Testausta helpottaa hyvin määritellyt ei-toiminnalliset vaatimukset. (Gregory & Crispin 2015, 126.)

## 11.2 Riskilähtöinen testaus

Tuoteomistajan vastuulla on jakaa tietoa riskeistä ja toiminnallisuuden arvosta kehitystiimille. Näiden tietojen perusteella kehitystiimi yhdessä tuoteomistajan kanssa tekee riskianalyysin kriittisiksi luokitelluille toiminnallisuuksille. Riskeihin tulee huomioida myös toteutuksen kompleksisuus. Hyvä yhteistyö tuoteomistajan ja kehitystiimin välillä helpottaa riskialttiiden toiminnallisuuksien tunnistamista ja analyysin tekoa. Riskit tulee ottaa huomioon testauksen suunnittelussa ja testauksessa. Riskilähtöisessä testauksessa testien priorisointi on tärkeää. Riskien perusteella päätetään mitä testataan, kuinka paljon ja missä järjestyksessä. Järjestykseen vaikuttaa toiminnallisuuksien valmistuminen sprintin tai iteraation aikana. Testausmenetelmä varmistaa sen, että toiminnot, jotka ovat liiketoimintakriittisiä tai aiheuttavat suurimmat vaarat liiketoiminnalle suoritetaan ensin. (Baumgartner ym. 2021, 126-129; Black 2014, 84-85.)

Riski on negatiivinen tai ei haluttu tapahtuma, joka voi tapahtua. Testauksen riskit voidaan jakaa kahteen luokkaan, tuotteen riskit ja projektin riskit. Tuotteen riskit vaikuttavat laatuun ja projektin riskit projektin onnistumiseen. Riskit käsitellään samoin vaikutuksen ja todennäköisyyden perusteella. (Black 2014, 84-85).

Tuotteen riskejä voivat olla virhe toiminnallisuudessa, vaatimusten vastainen toiminta tai toiminnallisuuden puuttuminen. Puutteet tiedon eheydessä, suorituskyyvyssä, turvallisuudessa, luotettavuudessa tai käytettävyydessä. Tuoteriskejä voidaan vähentää muutoin kuin testauksen keinoin. Ymmärrettävät ja hyvä laatuiset vaatimukset vähentävät tuoteriskejä merkittävästi. (Hambling 2015, 143–144.)

Projektiin liittyviä riskejä voivat olla esimerkiksi tilanne, jossa testiympäristöt eivät valmistu ajoissa, tarvittavat henkilöt eivät vapaudu projektiin, sairastapaukset, testaajien ammattitaito, kehitystä tai testausta ei ole tehty sovittujen käytäntöjen mukaisesti. Vaatimusten, koodin ja testauksen huono laatu. Projektin riskeihin voidaan lukea myös ongelmat, jotka liittyvät asiakas toimittaja suhteisiin tai kolmansiin osapuoliin. (Hambling 2015, 142.)

Riskien tunnistamiseen voidaan käyttää seuraavia tekniikoita tai yhdistelmiä niistä, asiantuntijahaastattelut, riippumattomat arvioinnit, riskityöpajat, aivoriihet, tarkistuslistat tai aikaisempien kokemusten hyödyntäminen. Riskien tunnistamisen jälkeen riskit arvioidaan todennäköisyyden ja vaikutuksen mukaan. Riskit voidaan luokitella eri riskityypeiksi esimerkiksi suorituskyyvyn, luotettavuuden, toiminnallisuuden mukaan. Organisaatiot

muodostavat heille sopivan luokittelun. Riskeille määritetään todennäköisyys ja vakavuus, joiden perusteella lasketaan riski-indeksi kertomalla vakavuus todennäköisyydellä. Riski-indeksin ylittäessä organisaation määrittelemän riskitason riskiä tulee pienentää. (Black 2014, 87; ISTQB (2015a), 24-25).

Riskejä voidaan pienentää jo ennen testauksen alkua lisäämällä kehitysprosessiin katselmoinnit vaatimuksille ja koodille. Projektin aikana kehitystiimin ja tuoteomistajan pitäisi arvioida riskejä uudestaan tietojen tarkentuessa tai muuttuessa. Riski-indeksiä tulee laskea esimerkiksi, jos testauksen aikana tietyltä ohjelmiston osa-alueelta löytyy huomattavasti odotettua enemmän vikoja. Tämä johtaa kyseisen osa-alueen testausmäärän kasvattamiseen. Riskien hallintaa tulisi tapahtua projektin kaikilla tasoilla. (ISTQB (2015a), 25-26.)

Riskipohjaisia testaustekniikoita on paljon. Jotkut näistä tekniikoista ovat hyvin epämuodollisia. Esimerkiksi, testaajan analysoidessa tuoteriskejä tutkivaa testausta suunnitella tai sen aikana. Tämä voi auttaa ohjaamaan testausta, mutta tällaiset lähestymistavat ovat subjektiivisia ja riippuvaisia yksittäisen testaajan taidoista, kokemuksista ja mieltymyksistä. Kevyitä riskilähtöisiä lähestymistapoja käytettäessä pyritään saavuttamaan testauksen hyödyt, mutta minimoimaan aikaa ja kustannuksia. Pragmatic Risk Analysis and Management (PRAM), Systematic Software Testing (SST) ja Product Risk Management (PRisMa) ovat esimerkkejä kevyistä tekniikoista. Yhteistä näille tekniikoille on, että ne ovat kehittyneet ajan myötä ja paras hyöty niistä saadaan, jos ne otetaan käyttöön heti projektin alussa. Riskien arviointi ja tunnistaminen perustuu liiketoimintaa ja teknistä näkökulmaa ymmärtävästä henkilöistä, jotka tekevät arvioinnin yhdessä. Formaaleja ja raskaita tekniikoita ovat vaara-analyysi, joka yrittää tunnistaa jokaisen riskin taustalla olevan vaaratilanteen. Vikavaikutusanalyysi Failure Mode and Effect Analysis (FMEA) ja sen muunnelmat keskittyvät tunnistamaan tuoteriskit ja niiden mahdolliset syyt sekä todennäköiset vaikutukset. Riskeille annetaan vakavuus-, kiireellisyys- ja löytöarvot. Quality Function Deployment (QFC) on riskien hallintamenetelmä, jossa keskitytään väärästä tai riittämättömästä asiakkaan tai loppukäyttäjän vaatimusten ymmärtämisestä syntyviin riskeihin. Vikapuuanalyysi Fault Tree Analysis (FTA) keskittyy erilaisiin joko testauksessa tai tuotannossa todettuihin häiriöihin tai mahdollisiin häiriöihin. Analyysi lähtee liikkeelle ongelmista, jotka voivat aiheuttaa häiriön, etenee virheisiin tai vikoihin analyysi jatkuu eteenpäin, kunnes vikojen juurisyyt on tunnistettu. (ISTQB (2015a), 27-28.)

### 11.3 TDD (Test Driven Development)

Menetelmää käyttävät kehittäjät ja sitä suositellaan käytettäväksi automatisoiduissa yksikkö- ja integraatiotestauksessa. Testit kirjoitetaan ennen koodia. Laaja ja varhainen testaus on olennaista ketterässä testauksessa. Testivetoinen kehitys sopii toimintata- vaksi ketterässä kehityksessä. Testejä tulee käyttää uudelleen varmistamaan, että van- hat toiminnot toimivat oikein uusien toiminnallisuuksien lisäyksen tai muutosten jälkeen. (Baumgartner ym. 2021, 48.)

Testauksen organisointi ketterässä kehityksessä vaatii sitoutumista kehittäjiltä testauk- seen. Tiimi voi aluksi olla motivoitunut testivetoiseen kehitykseen, mutta hylkää sen tiuk- kien aikataulujen vuoksi. Jos koodia on kirjoitettu paljon ilman testitapauksia tekninen velka kasvaa liian suureksi ja tiukassa aikataulussa testien korjaamiseen ja kirjoittami- seen kuluu aikaa, jos tätä ei tehdä hyöty testeistä menetetään. Yksikkötestien automati- soinnin tulisi olla ehdotonta ketterässä kehityksessä. TDD:n käyttö on perusteltua ja maksaa itsensä takaisin kohentuneen laadun seurauksena. (Baumgartner ym. 2021, 49- 50.)

TDD:n käyttö johtaa joustavaan ja helpommin ylläpidettävään koodiin. Se vähentää vir- heitä ja lisää testien koodikattavuutta. Testeillä pystytään varmistamaan, että koodi toimii muutosten jälkeen, joka helpottaa refaktorointia. TDD:n käyttö nopeuttaa kehitystä sääs- tämällä aikaa manuaaliselta regressiotestaukselta. (Callawey & Hunt 2018, 8 -15.) Testit toimivat myös dokumentaationa ja helpottavat koodikatselmointia sekä kehitystä. (To- daro 2019, 144-145). Kuvio 14 kuvaa TDD tekniikan työtavan.



Kuvio 14. TDD käytäntö (mukaillen Cloudt 2021, 168)

Baumgartner ym. esittelevät kirjassaan TDD:n prosessin:

### 1. Testin kirjoittaminen

Luodaan testitapaus, joka testaa uuden toteutettavan toiminnallisuuden. Testi suoritetaan, mutta uuden testin tulee ajautua virheeseen, koska toteutusta ei ole vielä tehty.

