HAAGA-HELIA
ammattikorkeakoulu

Tuotekehitysprosessin kehittäminen laadunvarmistuksen näkökulmasta

Jouko Holappa

Opinnäytetyö
Liiketalouden ylempi ammattikorkeakoulututkinto
Tietojärjestelmäosaamisen koulutusohjelma
2012

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

7.11.2012

Tietojärjestelmäosaamisen koulutusohjelma

Tekijä tai tekijät Jouko Holappa	**Ryhmätunnus tai aloitusvuosi** YTI11K
Raportin nimi Tuotekehitysprosessin kehittäminen laadunvarmistuksen näkökulmasta	**Sivu- ja liitesivumäärä** 126 + 21
Opettajat tai ohjaajat Matti Kurki, Janne Vuoti	

Tämä opinnäytetyö käsittelee ohjelmistotuotekehitystä tekevän tiimin tuotekehitysprosessin kehittämistä laadunvarmistuksellisesta näkökulmasta.

Ohjelmistokehityksessä virheet ja muut laatuongelmat voivat aiheuttaa merkittävää haittaa ja ylimääräistä työtä. Puutteet ohjelmistojen laadussa pakottavat yritykset panostamaan laatuun monella eri tavalla. Ohjelmistojen laatuun voidaan vaikuttaa ohjelmistojen elinkaaren eri vaiheissa monin eri tavoin. Nykyaikaiset ja kehittyneet ketterät menetelmät tarjoavat hyvän lähtökohdan laadukkaiden ohjelmistojen tuottamiseen ketterästi toimimaan kykenevissä tiimeissä ja organisaatioissa.

Ketterät menetelmät eivät kuitenkaan yksin pysty takaamaan laadullisesti hyviä ohjelmistoja. Laadun tuottamiseksi tulee sekä laadunvarmistuksen että ohjelmistokehitykseen ja laadunvarmistukseen liittyvien prosessien olla kunnossa. Sopivilla ja joustavilla prosessien kehittämisen menetelmillä voidaan tukea ketterien kehitysmenetelmien päämääriä laadullisesti hyvien ohjelmistojen valmistamiseen ja päinvastoin.

Kehittämistehtävässä prosessin kehittämisen viitekehyksenä käytettiin soveltaen CMMI-DEV 1.3:a sen joustavuuden vuoksi, sekä sen vuoksi että se soveltuu hyvin käytettäväksi ketterien menetelmien kanssa. Kehitysprojektissa havaittiin, että laadulliset ongelmat ovat usein seurausta puutteellisista laadunvarmistuksen toimenpiteistä. Lisäksi projektissa havaittiin, että laadullisiin ongelmiin voidaan vaikuttaa pienilläkin toimenpiteillä. Jatkuva laadullisesti hyvien ohjelmistotuotteiden tuottaminen sen sijaan vaatii hyviä toistettavia prosesseja, jotta laatu säilyy.

CMMI-DEV 1.3 osoittautui hyväksi prosessin kehittämisen suuntaviivoja antavaksi viitekehykseksi pienessä ohjelmistokehitystyötä tekevässä tiimissä. Viitekehyksessä on selvästi huomioitu kohdeympäristön tarpeet ja alan parhaat käytännöt, joita on hyödynnetty prosessien kehittämisen ohjaamisessa. Kehittymisen ja laadun mittareiden osalta havaittiin, että mittareiden valinnassa ja määrittelyssä on oltava hyvin kriittinen. Näennäisesti hyvillä mittareilla ei välttämättä saada tarkoituksenmukaisia mittaustuloksia, eikä selviä syy-seuraussuhteita pystytä esittämään tutkittavasta ilmiöstä.

Asiasanat
ketterät menetelmät, CMMI-DEV, laatu, laadunvarmistus, mittaus

HAAGA-HELIA
University of Applied Sciences

Abstract

7.11.2012

Masters Degree programme in Information Systems Management

Authors Jouko Holappa	**Group or year of entry** YTI11K
The title of thesis SOFTWARE PROCESS IMPROVEMENT FROM QUALITY ASSURANCE POINT OF VIEW	**Number of pages and appendices** 126 + 21
Supervisor(s) Matti Kurki, Janne Vuoti	

This thesis discusses the improvement of a software process within a small software development team from the perspective of quality assurance.

Errors, defects and other quality issues can cause significant harm and excessive work in software development teams. Flaws in software quality force organizations to invest in quality in many ways. Software quality can be affected in many different ways in different points of software's life cycle. Modern and advanced agile development methods provide a good basis for good quality software development in agile-ready organizations.

However, agile methods alone cannot guarantee good quality software. For good quality development, software development processes and quality assurance processes must be in good shape. The goal of agile development methods of providing good quality software can be supported with appropriate and flexible process development frameworks and vice versa.

In this development project, CMMI-DEV 1.3 was used as a framework for process development because of its flexible nature. It also fits well to be used with agile development methods. In the development project, it was found out that quality issues often result from lacking quality assurance activities. The study also indicated that quality can be affected with relatively small changes in quality assurance activities. Continuous development of good quality software requires good repetitive processes in order to maintain the quality.

CMMI-DEV 1.3 framework proved to provide good guidelines for process development in a small software development team. The target environment was clearly taken into account in the framework development. Best practices in software development were made use of in the framework development to guide the process development. What comes to improvement and quality metrics, the study showed that one should be very critical when defining the metrics to be used. Metrics may be seemingly good, but the measures gained may not be appropriate at all and metrics may not be able to provide clear cause-and-effect relation to the examined phenomenon.

Key words
agile development, CMMI-DEV, quality, quality assurance, measurement

Sisällys

Kuviot

Taulukot

1 Johdanto

Ohjelmistojen laadunvarmistukseen liittyviin prosesseihin ja niiden kehittämiseen on olemassa jo valmiita prosessimalleja, mutta niiden ongelmana on pääsääntöisesti liiallinen yleisyys. Valmiit mallit eivät ota riittävällä tasolla huomioon eri organisaatioiden välisiä eroja. Tämän johdosta valmiita malleja pystytään harvoin soveltamaan sellaisenaan ja mallit vaativat räätälöintiä organisaation tarpeita vastaavaksi. Mallien liiallisen yleisyyden vuoksi prosessi joudutaankin usein kehittämään itse. Kehitettäessä laadunvarmistukseen liittyviä prosesseja, tärkeänä kehittämisen lähtökohtana voidaan pitää prosessien joustavuutta, jotta prosessit voidaan tarpeen mukaan käyttää uudestaan joko toisissa projekteissa tai jopa kokonaan toisissa organisaatioissa. Uudelleenkäytettävän mallin avulla laadunvarmistukseen kuluva aika pienenee, kun samaa mallia voidaan käyttää aina uusien tuotejulkaisujen yhteydessä uudestaan. (Lazic, Kolašinac & Avdic 2009, 23-24.)

1.1 Kohdeorganisaatio ja kehitysprojektin lähtökohdat

Tämä opinnäytetyönä tehty kehitysprojekti toteutettiin ohjelmistokehitystä tekevässä tuotekehitystiimissä, joka koostui kolmesta täysipäiväisestä työntekijästä. Lisäksi tiimin työssä oli mukana konsultti, joka organisatorisesti on tiimin ulkopuolinen, mutta työskenteli samoissa tehtävissä muiden tiimin jäsenten kanssa. Tiimin kokonaisvahvuus oli näin ollen neljä henkilöä.

Kehittämistehtävän tarkoituksena oli kehittää ohjelmistokehitystä tekevän tiimin tuotekehitysprosessia laadunvarmistuksen näkökulmasta. Projektin alkutilanteessa tiimin prosessi oli säilynyt muuttumattomana pitkän aikaa. Muuttumattomaan tilanteeseen haluttiin etsiä keinoja parantaa tuotekehitysprosessia uusin menetelmin siten, että uusi menetelmä soveltuisi luontevasti ja joustavasti käytettäväksi tuotekehitystiimissä käytössä olevien menetelmien kanssa.

Kehitysprojektin alkuvaiheessa tuotekehitystiimin prosessi oli säilynyt hyvin stabiilina ja käytännössä täysin muuttumattomana pitkän aikaa. Tämä tarkoitti sitä, ettei prosessi ei ollut kehittynyt ollenkaan, joskaan ei myöskään huonontunut. Prosessin muuttumatto-

muus on ketterien menetelmien yhtä perusideaa vastaan, jonka mukaan prosessia pyritään kehittämään jatkuvasti. Prosessin muuttumattomuus voi pidemmän ajan myötä johtaa ohjelmistokehityksessä laadullisiin ongelmiin ja ohjelmistotuotteiden laadun heikkenemiseen. Tähän ongelmaan haluttiin kohdeorganisaation tuotekehitystiimissä reagoida, jo ennen kuin ongelmat esiintyvät.

1.2 Tavoitteet ja tutkimuskysymykset

Tuotekehitysprosessin muuttumaton tila asetti lähtökohdat tälle tutkimukselle. Tutkimuksen tavoitteena oli kehittää tuotekehitystiimin laadunvarmistusta sekä laadun mittauksen, että prosessin kannalta. Tutkimusongelma jakautui kahteen tutkimuskysymykseen seuraavalla tavalla.

Tutkimuskysymys 1: Mikä on tuotekehitysprosessin nykytilanne ja mitkä ovat sen ongelmakohdat?

Kysymyksen tavoitteena on kartoittaa alkutilanne ja luoda pohja tutkimustyölle ja sen myötä koko kehitysprojektille. Arvioimalla prosessin ongelmakohtia alkuvaiheen kartoituksen yhteydessä, voidaan asettaa suuntaviivoja projektille sen alkuvaiheessa. Tehtyjen havaintojen perusteella projektia voidaan ohjata havaittujen ongelmien avulla kehityksen kannalta oikeaan suuntaan.

Tutkimuskysymys 2: Minkälaisia prosessin muutos- ja parannusehdotuksia kirjallisuus ja havainnot tukevat? Miten prosessia voidaan kehittää laadunvarmistuksen näkökulmasta teorian mallien avulla ja miten sitä voidaan arvioida?

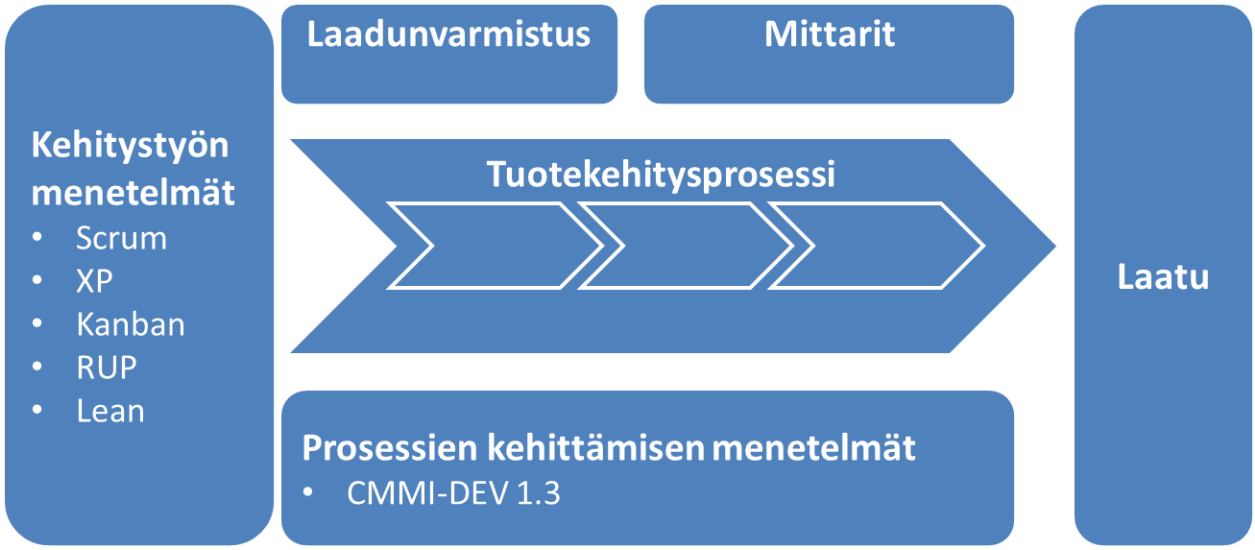
Kysymyksen tavoitteena on luoda perusta tuotekehitysprosessin kehittämiselle prosessin lähtökohtaan nähden. Prosessin kehittämistä on tarkoitus tutkia sekä prosessin ongelmien näkökulmasta, että myös kokonaisvaltaisemmin. Tavoitteena on myös luoda teoriaperusta prosesseista ja malleista, jotka liittyvät kiinteästi kohdeorganisaation toimintaan. Samalla kysymyksen tavoitteena on rakennetun teoriapohjan ja havaintojen avulla selvittää millä keinoin tuotekehitysprosessin suoritusta voidaan mitata. Mittareilla

pyritään seuraamaan myös laadunvarmistuksen toimenpiteiden vaikutusta prosessiin ja siinä havaittuihin puutteisiin.

1.3 Tutkimuksen viitekehys

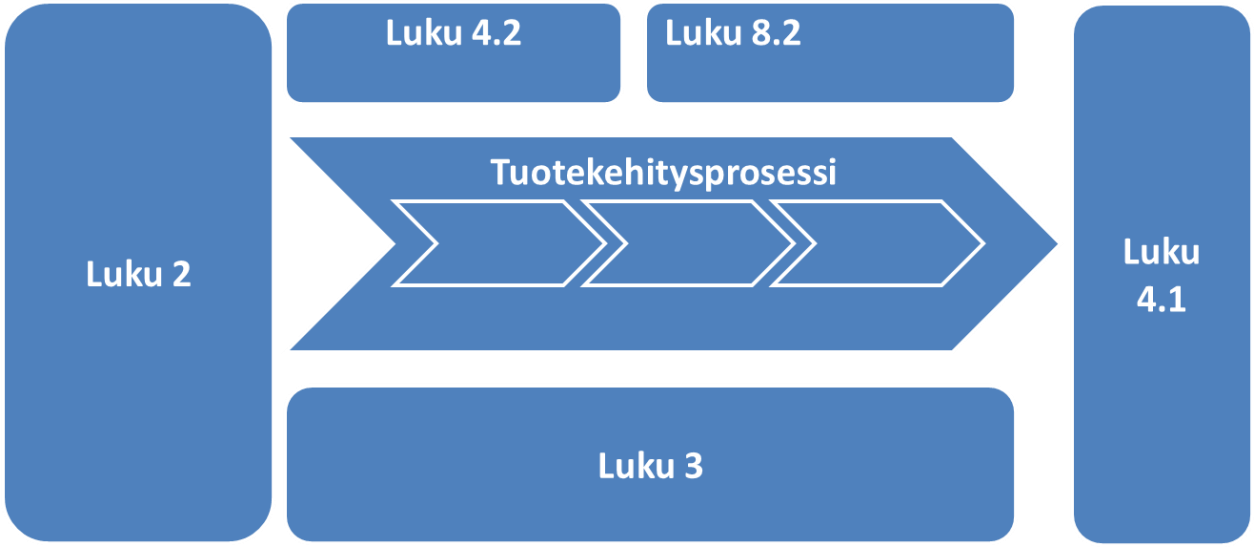
Tutkimusten yhteydessä käsitellään usein tutkimuksen teoreettista viitekehystä. Toisinaan asiasta käytetään myös termejä teoriatausta tai teoriakatsaus. Teoreettisen viitekehyksen tehtävänä on luoda pohja tutkimukselle, sekä ohjata tutkimuksen käytännön toteutusta. Viitekehys sisältää tutkimuksen aiheen kannalta keskeiset ja toisiinsa liittyvät asiat joiden ympärille tutkimus rakennetaan. Usein viitekehys muodostuu useista eri näkökulmista käsiteltävään asiaan tai ongelmaan. Tällöin viitekehys koostuu toisiinsa kietoutuneista näkökulmista, joista jokin voi olla merkitykseltään tai painoarvoltaan suurempi. Viitekehyksen tulisi kytkeytyä tutkimusongelmaan jolloin tutkimuksen teoria ja empiria kytkeytyvät toisiinsa. Usein viitekehys muodostuu käytännössä aiheesta julkaistun kirjallisuuden, artikkeleiden tai tieteellisten keskusteluiden varaan. (Saaranen-Kauppinen & Puusniekka 2009, 10.)

Tämän tutkimuksen keskiössä on ohjelmistokehitystä tekevän tiimin tuotekehitysprosessin kehittäminen. Prosessin kehittämisen näkökulmana on laadunvarmistuksen kehittäminen prosessissa. Laadunvarmistuksen kehittämistä lähestytään kahdesta näkökulmasta. Ensimmäinen näkökulma laadunvarmistuksen kehittämistä on se, mitä keinoja ohjelmistokehityksen eri työmenetelmät tarjoavat laadukkaampien ohjelmistotuotteiden tuottamiseen. Toisena näkökulmana ovat prosessien kehittämisen menetelmät ja millä tavalla ne voivat auttaa päämärän saavuttamisessa. Erilaiset mittarit tarjoavat työkaluja laadun sekä myös prosessien seurantaan ja valvontaan. Kuvio 1 esittää mistä asioista tämän tutkimuksen viitekehys koostuu.



