Opinnäytetyö

Teknologiaosaamisen johtaminen

YTEJOS17

2024

Anne Heikkilä

VAATIMUSHALLINNAN KEHITTÄMINEN OHJELMISTOPROJEKTEISSA

– Testauksen näkökulma

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (YAMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Insinööri (ylempi AMK), Teknologiaosaamisen johtaminen

2024 | 65 sivua

Anne Heikkilä

VAATIMUSHALLINNAN KEHITTÄMINEN OHJELMISTOPROJEKTEISSA

- Testauksen näkökulma

Opinnäytetyössä käsiteltiin vaatimusten hallintaa ketterässä kehityksessä testauksen näkökulmassa, kartoitettiin sen vahvuuksia sekä heikkouksia. Kehittämisen kohteita kartoitettiin esille perehtymällä alan kirjallisuuteen sekä selvittämällä aiheen ympärille rakentuvien tutkimustöiden lopputuloksiin. Tutkimustyön taustavaikuttajana ja motivaattorina vaikutti toimeksiantaja, jonka asiakasprojektit ovat olleet arvokkaita tilaisuuksia peilata teorian tietoja käytännössä todennettuihin havaintoihin.

Kehittämiskohteiden selvittämiseen käytiin läpi ketterän menetelmän periaatetta, selvittämällä vaatimusten elinkaarta ja mitä odotuksia asetetaan vaatimusten rakenteelle, perehtymällä testauksen luonteeseen ja sen merkitykseen ohjelmiston laatuun. Työn keskipisteenä ei ole ollut suositella tiettyä mallia tai menetelmää, vaan tavoitteena on ollut tuoda esille kehittämiskohteita, jotka on huomioitavissa missä tahansa projektissa.

Työn päätelmissä ja kehitysehdotuksissa todetaan, että tehokkaita ja monipuolisia menetelmiä, prosesseja ja työkaluja on tarjolla runsaasti, jotka ovat konfiguroitavissa jokaiselle projektille alasta riippumatta. Kansainväliset standardit myötävaikuttavat alan ammattilaisten yhteisymmärryksen muodostumiseen yli kulttuurirajojen, mutta edellytys onnistuneelle projektille muodostuu kuitenkin toimivasta yhteistyöstä ja riittävästä ammattitaidosta. Tämä vaatii varsinkin konsulteilta nopeaa adaptoitumiskykyä sekä jatkuvaa itsensä kehittämistä.

ASIASANAT:

Agile, Jira, Kanban, Scrum, TMap

MASTER'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Master of Engineering, Technology Competence Management

2024 | 65 pages

Anne Heikkilä

DEVELOPING REQUIREMENTS MANAGEMENT IN SOFTWARE PROJECTS

- Testing perspective

In this thesis, the concept of management in agile software development projects from the perspective of testing, its strengths and weaknesses was researched. The aim was to find development topics from the literature and articles in the field. The background influencer and motivator of the research work was the client, whose customer projects have been valuable opportunities to mirror theoretical knowledge in practically verified observations.

Principles of agile method was reviewed, by finding out the life cycle and what expectations are placed on its structure, by learning about the nature of testing and its importance to the quality of the software. The focus of the work was not to recommend specific models or methods. Rather, the goal was to highlight areas of development that can be considered in any project.

In the conclusions and development proposals of this thesis, it can be said that software development projects must carefully choose methods, processes and tools, pay attention to configure tools and use them systematically and consistently. International standards contribute to the formation of a consensus among professionals in the field across cultural boundaries, but the prerequisite for a successful project is still functional cooperation and sufficient professional skills. This setting, especially for consultants, provides quick adaptability and continuous self-development.

KEYWORDS:

Agile, Jira, Kanban, Scrum, TMap

SISÄLTÖ

KÄYTETYT LYHENTEET TAI SANASTO **7**

1 JOHDANTO **8**
1.1 Työn taustaa 8
1.2 Työn tavoitteet ja tutkimusongelma 9
1.3 Työssä käytettävät tutkimusmenetelmät 10

2 KETTERÄ MENETELMÄ **11**
2.1 Scrum-tiimin roolit, tehtävät ja toimintaperiaate 11
2.2 Scrum-viitekehyksen soveltuvuus ja haasteet 16
2.3 Kanban-tiimin roolit, tehtävät ja toimintaperiaate 17
2.4 Kanban-viitekehyksen soveltuvuus ja haasteet 18
2.5 Scrum- ja Kanban-prosessia tukeva työkalu – Jira 19
2.6 Ketterässä menetelmässä todennettuja havaintoja 24

3 VAATIMUSTENHALLINTA **26**
3.1 Laadunhallintastandardit ISO 9000 26
3.2 Vaatimusmäärittely 29
3.3 Käyttäjävaatimukset (user requirements) 31
3.4 Järjestelmävaatimukset (system requirements) 33
3.5 Muutosvaatimusten hallinta ja jäljitettävyys 35
3.6 Vaatimusmäärittelyssä todennettuja havaintoja 37
3.7 Vaatimusten jäljitettävyydessä todennettuja havaintoja 38

4 OHJELMISTOTESTAUS **40**
4.1 Testausstandardi ISO/IEC/IEEE 29119 40
4.2 TMap elinkaarimalli järjestelmä- ja hyväksymistestauksessa 41
4.3 Järjestelmätestaus 43
4.4 Hyväksymistestaus 43
4.5 Regressio- ja automaatiotestaus 44
4.6 Työkalut ja menetelmät 44
4.7 Testauksen kattavuus 45
4.8 Ohjelmistotestauksessa todennettuja havaintoja 47

5 OHJELMISTON LAATU **49**

5.1 Laatustandardi ISO/IEC 25010 49
5.2 Näkökulmia ohjelmiston laatuun 50
5.3 Hyväksymiskriteerit 52
5.4 Testauksen kustannukset 53
5.5 Testauksen luotettavuusarviointi 54

6 PÄÄTELMÄT JA KEHITYSEHDOTUKSET 55

LÄHTEET 63

KUVAT

Kuva 1 Ohjelmistoprojektin vaatimustasojen hierarkia (Pelin 2011, 200). 8
Kuva 2 Tuotteen työlistan skaalauksen perustasot (Auer ym. 2013, 11). 13
Kuva 3 Scrum-malli (Kasurinen 2013, 28). 15
Kuva 4 Evo- ja protoilumallien toimintaperiaate (mukaillen Haikala & Mäkijärvi 2001, 30, 33). 15
Kuva 5 Kanban-taulu (Juvonen 2018, 24). 18
Kuva 6 Jira sovelluksen työkulku (Jira Software Support 2024). 19
Kuva 7 Jirassa uuden ohjelmistoprojektin Scrum-taulun yleisnäkymä 20
Kuva 8 Jirassa ohjelmistoprojektin Scrum-taulun suunnittelun työalusta (Jira Software Support 2024). 20
Kuva 9 Jirassa ohjelmistoprojektin Scrum-taulun tarjoamat raportit (Jira Software Support 2024). 21
Kuva 10 Jirassa uuden ohjelmistoprojektin Kanban-taulun yleisnäkymä 22
Kuva 11 Jira projektin Kanban-taulun suunnittelun työalusta 23
Kuva 12 Prosesseihin perustuvan laadunhallintajärjestelmän toimintamalli (SFS-EN ISO 9000 2005, 14). 27
Kuva 13 Vaatimuksien yhteydessä käytettävä termistö (SFS-EN ISO 9000 2005, 58) 28
Kuva 14 Vaatimusmäärityksien tarkastuksessa käytettävät termit (SFS-EN ISO 9000 2005, 60) 28
Kuva 15 Projektiin kohdistuvat vaatimukset (Ruuska 2012, 280). 29
Kuva 16 Toiminnallisuuteen kohdistuvia vaatimuksia 30
Kuva 17 Käyttäjävaatimusten muotoutuminen järjestelmän ominaisuuksiksi (Haikala & Mäkijärvi 2001, 145). 33
Kuva 18 Järjestelmävaatimusten määrittely- ja hallintaprosessi (mukaillen Forselius 2013, 31). 34
Kuva 19 Järjestelmävaatimusten tunnistamisen prosessi (mukaillen Terho ym. 2008, 16). 34
Kuva 20 Vaatimustenhallinnan prosessikuvaus 36
Kuva 21 Testauksen V-malli (Kasurinen 2013, 51). 40
Kuva 22 Standardin mukainen testauksen hallintaprosessi (ISO/IEC/IEEE 29119-1:2022) 41
Kuva 23 Standardin mukainen dynaaminen testausprosessi (ISO/IEC/IEEE 29119-1:2022) 41
Kuva 24 TMap elinkaarimalli (TMap 2016). 42

Kuva 25 Jira työkalun Xray osio 44
Kuva 26 Testauksessa Jira tikettien assosiaatiot (mukaillen Xray 2023). 45
Kuva 27 Jira-työkalun testauskattavuus raportti 46
Kuva 28 Kuvaus Jira-työkalun jäljitettävyysraportin rakenteesta 46
Kuva 29 Ohjelmiston standardin mukaiset laatuominaisuudet (mukaillen ISO 25010, 2024). 50
Kuva 30 Laadun hinta (mukaillen Kasurinen 2013, 133) 53
Kuva 31 Vaatimustenhallinnan prosessikuvaus 61

TAULUKOT

Taulukko 1 Sidosryhmien projektiin kohdistuvien odotusten prioriteetit (Ruuska 2012, 281). 30

KÄYTETYT LYHENTEET TAI SANASTO

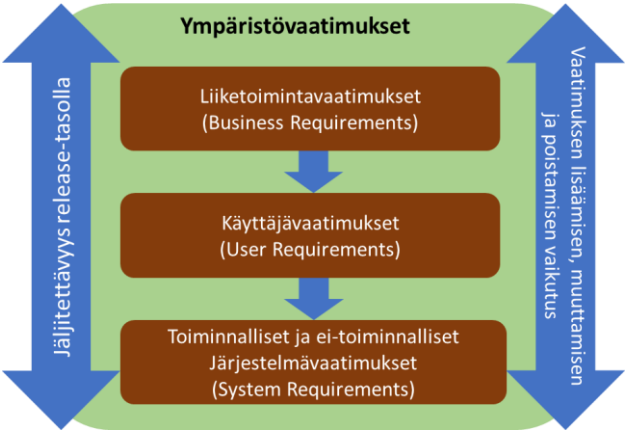
AI	Artificial Intelligence. Tekoäly eli keinoäly
Backlog Grooming	Kehitysjonon jalostaminen
Daily Scrum	Päivittäinen Scrum-palaveri
Developers	Kehittäjät
DoD	Valmiin määritelmä (Definition of Done)
DoR	Valmistellun määritelmä (Definition of Ready)
Epic	Kehitysjonon laajaa osa-aluetta käsittävä asia (eepos, eeppinen)
Feature	Ohjelmiston ominaisuus
ICT	Information and communication technology
Kanban	Projektin/tuotannon ajoitus-/ohjausjärjestelmä
Product Backlog	Tuotteen kehitysjono
Product Owner	Tuoteomistaja
Sprint Retrospective	Sprintin restrospektiivi palaveri (Retro)
TMap	Test Management Approach
Scrum	Projektinhallinnan viitekehys, ketterä kehitystoimintamalli
Scrum Master	Projektihallinnan viitekehyksen rooli Scrum-mestari
Scrum Team	Projektihallinnan viitekehyksen tiimi
Sprint Backlog	Sprintin kehitysjono
Sprint Planning	Sprintin suunnittelupalaveri
Sprint Review	Sprintin katselmuspalaveri
Sprint Goal	Sprintin tavoite
Story Point	Käyttäjätarinalle määritellyt pisteet
User story	Käyttäjätarina

1 JOHDANTO

1.1 Työn taustaa

Toimeksiantaja on yksityinen, maailmanlaajuinen, konsulttiyritys, joka keskittyy ohjelmistojen testaukseen, laadunvarmistukseen, kyberturvallisuuteen sekä automaatio-, AI- ja analytiikkapalveluihin. Opinnäytetyössä tutkitaan julkishallinnon ohjelmistoprojektin vaatimusmäärittelyn hallintaa testauksen näkökulmasta ja kartoitetaan mahdolliset kehittämisen kohteet. Projektia hallitaan ketterässä ohjelmistokehityksessä, jossa projektinhallinnan viitekehyksenä toimii kehitystiimissä Scrum ja ylläpitotiimissä Kanban. Varsinainen projekti perustuu verkkoselaimella käytettävään käyttöliittymään, jonka ohjelmistokehyksenä toimii Angular.

Projektin perustuksena on vaatimusmäärittelyt, jonka ehdoilla projekti etenee ja toteutuu. Vaatimusmäärittelyt sanelevat myös resurssitarpeen ja asettaa aikatauluhaasteet. Kuvassa 1 on havainnollistettu kokonaisuudessaan ohjelmistoprojektin vaatimusmäärittelyn ympäristö. Hitaasti tai jopa nopeastikin muuttuvan ympäristön vaatimukset edellyttävät vaatimusmäärittelyjen muutosten hallintaa eli jonkin asteista muutoshallintaprosessia ja tämä sisältää tarpeen jäljittää tietty vaatimusmäärittely jopa release-tasolle asti (Juvonen 2018, 59).



Kuva 1 Ohjelmistoprojektin vaatimustasojen hierarkia (Pelin 2011, 200).

1.2 Työn tavoitteet ja tutkimusongelma

Tutkimuksen tavoitteena on ketterässä ohjelmistokehityksessä testauksen näkökulmasta selvittää ja kehittää vaatimusmäärittelyjen jäljitettävyyttä release-tasolle asti sekä pohtia testauksen roolin merkitystä vaatimusten määrittelyssä jo projektin alkuvaiheessa. Tutkimuksessa ei oteta kantaa eri testausmenettelyihin, koska tutkimusaiheeseen nähden se on epärelevanttia tietoa. Tutkimuksessa seurataan ohjelmiston vaatimusmäärittelyjen elinkaarta kehitystiimistä ylläpitotiimiin.

Tutkimusongelma on siten:

Miten vaatimusmäärittelyt ovat jäljitettävissä ketterässä projektissa?

Tutkimusongelmaa lähestytään testauksen näkökulmasta seuraavilla kysymyksillä:

Miksi vaatimusmäärittelyt on oltava helposti jäljitettävissä ketterässä ohjelmistoprojektissa?

Millainen vaatimusmäärittely on edellytys ketterässä projektissa?

Ketterän ohjelmistokehityksen Scrum ja Kanban käytettävyys, vahvuudet ja heikkoudet?

Miksi testauksen osallisuus on tärkeää jo määrittelyn alkuvaiheessa ja millainen vaikutus sillä on kustannustehokkuuden kannalta?

Tutkimuskysymyksiin kohdistuu seuraavanlaisia hypoteeseja:

1. Vaatimusmäärittelyjä ei kyetä jäljittämään - Hypoteesi: Duplikaattien, yhdestä virheestä laadittujen useiden havaintoraporttien, määrä kasvaa ja niistä aiheutuva ylimääräinen työ. Aiheettomien ylimääräisten havaintojen kasaantuminen ja niistä aiheutuva ylimääräinen selvitystyö voi aiheuttaa ristiriitoja muutosvaatimuksien yhteydessä.
2. Projektitiimin henkilöstön vaihtuminen - Hypoteesi: Vaikutus työmääräarviointeihin aiheuttaen arviolle suuremman toleranssin. Uudella henkilöllä vaikeaa sisäistää projektin sisältöä, joka hidastaa projektin etenemistä. Tiimistä poistuva henkilö vie mukanaan arvokasta tietoa.
3. Asiakkaan ymmärrys tuotteesta - Hypoteesi: Vaikutus projektitiimiin, loppukäyttäjiin, projektin toteutumiselle siten, että aikataulu viivästyy, kustannukset kasvavat. Aikataulussa ja budjetissa pidättäytyminen taasen heikentää tuotteen laatua.

4. Projektitiimin ymmärrys tuotteesta - Hypoteesi: Vaikuttaa projektin aikatauluun, budjettiin ja laatuun siten, että aikataulu viivästyy, kustannukset kasvavat. Aikataulussa ja budjetissa pidättäytyminen taasen heikentää tuotteen laatua.
5. Ylläpidolla ei läpinäkyvyyttä vaatimusmäärittelyihin - Hypoteesi: Käyttäjätuen ja jatkuvan kehitystyön laatu heikkenee.
6. Työkalun puutteellinen ja osaamaton käyttö - Hypoteesi: Raportit puutteellisia ja epäluotettavia, jolloin hyöty jää vähäiseksi ja voivat antaa jopa väärää indikaatiota projektin ja tuotteen tilanteesta.
7. Sidosryhmien välinen yhteisymmärryksen ja osaamisen puute - Hypoteesi: vaikutus aikatauluun (venyy), budjettiin (kasvaa) ja laatuun (heikkenee).
8. Kohtien (1–7) vaikutus testauksen kustannusten arviointiin, kustannustehokkuuteen - Hypoteesi: Arviointi vaikeutuu, kustannustehokkuus kärsii.

1.3 Työssä käytettävät tutkimusmenetelmät

Tutkimusongelmaa lähestytään kirjallisuustutkimuksella, joka tutkimusmenetelmänä on aikataulullisesti joustava, koska tutkimus ei välttämättä ole riippuvainen ulkopuolisista tekijöistä kuten haastatteluista, mutta edellyttää monipuolista, oikeanlaista ja kattavaa tutkimuskirjallisuuden kokoamista ja analysointia.

Tutkimusongelma itsessään voidaan mieltää ilmiöksi: Ongelma on kaikille projekteille yhteinen (toimialasta riippumatta), mutta sen juurisyy on kuin veteen piirretty viiva. Tästä syystä kyseistä menetelmää hyödyntäen tutkimus toteutetaan tutkimalla aiheesta jo tehtyjä tutkielmia päätelmineen sekä aiheen kirjallisuuden teoriaa toisiinsa vasten peilaten.

2 KETTERÄ MENETELMÄ

Ohjelmistoprojektin menetelmällä, käytettävillä viitekehyksillä sekä konkreettisilla, digitaalisilla, työkaluilla on merkittävä vaikutus testauksen laadulliseen suunnitteluun, toteutukseen ja ylläpitoon, siten syy-seuraus-suhde ohjelmiston laatuun. Kaikkien mainittujen osatekijöiden luonteen ymmärtäminen on oleellista havaitakseen vaatimushallinnan mahdolliset puutteet ja heikkoudet, joten tulevissa kappaleissa syvennytään teoriassa ohjelmistoprojekteissa tyypillisesti käytettyyn ketterän menetelmän (agile method) Scrum ja Kanban viitekehyksiin niin tuotekehitys- kuin jatkuvan kehittämisen tiimissä. Sen lisäksi käydään läpi näiden viitekehysten tyypillisiin työmääräarvion menettelyihin sekä aikataulun laatimiseen ja kustannusarvio/-seurannan käytäntöihin. Viimeisessä kappaleessa otteita muiden tutkimustöiden huomioista ja havainnoista, koskien ketterää menetelmää.

2.1 Scrum-tiimin roolit, tehtävät ja toimintaperiaate

Ketterän menetelmän ei-hierarkkinen Scrum-tiimi (Scrum Team) koostuu seuraavista rooleista: Tuoteomistajasta (Product Owner), Scrum-mestarista (Scrum Master) ja kehittäjistä (Developers), joista kehittäjät, sisältäen myös testaajat, toteuttavat varsinaisen tehtävätasoisen suunnittelun, toteutuksen ja testauksen (Juvonen 2018, 19).

Projektin kokoaikainen tuoteomistaja (Product Owner) voi olla asiakkaan projektipäällikkö tai toimittajan tekninen projektipäällikkö, jolla on vahvaa substanssiosaamista, mutta rooli ei edellytä varsinaista ohjelmistoalan osaamista. Tehtäviin kuuluu muun muassa päättää ja määritellä riittävällä tiedolla tuotteen ominaisuudet (käyttäjätarinat valmiinmääritelmillä, tuotteen kehitysjono), määrittää sprinttien tavoitteet (sprint goal), olla ajan tasalla toiminnallisuuteen vaikuttavista tekijöistä sekä valmius reagoida nopeasti tiimin kysymyksiin koskien tuotetta. Scrum-menetelmä edellyttää, että tässä roolissa on vain yksi henkilö, joten on huomioitavaa mahdollisen päätösprosessin laatiminen, josta vastaa projektipäällikkö. Projektin koosta ja rakenteesta riippuen voi olla lisäksi nimetty eeppisen tason tuoteomistaja (Epic Product Owner) vastaamaan eeppisen tason työlistasta ja priorisoinnista, ominaisuuden tuoteomistaja (Feature Product Owner) vastaamaan ominaisuustason (feature) työlistasta ja priorisoinnista sekä tiimin tuoteomistajat

vastaanottamaan ohjeita priorisoida tuotteen työlistaa. (Juvonen 2018, 19; The Scrum Guide 2020, 5.)

Scrum-mestari (Scrum Master) nimensä mukaisesti huolehtii, että projekti etenee menetelmän mukaisesti ja että tiimi ymmärtää ja on sitoutunut noudattamaan Scrum-sääntöjä. Roolin tehtäviin kuuluu lisäksi suunnitella sprintin tehtävälistat, järjestää ja johtaa palavereja, huolehtia tiimin toiminnan tehokkuudesta tuottaa aikataulun mukaisesti valmis määritelmiä täyttäviä inkrementtejä. Scrum-mestari toimii tuoteomistajan, kehitystiimin ja organisaation rajapinnoilla. (Juvonen 2018, 19; The Scrum Guide 2020, 6–7.)

Ketteryydelle tyypillinen itseohjautuva ja itsenäinen tiimi käsittää kaikki ne henkilöt, jotka käytännössä tekevät varsinaista tuoteversiota. Tiimin jäsenet halutaan nähdä tasavertaisina ja tasa-arvoisina toimijoina, joilla kaikilla on tarvittava tieto ja osaaminen. Tarvittaessa jokainen pystyy suunnittelemaan käyttöliittymää, koodaamaan ja testaamaan sovellusta. Kaikkien tiimiläisten tittelinä käytetään kehittäjä, jos titteleitä halutaan käyttää, mutta ketään ei tulisi erikseen nimetä kehittäjäksi, testaajaksi tai käyttöliittymän suunnittelijaksi eli tiimiin ei tulisi muodostaa alitiimejä. (Juvonen 2018, 19–20; The Scrum Guide 2020, 5–6.)

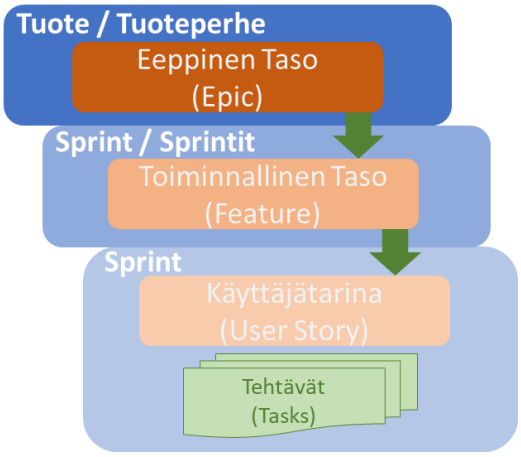
Tiimin tulee olla pieni, jotta menetelmä pysyy ketteränä, mutta riittävän suurena (vähintään 3 kehittäjää), jotta sprintteihin sopiva tehtävälista saadaan merkittävän tuottoisaksi ja julkaisukelpoinen versio. Liian suurena tiiminä pidetään yli yhdeksän toimijaa, jolloin prosessi ei ole enää hallittavissa ketterin menetelmin. (The Scrum Guide 2020, 6.)