### 2. Toiminnallisuuden toteutus ja testin suoritus

Uusi toiminnallisuus toteutetaan ja testit suoritetaan uudelleen yhdessä aikaisemmin tehtyjen testien kanssa. Tällä kertaa testien tuli ajautua onnistuneesti lävitse.

### 3. Refaktorointi

Refaktoroinnin tarkoituksena on yksinkertaistaa koodia ilman, että ohjelman toiminnallisuus muuttuu. Testejä saatetaan joutua muuttamaan refaktoroinnin yhteydessä. Tällä varmistetaan, että koodi pysyy ylläpidettävänä ja teknistä velkaa ei synny.

#### 11.4 BDD (Behaviour Driven Development)

BDD lähtee liikkeelle liiketoiminnan tavoitteista ja on TDD:n laajennus (Irshad ym. 2022, 1). Toiminnallisuudet esitetään reaali maailman esimerkeillä ja näistä syntyy kehitystä ja testausta tukeva dokumentaatio. Testaus perustuu käyttäjän käyttäytymisen ymmärtämiseen, arvoa tuottaviin toiminnallisuuksiin ja liiketoiminta prosesseihin sekä hyväksymiskriteereihin. (Smart 2015, 15–18.)

Testi tulisi tehdä yhteistyössä tuoteomistajan ja kehitystiimin kesken. Testit tulisi kirjoittaa ymmärrettävällä kielellä. Testit kirjoittamisessa hyödynnetään Gherkin-syntaksia. Näin kirjoitetut testit ovat muodossa, joka tukee testiautomaation tekemistä. (Smart 2015, 19.)

Gherkin-syntaksi (Matsinopoulos 2020, 231-237):

- Feature: Toiminnallisuus
- Scenario: Kuvaus
- Examples: Reaalielämän esimerkki
- Given: Konteksti
- When: Tapahtumat
- Then: Odotettu tulos
- And: Lisäys
- But: Rajoitus

BDD:n käyttö vähentää kustannuksia keskittymällä arvoa liiketoiminnalle tuottavaan toiminnallisuuteen, vähentämällä virheitä ja lisäämällä kehitystiimin ymmärrystä liiketoiminnan vaatimuksista (Smart 2015, 28 - 29).

#### 11.5 ATDD (Acceptance Test Driven Development)

Testit perustuvat hyväksymiskriteereihin ja ohjelmiston toiminnallisuuden varmistamiseen. Hyväksymistestit ovat liiketoimintalähtöisiä ja linkitetty vaatimuksiin. Testit tulisi suunnitella niin, että yksi tai useampi hyväksymiskriteeri käyttäjätarinasta tulee testatuksi kerrallaan. Testit tulisi suunnitella yhteistyössä tuoteomistajan ja kehitystiimin kesken. Vaatimusten rinnalla olisi hyvä käyttää reaalielämän esimerkkejä, lisäämään ymmärrystä toiminnallisuudesta. Vaatimuksella täytyisi olla mitattavat tai muuten yksiselitteiset hyväksymiskriteerit kirjattuna, jotta testin odotettu tulos on selvä. Vaatimukset ja testit tulisi

kirjoittaa ymmärrettävällä kielellä, samoin kuin edellisessä tekniikassa. Testattavat vaatimukset auttavat hyväksymistestauksen suunnittelussa ja toteutuksessa. Mahdollisimman suuri määrä testeistä olisi hyvä saada automatisoiduksi, jolloin testejä voidaan hyödyntää regressiotestauksessa. ATDD-tekniikan on koettu parantavan tehokkuutta ja nopeuttavan palautteen saamista edellyttäen, että testit suunnitellaan yhdessä tuoteomistajan tai liiketoiminnan kanssa. (Black ym. 2017, 81 - 82).

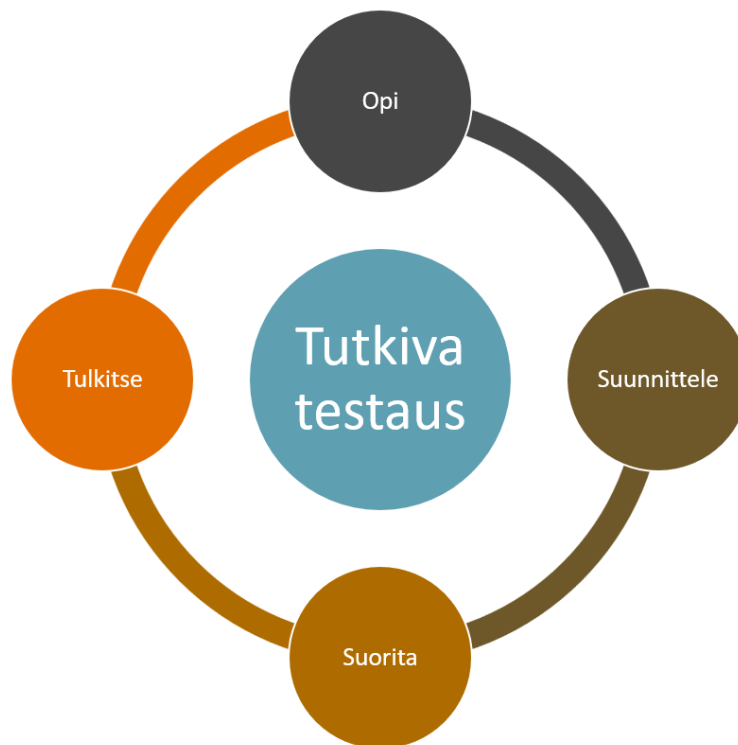
Hyväksymistestilähtöisessä kehityksessä koko kehitystiimi analysoi käyttäjätarinat ennen sprintin suunnittelukokousta sekä perehtyy tuoteomistajan ennalta määrittelemiin hyväksymiskriteereihin. Jos hyväksymiskriteerit tai käyttäjätarinat eivät ole yksiselitteisiä tuoteomistaja tarkoittaa vaatimuksen tai jatkaa toiminnallisuuden määrittelyä, jolloin toiminnallisuus ei ole vielä valmis toteutettavaksi. (Baumgartner ym. 2021, 133.)

Kriteereistä muodostetaan hyväksymistestitapaukset, myös vaihtoehtoisista toimintatavoista sekä virheiden käsittelystä tulee tehdä testitapaukset. Testit olisi hyvä olla päästä päähän testejä. Testien tulisi testata myös ei-toiminnallisia vaatimuksia. (Pugh 2011, 23-68.)

## 11.6 Tutkiva testaus

Tutkiva testaus on lähestymistapa, jota kuvataan usein samanaikaiseksi oppimiseksi, testin suunnitteluksi ja suorittamiseksi. Testaus voidaan tehdä yksin tai useamman testaajan ryhmässä. (Kinsbruner 2018, 117-121.) Kuvio 15 kuvaa tutkivan testauksen vaiheita.





Kuvio 15. Tutkiva testaus (mukaillen Kinsbruner 2018, 118)

Tutkiva testaus on tehokas manuaalinen testaus menetelmä. Menetelmässä testaaajan tietämys ja kokemus korostuvat. Testaus on suunniteltua ja aikarajoitteista. Erona tarkasti käsikirjoitetulle testaukselle testisession aikana voidaan poiketa suunnitelmasta ja tämä itseasiassa on suositeltavaa. Tutkivan testauksen tarkoituksena on löytää virheitä, joten testeihin kannattaa ottaa mukaan myös negatiivisia testitapauksia. Tehokas, luova ja ammattitaitoinen testaaaja on hyvä arvaamaan, missä virheet ovat. (Baumgartner ym. 2021, 129 -130.)

Tutkivan testauksen vaiheet:

- Ennen aloitusta testaaaja päättää mitä testaa ja tekee testisessiolle suunnitelma ja määrittää testaukseen käytettävän ajan.
- Testauksen edetessä suunnitelmasta voidaan poiketa. Tällöin testaaajan tulee kirjata uudet testitapaukset testiohjeeseen. Testiohjeeseen kirjataan testauksen aikana löydettyjen virheiden tunnus.

- Testauksen lopuksi testaaja arvioi testauksen kattavuuden ja riittävyyden. Testausohjeeseen kirjataan testaajan tekemät huomiot testauksen aikana.

Tutkivassa testauksessa testien suunnittelu ja suoritus tapahtuvat osin samaan aikaan ja niitä ohjaa etukäteen tehty testausohje. Testausohje kuvaa testausession aikana testattavat asiat, olematta kuitenkaan liian yksityiskohtainen. Testiohje suunnitellaan etukäteen ja se sisältää, linkin vaatimukseen, testauksen tavoitteen, prioriteetin, alustavat testitapaukset, aikarajoitteen ja mahdolliset tarvittavat järjestelyt ennen kuin testaus voi alkaa ja testiaineiston. (ISEB (2015b), 38–39; Hendrickson 2013, 11-24.) Kuvio 16 kuvaa yhden esimerkin tutkivan testauksen testiohjeesta.

Tavoite:  
Vaatus:  
Aika:

ID	Testitapaus	Tulos	Virhe ID

Huomiot testauksen aikana:

Kuvio 16. Esimerkki tutkivan testauksen testiohjeesta

## 12 TESTIAUTOMAATIO

Testiautomaation käytöstä hyötyvät kaikki ohjelmistokehitys tavat ei pelkästään ketterät. Testiautomaatio tukee melkein kaikkia testaustyypppejä ja -vaiheita, mukaan lukien toiminnallinen testaus ja ei-toiminnallinen testaus. (Boby 2021,31-33.)