Kuvio 1. Tutkimuksen viitekehys kuvana

Kuvio 2 esittää miten tutkimuksen viitekehys on sijoittunut tämän opinnäytetyön sisällä.



Kuvio 2. Tutkimuksen viitekehityksen sijoittuminen opinnäytetyön sisällä

1.4 Menetelmät

Tutkimuksen tekemisessä käytettiin kirjallisuuskatsausta tutkimuksen teoriaosuutta varten. Teoriaosuuden lisäksi tutkimuksessa on myös empiirinen osuus, jossa hyödynnettiin teorian, sekä havaintojen perusteella kerättyjä tietoja sekä sovellettiin niitä kohdeorganisaation käyttöön. Opinnäytetyön tehnyt, tutkijan roolissa ollut opiskelija, oli kiinteästi mukana kohdeorganisaation päivittäisessä toiminnassa. Näin ollen varsinaisena tutkimusmenetelmänä käytettiin toimintatutkimusta.

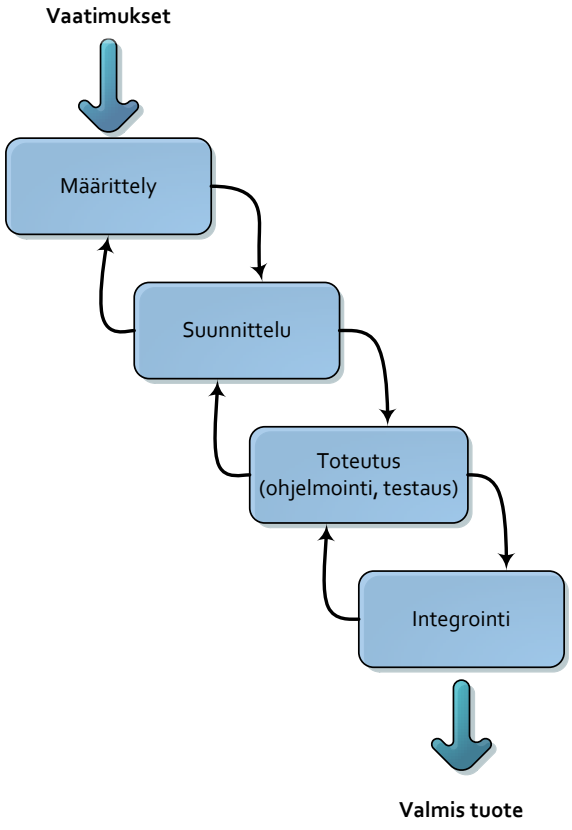
1.5 Rajaus

Tämän työelämän kehittämistehtävän piiriin kuuluvat vain tuotekehitystiimin tuotekehitys- ja laadunvarmistusprosessit. Kehittämistehtävässä on rajattu pois kaikki muut organisaation tiimit, yksiköt ja organisaatiotasot.

2 Kehitystyön menetelmät

2.1 Vesiputous

Perinteisesti ohjelmistoja ja tietojärjestelmiä on toteutettu vesiputousmallin mukaisesti. Vesiputousmallissa järjestelmäkehityksen eri vaiheet seuraavat toinen toisiaan edeten aina alusta lähtien määrittelyn, suunnittelun ja toteutuksen kautta testaukseen. Tämän jälkeen järjestelmän pitäisi olla käytännössä valmis. Ongelmia tulee vastaan, mikäli määrityksissä havaitaan virheitä tai puutteita. Tällöin koko kehitysprosessi on suoritettava uudestaan liki alusta lähtien. Muutosten myötä vesiputousmallin mukaan toteutetut projektit usein viivästyvät aikataulustaan. Tämä saattaa johtaa tilanteeseen, jossa järjestelmä tai ohjelmisto on jo vanhentunut ja tarpeeton sen valmistuessa, kun liiketoimintaympäristö on ehtinyt muuttua merkittävästi kehitystyön aikana. (Sommerville 2006, 392).



Kuvio 3. Ohjelmistokehityksen vesiputousmalli (Kruchten 2001, 54)

Vesiputousmalli voi olla toimiva lähestymistapa ohjelmistoprojekteihin joissa kokonaisuus on hyvin selkeä ja hallittu. Se edellyttää että tuleva toteutus tai kohdeympäristö

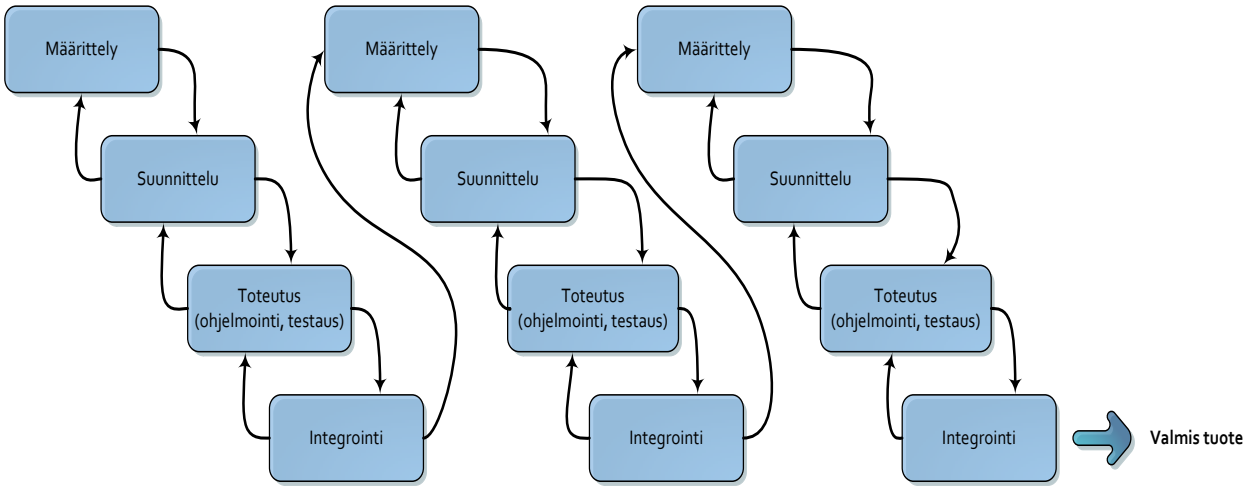
ymmärretään kaikilta osin, eikä yllätyksiä ole odotettavissa. Yleensä tällaiset ehdot pätevät vain hyvin pienissä tai aikataulullisesti hyvin lyhyissä projekteissa. Mitä monimutkaisemmasta tai laajemmasta järjestelmästä on kyse, sitä enemmän on myös epävarmuustekijöitä. Tällöin riski epäonnistua kehitysprojektissa käyttäen vesiputousmallia kasvaa merkittävästi. (Kruchten 2001, 57-58.)

Suuremmissa kehitysprojekteissa muutokset ovat usein hitaita toteuttaa vesiputousmallilla kehitettäessä. Hitauden lisäksi muutosten kustannukset nousevat helposti korkeiksi. Näiden vesiputousmallissa olevien puutteiden vuoksi ketterät menetelmät ovat entistäkin tarpeellisempia ja hyödyllisempiä ohjelmistokehityksessä. Ketterien menetelmien yhtenä päämääränä on pienentää muutoksista aiheutuvia kustannuksia ja siten laskea kokonaiskehityskustannuksia. (Dumke, Kunz & Schmietendorf, 2008, 95-96.)

2.2 Ketterät menetelmät

Ketterien menetelmien etuna perinteiseen vesiputousmallin mukaiseen ohjelmistokehitykseen on se, että niiden tavoitteena on tuottaa käytettäviä ja hyödyllisiä ohjelmistoja nopeasti ja usein. Ketterissä menetelmissä kehitystyön prosessi on iteratiivinen, missä määrittely, suunnittelu, toteutus ja testaus ovat sidoksissa toisiinsa, eikä niiden ole ajateltu olevan toisistaan eroteltuja itsenäisiä vaiheita kuten vesiputousmallissa. Ohjelmistoja ei kehitetä ja toimiteta kerralla kokonaisuudessaan, vaan kehitystyö on inkrementaalista (incremental) eli ajan myötä kasvavaa, jolloin jokainen inkrementti (increment) eli tuoteversio sisältää uusia ominaisuuksia tai uutta toiminnallisuutta. (Sommerville 2006, 392)

Periaatteessa iteratiivisessa kehityksessä on tavoitteena toteuttaa suuret asiat pienissä osissa. Jaettaessa projekti pienemmiksi osiksi tai pienemmiksi kokonaisuuksiksi, voidaan myös suuria monimutkaisia projekteja koskevat riskit jakaa pienemmiksi ja täten pienempää osakokonaisuutta koskeviksi. Iteratiivisen kehityksen syklien – eli iteraatioiden – voidaan ajatella olevan myös minivesiputouksia, sillä iteraatioissa toistuvat samat vaiheet kuin vesiputousmallissa mutta pienemmässä mittakaavassa. (Kruchten 2001, 60; Kniberg & Skarin 2010, 3-4.)



Kuvio 4. Iteratiivinen ohjelmistokehitysmalli (Kruchten 2001, 61)

Ajan myötä kasvavassa kehityksessä myös asiakkaan edustus on mukana kehitystyössä ideoimassa kehitystarpeita tulevia kehitysjaksoja varten. Asiakasnäkökulman avulla ohjelmistosta saadaan varmemmin sellainen kuin halutaan. Loppukäyttäjien ottamisella mukaan kehitysprosessiin, voidaan heitä sitouttaa lopputuloksena syntyvään ohjelmistoon ja sen käyttöön. (Sommerville 2006, 392-393)

Yhtenä etuna iteratiivisessa kehityksessä voidaan pitää sitä, että tehtävien priorisointia voidaan muuttaa kulloinkin tärkeimmäksi arvioidun ominaisuuden suuntaan. Vaatimukset muuttuvat useissa kehitysprojekteissa moneen kertaan projektin aikana. Iteratiivinen kehitys mahdollistaa nopeamman reagoinnin muuttuneisiin vaatimuksiin, kuin vesiputousmalli. Samoin mahdollisiin esiin tuleviin ongelmiin voidaan reagoida nopeasti. Myös testauksen taakka jakaantuu tasaisemmin kuin vesiputousmallissa, koska testaus on jatkuvaa. (Kruchten 2001, 8, 25.) Jatkuvan testauksen seurauksena iteratiivisesti kehitetyt ohjelmistot ovat yleisesti ottaen laadullisesti parempia, kuin vesiputousmallin mukaisesti tehdyt ohjelmistot. Testauksen lisäksi määritykset sekä toteutus on voitu tehdä niin hyvin vastaamaan lopullisia käyttäjätarpeita, kuin se on mahdollista. Siinä vaiheessa kun perinteisesti vesiputousmallin mukainen lopullinen käyttöönottovaihe tulee vastaan, on iteratiivisesti kehitetty ohjelmisto ollut käytössä jo pitkän aikaa. (Kruchten 2011, 78.)

Ketterissä menetelmissä pyritään välttämään massiivisia organisaation yksiköihin ja toiminnallisuuksiin perustuvia hierarkioita. Sen sijaan pyrkimyksenä on muodostaa tii-

meistä organisaation yksiköiden rajat ylittäviä toiminnallisuuksia tehostamaan päätösten tekoa ohjelmistokehityksessä. (Benefield, Deemer, Larman & Vodde, 2010, 4.)

Hyödyistään huolimatta eivät ketterät kehitysmenetelmätkään ole täysin ongelmattomia. Vaikka ketterillä menetelmillä pyritään vastaamaan muuttuviin vaatimuksiin, voi niistä koitua ongelmia. Jatkuvien muutosten seurauksena esimerkiksi dokumentaation ylläpito voi käydä hyvin työlääksi tai jopa mahdottomaksi. Samalla jatkuvat muutokset voivat altistaa ohjelmakoodin laadulliselle heikkenemiselle. Monien muutosten myötä alkuperäinen rakenne saattaa rikkoutua, jolloin muilla kuin kehitystyössä alusta asti mukana olleilla henkilöillä voi olla vaikeuksia ymmärtää mistä on kyse tai kuinka ohjelmiston tulisi toimia. Muuttuvat vaatimukset ja sitä myötä muuttuva ohjelmistotoimituksen sisältö voivat aiheuttaa ongelmia myös hallinnolliselta kannalta asiaa katsottaessa. Voi olla mahdotonta tehdä sopimusta siitä mitä toimitus pitää sisällään, kun määritykset muuttuvat jatkuvasti. Toimittajan kannalta asiaa katsottaessa voi kiinteähintaisen projektin sopiminen olla hankalaa, varsinkin mikäli muutoksia ei kontrolloida riittävästi ja hallitusti. Tilaaja ei puolestaan välttämättä ole halukas suorittamaan maksua vain sen perusteella minkä verran aikaa kehittäjät ovat työhön käyttäneet. (Sommerville 2006, 393-394)

2.3 Scrum

Scrum on ketteristä menetelmistä suosituin. Kehitys tapahtuu ketterien periaatteiden mukaisesti iteratiivisesti ja jatkuvasti kasvavasti sprinteissä. Scrumin mukaisesti sprintit kestävät tyypillisesti 2-4 viikkoa. Sprintiä ei jatketa, vaikka kaikkia ominaisuuksia ja toiminnallisuuksia ei ole valmiiksi sprintin loppuun mennessä. Sprintti on aina määrätyn mittainen, ei koskaan pidempi. (Benefield ym. 2010, 4; Kniberg & Skarin 2010, 13; Scrum Alliance 2011.) Käytännössä sprintin pituudelle ei ole yhtä yleistä kaikille yhtä hyvin sopivaa pituutta. Sopiva pituus sprintille löytyy yleensä kokeilemalla pituudeltaan vaihtelevia ajanjaksoja. Sprintin pituutta miettiessä on kuitenkin syytä pitää mielessä se, että pituuden on oltava riittävän lyhyt jotta organisaation tai tiimin ketteryys ja mahdollisuus nopeisiin suunnan muutoksiin säilytetään. Samalla sprintin pituuden on oltava kuitenkin riittävän pitkä, jotta scrumtiimi pääsee sujuvaan työskentelytilaan työssään, eikä sprintti katkea liian aikaisin. Sprintin aikana työn tavoitteellisuutta voidaan lisätä

asettamalla sprintille tavoite. Tavoite voi olla esimerkiksi saada beta-julkaisu aikaiseksi. Tavoitteen on perusperiaatteeltaan kuitenkin asetettava päämäärä sille miksi sprintiä ollaan suorittamassa. (Kniberg 2007, 20.)

Scrumin periaatteista tärkein on ”tutki ja mukaudu” (inspect and adapt). Periaatteen ajatus on se, että kehitystyö vaatii jatkuvaa oppimista. Periaatteen mukaiseen tavoitteeseen pyritään sillä, että kehitystyötä tehdään riittävän pienin askelin, jolloin voidaan arvioida uusien tapojen ja teknologioiden mahdollisuuksia ja hyötyjä. Samalla niistä voidaan tehdä tarpeen mukaan tiimin pysyviä käytäntöjä ja pysyvästi käytössä olevia teknologioita. (Benefield ym. 2010, 5.) Tämän seurauksena scrumin prosessi on jatkuvasti kehittyvä ja optimoituva. Näin on varsinkin silloin, mikäli jokaisen sprintin jälkeen pidetään scrumtiimin kesken retrospektiivi (retrospective), jonka nimenomaisena tarkoituksena on kehittää prosessia jatkuvasti ja arvioida nykyisiä käytäntöjä sekä niiden toimivuutta ja tarkoituksenmukaisuutta. (Kniberg & Skarin 2010, 4.)

Vaikka scrumissa on tiettyjä periaatteita joiden mukaan toimitaan, ei scrum ole varsinaisesti viitekehys, joka kertoisi mitä pitää tehdä tai miten asiat pitäisi tehdä. Näin ollen ei ole myöskään yhtä ainoaa oikeaa tapaa toimia scrumin mukaisesti. Scrumin käytännön toteutukset voivat vaihdella paljonkin organisaatiosta tai jopa tiimistä riippuen. Yksi tapa toimia jossain scrumtiimissä ei välttämättä toimikaan toisessa tiimissä. (Kniberg 2007, 7-8.)

2.3.1 Roolit

Scrum pitää sisällään kolme roolia: tuotteen omistaja (product owner), kehitystiimi (team) ja scrummaster (Scrum Master), jotka yhdessä muodostavat scrumtiimin (Scrum Team). Scrummasterin tehtävänä on tukea sekä tiimiä että tuotteen omistajaa työssään. Hänen vastuullaan on varmistua siitä, että kaikilla osapuolilla on yhtäläinen näkemys siitä, mitä ollaan tekemässä ja minkä tulee olla lopputulos. Keskeistä scrummasterin roolissa on varmistaa tiimille mahdollisuus työskennellä täysipainoisesti ilman ulkopuolisia häiriöitä. Pienissä scrumtiimeissä joku tiimin jäsen voi toimia scrummasterin roolissa ja suuremmissa tiimeissä voi olla täysipäiväinen scrummaster, joka ei osallistu itse kehitystyön tekemiseen. Scrummasterin rooli ei ole toimia projektipäällikkönä ja osoit-

taa tehtäviä tiimin jäsenille. Scrumissa tiimi on itseohjautuva ja jakaa tehtävät itsenäisesti tiimin jäsenten kesken. (Benefield ym. 2010, 6-7.)