Scrum-menetelmän perusperiaatteena on tuottaa asiakkaalle ohjelmisto nopeasti, mutta mahdollisimman vähäisellä resurssilla sekä kyky reagoida nopeasti muutos- ja ongelmatilanteissa. Jotta menetelmän toimintaperiaate pääsee oikeuksiinsa, edellyttää se asiakkaan ohjelmistoon kohdistuvien määrittelyjen nopean kokonaiskuvan kuin myös yksityiskohtien ymmärtämistä. (Auer ym. 2013, 43; Kasurinen 2013, 27.)

Skaalauksella tarkoitetaan ohjelmiston jakamista pienempiin teknisiin järjestelmän osiin eli komponentteihin, jotka voivat jakaantua vielä pienempiin komponentteihin. Yksittäisen komponentin edistämisestä huolehtii työmäärästä tai sen laajuudesta riippuen tiimi tai yksittäinen ihminen. Ohjelmiston skaalauksesta huolimatta ketterässä menetelmässä edellytetään, että kehittäjä tieto- ja taitotasoltaan kykenee ymmärtämään ohjelmistoa kokonaisuudessaan, jolloin ennaltaehkäistään rikkomasta jo toteutettua toimivaa toiminnallisuutta. Koska periaatteena on edistää ohjelmistoa nopeasti, tapahtuu kehittäjien kes-

kuudessa lyhyessä ajassa runsaasti muutoksia koodiin, edellyttää tämä versionhallinnassa muutosten läpinäkyvyyttä kooditasolla asti. Tällä metodilla vältetään varsinaiseen kommunikaatioon kuluva aika, kun kaikki viimeiset muutokset näkyvät koodissa, kehittäjästä riippumatta. (Auer ym. 2013, 11.)

Tuotteen hierarkkisesti skaalautuva työlista (product backlog) eli kehitysjono sisältää tuoteomistajan listaamat toiminnallisuudet, vaatimukset ja muutokset, jotka kirjataan esimerkiksi käyttäjätarinoiksi (user story), joiden ei-toiminnalliset vaatimukset vastaavasti muodostavat varsinaiset testitapausten kriteerit. Backlog grooming -palaverissa käsitellään tuotteen kehitysjonoa pitkälle aikajaksolle, johon osallistuu tilaajan ja projektiryhmän edustajat. Tuotteen kehitysjono elää koko projektin ajan, jonka sisällöstä, saatavuudesta ja priorisoinnista on vastuussa tuoteomistaja. Käytännössä on havaittu, että jos tuotteen työlistalla alkaa olla yli 100 artikkelia, priorisoiminen alkaa käydä haastavaksi. Tällöin nostetaan abstraktiotasoa, jolloin käyttäjätarinan seuraava ylätaso olisi toiminnallinen taso eli featuretaso. Featuretason elinkaari voi kestää useammankin sprintin yli. Seuraava abstraktiotaso featuretasosta olisi eeppinen taso, jonka elinkaari voi käsittää yhdestä tuotteesta jopa kokonaisen tuoteperheen. Kuvassa 2 on havainnollistettu yleisesti vaatimushierarkia, mutta jos halutaan käyttää varsinaista skaalauksen menetelmää, skaalauksen viitekehys SAFe (Scaled Agile Framework) on tunnettu metodi. (Auer ym. 2013, 11–12; Juvonen 2018, 21; The Scrum Guide 2020, 10–12; Atlassian 2024.)



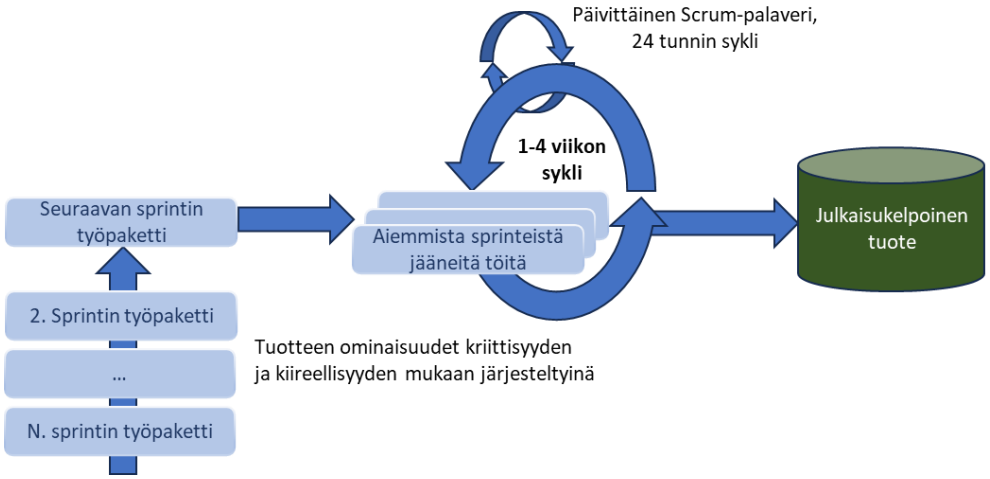
Kuva 2 Tuotteen työlistan skaalauksen perustasot (Auer ym. 2013, 11).

Sprinttien suunnittelussa (sprint planning) Scrum-mestari poimii tuotteen kehitysjonosta sprintin kestolle (1–4 viikkoa) sopivaksi mitoitetun tehtävälistan (sprint backlog), joka on muokattavissa ainoastaan kehitystiimin keskuudessa. Tehtävälistan käyttäjätarinoille on

määritelty DoR (definition of ready) eli valmistellun määritelmä sekä DoD (definition of done) eli valmiin määritelmä, ennen kuin käyttäjätarina on voitu ottaa sprinttiin työstettäväksi. Siinä missä tuotteen kehitysjono elää ja muotoutuu projektin edetessä, myös sprintin kehitysjono muovautuu tiimin yhteistyössä muun muassa päivittäisien palaverien (daily scrum) tuotoksena. Sprintti on kestoltaan enintään 4 viikkoa, jonka aikana tavoitteena on toteuttaa sprintin suunnittelussa tuotteen kehitysjonosta poimitut tehtävät. Sprintin päätteeksi tuotetaan inkrementti, jossa valmiin määritelmä on saavuttanut tilan Done eli valmis. (Auer ym. 2013, 75; Juvonen 2018, 20; The Scrum Guide 2020, 7–8, 12–13.)

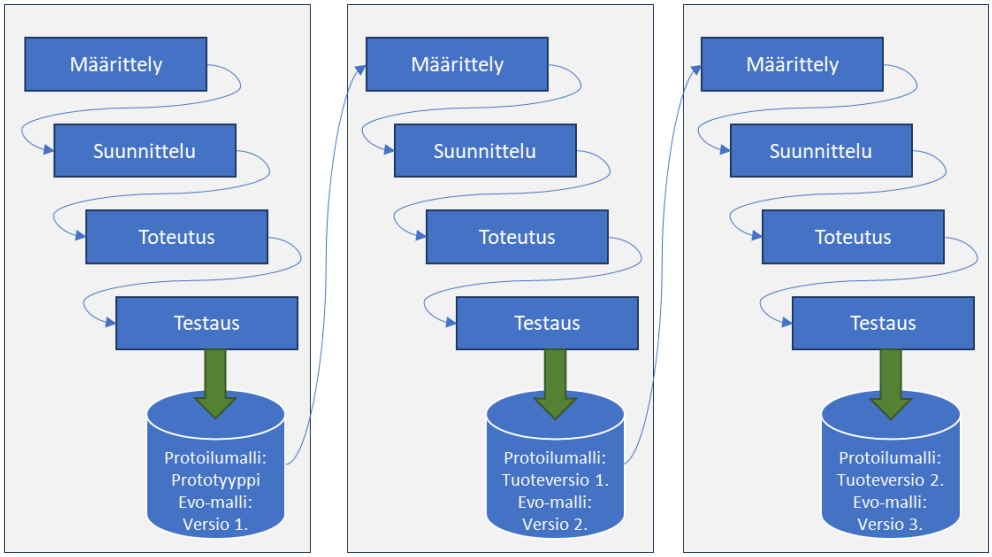
Sprintin päätyttyä pidetään sprintin katselmointi (sprint review), jossa demotaan iteraation koeversio eli julkaisukelpoinen sovellus sidosryhmien keskuudessa, jossa käydään läpi sprintin alussa asetetut tavoitteet ja analysoidaan tavoitteiden onnistuminen. Demotilaisuuden jälkeen pidetään tiimin ja tuoteomistajan kesken retrospektiivi (sprint retrospective), jossa analysoidaan ja ratkotaan kokonaisuudessaan menneen sprintin heikkoudet ja vahvuudet, jotka tullaan huomioimaan tulevissa sprinteissä. Tärkeää on myös pohtia refaktoroinnin tarve ja suunnitella tarvittava aika ja resurssi myös refaktoroinnille. Seuraava sprint alkaa suunnitelmapalaverilla, jossa mitoitetaan seuraavan sprintin tehtävälistaa työkuormineen ja määritellään samalla tuotteen kehitysjonon käyttäjätarinoille valmiin määritelmä eli millä edellytyksellä ne voidaan poimia tulevan sprintin tehtävälistalle. (Auer ym. 2013, 12, 75; Juvonen 2018, 22–23: The Scrum Guide 2020, 10–12.)

Scrum-prosessi voidaan siis nähdä jaksona (kuva 3), joka kokonaisuudessaan toistuu projektin edetessä 1–4 viikon jaksoina. Jokainen sprintti ei välttämättä tarkoita sitä, että tuotetaan valmis tuotantoversio, mutta menetelmä mahdollistaa jo tuotannossa olevan version päivittämistä tai indikoi ei-tuotantovalmiin version konkreettista edistymistä asiakkaalle eli asiakas saa arvion, koska sovellus on julkaisuvalmis.



Kuva 3 Scrum-malli (Kasurinen 2013, 28).

Tuotekehityshankkeiden läpiviennissä on tyypillisesti ollut käytössä vaihejakomalleja kuten Evo-malli (evolutionary delivery), joka tunnetaan myös inkrementaaliseksi malliksi sekä erilaiset protoilumallit. Kuvassa 4 on sisällytetty samaan kuvaan molempien mallien, Evo- ja protoilumallin toimintaperiaate. Scrum-menetelmä toimii osittain samalla periaatteella, mutta on pyrkinyt selättämään kyseisien toimintamallien ongelmat. Näitä ovat olleet Evo-mallissa inkrementtien kehitystyön keskeytyminen asiakkaan ongelmien ja virheiden selvittely- ja korjaustöiden takia, protoilumallissa asiakkaan virheellinen käsitys tuotteen kypsyydestä tai loputon prototyypin kehittäminen. (Haikala & Mäkijärvi 2001, 30–33.)



Kuva 4 Evo- ja protoilumallien toimintaperiaate (mukaillen Haikala & Mäkijärvi 2001, 30, 33).

2.2 Scrum-viitekehyksen soveltuvuus ja haasteet

Scrum-viitekehys soveltuu varsinkin uutta sovellusta tuottaviin tuotekehitysprojekteihin tai konversioprojekteihin, joissa halutaan saada tuotantoon tuote, jonka varsinainen lopputulos ei ole kuitenkaan kiveen kirjoitettua. Menetelmä vaiheistaa ja kontrolloi projektin etenemistä, mutta ei ota kantaa varsinaisiin käytäntöihin. Tämä tekee projektin etenemisestä lineaarisuuden sijaan joustavan ja vaiherikkaan. Käytännössä tämä tarkoittaa sitä, että tuotetta voidaan kehittää ja tarkentaa projektin edetessä, jolloin tilaajan on helpompi tehdä tarvittavia muutoksia määrittelyihin projektin edetessä.

Menetelmä hyödyntää tehokkaasti toimijoiden tietoja ja taitoja, jolloin yrityksen ei tarvitse hakea vain suppeaa, tietyn osa-alueen osaajaa, vaan menetelmä mahdollistaa, että jokainen panostaa osaamistaan projektissa. Tämä hyödyntää yritystä siltäkin osin, että vaikka projekti vaihtuu, osaaminen tulee jakaantumaan ja joustamaan vaatimusten vaihtuessa. Scrum-tiimi ja kehitystiimi ovat taitavan Scrum-mestarin toimesta integroituneet toimivaksi kokonaisuudeksi, joka tuottaa mitä tahansa, joka soveltuu ketterän menetelmän projektiksi. Ohjelmiston sijaan esimerkiksi mikä tahansa ei-ohjelmistoa sisältävä prototyyppi tai ohjelmiston ja hardwaren integrointi voidaan kyseisellä menetelmällä varmistaa toimivaksi kokonaisuudeksi ja saattaa tuotantolinjalle.

Sprintin suunnittelupalaverin yhteydessä työmäärän arviointi ja siihen käytettävä menetelmä, joita on tarjolla lukuisa määrä (esimerkiksi COCOMO, toimintopisteanalyysi jne), voi koitua haasteelliseksi ja huonosti toteutettuna voi aiheuttaa suuria ja kauaskantoisia ongelmia, kuten projektin alustavan aikataulun ja siihen laaditun budjetin pettäminen. Ehkä tunnetuin sprintin suunnittelussa käytetystä menetelmästä pidetään suunnittelupokerin eri variaatiota, mutta menetelmästä riippumatta, onnistuneeseen realistiseen arvioon päästään vain, jos vaatimus on laadittu riittävän ymmärrettävästi ja selkeästi. Lisäksi jos suunnittelupokerin korttipöydässä on vain osa tiimin jäsenistä, riski sprintin virheelliseen työmääräarviointiin on suurempi, kuin että koko tiimi, mukaan lukien asiakas, osallistuisi suunnittelupalaveriin. (Grenning 2002; Juvonen 2018, 75, 77–79, 82–83.)

On huomioitava työmäärän arviota laadittaessa, että ajallisen suunnittelun tasoja on karkeasti jaettuna kaksi: Iterointisuunnittelu, joka on lyhytkestoinen ja käyttäjätarinat yksityiskohtaisesti kuvattuja (toteutustaso) sekä julkaisusuunnittelu, joka on pitkäkestoinen ja vaatimukset yleisellä tasolla kuvattuja. Projektin laajuuden määrittämistä pidetään kuitenkin prioriteetiltaan suurempana tavoitteena kuin yksittäiset iteraatiosuunnittelun aika-

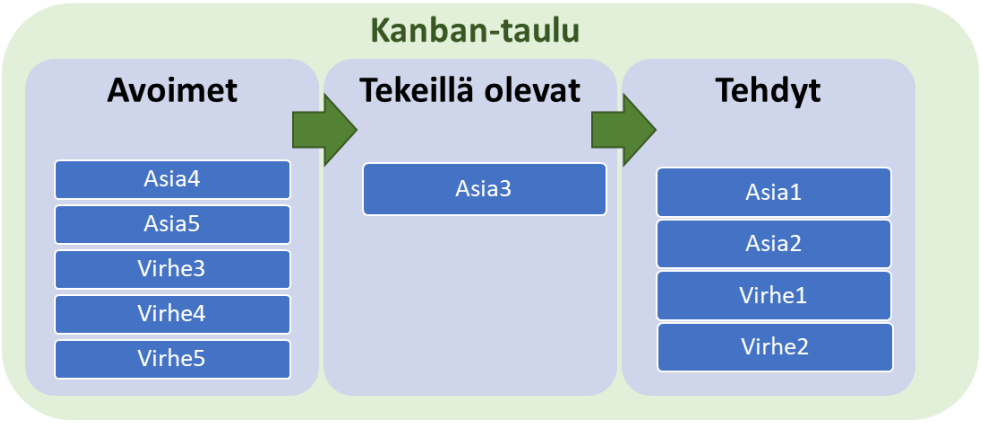
arviot, joten käyttäjätarinoiden työmääräarviot sallitaan yli- tai aliarvioitaviksi. Suuntaa antavat arviot ovat kuitenkin nopeampi tapa edetä, kuin pyrkiä erittäin tarkkaan arvioon. (Grenning 2002.)

Käyttäjätarinoiden työmääräarvion hankaluuksien lisäksi myös selkeän valmiin määritelmä tuottaa ongelmia eli koska käyttäjätarina voidaan asettaa tehdyksi ja koska sprintin tavoite on kokonaisuudessaan toteutunut. Käyttäjätarinan yhteydessä valmiinmäärittelyt edellyttävät konkreettisesti suoritettavia tehtäviä, joista on tiimin keskuudessa päästy yhteisymmärrykseen. (Sammons 2019, 118–119)

2.3 Kanban-tiimin roolit, tehtävät ja toimintaperiaate

Kanban-tiimi voi koostua ensimmäisen tuotantoon viennin Scrum-tiimin jäsenistä eli osa tiimin jäsenistä siirtyy jatkamaan ylläpidossa, jatkuvan kehittämisen parissa. Kanban-tiimi koostuu Scrum-tiimin tapaan tuoteomistajasta (Product Owner) sekä kehittäjistä (koodaajat ja testaajat), joiden tehtävät eivät poikkea Scrum-tiimin tehtävistä, mutta merkittävin muutos tapahtuu tiimin menetelmissä. Scrum-masterin roolia ei enää käytetä Kanban tiimissä.

Kanban on Scrum-menetelmän tapaan myös nopea ja kustannustehokas, mutta toimintaperiaatteeltaan suoraviivaisempi kuin Scrum-menetelmä. Kanban on terminä lähtöisin japanin kielen sanasta, joka tarkoittaa taulua tai kylttiä. Sanansa mukaisesti menetelmä visualisoi työnkulkua käyttämällä joko virtuaalista tai konkreettista taulua, jossa on kolme saraketta: avoimet tehtävät, tekeillä olevat tehtävät ja tehdyt tehtävät (kuva 5). Taululle tuodaan tuotteen työlistalta muun muassa Scrum-tiimin jäljiltä jääneet tekemättömät tehtävät sekä tuotannossa ilmenneitä virheitä, muuttuvia ja uusia määrityksiä (Juvonen 2018, 24).



Kuva 5 Kanban-taulu (Juvonen 2018, 24).

Kanban taulu helpottaa suunnittelemaan, priorisoimaan ja visualisoimaan realistista työmäärää, resurssitarvetta, noudattaa tekemisen ja asioiden läpinäkyvyyttä tiimiläisten keskuudessa sekä ehkäisee edistymisen pullonkauloja aikajanalla. Käytännössä taulu toimii siten, että avoimista siirretään tehtävä aina keskimmäiseen tekeillä olevat -sarakkeeseen ja valmiiksi suoritettu tehtävä siirretään tehtyjen sarakkeeseen. Taulu havainnollistaa välittömästi jumittuneet tehtävät, jolloin kuka tahansa tiimiläisistä voi ottaa tehtävän työnalle, kun on saanut keskeneräisen työnsä tehtyä. (Sammons 2019, 171–175.)

2.4 Kanban-viitekehyksen soveltuvuus ja haasteet

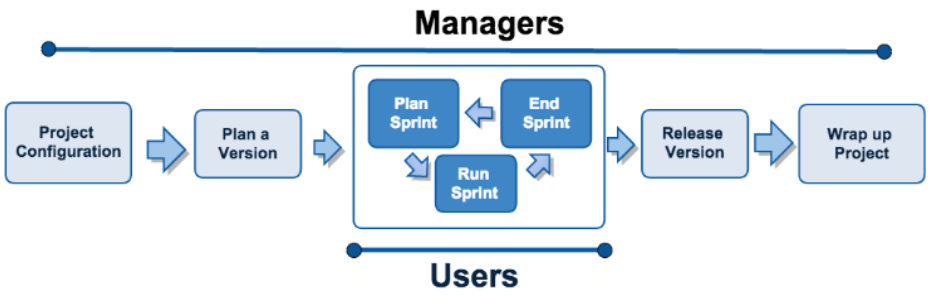
Kanbanin menestystarinoihin ehkä tunnetuimmaksi lukeutuu Toyotan tuotantolinja, mutta yksinkertaisuudessaan ja joustavuudessaan menetelmä soveltuu erinomaisesti myös ohjelmistotuotannon projekteihin, varsinkin tuotantoon viennin jälkeisessä jatkuvan kehittämisen (ylläpito) työskentelyssä, mutta projektin laajuuden mukaan se on sovellettavissa myös kehitysympäristössä ennen tuotantoon vientiä esimerkiksi suuressa projektissa feature-tason toteutuksessa (Juvonen 2018, 24). Varsinkin jatkuvan kehittämisen on reagoitava ja vastattava nopeasti tuotannossa ilmenneisiin, yllättäviinkin, ongelmatilanteisiin kuin myös asiakkaan ja loppukäyttäjän suunnalta esiin tulleisiin muutosvaatimuksiin tai tarpeisiin lisätä uusia ominaisuuksia. Näin ollen myös tuotannon päivitysversioita odotetaan toteutuvan nopeallakin aikataululla edellyttäen ketterää viitekehystä (Haikala & Mäkijärvi 2001, 43). Testaukseen kuluvan ajan huomioinen eli ennakoiminen näissä olosuhteissa tuo aikatauluun omat haasteensa, koska tuotannossa ilmenevät ongelmatilanteet eivät ole ennakoitavissa ja työmääräarvio on tehtävissä vasta kun ongelma ja toteutuksen suuruus on onnistuttu rajaamaan.

Kanban-menetelmä ei onnistu täysin välttämään Scrum-menetelmälle ominaisia haasteita ja ongelmia eli käsiteltävät määrittelyt voivat edelleen tuottaa erinäisiä haasteita, vaikka edistymisen pullonkauloja se Scrum-menetelmää nopeammin kykenisikin taklaamaan. Menetelmän taustalla toimii tiimi ja eri vaiheiden osa-alueilla voidaan tehdä inhimillisiä virheliikkeitä, jotka vaikuttavat Kanba-menetelmän tarkoitukseen ja tehokkuuteen. Esimerkiksi tuotteen työlistalta poimitut tehtävät avoimien listalle voivat muodostaa liian pitkän tehtävälistan, joka tuottaa tiimille henkistä kuormitusta jo sen nähdessään, tehtävien prioriteettien järjestys ei ole asianmukainen ja keskeneräisiä tehtäviä siirretään valmiiksi. (Sammons 2019, 171–175.) Toisaalta kokonaisuuden tilannearvio välittyy nopeasti taululta ja näin ollen esimerkiksi testauksen resurssitarpeisiin voidaan reagoida ennakoivasti.