Testausautomaation tärkeys korostuu ketterässä kehittämisessä nopean kehityssyklin ja muutosten vuoksi. Parhaimmillaan testiautomaatio saadaan lisättyä jatkuvaan integraatioon, jolloin testaus tehostuu entisestään. Testiautomaatio osaamista tulisi olla sekä testaajilla että kehittäjillä ja testiautomaatiota tulisi hyödyntää eri testitasoilla. On tärkeää saada testiautomaatio heti alusta asti käyttöön. Projektin alusta asti käytössä ollut testiautomaatio laajentaa testikattavuutta.

Testausautomaatio soveltuu tiimiä tukeviin testeihin, joiden tarkoituksena on varmistaa ohjelmiston toimivuus. Testiautomaatiota on hyvä hyödyntää aina kun se on mahdollista. Testit helpottavat kehitystiimin muutosten tekemistä toiminnallisuuksiin sekä uutta kehitystä. Testiautomaatio alentaa kustannuksia pitkällä aikavälillä. (Boby 2021, 31; Saleem ym. 2014).

Testiautomaatio tekee testin suorittamisesta nopeaa ja luotettavaa. Se eliminoi manuaalisessa testauksessa mahdollisesti syntyneet inhimilliset virheet ja on johdonmukainen kaikille syötteille. Testausautomaatio voi toistaa saman testin tai saman testin eri datalla pienemmillä kustannuksilla ja samalla säästää aikaa. Uudelleenkäytettävyys parantaa testauksen tehokkuutta. Testiautomaation kattavuutta voidaan helposti laajentaa eri alustoihin, ympäristöihin, käyttöjärjestelmiin, selaimiin ja mobiililaitteisiin. Testiautomaatio työkalut tuottavat yleensä yksityiskohtaisen testilokin, mikä nopeuttaa ja helpottaa suoritettujen testien analysointia. Testiautomaation suoritus voidaan ajastaa, jolloin testien ajo ei ole kiinni työtunneista. Testiautomaatiolla voidaan vähentää riippuvuutta ajasta, paikasta ja resursseista. Regressiotestauksessa testiautomaation käyttö säästää kehitystiimin aikaa ja aikaansaa muutosten jälkeisen luottamuksen ohjelman toimivuuteen. (Boby 2021, 32 -33.)

Testiautomaatio ja manuaalinen testaus eivät ole toisiaan pois sulkevia vaan niitä tulisi käyttää rinnakkain. Manuaalinen testaus sopii hyvin tutkivaan testaukseen, koska testaus voi muuttaa suuntaa testauksen aikana. Käyttökokemuksen ja käytettävyyden testausta ei voi testata testiautomaatiolla. Lisäksi ihminen voi havaita vikoja, jotka

automaatio jättää huomiotta. Staattinen testaus kuten kooditarkistukset suoritetaan yleensä manuaalisesti. Automaatiotyökaluista aiheutuu kustannuksia, mukaan lukien infrastruktuuri, ylläpito, kehitys ja henkilöstöresurssit, siitäkin huolimatta, että automaatiotyökalu olisi avoimen lähdekoodin ohjelmisto. Lisäksi kustannuksia aiheuttaa työkalujen päivitykset, mahdolliset muutokset tai konfiguraatiot. Lyhyt aikaisiin projekteihin testiautomaatio työkaluun investoiminen ei ole järkevää. Manuaalinen testaus on aikaa vievää verrattuna testiautomaation käyttämään aikaan. Testaajia tarvitaan määrällisesti enemmän, jos regressiotestausta ei voida toteuttaa testiautomaation avulla. (Boby 2021, 32-34.)

## 12.1 Jatkuva integraatio

Testiautomaation käyttö mahdollistaa jatkuvan integraation rakentamisen.

Jatkuva integraatio kokoaa yhteen ohjelmistoon tehdyt muutokset ja uudet toiminnallisuudet integroimalla toiminnallisuudet säännöllisesti jo kehitettyyn ohjelmistoon. Kehittäminen ja testaus on yhdistetty automatisoiduksi toistettavaksi prosessiksi, jolloin viat koodissa voidaan havaita nopeammin. Jatkuvan integrointi koostuu seuraavista automatisoiduista tehtävistä staattinen koodianalysointi, koodin kääntäminen, automaattisten testien suorittaminen ja testitulosten raportointi. Jatkuva integraatio antaa tiimille mahdollisuuden suorittaa automatisoidut testit säännöllisesti päivittäin. Prosessiin voidaan lisätä vaihe, joka lähettää muutoksen tehneen koodin kirjoittajalle palauteen koodin laadusta. Testitulokset tulisi olla näkyvissä koko kehitystiimille ja kiinnostuneille sidosryhmille. Automaattisten regressiotestien hyvä kattavuus lisää luottoa ohjelman toimivuuteen ja antaa mahdollisuuden testaajien käyttää aikaa tutkivaan testaukseen. Jatkuva integraatio tähtää ohjelmiston laadun parantamiseen, antaa tiimille säännöllistä palautetta koodin laadusta ja vähentää riskiä refaktoroinnin yhteydessä. Jatkuvaan integraatioon liittyy haasteita, kuten työkalut pitää ottaa käyttöön ja niitä pitää ylläpitää, jatkuvan integraation prosessi pitää määrittää ja ottaa käyttöön, testiautomaatio vaatii resursseja ja testikattavuuden tulee olla laaja, jotta saavutetaan mahdollisimman suuret hyödyt. (ISTQB (2015b), 14-15.)

## 13 KETTERÄN TESTAUKSEN METRIIKAT

### 13.1 Projektin metriikat

Nopeus on yksi ketterän kehityksen tavoitteista, joten seuraavia mittareita voidaan käyttää analysoimaan ja määrittämään nopeuteen kohdistuvia parannuskohteita. Vikoja analysoimalla organisaatio voi havaita, milloin ja miksi vikoja syntyy. Vähentämällä virheitä säästetään aikaa virheiden korjaamisessa ja uudelleen testaamisessa. Mittaus kannattaa aloittaa pienellä määrällä mittareita ja sitoa mittarit organisaation tavoitteisiin. Mittareilla kannattaa keskittyä etsimään parannuskohteita. Jos mittarin tuloksista ei ole hyötyä se kannattaa vaihtaa toiseen. Nopeuden saavuttaminen ei johda laatuun, mutta laadun saavuttaminen johtaa yleensä suurempaan nopeuteen. (Lew 2016.)

Esimerkkejä seurattavista metriikoista (Lew, 2016):

1. Avoinna olevat viat prioriteetin mukaan.
2. Viat ongelmatyyppin mukaan prosentteina. Tällä pyritään varmistamaan, että samantyyppisiä vikoja ei esiinny kerrasta toiseen. Jos näin tapahtuu, prosessia pitää parantaa. Testausta voidaan myös keskittää löytämään tietyn ongelmatyyppin vikoja. Ongelmatyyppien luokittelu on organisaation päätettävissä.
3. Vian keskimääräinen korjausaika. Vikoja voidaan seurata tyypeittäin, joiden korjaus kestää kauan. Vikojen korjaukseen ja uudelleen testaukseen kulunutta aikaa voidaan seurata, jolloin tiedetään virheiden korjaamiseen kulunut kokonaisaika.
4. Asiakas- ja teknisen tuen pyyntöjen määrä.
5. Virheiden tila prosenttina. Avoimille virheille voidaan asettaa enimmäismäärä tai prosenttiosuusmäärä vikatyypeittäin tai vakavuusasteittain.
6. Uudelleen avattujen virheiden määrä.
7. Ennen tuotantoon vientiä löydetty viat verrattuna tuotannossa löydettyihin vikoihin. Analysoimalla tuotannosta löytyneitä vikoja voidaan testausta keskittää estämään nämä tulevaisuudessa.
8. Vian poiston tehokkuus.
9. Vikatyyppit testaajaa kohden. Tarkoituksena on tunnistaa kehityskohteita kouluttamalla testaajia eri osa-alueille. Tämä antaa mahdollisuuden tehostaa toimintaa käyttämällä testaajia heidän osaamillaan osa-alueella.
10. Hylätyt viat.

11. Vaatimusten ja suunnittelun virheet, jotka selviävät katselmoinneissa.
12. Tuotettu arvo.
13. Suunnittelemattoman työn määrä.

Nicolette jakaa metriikat kirjassaan edistymisen seuraamiseen ja kehittymiseen. Seuratavuutta edistäviä metriikoita ovat tavoitteiden saavuttaminen, tuotettu arvo ja liiketoiminnan savuttama arvo, budjetin käyttöaste, testattujen toiminnallisuuksien määrä. Kehittymisen metriikoita ovat nopeus, virtaus ja virtauksen tehokkuus, kumulatiivinen virtaus, suorituskyyky ja ”Burn down tai up” – kaaviot. (Nicolette 2015, 9- 07.)