Tuotteen omistajan vastuulla on tunnistaa kehitettävän tuotteen ominaisuudet, ylläpitää tuotteen kehitysjonoa (backlog) ja priorisoida ne sen mukaan, mikä on kulloinkin tärkeintä ja millä saavutetaan suurin hyöty. Tuotteella saavutettava hyöty on hyvin epämääräinen termi ja voidaan määritellä useallakin tavalla. Toisaalta hyödyn voidaan nähdä olevan se millä saavutetaan suurin myynti ja toisaalta sisäisten tuotteiden kohdalla se, millä voidaan tehostaa toimintaa parhaiten. Vaikka tuotteen omistaja ei välttämättä ole osa scrumtiimiä, on tuotteen omistaja kuitenkin tiiviissä yhteistyössä scrumtiimin kanssa. Käytännössä tuotteen omistaja voi olla myös tiimin jäsen ja tarpeen mukaan antaa esimerkiksi priorisoinnin tiimin tehtäväksi. (Benefield ym. 2010, 5-6; Forss 2009, 37-43; Kniberg 2007, 76-77; Schwaber & Sutherland 2011, 5; Scrum Alliance 2011.)

Tuotteen omistajan rooli voidaan myös yhdistää scrummasterin rooliin, mutta siinä on vaaransa ja roolien yhdistäminen voi aiheuttaa sekaannuksia. Erityisenä vaarana ja sekaannuksen aiheuttajana voi olla se, ettei tiimi tiedä esittääkö yhdistettyjen roolien edustaja asiaansa scrummasterin, vai tuotteen omistajan kannalta. Suositeltavaa olisi siis pitää roolit erillään toisistaan. (Forss 2009, 40-41.) Käytännön työssä scrummaster joutunee aika ajoin vastakkain tuotteen omistajan kanssa, mikäli tuotteeseen yritetään sisällyttää uusia ominaisuuksia kesken sprintin. Tällaisessa tilanteessa scrummasterin on roolinsa mukaisesti suojattava kehitystiimiä ulkopuolisilta ärsykkeiltä ja mahdollistaa kehitystiimille työskentely sprintin suunnitelman mukaisesti. (Benefield ym. 2010, 7.)

Kehitysiimin vastuulla on varsinainen toteutustyö tuotteen omistajan ohjeiden mukaisesti. Kehitysiimi itsessään on autonominen ja käytännössä voi määrätä itse hyvin pitkälle kuinka jokin asia toteutetaan. Useimmiten scrumtiimin koko vaihtelee 5-10 henkilöön, mikä tarkoittaa sitä että suurempien tuotteiden tai järjestelmien yhteydessä voi tiimejä olla useampia. Yksi tiimi voi olla vastuussa rajapintojen suunnittelusta, yksi tiimi tietokantasuunnittelusta ja niin edelleen. Pienemmissä kokonaisuuksissa yksi tiimi vastaa käytännössä kaikista osa-alueista ja sellaisia tiimejä kutsutaan myös ominaisuustiimeiksi (feature team). (Benefield ym. 2010, 6.)

Scrumtiimeissä voi olla roolina myös pääsuunnittelija (technical lead). Siinä missä scrummaster ei välttämättä ota kantaa teknisiin asioihin, on pääsuunnittelijan nimenomaisena roolina olla teknisten asioiden visionääri. Hänen vastuullaan on varmistaa se, että tiimillä on yhtenäinen näkemys käytettävistä teknologioista ja toteutustavoista joilla ohjelmisto tullaan tekemään. Pääsuunnittelija ei ole roolinsa puolesta niinkään kiinnostunut itse prosessista ja sen kehittämisestä, kuin lopputuotteesta. Pääsuunnittelija on siis toteuttaja siinä missä tiimin muutkin jäsenet. Scrummaster ei sitä vastoin ota välttämättä itse osaa varsinaiseen kehitystyöhön. (Singleton 2010.)

Pääsuunnittelijan roolina on toimia myös eräänlaisena opastajana kehitystiimille varmistaen, että tiimin jäsenten osaaminen kehittyy. Pääsuunnittelijan tehtävänä on poistaa teknisiä esteitä, joita kehitystyössä voi tulla vastaan ja toimia keskustelun edistäjänä erilaisten ratkaisumallien löytämiseksi. Vaikka pääsuunnittelijan tuleekin ajoin varmistaa ja tarpeen tullen muistuttaa tiimiä päämäärästä johon ollaan pyrkimässä, ei hän ole kuitenkaan yksinään vastuussa esimerkiksi koodikannasta. Myös vaikeampia päätöksiä kohdattaessa on pääsuunnittelijan voitava luottaa tiimin apuun päätösten teossa. (Trindale 2009.)

Scrummasterin rooli voidaan myös korvata pääsuunnittelijan roolilla, jolloin pääsuunnittelijan tehtäväksi jää scrumin vetäminen. Pääsuunnittelijan roolin lisäksi scrumiin voidaan tarpeen mukaan tuoda lisärooleja tarpeen mukaan, määrittelemällä roolit esimerkiksi teknisille kirjoittajille, tiimin vetäjille ja testaajille. (Ergin 2011.)

2.3.2 Tehtäväpisteet ja vauhti

Scrumissa ominaisuuksia (story) ja muita kehitystyön tehtäviä pisteytetään tiettyjen periaatteiden mukaisesti. Ominaisuuden monimutkaisuudesta, laajuudesta tai muista asioista riippuen eri ominaisuuksilla ja tehtävillä voi olla eri määrä tehtäväpisteitä (story point). Yksi tapa pisteyttää tehtävät on käyttää fibonaccin lukusarjaa, jossa hyvin yksinkertainen ja helppo ominaisuus saa arvon 1. Ominaisuuden haastavuuden kasvaessa tehtävän pistemäärä kasvaa fibonaccin lukusarjan mukaisesti ja saa esimerkiksi arvon 2, 3, 5, 8, 13 ja niin edelleen. Lukuarvot voivat myös olla – scrumtiimin niin halutessa – jollain aivan muulla tavalla määritelty. Tehtäväpisteillä ja tehdyllä työllä ei ole suoraa

yhteyttä tehtyjen työtuntien määrään, vaan tehtäväpisteiden tarkoitus on auttaa scrumtiimiä arvioimaan ominaisuuksien vaatimien ponnistusten määrää haasteiden voittamiseksi. (Srinivasan 2007a.) Tehtäväpisteet kuvaavat siis tehtävien vaativuutta ja työmäärää suhteessa toisiinsa. (Kniberg & Skarin 2010, 31.)

Kun scrumtiimeissä kunkin sprintin kohdalla lasketaan yhteen tehtyjen ominaisuuksien tehtäväpisteiden summa, saadaan vauhti (velocity). Vauhti kuvaa siis tehdyn työn määrä, eikä näin ollen välttämättä vastaa suunnitellun työn määrää. (Srinivasan 2007b.) Vauhti vakiintuu yleensä muutaman sprintin kuluessa, jonka jälkeen tiimi saavuttaa vauhdissa vakaan tilan. Vauhdin seuraamisen avulla saaduilla tilastotiedoilla voidaan arvioida karkealla tasolla, kuinka kauan jonkin isomman ominaisuuskokonaisuuden toteuttaminen kestäisi sen jälkeen, kun tehtävät on tunnistettu ja pisteytetty. (The Agile Management Company 2012.)

Kun tiimi on saavuttanut vakaan tilan, voi esimerkiksi tuotteen omistaja asettaa vauhtia vastaavan määrän tehtäviä jo etukäteen tulevaa sprinttiä varten, kun tiedossa on kuinka monen tehtäväpisteen edestä ominaisuuksia pystytään toteuttamaan. Vakaan tilan saavuttamisen jälkeen on myös periaatteessa mahdollista pitää yllä yhtä hyvää työtehoa kuukaudesta toiseen ilman vaaraa tiimin jäsenten loppuun palamisesta. (Milunsky 2009.) Tällaisessa lähestymistavassa on tosin se ongelma, että tehdyt arviot eivät välttämättä olekaan riittävän tarkkoja ja tehtävät on esimerkiksi arvioitu alakanttiin. Lisäksi laajuus voi muuttua tai eteen voi tulla ennalta arvaamattomia esteitä. (Kniberg 2007, 25.)

Yksi keino arvioida tulevan sprintin vauhtia on käyttää fokus-kerrointa. Kertoimessa on yksinkertaisesti kyse käytettävissä olevien henkilötyöpäivien kertomisesta fokus-kertoimella. Fokus-kerroin voidaan laskea esimerkiksi aikaisemman sprintin vauhdin mukaan jolloin saavutettu vauhti jaetaan käytettyjen henkilötyöpäivien lukumäärällä. (Kniberg 2007, 26-27.)

$$fokus - kerroin = \frac{tehokkuus}{käytettyjen\ henkilötyöpäivien\ lkm}$$

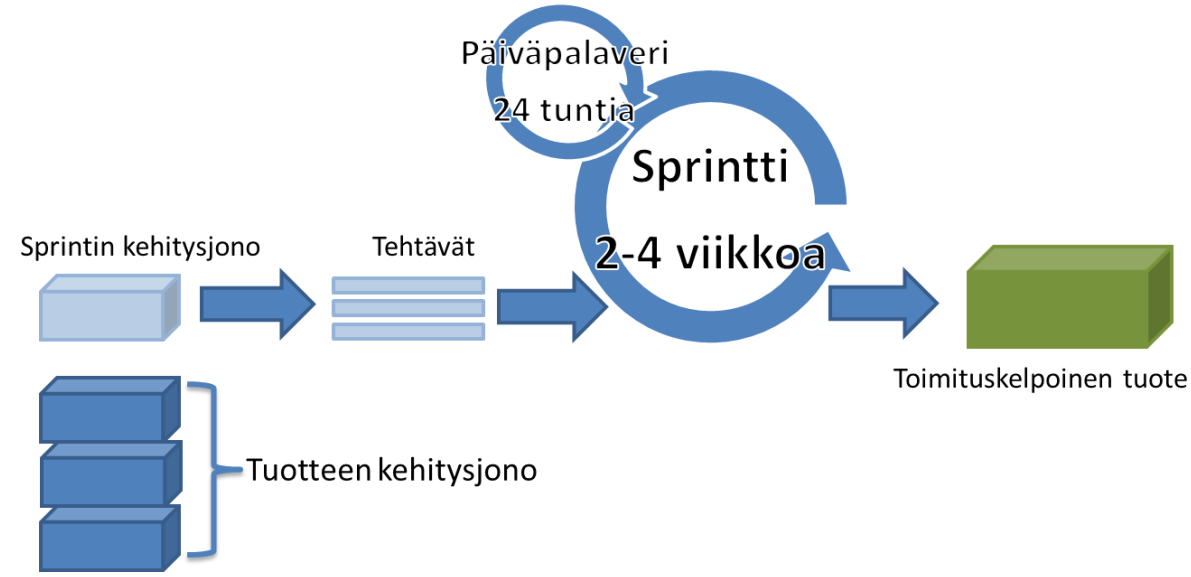
$$käytettävissä\ olevien\ henkilötyöpäivien\ lkm \times focus - kerroin = arvioitu\ vauhti$$

2.3.3 Scrum-prosessi

Lähtökohtana scrumissa on tuotteen omistajan luoma visio tuotteelle, eli tuotteen kehitysjono, joka sisältää tuotteeseen toivotut ominaisuudet. (Benefield ym. 2010, 8; Kniberg 2007, 9-13.) Scrum ei määrittele sitä, kuinka kehitysjonossa olevat ominaisuudet priorisoidaan tai millä menetelmällä työmääräarviot tehdään. Priorisointi on itsessään tuotteen omistajan tehtävä, jossa scrummaster tai scrumtiimi voi tarpeen mukaan avustaa. (Benefield ym. 2010, 8; Scrum Alliance 2011; Kniber 2007, 15.)

Tiimin päivittäisen toiminnan tehokkuudesta vastaa scrummaster, jonka johdolla käydään päivittäin päiväpalaveri (daily scrum), jossa käydään läpi päivän tilanne töiden osalta. Kunkin sprintin jälkeen tuotteen tulisi olla sellaisessa tilassa, että siitä voidaan luovuttaa asiakkaalle jokin versio (lopullinen, beta- tai jokin muu versio). Sprintin päätyttyä tehdään sprinttikatselmus (sprint review), eli katsaus sprintin aikana tehtyihin asioihin. Tätä sykliä toistetaan niin kauan kunnes tuotteen kehitysjono on käyty läpi tai taloudelliset resurssit loppuvat. (Scrum Alliance 2011.)

Scrumissa sprintin aikana tehtävät työt visualisoidaan taulun avulla, josta voidaan nähdä jäljellä olevat työt ja työn alla olevien tehtävien tilanne. Periaatteessa tavoitteena on saada taulu ”tyhjäksi”, eli tehtävät valmiiksi sprintin loppuun mennessä. Seuraavan sprintin alkaessa taulu tyhjennetään vanhoista tehtävistä ja tilalle laitetaan uudet tehtävät, siten kuten tuotteen omistaja on ne priorisoinut. (Kniberg & Skarin 2010, 25.)



Kuvio 5. Scrum-prosessi (Murphy 2004, 13)

Scumtiimeissä erimielisyyttä voi aiheuttaa määritelmä siitä, milloin tehtävä on valmis (definition of done). Määritelmä voi vaihdella sen mukaan, minkälaisesta tehtävästä on kyse. Selvästikin esimerkiksi dokumentointia koskevan tehtävän kriteerit valmistumiselle ovat erilaiset, kuin jollekin ohjelmiston ominaisuudelle. Tehtävän luonteesta riippumatta kaikilla tulisi kuitenkin olla yhtäläinen näkemys siitä, mitä tulee olla tehtynä, jotta tehtävän voidaan katsoa olevan valmis. Mikäli scrumtiimissä törmätään usein erimielisyyksiin valmistumisen määritelmän kanssa, voi olla tarkoituksen mukaista kirjata määritelmä jokaisen tehtävän yhteyteen. (Kniberg 2007, 32.) Hyödyllinen lisätieto tehtävien kuvaukseen voi olla myös se, kuinka kyseistä tehtävää voidaan esitellä (how to demo) sprintin lopussa esiteltävässä demossa. Kuvaus itsessään voi olla hyvinkin korkean tason yksinkertainen kuvaus mitä demossa tulee esitellä. (Kniberg 2007, 10.) Kentän hyödyllisyys ja tarkoituksenmukaisuus noussee esille silloin, mikäli ei ole etukäteen tiedossa kuka tulee demon esittämään sprintin päätteeksi, tai jos demon esittäjä vaihtelee.

Demojen pitämisestä sprintin päätteeksi voidaan nähdä olevan suuriakin hyötyjä. Luonnollisesti asianosaiset ja muut kyseisestä projektista kiinnostuneet näkevät missä vaiheessa projekti on menossa ja mitä on saatu aikaiseksi. Lisäksi tiimi voi saada palautetta tekemästään työstä. Demojen pitäminen myös kannustaa tiimiä saattamaan asioita

valmiiksi asti. Ilman demojen pitämistä voitaisiin päätyä tilanteeseen jossa asiat eivät välttämättä koskaan lopulta valmistuisi ja aikataulu venyisi. (Kniberg 2007, 65.)

2.4 Extreme Programming (XP)

Perinteisesti ohjelmistojen suunnittelussa on lähdetty siitä, että ohjelmistot muuttuvat ja suunnittelussa pyritään ennakoimaan tulevia muutoksia. Extreme Programming:ssa (XP) – joka on scrumin ohella yksi laajimmalle levinneistä ketterän kehityksen menetelmistä – kyseisen kaltainen ajattelumalli on hylätty kokonaan, koska useimmiten muutokset joita ohjelmistoihin myöhemmin vaaditaan, eivät ole aikaisempien ennakkoodotusten mukaisia. Näin ollen ennakkosuunnittelu olisi täysin hukkaan heitettyä työtä. (Sommerville 2006, 401.)

2.4.1 XP-kehitystyön prosessi

XP:ssä tuotteiden kehitys tapahtuu ketterien menetelmien periaatteiden mukaisesti iteratiivisesti pienissä osissa. Asiakkaiden osallistamista ja asiakasnäkökulman mukaan saamista kehitystyössä pidetään XP:ssä ensiarvoisen tärkeänä. XP:ssä lähdekoodia kehitetään myös vanhempien ominaisuuksien osalta refaktoroimalla koodikantaa jatkuvasti. Tämä tarkoittaa sitä, että jokaisen kehittäjän oletetaan kehittävän koodikantaa paremmaksi sekä laadun, että ylläpidettävyyden osalta aina kehitysmahdollisuuksia havaitessaan. (Sommerville 2006, 399-400.)