2.5 Scrum- ja Kanban-prosessia tukeva työkalu – Jira

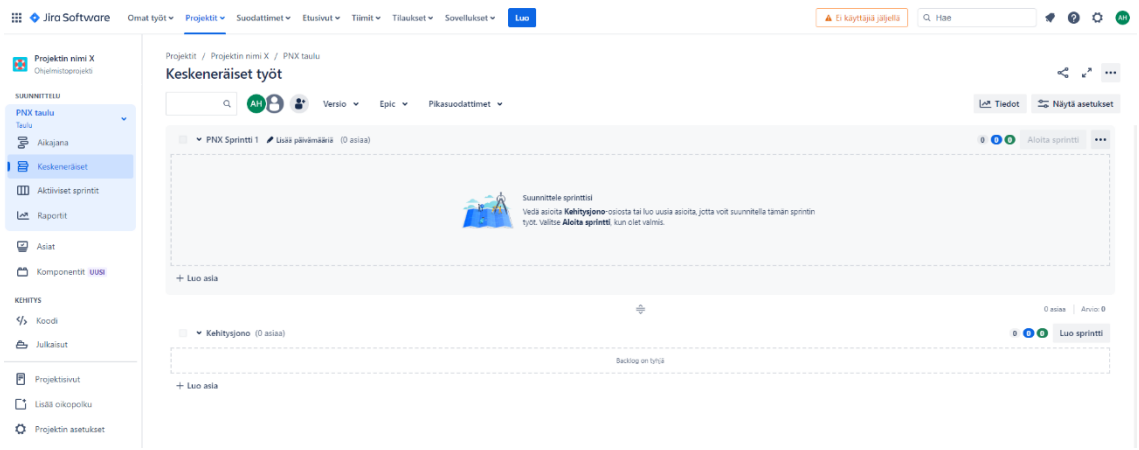
Ketterää menetelmää tukevan työkalun tulee ominaisuuksiltaan myös jäljitellä ketterän menetelmän luonnetta eli joustaa erilaisille käytännöille, mutta kuitenkin pitää muuttujat organisoidusti helposti saavutettavissa ja luettavassa muodossa: Ketterät taulut, tuotteen ja sprintin kehitysjonon helppo hallinta, projektien ja asioiden hallinta, ketterät raportit, mukautetut työnkulut jne. Varsinaiseen työkalun toiminnallisuuteen tutkimuksessa ei perehdytä sen syvällisemmin, mutta tulee kuitenkin huomioida kokonaisuudessa, koska työkalu toimii konkreettisena, visuaalisena muuttujana ketterässä ympäristössä, jossa kaikki aktiviteetti muuttuu näkyväksi ja odotetusti jäljitettäväksi. Tutkimuksessa tutustutaan lähemmin Atlassian tarjoamaan Jira käyttöliittymään.

Jira sovellus tukee erilaisia projektin hallintamenetelmiä kuten Scrum- ja Kanban-menetelmää. Kuvassa 6 havainnoidaan Scrumia käyttävän tiimin työnkulkua sovelluksessa.



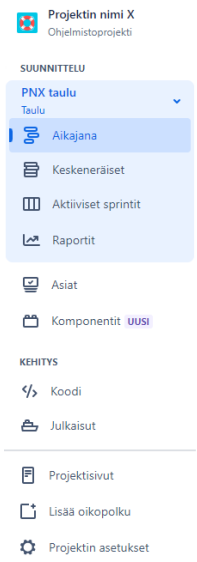
Kuva 6 Jira sovelluksen työkulku (Jira Software Support 2024).

Sovelluksen käyttö aloitetaan konfiguroimalla sovellus projektitiimin tarpeisiin soveltuvaksi, joten esimerkissä (kuva 7) on luotu ensin ohjelmistoprojekti, jota hallitsee scrum-tiimi ja luotu projektille Scrum-taulu, jossa kehitystiimin tuotetta konkreettisesti käsitellään (Jira Software Support 2024).



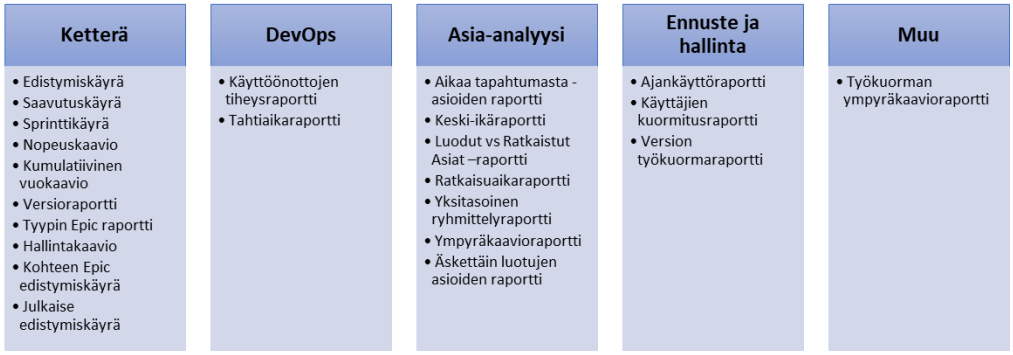
Kuva 7 Jirassa uuden ohjelmistoprojektin Scrum-taulun yleisnäkymä

Sivun vasemmasta reunasta (kuva 8) löytyy projektin suunnittelua ja toteutusta tukevat toiminnallisuudet: Aikajana, keskeneräiset, aktiiviset sprintit ja raportit.



Kuva 8 Jirassa ohjelmistoprojektin Scrum-taulun suunnittelun työalusta (Jira Software Support 2024).

Aikajana-osiossa on seurattavissa sanansa mukaisesti aikajanalla sprinttien ja julkaisujen edistymistä. Keskeneräiset-osiossa suunnitellaan varsinainen sprintti, jossa kehitysjonon sisältä on nähtävissä ja josta on helposti tuotavissa tehtäviä sprinttiin tai voidaan luoda uusia tehtäviä eli sprintin kehitysjono. Aktiiviset sprintit-osiossa tapahtuu varsinainen sprint-aikajanalla tehtävien eteneminen sarakkeissa tehtävälista, käynnissä ja valmis. Raportit-osiossa on tarjolla lukuisia eri tarkoituksien seurantaan soveltuvia raportteja: Ylätasolla mainittuna Ketterä, Devops, Asia-analyysi, Ennuste ja hallinta sekä Muu (kuva 9). Raportti-osuuden heikkoutena mainittakoon Jira-serverin asettama rajoitus, jonka kapasiteetti rajoittuu lataamaan CSV-tiedostoon korkeintaan 1000 asiaa. (Jira Software Support 2024.)



Kuva 9 Jirassa ohjelmistoprojektin Scrum-taulun tarjoamat raportit (Jira Software Support 2024).

Asiat-osiossa luodaan projektille asiatyypit kuten eepos, tarina, tehtävä (sekä alitehtävä) tai virhe eli kyseisessä osiossa käsitellään varsinaisen tuotteen kehitysjonon tehtäviä. Asiatyypit sisältävät oletusarvoisia pakollisia ja ei-pakollisia kuvaus- ja kontekstikenttiä. Esimerkiksi asiatyyppi tarina sisältää oletusarvoisesti kentät: Tila, yhteenveto, kuvaus, käsittelijä, leimat, isä (esimerkiksi eeppinen-taso), sprint, story point estimate, raportoija, liite ja linkitetyt asiat. Kuvaus- ja kontekstikentät ovat muokattavissa projektin tarpeiden mukaiseksi eli tarpeettomat kentät ovat poistettavissa ja muita kenttiä lisättävissä kuten ihmisiä ja ryhmiä koskevat kentät, järjestelmää käsittävät kentät, päivämäärää ja aikaa koskevat kentät sekä muut kentät. Tässä kohden huomioitava raporttien hyödyllisyys eli jos jätetään jokin konteksteista pois, sitä kontekstia käyttävä raportti on myös tällöin hyödytön. (Jira Software Support 2024.)

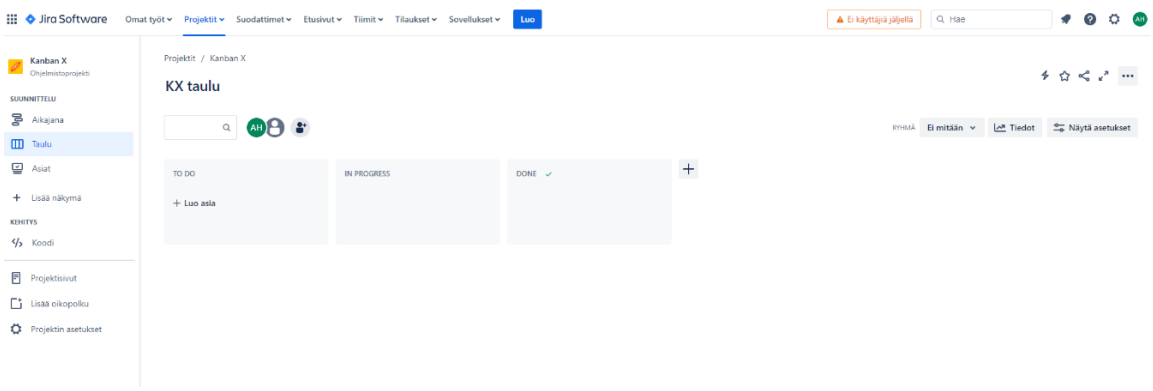
Kehitys työalustassa on valittavissa koodi ja julkaisut -osiot, jossa koodiosiossa voidaan liittää yhteys toisiin palveluntarjoajiin kuten projektin versionhallintaan. Julkaisut-osiossa

näytetään version tiedot, eli julkaisun eteneminen ja historiatieto. Projektisivut-osiossa voidaan hallinnoida Scrum-projektin dokumentaatiota eli työkalu tarjoaa työalustan myös palaverien pöytäkirjoille ja yleisesti projektin suunnitelmille. Lisää oikopolkuosion kautta voidaan liittää esimerkiksi lähdekoodin hallintapalvelu tai muita projektille tärkeitä verkkosivujen linkityksiä. (Jira Software Support 2024.)

Projektin asetuksissa löytyy muun muassa asioiden manuaalista käsittelyä korvaava automatiikka, jolla säästetään huomattavasti aikaa esimerkiksi ohjelmistoa koskevaa automatiikka asettamaan prioriteetin perusteella määräajan tai projektin hallintaa koskevaa automatiikkaa asettamalla virheen luonnin yhteydessä seuraajan (Jira Software Support 2024).

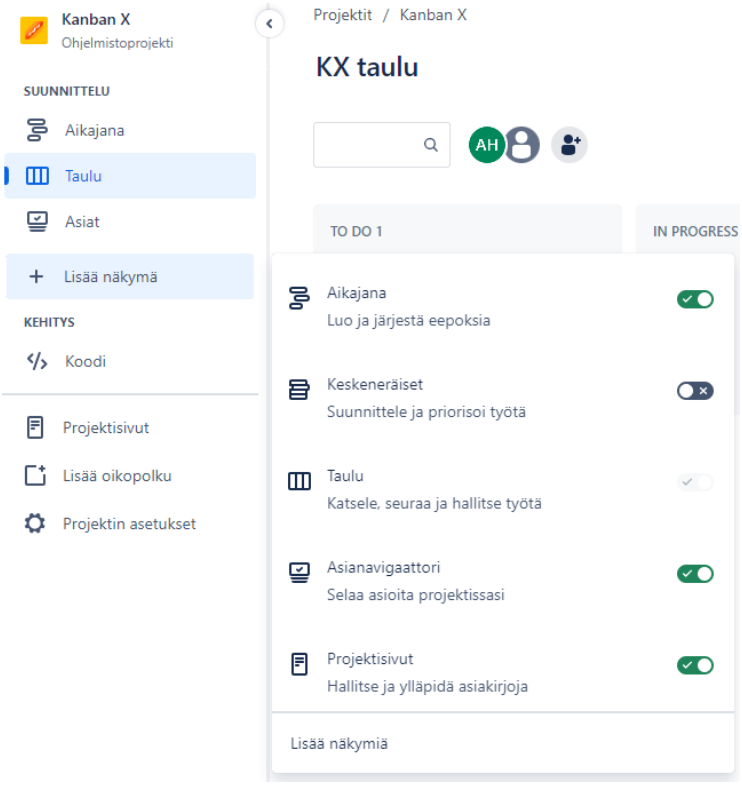
Kanban viitekehyksen yhteydessä yleisesti käytettynä digitaalisena työkaluna mainitaan esimerkiksi Meistertask tai Trello, mutta tässä yhteydessä on johdonmukaisempaa jatkaa esittelyä Jira työkalussa, jossa voidaan luoda projektin Kanban-taulu aiemman Scrum-projektin alle, jolloin tuotantoon viennin jälkeiset keskeneräiset asiat, tekninen velka, on vaivattomasti jäljitettävissä ja käsiteltävissä Kanban-taulussa tai vaihtoehtoisesti luoda täysin uusi oma projekti elinkaaritöille, mutta periaatteena olisi, että kaikki digitaalinen työnjälki jatkuisi läpi tuotteen elinkaaren samassa digitaalisessa työkalussa.

Jirassa ylläpitoprojektia käsitellään nimensä mukaisesti Kanban-taulussa, joka on myös konfiguroitavissa tiimin toimintatapoja myötäileväksi (kuvassa 10 aloitetun Kanban-projektin yleisnäkymä).



Kuva 10 Jirassa uuden ohjelmistoprojektin Kanban-taulun yleisnäkymä

Sivun vasemmasta reunasta löytyy jatkuvan kehityksen eli elinkaaritöiden suunnittelua ja toteutusta tukevat toiminnallisuudet: Aikajana, taulu ja asiat (kuva 11). Lisää näkymä-osiosta saa halutessaan käyttöön muun muassa Raportit ja Keskeneräiset-osiot, mutta oletusarvoisesti ne ovat poistettu käytöstä. (Jira Software Support 2024).



Kuva 11 Jira projektin Kanban-taulun suunnittelun työalusta

Aikajana-osiossa on seurattavissa sanansa mukaisesti aikajanalla asiatyyppien edistymistä tuleviin tuotantoversiopäivityksiin. Jos Keskeneräiset-osio on otettu käyttöön, voidaan tätä kautta tuoda taululle näkyviin tuotteen kehitysjonosta tehtävät asiat tulevaan tuotantoversioon. (Jira Software Support 2024).

Taulu-osiossa tapahtuu varsinainen tehtävien eteneminen sarakkeissa tehtävälista, käynnissä ja valmis. Tehtävälistalle voidaan tuoda lisää asioita Asiat-osiosta tai luoda kokonaan uusia asioita. (Jira Software Support 2024).

Asiat-osiossa näytetään asiatyypit eepos (epic) ja tehtävä (task) eli kyseisessä osiossa käsitellään varsinaisen tuotteen kehitysjonon asiat, jotka ovat mahdollisesti myös periytyneet kehitystiimiltä ensimmäisen tuotantoonviennin jälkeen. Asiatyypit sisältävät oletusarvoisia pakollisia ja ei-pakollisia kuvaus- ja kontekstikenttiä. Esimerkiksi asiatyyppi

tehtävä sisältää oletusarvoisesti kentät: Tila, yhteenveto, käsittelijä, leimat, isä (esimerkiksi epic-taso), liite, linkitetyt asiat. Kuvaus- ja kontekstikentät ovat muokattavissa projektin tarpeiden mukaiseksi eli tarpeettomat kentät ovat poistettavissa ja muita kenttiä lisättävissä. (Jira Software Support 2024).

Työmäärän laajuuden mukaan Jiran kaltaisten sovellusten sijaan voidaan työkaluna käyttää myös tiimihuoneeseen kiinnitettyä Kanban-taulua, jossa tarralaput edustavat varsinaisia tehtäviä, mutta tiimin on tehtävä arvion siitä, minkä työkalun toimintamalli soveltuu parhaiten tiimin tarpeisiin (Juvonen 2018, 24).

2.6 Ketterässä menetelmässä todennettuja havaintoja

Luurila (2023) tutki toimeksiannossaan ketterän menetelmien käyttöönottoa ja sen vaikutusta sidosryhmien keskuudessa. Hän mainitsee tuloksissaan työntekijöiden kokeneen sprinttien helpottavan kokonaisuuden ymmärtämistä ja keskittymistä oikeisiin asioihin sekä tehokkaan kommunikaation tiivistävän yhteistyötä. Luurilan tuloksissa ilmeni kuitenkin myös haasteita muutosjoustavuudessa muun muassa bugikorjauksista aiheutuvan työtaakan ja aikaikkunan suunnittelussa. Lisäksi oli toivottua, että tavoitteiden ja suunnittelun sijaan keskityttäisiin tehokkaammin prioriteettien ja työlistan päivittämiseen, myös päivittäiset palaverit koettiin kuormittavaksi. Luurila kuitenkin toteaa toimivan julkaisuversion taustalla vaikuttavan tehokkaan testausvaiheiden usean iteroinnin.

Kamath (2023) käyttää omassa tutkimustyössään SAFe (scaled agile framework) eli skaalattua ketterää viitekehystä, jota teoriassa yleisellä tasolla jo osin kuvattiinkin, ja tämän vaatimushierarkian perusteella Jira räätälöitiin sille sopivaksi. Testihallintatyökaluna hän mainitsee qTest, jossa suorittaa testien hallintakäytäntöjä ja erinäisiä seurantaraportteja. Kamath korostaa sovittuja yhteisiä käytäntöjä ja perehdytystilaisuuksia, mutta myös sovittujen asioiden dokumentointia kaikkien osapuolen kesken.

Kokko (2013) korostaa tutkimustyössään käytettävien menetelmien sijaan tiimin sujuvan yhteistyön tärkeyttä, koska menetelmät sellaisenaan ei takaa varsinaista projektin onnistumista. Hän pitää tärkeänä viestinnän avoimuutta ja läpinäkyvyyttä. Venäläinen (2018) Kokkon tapaan korostaa vuorovaikutusta, varsinkin asiakkaan ja ohjelmistokehittäjien välillä. Hän ehdottaa tutkimustyössään jatkotutkimusta erityisesti asiakkaan ja kehittäjien väliseen yhteistyöhön.

Hara (2013) on tutkimuksessaan selvittänyt asiakkaan näkökulmaa ketterässä ohjelmistossa ja kolmesta eri näkökulmasta yksi koskee asiakkaan aktiivisuutta ja osallistumista kehittämisen aikana. Hän asettaa asiakkaalle aktiviteetteja, joita tulisi suorittaa projektin aikana kuten projektinhallinta, sisällöstä päättäminen, testaus ja hyväksyntä sekä yhteydenpito ja konsultointi eli toisin sanoen asiakkaan olisi ymmärrettävä oman roolin merkitys ja toimia sen mukaisesti.

Lindvall (2022) korostaa tutkimustyössään, että toteutettavat ominaisuudet tulisi suunnitella valmiiksi etupainotteisesti ennen toteutusvaihetta. Sen lisäksi hän tuo esille yhteistyön ja yhteisymmärryksen tärkeyden sidosryhmien kanssa, koska se edesauttaa, että toiminnallisuuden tulevat hyvin määritellyksi. Sosiaalisten taitojen tärkeyden lisäksi Lindvall mainitsee myös käytettävät menetelmät ja työkalut.

Kaukavuori (2000) tuo tutkimuksessaan esille vaatimusten luokittelun tärkeyden, esimerkiksi priorisoinnin mukaan, ja jossa jokainen vaatimus käsitellään yksilöllisesti ja jotka omaavat joukon perusattribuutteja.

3 VAATIMUSTENHALLINTA

Testauksella on ohjelmiston tuotekehityksessä niin sanotusti viimeinen vaikutusvalta määrittelyihin eli testauksen rooli on avainasemassa vaatimustenhallinnassa joko todentaen ne toimivaksi, ei-toimivaksi tai asettaen niihin muutosvaatimuksien paineita. Kappaleessa 2 pohjustettiin ohjelmistoprojekteissa suosioon noussutta ketterän menetelmän viitekehyksiä sekä esimerkkinä työkalua, jossa ketterää kehitystyötä voidaan edistää. Menetelmään ja työkaluun nivoutuu myös vaatimustenhallinta, joka on riippuvainen kyseisistä ympäristöistä, jotka asettavat omanlaisensa haasteensa vaatimustenhallintaan. Tässä kappaleessa selvitetään mitä ulottuvuuksia vaatimustenhallinta sisältää, mitä tavoitteita, odotuksia ja toimenpiteitä se asettaa myös testaukselle. Kappaleissa myös otteita tutkimustöissä esiin tulleista havainnoista ja haasteista koskien vaatimustenhallintaa ja jäljitettävyyttä.

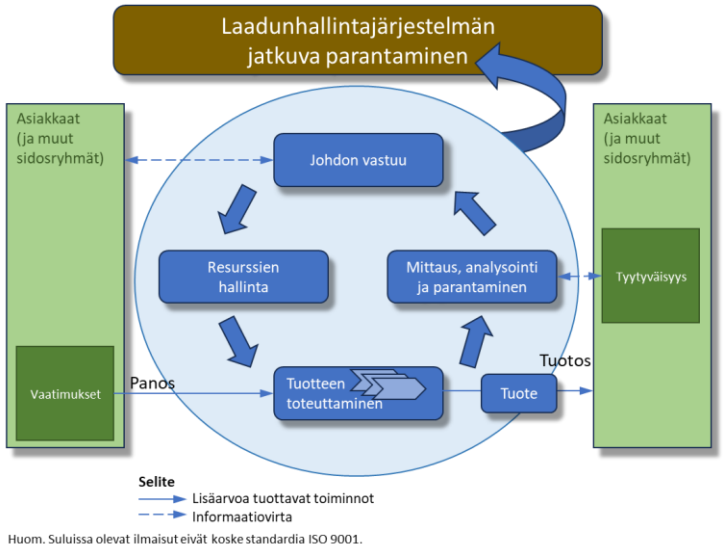
3.1 Laadunhallintastandardit ISO 9000

Vaatimuksiin kohdistuu laadun ongelmien ja riskien hallintakeinoja, joita ovat muun muassa tuotteen katselmoinnit, tarkastukset, arvioinnit, testaus sekä verifiointi ja validointi. Lisäksi hallintakeinoiksi luetaan prosessien, käytettyjen työkalujen ja menetelmien kehittäminen, osaamisen kehittäminen sekä erilaisten laatujärjestelmien käyttöönotto, joita ohjeistaa erilaiset standardit.

Laatujärjestelmän laadunhallintastandardi ISO 9000 sisältää kahdeksan periaatetta: Korostettu vahvaa asiakaslähtöisyyttä, ylimmän johdon aktiivinen osallistuvuus ja henkilöstön sitouttaminen, prosessilähtöisyys, järjestelmällinen johtamistapa, sitoutuminen jatkuvaan kehittämiseen, päätöksenteko perustetaan tosiasioihin sekä tasavertainen toimittajasuhde. Asiakastyytyväisyyttä edistävään ISO 9000 (perusteet ja termistö) laadunhallintajärjestelmän standardiperheeseen kuuluu lisäksi ISO 9001 (vaatimukset) sekä ISO 9004 (organisaation laatu. Ohjeita jatkuvan menestyksen saavuttamiseen). (ISO 9000:2015; ISO 9001:2025; ISO 9004:2018; SFS-EN ISO 9000 2005, 8.)