SAFe tuo mukanaan itsearviointit, joilla tavoitellaan liiketoiminnan ketteryyttä ja löydetään kehityskohteita. Taulukosta 1 löytyvät SAFessa käytettävät arviointit. Liiketoiminnan ketteryydellä tavoitellaan korkeaa laatua ja tuottavuutta, kehitystiimin sitoutumista ja nopeampaa toiminnallisuuksien markkinoille saamista. (Scaled Agile.) Tiimien ennustettavuutta mitataan PI:n jälkeen laskemalla arvon tuotto, niistä toiminnallisuuksista mihin tiimi on sitoutunut, tavoiteltava prosenttiosuus on 80 – 100 %. PI:n ja iteraatioiden etenemistä voidaan seurata kuviossa 17 esitettyjen metriikoiden avulla. (Leffingwell ym. 2018, 435 – 455.)

Toiminnallisuus	PI 1	PI 2	PI 3	
Nopeus				
Laskettu ennustettavuus				
Suunnitellut toiminnallisuudet	Toiminnallisuus	Iteraatio 1	Iteraatio 2	Iteraatio 3
Hyväksytyt toiminnallisuudet	Suunniteltu nopeus			
Suunnitellut mahdollistajat	Mitattu nopeus			
Hyväksytyt mahdollistajat	Suunnitellut käyttäjätarinat			
Suunnitellut käyttäjätarinat	Hyväksytyt käyttäjätarinat			
Hyväksytyt käyttäjätarinat	Hyväksytyjen käyttäjätarinoiden osuus %			
Laatu	Laatu			
Yksikkötestien kattavuus %	Yksikkötestien kattavuus %			
Virheet	Virheet			
Testien määrä	Uudet testitapaukset			
Testiautomaation kattavuus %	Uudet automatisoidut testitapaukset			
Ei-toiminnallisten testien määrä	Testien määrä			
	Testiautomaation kattavuus %			
	Refaktoroitien määrä			

Kuvio 17. SAFen PI ja iteraatio metriikat (mukaillen Leffingwell ym. 2018, 446 - 453)

Taulukko 1. SAFen arvioinnit (Scaled Agile)

Arvioinnit	
Organisaation ketteryys	Ihmiset ja ketterät tiimit
	Lean liiketoiminta
	Strategian ketteryys
Salkunhallinta	Strategia ja investointirahoitus
	Ketterä salkunhallinta
	Lean hallinto
Yritysratkaisujen toimitus	Lean järjestelmät ja ratkaisusuunnittelu
	Junien ja tavarantoimittajien koordinointi
	Jatkuva kehittyminen
Ketterä tuotetoimitus	Asiakaslähtöisyys ja muotoiluajattelu
	Kadenssin ja pyynnöstä julkaisemisen kehittäminen
	DevOps ja jatkuva julkaisuputki
Tiimi ja tekninen ketteryys	Ketterät joukkueet
	Sisäänrakennettu laatu
	Ketterä tiimi
Jatkuvan oppimisen kulttuuri	Oppiminen
	Kehittyminen
Ketterä johtaminen	Ajattelutapa ja periaatteet
	Esimerkillä johtaminen
	Muutoksen johtaminen

### 13.2 Testauksen metriikat

Testauksen tehokkuutta ja onnistumista kannattaa seurata pidemmällä aikavälillä. Testauksen tehokkuutta seuraavat metriikat tulisi kirjata testauksen politiikka dokumenttiin. Testausta voidaan seurata esimerkiksi seuraavilla metriikoilla (Black ym. 2017, 9-11):

- Tehokkuus virheiden löytämisessä
  - Löydetyt virheet testauksen aikana / testauksen aikana löydetyt virheet + tuotannosta löydetyt virheet (6 kk)
- Riskit
  - Testitapaukset riskeille/ riski analyysissä tunnistetut riskit
- Testauksen tuottama arvo
  - $((\text{keskimääräinen kustannus tuotannon virheistä} \times \text{virheet}) - \text{kustannus ennen tuotantoa löydetyistä virheistä}) / \text{testauksen hinta}$
- Regressiotestien automaatioaste
  - Automaattisten regressiotestien määrä/ regressiotestien määrä



## 14 KETTERÄN TESTAAJAN TARVITSEMAT TAIDOT

Burnstein erottelee testaajien taidot henkilökohtaisiin ja teknisiin taitoihin. Suunnitelmallisuus, kyky kiinnittää huomiota yksityiskohtiin, ongelman ratkaisutaidot, yhteistyökyky- ja vuorovaikutustaidot kirjallisesti ja suullisesti sekä halu jakaa osaamista ja tukea muita ovat henkilökohtaisia taitoja. Teknisiin taitoihin luetaan ymmärrys ohjelmistotekniikan käytännöistä ja metodeista, koodaus taidot testiautomaation toteuttamiseen, ymmärrys testauksen käytännöistä, strategiasta, metodeista ja tekniikoista, kyky suunnitella testausta ja testitapauksia useilla eri tasoilla, ymmärrys testaukseen ja laatuun liittyvistä prosesseista, verkojen toiminnasta, tietokannoista ja käyttöjärjestelmistä, kyky määrittellä, kerätä ja analysoida testauksen tuottamaa tietoa sekä ymmärrys laadun merkityksestä. (Burnstein 2003, 237 -240.)

### 14.1 Testaajan rooli ketterässä kehityksessä

Testaajan rooliin ketterässä tiimissä sisältyy tehtäviä, jotka perinteisessä testauksessa olisivat kuulunut testauspäälikölle. Ketterässä kehityksessä testaajat työskentelevät läheisessä yhteistyössä kehittäjien ja liiketoiminnan edustajien kanssa, jolloin ihmissuhdetaidot korostuvat. Testaajien tulisi olla laatutietoisia ja osata suhtautua kriittisesti kehitettävään ohjelmistoon. Testitulosten raportointi, seuranta ja arvio testauksen edistymistä sekä ohjelmiston laadusta, jää ketterässä testauksessa testaajien tehtäväksi. Testaajien tulee seurata testausstrategiaa ja ehdottaa siihen parannuksia. Testauskattavuuden arviointi ovat testaajien tehtäviä. Tiedon jakaminen testauksesta kehitystiimille ja osallistuminen kehitystiimin retrospektiiveihin, ehdottamalla testaukseen tai kehitykseen liittyviä parannuksia. Lisäksi testaajien tulisi tukea tuoteomistajaa testattavien vaatimusten kirjoittamisessa ja kehitystiimiä testitapausten kirjoittamisessa. Testaajien tulisi organisoida omaa työtänsä ja omata halu jatkuvan kehittymiseen. (ISTQB (2015b), 26-27.)

Itseohjautuvuutta voidaan tukea priorisoimalla työtä. Priorisoinnin tärkeys kasvaa, jos työ on nopeatempoista, sisältää keskeytyksiä ja vaatii asiantuntemusta. Ajankäytön suunnittelulla ja ymmärryksellä, koska työ on tehty riittävän hyvin, voidaan lisätä itseohjautuvuutta. Kiinnostuksesta osaamisen kehittämiseen, uteliaisuudesta ja halusta oppia uutta tunnistaa hyvän itsensä johtajan. Itseohjattavuuden periaatteisiin kuuluu kyky sanoa ei, jos työtä on liikaa tai tavoitteet ovat epärealistisia. (Sarvaspuro 2019, 168 – 170.)

## 15 TESTAUKSEN KEHITTÄMINEN

### 15.1 Organisaatiotasoinen kehittäminen

Organisaatiotasolla testausta voidaan kehittää useilla eri malleilla. Yleisimmät mallit ovat TPI NEXT (Test Process Improvement Next), TMMi (Testing Maturity Model Integration), CTP (Critical Testing Processes) ja STEP (Systematic Test and Evaluation Process). (Bath & van Veenendaal 2014, 216-220.)

Prosessien kehittäminen auttaa organisaatiota toimittamaan ohjelmistoja ajallaan ja tehostamaan toimintaansa. Prosessien parantamisella tavoitellaan jatkuvaa kehittymistä, avoimuuden ilmapiiriä, missä voidaan keskustella ongelmista. (O'Regan 2019, 219.)

Prosessien kehittäminen kannattaa, koska prosessin laatu vaikuttaa suuresti ohjelmiston laatuun. Ohjelmiston parantunut laatu helpottaa muun muassa ylläpitoa ja vähentää teknistä velkaa, jättäen enemmän aikaa ohjelmiston kehittämiseen. Alkuun tulisi määrittää prosessin nykyinen tila. Arvioinnin jälkeen TMMi ja TPI NEXT tarjoavat etenemispolut testausprosessin kehittämiseen. CTP ja STEP tarjoavat keinon määrittellä kehityskohdet, mutta jättävät valinnan ja kehitettävät toimenpiteet organisaatiolle. (ISTQB (2015a), 57 - 58).

#### TMMi

Malli muodostuu viidestä kypsyystasosta. Tasot pitävät sisällään määritellyt prosessialueet, joiden tavoitteet täytyy täyttyä 85-prosenttisesti seuraavalle tasolle siirryttäessä. (ISEB (2015a), 59) Taulukko 2 kuvaa mallin kypsyystasot.

Taulukko 2. TMMi tasot (Black 2014, 225-226)

<b>Optimoitu</b>	Testausprosessista saatuja tietoja hyödynnetään vikojen ehkäisemiseen ja prosessin parantamiseen.
<b>Mitattu</b>	Testausprosessia mitataan ja hallitaan organisaatiotasolla.
<b>Määritelty</b>	Testausprosessi on integroitu ohjelmistokehityksen elinkaareen. Katselmoinnit ovat käytössä ja testauksen edistymistä seurataan.
<b>Hallittu</b>	Testausprosessi on määritetty. Testauspolitiikka ja -strategia on määritetty. Testauksen perustekniikat ja menetelmät ovat käytössä.