XP:ssä kehitettävät ominaisuudet esitetään erilaisina skenaarioina (user story). Jokainen skenaario pilkotaan vielä pienemmiksi tehtäviksi (task). Kehitystyö itsessään XP-menetelmää käyttäen tehdään pariohjelmoiden. (Sommerville 2006, 398)

Uusia koontiversioita (build) saatetaan tehdä XP:ssä useita päivässä, mutta varsinainen julkaisuversio luovutetaan asiakkaalle kahden viikon välein. Jokaisen version käännöksen yhteydessä suoritetaan versiolle kaikki vanhat automaattiset testit, sekä uutta ominaisuutta varten tehdyt testit. Vain versiot joilla kaikki automaattiset testit saadaan suoritetuksi onnistuneesti, voidaan hyväksyä julkaisuversioksi kelpaavaksi ohjelmaversioksi. (Sommerville 2006, 401.)

2.4.2 Pariohjelmointi ja protoilu

Pariohjelmointi on kehitystyömenetelmä jossa kaksi kehittäjää työskentelee samalla työasemalla samanaikaisesti. Toinen kehittäjä toimii vetäjänä (driver), joka tekee ohjelmointityötä. Toinen pareista toimii puolestaan tarkkailijana (observer), jonka tehtävänä on seurata vetäjän työtä ja huomauttaa mahdollisista ongelmakohdista joita huomaa esiintyvän. Rooleja vaihdetaan useasti työpäivän aikana, esimerkiksi puolen tunnin tai tunnin välein. (Jones 2010, 286; Kroll & Kruchten 2003, 356-357.) Pariohjelmointia voidaan soveltaa myös Test Driven Development (TDD)-ideologiaa noudattavissa tiimeissä siten, että toinen kehittäjistä kirjoittaa ensin yksikkötestit tulevalle toteutukselle, jonka jälkeen toinen kehittäjä kirjoittaa varsinaisen ohjelmakoodin, jonka toimivuus verifioidaan osin aikaisemmin tehdyillä yksikkötesteillä. (Jones 2010, 287.)

XP:ssä käytetään aktiivisesti pariohjelmointia kehitystyössä. Parit muodostetaan dynaamisesti, jotta kaikki kehittäjät tulevat työskentelemään keskenään ja toteuttamaan eri ominaisuuksia. XP:n yhtenä periaatteetta on se, että koodikanta on kollektiivisesti omistettu koko tiimin kesken, eikä mitään yksittäistä ominaisuutta ja koodiosaa ole omistettu tai osoitettu kenellekään tietylle kehittäjälle. Pariohjelmointi muistuttaa osittain koodikatselmointeja, joiden on myös todettu vähentävän virheitä ohjelmakoodissa. Vaikka pariohjelmointi ei välttämättä ole aivan yhtä tehokasta virheiden havaitsemiseen kuin koodikatselmointi, on se kuitenkin huomattavasti vähemmän aikaa vievä ja kuormittava prosessi. (Sommerville 2006, 404; Kniberg 2007, 81-85.)

Jonesin (2010, 287-289) mukaan pariohjelmoinnin hyötyjä pohdittaessa asiaa voidaan tarkastella sekä tuottavuuden, että laadun näkökulmasta. Kun pariohjelmointia verrataan yksin tapahtuvaan ohjelmointiin ja tuottavuutta verrataan toimintopisteillä (function points), on pariohjelmoinnin tuottavuus keskimäärin 30% pienempi. Toisaalta samanaikaisesti kehitysaika lyhenee pariohjelmoinnin avulla 10% - 30%. Laadullisesta näkökulmasta tarkasteltuna pariohjelmoinnilla lopputuotteessa olevien virheiden määrä on noin 15% pienempi, kuin jos kehitystyötä tekisivät yksittäiset kehittäjät toisistaan erillään.

Pariohjelmoinnin hyödyntämisessä on myös syytä miettiä asian sosiaalisia näkökulmia. Ohjelmistokehitystä tekevät henkilöt ovat usein sisäänpäin kääntyneitä, jonka vuoksi monet saattavat kokea pariohjelmoinnin epämukavaksi työtavaksi. Näin ollen pariohjelmointi ei välttämättä sovellu täysin varauksetta kaikkien kehitystiimien ja –organisaatioiden päivittäiseksi työtavaksi. (Jones 2010, 288; Kniberg 2007, 82.)

XP:n mukaisesti toimittaessa joissakin tilanteissa kehitystyötä ei voida tehdä normaalien käytäntöjen mukaisesti ja paras tapa esitellä uusia ominaisuuksia voi olla tehdä prototyyppi. Tällaisen protoilun ideana on tutkia toteutusmenetelmiä ominaisuuksille sekä pohtia niiden mahdollisuuksia ja liiketoimintahyötyjä. Toisaalta protoilun yhtenä tarkoituksena voi myös olla selvittää voidaanko haluttua ominaisuutta ylipäätään toteuttaa. Protoilulla voidaan lopulta vaikuttaa tuleviin järjestelmävaatimuksiin, kun tiedetään toteutuksen mahdollisuudet. Samalla protoilulla voi olla käyttöliittymäsuunnittelua ja tulevan toteutuksen käytettävyyttä ohjaava rooli. Protoilu on siis eräänlainen keino hallita ohjelmistokehitystyöhön liittyviä riskejä. Prototyypin evaluoinnin jälkeen varsinainen suunnittelu ja toteutus aloitetaan kuitenkin ”tyhjältä pöydältä”. Prototyypit eivät useinkaan ole laadukkaita verrattuna ohjelmistoihin jotka on alusta alkaen rakennettu suunnitelmallisesti ja laatua parantavia menetelmiä käyttäen. Sen vuoksi protoilun tuloksena syntynyt lähdekoodi hylätään. (Sommerville 2006, 409-410; Kruchten 2011, 185-188.)

2.5 Kanban

Kanbanissa on kyseessä myös ketterän ohjelmistokehityksen malli. Tietyllä tapaa Kanbania voidaan pitää scrumin laajennoksena. Scrumista poiketen Kanbanissa ei kuitenkaan ole varsinaista suunnittelupalaveria ennen sprintin aloittamista, vaan suunnittelua tehdään tarpeen mukaan, vaikka päivittäin. Lisäksi Kanbanissa käytetään apuna sarakkeisiin jaettua taulua, johon kirjataan lapuille kukin tehtävä. Kukin sarake taululla kuvastaa eri tilaa, jossa tehtävä voi olla. Kunkin tehtävän kulloinenkin tilanne määrittyy sen mukaan mikä on sen sijainti taululla. (Lekman 2009.) Kanbanissa taululle lisätään uusia tehtäviä aina tarpeen mukaan eikä näin ollen koskaan tyhjene samalla tavalla, kuin Scrumissa. Tietyllä tapaa Kanban-taulu muistuttaakin liukuhihnaa joka pysyy jatkuvassa liikkeessä, kun uusia tehtäviä ilmaantuu. (Lekman 2009; Kniberg & Skarin 2010, 24.)

Kanban-taulun sarakkeille ei ole olemassa mitään ohjeellista määrää, kuinka paljon niitä tulee olla. Prosessit ovat erilaisia eri organisaatioissa ja näin ollen myös prosessien vaiheiden lukumäärät vaihtelevat. Sarakkeita voidaan lisätä Kanban taululle tai niitä voidaan poistaa taululta aina tarvittaessa. Taulun suunnittelun ohjenuorana Kanbanissa tulisi kuitenkin pitää ajatusta ”vähemmän on enemmän”, eli lähteä liikkeelle niin vähäisestä määrästä eri sarakkeita, kuin mahdollista. (Kniberg & Skarin 2010, 47.)

Kanban-taululla visualisoidaan työn kulku ja kussakin vaiheessa olevien töiden määrä. Kussakin taulun sarakkeessa olevien tehtävien määrä tulee rajoittaa mahdollisimman hyvän läpimenoajan (lead time, cycle time) saavuttamiseksi. Prosessin kehittymisen myötä läpimenoaika pyritään saamaan pieneksi ja ennustettavaksi. (Kniberg & Skarin 2010, 3-5.) Läpimenoajan seurannan lisäksi Kanbanissa voidaan haluttaessa käyttää myös muita kaavioita tehdyn työn visualisoimiseksi. Esimerkiksi kumulatiivisen työn virtaa (cumulative flow) avulla voidaan visualisoida, kuinka hyvin työt etenevät ja mikä on kullakin hetkellä olevien työtehtävien määrän vaikutus läpimenoaikaan. (Kniberg & Skarin 2010, 39.)

Scrumissa työmäärä rajoitetaan aikataulun mukaan siten, että määritellään mitkä tehtävät on saatava valmiiksi iteraation aikana. Kanbanissa työmäärää rajoitetaan työtaulun vaiheiden mukaan. Rajoittamalla sallittujen tehtävien määrää eri työvaiheissa voidaan välttää tilannetta jossa liian monta tehtävää on samanaikaisesti työn alla ja kaikkien tehtävien valmistuminen viivästyy. Mitään tarkkaa lukumäärää on mahdotonta määritellä, kuinka monta tehtävää saa olla missäkin prosessin vaiheessa. Prosessit ja niiden vaiheet vaihtelevat tiimeittäin joten paras vaihtoehto kullekin tiimille löytyy vain kokeilemalla eri vaihtoehtoja. Lopulta yrityksen ja erehdyksen kautta lopulta voidaan nähdä mitkä ovat Kanbanissa prosessin ongelmakohdat, jos taululla näkyy usein samassa kohdassa tyhjiä taulukon soluja tai tehtävät kertyvät pitkäksi aikaa johonkin vaiheeseen. Mikäli tehtävien sallittu lukumäärä jossakin vaiheessa on liian suuri, voi se viivästyttää ongelmien ratkaisemista. Näin voi käydä, mikäli ongelman esiintyessä on mahdollista lukumäärän puitteissa ottaa työstettäväksi jokin uusi tehtävä, sen sijaan että lukumäärärajoitin pakottaisi ratkaisemaan esiin tulleen ongelman. (Kniberg & Skarin 2010, 16-21.)

Vaikka Kanbanin voi ajatella ohjelmistokehityksessä olevan scrumin laajennos, se ei välttämättä sisällä samoja elementtejä, kuin scrum. Kanbanissa ei esimerkiksi ole määriteltynä samanlaisia rooleja, kuin scrumissa. Samat roolit voidaan haluttaessa määritellä myös Kanbanissa, mutta pakollista se ei ole. Ylipäätään Kanbanissa voidaan prosessiin tarvittaessa lisätä uusia elementtejä tarvittaessa. (Kniberg & Skarin 2010, 11.)

2.6 Rational Unified Process (RUP)

Rational Unified Process (RUP) on Rational Softwaren kehittämä prosessimalli ohjelmistokehitysorganisaatioiden tarpeisiin. Prosessin lähtökohtana on tehtävien ja vastuiden selkeä osoittaminen kehitystiimissä oleville tiimin jäsenille. Prosessin tavoitteena on tuottaa laadukkaita ohjelmistoja jotka sekä pysyvät budjetissa, että myös vastaavat asiakkaiden tarpeita. (Kruchten 2001, xiii.)

RUP:ssa tunnistetaan 6 parasta käytäntöä (best practise) joilla pyritään ohjaamaan kehitystyötä oikeaan suuntaan. RUP:in parhaita käytäntöjä noudattamalla pyritään eliminoimaan ohjelmistoprojektien perinteisesti kohtaamat ongelmat. RUP:n kehitystyötä ohjaavat periaatteet (Kruchten 2001, 5-6.) ovat:

1. Kehitä iteratiivisesti (Develop software iteratively).
2. Hallitse vaatimuksia (Manage requirements).
3. Käytä komponenttiarkkitehtuuria (Use component-based architectures).
4. Mallinna ohjelmisto visuaalisesti (Visually model software).
5. Varmista ohjelmiston laatu jatkuvasti (Continuously verify software quality).
6. Hallitse muutoksia (Control changes to software).

Muutosten hallinnan kannalta tärkeimpänä seikkana RUP:ssa nähdään olevan muutosten dokumentoinnin. Kehitystyön kannalta on kehittäjien tärkeää oivaltaa ja sisäistää se tosiasia, että kehitettäessä ohjelmistoja iteratiivisesti, määritykset tulevat muuttumaan. Senkin vuoksi muutokset tulee dokumentoida. Parhaimmillaan ja sopivien työkalujen avustuksella muutoshistoria saadaan dokumentoitua kaikkien ajan myötä muuttuneiden määritysten osalta. (Kruchten 2001, 8-9.)

Komponentteihin perustuvalla arkkitehtuurilla pyritään jakamaan ohjelmisto selkeiksi kokonaisuuksiksi ja tekemään osista uudelleen käytettäviä myös muiden ohjelmistojen tarpeisiin. Samoin komponenttiarkkitehtuurilla voidaan myös hyödyntää kolmansien osapuolten tekemiä komponentteja joiden avulla omaa työtaakkaa voidaan pienentää. (Kruchten 2001, 9-10.)

Ohjelmiston mallinnuksen pyrkimyksenä on kuvata ohjelmisto eri näkökulmista eri käyttötarpeita silmällä pitäen. Mallinnuksen avulla voidaan huomata mahdollisia ongelmia suunnittelussa, mutta samalla saadaan myös kuva ohjelmiston toiminnassa esimerkiksi tiettyjen käyttötilanteiden yhteydessä. (Kruchten 2001, 11-12.)

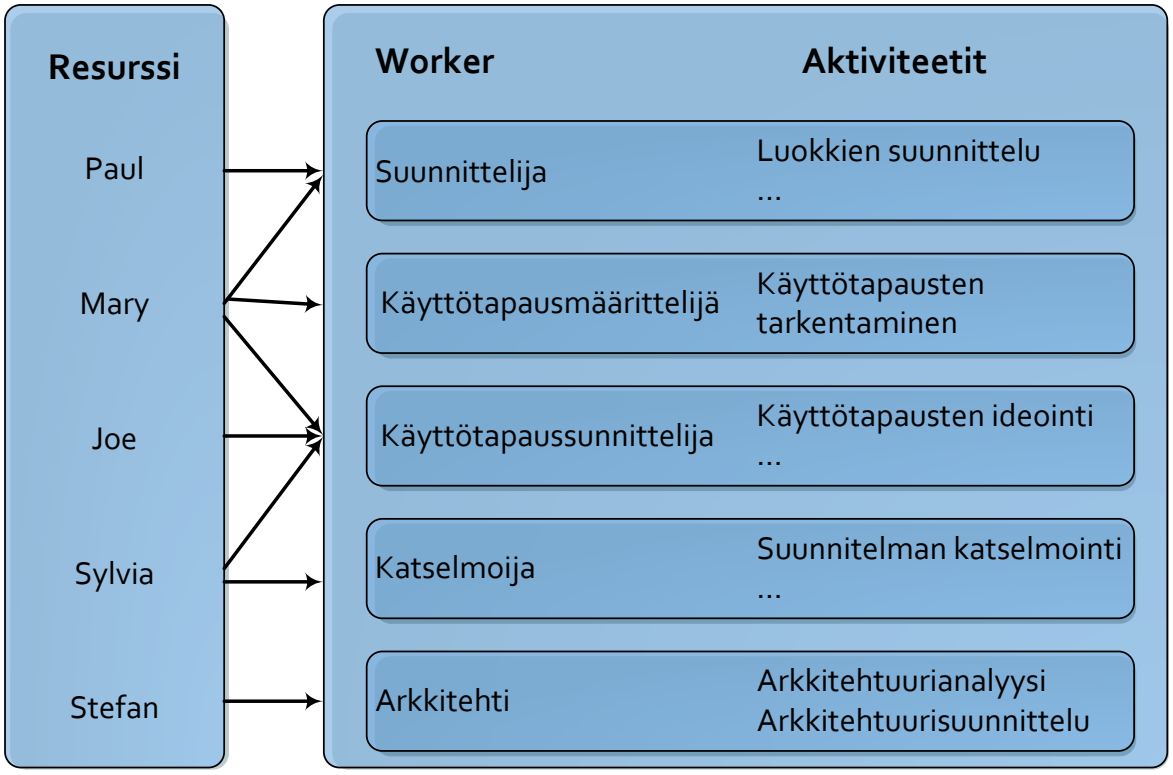
Jatkuvalla laadunvarmistuksella pyritään välttämään tilanteita joissa virheitä pääsisi tuotantoon asiakasympäristöissä. Iteratiivisen kehityksen ja laadunvarmistuksen yhdistämisen hyötynä on se, että kunkin iteraation kohdalla voidaan esimerkiksi testaus kohdistaa iteraation alkuvaiheessa määriteltyihin kriittisiin toimintoihin. (Kruchten 2001, 13.)

Koordinoimalla usein toistuvia tehtäviä ja muutoksia ohjelmistoprojekteissa, vältytään prosessin muuttumiselta kaoottiseksi ja sekavaksi. Koordinoinnilla voidaan samalla myös resursoida kehittäjiä paremmin kykyjä vastaaviin tehtäviin. Kaikkiaan jatkuvalla muutosten seuraamisella voidaan huomata ongelmia ja reagoida niihin nopeasti. Samalla saadaan myös kattava historia muutoksista riittävien työkalujen avulla. (Kruchten 2001, 14.)