Kuvassa 12 on havainnollistettu prosesseihin perustuvan laadunhallintajärjestelmän toimintamallia, joka sisältää ISO 9000 -sarjan kaikki elementit ja soveltamiseen kansainvälinen standardi kannustaa. Standardin kannustama toimintamalli jättää organisaatioille

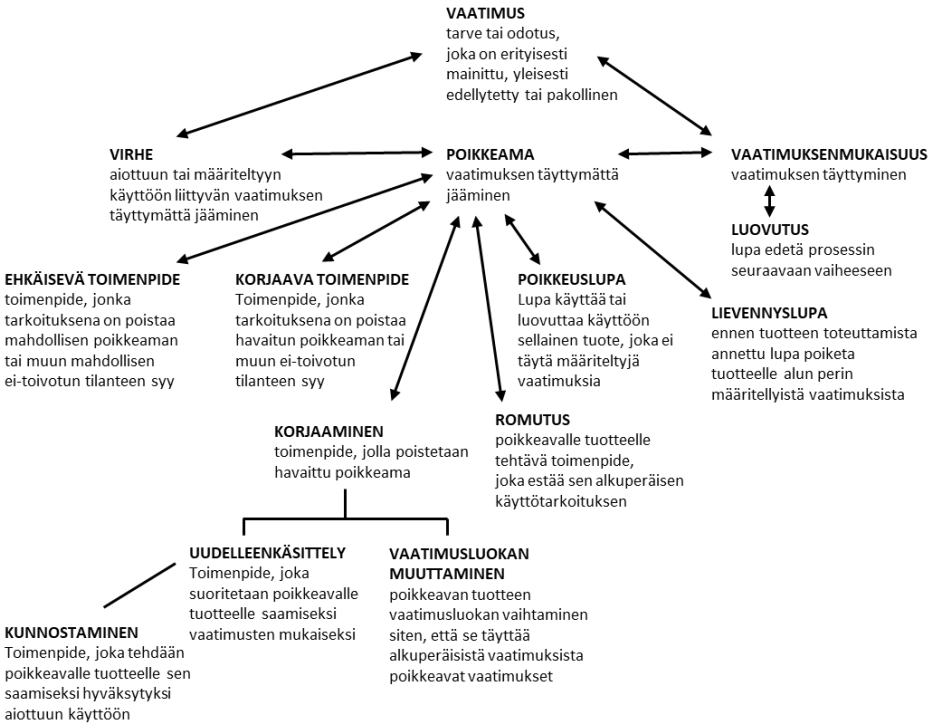
joustavuutta suunnitella ja rakentaa yksityiskohtaisempia prosesseja. (SFS-EN ISO 9000 2005, 12)



Kuva 12 Prosesseihin perustuvan laadunhallintajärjestelmän toimintamalli (SFS-EN ISO 9000 2005, 14).

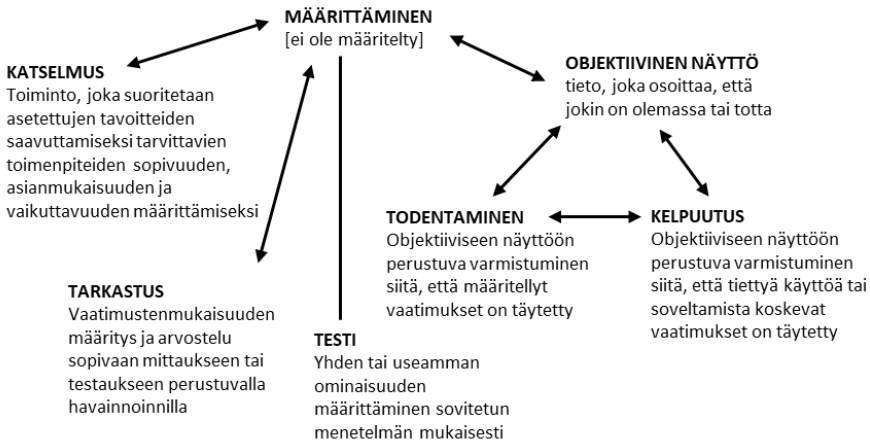
Laatustandardit ohjaavat tuotantoa ja tuotannon laatua, joka käsittää seuraavat osat: Laatupolitiikka, laatujärjestelmän dokumentointi ja sertifiointi, laatusuunnitelmat sekä laadunvarmistus. Tuotannon laatuun liittyvät ongelmat ja riskit, jonka laatujärjestelmä pyrkii ennaltaehkäisemään, minimoimaan tai ratkaisemaan, ovat: Budjetin ylitys, aikataulun viivästyminen (voi vaikuttaa lisäksi tulevien projektien aikatauluihin), tuotetta ei saada markkinoille, tuotannossa olevan tuotteen viat, hallinnoinnin lisäys, sisällön asiavirheet, ylläpito-ongelmat, sisäiset kustannukset (vaikutus kilpailukykyyn), hyödyttömät palaverit, tuplatyö, päivitykset, asiakaspalautteet, lisääntyneen asiakastuen tarve, asiakaspalautukset ja maineen menetys. (Haikala & Mäkijärvi 2001, 43–45.)

ISO 9000 kuvaa laadun vaatimuksiin käytettävää sanastoa ja kuvan 13 käsitekaavio havainnollistaa kyseistä termistöä, mitä vaatimushallinnan yhteydessä suositellaan käyttämään. Tässä tutkimuksessa kaikkien standardien varsinaisten termien sisältöön ei syvennytä sen jokaisella saralla, mutta on hyvä tiedostaa näiden olemassaolo ja taustavaikutus sidosryhmien keskuudessa vallitsevaan yhteisymmärrykseen.



Kuva 13 Vaatimuksien yhteydessä käytettävä termistö (SFS-EN ISO 9000 2005, 58)

Vastaavasti määrittelyn laadunhallinnan tueksi ISO 9000 tarjoaa kuvan 14 mukaista käsitekaaviota, joka on sovellettavissa myös esimerkiksi testitapausten laadinnassa eli standardi tukee näin ollen laadulliseen tekemiseen projektin joka osa-alueella.

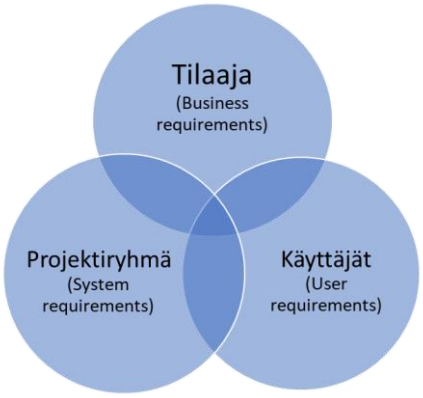


Kuva 14 Vaatimusmäärityksien tarkastuksessa käytettävät termit (SFS-EN ISO 9000 2005, 60)

3.2 Vaatimusmäärittely

Testauksen kohteena on vaatimusmäärittelyt, jotka muodostuvat liiketoiminnallisesta tavoitteesta eli liiketoimintavaatimuksista (business requirements), jota tilaajan ohjelmisto tulee tukemaan ja toteuttamaan. Liiketoimintavaatimuksista rakentuu ja tarkentuu sidosryhmävaatimukset (stakeholder requirements) sekä käyttäjävaatimukset (user requirements). Järjestelmän käyttäjäryhmä voi olla tilaajan oma organisaatio ja/tai sen asiakas tai tilaajaorganisaatiosta täysin ulkopuolinen käyttäjäryhmä. Määrittely on toimenpide, joka tavanomaisesti suoritetaan ennen kuin tilaaja on yhteydessä ohjelmiston toimittajaan tai vaihtoehtoisesti tarvittavat määrittelyt voidaan kartoittaa yhdessä toimittaja kanssa ennen virallista vaatimusmäärittelydokumentoinnin kirjaamista, tarkastamista ja hyväksymistä. Tavoitteena on liiketoiminnallisten tavoitteiden ja määrittelyjen lisäksi, että kaikille osapuolille muodostuu yhtenevä käsitys tarvittavan tuotteen toiminnasta. (Terho ym. 2008, 14; Pelin 2011, 198, 202; Ruuska 2012, 162–163.)

Vaatimusmäärittelyt laativat luonnolliset henkilöt (kuva 15), jotka kohdistavat erinäisiä tavoitteita ja vaatimuksia ohjelmistoprojektille koko projektin elinkaaren ajan. Liiketoiminnallisista tavoitteista kootut vaatimuksista tulevat projektin kehitysvaiheessa kohtaamaan myös projektiryhmän (system requirements) vaikutukset.



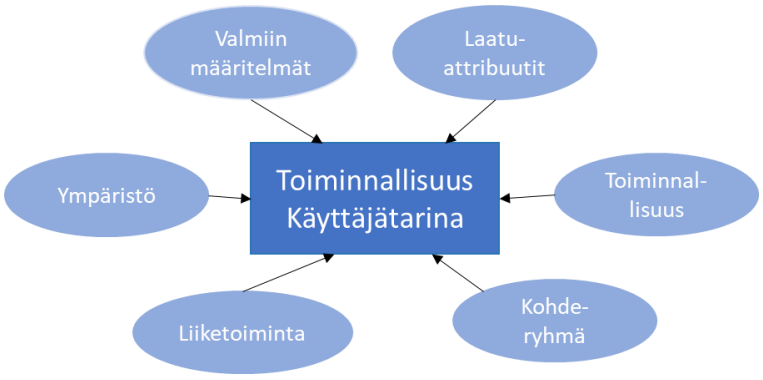
Kuva 15 Projektiin kohdistuvat vaatimukset (Ruuska 2012, 280).

Projektin päätavoitteet yleisellä tasolla ovat: Sisällölliset, laadulliset ja toteutukselliset tavoitteet, taloudelliset tavoitteet sekä ajalliset tavoitteet. Tavoitteiden painotukset luonnollisten henkilöiden muodostamien sidosryhmien keskuudessa voidaan esittää taulukossa 1, jossa kunkin ryhmän projektiin kohdistuvat odotukset voidaan priorisoida. (Ruuska, 2012, 274).

Taulukko 1 Sidosryhmien projektiin kohdistuvien odotusten prioriteetit (Ruuska 2012, 281).

Prioriteetti	Tilaaja	Käyttäjät	Projekti-ryhmä
1	budjetti	laatu	aikataulu
2	aikataulu	aikataulu	laatu
3	laatu	budjetti	budjetti

Luonnollisten henkilöiden asettamien vaatimusten lisäksi vaatimusmäärittelyissä on huomioitava ympäristövaatimukset, joita ovat muun muassa laki, systeemi ja tulevaisuus. Vaatimuksia on niin ikään paljon, joten todennäköiset ristiriidat eri sidosryhmien vaatimusten välillä on tunnistettava mahdollisimman aikaisessa vaiheessa. Hyvä vaatimusmäärittely on lisäksi selkeä, ymmärrettävä ja täsmällinen eikä rajoita suunnittelua liikaa. Vaatimusmäärittelyn prosessin lopputuloksena rakentuu tärkein dokumentti, toiminnallinen määrittely (kuvassa 16 on havainnollistettu osin erilaisia toiminnallisuuteen vaikuttavia vaatimuksia), joka sisältää lisäksi vaatimukset täyttävän järjestelmän kuvauksen. (Pelin 2011, 66, 199).



Kuva 16 Toiminnallisuuteen kohdistuvia vaatimuksia

Lisäksi vaatimusmäärittelyihin sisältyy seuraavanlaisia vaadittavia dokumentaatioita kuten käyttäjätarinat, käyttöliittymäkuvaus, kohdealuekuvaus, arkkitehtuurikuvaus, tietokantakuvaus, käyttövaltuuskuvaus sekä rajapintakuvaus. Käyttäjätarina määrittää lyhyesti tekstimuodossa yhden toiminnallisuuden vaatimusta, joka on tunnistettu käyttäjän näkökulmasta. Käyttöliittymäkuvaus voidaan mallintaa paperille, käyttää siihen soveltuvaa mallinnusvälinettä tai luoda käyttöliittymästä protoversio sillä ohjelmointikielellä, jolla varsinainen toteutus tehdään. Tässä yhteydessä on huomioitava saatavuusdirektiivien

asettamat vaatimukset. Käyttövaltuudet asettavat vaatimukset autentikointiin, roolirajauksiin ja niihin rajoittuviin tehtäviin. Arkkitehtuurissa kuvataan toiminta- ja tietoarkkitehtuuri sekä tietojärjestelmä- sekä teknologia-arkkitehtuuri, joille vastaavasti sovellus-, tietokanta- sekä tietovarastointiarkkitehtuuri. Rajapinnat havainnollistavat mahdolliset ulkoisien sidosryhmien vaikuttamisen projektin kulkuun ja määrittelyyn. (Atlassian 2024).

Määrittelyt voidaan kuvata skaalaustasojen mukaisesti eeppisen (epic), toiminnallisen (feature) ja käyttäjätarina (user story) muodoissa, joista eepos hallinnoi suuria työkokonaisuuksia kuten joukko toiminnallisuuksia tai suoraan joukko käyttäjätarinoita. Käyttäjätarinat ovat lyhyitä vaatimuksia, jotka on kirjoitettu loppukäyttäjän näkökulmasta, kun vastaavasti eepokset kuvaavat yleisemmällä tasolla käyttäjätarinoiden vaatimuksia. Ajanjaksolla skaalaukset eroavat toisistaan myös aikajanalla, jossa käyttäjätarinat toteutetaan sprintin aikana, kun eeposten valmistuminen voi kattaa useammankin sprintin, riippuen niiden alatasolla valmistuvista käyttäjätarinoista. (Atlassian 2024).

Jira työkalussa on valmis lomakepohja, jolla eepos voidaan määritellä. Lomakepohjassa annetaan eeppinen nimi eli lyhyt kuvaus, jota käytetään tunnisteena tarinoissa, jotka tulevat kuulumaan kyseiseen eepokseen sekä eeppinen yhteenveto. Eepos jakaantuu käyttäjätarinoiksi, jotka yksittäisenä toteutuksen suuruisina pystyä toteuttamaan sprintin aikana. Käyttäjätarinan tulee sisältää asiakkaan näkökulmasta hyväksymiskriteerit eli ehdot, jolloin ominaisuus on valmis tuotantoon. Näiden lisäksi käyttäjätarinan tulee tiimin näkökulmasta sisältää toteutukselle tärkeät DoR kriteerit, joiden ehtojen täyttyessä ominaisuus on valmis otettavaksi sprintin työlistalle sekä DoD kriteerit, joiden ehtojen täyttyessä ominaisuus on valmis lisättäväksi ohjelmistoon. (Atlassian 2024).

3.3 Käyttäjävaatimukset (user requirements)

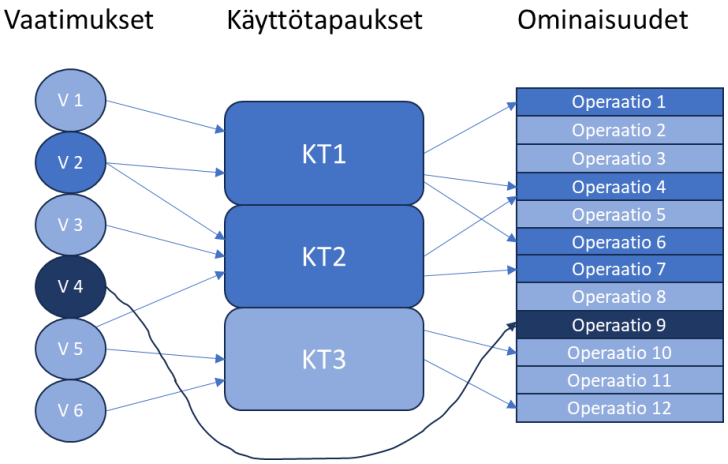
Tuotantoon viennin jälkeen järjestelmän käyttäjäryhmä toimii laadun mittarina muun muassa asiakaspalautteen ja käyttäjätuen tarpeen muodoissa, jotka testauksen on huomioitava jo ennen tuotantoon vientiä. Järjestelmän käyttäjäryhmä muodostaa ohjelmistolle käyttäjävaatimukset (user requirements), jossa käyttäjäryhmä voi olla sama taho kuin tilaaja tai tilaajalla voi olla asiakkaana organisaation ulkopuolinen käyttäjäryhmä. Toimittajan ja tilaajan välisestä kommunikaatiosta voidaan myös puhua asiakassuhteesta, koska tilaaja määrittelee ja päättää minkälaisen tuotteen haluaa, hyväksyy tai hylkää lopputuloksen ja ennen kaikkea päättää budjetista. Tässä käyttäjävaatimuksilla kuitenkin

tarkoitetaan ja rajataan varsinaisen järjestelmän käyttäjäryhmään ja -näkökulmaan. Testauksen näkökulmasta on näin ollen merkitystä, onko tilaaja sama kuin käyttäjäryhmä vai täysin ulkopuolinen käyttäjäryhmä. Jos tilaaja on sama kuin käyttäjäryhmä, ohjelmiston laadun kannalta arvokas palaute saadaan jo ennen tuotantoon vientiä. (Ruuska 2012, 162–163.)

Kappaleen 3.2 taulukossa 2 ohjelmiston käyttäjä asettaa odotuksissaan korkeimman prioriteetin järjestelmän käytettävyyteen, vähemmän projektin aikataululle (riippuen siitä, onko koettu käyttäjien taholta järjestelmälle akuutti tarve), ja viimeisenä prioriteettina projektiin kuluva budjetti. Epäsuorasti projektin budjetti voi kuitenkin vaikuttaa myös loppukäyttäjän kustannuksiin, jos tuleva järjestelmä on käyttäjälleen maksullinen. (Ruuska 2012, 280.)

Ohjelmiston käyttäjäryhmän tarpeen kartoittaminen on kiinni projektin lähtökohdista, onko tuleva järjestelmä täysin uusi vai korvaava järjestelmä, onko käyttäjämäärä yhdestä tuhansiin. Korvaavan järjestelmän yhteydessä tuotantoversiosta saadut palautteet ovat voineet nostaa tarpeen luoda korvaava järjestelmä tai uusi palvelu on edellyttämässä uutta järjestelmää. Loppukäyttäjien kuuleminen on kuitenkin tärkeää, ei ainoastaan yleisten toiminnallisuuksien suhteen, mutta myös järjestelmävaatimuksissa on aiheellista huomioida loppukäyttäjien suhtautuminen jo ennen tuotantoon vientiä. (Juvonen 2018, 58–59.)

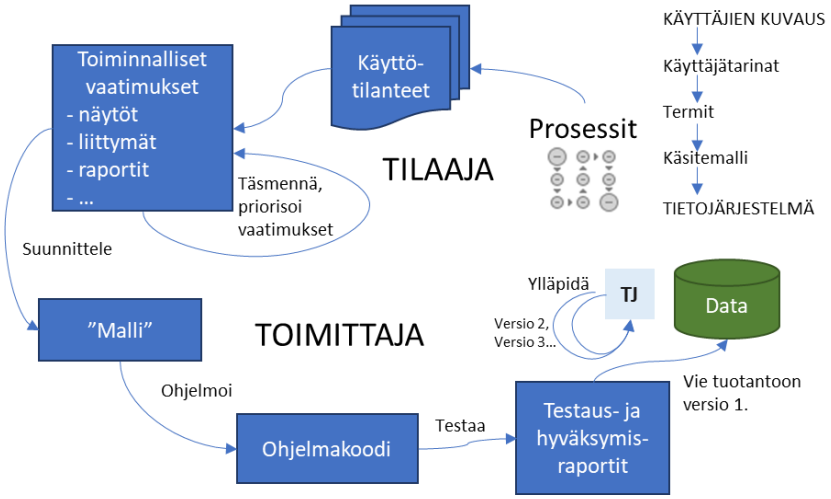
Käyttäjävaatimuksista voidaan kartoittaa ohjelmiston käyttötapaukset (use cases) ja käyttäjäroolit, joilla kuvataan järjestelmän toiminnallisuutta. Käyttötapauksen tulee sisältää seuraavia ominaisuuksia: Ymmärrettävyys eli kuvaus mahdollisimman ymmärrettävä, kuvaa asiakasvaatimusta ottamatta kantaa varsinaiseen toteutukseen, testattavuus eli järjestelmätestaus perustuu käyttötapauksiin, koko eli ei sisällä laajaa kuvausta sekä sopiva tarkkuus, mutta ei liian yksityiskohtainen eli kirjataan tärkeimmät asiat. Käyttötapauksilla kartoitetaan asiakasvaatimuksia, joista muodostetaan myös ohjelmistovaatimukset. Kuvassa 17 on havaittavissa, että kaikkia ohjelmiston ominaisuuksia ei välttämättä ole kytketty käyttötapauksiin, vastaavasti yksi käyttötapaus voi kattaa useita ominaisuuksia. Järjestelmätestauksen testitapauksia suunnitellessa ja laadittaessa tämä on yksi tärkeistä huomioitavista seikoista testauskattavuuden ja sen osoitettavuuden kannalta. (Haikala & Mäkijärvi 2001, 142–145).



Kuva 17 Käyttäjävaatimusten muotoutuminen järjestelmän ominaisuuksiksi (Haikala & Mäkijärvi 2001, 145).

3.4 Järjestelmävaatimukset (system requirements)

Liiketoiminnalliset vaatimukset ja käyttäjävaatimukset edustavat niin sanotusti käytännön tasolla kuvattuja vaatimusmäärittelyjä sekä yleisiä järjestelmävaatimuksia, joista analysoimalla johdetaan varsinaiset tekniset ja ohjelmistovaatimukset. Teknistä toteutusta sanelee tietynlaiset reunaehdot ja standardin mukaiset laatuvaatimukset (näistä enemmän kappaleessa 5.), jotka on otettava huomioon määritellessä järjestelmävaatimuksia. Määrittelyn tuloksena tuotetaan dokumentti, josta voidaan käyttää termiä toiminnallinen määrittely. Kuvassa 18 on havainnollistettu järjestelmävaatimusten määrittely- ja hallintaprosessia, jossa voidaan, puuttuvista nuolista huolimatta, toistaa tiettyjä prosessin osa-alueita useita kertoja ketterän menetelmän sprintin mukaisesti eli vaatimusmäärittelyt eivät ole kiveen hakattuja, vaan puhutaan näiden yhteydessä myös muutosvaatimuksista ja muutosten hallinnasta, joka muodostaa oman prosessinsa. (Haikala & Mäkijärvi 2001, 26–27; Forselius 2013, 29–31.)



Kuva 18 Järjestelmävaatimusten määrittely- ja hallintaprosessi (mukaillen Forselius 2013, 31).

Kuvassa 18 kuvataan yleisellä tasolla järjestelmävaatimusten prosessia koko sen elinkaaren ajalta, kuvassa 19 on havainnollistettu vastaavasti varsinaisen järjestelmävaatimuksen tunnistamisen prosessi, joka kohdistuisi kuvassa 18 olevaan toiminnallisien vaatimuksien tunnistamiseen.