<b>Lähtötaso</b>	Ei määriteltyä testausprosessia. Testit tehdään suunnittelemattomasti testauksen jälkeen.
------------------	---

## **TPI NEXT**

TPI NEXT määrittelee 16 osa-aluetta, sidosryhmien sitoutuminen, osallistumisen aste, testauksen strategia, testaus organisaatio, viestintä, raportointi, testausprosessin hallinta, suunnittelu ja työmäärän arviointi, metriikka, virheiden hallinta, testauksessa käytettävät ohjelmistot, testauksen menetelmät, testaajien osaaminen, testitapausten suunnittelu, testityökalujen hallinta ja testiympäristöjen hallinta. (Black 2014, 257.)

Mallissa on neljä kypsyystasoa lähtötaso, hallittu, tuottava ja optimoitu. Osa-alueiden arviointiin on määritelty tarkastuspisteet. Löydökset vedetään yhteen ja niistä muodostetaan kypsyysmatriisi. Kehitys tehdään organisaation tarpeiden ja kyvykkyyden mukaan. (ISEB (2015a), 59.)

## **CTP**

CTP:n perusajatuksena on jaotella testauksen osa-alueet 12 osaan kriittisyyden mukaan. Menetelmä on tilanneriippuvainen malli, joka sallii menetelmän muokkauksen organisaatiolle sopivaksi. Tällaisia muutoksia voivat olla yksittäisten haasteiden tunnistamisen, prosessin hyvien ominaisuuksien tunnistamisen sekä parannusten järjestyksestä päättämisen. Haastatteluiden lisäksi mallissa käytetään mittareita, joiden avulla organisaatio voi verrata suoriutumistaan teollisuudenalan keskiarvoihin. (ISEB (2015a), 60.)

## **STEP**

STEP perustuu elinkaari ajatteluun, joka alkaa vaatimusmäärittelystä ja jatkuu aina ohjelmiston elinkaaren loppuun asti. Testauksen lähestymistapa on testin toteutus ennen koodia. Menetelmä perustuu vaatimus pohjaiseen testausstrategiaan, löydetty viat yritetään löytää mahdollisimman varhaisessa vaiheessa ja löydetty viat analysoidaan. Menetelmässä testaajat ja kehittäjät työskentelevät yhdessä. (ISEB (2015a), 60.)

Testaukselle tulisi määrittää johdon tukema politiikka ja strategia. Testauspolitiikassa määritellään laatutaso, joka organisaatiossa halutaan saavuttaa ja strategia määrittelee pelisäännöt testauksen tekemiselle. (Kasurinen 2013, 86.) Ketterästä kehittämisestä

puuttuu testipäällikön rooli, joten mallien avulla kehittäminen voidaan antaa organisaation laatupäällikön tehtäväksi.

## 15.2 Projektitasoinen kehittäminen

Scrum:ssa kehittäminen tapahtuu retrospektiiveissä sprintin jälkeen (Rubin 2013, 381). SAFe:ssa keinoja on retrospektiivin lisäksi LACE (Lean-Agile Center of Excellence), jossa kehitetään organisaation toimintaa muun muassa määrittelemällä työohjeita junille sekä CoP (Communities of Practice) – yhteisöt. Yhteisöjen tarkoituksena on verkostoitua samaa työtä tekevien tai saman kiinnostuksen jakavien henkilöiden kesken. Testaukseen liittyvässä yhteisössä voidaan jakaa tietoa, ratkaista ongelmia tai miettiä laatuun kohdistuvia parannuksia. (Leffingwell ym. 2018, 127.)

Testauksen kehitys voidaan kohdentaa myös yksittäiseen laadun osa-alueeseen (Chopra 2018, 55):

- Vaatimusten mukaisuus eli kuinka ohjelmisto täyttää määritykset ja käyttäjien vaatimukset.
- Luotettavuus eli ohjelmiston toiminnan jatkuminen ilman keskeytyksiä.
- Suorituskyky eli kuinka ohjelmisto suoriutuu tehtävistään vaatimuksissa määritellyssä ajassa.
- Tiedon eheys eli tietojen oikeellisuus.
- Käytettävyys eli ohjelmiston helppokäyttöisyys.
- Ylläpidettävyys eli korjaamisen vaivattomuus.
- Joustavuus eli ohjelmiston muuttamisen helppous.
- Testattavuus eli ohjelmiston testaamisen helppous.
- Siirrettävyys eli ohjelmiston siirtämiseen eri laitteisto- ja/tai ohjelmistoalustalle vaadittu vaiva.
- Yhteen toimivuus eli työmäärä, joka vaaditaan, jotta ohjelmisto toimisi yhdessä muiden ohjelmistojen kanssa.
- Suojaus eli ohjelmiston suojaus ulkoisilta hyökkäyksiltä, joka voivat vahingoittaa sitä ja haitata sen käyttöä.

## 16 TESTAUKSEN JOHTAMINEN

Vesiputousmallissa testaus suunnitellaan yleensä itsenäiseksi toiminnoksi, jota vetää testauspäällikkö. Ketterässä testauksessa koko tiimi on vastuussa tuotteen testaamisesta ja testauspäällikköä ei ole. Resurssoinnin ja tarvittavan osaamisen varmistamisen osalta vastuu siirtyy scrum masterille, samoin kuin testauksen esteiden poistaminen. Ketterän kehittämisen käytännöillä korvataan testauksen kehittämiseen liittyvät tehtävät. (Linz 2014, 35-37.)

Kehittäjät ja testaajat ajattelevat usein eri tavalla. Kehityksen tavoite on suunnitella ja toteuttaa ohjelmisto. Testauksen tehtäviin kuuluu varmistaa, että toteutettava ohjelma on vaatimusten mukainen ja löytää viat ennen tuotantoon vientiä. Nämä ovat erilaisia tavoitteita, jotka vaativat erilaiset ajatusmallit. Testaajan ajatusmalliin pitäisi kuulua uteliaisuutta, kriittistä silmää, yksityiskohtien huomioimista sekä kyky hyvään vuorovaikutukseen. Kokemus muuttaa testaajan ajatusmallia tiedon lisääntyessä. Kehittäjän ajatusmalliin voi kuulua piirteitä testaajan ajatusmallista, mutta kehittäjät ovat usein kiinnostuneempia ratkaisujen suunnittelusta ja toteutuksesta, kuin sen pohtimisesta, mikä kyseisissä ratkaisuissa voisi olla väärin. (ISTQB 2018, 23.)

Siirryttäessä käyttämään ketterää ohjelmistokehitystä testaajien työ vaihtuu riippumattomasta testaustiimistä integroitumiseen kehitystiimiin. Muutos on suuri ja siitä voi aiheutua muutosvastarintaa. Koulutuksen tärkeyttä ketteriin käytäntöihin ei tule unohtaa siirtymävaiheessa (Gregory & Crispin 2015, 71). Myös kehittäjien työ muuttuu, sillä heidän oletetaan ottavan vastuuta testauksesta. Näihin tilanteisiin tarvitaan muutosjohtajuutta.

Muutosvastarinnan käsittely on muutosprosessin kriittisempiä osa-alueita. Jos muutos ei etene halutulla vauhdilla tai epäonnistuu, saatetaan syyksi nimetä muutosvastarinta. Muutoksen aikaan saaminen on vaikeaa, toisaalta muutosvastarinta on merkki siitä, että muutos on alkanut. Organisaatiot, jotka parhaiten onnistuvat muutosten läpiviennissä kerryttävät ymmärrystä muutoksesta ja kuuntelevat niitä työntekijäryhmiä, joita muutos koskee. Työkulttuurin tulisi olla avoin ja sallia myös negatiiviset tunteet. Negatiivisia tunteita herättävistä asioista pitää pystyä keskustelemaan. Onnistumisista ja muutoksella aikaansaaduista tuloksista kannattaa viestiä avoimesti. Vahvistuva muutosmyönteisyys ja positiivisuus ovat tärkeitä ominaisuuksia. Onnistumiseen tarvitaan, että henkilölle syntyy muutostahto ja tietoisuus muutoksen tärkeydestä. (Heiskanen & Lehikoinen 2010, 56-60.)

Edellisten lisäksi kannattaa viestiä muutoksen liiketoiminnalliset syyt ja tavoitteet, sekä se miksi muutos tarvitaan. Myös ero nykytilan ja tulevan käytännön välillä helpottaa muutoksen hyväksymistä. Yksilöllinen tuen määrä on eri henkilöillä erilainen ja tämä tulee ottaa huomioon muutoksessa. (Åhman 2004, 78 -80.)

Viitekehyksessä Scrumissa tiimin haluttaisiin työskentelevän samassa tilassa. Ennen pandemiaa tämä saattoi osin onnistua. SFe:ssa tiimien haluttaisiin kokoontuvan yhteiseen kaksi päivää kestävään PI-suunnitteluun. Pandemia on muuttanut myös tätä ja osin syynä voi olla, että SFe:ssa tarvitaan kehittäjiä paljon ja tiimejä on useita. SAFen tiimeissä on kehittäjiä useista eri kaupungeista tai jopa eri maista, jolloin tiimin työskentely siirtyy verkkoon. Tällöin tarvitaan etäjohtajuutta. Kuvio 18 kuvaa etäjohtamisen kulmakivet.