Laadun osalta RUP erittelee tuotteen ja prosessin laadun. Tuotteen laatu sisältää kaikkien toimitettavaan ohjelmistoon liittyvien osien laadun, kuten esimerkiksi arkkitehtuuri. Prosessin laadun osalta seurataan, millä tasolla prosessi saatiin läpi vietyä tiettyjen prosessin seurannan kriteeristöjen mukaisesti. (Kruchten 2001, 29.)

Käytännön ohjelmistokehitystyössä RUP:ssa hyödynnetään ketteriä menetelmiä, mutta roolit (roolit tunnettiin RUP:ssa aikaisemmin nimellä: Worker) ovat määrittelyiltään väljempiä kuin esimerkiksi Scrumissa. Käytännössä yhdeksi rooliksi voi olla määriteltynä esimerkiksi suunnittelija tai arkkitehti. Rooli on puolestaan määritelty yksi tai useampia aktiviteettejä joiden suorittaminen kuuluu roolin vastuulle. Aktiviteetit (Activity)

kuvaavat toimintaa jonka tuloksia hyödynnetään kehitysprosessissa. Aktiviteetit puolestaan jakautuvat vielä useammaksi vaiheeksi (step), jotka on suoritettava aktiviteettia toteutettaessa. Roolin määrittely ei vielä yksilöi ketään yksittäistä henkilöä siten, että se kertoisi kuka aktiviteettejä käytännössä suorittaisi. RUP:in ideana kuitenkin on, että roolit myös liitetään organisaation resursseihin. RUP-prosessin yhteydessä resurssit ovat yksittäisiä työntekijöitä, henkilöitä. Yksi henkilö voidaan liittää yhteen tai useampaan rooliin jonka aktiviteetteja hän käytännössä suorittaa. Tämä mahdollistaa myös sen, että vaikka prosessissa tietyt aktiviteetit toistuisivat moneen kertaan, eivät kaikki aktiviteettien yksittäiset suoritukset lopulta välttämättä ole saman henkilön tekemiä, mikäli useampi henkilö on assosioitu samaan workeriin. (Kruchten 2001, 36-37; Kroll & Kruchten 2003, 13-15.)



Kuvio 6. RUP:n ihmiset ja workerit. (Kruchten 2003, 38)

Käytännön tasolla RUP:ia voidaan hyödyntää useammalla eri tavalla. RUP:in menetelmistöä voidaan noudattaa hyvinkin tarkasti ja muodollisesti. Joissakin organisaatioissa ei välttämättä ole halua tai tarvetta niin järjestelmälliseen mallin seuraamiseen ja RUP:ia voidaan pitää niissä tapauksissa vain suuntaviivoja antavana mallina. RUP:ia voidaan myös hyödyntää vain soveltuvin osin, ilman koko RUP:in menetelmistön hyödyntämis-

tä. Vaikka joihinkin organisaatioihin RUP voi hyvin soveltua sellaisenaan käytettäväksi, niin organisaatioissa joissa on jo kehittyneempi ohjelmistojen kehitysprosessi, tulee RUP:in soveltamisessa olla hyvin kriittinen ja muokata sitä tarpeellisin osin organisaation omia tarpeita vastaavaksi. (Kruchten 2001, 22, 31.)

2.7 Lean

Lean-ajattelun taustat johtavat Toyota tehtaille jossa Taiichi Ohno ja Shigeo Shingo kehittivät menetelmän tuotantoketjujen ja tuotantolaitosten hallintaan. Keskeistä Leanissä on, että Leanissä luotetaan jokaiseen yksittäiseen työntekijään merkittävänä osana tuotantoketjua. pyrkimyksenä on saada tuotantoprosessin kiertonopeus niin nopeaksi kuin mahdollista. Ketjun kiertonopeutta pyritään optimoimaan tekemällä kokonaisuuksista mahdollisimman pieniä (small batch) ja tehdä ne silloin kun niitä oikeasti tarvitaan (just-in-time). (Ries 2011, 18.) Menetelmää tunnetaan myös nimellä Toyota Production System (TPS) (Ries 2011, 86).

Pienet kokonaisuudet (pienet erät) olivat oikeastaan pienen japanilaisen autotehtaan keino taistella suurten yritysten massatuotantokoneistoja vastaan. Koska Toyotalla ei ollut mahdollisuutta kilpailla suurilla koneistoilla, jotka tuottaisivat suuria määriä erikoisosia, he päätyivät tekemään koneita jotka tuottivat monia erilaisia yleisempään käyttöön kelpaavia osia, mutta pienempiä määriä. Tämä vaati että koneistojen konfigurointia voitiin muuttaa hyvin nopeasti tarpeen mukaan. Optimoimalla vaihtoajan mahdollisimman pieneksi, Toyota pystyi valmistamaan osia juuri silloin kun niitä oikeasti tarvittiin, eikä ollut tarvetta pitää isoa varastoa osista ihan vain varmuuden vuoksi. (Ries 2011, 186.)

Leanissä asiakkaan ja asiakkaan tarpeiden ymmärtäminen on ensiarvoisen tärkeää. Leanin mukaisesti asiakkaan kanssa tulee olla niin läheisessä kontaktissa, kuin mahdollista ja välikäsiä tulee välttää, jotta väärinkäsityksiltä vältytään. Lean kannustaakin kehityksestä vastuussa olevia henkilöitä menemään itse asiakkaan luokse seuraamaan miten asiakkaat toimivat. Asiakkaan tarpeita ja toiveita ymmärretään parhaiten siinä ympäristössä jossa asiakkaat toimivat. (Ries 2011, 86-88.) Asiakkaisiin ja asiakastarpeiden tyydyttämiseksi vaaditaan koko organisaation johdolta selkeää ”asiakas ensin”-asennoitumista.

TPS:ssä on huomioitava se, että asiakas ei ole vain se, joka on tuotteen tai palvelun loppukäyttäjä. Asiakkaat voivat olla myös organisaation sisäisiä, eli asiakas voi olla toinen prosessi joka saa syötteeksi toisen prosessin tuotoksen. Tämän vuoksi pelkästään loppuasiakkaisiin keskittyminen on vain osa kokonaisuutta TPS:ssä. Asiakkaaseen keskittymisen lisäksi tulee huomiota kiinnittää prosessien välisiin rajapintoihin ja eliminoimaan kitka niiden välissä mahdollisimman sujuvan toimintoketjun saavuttamiseksi. (Poppendieck 2010b, 162.)

Lyhyesti sanottuna Leanin tavoitteena on tuottaa nopeasti niitä asioita, jotka tuottavat asiakkaalle arvoa. Nopeuteen pyritään lyhentämällä kaikkien prosessien kiertonopeutta laadun kuitenkaan siitä kärsimättä. Arvo määritellään Leanissa sellaisiksi asioiksi, joista asiakas on valmis maksamaan. Sen lisäksi, että päämääränä on tuottaa arvoa, on yhtälailla tärkeää pyrkiä eliminoimaan niin paljon jätettä (waste), kuin mahdollista. Jätettä on kaikki se mistä asiakas ei ole valmis maksamaan. Sen lisäksi jätettä ovat myös monet muut asiat kuten viiveet, ylituotanto sekä useat muut asiat. (Larman & Vodde 2009, 9-20.)

Työskenneltäessä vaiheittain etenevässä prosessissa, työsuoritteiden määrän joka siirtyy aikaisemmasta vaiheesta seuraavaan, kertoo eräkoko (batch size). Lean ajattelun tavoitteena on pitää eräkoko mahdollisimman pienenä, jolloin prosessissa seuraavaan vaiheeseen siirtyvien suoritteiden määrä olisi lopulta aina vain yksi. Käsittelemällä pientä määrää suoritteita yhdellä kertaa, pyritään optimoimaan koko prosessin suorituskykyä ja sitä myötä prosessin läpimenoaikaa. Yksittäiseen vaiheeseen kuluva aika ei ole kokonaisuuden kannalta merkittävä. Samalla myös pyritään mahdollisimman aikaisessa vaiheessa saamaan palaute prosessin aikaisemmalle vaiheelle ongelmien esiintyessä. Parhaassa tapauksessa ongelma- tai virhetilanteessa pilalle menee vain yksi suorite, kun taas eräkoon ollessa suuri voi pilalle lopulta mennä satoja tai jopa tuhansia suoritteita. Mikäli eräkoko on suuri, on siitä myös seurauksena se, että prosessin seuraavat vaiheet joutuvat aina odottamaan kunnes edellinen vaihe on käsitellyt koko erän, ennen kuin ne voivat alkaa tekemään omaa suoritettaan. (Ries 2011, 184-185.)

2.7.1 Lean ohjelmistokehityksessä

Scrum ja Kanban ovat luonteeltaan hyvin Lean-ajatteluun sopivia. Esimerkiksi prosessien jatkuva kehittäminen kuuluu molempiin kehitysmenetelmiin. Myös ketteriin menetelmiin kuuluvat kehitystiimin itseohjautuvuus aikataulutuksen ja priorisoinnin suhteen soveltuu hyvin Leanin ideologiaan. Lisäksi sekä Leanissa, että ketterissä menetelmissä enemmänkin odotetaan muutoksia tapahtuvaksi sen sijaan, että pyrittäisiin tekemään hyvin tarkat määritykset etukäteen. (Kniberg & Skarin 2010, 35.) Näiden ideologisten yhteneväisyyksien vuoksi Lean-ajattelu sopii hyödynnettäväksi myös ohjelmistokehitystyöhön yhdessä ketterien kehitystyön menetelmien kanssa.

2.7.2 Pienet ja suuret erät ohjelmistokehityksessä

Ohjelmistokehityksen yhteydessä Leanin terminologiaa ja menetelmistöä voidaan suhtetuttaa esimerkiksi seuraavalla tavalla: Toteutettaessa ohjelmistoja perinteisen vesiputousmallin mukaisesti, tehdään julkaisuja useimmiten tietyin väliajoin esimerkiksi kerran tai kaksi kertaa vuodessa. Tällöin on kyseessä selkeästi suuri erä (large batch), koska kaikki muutokset ohjelmistoon julkaistaan yhdellä kertaa. (Ries 2011, 245; Poppendieck 2010b, 114.) Vastaavasti ketterät menetelmät ovat enemmän Leanin periaatteiden mukaisia pienten erien suhteen. Pienet erät tulevat esiin jo ketterien menetelmien luonteessa itsessään, kun tavoitteena on saada uusi julkaisu tehtyä esimerkiksi kahden viikon välein. Nopea julkaisusykli mahdollistaa nopeat suunnan muutokset ja nopeat reagoinnit vaatimusten ynnä muiden tuotteisiin liittyvien asioiden muutoksiin. Ohjelmistokehityksessä pieni eräkoko voi olla esimerkiksi yhden tai kahden henkilötyöpäivän kokoinen työ. (Ries 2011, 133.) Ohjelmistokehitystyössä pieniä eriä voidaan hyödyntää myös esimerkiksi suorittamalla usein toistuvasti automaattisia testejä. Jokainen testi itsessään on kohtalaisen pieni ja yksinkertainen, mutta suorittamalla kyseisiä testejä useita kertoja päivän aikana saadaan samalla ratkaistua oikeastaan kaksi asiaa. Ensiksi kyseessä toteutetaan Leanin ideologiaa useiden toistuvien pienten erien muodossa. Toiseksi kyseisellä tavalla saadaan osa tuotteen laadusta rakennettua kiinteästi osaksi tuotetta suorittamalla laadunvarmistuksen kannalta olennaisia testaustoimenpiteitä toistuvasti. (Larman & Vodde 2009, 22.)

Ketterät menetelmät eivät itsessään estä jätteen syntymistä ohjelmistokehityksessä, vaikka niiden soveltuminen Lean-ideologiaan voisi antaa niin ymmärtää. Kaikista hyödyistään huolimatta myös ketterillä menetelmillä voidaan saada tuotetuksi täysin tarpeettomia ohjelmistoja joita kukaan ei tarvitse tai kukaan ei halua käyttää. Tämä on vaarana varsinkin, mikäli unohdetaan Leanin yksi tärkeimmistä periaatteista eli asiakkaan ja asiakastarpeiden ymmärtäminen. (Ries 2011, 46-47.)

2.7.3 Jäte ohjelmistokehityksessä

Ohjelmistokehityksen yhteydessä jäte on luonnollisesti hieman erilaista verrattuna teollisuuden prosesseihin. Ohjelmistokehityksessä harvoin on esimerkiksi liiallisia varaosavarastoja, jotka olisivat jätettä. Jätettä kuitenkin esiintyy, mutta eri muodossa. jäte ja sen esiintyminen voi johtua heikoista käytännöistä tai heikosta johtamisesta. Heikot käytännöt voivat luonnollisesti olla myös seurausta heikosta johtamisesta.

Ketterillä menetelmillä voidaan pyrkiä eliminoimaan jätettä. Jätteen eliminoimiseksi tärkeintä on tietää mistä jätettä voi etsiä. Ohjelmistotuotannon ollessa kyseessä, voivat jotkut esittää niinkin radikaaleja ajatuksia, kuin että jätettä on kaikki muu paitsi analysointi ja ohjelmoiminen. TPS-menetelmän ideoijat Taiichi Ohno ja Shigeo Shingo listasivat seitsemän erilaista mahdollisuutta jätteeseen teollisuuden prosesseissa. Ohjelmistoteollisuudessa nuo vastaavat jätetyypit voidaan esittää esimerkiksi taulukon 1 mukaisella tavalla. (Poppendieck 2010a, 4.)

Taulukko 1. Teollisuuden ja ohjelmistotuotannon jätetyypit (Poppendieck 2010, 4)

Teollisuus	**Ohjelmistotuotanto**
Varastot (In-process Inventory)	Osittainen työ (Partially Done Work)
Ylituotanto (Over-Production)	Ylimääräiset ominaisuudet (Extra Features)
Yliprosessointi (Extra Processing)	Ylimääräiset prosessit (Extra Processes)
Kuljetukset (Transportation)	Tehtävän vaihto (Task Switching)
Liike (Motion)	Liike (Motion)
Odotusaika (Waiting)	Viiveet (Delays)
Virheelliset tuotteet (Defects)	Virheet (Defects)

Osittain tehdyn työn ongelma piilee siinä, että sillä on vaara tulla ajan myötä tarpeettomaksi ja hyödyttömäksi. Niin kauan kuin työ on keskeneräinen, ei ole varmuutta siitä toimiiko lopputulos kuten pitäisi. Sen lisäksi ettei lopputuloksesta ole toimintavarmuutta, keskeneräinen työ myös sitoo toteutusvaiheessa resursseja ja sitä kautta myös pääomia. Minimoimalla keskeneräisen työn määrä organisaatiossa, voidaan hallita kehitystyöhön liittyviä riskejä ja samalla voidaan vähentää jätettä. (Poppendieck 2010a, 5.)

Ohjelmistojen kehitystyössä voi piillä houkutus lisätä tuotteeseen uusia ominaisuuksia varmuuden vuoksi tai sen vuoksi, että niitä arvellaan tarvittavan. Harmittomalta ja hyödylliseltä vaikuttavassa ajattelutavassa piilee kuitenkin siinäkin riskejä. Jokainen ominaisuus on toteutuksen lisäksi testattava, verifioitava ja hallinnoitava muutosten varalta koko tuotteen elinkaaren ajan. Sen lisäksi ominaisuuksissa voi piillä mahdollisuuksia ohjelman virheelliseen toimintaan, sekä ne voivat myös lisätä tuotteen monimutkaisuutta. Kaikesta tästä aiheutuu tuotteelle kustannuksia koko elinkaaren ajalle. Sen vuoksi kehitystyössä tulisi pyrkiä välttämään sellaisten ominaisuuksien toteuttamista, joille ei ole kysyntää ja tarvetta juuri toteutushetkellä. (Poppendieck 2010a, 6.)

Ylimääräiset prosessit esiintyvät ohjelmistokehityksessä lähinnä erilaisten paperitöiden muodossa. Mikäli kehitystyön yhteydessä herää kysymys jonkun paperityön tarpeellisuudesta, todennäköisesti silloin on syytä miettiä voitaisiinko jostakin turhasta ja hyödyttömästä paperityöstä luopua. Paperitöitä on toki monenlaisia ohjelmistokehitykseenkin liittyen, kuten käyttötapauskaavioita tai koulutusmateriaaleja. Tällöin merkityksellistä on kuitenkin se, tuottaako paperityö oikeasti arvoa asiakkaalle vai aiheuttavatko paperityöt viivästyksiä. Paperitöissä ongelmana voi olla myös osittaisen työn ongelma, eli ajan myötä erilaiset dokumentit voivat jäädä hyödyttömiksi. Sen vuoksi erilaiset dokumentoinnit, kuten ominaisuuksien tarkemman tason määrittelyt, tulisi tehdä vasta siinä vaiheessa kun niitä oikeasti tarvitaan. (Poppendieck 2010a, 5-6.)