Kuva 19 Järjestelmävaatimusten tunnistamisen prosessi (mukaillen Terho ym. 2008, 16).

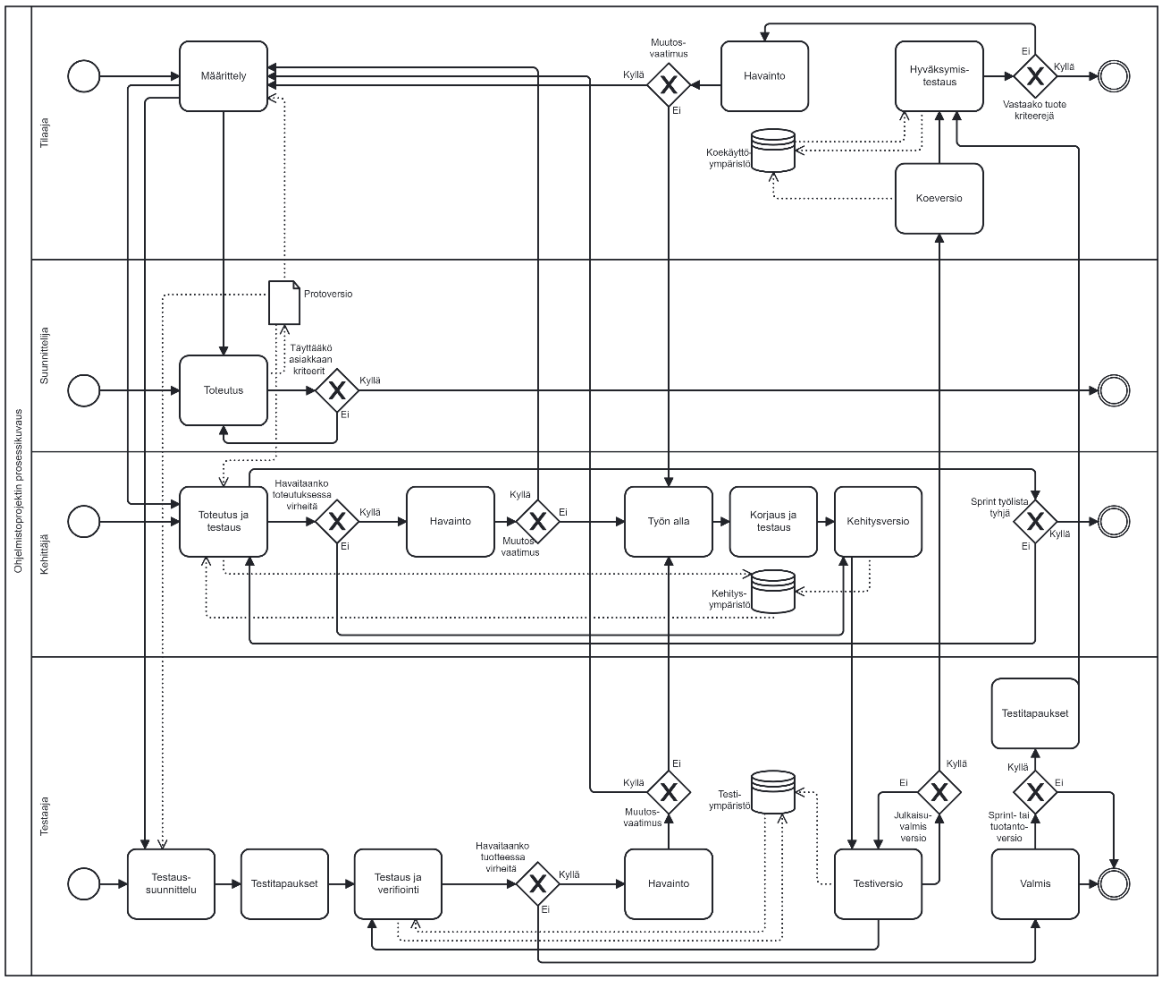
Järjestelmävaatimusten yhteydessä puhutaan toiminnallisista vaatimuksista (functional requirements) sekä ei-toiminnallisista (non-functional requirements) vaatimuksista, jotka määrittävät vastaavasti laatuattribuutit. Ei-toiminnalliset vaatimukset noudattavat pitkälti ISO/IEC 25010 standardin sisältöä kuten suoritustehoa, vasteaikaa ja käytettävyyttä, kun taas toiminnalliset vaatimukset on kuvattu selkeästi, varsinaisia ohjelmiston ominaisuuksia, joista muodostuu käyttäjätarinoita. (Haikala & Mäkijärvi 2001, 27.)

3.5 Muutosvaatimusten hallinta ja jäljitettävyys

Esitutkimus ja määrittelyvaiheessa kartoitetaan asiakasvaatimukset, joista johdetaan ohjelmistovaatimukset, mutta muutosvaatimuksiin on varauduttava koko ohjelmiston elinkaaren ajan kehitystiimistä viimeiseen tuotantoversioon asti. Syy muutosvaatimusten taustalla on useita: Projektin alkuvaiheessa laadittuja asiakasvaatimuksia ei ole ymmärretty oikein tai on jäänyt huomaamatta, ohjelmiston toimintaympäristön muutokset projektin aikana, määritelty asiakasvaatimus on myöhemmin todettu käyttökelvottomaksi, aikataulullisista syistä vähemmällä prioriteetilla olevat ominaisuudet jätetään toteuttamatta, markkinatilanteiden muutokset sekä projektin alkuvaiheessa päätetyt teknologiavalinnat osoittautuvat myöhemmässä markkina-asemassa huonoiksi valinnoiksi. Vaatimustenhallinnassa muutostenhallinnan suuri ja vaikuttava rooli edellyttää sovittuja menetelmiä kuten hyväksymismenetelmää sekä muutoksen vaikuttavuuden arviointia eli vaatimuksen jäljitettävyyttä. (Haikala & Mäkijärvi 2001, 86–87.)

Vaatimusten jäljitettävyydellä (traceability) tarkoitetaan ominaisuutta, jolla voidaan osoittaa verifioinnin tuloksella toiminnallisen vaatimuksen toteuttavan tietyn asiakasvaatimuksen. Vaatimusmäärittelyjen jäljitettävyyden merkitys korostuu, kun ohjelmistoon kohdistuu muutoksia sekä regressiotestauksessa on ilmennyt havaintoja, huomioiden laatuattribuutit luotettavuus, tehokkuus, turvallisuus, ylläpidettävyys ja koko, jotta kokonaisuus pysyy ehjänä. Tällöin kaikkia muutosta koskeviin dokumentointeihin pitää kyetä palaamaan ja tekemään tarvittavat kirjaukset ja toimenpiteet. Tilaajan kannalta jäljitettävyys tapahtuu vaatimusmäärittelyjen tasoilla eeppisestä käyttäjätarinaan, kun taas kehittäjän kannalta jäljitettävyyden tarve tapahtuu käyttäjätarinoiden lisäksi release-tasolla. Vastaavasti testauksen näkökulmasta jäljitettävyysketju testitapauksista käyttäjätarinoihin on myös release-tasolla oleellista tietää, mitä ja milloin on testattu toimivaksi ja ei-toimivaksi, mutta käyttäjätarinoiden testaukseen liitetyt kirjalliset kommentit ja havainnot ovat myös arvokasta tietoa. Taaksepäin jäljitettävyydellä saadaan kartoitettua niin toteutetut kuin toteuttamatta jäävät asiakasvaatimukset, mutta myös osoitettua testauksen suorittama testauskattavuus ja edistyminen. (Haikala & Mäkijärvi 2001, 85–87; Kasurinen 2013, 137.)

Tähänastisen tutkimuksen perusteella voidaan vaatimustenhallinnasta kuvata kuvan 20 mukainen prosessi, joka on luotu BPMN-piirto-ohjelmalla. Tarkoitus on kuvantaa vaatimusmäärittelyjen kulku prosessikuvaustasolla, josta on havaittavissa muutosvaatimusten lisäksi myös mahdollisten uusien vaatimusten synty- ja käsittelykohdat.



Kuva 20 Vaatimustenhallinnan prosessikuvaus

Prosessikuvaus kokonaisuudessaan helpottaa tunnistamaan prosessin kannalta kriittiset asiat, esittää asioiden väliset riippuvuudet, tukee kokonaiskuvan ymmärtämistä sekä eri roolien tavoitteita ja tehtäviä, edistää prosessin keskuudessa toimivien työntekijöiden yhteistyötä sekä joustaa tilanteen vaatimusten mukaan (Laamanen, 2001 75–76). Kuvan 30 prosessikuvauksessa määrittelyihin pohjautuvan käyttöliittymän protoversio ei ole kaikissa asiakasprojekteissa käytettävissä, joten kyseinen prosessi on luettavissa myös siten, että suunnittelijan osuus voidaan kokonaan peittää. Käytettävyystestauksen yhteydessä protoversion merkittävyys korostuu, jos toteutettavasta ohjelmistosta ei ole aiempaa referenssiä käyttäjäkokemuksineen, mutta protoversio ei kuitenkaan ole välttämätön (Haikala & Mäkijärvi 2001, 273). Kuvaus ei ainoastaan havainnollista vaatimusten elämää, vaan osoittaa myös raportoinnille suotuisat mittauspisteet.

3.6 Vaatimusmäärittelyssä todennettuja havaintoja

Paakkari (2022) korostaa tutkimuksessaan vaatimusten määrittelyn ja suunnittelun merkittävyyttä koko tuotekehitysprosessin ajan ja lisäksi havainnoi, että prosessitasolla vaatimusmäärittelyjen tulisi edetä kehitysprosessin edellä. Määrittelyprosessin hän jakaa viiteen osa-alueeseen: Käynnistäminen, kerääminen, vaatimusten tarkentaminen, valmis vaatimusmäärittely sekä sudenkuopat ja erityistilanteet. Paakkari myötäilee teoriaa siltäkin osin, että määrittelyjä ei saisi alussa kuvata liian yksityiskohtaisesti, vaan ne tulisi jättää yleisemmälle tasolle, jolloin myös muutosvaatimukset ovat helpommin hallittavissa. Hän huomioi myös projektin koon eli suurempi projekti edellyttää hallitumpaa otetta ja tämän yhteydessä korostaa ratkaisuksi koulutusta ja saatavilla olevaa ohjeistusta.

Hou (2021) pohtii tutkimustyössään vaatimusmäärittelyjen kartoitusta, jossa hän ensin kartoittaa järjestelmän tärkeyttä tilaajalle, tämän määrittelyn jälkeen hän tekee vertailun olemassa olevan järjestelmän ominaisuuksista ja tunnistaa vaatimuksia etukäteen sekä suorittaa lopuksi vaatimusmäärittelyn kyselyn avulla. Hän vetoaa kirjoituksessaan myös muuttuvaan ympäristöön, vaikka tilaaja ei kokisi sitä tarpeelliseksi. Nykyhetkellä järjestelmä voi palvellakin moitteettomasti, mutta on kuitenkin varauduttava tulevaisuudessa muutoksiin.

Hara (2013) on tarkastellut tutkimuksessaan asiakkaan näkökulmaa ketterässä ohjelmistoprojektissa, jonka on jakanut kolmeen osaa: Sopimuksen laatimiseen, arvon määrittämiseen sekä osallistumiseen kehittämisen aikana. Sopimusnäkökulmassa hän nostaa esille henkilöstön sitoutumisen, hintaperusteen ja aikasidonnaisuuden. Houn lisäksi Haran mainitsema arvon määrittämisen tärkeys on huomioitava muun muassa siten, että sen on tuotettava kustannuksia enemmän arvoa, mutta ongelmaksi muodostuu arvon abstraktinen luonne, joka on vaikeasti mitattavissa sekä tästäkin syystä juontuva puutteelliset määritelmät. Haran mainitsemasta kolmesta asiakkaan näkökulmasta sopimuksen laatimisen tärkeyttä tukee Luojus (2020) tutkimus, joka keskittyy vaatimusmäärittelyn kartoittamiseen ja dokumentointiin, jota myös käytetään tarjouspyynnön yhteydessä.

Salmi (2020) tuo esille tutkimuksessaan vaikeutta kuvata vaatimusmäärittelyn prosessia, koska vaatimusmäärittelyt elävät projektin mukana koko sen elinkaaren ajan ja näin ollen korostaakin oman osaamisen mukauttamista muuttuvassa ympäristössä, jolloin oleel-

lista olisi ymmärtää kehitysmallien toimintaperiaatteet ja käytännöt. Saarinen (2016) vastaavasti korostaa asiakaslähtöisemmän vaatimushallinnan prosessin määrittämisen ja kuvantamisen tärkeyttä, joka huomioisi vaatimusten seurannan lisäksi myös niiden jäljitettävyyttä käyttäjätarinoiden tasolle asti. Hän tuo tutkimuksessaan esille myös ongelmaa, että vaatimuksia ei ole osattu tulkita alusta alkaen oikein, joten ensisijaiseksi tavoitteeksi asetettiin asiakkaan ymmärtäminen ja laadukkaat käyttäjätarinat. Lisäksi hän kokee tuotantoversioiden myöhästymiseen johtaneita syitä ongelmalliseksi selvittää. mitkä vaikuttavat tekijät olivat myöhästymisen taustalla.

3.7 Vaatimusten jäljitettävyydessä todennettuja havaintoja

Kujala (2000) on työssään tutkinut jäljitettävyyttä ja mainitsee jäljitettävyyden perustekniikoiksi: Jäljitettävyystaulukot ja -luettelot, viitteet sekä automatisoidut jäljitettävyyslinkit. Kyseisissä tekniikoissa huomioidaan myös vaatimusten riippuvuussuhteen, jotka asettavat myös itse vaatimuksiin ja prosessiin tietynlaista ihannepiirrettä: Vaatimus tulisi kirjata vain kerran, muutosvaatimuksessa muutos tehdään vain yhteen kohtaan sekä vaatimusprosessi itsessään ei saisi olla liian raskas. Nämä ominaisuudet helpottaisivat jäljitettävyyttä muutoksista riippumatta ja testauksen osalta saadaan realistinen tilanne testauksen kattavuudesta testatuista ja testaamattomista vaatimuksista.

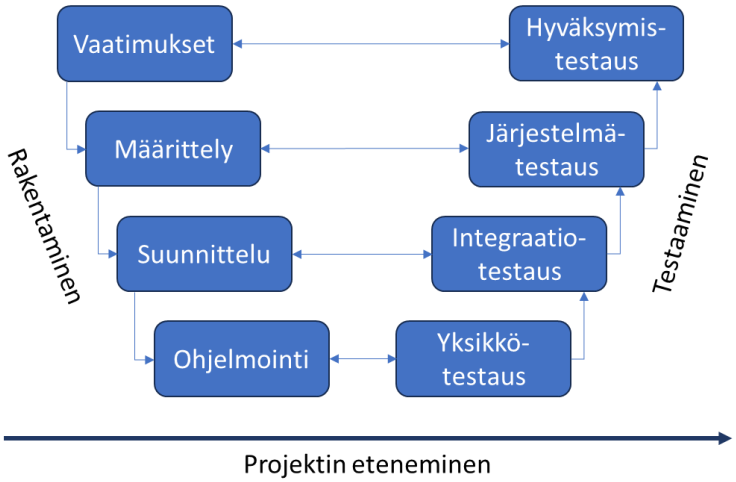
Hokkanen (2001), tutkiessaan vaatimusten jäljitettävyyttä, on korostanut kaksisuuntaisen jäljitettävyyden merkityksen lisäksi myös vaatimusten laatua. Jäljitettävyys vaikeutuu, jos vaatimukset ovat puutteellisia ja niiden kulkuketju epäselvää, jolla on suora vaikutus myös testaukseen: Vaikeuttaa testitapausten uudelleenkäyttöä, jäljittämään vikaa ja ongelmanratkaisua sekä varmistamaan versiossa toteutetut ja vielä toteutumattomat ominaisuudet. Hokkanen huomauttaa, että toimiva jäljitettävyys edellyttää tiimin vastuuta luoda ja ylläpitää dokumentointia jäljitettävyyttä mahdollistavalla tasolla, joka edellyttää tietynlaisia ominaisuuksia myös työkalulta tätä suorittaa.

Dawoud (2014) tutkimus perehtyy vaatimusten jäljitettävyyteen, jossa toistuu muiden tutkimusten mukaisien haasteiden lisäksi ei-toiminnallisten vaatimusten jäljitettävyyttä tiettyyn moduuliin, koska esimerkkinä MBT on suunniteltu vain toiminnallisten vaatimusten jäljittämiseen. Hän esittää jatkotutkimusehdotuksena selvittää, kuinka pitkälle vaatimuksia voidaan jäljittää monimutkaisissa järjestelmissä. Lopuksi Dawoud tuo esille tiimin erityistaitojen ja koulutuksen tarvetta.

Tikka (2015) on työssään tutkinut käytettävien jäljitettävyysmallien yhteydessä vaatimusten kaksisuuntaista jäljitettävyyttä osana vaatimustenhallintaa, jossa korostuu prosessin läpinäkyvyys, oikeanlainen ylläpito sekä tiimiläisten sitoutuminen projektiin. Haasteiksi Tikka mainitsee, että vaatimussuunnittelua pidetään jopa käsittämättömänä tehtävänä ja suuri vaiva järjestää ja ylläpitää jäljitettävyyssuhteita. Lisäksi hän mainitsee, että tämä edellyttää työkalujen vaivattomuutta ja helppoutta käyttää sekä ennen kaikkea jäljitettävyyden hyödyt on kompensoitava kustannuksia.

4 OHJELMISTOTESTAUS

Vaatimusten todentaminen toimivaksi ja toiminta vaatimusten mukaiseksi voidaan osoittaa ainoastaan testauksen avulla, joten testaus on avainasemassa vaatimustenhallinnassa. Kappaleessa käsitellään kuvan 21 mukaisen V-mallin testaustasoja ja -vaiheita osana vaatimustenhallintaa, mutta myös niiden isäksi testauksen standardeja, jotka luovat laadullisen lähestymistavan suorittaa ohjelmistotestausta.



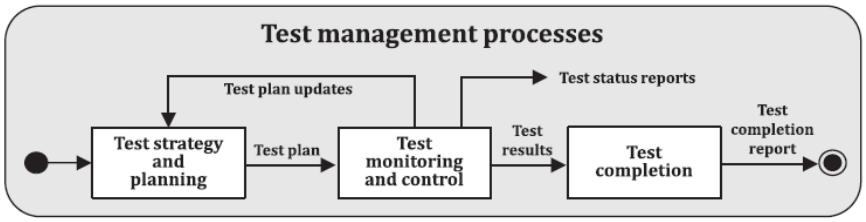
Kuva 21 Testauksen V-malli (Kasurinen 2013, 51).

Testaus jakaantuu kehitysympäristössä tapahtuvan yksikkö- ja integrointitestauksen lisäksi kehitysprosessin rinnalla suoritettaviin, omissa testiympäristöissä toteutettaviin, järjestelmä- ja hyväksymistestaukseen. Kyseisiä testauksen muotoja pidetään itsenäisinä testausprosesseina, jotka on käynnistettävä toiminnallisten määrittelyjen yhteydessä omalla testisuunnitelmalla, budjetilla sekä testiympäristöllä. Laadunvarmistusta ja -valvontaa suoritetaan lisäksi sille tärkeällä testauksen osa-alueella, regressiotestauksella. (TMap 2016, 171–172.)

4.1 Testausstandardi ISO/IEC/IEEE 29119

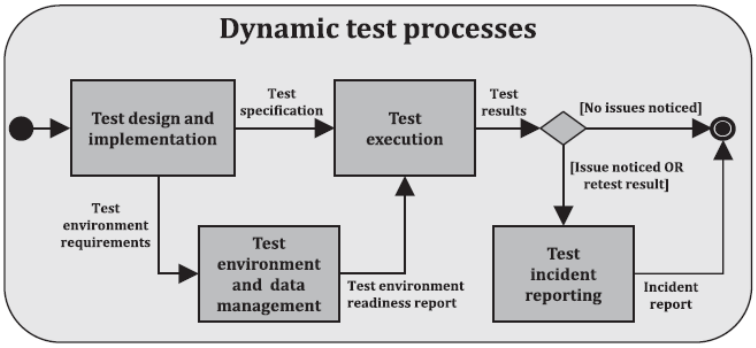
Tässä kappaleessa esitellään lyhyesti kansainvälinen ohjelmistotestaukselle hyväksytty standardijoukko, jotka tarjoavat testauksen saralle korkealaatuista lähestymistapaa. ISO 29119 standardiperheeseen kuuluu kaikkiaan 14 osaa, joista viisi ensimmäistä ISO/IEC 29119-1 sisältää testauksessa käytettävät käsitteet ja määritelmät, ISO/IEC 29119-2,

jossa kuvataan testiprosessit, ISO/IEC 29119-3, jossa määritellään testidokumentaatio, ISO/IEC 29119-4, joka sisältää testaustekniikat sekä ISO/IEC 29119-5, joka sisältää avainsanalähtöisen testauksen. (ISO/IEC/IEEE 29119 2024).



Kuva 22 Standardin mukainen testauksen hallintaprosessi (ISO/IEC/IEEE 29119-1:2022)

Kuvassa 22 on ote standardijoukon testauksen hallintaprosessista ja vastaavasti kuvassa 23 dynaaminen testausprosessi, jotka ovat ohjelmistoprojektien sovellettavissa.



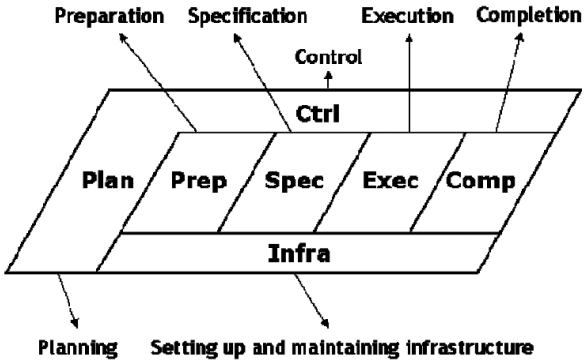
Kuva 23 Standardin mukainen dynaaminen testausprosessi (ISO/IEC/IEEE 29119-1:2022)

Seuraavassa kappaleessa käsitellään syvällisemmin TMap-mallia, jonka taustalla vaikuttaa ISO 29119 standardiperheen periaatteellisuus.

4.2 TMap elinkaarimalli järjestelmä- ja hyväksymistestauksessa

Testaus koostuu useista eri toiminnoista, joten kuvassa 24 on havainnollistettu TMap elinkaarimallin vaiheet toimintoineen sekä niiden keskinäiset riippuvuussuhteet, jotka ovat: Suunnittelu, valvonta, infrastruktuurin perustaminen ja ylläpito, valmistelu, määrittely, toteutus ja viimeistely. Vaikka kuvan 24 vaiheet on esitetty osin peräkkäin, niiden

suoritusjärjestys ei kuitenkaan ole ehdoton. Ei ole tavatonta, että esimerkiksi määrittelyvaihetta suoritetaan jollakin osa-alueella, kun samaan aikaan toisella osa-alueella suoritetaan testausta, joten TMap-mallin vahvuuksia on muun muassa sen ketterä ja joustava luonne. (TMap 2016, 177–178.)