Kuvio 18. Etäjohtamisen kulmakivet (mukaillen Vilkmann 2016, 26)

Etäjohtajuuden kulmakivet ovat luottamus, arvostus, avoimuus, toimivat pelisäännöt, vuorovaikutus ja yhteisöllisyys. Kulmakivet linkittyvät toisiinsa. Luottamuksen rakentaminen kehitystiimin on tärkeää, jotta kehityskohteista voidaan puhua avoimesti. Luottamus kasvattaa luottamusta ja synnyttää halun olla tuottamuksen arvoinen. Arvostus puolestaan mahdollistaa hyvän yhteistyön. Työyhteisön tuki vähentää työn kuormittavuutta ja ehkäisee työuupumusta. Tiimi ja yksilötasolla on tärkeää, että kaikki tuntevat olonsa arvostetuksi ja että heidän työpanostaan pidetään arvossa. Avoimuus lisää viihtyvyyttä työssä, joka puolestaan vaikuttaa työmotivaatioon. Avoimuudella tarkoitetaan toiminnan läpinäkyvyyttä yhteistyössä ja päätöstenteossa sekä avointa kommunikaatiota, jossa kommunikaatio kulkee muuttumattomana organisaatiossa. Ilman avoimuutta yhteisöllisyys kärsii ja yksilöiden sitoutumiseen organisaatioon vähenee. Pelisäännöt raamittavat yhteistyötä. Selvät toimintatavat ja pelisäännöt helpottavat yhteistyötä, mutta niistä on sovittava yhdessä. Keskustelun kautta syntyy yhteinen näkemys ja tulkinta säännöistä. Jatkuva vuorovaikutus on merkittävä osa työn tekemistä ja sen kehittämistä. Kulmakivistä tämä sitoo kaikki muut tarvittavat asiat yhteen, sillä ilman vuorovaikutusta muut kohdat eivät toteudu. Yhteisöllisyyden kokemus lisää vuorostaan vuorovaikutusta. Hyvä yhteishenki, tekemisen meininki ja arvostava työilmapiiri ovat tärkeitä tekijöitä työssä viihtymisessä, näissä onnistuminen johtaa hyviin työtuloksiin. (Vilkman 2016, 25–44.)

## 17 ANALYYSI

### 17.1 Testauksen parhaat käytännöt

Laadukas testaus ja kehitys lähtee liikkeelle hyvistä vaatimuksista. Vaatimukset tulee olla yksiselitteisiä ja yhteisesti ymmärrettyjä. Epämääräiset vaatimukset tulee palauttaa takaisin määrittelyyn. Vaatimusmäärittelyn lisäksi tulisi tehdä tekninen dokumentaatio. Tämä tukee kehittäjiä ohjelmiston kehityksessä, toiminnallisuuden muutoksissa ja ylläpidossa sekä testaajia testitapausten kirjoittamisessa. Scrum viitekehyksessä ennen toiminnallisuuksien valintaa sprintille tulee järjestää palaveri, jossa tuotteen kehitys jonon yläpäässä olevat vaatimukset käydään lävitse tuoteomistajan johdolla. Ennen palaveria kehitystiimi on tutustunut vaatimuksiin. Palaverissa kehitystiimillä on mahdollisuus kysyä tarkennuksia vaatimuksiin. Vaatimuksissa tulee olla hyväksymiskriteerit, jotka pitävät sisällään vaatimukset myös ei-toiminnallisille ominaisuuksille kuten käytettävyydelle, tietoturvalle ja suorituskyvylle. Hyväksymiskriteereitä voidaan täydentää esimerkeillä. Esi-merkin tulee olla realistinen ja konkretisoida vaatimusta.

Ominaisuuksille tulisi kirjoittaa hyväksymistestit jo määrittely vaiheessa. Hyväksymistestien olisi hyvä olla päästä päähän testejä. Hyväksymistesteihin tulisi saada testaajaksi mielellään loppukäyttäjän edustaja tai liiketoimintaa ymmärtävä asiantuntija. Testitapaukset tulisi linkittää vaatimuksiin, näin voidaan varmistua, että kaikki vaatimukset tulee testatuksi. Useimmat testauksenhallintatyökalut sisältävät tämän toiminnallisuuden.

Testauksen suunnittelussa tulee päättää testauksen laajuus, testiautomaation käyttö sekä tarvitaanko erikoisosaamista vaativaa testausta kuten suorituskky, tietoturva tai käytettävyyys. Usein erikoisosaamista tarvitseva testaus ostetaan talon ulkopuolelta tai sen tekee asiaan perehtyneet henkilöt talon sisältä. Tällöin alustava aikataulu testaukseen täytyy olla tiedossa. Riippuvuudet toisiin järjestelmiin tai tiimeihin tulee ottaa huomioon testauksen suunnittelussa. Testaukseen liittyvät riskit tulisi tunnistaa, analysoida ja kirjata. Organisaation tulee päättää riski-indeksin taso, jonka ylittyessä riskiä pitää pienentää. Riski-indeksin tasoon vaikuttaa kehitettävän järjestelmän kriittisyys. Testauksen suunnittelussa pitää päättää metriikat, joilla testauksen edistymistä seurataan. Metriikat on hyvä valita niin, että ne saadaan automaattisesti työkaluista. Testisuunnitelmalla on kaksi roolia, se on sidosryhmille sopimus testauksen suorittamisesta tietyllä laajuudella ja tavalla sekä työohje testaajille.



Ei-toiminnallisen testauksen tarpeen tunnistaminen voi itsessään olla haaste. Scrumissa asiaa voidaan helpottaa luomalla tarkistuslistoja tarpeen tunnistamiseen. SAFessa ominaisuudet esitellään muille tuoteomistajille vaatimusten valmistuessa. Läpikäynnin tarkoitus on tunnistaa tiimit, jotka tarvitaan ominaisuuden toteuttamiseen. Läpikäynnissä saadaan liiketoiminnan näkemys ja mahdolliset tarkennukset ominaisuuteen. Tähän palaveriin kannattaa lisätä henkilöt, jotka pystyvät arvioimaan ei-toiminnallisten testausten tarpeen, silloin kun testaus tehdään tiimin ulkopuolella.

Testitapausten suunnittelu kannattaa aloittaa riskilähtöisenä testauksena, näin varmistetaan se, että testit tulee priorisoitua. Kehityksen kannatta hyödyntää riskianalyysiä töiden priorisoinnissa, tällöin virheiden korjaamiseen ja uudelleen testaukseen saadaan mahdollisimman paljon aika sprintin sisällä. Tarkoituksena on testata valmistuneista toiminnallisuuksista riskialttiimmat toiminnallisuudet ensin ja laajemmin. Testitapausten tarkkuuteen vaikuttavat, kohderyhmä, joille testitapaukset tehdään, sovitut käytännöt, sopimukset tai säädökset, testiautomaation käyttö sekä järjestelmän kriittisyys sekä monimutkaisuus. Testitapaukset tulee olla ymmärrettäviä, yksiselitteisiä ja uudelleen käytettäviä. Testitapausten tarkoituksena on löytää virheitä, lisätä luottamusta ohjelmiston toimintaa ja tuottaa tietoa ohjelmiston laadusta sidosryhmille.

Testauksen edistymistä tulee seurata sprintin aikana. Virheiden kirjoittaminen tulee ohjeistaa. Tavoitteena on, että kehittäjä pystyy toistamaan virheen ilman lisätietojen pyytämistä, näin voidaan nopeuttaa virheen korjaamista. Virheilmoituksen perusteella uudelleen testaus tulisi onnistua myös muilta tiimiläisiltä, kuin virheen kirjoittajalta. Isommissa projekteissa kannattaa hyödyntää virheraatia.

Testauksen työtavat tulisi olla samanlaiset tiimistä riippuen, jolloin tiedetään, että laadulle asetetut tavoitteet täyttyvät jokaisessa tiimissä. Tämä tuo mukanaan myös joustavuutta työkuorman jakamiseen, jos tiimit voivat aikataulun salliessa auttaa toisiaan.

## 17.2 Testiautomaatio

Testiautomaation avulla testauksesta saadaan nopeampi ja tehokkaampi pois lukien lyhyet projektit. Ketterässä testauksessa on melkein mahdotonta selvitä ilman testiautomaatiota nopean kehityssyklin ja osissa toteutettavan toiminnallisuuden vuoksi. Regressiotestauksen tarve on suuri ja koko kehitystiimin tulisi osallistua testiautomaation tekemiseen. Testiautomaatio osaamista tulisi olla kehittäjillä ja testaaajilla.

Testiautomaation tarve kasvaa mitä enemmän testattavaa, ketterässä kehityksessä ohjelmisto kasvaa jokaisen sprintin tai iteraation jälkeen, jolloin myös regressiotestauksen tarve kasvaa. Testiautomaatio mahdollistaa testien uudelleenkäytettävyyden investoinnin jälkeen pienin kustannuksin. Testejä voidaan ajaa eri ympäristöissä, käyttöjärjestelmissä, selaimissa ja eri tiedoilla, jolloin testauksen nopeutta ja kattavuutta saadaan laajennettua.