Ihmisten työskennellessä useissa projekteissa samanaikaisesti törmätään tehtävän vaihdon ongelmaan. Samanaikaisesti työskentely useammassa projektissa tarkoittaa useimmiten myös useampia keskeytyksiä sekä tehtävien vaihtoja projektien välillä. Jokaiseen tehtävän vaihtoon kuluu aikaa toiseen tehtävään orientoitumisen muodossa. Jokainen

tehtävän vaihto myös vääjäämättä aiheuttaa viivästyksiä jokaiseen projektiin jossa kyseinen henkilö työskentelee samanaikaisesti. (Poppendieck 2010a, 6.)

Liike tarkoittaa ohjelmistokehityksen osalta useita asioita. Liike on esimerkiksi työntekijän fyysistä liikkumista huoneesta toiseen organisaation tiloissa. Tämän vuoksi kehitystyötä tekevä tiimi tulisi sijoittaa samaan tilaan, jotta ylimääräiseltä liikkumiselta vältyttäisiin ja ratkaisut esiin nouseviin ongelmiin voitaisiin ratkaista tiimin henkilöiden kesken samassa tilassa. Henkilöiden lisäksi myös asiat voivat liikkua. Esimerkiksi erilaiset määrittelydokumentit voivat siirtyä esimerkiksi suunnittelijoilta kehittäjille. Tämä voi johtaa tilanteeseen jossa esiin nousseeseen ongelmaan voidaan joutua etsimään vastausta usean välikäden kautta. Kehitystyö on keskittymistä vaativaa työtä ja liike voi omalta osaltaan aiheuttaa keskittymisen herpaantumisen ja sitä kautta vähentää työtehoa. Liike tulisi näin ollen pyrkiä minimoimaan ja asianosaiset henkilöt sijoittamaan samoihin tiloihin mahdollisuuksien mukaan. (Poppendieck 2010a, 7-8.)

Viiveitä esiintyy kehitystyötä tekevissä organisaatioissa usein. Projektin alku saattaa viivästyä, resurssien saaminen projektiin voi viivästyä, katselmoinnit voivat viivästyä jne. Viiveiden voi käsittää olevan luonnollinen osa kehitystyötä. Asiaa voidaan kuitenkin tarkastella myös asiakasnäkökulmasta. Tällöin voidaan havaita, että viiveet yleensä johtavat siihen, että viiveet tosiasiassa aiheuttavat viivästyksen asiakkaan saamalle arvontuotolle. Asiakkaan arvontuoton viivästyttäminen on Leanin periaatteiden vastaista ja näin ollen viiveet tulisi pitää minimissään. (Poppendieck 2010a, 7.)

Virheiden osalta jäte muodostuu siitä, kuinka kauan kestää että virheet havaitaan. Nopeasti havaittuja virheitä ei voida pitää jätteenä. Sen sijaan virhe jonka havaitaan pitkän ajan kuluttua, on jätteen osalta paljon suurempi. Jäte on suurempi sen vuoksi että mitä myöhemmin virhe havaitaan, sitä myöhemmin myös havaitaan sen vaikutuksen kaikkeen muuhun toiminnallisuuteen. Virheiden havaitsemisen kannalta onkin tärkeää että laadunvarmistus on hyvin hoidossa, jotta virheet havaitaan mahdollisimman pian. (Poppendieck 2010a, 8.)

Jäte on monitahoisempi asia, kuin mitä taulukko 1 antaa aluksi ymmärtää. jätettä voidaan lopulta löytää hyvinkin monista paikoista ja moniin eri tilanteisiin liittyen ohjel-

mistokehitystyössä. Erilaiset virhe- ja selvitystilanteet voivat ylimääräistä ja odottamatonta työtä ja ovat siten epätoivottuja tapahtumia kehitystyötä tekevissä organisaatioissa.

Virhepyynnöissä (failure demand) on käytännössä kyse erilaisista ongelmatilanteista ja niiden ratkaisutavoista. Virhepyynnöille ominaista on se, että se vaatii käytännössä organisaation resursseja ongelmatilanteen ratkaisemiseksi, jotka johtuvat organisaation itse itselleen aiheuttamista ongelmista. Esimerkiksi liki kaikki tukipyynnöt ovat virhepyyntöjä, jotka johtuvat esimerkiksi siitä etteivät asiakkaat osaa käyttää tuotteita, koska käyttö on hankalaa tai dokumentaatio puutteellista. Samoin muutospyynnöt voivat olla virhepyynnöiksi laskettavia mikäli osoittautuu, ettei vaatimuksia ole ymmärretty oikein. Tekniseltä näkökannalta katsottuna asia on hieman monimutkaisempi. Mikäli itse toteutustyössä on päästetty tuotantoon asti heikkolaatuista ohjelmistokoodia, joka sisältää esimerkiksi saman ohjelmakoodin monistamista, on sekavasti kirjoitettu tai sisältää muita huonoja ohjelmistokehityksen käytäntöjä, voi seurauksena olla virhepyyntöjä. Kaikki huonot käytännöt toteutuksessa lopulta kasvattavat teknistä velkaa (ohjelmistokoodin tiedetään olevan heikkolaatuista ja se vaatii uudelleen kirjoittamista) joka ajan myötä vaikeuttaa muutospyyntöihin vastaamista. Kaikki asiat jotka vaikeuttavat muutosten tekemistä ohjelmakoodiin ovat laskettavissa virhepyynnöiksi. (Poppendieck 2010b, 10.)

Useimmiten kun organisaatiossa aletaan etsiä mahdollisuuksia virhepyyntöjen esiintymisille, niitä löydetään yleensä paljon. Koska virhepyynnöt ovat usein suoraa seurausta siitä kuinka asioita organisaatiossa tehdään, on virhepyyntöjen kartoitus myös mahdollisuus huomattavaan prosessien parannukseen organisaatiossa. Lopulta kyseisten aktiviteettien tavoitteena on löytää ja eliminoida niin paljon virhepyyntöjä pois kuin mahdollista. Joidenkin arvioiden mukaan virhepyyntöjen määrä vaihtelee alkuvaiheessa 30%:n ja 70%:n välillä. Eli käytännössä kehitystyöstä suurin osa onkin itse asiassa kaikkea muuta kuin mitä sen pitäisi olla. Eliminoimalla virhepyynnöt voidaan siis saada kehitystyön kapasiteettia kasvatettua huomattavasti. (Poppendieck 2010b, 11.)

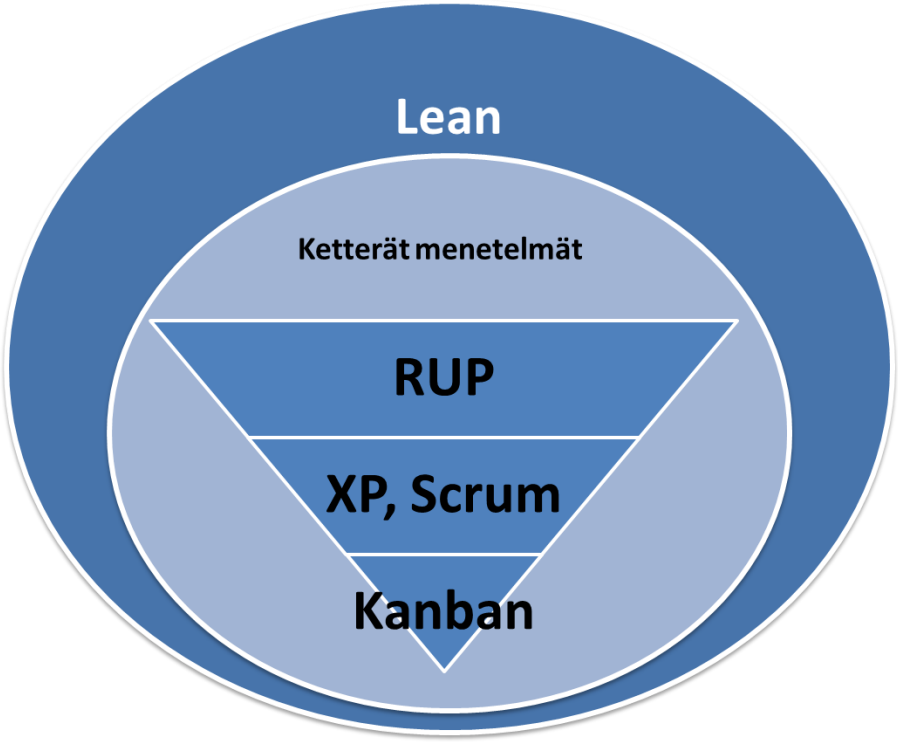
Jätettä voi myös esiintyä toimintatavoista johtuen. Joissakin organisaatioissa ei esimerkiksi välttämättä haluta että kehittäjät ovat suorassa asiakaskontaktissa, sillä sen kuvitel-

laan heikentävän kehitystyön tehoa ja tuottavuutta. Asiakaspalaute välittyy sen sijaan kehittäjille välikäsien, kuten asiakastukitiimin kautta. Välikäsien määrän kasvaessa prosessin alusta loppuun kulkevan kokonaisuuden keskellä, voi välikäsien määrä aiheuttaa lopulta enemmän viivästyksiä kuin hyötyjä. (Poppendieck 2010b, 25.)

2.8 Yhteenveto ketteristä menetelmistä

Ketterät menetelmät ovat ohjelmistokehityksen prosessityökaluja ja niihin tulisi suhtautua sen mukaisesti. Tärkeää on tietää mitä työkalua käyttää missäkin tilanteessa. Työkalujen valintaan vaikuttaa luonnollisesti se, millaisia ehtoja ja vaatimuksia työkalu itsessään asettaa. Työkalu voi olla sen vuoksi luonteeltaan hyvin määräävä sen puolesta mitä ja minkälaisia asioita tulee olla määriteltynä, jotta menetelmää voidaan hyödyntää tehokkaimmin siinä mielessä, kuin sitä on tarkoitettu hyödynnettävän. Toisessa ääripäässä ovat sitten sopeutuvat työkalut, jotka jättävät paljon valinnan varaa sen suhteen mitä halutaan ja voidaan tehdä. Aikaisemmin esitetyistä menetelmistä toisessa ääripäässä sen suhteen kuinka määräävä työkalu on RUP, jossa on yli 30 erilaista roolia, 20 aktiviteettia ja 70 artefaktia. Määrä on hyvin suuri hallittavaksi ja voi lopulta osoittautua hyvin kankeaksi varsinkin pienemmissä projekteissa. Esimerkiksi kaikille RUP:in määrittämille rooleille ei yksinkertaisesti vain ole tarvetta läheskään kaikissa projekteissa. Scrumissa määriteltyjen prosessiin kuuluvien asioiden määrä on huomattavasti pienempi. Roolien, artefaktien ja pakollisten toimintatapojen yhteenlaskettu summa scrumissa on 12. XP:ssä vastaava luku on 13 ja Kanbanissa vain 3. (Kniberg & Skarin 2010, 7-9.)

Kuviossa 7 on esitetty ketterien menetelmien hierarkinen suhde sen mukaan kuinka kuvaava se on. Kuvaavuus tarkoittaa sitä kuinka paljon erilaisia määrityksiä ja vaatimuksia se asettaa. Kuviossa Lean ympäröi koko ketterien menetelmien hierarkian, joka on tavallaan koko toimintaa ohjaava ajatusmalli.



Kuvio 7. Ketterien menetelmien hierarkinen kuva

RUP:issa ja esimerkiksi scrumissa on sellainen perustavaa laatua oleva ero, että RUP:issa prosessista poistetaan ne elementit joita ei tarvita. Elementtejä on niin paljon, että niin oletetaan tehtävän. Scrumissa sen sijaan elementtejä on vähän ja niitä lisätään sitä mukaan, kun jotain huomataan puuttuvan. (Kniberg & Skarin 2010, 10.)

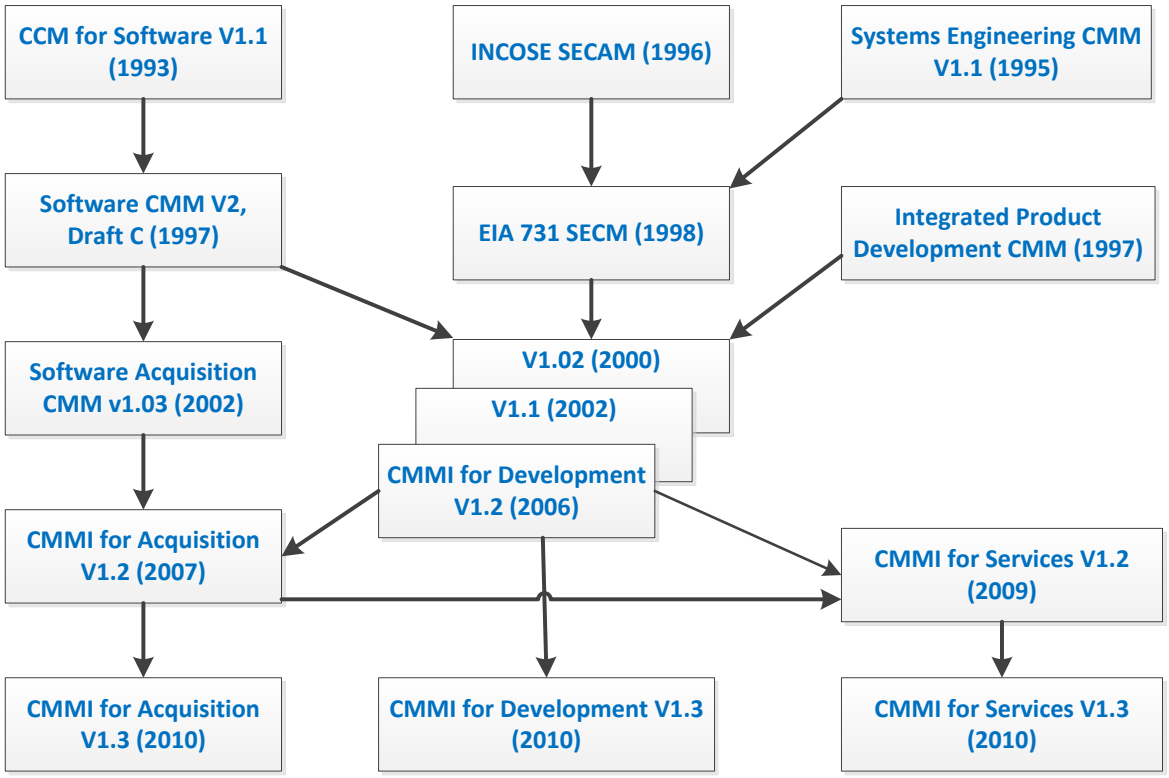
Scrumin ja XP:n erona voidaan nähdä olevan se, että scrum keskittyy enemmän hallinnointiin ja organisaatiotasoisiin käytäntöihin, kun taas XP keskittyy enemmän ohjelmoinnin käytännön toteutukseen ja siihen liittyviin asioihin. Tämän vuoksi scrum ja XP sopivat varsin hyvin yhteen, kun huomaa niiden toimivan eri tasolla tai koskevan eri aihealueita. (Kniberg 2007, 81.)

Olipa lähtökohtana mikä menetelmä vain, paras lopputulos saavutettaneen kuitenkin useamman menetelmän hyvien puolien yhdistelemisellä. (Kniberg & Skarin 2010, 10). (Shelton 2008). Mikäli kuitenkin poistetaan jotain hyvin keskeisiä elementtejä, ei menetelmää enää tulisi kutsua menetelmän alkuperäisellä nimellä. Jos esimerkiksi Kanbanista poistettaisiin hyvin keskeisessä asemassa oleva Kanban-taulu, ei menetelmää voisi enää perustellusti kutsua Kanbaniksi. (Kniberg & Skarin 2010, 10.)

3 CMMI for Development

Capability Maturity Model Integration –mallit (CMMI), mukaan lukien CMMI for Development, ovat lähtöisin Capability Maturity Model-mallista (CMM). CMM pitää sisällään keinoja ja tapoja prosessien tehostamiseen. CMM keskittyy organisaatioiden sisäisten prosessien tehostamiseen. CMM:n avulla prosessit voidaan kehittää ad hoc -tyylisestä epäkypsästä prosessien suorittamisesta järjestelmälliseksi suorittamiseksi. Kehittyneet prosessit taas voidaan kehittää paremmiksi ja parempaan laatuun johtaviksi prosesseiksi. (Software Engineering Institute 2010, 5-6.)