Kuva 24 TMap elinkaarimalli (TMap 2016).

Tärkeä, mutta testausvaiheen aliarvioiduksi jäävä suunnitteluvaihe (planning) luo perustan korkealaatuiselle ja hallittavalle testausprosessille, joten kyseinen vaihe on ohjelmistoprojekteissa suositeltavaa aloittaa mahdollisimman nopeasti, vaikka yleisellä tasolla testaussuunnitelma olisikin jo laadittu, koska testauksen yksityiskohtaisempi suunnittelu suoritetaan jo tässä vaiheessa. Kyseinen vaihe sisältää myös projektille tärkeän riskianalyysiprosessin, jolla pyritään rajaamaan muun muassa testaukseen kuluvan ajan ja kustannuksen arvion varmistamaan, että tuotteen laadunvarmistus saavuttaisi riittävän kattavuuden. Valvontavaiheen (control) tavoitteena on ylläpitää suunnitelmaa, monitoroida ja raportoida testauksen edistymisestä tilaajalle ja kyseisen vaiheen toimintaa suoritetaan koko elinkaaren ajan. Vaatimustenhallinnassa tämä toiminta on oleellinen osa, jonka on kyettävä reagoimaan ja mukautumaan nopeisiin ja suuriinkin muutoksiin. Suunnittelun ja valvontavaiheen rinnalla alkaa varsinaisen konkreettisen testauksen myötä infrastruktuuri ja ylläpitovaihe (setting up and maintaining infrastructure), joka huolehtii varsinaisesta testauksen toiminnan suorittamiseen mahdollistavista asioista kuten työympäristö, resurssit ja työkalut elinkaaren loppuun asti. (TMap 2016, 171–172.)

Valmisteluvaiheessa (preparation) suoritetaan vielä testipohjan testattavuustarkistus, joka luo pohjan laadukkaiden testien suunnitteluun. Koska kehittäminen aloitetaan dokumentaatioista, jotka voivat itsessään sisältää virheitä, joten vaiheen toimenpide on tärkeä osa laadunvarmistusta, jolla ennaltaehkäistään kalliiksi koituvia virheitä. Valmisteluvaiheesta siirrytään varsinaiseen määrittelyvaiheeseen, joka määrittelee testiobjektille

tehtävät testit, jotta ne voidaan suorittaa testaajan toimesta mahdollisimman nopeasti. Suoritusvaiheessa (execution) suoritetaan testiobjektin testaus, jolla saadaan käsitys testiobjektin laadusta. Valmistumisvaiheessa (completion) arvioidaan testausprosessia, päivitetään tarvittaessa uudelleen käytettävät testitapaukset myöhempää käyttöä varten sekä luodaan loppuraportti. (TMap 2016, 173.)

4.3 Järjestelmätestaus

Kehitysympäristössä kehittäjien suorittamien yksikkö- ja integrointitestausten jälkeen suoritetaan kehitystyöstä mahdollisimman riippumattoman testaajan toimesta V-mallin (kuva 21) mukainen kolmas vaihe, järjestelmätestaus (system testing), joka testiympäristössä testaa kokonaisvaltaisesti ohjelmistoon toteutettuja toiminnallisia ominaisuuksia varmistaen, että ohjelmisto vastaa määritettyjä vaatimuksia ja odotuksia. Toiminnallisten ominaisuuksien lisäksi järjestelmätestaus kattaa myös ohjelmiston ei-toiminnallisten ominaisuuksien osuuden. Järjestelmätestaus ei tarkoita varsinaista testaustapaa, vaan se on yleisnimike kaikelle toiminnallisen kokonaisuuden testaukselle, jossa suoritetaan virheiden haravointia myös yksittäisten komponenttien tasolla. Lisäksi sen ominaispiirteisiin sisältyy muutokset eli testauksen osa-alueella on odotettavissa muutoksia vaatimuksiin. (Haikala & Mäkijärvi 2001, 271–272; Kasurinen 2013, 57.)

4.4 Hyväksymistestaus

Järjestelmätestauksen jälkeen, kun testiympäristön versio todetaan valmiiksi, suoritetaan hyväksymistestaus (acceptance testing) tuotantoa vastaavassa testiympäristössä. Hyväksymistestauksessa ei järjestelmätestauksen tapaan enää pyritä tekemään havaintoja, vaan todentamaan ohjelmiston kyvykkyys täyttää sille asetetut vaatimusmäärittelyt. Hyväksymistestaus on nimensä mukaisesti virallinen, itsenäinen, testausvaihe, jossa tilaaja suorittaa testauksen varsinaisessa kohdeympäristössä todetakseen ohjelmiston täyttävän siihen kohdistuvat hyväksymiskriteerit, vaatimukset ja odotukset. Hyväksymistestauksen myötä voidaan myös todeta, että tuote ei vielä täytä vaatimuksia, jolloin lakiteknisesti ohjelmisto ei siirry asiakkaan omaisuudeksi, vaan huolto- ja korjausvelvoite säilyy toimittajalla. Jos tuote ei täytä vaatimuksia, se voi johtaa muutosvaatimuksiin, uusien vaatimusten määrittelyyn tai jopa jonkin vaatimuksen poistamiseen (Haikala & Mäkijärvi 2001, 272.)

4.5 Regressio- ja automaatiotestaus

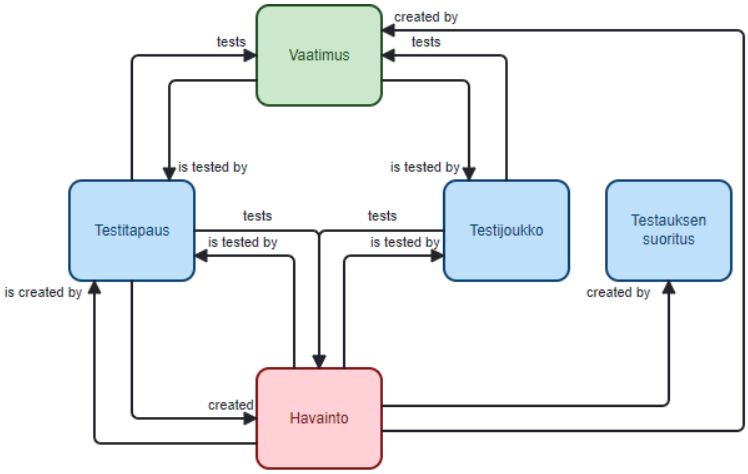
On huomioitavaa, että virheiden korjaus tulee sitä kalliimmaksi, mitä korkeammalla V-mallin testaustasolla virheet havaitaan, mutta tämän lisäksi virheiden korjaus voi aiheuttaa uusia virheitä eli muutoksia jo aiemmin toimivaksi todetuissa moduuleissa. Jotta tämä muutos ei jäisi huomaamatta, tarvitaan ohjelmiston uudelleentestausta. Tällöin tarttuu ohjaksiin regressiotestaus, joka lyhyesti kuvattuna tarkoittaa jo ohjelmiston oikein toimivassa osassa tapahtuvan muutoksen vaikuttavuutta koko ohjelmiston toiminnallisuudessa. Automaatiotestaus on oivallinen menetelmä regressiotestauksen luonteen vaatimaan toistuvuuden suorittamiseen aina uuden version myötä, jolla säästetään niin testaukseen kuluvaa aikaa kuin myös testausresursseja vapauttamaan vaativimpiin manuaaliseen testaukseen, johon automaatio menetelmänä ei kykene. Lisäksi automaatiotestaus nopeuttaa vaatimustenhallinnassa havainnoimaan määrittelyissä tapahtuvien muutosten vaikutusta siitä riippuviin moduuleihin. (Haikala & Mäkijärvi 2001, 272; Kasurinen 2013, 68–70)

4.6 Työkalut ja menetelmät

Ketterän menetelmän työkaluna esiteltiin aiemmin Jiraa, joten kyseinen työkalu tarjoaa puitteet myös testauksen suunnittelulle ja raportoinnille Xray-osiossa (kuva 25).

Kuva 25 Jira työkalun Xray osio

Jirassa testauksen hallinnointiin varattavissa oma osionsa, Xray, jossa testauksessa käytettävät elementit ovat Test (testitapaus), Test Set (testijoukko), Test Execution (testauksen suoritus muodostetaan Test tai Test Set yhteydessä), Test Plan (laaja kokonaisuus, joka sisältää testitapaukset ja/tai testijoukon) sekä Pre-Condition (yhteiskäyttöiset ehdot). Kuvassa 27 on havainnollistettu yleisimmät testauksessa käytettyjen sinisten tikettien välinen assosiaatio.



Kuva 26 Testauksessa Jira tikettien assosiaatiot (mukaillen Xray 2023).

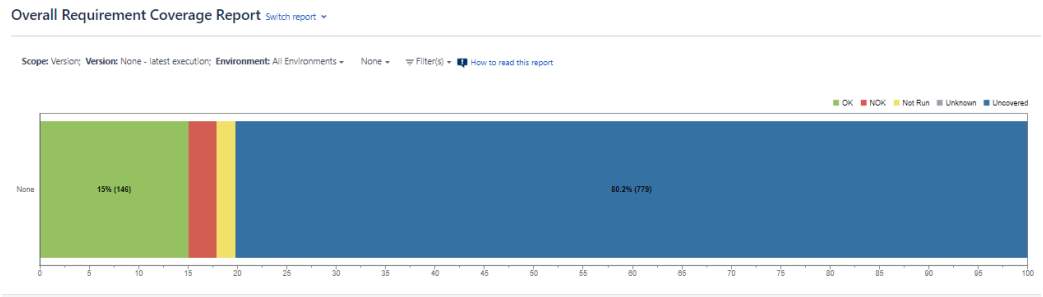
Näiden assosiaatioiden joukossa kulkeutuu kaikki se tieto, jotka on projektin alussa työkalussa konfiguroitu. Tiketin yksilöivä tunnus, joka koostuu projektin nimimerkistä ja numerosta (sekvenssissä) generoituu automaattisesti, mutta lisäksi projektin seurannan kannalta oleelliset kentät on määriteltävä erikseen.

Testitapausten avulla tutkitaan, toimiiko järjestelmä vaatimuksen mukaisesti, joten testitapauksen tulee sisältää ne tiedot, joilla tämä voidaan todentaa: Alkutilanne, toiminnot, ja ennustettu tulos (TMap 2016). Vaatimukseen linkitetty testitapaus tulee säilymään koko projektin elinkaaren ajan eli niitä ei saisi poistaa, jotta linkitys historiatietoineen säilyisi, koska tällä osuudella on merkittävä vaikutus muutoshallinnassa ja jäljitettävyydessä. Testitapauksen alla suoritetaan varsinainen testaus eli luodaan testitapaukseen testauksen suorittava tiketti, johon voidaan sisällyttää kaksi versiotietoa: Vaikuttava- sekä korjausversio. Lisäksi testauksen aikana laaditut havainnot linkittyvät automaattisesti testitapaukseen ja tikettiin tallentuu automaattisesti muun muassa testaukseen kulunut aika.

4.7 Testauksen kattavuus

Vaatimusten lisäksi myös testaus edellyttää hyväksymiskriteerit ja ne on määriteltävä testaussuunnitelmassa, joiden todentamiseen tarvitaan erilaisia mittareita. Yksi tapa mitata hyväksymiskriteerien täyttymistä on toiminnallisella kattavuudella, joka tarkoittaa testauksen kattavuutta määriteltyjen ominaisuuksien osalta (Haikala & Mäkijärvi 2001, 275, 277).

Kuvassa 27 on esimerkki projektista, jossa on kehitystiimin ja jatkuvan kehityksen aikana kertynyt Jira-työkaluun yhteensä 971 tapausta, joista voidaan tietoja suodattamatta laatia testauksen kattavuus sekä jäljitettävyysraportti (kuvassa 28 kuvaus jäljitettävyysraportin rakenteesta).



Kuva 27 Jira-työkalun testauskattavuus raportti

Kuten kuvasta 27 voidaan yhdellä silmäyksellä havaita, raportin mukaan vaatimusten kokonaismäärästä 80.2 % olisi testaamatta, vain 15 % todettu toimivaksi, 2.9 % todettu ei-toimivaksi ja 1.9 % jäänyt testaus kesken. Kyseinen raportti ei kuitenkaan kuvasta ohjelmiston todellista tilaa, vaan Jiran käyttö on tässä tapauksessa jäänyt puutteelliseksi.

Requirement	Test	Test Runs	Defects
Task AA-643 DONE UNCOVERED Version: 0.0.12 Title			
User Story AA-434 DONE OK Version: 0.0.10 Title	Test AA-435 TEST Title PASS	FAIL Test Execution AA-450 Version: 0.0.09 Finished On: Executed By: Tests environments: Revision: Test Version: v1 PASS Test Execution AA-450 Version: 0.0.10 Finished On: Executed By: Tests environments: Revision: Test Version: v1	Bug AA-451 DONE Title
Bug AA-532 DONE UNCOVERED Version: 0.0.10 Title			

Kuva 28 Kuvaus Jira-työkalun jäljitettävyysraportin rakenteesta

Kuvassa 28 on havainnollistettu jäljitettävyysraportin hierarkkisesta taulukkonäkymästä muutaman rivin esimerkki, miltä jäljitettävyys näyttää. Yksi valmiiksi siirretty tehtävätiketti on puutteellisin tiedoin jäänyt tilaan uncovered (kattamaton) ja heti sen alla myös valmiiksi siirretty tiketti, jossa testaus suoritettu kahdessa eri testiversiossa, josta toisessa versiossa todettu ei-toimivaksi ja tästä on laadittu havainto. Kuvan alareunassa lisäksi yksi havainto siirretty valmiiksi, mutta johon ei ole liitetty varsinaista testitapausta tai kyseinen havainto ei ole hierarkkisesti liitetty vaatimuksen tai tehtävän alatasolle. Ilmiön taustalla näkyy kuvan 27 mukainen syy eli Jiran puutteellinen käyttö. Perusperiaate olisi, että jokaiseen vaatimukseen linkitetään testitapaus ja testitapaukseen testaussuoritus versiotietoineen. Versiotiedon lisäys vaatimukseen sekä testitapaukseen vahvistaa jäljitettävyyttä ja esimerkiksi Jira työkaluna tarjoaa monipuolisen ja tehokkaan versiotiedon käytön. Testauksen yhteydessä laaditut havainnot tulee myös linkittää testisuoritukseen, jota kautta havainnon näkyvyys välittyy testitapaukseen sekä vaatimukseen.

Puutteellisen Jira työkalun käytön yhteydessä voidaan todeta, että raporttien kautta projektin seuranta on harhaanjohtavaa ja vaikeuttaa halutun tilanteen analysointia. Menneisyyden, nykyhetken ja tulevaisuuden analysoiminen ei ole mahdollista, jos suuntaa antavanakaan. Hyväksymistestaus on näin ollen ainoa keino tilaajan näkökulmasta todeta, onko versio riittävällä tasolla tuotantoon vientiin. Testaajan näkökulmasta arviointi perustuu viimeisen regressiotestin lopputulokseen.

4.8 Ohjelmistotestauksessa todennettuja havaintoja

Hara (2013) on tutkimuksessaan tarkastellut ketterää ohjelmistoa asiakkaan näkökulmasta ja tuo esille tutkimustuloksessaan asiakkaan aktiivisempaa osallistumista kehittämisen aikana. Tällä hän korostaisi aktiviteetit muun muassa testien tuottamiseen, testaukseen osallistumiseen ja testauksen suorittamiseen. Testauksen suorittamisella hän ehdottaa joka sprintin päätteeksi koekäyttöä eli ei ainoastaan tiimin toimesta sprint review esittelyä iteraation tuotoksesta, vaan asiakas konkreettisesti kokeilisi tuotteen kypsyyttä.

Lindvall (2022), tutkiessaan vaatimusmäärittelyä ketterässä ohjelmistokehityksessä, vetoaa laadukkaan ja kattavan testauksen vaikuttavan merkittävästi projektin aikatauluun, budjettiin sekä tuotteen laatuun. Laadukkaan ja kattavan testauksen toteutumisen edellytyksenä hän kuitenkin pitää toteutettavien ominaisuuksien etupainotteista suunnittelua ennen varsinaista toteutusvaihetta, mutta käytännössä tätä periaatetta ei ole noudatettu.

Tikka (2015) jäljitettävyyshaasteita tutkiessaan korostaa jäljitettävyysketjun tärkeyttä vaatimuksista suunnitteluun, suunnittelusta toteutukseen, toteutuksesta testin suorittamiseen ja testin suorittamisesta verifiointiin ja validointiraportointiin. Linkityksen säilyvyys edellyttää oikeanlaista ylläpitoa, jota hallinnoi ihmiset eli sitoutuneisuutta tehdä asioita oikein. Hokkanen on jo vuoden 2001 tutkimuksessaan todennut, että vaatimusten kulkuketju on oltava selkeä vaatimuksesta testaukseen, jolloin muutosten vaikutusten arviointikin helpottuu. Hänen mukaansa tämä edesauttaa testaajia varmistamaan, että kaikki vaatimukset on testattu ja otettu käyttöön sekä helpottaa kartoittamaan vikoja. Hokkanen on ehdottanut jäljitettävyyden helpottamiseksi, että testitapauksen tunnus olisi identifioitavissa siihen liittyvällä käyttötapaustunnuksella, joka takaisi jäljitettävyyden. Kaikesta huolimatta hän pitää ongelmana, että jossain vaiheessa ketju kuitenkin katkeaa vaatimuksista testitapauksiin.

Tutkimustuloksien perusteella jäljitettävyys ei olisi riippuvainen ainoastaan versiotiedon päivittämisestä asiakirjoihin, vaan myös tunnusten avulla olisi mahdollista linkittää ja jäljittää. Muutosvaatimuksen yhteydessä muutoksen päivittäminen jokaiseen työkaluun on kuitenkin työlästä. Linkkien katkeaminen voi kuitenkin johtaa duplikaatteihin, jotka kuormittavat koko tiimiä. Scrum- ja ylläpitotiimi itsessään pienen koon vuoksi jonkin verran ehkäisee duplikaatteja sekä menetelmänä sprintille palastetut työlistat, mutta miten jäljittää vaatimukset, jo suoritetut testaukset, havainnot, korjaukset sekä kehitystiimillä työlistalle jääneet, ylläpidolle siirtyvät tehtävät. Ongelman taitekohdaksi muodostuu tuotantoon viennin jälkeen siirryttäessä jatkuvaan kehitykseen, näkyvyys ennen tuotantoa tapahtuneisiin muutoksiin.

5 OHJELMISTON LAATU

Laatu on laaja, abstrakti käsite, joten ohjelmistoprojekteissa olisi tärkeää määrittää ja rajata käsitteen merkitys ennen kuin sille kannattaa suunnitella laadun arviointikriteerejä ja mittareita. Laadulle voidaan tunnistaa esimerkiksi kuusi laadun osa-aluetta: Määrittelemätön, tuotepohjainen, käyttäjäpohjainen, valmistuspohjainen sekä arvopohjainen laatu. Laadun käsite on siis moniulotteinen, riippuvainen siitä, kenen näkökulmasta kysytään. (Kasurinen 2013, 136–137.) Testaus on ohjelmistoprojektille arvokas mittari varmistamaan ohjelmiston laatua, täyttääkö se yhdessä ISO-standardin ja tilaajan ohjelmistolle asettamat toiminnalliset sekä ei-toiminnalliset vaatimukset.

5.1 Laatustandardi ISO/IEC 25010

Vanhan laatustandardin arviointijärjestelmän kulmakiven ISO/IEC 25010:2011 (järjestelmän ja ohjelmiston laatumalli) korvaa nykyisin ISO/IEC 25010:2023 (tuotteen laatumalli), joka on sovellettavissa ICT- ja ohjelmistotuotteisiin. Tästä esimerkkinä tunnettu menetelmä TMap (test management approach), jonka taustalla huomioitu myös kyseinen standardi. Tuotteen laatumalli sisältää järjestelmän ja ohjelmiston laatumallin kahdeksan pääominaisuuden lisäksi yhdeksännen pääominaisuuden ja nämä ovat jaettu vastaavasti alaominaisuuksiin (kuva 29). Laatustandardin laatuominaisuudet ovat viitteellisiä määritettävien, mitattavien ja arvioitavien tuotteiden laadulle, mutta on kuitenkin suotavaa ottaa huomioon ohjelmistotuotteen ominaisuuksia arvioitaessa. Laatumallin ominaisuudet ovat muotoutuneet eri sidosryhmien ilmaistuja ja oletettuja tarpeita huomioiden ja siten ominaisuudet tuottavat arvoa tuotteen laadulle. (ISO/IEC 25010 2024; ISO/IEC 25010:2023, 2024.)

Ohjelmistotuotteiden laatu				
Toiminnallinen soveltuvuus	Suorituskyvyn tehokkuus	Yhteensopivuus	Käytettävyys	Luotettavuus
• Kattavuus • Oikeellisuus • Soveltuvuus	• Vasteaika • Resurssien käyttösuhde • Kapasiteetti	• Rinnakkaiselo • Yhteentoimivuus	• Sopivuuden tunnistettavuus • Opittavuus • Toimivuus • Käyttövirheiden estäminen • Käyttäjäsitoutuminen • Inklusiivisuus • Käyttäjäapu • Itsekuvaavuus	• Virheettömyys • Saatavuus • Vikasietoisuus • Palautettavuus
Tietoturva	Ylläpidettävyys	Joustavuus	Turvallisuus	
• Luottamuksellisuus • Rehellisyys • Kiistämättömyys • Vastuullisuus • Aitous • Resistanssi	• Modulaarisuus • Uudelleen käytettävyys • Analysoitavuus • Muokattavuus • Testattavuus	• Sopeutumiskyky • Skaalautuvuus • Asennettavuus • Vaihdettavuus	• Toiminnallinen rajoitus • Riskien tunnistaminen • Vikaturvallinen • Vaaravaroitus • Turvallinen integrointi	

Kuva 29 Ohjelmiston standardin mukaiset laatuominaisuudet (mukaillen ISO 25010, 2024).

Tuotteen laatumallia voi hyödyntää vaatimusten määrittelyssä ja tuotteen laadun arvioimisessa koko sen elinkaaren ajan. ISO/IEC 25010:2023 mukaan laatumalleja voidaan hyödyntää: Tuote- ja tietojärjestelmävaatimusten selvittämisessä ja määrittelemisessä, vaatimusmäärittelyjen kattavuuden validoinnissa, tuotteiden ja tietojärjestelmien testaustavoitteiden tunnistamisessa, määritettäessä laadunvalvontakriteerit osaksi laadunvarmistusta, tuotteen ja/tai tietojärjestelmän hyväksymiskriteerien tunnistamisessa sekä tuotteiden laatuominaisuuksia koskevien toimenpiteiden vahvistamisessa näiden toimien tueksi. (ISO/IEC 25010 2024; ISO/IEC 25010:2023, 2024.)