Testiautomaatiota kannattaa lähteä rakentamaan liiketaloudelle kriittisistä testeistä. Hyviä automatisoitavia testejä ovat myös testit, jossa samaa testiä ajetaan monilla eri arvoilla tai eri ympäristöissä. Jos käytettävissä on historia tietoa aikaisemmin rikkoutuneista toiminnallisuuksista, usein rikkoutuvat toiminnallisuudet olisivat hyvä saada testi-automaation piiriin.

Testiautomaation tulisi olla osa testauksen strategiaa ja sen käyttö tulisi huomioida testisuunnitelmassa. Testistrategian tulisi vastata kysymyksiin ”mitä” ja ”miksi”. Testisuunnitelman tulisi vastata kysymyksiin ”miten” ja ”milloin”. Lisäksi testiautomaatiolle tulisi luoda arkkitehtuurisuunnitelma, jossa perustellaan valittu työkalu ja mahdollinen jatkuvan integraation toteutus. Testiautomaation kehittämisessä olisi hyvä hyödyntää ohjelmistokehityksen parhaita käytäntöjä.

### 17.3 Riskit ja vaaranpaikat

Ketterässä testauksessa kehitystiimi on vastuussa testauksesta, jolloin koko tiimin pitää sitoutua testaukseen. Tiimillä ei ole käytössä testauspäällikköä, joka innostaisi ja sitouttaisi tiimiä testaukseen. Liiketoiminta tai loppukäyttäjät tulee myös sitouttaa hyväksymistestaukseen. Hyväksymistestausta tehdään yleensä oman työnsä ohessa ja loppukäyttäjän edustajien ollessa kiireisiä, sopivan ajan löytäminen ei välttämättä ole helppoa ja usein he tarvitsevat tukea testauksessa. Testauspäällikön puuttuminen ei helpota asiaa.

Testiautomaation osaaminen on kysytty ominaisuus ja yritykset joutuvat kilpailemaan osaajista. Kehittäjät joutuvat opettelemaan testiautomaation tekemisen. Testiautomaatiota tarvitsee ylläpitoa ja tämä tulee huomioida sprintin tai iteraation kapasiteetissa laskettaessa.

Kulttuurin muutos siirryttäessä riippumattomasta laadunvarmistustiimistä kehitystiimeihin saattaa aiheuttaa testaajissa yhteisöllisyyden menettämisen tunnetta. Muutoksessa tarvitaan muutosjohtajuutta, ettei ajauduta tilanteeseen, jossa testaajat päättävät vaihtaa

työpaikkaa. Kehittäjien työ muuttuu myös, sillä heidän tulee ottaa enemmän vastuuta testauksesta ja mahdollisesti laajennettava osaamistaan testiautomaation tekemiseen.

Testauksen tavoitteet ja käytännöt tulisi määrittää testauspolitiikka ja -strategia dokumentteihin. Yhtenäisten testaus ja laadunvarmistus käytäntöjen levittäminen tiimeihin voi osoittautua haastavaksi.

Testauksen resurssointi voi olla haastavaa, sillä kehitystiimin kokoaminen jää scrum masterille, jolloin hänellä tulisi olla ymmärrystä testauksesta ja testaajan osaamisesta. Tietoturvatestaus, suorituskkytestaus ja käytettävyyden testaus vaativat testaajilta erikoisosaamista ja yleensä testaajat ovat erikoistuneet näihin testauksiin. Testaustarpeen tunnistaminen ja sen ajoittaminen projektissa saattaa osoittautua haastavaksi samoin kuin testaajien löytäminen. Erikoistuneita testaajia ei kannata pitää tiimissä koko aikaa, vaan vain tarvittaessa.

Ketterässä testauksessa testaajan ammattitaito korostuu, sillä tiimissä on usein vain yksi testaaja. Uusien juuri alalle tulleiden testaajien perehdyttämisessä, tulee ongelma, ellei häntä voida laittaa hetkeksi kokeneemman testaajan kaveriksi.

Haasteita saattaa tulla myös testaajien rekrytoinneissa, jos testaukseen perehtyneitä henkilöitä ei rekrytoivien henkilöiden joukosta löydy. Työpaikkailmoitus kertoo osaavalle testaajalle paljon rekrytoivasta yrityksestä, jos vaadittavat taidot ovat ilmoitettu epämääräisesti, tehtävän kuvaukseen sopimattomasti tai ilmoitukseen on listattu kaikki mahdollinen osaaminen, eivät ammattitaitoiset testaajat jätä hakemusta.

Kehitystiimissä on testaajia vähemmän kuin kehittäjiä. Vaarana on, että testauksen kehitys jää retrospektiivissä vähemmälle huomiolle.

Kehittäjän tekemät yksikkötestit kirjoittaja yleensä sama henkilö, joka kirjoittaa koodin. Väärin ymmärretyt vaatimukset periytyvät testeihin, tästä syystä katselmoinnit tulisi koskea koodin lisäksi testejä. Laaja testikattavuus saattaa antaa väärän turvallisuuden tunteen, jos testit ovat laadultaan heikkoja.

#### 17.4 Testauksen kehittäminen ja johtaminen

Testauksen kehittäminen tapahtuu ketterässä kehityksessä sprintin tai iteraation jälkeen pidettävässä retrospektiivi -palaverissa. Palaverissa käsitellään koko ohjelmistokehityksen parantamista ja tiimi äänestää valittavan kehityskohteen yhdessä. Testauksen ja

laadunvarmistuksen puolesta testaajan pitää pystyä kommunikoimaan perustellusti ja vakuuttavasti, jotta testaukseen liittyvät parannusehdotukset tulevat valituksi.

Testaajien motivaation ja yhteisöllisyyden tunteeseen tulee kiinnittää huomiota. Testaajien työ muuttuu siirryttäessä vesiputousmallista ketterään kehittämiseen. Muutos luultavasti tuo lisätarpeita kouluttautumiselle testiautomaation sekä ketterän kehityksen ja testauksen osalta. Muutoksessa testaajat täytyy saada sitoutumaan uusiin tehtäviin ja tämä koskee myös kehittäjien sitouttamista testaamiseen.

Ketterässä testauksessa testauspäällikön työt jakautuvat testaajille, scrum masterille ja osin laatupäällikölle. Testaajan työstä tulee vastuullisempaa ja muuttuneet työtehtävät voivat myös innostaa ja motivoida testaajia. Tämä on kuitenkin yksilöllistä ja esihenkilön tulee huomioida tämä ja tarjota tukea, jos suhtautuminen on päinvastainen.

Testausta ohjaavat testauksen politiikka ja -strategia, jossa määritellään tavoitteet ja yhteiset toimintatavat. Toimintatavat tulee saada levitettyä kehitys organisaatioon. Toimintatapojen kehittyessä niihin tehdyt muutokset tulee myös jalkauttaa tiimeihin.

## 18 JOHTOPÄÄTÖKSET

Johtopäätöksissä pohditaan ketterää testausta opinnäytetyön kirjoittajan oman kokemuksen kautta. Siirryttäessä vesiputousmallista ketterään kehittämiseen tarvitaan kouluttamista, johdon tukea ja aikaa. Ilman aikaa uuden työtavan käyttöönotto on mahdottomaa. Testaajien työ siirryttäessä ketterään kehitykseen muuttuu ja vastuuta tulee testaajalle enemmän, sillä osa testauspäällikön työstä siirtyy heille. Tässä vaiheessa tarvitaan muutosjohtajuutta ja motivointia. Kokeneet testausasiantuntijat ovat erittäin kysytyjä ja on oikea vaara, että he vaihtavat työpaikkaa, jos muutoksen johtamisessa epäonnistutaan. Tämä toki koskee myös kehittäjiä, mutta työtavan muutos ei heillä ole yhtä suuri. Ketterässä testauksessa tarvitaan kokenutta ja ammattitaitoista testaajaa, sillä usein tiimissä on vain yksi testaaja.

Kun siirtyminen on saatu vietyä lävitse ja organisaatio on valinnut omaan organisaatioon sopivan ketterän kehittämisen viitekehyksen ja mahdollisesti tehnyt malliin omat muutoksensa päästää tutkimuskysymyksen - miten testausta tulee tehdä ketterässä ohjelmistokehityksessä?

Testaus saattaa jäädä kehityksen varjoon, vaikka organisaatioille olisi tärkeää määrittää haluttu laatutaso ja keinot sen saavuttamiseksi. Testipolitiikka ja -strategia vastaa näihin kysymyksiin, mutta aika harvassa organisaatiossa ne on tehty. Testauksen onnistumista kannattaa mitata, jolloin organisaation kehitystiimit saadaan vertailukelpoisiksi keskenään ja tuloksia voidaan käyttää testauksen kehittämisessä. Tuloksia voidaan käyttää testauksen kehittämiseen. Yhteiset työtavat organisaatiossa on erittäin hankalaa saada aikaiseksi, varsinkin jos niitä ei ole määritelty. Valmiin määritelmät auttavat, mutta eivät korvaa testipolitiikka ja -strategia dokumentteja.

Testilähtöinen kehitys on avain tekijä ketterän kehittämisen onnistumisessa. Testilähtöisessä kehityksessä kirjoitetaan ensin testi ja vasta sen jälkeen koodi. Työtapa pitää sisällään paljon testiautomaatiota kaikilla muilla testitasoilla paitsi hyväksymistestauksessa. Testiautomaation tekemiseen pitää saada sitoutettua sekä kehittäjät että testaajat, myös johdon ja asiakkaan pitää ymmärtää, että testiautomaation ylläpitäminen vaatii työtä ja tämä tulee laskea mukaan tiimin kapasiteetti.