CMMI syntyi tarpeesta päästä eroon ongelmista joita esiintyi monien eri CMM-mallien yhteiskäytön yhteydessä. Muista malleista haluttiin kerätä halutut ominaisuudet yhdeksi kehityksen perustana toimivaksi viitekehykseksi. Ensimmäinen CMMI versio (V1.02) julkaistiin vuonna 2000. Ensimmäinen versio tehtiin puhtaasti ohjelmistokehitystyötä tekevien organisaatioiden tarpeita vastaavaksi prosessien kehittämisen malliksi. Myöhemmin, kun laajennosten käsite esitettiin CMMI-mallien yhteydessä, ensimmäisen CMMI-mallin nimi muutettiin CMMI-DEV:iksi. Myöhemmin CMMI onkin saanut kaksi laajennosta: CMMI for Acquisition, joka julkaistiin vuonna 2007 ja CMMI for Services vuonna 2009. Kaikki mainitut kolme CMMI-versiota ovat tällä hetkellä ehtineet versioon 1.3, jotka on kaikki julkaistu marraskuussa 2010. (Software Engineering Institute 2010, 6-7; Chrissis, Konrad & Shrum 2011, 10-11.)



Kuvio 8. CMM-versioiden kehityshistoria (Chrissis, Konrad & Shrum 2011, 11)

CMMI kuten, myös muut CMM-mallit, tarjoavat ohjeistusta prosessien kehittämistä varten. Ne eivät ole valmiita prosessimalleja tai prosessikuvauksia. CMMI.n mallit eivät sellaisinaan sovellu käytettäväksi prosessien kehittämiseen, koska prosessit vaihtelevat hyvin paljon organisaatiokohtaisesti. CMMI:n tarjoamia käytäntöjä voi olla tarpeen soveltaa ja ”räätälöidä” organisaation tarpeita ja prosesseja vastaaviksi. (Software Engineering Institute 2010, 5-8; Chrissis, Konrad & Shrum 2011, 10.)

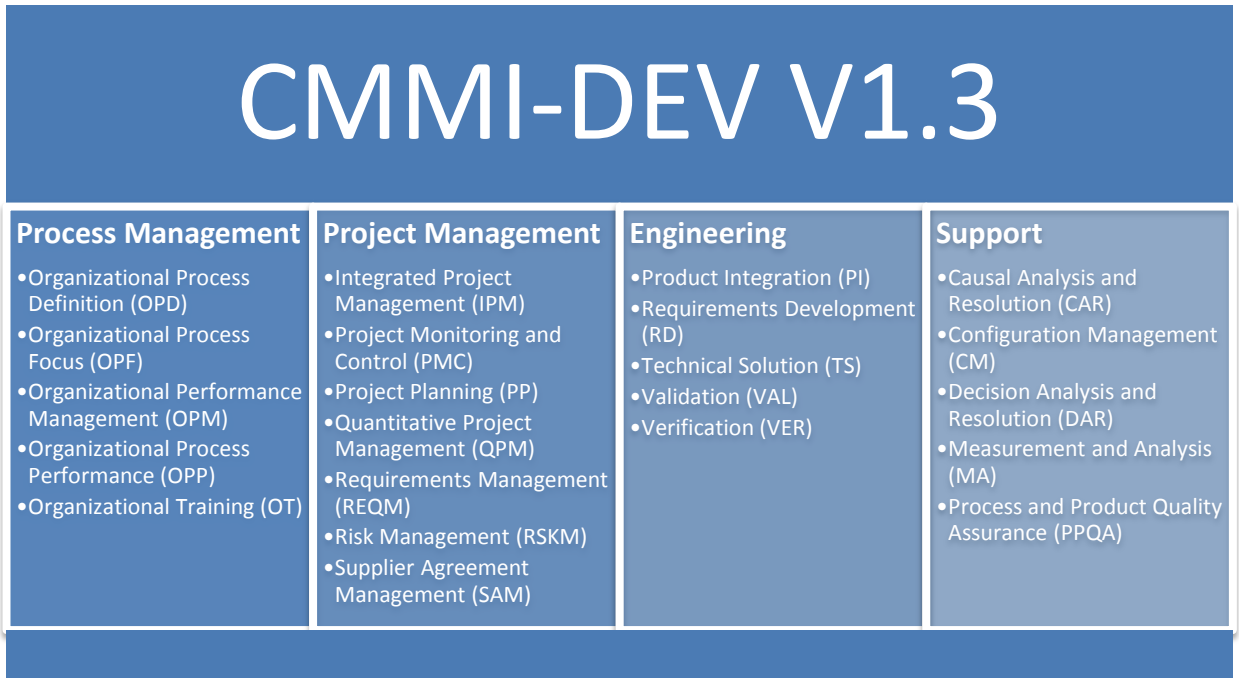
Jokainen CMMI-malli pitää sisällään 16 prosessikehityksen kannalta keskeistä ydinprosessia. Ydinprosessit ovat peräisin CMMI perusviitekehyksestä, joka on kaikkien CMMI mallien perusta. Laajennettujen mallien ydinprosessit voivat olla joko kokonaan tai osittain samanlaisia, kuin CMMI perusviitekehyksessä. Ydinprosesseja on voitu muuttaa laajennetuissa malleissa paremmin tietyn kohdealueen tarpeita vastaaviksi. (Software Engineering Institute 2010, 9; Chrissis, Konrad & Shrum 2011, 19.)

3.1 Prosessikategoriat ja prosessialueet

CMMI for Development (CMMI-DEV) pitää sisällään kehitysmalleja sekä tuotteiden, että palveluiden kehittämistä varten. CMMI-DEV sisältää käytäntöjä projektien johta-

misen, projektien hallinnan, järjestelmäkehityksen, laitekehityksen, ohjelmistokehityksen sekä kehityksen tuki- ja ylläpitoprosesseja varten. (Software Engineering Institute 2010, 7- 8.)

CMMI-DEV sisältää CMMI:iin sisältyvien 16 ydinprosessialueen lisäksi 5 ohjelmistokehitystyökohtaista prosessialuetta sekä yhden muun prosessialueen (Software Engineering Institute 2010, 9). CMMI for Development jakautuu neljään prosessialueeseen ja edelleen osaprosesseihin kuvion 9 mukaisella tavalla.



Kuvio 9. CMMI-DEV 1.3 prosessialueet (Chrissis & Konrad & Shrum 2011, 49)

Prosessialueiden kuvaukset sekä prosessialueiden kehittämisen tavoitteet on esitelty liitteessä 1.

3.2 Kehittämisen elementit ja tasot

CMMI:ssä kehittämessä käytettävät komponentit on jaettu kolmeen eri kategoriaan: vaaditut komponentit (required components), odotetut komponentit (expected components) ja informatiiviset komponentit (informative components). Kehittämisen komponentit kuvataan seuraavalla (Software Engineering Institute 2010, 9-10; Chrissis, Konrad & Shrum 2011, 19-20.) tavalla:

- **Vaaditut komponentit** (Required components). Komponentit ovat ensiarvoisen tärkeitä, jotta voidaan kehittyä jollain tietyllä prosessialueella. Komponentit voivat olla prosessialueen joko yleisiä tai erityisiä tavoitteita, jotka on tuotava näkyvästi esille jotta haluttu taso voidaan saavuttaa.
- **Odotetut komponentit** (Expected components). Kuvaavat aktiviteettejä jotka ovat tärkeitä tavoitteen saavuttamisen kannalta. Komponentit ohjaavat käytännön kehitystyötä tai niitä jotka arvioivat prosessin kehittymistä. Ovat prosessialueen yleisiä tai erityisiä tavoitteita jotka on saavutettava, joko siten kuten ne on kuvattu, tai jollakin vaihtoehtoisella hyväksytyllä tavalla.
- **Informatiiviset komponentit** (Informative components). Nimensä mukaisesti ovat informatiivisessa roolissa CMMI-prosesseissa. Käytännössä komponentit voivat olla esimerkiksi yksityiskohtaisia kuvauksia muista komponenteista tai esimerkkejä valmiista komponenteista. Komponenttien tarkoituksena on auttaa ymmärtämään mallia ja prosessialueen tavoitteita paremmin.

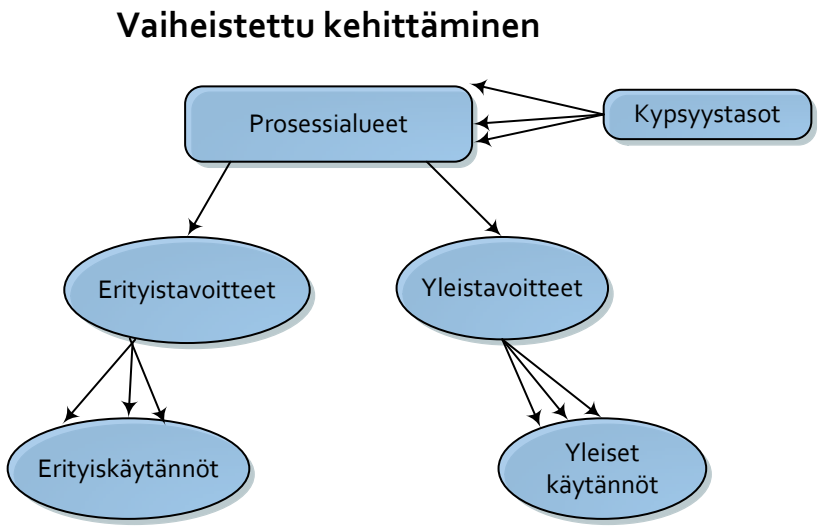
Merkittävänä osana CMMI-malliin liittyvät myös prosessialueiden erityistavoitteet (specific goal) ja yleistavoitteet (generic goal). Erityistavoitteiden tulee täyttyä, jotta taso voidaan katsoa saavutetuksi. Erityistavoitteisiin liittyy erityiskäytäntöjä (specific practice) ja yleisiin tavoitteisiin yleiskäytäntöjä (generic practice). Käytännöt kuvaavat aktiviteetteja joita odotetaan suoritettavan prosessissa, jotta prosessin tavoitteet saavutetaan. Yleistavoitteet ovat luonteeltaan sellaisia, että ne toistuvat samankaltaisina useilla eri prosessialueilla. Erityiskäytännöt koskevat vain tiettyä prosessialuetta. Erityistavoitteet ja yleistavoitteet ovat odotettuja komponentteja ja siten vaadittuja tavoitellun kypsyys- tai kyvykkyystason saavuttamiseksi. (Software Engineering Institute 2010, 12-13; Chrissis, Konrad & Shrum 2011, 22-24.)

CMMI on prosessien kehittämisen suhteen joustava, eikä pakota noudattamaan kehitystyössä tiettyä järjestystä prosessien tai prosessialueiden suhteen. Organisaatio voi halujensa ja tavoitteidensa mukaisesti päättää kehittääkö useampia toisiinsa liittyviä prosessialueita vai vain yhtä prosessialuetta. Olipa kehitystapa kumpi vain, liittyy kehitykseen kiinteästi sekä kyvykkyystasot (capability), että kypsyystasot (maturity). Erona näillä on se, että kyvykkyystasoja savutetaan jatkuvan kehittämisen mallilla (continuous

representation) ja kypsyystasoja puolestaan vaiheistetun kehitysmallin (staged representation) mukaisesti. (Software Engineering Institute 2010, 21- 22; Chrissis, Konrad & Shrum 2011, 31-32.) Erona kyvykkyys- ja kypsyystasojen saavuttamisella on se, että kypsyystasojen saavuttamiseksi on täytettävä kaikki yleisten ja erityistavoitteiden vaatimukset (Software Engineering Institute 2010, 29; Chrissis, Konrad & Shrum 2011, 45). Kyvykkyystasot puolestaan saavutetaan suorittamalla yleisten tavoitteiden asettamat toiminnot tai täyttämällä niiden vaatimukset jollain vaihtoehtoisella tavalla (Software Engineering Institute 2010, 25; Chrissis, Konrad & Shrum 2011, 36).

3.2.1 Vaiheistetun kehittämisen malli

Vaiheistetun kehittämisen mallissa kypsyystasoa vertaillaan laajemmassa kontekstissa, kuin jatkuvan kehittämisen mallissa kyvykkyystasoja. Vaiheittaisessa kehittämisessä tarkastellaan prosessialueiden suhdetta koko organisaation toimintaan ja kypsyystaso määräytyy kokonaisuuden perusteella. (Software Engineering Institute 2010, 23; Chrissis, Konrad & Shrum 2011, 32.)



Kuvio 10. CMMI-DEV vaiheistetun kehityksen malli (Chrissis, Konrad & Shrum 2011, 33)

Vaiheistetussa kehittämisessä kypsyystasot koostuvat toisiinsa liittyvistä erityisistä ja yleisistä tavoitteista. Kypsyystasot tarjoavat näin organisaatiolla mahdollisuuden havainnollistaa suorituskykyään kypsyystasojen avulla. Prosessien kehittäminen on tehokkaampaa silloin kun kehittämisessä keskitytään riittävän pieneen määrään prosesseja

samanaikaisesti ja kehittämällä niitä pitkäjänteisesti hienostuneempaan ja parempaan suuntaan. Kypsyystasot kuvaavat kypsyyttä välillä 1-5. Kyvykkyys- ja kypsyystasojen erona on se, että kypsyystaso kuvaa kehittymistä koko organisaation tasolla, kun taas kyvykkyystaso kuvaa kehittymistä valitulla tietyllä prosessialueella. Kypsyystasojen kuvaukset (Software Engineering Institute 2010, 26-29; Chrissis, Konrad & Shrum 2011, 41-44.) lyhyesti:

- Kypsyystaso 1: Alkuvaihe (Initial). Prosessi on luonteeltaan hyvin kaoottinen ja ad hoc-tyylinen, eli käytännön prosessin suoritus vaihtelee hyvin suuresti. Usein prosessin onnistuminen riippuu yksittäisten henkilöiden valtavasta panostuksesta muihin nähden. Luonteenomaista tasolle 1 on se, että kriisin, kuten aikataulupaineiden esiintyessä, prosessin toimintatavat hylätään eikä onnistuneita prosessin suorituksia pystytä toistamaan.
- Kypsyystaso 2: Hallittu (Managed). Prosessi suoritetaan suunnitelmallisesti, prosessilla on käytössään tarvittavat resurssit ja asianosaiset osallistetaan prosessiin. Prosessia monitoroidaan, kontrolloidaan ja arvioidaan. Tasolla 2 pyritään varmistamaan prosessin onnistunut suorittaminen myös kriisitilanteissa, jotta voitaisiin välttyä tasoon 1 liittyviltä ongelmatilanteilta.
- Kypsyystaso 3: Määritelty (Defined). Lähtökohtana tasolle 3 ovat organisaation prosessistandardit, joilla varmistetaan prosessien yhdenmukaisuus organisaatiossa. Tasolla 3 prosessi on hyvin määritelty sekä ymmärretty. Prosessi on vakaampi kuin tasolla 2, eikä prosessissa esiinny vaihtelua samalla tavalla, kuin tason 2 prosessissa.
- Kypsyystaso 4: Määrällisesti hallittu (Quantitatively Managed). Prosesseille on määritelty kvantitatiiviset tavoitteet laadulle ja suorituskyvylle joiden mukaan projekteja hallitaan organisaatiossa. Laatua ja suorituskykyä hallitaan tilastollisesti projektien ja prosessien suoritusten elinkaaren ajan. Prosesseille on määritelty mittarit joita analysoidaan. Erona tasoon 3 on se, että tasolla 4 prosessin suorituskykyä pystytään ennustamaan mittausdatan perusteella. Määriteltäessä mittareita aliprosesseille, on merkittävää ymmärtää niiden vaikutus ylemmän tason prosessin suorituskykyyn.

- Kypsyystaso 5: Optimoituva (Optimizing). Organisaation prosessit kehittyvät jatkuvasti liiketoiminta- ja suorituskykytarpeiden mukaan. Prosesseille on asetettu suorituskyky- ja laatutavoitteet joita seurataan jatkuvasti. Tarvittaessa niitä myös muutetaan. Erona tasoon 4 on se, että tasolla 5 keskitytään hallinnointiin ja koko organisaation suorituskyvyn parantamiseen, kun tasolla 4 keskityttiin prosessien ja aliprosessien väliseen suorituskykyyn. Tasolla 5 suorituskykymittauksen dataa kerätään useasta eri projektista, kun tasolla 4 prosessien ja aliprosessien välinen mittausdata koski yhtä projektia ja sen suoritusta.