5.2 Näkökulmia ohjelmiston laatuun

Tuotteen tilaaja on asiakas, oli se sitten yrityksen ulkopuolinen tilaaja tai saman yrityksen organisaation toisen yksikön vaatimuksiin perustuva tilaus. Lähtökohtana kuitenkin on, että projekti nähdään tilaustyönä, jossa korostuu asiakasnäkökulma eli syntyy tilaaja-toimittaja-suhde. Tilaaja on kuitenkin se taho, joka maksaa, määrittelee ja asettaa tuotteen vaatimukset eli mitä haluaa ja arvioi projektin lopussa vastaako toimitus vaatimusmäärittelyjä. (Ruuska 2012, 162–163.)

Tilaaja johtaa laatua eli laatujohtaminen asiakkaan näkökulmasta edellyttää, että prosessissa on huomioitu myös tuotteen substanssipuoli eli asiakkaan projektipäällikkö. Laadun varmistamisessa vastuussa on sekä toimittaja, että asiakas, mutta projektin onnistumisen kannalta oleellista on, että asiakas on projektin alusta loppuun asti tiiviisti mukana tekemisessä.

Sertifioidun laatujärjestelmän käyttöönotto asettaa projektille seuraavia vaatimuksia: Toiminnot kuvataan kirjallisesti, vastuut ja valtuudet määritellään selkeästi, henkilöstölle on jaettava heitä koskeva laatuohjeisto, käytännön toiminnan on vastattava ohjeita, työnkulku on jäljitettävissä, versioiden hallinta, pöytäkirjojen, muistioiden yms. laadinta korostuu, laatuvaatimus koskee myös alihankkijoita. Jos toimittaja ei käytä sertifioitua laatujärjestelmää, asiakas voi vaatia, että projektissa noudatetaan kyseisiä vaatimuksia. Tämä turvaa myös asiakkaan näkökulmasta projektin onnistumisen. (Pellin 2011, 40.)

Projektiryhmälle projektin tärkein prioriteetti on pysyä aikataulussa ja vasta toiseksi tärkeänä prioriteettina pidetään laatua, mutta varsinaisen toteutuksen koodin kirjoittajan tavoitteena on kirjoittaa virheetöntä koodia eli koodin on toimittava virheettömästi määritelmän mukaisesti. Testausta ei kehitystyössä pidetä laadun tärkeimpänä lähteenä, vaan laatu on kehittäjän näkökulmasta lähtöisin hyvin tehdystä suunnitelmasta ja kehitystyöstä. Vaikka kehitystyö tehtäisiin erityisellä huolella, huonosti suoritettu laadunvarmistaminen voi kuitenkin johtaa huonoon lopputulokseen. (Kasurinen 2013, 133).

Testauksen tarkoituksena on tutkia, toimiiko toteutus määrittelyn mukaisesti, joten testauksen näkökulmasta projektin aikataulussa pysymisen lisäksi laatu on korkealla prioriteetilla. Vaatimusmäärittelyjen toiminnallisuuksien toimivaksi todentamisessa testaus noudattaa standardien mukaisia menetelmiä tavoitteena täyttää ohjelmistolle asetetut laatuvaatimukset, joten testausta voidaan pitää laadunhallinnan (quality control, QC) ja laadunvarmennuksen (quality assurance, QA) menetelmänä. (Broekman ym 2007, 35; Kasurinen 2013, 133.)

Loppukäyttäjällä voidaan tarkoittaa luonnollista ja/tai ei-luonnollista käyttäjäryhmää kuten toinen tietojärjestelmä, jonka taakse kätkeytyy oma käyttäjäryhmä, joten laadun näkökulma voi muotoutua ihmisen käsityksestä tai tietojärjestelmän asettamista laatuvaatimuksista. Loppukäyttäjän kohderyhmän koostumuksesta riippumatta käyttäjä asettaa korkeimman prioriteetin tuotteen laadulle, johon kulminoituu niin laatustandardit kuin määriteltyjen vaatimusten toimivuus. Budjetin merkitys loppukäyttäjälle voi olla välillinen

siten, että projektiin käytetty ylibudjetti voi vaikuttaa tuotannossa ohjelmiston käyttökustannuksiin.

5.3 Hyväksymiskriteerit

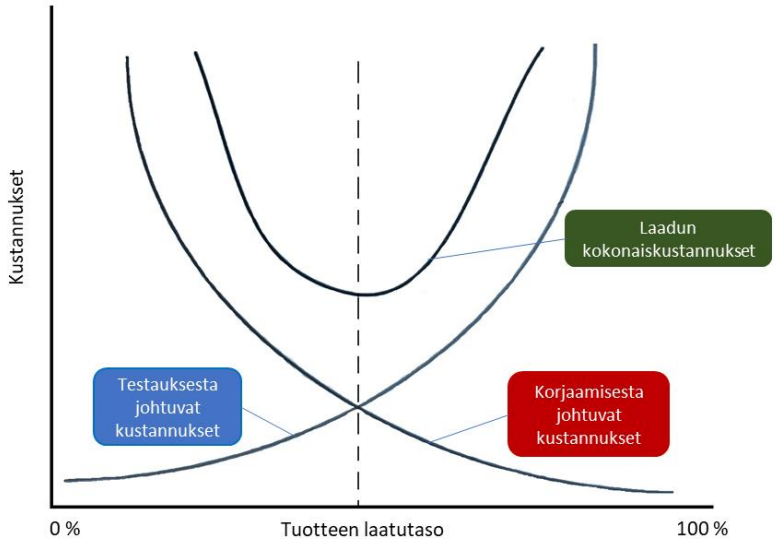
Hyväksymiskriteerit ovat lyhyesti kuvailtuna käyttäjätarinoiden ehtoja, joiden täyttyessä vaatimus hyväksytään, joten testauksella on tärkeä rooli todentaa kyseisten ehtojen täyttyneen. Tieto ei ainoastaan kuvasta ohjelmistoversion kypsyyttä, vaan ilmaisee myös milloin testaus voidaan lopettaa, koska testauksen hyväksymiskriteerit ovat riippuvaisia vaatimuksiin määritellyistä kriteereistä. Eeppisen, ominaisuuden ja käyttäjätarina -tasojen hyväksymiskriteerien toteutumiseen liittyy ominaisuuden toteutuksen aloituksen ehtoja määrittelevä Definition of Ready (DoR) sekä toteutuksen valmistumisen ehtoja määrittelevä Definition of Done (DoD). (Atlassian 2024; ISO/IEC/IEEE 29119-1:2022.)

DoD ilmaisee myös valmiiden ja keskeneräisten lukumäärän ja suhdanteen ohjelmistoversiossa, mutta määritelmän hyödyllisyys on riippuvainen tiimin ja sidosryhmien yhteistyön aktiivisuudesta ylläpitää kyseisiä kriteerejä. DoD koostuu kriteereistä, jotka määrittelevät ominaisuuden valmiuden tuotteeseen, joten sprintin päätyttyä nämä kriteerit tarkistetaan ja niiden perusteella päätetään, onko ominaisuus valmis lisättäväksi tuotteeseen. DoR suoritetaan tuotteen työlistalla niin sanottuina matalan tason kriteereinä, jotka määrittävät, milloin ominaisuus on valmis tiimin poimittavaksi ja toteutettavaksi tulevaan sprinttiin. (Atlassian 2024.)

Valmiin määritelmät parantavat ohjelmiston laatua ja takaavat laatustandardien täyttymisen, minimoi riskejä ennalta ehkäisten mahdollisten uudelleentyöskentelystä aiheutuvia viiveitä, parantaa tiimien yhteistyötä, koska kaikilla sama käsitys mitä tarkoittaa ”valmis” ominaisuus sekä mittaa edistymistä eli voidaan jäljittää tekemättömiä, keskeneräisiä ja valmiita ominaisuuksia. DoD määrittely edellyttää aktiivista yhteistyötä koko tiimiltä ja hyvä DoD pitää olla tarkka, mitattavissa ja saavutettavissa oleva, relevantteja ja määräaikaisia. DoD tietoa voidaan hyödyntää myös vaatimusten jäljitettävyydessä, mutta vasta hyväksymiskriteerien tasolla voidaan varsinainen ohjelmiston valmius punnita. Tuolloin vaatimus ei ole ainoastaan sisällytetty valmiin määritelmän perusteella tuotteeseen, vaan on testauksen avulla todennettu koko ohjelmistossa toimivaksi täyttäen asiakkaan näkökulmasta hyväksymiskriteerien ehdot, rikkomatta ohjelmiston muita toiminnallisuuksia. (Atlassian 2024.)

5.4 Testauksen kustannukset

Kustannukset ovat sidottu muun muassa kolmeen muuttujaan: Aika, käytettävissä oleviin resursseihin sekä ohjelmiston laatuvaatimuksiin. Kustannukset kulkevat käsi kädessä laadun kanssa eli mitä korkeampaa laatua tavoitellaan, sitä korkeampia kustannuksia siitä seuraa, niin ajan kuin resurssin akselilla. Tavoitteen toteutuminen edellyttää laadukkaita vaatimusmäärittelyjä, testaussuunnitelmaa sekä erityistä panostusta testaustyöhön, laadunvarmistukseen ja laadunvalvontaan. Kuvassa 30 on havainnollistettu laadun hinnan riippuvuussuhteista.



Kuva 30 Laadun hinta (mukaillen Kasurinen 2013, 133)

Mitä enemmän panostetaan laadunvarmennukseen eli testaukseen, sitä suuremmat kustannukset, mutta laadukkaampi tuote. Toisaalta säästetään ylläpitokustannuksista, kun suurin panostus on tehty tuotteen kehitysvaiheessa. Kuvaajasta voidaan myös havaita kohta, jossa voidaan todentaa, onko laatu riittävällä tasolla tuotantoon. Hyväksymiskriteerien yhteydessä on hyvä määrittää riittävä hinta-/laatusuhde eli millä kustannuksilla saavutetaan riittävä laatu, koska tuotantoon vienti ei edellytä korjausvaatimusten suhteen nollatoleranssia. Ketterässä ohjelmistokehityksessä myös jatkuvan priorisointityön merkitys korostuu koko elinkaaren ajalla kuvaa 30 katsomalla. (Kasurinen 2013, 133–134.)

5.5 Testauksen luotettavuusarviointi

Testauksen laadulla on suora vaikutus tuotteen laatuun eli yksi näkökulma luotettavuusarviointiin voidaan suorittaa testauksen osa-alueella. Testitapauksien kattavuus voidaan selvittää virheiden kylvämisellä koodiin, jolloin selvitetään lisättyjen virheiden määrä suhteessa löydettyihin virheisiin. Mutaatiotestaus menetelmänä eroaa kylvettyjen virheiden testauksessa siltä osin, että mutaatiossa koodiin tehdään vaatimuksesta eriäviä toiminnallisia muutoksia, ei välttämättä varsinaisia virheitä, ja tarkistetaan tunnistaako testitapaukset muuttuneet tai toimimattomat, ei määrittelyjä vastaavia toiminnallisuuksia. Ongelmaksi kyseiset menetelmät koituvat, jos kaikkia kylvettyjä virheitä ei huomata poistaa ohjelmasta, jolloin toimiva jäljitettävyysprosessi ehkäisisi kyseisen riskin. (Kasurinen 2013, 80–81).

6 PÄÄTELMÄT JA KEHITYSEHDOTUKSET

Ketterää menetelmää on hyödynnetty jo vuosikymmeniä, joten opinnäytetyössä käytettävän kirjallisuuden aineistossa käytettiin jopa vuoden 2000 tutkimustyön tuloksia. Tutkimustyöt, kaikkiaan 16 kappaletta, poimittiin tieteen alasta riippumatta satunnaisotannalla FINNA:sta esimerkiksi hakusanoilla: Requirements traceability, vaatimusten jäljitettävyys ja vaatimustenhallinta. Hakuehdot tuottivat runsaasti hakutuloksia mikä jo itsessään viittaa aiheen tärkeyteen, mutta myös edelleen aiheessa piilevistä haasteista vuodesta 2000 vuoteen 2024 asti.

Tässä opinnäytetyössä käytiin läpi, niin teorian kuin myös tutkimustöiden tulosten pohjalta, ohjelmistoprojektille oleellisia osa-alueita: Menetelmä, jolla projekti toteutetaan, vaatimustenhallinta, ohjelmistotestaus tärkeänä osana laadunvarmistusta sekä ohjelmiston laatua. Nämä osa-alueet muodostavat erittäin laajan ja herkästi haavoittuvan kokonaisuuden runsaiden menetelmien, mallien, viitekehysten ja työkalujen kirjoilla.

Tutkimuksen pääkysymys oli: Miten vaatimusmäärittelyt ovat jäljitettävissä ketterässä projektissa? Kysymykseen lähdettiin kartoittamaan vastausta selvittämällä ensin vastausta seuraaviin kysymyksiin:

Miksi vaatimusmäärittelyt on oltava helposti jäljitettävissä ketterässä ohjelmistoprojektissa?

Tutkimustuloksiin vedoten testauksen näkökulmasta vaatimusmäärittelyjen helppo jäljitettävyys tarkoittaa nopeaa testauksen tilanteen analysointia niin menneisyyteen, nykyhetkeen kuin tulevaisuuteen. Raportit ovat erinomaisia edistymisen ja tilanteen seurantavälineitä, mutta niiden hyödyntäminen ja luotettavuus on täysin digitaalisia työkaluja käyttävien tiimiläisten armoilla: Jos digitaalista työkalua ei ole konfiguroitu riittävän hyvin eli tulee jäämään tiedoiltaan vajavaiseksi tai jos tietokenttiä muutetaan kesken projektin radikaalisti, näillä on radikaali vaikutus raportin luotettavuuteen, koska tällöin tulos projektin alusta nykyhetkeen ja ennuste on puutteellinen tai muuttuu merkittävästi. Useassa tutkimustyön tuloksissa korostetaan projektin keskuudessa yhteisten pelisääntöjen ja yhteisymmärryksen merkitystä, joten keskeiseksi ratkaisuksi lähes kaikissa tutkimustuloksissa nousi koulutuksen tarve. (Hokkanen 2001; Tikka 2015.)

Jäljitettävyyden merkitys testauksessa korostuu myös, kun testausta on suorittamassa useampi henkilö, vaikkakin testattavien tehtävien allokointi olisikin hyvin organisoitua,

duplikaattien mahdollisuus on aina olemassa jopa projektin ainoan testaajan toimesta. Testaajan on siis saatava helposti jäljitettyä aiemmat havainnot välttääkseen testauksen tuplatyötä, kuormittamasta kehittäjien työtä ylimääräisillä havainnoilla sekä minimoidakseen tilastollista vääristymää. Tutkimustuloksissa korostuu, että vaatimus tulisi kirjata vain kerran ja muutosvaatimuksessa muutos tulisi tehdä vain yhteen kohtaan, joka helpottaa jäljitettävyyttä. Tämä vaikuttaa testitapauksiin siten, että ne ovat uudelleen käytettävissä, jolloin työmäärä vähenee ja helpottaa havainnon paikantamista. Vaikka vaatimuksiin kohdistuisi muutoksia, se ei kuitenkaan hankaloittaisi jäljitettävyyttä, jolloin saadaan myös realistinen tilannekatsaus testauksen kattavuudesta käyttäjätarinasta jopa versiotasolle asti, mitkä vaatimuksista on testattu, mitkä vaatimuksista on vielä testaamatta. (Kujala 2000; Hokkanen 2001; Saarinen 2016.)

Jäljitettävyyteen liittyvän tutkimushavainnon huomioiden, jotta testauskattavuutta koskeva raportti olisi mahdollisimman luotettava, tulisi olla jäljitettävissä myös ei-toiminnalliset ominaisuudet, joka asettaa erityisiä vaatimuksia digitaalisen työkalun ominaisuuksille ja käytölle (Dawoud 2014).

Millainen vaatimusmäärittely on edellytys ketterässä projektissa?

Ketterässä projektissa vaatimusmäärittelyltä edellytetään tutkimustulostenkin mukaan jokseenkin ristiriitaisia vaatimuksia: Se ei saisi olla alussa liian yksityiskohtainen, jotta mahdolliset muutokset ovat joustavasti ja paremmin hallitusti tehtävissä, mutta toisaalla vaaditaan mahdollisimman yksityiskohtaista kuvausta jo projektin alussa, jotta voidaan ennakoida kustannuksia, työmääriä ja aikataulua. Vaatimusmäärittelyn perusteella rakentuu käyttäjätarinat ja varsinaiset työmääräarviot, joiden avulla voidaan ennustaa projektin aikataulua ja kustannuksia sekä tuotteen kypsyyttä. (Hara 2013; Paakkari 2022.)

Tutkimustuloksissa yhteisesti korostuu tilaajan/asiakkaan ja tiimin yhteisymmärryksen ja yhteistyön merkitys, joiden yhteisenä nimittäjänä on vaatimukset (Salmi 2020). Vaatimusmäärittelyt tulee olla kuvattuna, ei liian yksityiskohtaisesti, mutta riittävällä tiedolla, joka mahdollistaa projektin etenemisen oikeaan suuntaan. Vaatimusmäärittelyn tulee sisältää hyväksymiskriteerit, koska ollaan valmiita aloittamaan toteutus ja koska toteutus voidaan todeta valmiiksi. Lisäksi vaatimukselle tulisi määrittää arvo eli mille prioriteetille vaatimus asettuisi, joka sanelee toimintojen toteutusjärjestystä. Tällä on vaikutusta myös kustannustehokkuuteen, jos budjetin katto saavutetaan ja aikajanan oikea pääty, tuotteessa on saavutettu laadullisesti kaikki tärkeimmät ominaisuudet valmiina tuotantoon. Vaatimusmäärittely tulee olla sidottavissa testitapaukseen, jotta sen jäljitettävyys olisi

mahdollista kaikissa sen olomuodoissa ja ajanjaksoilla. Ajanjaksoksi voidaan laskea myös tuotannonviennin jälkeinen tulevaisuus, vaikka tuote toimii nykyhetkellä moitteettomasti, on varauduttava mukautumaan muutoksiin (Hou 2021).

Ketterän ohjelmistokehityksen Scrum ja Kanban käytettävyys, vahvuudet ja heikkoudet?

Ketterät ohjelmistot ovat pyrkineet taklaamaan vesiputousmallin heikkoudet, mutta ketterästä menetelmästä huolimatta tässä tutkimuksessa näkyy vesiputousmallia siitä, miten yksi asia tehdään huonosti, vaikuttaa se aina seuraavaan ja ongelma kertaantuu. Tutkimustuloksia tarkastellessa osoittautui menetelmä tiimille itsessään osin kuormittavaksi, mutta kuormittavuus periytyy määritellyistä vaatimuksista: Huonosti ymmärrettävistä vaatimuksista on vaikeaa ja hidasta lähteä koostamaan toimivaa tuotetta. Tutkimustöiden tuloksissa toistuu sama asia: Yhteistyö, yhteiset pelisäännöt ja priorisointi. Jokaisen tulisi myös ymmärtää oman roolin tehtävät ja toimia sen mukaisesti. Vahvuuksina pidetään menetelmän sanansa mukaisesti ketteryyttä, mutta ongelmalliseksi koetaan havaintojen korjauksen aiheuttama ylimääräinen työtaakka ja iteraatioiden suunnittelu ajanjaksolle sopivaksi. (Kaukavuori 2000; Hara 2013; Kokko 2013; Lindvall 2022; Luurila 2023; Kamath 2023.)

Miksi testauksen osallisuus on tärkeää jo määrittelyn alkuvaiheessa ja millainen vaikutus sillä on kustannustehokkuuden kannalta?

Testaus on merkittävä osa projektia koko sen elinkaaren ajan, on kuitenkin kyse laadunvarmistamisesta, jolloin testaus toimii mittarina tuotteen kypsyyden määrittämisessä. Resurssitarve eli testauksen henkilöstömäärän tarve elinkaaren eri vaiheissa muuttuu, mutta onnistuneen projektin edellytys olisi testauspäällikön osallisuuden alkavan jo vaatimusmäärittelyn alkuvaiheessa. Testauspäällikkö tekee testaussuunnitelman, joten on odotettua testauspäällikön pysyvän tehtävässään projektin alusta loppuun eli kykenee sitoutumaan työtehtäviinsä tarvittavan elinkaaren ajan. Koska testitapaukset tulevat perustumaan tuotteen vaatimuksiin, testauspäällikkö voisi olla osallinen itse vaatimusmäärittelyissä, yhteistyössä tuoteomistajan kanssa. Tilaajan tulee siis ymmärtää testauksen merkitys, osallisuus ja ajallinen tarve suunnitelmassaan.

Vaatimusmäärittelyt elävät läpi projektin elinkaaren ja eri määrittelyn tasoilla toteutetaan erilaisia ehtojen joukkoja kriteereineen, joten testauksen osallisuuden tarve projektin alkuvaiheessa tulee huomioida myös kaikissa sen määrittelyn muissa vaiheissa. Tutkimustuloksissa korostui ominaisuuksien suunnittelutyön etupainoisuus ennen toteutusvaihetta, joten kyseinen vaihe edellyttäisi myös testauksen läsnäoloa (Lindvall 2022).

Esimerkiksi sprintin suunnittelun yhteydessä DoD perustuvaan työmääräarvioissa jätetään helposti testauksen osuus kokonaan pois, vaikka testaus siellä mainittaisiinkin, ja pisteytyksessä keskitytään vain kehittäjien työmääräarvioihin. Tämä johtaa siihen, että testauksen työt kasautuvat ajan myötä pullonkaulaksi, jolloin kokonainen sprintti voi vierähtää kasautuneiden tehtävien verifiointiin. Testauksen osallisuus määrittelyjen jokaisessa vaiheessa edellyttäisi myös sprintin aikana useamman testiversion jaksotus, vaikkakin uusi testiversion on riippuvainen siitä, toteutetaanko myös ominaisuuksia vaiheittain vai kaikki kerralla sprintin loputtua. Ominaisuuksien jaksotus sprintin aikana testaukseen ehkäisisi pullonkaulaa ja havaintojen kasaantumista. Testaus laatii testitapaukset käyttäjätarinoista, joten on päästävä suunnittelemaan ja saattamaan sprintille poimittujen käyttäjätarinoiden testitapaukset valmiiksi viimeistään sprintin alussa, ennen ensimmäistä testiversiota.

Jos testitapaukset on saatu laadituksi ennen sprinttiä, ennen suunnittelupalaveria, työmääräarviossa voidaan huomioida paremmin myös sprintin aikana testaukseen kuluva aika, jolloin testaus ei aiheuttaisi aikatauluun niin sanottua sprinttien ylivuotoa eli ei aiheuta testattavien asioiden kasautumista aina seuraavaan sprinttiin. Testauksen osallistuminen etupainoisesti mahdollistaa myös koeversion valmiiksi testaamista. Näillä havainnoilla ja huomioilla on suora vaikutus kustannuksiin ja niiden arviointi vaikeutuu. Testauksen vaikuttavuus kaikessa määrittely- ja suunnitteluvaiheissa johtaa kustannustehokkuuden kannalta realistisempiin kustannusarvioihin sekä palvelee ylläpitoa välttämällä yllättäviä lisäkustannuksia.