Testilähtöinen kehittäminen vähentää selvästi virheiden määrää, mutta silti sitä ei ole otettu lähestymistavaksi useissakaan ketterää kehittämistä toteuttavassa tiimissä.

Testiautomaation käyttöönotto on soittautunut haastavaksi monessa organisaatiossa. Testiautomaation käyttöönottoa on mahdollisesti kokeiltu konsulttien tehdessä pohjatyön tarkoituksena, että kehitystiimi jatkaa työtä. Valitettavasti usein on käynyt niin, että kiireessä testiautomaatio on se, joka jää ensin pois. Kiire harvoin hellittää niin paljon, että testiautomaatiota ehditään ottaa uudestaan käyttöön ja tehdä kehitetyille toiminnallisuuksille testit. Testiautomaation pitäisi luoda käytäntö, jossa testiautomaation rikkoutuessa kuka tahansa tiimistä ennen seuraavan työn aloittamista korjaa rikkoutuneet testit. Näin testien omistajuus saadaan jaettua tiimin kesken. Testiautomaatiota tulisi tehdä testauksen eri tasoilla, jos testiautomaatiota tehdään vain järjestelmätestauksessa, testipyramidi kääntyy ympäri ja laajaa testikattavuutta testiautomaation osalta ei saavuteta ja yleisesti ottaen käyttöliittymää testaavat testit ovat hitaampia. On muistettava, että ohjelmisto kasvaa jokaisen tuotantoon viennin jälkeen ja tämä lisää regressiotestauksen tarvetta. Ilman testiautomaatiota testauksen tarve kasvaa liian suureksi nopeassa kehitysyklissä.

Hyväksymistestaukseen suositellaan tutkivaa testausta, joka on tehokas tapa testata ja löytää hyvin virheitä. Hyväksymistestaukseen tulisi saada testaajiksi loppukäyttäjia tai liiketoiminnan asiantuntijoita. Parhaassa tapauksessa tutkivan testauksen lisäksi tehdään hyväksymiskriteereihin perustuvat testitapaukset samaan aikaan vaatimusten kanssa. Hyväksymiskriteerit tulee olla yksiselitteiset ja mitattavat ja niiden tulisi kattaa myös ei-toiminnalliset vaatimukset. Useat organisaatiot eivät kirjoita vaatimuksiin hyväksymiskriteereitä tai ne eivät ole laadukkaita. Tässä tilanteessa kehitystiimin tulisi vaatia tai pyytää tuoteomistajalta niiden kirjaamista ja pyytää tarkennuksia, jos vaatimus tai hyväksymiskriteeri on epäselvä.

Ei-toiminnallisen testauksen tarve tulee määrittää ja aikatauluttaa. Kehitystiimeissä ei yleensä ole näiden testausten asiantuntijoita käytettävissä koko ajan ja usein testaukset ostetaan organisaation ulkopuolelta.

Ketterään testaukseen on paljon ohjeita ja nevoja kirjallisuudessa. Ketterän testaus tuo testaajille mahdollisuuden kasvattaa omaa osaamistaan ja ottaa lisää vastuuta testauksesta. Yksi ketterän testauksen parhaita puolia on, että testaajat pääsevät mukaan heti projektin alusta asti ja voivat tuoda laadun näkökulmaa esiin alusta asti.

## LÄHTEET

- Bath, G., & van Veenendaal, E., (2014). *Improving the Test Process*, Rock Nook.
- Baumgartner, M., Klonk, M., Mastnak, C., Pichler, H., Seidl, R., & Tanczos, S., (2021). *Agile Testing : The Agile way to quality*. Springer.
- Black, R. (2014). *Advanced Software Testing Vol.2*. Rock Nook.
- Black, R. (2017). *Agile Testing foundations: an ISTQB foundation level agile tester guide*. BCS Learning & Development.
- Black, R., van der Aalst, L., & Rommens, J. L., (2017). *The Expert Test Manager*. Rock Nook.
- Boby, J. (2021). *Test Automation A manager's guid*. BCS Learning and Development.
- Burnstein, I. (2003). *Practical Software Testing*. Springer.
- Callaway, J., & Hunt, C., (2018). *Practical Test-Driven Development using C# 7*. Packt Publishing.
- Chopra, R. (2018). *Software Quality Assurance: A Practical Approach*. Mercury Learning and Information.
- Cloudt, G., (2021). *What is software quality? Understanding what really matters in software development*. Independently published.
- Cohn, M. (2010). *Succeeding with Agile Software Development Using Scrum*. Pearson Education.
- Gregory, J., & Crispin, L. (2015). *Agile Testing: A Practical Guide for Testers and Agile Teams*. Pearson Education.
- Hambling, B. (2015). *Software testing: an ISTQB-BCS certified tester foundation guide*. BCS Learning & Development.
- Heiskanen, M., & Lehtikainen, S. (2010). *Muutosviestinnän voimapaperi*. Talentum.
- Hendrickson, E. (2013). *Explore It! Reduce Risk and Increase Confidence with Exploratory Testing*. The Pragmatic Programmer.
- Irshad, M., Börstler, J., & Petersen, K., (2022). Supporting refactoring of BDD specifications—An empirical study, *Information and software technology*, 2022-01, Vol.141, p.106717. <https://doi.org/10.1016/j.infsof.2021.106717>
- ISTQB (2015a). Sertifioitu Testaaja. Jatkotason sertifikaattisisältö Testauspäällikkö. (käännösversio 2015, alkuperäisversio 19.10.2012,) <https://tivia-jasenyhdistyksen.fi/fistb-testi/wp-content/uploads/sites/30/2020/12/Advanced-Syllabus-2012-TM-Financial.pdf>
- ISTQB (2015b). Sertifioitu testaaja. Perustason sertifikaattisisältö Ketterä testaaja. (käännösversio 27.4.2015, alkuperäisversio 31.5.2014), [https://fistb.fi/wp-content/uploads/sites/30/2021/05/FND-Agile-Syllabus\\_FI.pdf](https://fistb.fi/wp-content/uploads/sites/30/2021/05/FND-Agile-Syllabus_FI.pdf)

ISTQB (2018). Sertifioitu testaaaja. Perustason sertifikaattisisältö. (käännösversio 10.10.2018, alkuperäisversio 4.6.2018) <https://tivia-jasenyhdistykset.fi/fistb-testi/wp-content/uploads/sites/30/2020/12/CTFL-2018-Sertifikaattisisalto-20181010-1-Valmis.pdf>

Kasurinen, J. (2013). *Ohjelmistotestauksen käsikirja*. Docendo

Kinsbruner, E. (2018). *Continuous Testing for Devops Professionals*. ICGtesting

Lew, P. (2016). Agile Testing Metrics: Quality Before Velocity. *Software quality professional*, 2016-06-01, Vol. 18 (3), p.51-60., [https://scholar.google.com/scholar?cluster=10007437569780302804&hl=fi&as\\_sdt=2005&sciodt=0,5](https://scholar.google.com/scholar?cluster=10007437569780302804&hl=fi&as_sdt=2005&sciodt=0,5)

Linz, T. (2014). *Testing in Scrum*. Rocky Nook

Lucassen, G., Dalpiaz, F., van der Werf, J. M., & Brinkkemper, S., (2016). Improving agile requirements: the Quality User Story framework and tool. *Requirements engineering*, 2016, Vol.21 (3), p.383-403. doi:<https://doi-org.ezproxy.cc.lut.fi/10.1007/s00766-016-0250-x>

McGreal, D., & Jocham, R., (2018). *The professional product owner leveraging scrum as a competitive advantage*. Addison-Wesley.

Nicolette, D., (2015). *Software Development Metrics*. Manning.

O'Regan, G., (2019). *Concise Guide to Software Testing*. Springer.

Perry, W., (2006). *Effective methods for software testing*. Wiley

Pugh, K., (2011). *Lean-Agile Acceptance Test-Driven Development*. Pearson Education.

Ripley, R., & Miller, T., (2020). *Fixing your Scrum*. The Pragmatic Programmers.

Rubin, K., (2013). *Essential Scrum*. Pearson Education.

Saalem, R. M., Qadri, S., ul Hassan, I., Bashir, R. N., & Ghafoor, Y., (2014). Testing Automation in Agile Software Development, *International journal of innovation and applied studies*, 2014-11-01, Vol.9 (2), p.541-541 <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.676.1465&rep=rep1&type=pdf>

Savaspuro, M., (2019). *Itseohjautuvuus tuli työpaikoille, mutta kukaan ei kertonut, miten sellainen ollaan*. Alma.

Scaled Agile. (2021). Design thinking. Viitattu 11.6.2022, <https://www.scaledagileframework.com/design-thinking/>

Smart, J. F., (2015). *BDD in Action*. Manning.

Stuard, R., (2017). *ISO/IEC/IEEE 29119 SOFTWARE TESTING STANDARDS, A Practitioner's Guid*. STA Testing Consulting

Todaro, D., (2019). *The Epic Guide to Agile*. R9.

Vilkman, U., (2016). *Etäjohtaminen*. Talentum



Watkins, J., & Mills, S., (2011). *Testing IT: An Off-the-Shelf Software Testing Process*. Cambridge University Press.

Åhman, H., (2004). *Menestyvä johtaminen – haasta itsesi*. WSOY