Tutkimuskysymyksiin kohdistui hypoteeseja, jotka voidaan tutkimustulosten perusteella todentaa: Pitää täysin, osittain tai ei lainkaan paikkaansa:

1. Jos vaatimusmäärittelyjä ei kyetä jäljittämään, duplikaattien määrä kasvaa ja niistä aiheutuu ylimääräistä selvitystyötä ja aiheuttaa ristiriitoja muutosvaatimusten yhteydessä.

Pitää osittain paikkaansa. Kattavalla ja tehokkaalla digitaalisen työkalun käytöllä duplikaatit voisivat pikemmin osoittaa toiminnallisuuden epästabiilisuudesta tai mahdollisesti indikoida huonosti skaalatuista ja luokitelluista vaatimusmäärittelyistä. Duplikaatit eivät välttämättä aiheuta vääristyneitä tilastoja, eivätkä kuormita ylimääräisellä työllä, jos linkitykset tehdään työkalussa alusta asti oikein.

2. Jos projektitiimin henkilöstö vaihtuu, työmääräarvioilla on suurempi toleranssi, uuden henkilön vaikeaa sisäistää projektin sisältöä, joka hidastaa etenemistä, tiimistä poistuva henkilö vie mukanaan arvokasta tietoa.

Pitää täysin paikkaansa. Työmääräarvioinneissa suurempi toleranssi, koska uuden henkilön tulee aina sisäistää projektin sisältöä. Koska ketterän menetelmän ominaisuuksiin kuuluu vähäinen dokumentointi, tiimistä poistuva henkilö vie mukanaan arvokasta tietoa ja taitoa.

3. Jos asiakkaan ymmärrys tuotteesta on heikko, sillä on vaikutus koko projektitiimiin, loppukäyttäjiin sekä projektin toteutumiselle siten, että aikataulu viivästyy ja kustannukset kasvavat. Aikataulussa ja budjetissa pidättäytyminen taasen heikentää tuotteen laatua.

Pitää täysin paikkaansa. Asiakkaalta ei edellytetä teknistä asiantuntemusta, mutta substanssiosaaminen tulee olla vahvaa, jolloin määrittelyt eivät jää puutteellisiksi ja tiimiä ei kuormiteta tarpeettomilla määrittelyillä.

4. Jos projektitiimin ymmärrys tuotteesta on heikko, sillä on vaikutus projektin aikatauluun, budjettiin ja laatuun siten, että aikataulu viivästyy sekä kustannukset kasvavat. Aikataulussa ja budjetissa pidättäytyminen taasen heikentää tuotteen laatua

Pitää täysin paikkaansa. Mitä nopeammin tiimi ymmärtää tuotteen, sitä nopeampaa on tehdä asioita heti oikealla tavalla, jolla on suora vaikutus kustannuksiin.

5. Jos ylläpidolla ei läpinäkyvyyttä vaatimusmäärittelyihin, käyttäjätuen ja jatkuvan kehitystyön laatu heikkenee.

Pitää täysin paikkaansa. Ylläpidolle periytyvä tekninen velka, niin vaatimusten kuin havaintojen muodossa, edellyttää läpinäkyvyyttä vaatimusmäärittelyihin. Siirtymävaihe ei saisi kadottaa tietoja kehitystiimin ja ylläpidon välillä.

6. Jos työkalun käyttö on puutteellista tai osaamatonta, sillä on vaikutus raporttien luotettavuuteen, jolloin raportin hyöty jää vähäiseksi ja voivat antaa jopa väärää indikaatiota projektin ja tuotteen tilanteesta.

Pitää täysin paikkaansa. Tutkimuksessa tuli esille, että digitaalisen työkalun oikeanlainen käyttö takaa myös laadukkaammat ja luotettavammat raportit ja niiden hyöty on tällöin tarkoituksenmukainen.

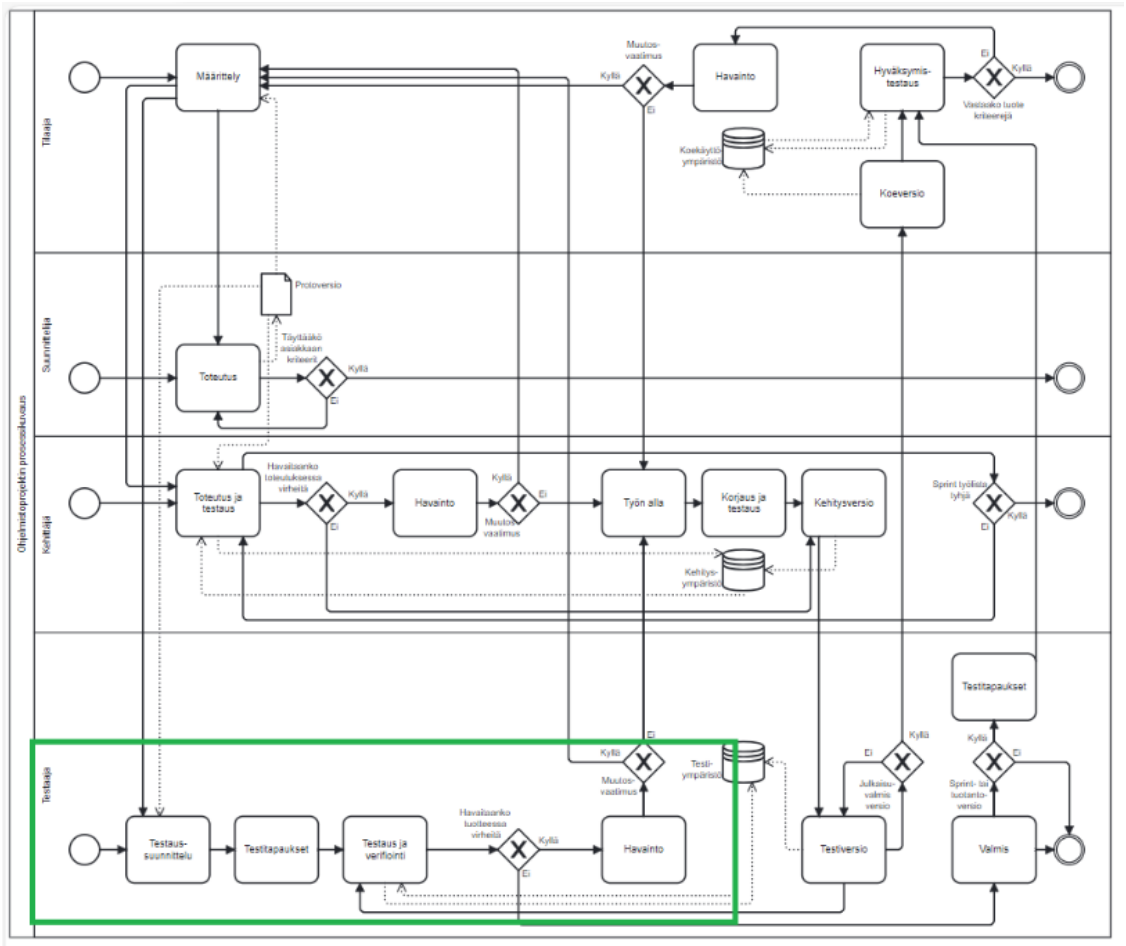
7. Jos sidosryhmien välillä yhteisymmärrys ja osaaminen on heikkoa, sillä on vaikutus projektin aikatauluun (venyy), budjettiin (kasvaa) ja laatuun (heikkenee).

Pitää täysin paikkaansa. Kommunikaation ja yhteisymmärryksen merkitys korostui tutkimustuloksessa tärkeäksi tavoitteeksi sidosryhmien välille. Myös riittävä ammattitaito tai vähintään nopea asioiden sisäistäminen koetaan tiimiläisen tärkeäksi ominaisuudeksi.

8. Kohtien (1–7) vaikutus testauksen kustannusten arviointiin, kustannustehokkuuteen vaikuttaa siten, että kustannusten arviointi vaikeutuu ja kustannustehokkuus kärsii.

Pitää täysin paikkaansa. Kokonaisuuden tulee olla toimiva. Jos yksikin kohta laiminlyödään tai ei anneta sille kuuluvaa painoarvoa, kokonaisuus kärsii.

Tutkimustuloksissa korostuu useasti koulutuksen tarve sekä yhteiset pelisäännöt ja työkalun oikeanlainen käyttö. Parannusehdotuksena voisi olla kehitystiimille uusi rooli tehtävineen, joka huolehtii, että käytännön asiat suoritetaan oikein, kouluttamisesta, analyysien ja raporttien tuottamisesta (kustannus-, edistymis-, ennustusraportit), koska raportit edellyttävät oikeanlaista työkalujen käyttöä. Jos ei kokonaan nimetä uutta roolia, voisi ketterän tiimin keskuudessa kirjata ja sopia työtehtävät tietylle henkilölle, jolloin tehtävään kuluva aika voidaan huomioida myös budjetissa, resurssissa ja aikataulussa. Käytännössä tätä voisi kokeilla niin sanotussa pilottiprojektissa ja verrata projektin edistymistä.



Kuva 31 Vaatimustenhallinnan prosessikuvaus

Kuvassa 31 on jo aiemmin esitetty prosessikuvaus, jossa korostettu vihreällä se osa-alue, jossa jäljitettävyyden tarve ja merkitys korostuu. Kehitysketjun viimeisenä testaaja leimaa asiakirjoihin viimeisimmän tiedon, joten uuden roolin tehtävät kohdistuisivat testauksen osa-alueelle.

Mahdolliset jatkotutkimusaiheet kohdistuisivat varsinaisiin Scrum-menetelmiin kuten sprint-suunnittelupalaverissa käytettävä työmääräarvioinnin menetelmät, miten kehittää tekniikoita kohti luotettavampia työarvioita? Tässä voisi hyödyntää työkalun tarjoamaa metriikka, joka automaattisesti laskee vaatimuksen käsittelyissä käytetyn ajanjakson.

Lopuksi pohdintaa opinnäytetyön perimmäiseen kysymykseen: Miten vaatimusmäärittelyt ovat jäljitettävissä ketterässä projektissa?

Vuosikymmenien tutkimus- ja kehitystyö vaatimushallinnassa on kuluvankin vuoden polttava aihe, ja varmasti tulevienkin vuosien, mutta johtunee muuttuvan ympäristön aiheuttamista muuttuvista vaatimuksista. Tämä osaltaan herättänee muutosten hallinnan piiriin turhauttavan kysymyksen, onko pyörä keksittävä aina uudelleen. Tässä korostuu suunnitelmallisuuden tärkeys kaikissa ohjelmistokehityksen osa-alueilla, jolla voisi olla vuosien päähän ulottuvat katseet. Yhteiskunta on siirtymässä kertakäyttöisestä kulutuksesta vanhan ajan kestävään kehitykseen, joka on kustannustehokkuudenkin kannalta huomionarvoinen ominaisuus myös ohjelmistomaailmassa. Kuten sanonta kuuluu, jotta osaa ennustaa tulevaisuutta, on tunnettava historiaa. Ohjelmistoprojektissa se tarkoittaa vaatimusmäärittelyjen jäljitettävyyttä, joten tässä yhteydessä on huomioitava myös digitaalisten työkalujen elinkaari. Vaatimusten jäljitettävyys on digitaalisten työkalujen varassa, joten projektin alkuvaiheessa on huomioitava myös käytettävän työkalun tulevaisuus, ylläpito, yhteensopivuus, kustannukset ja tiedon siirrettävyys. Siirrettävyydellä tarkoitetaan siirtymistä esimerkiksi toisen valmistajan digitaaliseen työkaluun, joten datan tulisi olla siirrettävissä ja arkistoitavissa uuteen digitaaliseen työkaluun. Tiedon kokonaisuuden säilyvyys ja jäljitettävyys on osa ohjelmistoprojektin kehitystyötä, eikä tällöin tarvitse keksiä pyörää uudestaan, vaan voidaan jäljitettävyyden avulla hyödyntää projekti- sekä projektien välisellä tasolla aiemman työn tuloksia. Tämä helpottaa projektin ylläpidon jatkuvaa kehitystyötä, mutta myös uusien projektien esitutkimustyötä.

Vaatimusmäärittelyn jäljitettävyyden ja uudelleen käytettävyyden edellytyksenä on riittävän laadukas vaatimusmäärittely, luotettava ja riittävän kattava digitaalinen työkalu, dynaaminen ja sitoutunut tiimi yhteisillä ja ymmärrettävillä pelisäännöillä, jossa testauksen rooli on yksi tärkeä avaintekijä.

LÄHTEET

Atlassian. 2024. User Stories with Examples and a Template. Viitattu 11.5.2024. https://www.atlassian.com/agile/project-management/user-stories

Atlassian. 2024. What is backlog grooming? Definition and benefits. Viitattu 4.4.2024. https://www.atlassian.com/agile/project-management/backlog-grooming

Atlassian. 2024. What is the Definition of Done. viitattu 3.4.2024. https://www.atlassian.com/agile/project-management/definition-of-done

Auer, A. Auer, L. Heinäsmäki, M. Hölttö, J. Kalliala, E. Laanti, M. Laine, K. Lekman, L. Miinalainen, P. Naski, H. Piiparinen, T. Puhakka, H. Pyhäjärvi, M. Pääkkönen, T. Räisänen, S. Sora, H. Taipale, M. Talvio, J. Tanninen, A. Toikkanen, T. Toivola, T. Toro, K. Valsta, A. Väyrynen, V. & vo Weissenberg, M. 2013. Ketterää Kehitystä. Finn Lectura

Broekman, B. Koomen, T. van der Aalst, L. & Vroon, M. 2007. TMap Next fro result-driven testing. 2. painos. Alankomaat: UTN Publishers

Dawoud, A. 2014. Traceabilty in model-based testing. Diplomityö. Informaatiotieteiden yksikkö. Tampere: Tampereen yliopisto. Viitattu 4.5.2024. https://urn.fi/URN:NBN:fi:uta-201407102004.

Forselius, P. 2013. Onnistunut tietojärjestelmän hankinta. 3., uudistettu painos. Vantaa: Hansaprint

Haikala, I. & Mäkijärvi, J. 2001. Ohjelmistotuotanto. 7. painos. Pieksämäki: RT-Print Oy

Hara, K. 2013. Ketterä ohjelmistokehitys asiakkaan näkökulmasta. Pro-gradu. Tietojenkäsittelytiede. Jyväskylä: Jyväskylän yliopisto. Viitattu 27.4.2024. http://urn.fi/URN:NBN:fi:jyu-201305221749.

Hokkanen, M. 2001. Requirements Traceability. Diplomityö. Tuotantotalous. Lappeenranta: Lappeenrannan teknillinen korkeakoulu. Viitattu 4.5.2024. https://urn.fi/URN:NBN:fi-fe20011576.

Hou, C. 2021. Asiakkuudenhallintajärjestelmän vaatimusmäärittely. Opinnäytetyö (YAMK). Liiketoiminnan kehittämisen YAMK. Oulu: Oulun ammattikorkeakoulu. Viitattu 27.4.2024. https://urn.fi/URN:NBN:fi:amk-2021112020824.

ISO/IEC/IEEE 29119-1:2022. 2022. Software and systems engineering. Software testing. Part 1: General concepts. Viitattu 8.4.2024. https://www.iso.org/standard/81291.html

ISO/IEC/IEEE 29119-2:2021. 2021. Software and systems engineering. Software testing. Part 2: Test processes. Viitattu 8.4.2024. https://www.iso.org/standard/79428.html

ISO/IEC/IEEE 29119-3:2021. 2021. Software and systems engineering. Software testing. Part 3: Test documentation. Viitattu 8.4.2024. https://www.iso.org/standard/79429.html

ISO/IEC/IEEE 29119-4:2021. 2021. Software and systems engineering. Software testing. Part 4: Test techniques. Viitattu 8.4.2024. https://www.iso.org/standard/79430.html

ISO/IEC/IEEE DIS 29119-5. 2023. Software and systems engineering. Software testing. Part 5: Keyword-Driven Testing. Viitattu 8.4.2024. https://www.iso.org/standard/87233.html

ISO/IEC 25010 2014. International Organization for Standardization. Viitattu 31.3.2024. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

ISO/IEC 25010:2023. 2023. Systems and software engineering. Systems and software Quality Requirements and Evaluation (Squiare). Product quality model. Viitattu 1.4.2024. https://www.iso.org/standard/78176.html

ISO/IEC 25010:2011. 2011. Systems and software engineering. Systems and software Quality Requirements and Evaluation (Squiare). System and software quality models. Viitattu 1.4.2024. https://www.iso.org/standard/35733.html

ISO 9000:2015. 2015. Quality management systems. Fundamentals and vocabulary. Viitattu 2.4.2024. https://www.iso.org/standard/45481.html

ISO 9001:2015. 2015. Quality management systems. Requirements. Viitattu 2.4.2024. https://www.iso.org/standard/62085.html

ISO 9004:2018. 2018. Quality management. Quality of organization. Guidance to achieve sustained success. Viitattu 2.4.2024. https://www.iso.org/standard/70397.html

Grenning, J. 2002. Planning Poker or How to avoid analysis paralysis while release planning. Viitattu 3.4.2024. https://wingman-sw.com/papers/PlanningPoker-v1.1.pdf

Jira Software Support. 2024. Jira Software Data Center 9.15 documentation. Viitattu 4.4.2024. https://confluence.atlassian.com/jirasoftwareserver

Juvonen, R. 2018. Ohjelmistoprojektin sudenkuopat ja miten ne vältetään. Helsinki: BoD - Books on Demand.

Kamath, D. 2023. Improving Agile Development Practices. Opinnäytetyö (YAMK). Tieto- ja viestitekniikka. Helsinki: Metropolia ammattikorkeakoulu. Viitattu 28.4.2024. https://urn.fi/URN:NBN:fi:amk-2023053116827.

Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. 1. painos. Jyväskylä: Docento

Kaukavuori, S. 2000. Requirements Management in the Software Development Process. Diplmomityö. Tuotantotalous. Lappeenranta: Lappeenrannan teknillinen korkeakoulu. Viitattu 28.4.2024. https://lutpub.lut.fi/handle/10024/34463.

Kokko, A. 2013. Improving requirements management practices in agile software development environment. Pro-gradu. Tietojenkäsittely. Helsinki: Haaga-Helia ammattikorkeakoulu. Viitattu 28.4.2024. https://urn.fi/URN:NBN:fi:amk-2013121020729.

Kujala, A. 2001. Achieving Traceability. Diplomityö. Tietotekniikan osasto. Lappeenranta: Lappeenrannan teknillinen korkeakoulu. Viitattu 4.5.2024. https://urn.fi/URN:NBN:fi-fe20011595.

Laamanen, K. 2001. Johda liiketoimintaa prosessien verkkona. Ideasta käytäntöön. Helsinki: Otavan kirjapaino.

Lindvall, K. 2022. Vaatimusmäärittely ketterässä ohjelmistokehityksessä. Pro gradu. Informaatioteknologian ja viestinnän tiedekunta. Tampere: Tampereen yliopisto. Viitattu 28.4.2024. https://urn.fi/URN:NBN:fi:tuni-202204294176.

Luurila, N. 2023. Ketterä ohjelmistokehitys yrityksen, työntekijän ja asiakkaan välisessä interaktiossa. Opinnäytetyö (AMK). Tieto- ja viestintätekniikka. Kajaani: Kajaanin ammattikorkeakoulu. Viitattu 28.4.2024. https://urn.fi/URN:NBN:fi:amk-2023051912219.

Luojus, J. 2020. Toiminnanohjausjärjestelmän vaatimusmäärittely. Opinnäytetyö (YAMK). Johtamisen ja palveluliiketoiminnan koulutusohjelma YAMK. Satakunta: Satakunnan ammattikorkeakoulu. Viitattu 27.4.2024. https://urn.fi/URN:NBN:fi:amk-202005128588.

Paakkari, P. 2022. Ketterä vaatimusmäärittelyprosessi. Opinnäytetyö (AMK). Tietojenkäsittely koulutus. Hämeenlinna: Hämeen ammattikorkeakoulu. Viitattu 27.4.2024. https://urn.fi/URN:NBN:fi:amk-202205047098.

Pelin, R. 2011. Projektihallinnan käsikirja. 7. uudistettu painos. Keuruu: Otavan kirjapaino Oy.

Ruuska, K. 2012. Pidä projekti hallinnassa. Suunnittelu, menetelmät, vuorovaikutus. 7. painos. Vantaa: Talentum Media Oy.

SFS-EN ISO 9000. 2005. Laadunhallintajärjestelmät. Perusteet ja sanasto. 2. painos. Helsinki: Suomen standardisoimisliitto SFS

Saarinen, M. 2016. Vaatimushallintaprosessin uudistaminen. Opinnäytetyö (AMK). Tietojenkäsittelyn koulutusohjelma. Helsinki: Haaga-Helia ammattikorkeakoulu. Viitattu 27.4.2024. https://urn.fi/URN:NBN:fi:amk-2017090314727.

Salmi, J. 2020. Vaatimusmäärittely ketterässä ohjelmistokehityksessä. Kandidaatintyö. Tieto- ja sähkötekniikan tiedekunta. Tampere: Tampereen Yliopisto. Viitattu 27.4.2024. https://urn.fi/URN:NBN:fi:tuni-202005044900.

Sammons, A. 2019. Agile Project Management With Scrum & Kanban 2 In 1. The Last Two Approaches You'll Need To Become More Productive And Meet Your Project Goals. M M Limitless Online Inc.

Terho, K. Vilpola, I. 2008. Tehokkuutta tuotannon tietojärjestelmiin – loppukäytttäjät mukaan määrittelyyn. Helsinki: Kopioniini Oy

Tikka, P. 2015. Requirements traceability in simulation driven mechanical engineering. Diplomityö. Tuotekehitys. Tampere: Tampereen teknillinen yliopisto. Viitattu 4.5.2024. https://urn.fi/URN:NBN:fi:tty-201604203842.

The Scrum Guide 2020. The Definitive Guide to Scrum: The Rules of the Game. Julkaistu 2020. Viitattu 14.3.2024. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

TMap. 2016. Workbook TMap Suite. Studying for the TMap Suite Certification. Viitattu 17.4.2024. https://www.tmap.net/sites/tmap/files/files/TMAP_Suite_Workbook_1.00_ENG.pdf

Venäläinen, H. 2018. Asiakaskeskeinen ohjelmistokehitys. Kandidaattitutkielma. Tietojärjestelmätiede. Jyväskylä: Jyväskylän yliopisto. Viitattu 28.4.2024. http://urn.fi/URN:NBN:fi:jyu-201901171225

Xray. 2023. Requirements and Defects. Viitattu 17.5.2024. Requirements and Defects - XRAY 7.4.0 - Yleinen sivusto (getxray.app)