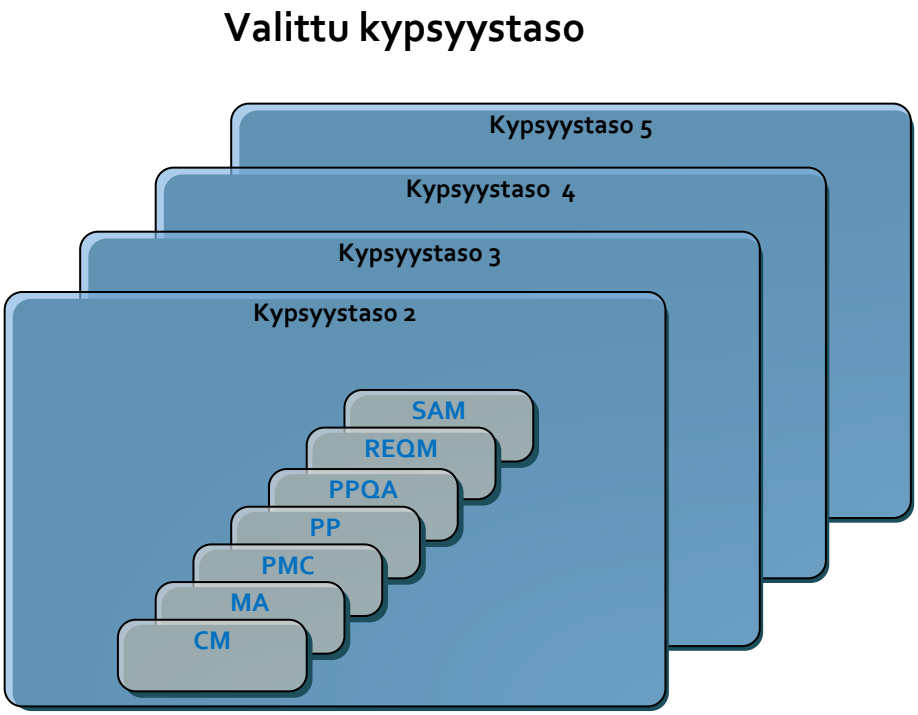
Taulukossa 2 on esitetty prosessialueet kypsyystasoittain.

Taulukko 2. CMMI-DEV prosessialueet, kategoriat ja kypsyystasot (Chrissis, Konrad & Shrum 2011, 49)

Prosessialue	**Kategoria**	**Kypsyystaso**
Causal Analysis and Resolution (CAR)	Support	5
Configuration Management (CM)	Support	2
Decision Analysis and Resolution (DAR)	Support	3
Integrated Project Management (IPM)	Project Management	3
Measurement and Analysis (MA)	Support	2
Organizational Process Definition (OPD)	Process Management	3
Organizational Process Focus (OPF)	Process Management	3
Organizational Performance Management (OPM)	Process Management	5
Organizational Process Performance (OPP)	Process Management	4
Organizational Training (OT)	Process Management	3
Product Integration (PI)	Engineering	3
Project Monitoring and Control (PMC)	Project Management	2
Project Planning (PP)	Project Management	2
Process and Product Quality Assurance (PPQA)	Support	2
Quantitative Project Management (QPM)	Project Management	4
Requirements Development (RD)	Engineering	3
Requirements Management (REQM)	Project Management	2
Risk Management (RSKM)	Project Management	3
Supplier Agreement Management (SAM)	Project Management	2
Technical solution (TS)	Engineering	3
Validation (VAL)	Engineering	3
Verification (VER)	Engineering	3

Kypsyystasojen puolesta on valmiiksi määritelty mitä prosessialueita tulee kehittää, jotta tietty kypsyystaso voidaan saavuttaa. Edettäessä kypsyystasoilla korkeammalle, tulee 2.

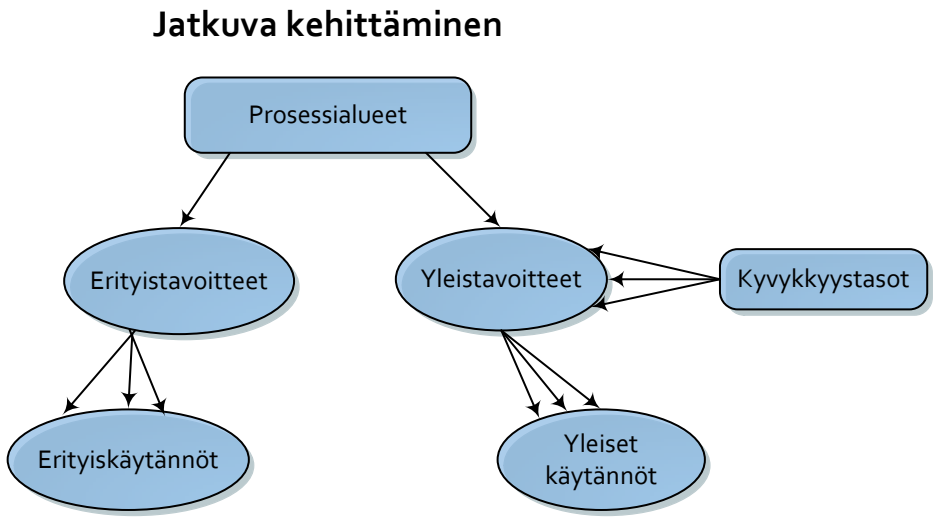
ja 3. tason saavuttamiseksi olla 2. tason yleistavoitteiden olla saavutettu. 3.-5. tason saavuttamiseksi tulee olla myös 3. tason yleistavoitteiden saavutettu. (Software Engineering Institute 2010, 31-32; Chrissis, Konrad & Shrum 2011, 46-48.)



Kuvio 11. CMMI-DEV-prosessialueet vaiheistetun kehityksen mallissa (Chrissis, Konrad & Shrum 2011, 47)

3.2.2 Jatkuvan kehittämisen malli

Jatkuvan kehittämisen mallissa kehityksen konteksti ja laajuus on suppeampi, kuin vaiheittaisen kehityksen mallissa. Jatkuvassa kehittämisessä kehittymistä tarkastellaan tiettyjen valittujen prosessien näkökulmasta, eikä toisiinsa yhteen liittyvinä asioina kuten vaiheistetun kehityksen mallissa. (Software Engineering Institute 2010, 23; Chrissis, Konrad & Shrum 2011, 32).



Kuvio 12. CMMI-DEV jatkuvan kehityksen malli (Chrissis, Konrad & Shrum 2011, 33)

Kyvykkyystasot kuvaavat kyvykkyyttä välillä 0-3 sen mukaan kuinka prosessi on suunniteltu ja mitä prosessi pitää sisällään. Kyvykkyystaso on saavutettu, mikäli kaikki tason yleistavoitteiden vaatimukset on täytetty. Yleistavoitteiden osalta on huomioitava, että niiden osalta tason 2 ja 3 vaatimukset ovat täsmälleen samoja. Kyvykkyystasot voidaan kuvata (Software Engineering Institute 2010, 24-25; Chrissis, Konrad & Shrum 2011, 34-36.) seuraavasti:

- Kyvykkyystaso 0: Puutteellinen (Incomplete). Olemassa olevaa prosessia ei joko suoriteta ollenkaan tai sitä suoritetaan vain osittain. Prosessialueen erityistavoitteet jäävät suorittamatta joko osittain tai kokonaan.
- Kyvykkyystaso 1: Suoritettu (Performed). Prosessissa suoritetaan vaaditut työtehtävät jotta halutut tuotokset saadaan tuotettua. Tarkemmin määritellyt prosessille asetetut spesifiset tavoitteet saadaan täytettyä.
- Kyvykkyystaso 2: Hallittu (Managed). Prosessi on hallittu ja sitä suoritetaan suunnitellusti. Prosessi osallistaa kaikki asianosaiset henkilöt. Prosessia monitoroidaan ja arvioidaan tarpeen mukaan.
- Kyvykkyystaso 3: Määritelty (Defined). Prosessi on räätälöity soveltumaan yhteensopivaksi organisaation toimintatapojen kanssa. Tasolla 3 prosessi on tarkemmin määritelty sen suhteen mitkä ovat prosessin syötteet ja tuotokset verrattuna tason 2 prosessiin. Tasolla 2 prosessin suoritus voi vaihdella paljonkin

prosessin eri suorituskerroilla, kun taas tason 3 prosessi on paljon vakaampi ja vähemmän altis vaihtelulle.

Jatkuvassa kehittämisessä organisaatio voi valita keskittyvänsä prosessien kehittämisessä vain yhteen tiettyyn prosessialueeseen tai kokoelmaan toisiinsa liittyviä prosesseja. On täysin organisaation omassa päätäntävallassa kuinka he toimivat ja minkä prosessialueen tai prosessialueiden kehittäminen palvelee parhaiten heidän päämääriään. Kun on päätetty mitä prosessialueita halutaan kehittää, on päätettävä kuinka paljon kutakin prosessialuetta halutaan kehittää. Tämä on myös täysin organisaation oman harkinnan ja mielihalujen mukaista mihin päädytään. Yhtä aluetta voidaan päättää kehitettäväksi tasolle 2 ja toista voidaan päättää kehitettäväksi tasolle 3. Useimmiten kuitenkin asetetaan minimitasoksi kaikille prosesseille taso 1, joka edellyttää että prosessialueen erityistavoitteet saavutetaan. (Software Engineering Institute 2010, 31-32; Chrissis, Konrad & Shrum 2011, 46-48.)

Jatkuvan kehityksen mallissa valituista kehitettävistä prosessialueista tehdään organisaation kyvykkyydelle kyvykkyysprofiili (capability level profile). Tästä profiilista johdetaan saavutusprofiili (achievement profile), jolla seurataan kuinka valittujen prosessialueiden kyvykkyys on kehittynyt suhteessa tavoitteeseen. Samanlaista menetelmää voidaan hyödyntää myös vaiheistetun kehittämisen mallissa. Profiilia kutsutaan tällöin kypsyysprofiiliksi (maturity level rating). Periaatteessa profiilit ovat samanlaisia, kuin jatkuvan kehityksen mallissa, mutta prosessialueet ovat ennalta määritellyt. (Software Engineering Institute 2010, 34-37; Chrissis, Konrad & Shrum 2011, 49-51.)

Myös jatkuvan kehittämisen menetelmällä pystytään saavuttamaan korkea kypsyystaso. Tällöin prosessien kyvykkyyttä tulee verrata halutun kypsyystason prosesseihin. Jotta haluttu kypsyystaso voidaan saavuttaa prosessialueiden kyvykkyyden perusteella, on kaikkien tietyn kypsyystason prosessien oltava kypsyystasolla 3. Kypsyystaso 4 saavuttamiseksi tulee kaikkien tasoon 4 vaadittujen prosessialueiden olla kyvykkyystasoltaan 3. (Software Engineering Institute 2010, 37; Chrissis, Konrad & Shrum 2011, 53.)

Kuvio 13. CMMI-DEV-prosessialueet jatkuvan kehityksen mallissa (Chrissis, Konrad & Shrum 2011, 47)

Jatkuvan kehittämisen ja vaiheistetun kehittämisen tasot poikkeavat hieman toisistaan. Jatkuvan kehityksen ja vaiheistetun kehitykset tasot on esitetty taulukossa 3.

Taulukko 3. Vaiheistetun ja jatkuvan kehityksen tasot. (Chrissis, Konrad & Shrum 2011, 34)

Taso	**Jatkuvan kehittämisen kyvykkyystasot**	**Vaiheistetun kehittämisen kypsyystasot**
Taso 0	Puutteellinen	
Taso 1	Suoritettu	Alkuvaihe
Taso 2	Hallittu	Hallittu
Taso 3	Määritelty	Määritelty
Taso 4		Määrällisesti hallittu
Taso 5		Optimoituva

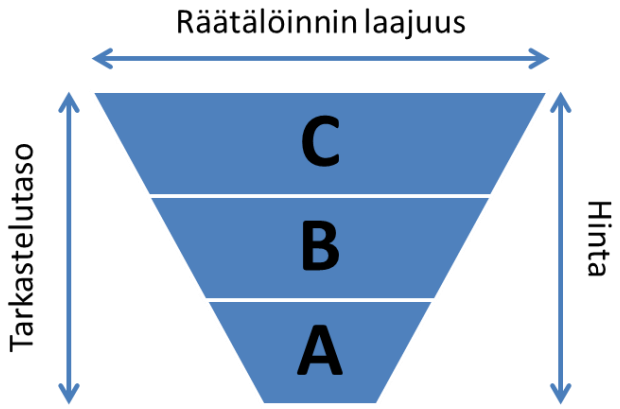
3.3 SCAMPI-arviointi

SCAMPI (Standard CMMI Appraisal Method for Process Improvement) on CMMI-arviointimenetelmä. Menetelmän avulla on tarkoitus varmistaa, että tietyt kypsyystason edellyttämät vaaditut käytännöt on organisaatiossa toteutettu. SCAMPI-arviointi ei ole varsinaisesti auditointityylinen arviointi tai testi kuinka prosessit käytännössä toimivat. Arviointia kuvataankin enemmän sanalla prosessikertomus (process story). Kertomuk-

sesta käyvät ilmi mm. käytetty CMMI-malli, minkälaisia kehitystoimenpiteitä tehdään ja kuinka ne sovitetaan organisaation kulttuuriin. Lisäksi pitää pystyä esittämään kehitystoimenpiteet koko organisaation tasolla ja osallistujien pitää myös itse pystyä esittämään tehtävät kehitystoimenpiteet. (Glazer, Dalton, Anderson, Konrad & Shrum 2008, 14.)

Käytännössä on arviointitiimin tehtävänä objektiivisesti arvioida täyttyvätkö ehdot kypsyystason saavuttamiseksi ja ovatko vaaditut käytännöt toteutettu. Tehtävä voi olla arviointitiimille hyvinkin haasteellinen, sillä käytännöt voivat vaihdella hyvin paljon ja käytännön toteutukset on suunniteltu toimimaan tietyn arvioitavan organisaation kontekstissa. Organisaatiossa voidaan nimellisesti toimia jonkin viitekehyksen puitteissa, mutta siihen on voitu tehdä muutoksia, jotta se soveltuisi organisaatioon paremmin. (Glazer ym. 2008, 15.)

SCAMPI-arviointeja on 3 eri tasoa: A, B ja C. Merkittävin ero eri tasojen arvioinneissa on se, että ainoastaan A-tason arvioinnista voi saada arvosanan. A-tason arviointi on arvioinneista kriteereiltään tiukin ja eniten resursseja sitova. Arviointia suorittava tiimi on suurin A-tason arviointia tehtäessä ja samoin suurimmat ovat myös arvioinnista koituvat kustannukset. B- ja C-tason arvioinnit ovat vähemmän muodollisia ja voivat toimia välietappeina kohti virallista kypsyys- tai kyvykkyystason saavuttamista. (Software Engineering Institute 2012b.)



Kuvio 14. SCAMPI-arvioinnin tasot (Software Engineering Institute 2012b)

3.4 CMMI & scrum ja XP

CMMI on yhteensopiva monien eri menetelmien ja viitekehysten kanssa. Sovittamalla yhteen CMMI ja yksi tai useampia yhteensopivia menetelmiä voidaan mahdollisesti saavuttaa suurempia hyötyjä, kuin käyttämällä jotain menetelmää yksin. Yhteensopivia menetelmiä ovat mm. RUP, Lean, scrum sekä lukuisat muut ketterät menetelmät. (Software Engineering Institute 2012a.) Ketterien menetelmien ja CMMI:n yhdistämisellä voidaan saavuttaa hyötyjä jotka voivat jäädä saavuttamatta ilman menetelmien yhdistämistä. Vaikka ketteriä menetelmiä hyödynnettäisiin oikein, voi kohtalona olla epäonnistuminen, mikäli käytäntöjä ei voida toistaa yhdenmukaisesti. Myös optimaalinen suorituskyky voi jäädä saavuttamatta, mikäli prosessi on puutteellisesti toteutettu. Prosessin käytännön toteuttamisessa voi olla hyötyä ketteristä menetelmistä. (Sutherland, Jakobsen & Johnson 2007, 273.) CMMI:n ja ketterien menetelmien yhdistäminen on mahdollista ja CMMI:n avulla voidaan ketterin menetelmiin saada kaivattua kaavamaisuutta ja vakautta (Baker 2006, 154).

Scrumin ja CMMI:n yhteensovittaminen sujuu joitakin osin hyvin vaivattomasti. CMMI:n tavoitteita vastaavat aktiviteetit ja toiminnot on joiltain osin löydettävissä suoraan scrumin menetelmistöstä. Puuttuvien vaatimusten täyttäminen edellyttää muutoksia scrumin käytäntöihin, mikäli CMMI:n vaatimukset halutaan täyttää. Scrumin etuna voidaan tässä tapauksessa nähdä hyvin kevyt prosessi joka sallii muutokset prosessiin tarvittaessa. (Potter & Sakry 2011.) Samoin myös CMMI on muutosten suhteen joustava, sillä CMMI:ssä painotetaan että implementaatiosta tulee tehdä organisaation omien käytäntöjen ja tarpeiden mukainen. CMMI asettaa päämäärät, mutta ei kerro miten niihin tulee pyrkiä. On kunkin organisaation omassa päätäntävallassa, kuinka CMMI:n vaatimukset haluaa lopulta täyttää. Keinot vaatimusten täyttämiseksi tulevat siis scrumista. Niiltä osin kun scrumista ei suoraan löydy vastaavuutta prosessialueen vaatimuksiin, voidaan CMMI:n ajatella laajentavan scrumin käytäntöjä. (Foegen, 2010, 2-3.)

Osa CMMI:n prosessialueista on täysin scrumin vaikutusalueen ulkopuolella. Tämä voi osoittautua ongelmalliseksi, mikäli CMMI ja scrum halutaan yhdistää vaiheistettua kehitysmallia käyttäen ja siten kypsyyttä kehittäen. Tietyn prosessialueen kyvykkyyden kehittäminen voi sen vuoksi osoittautua helpommaksi tavaksi, varsinkin mikäli vaatimuk-

sille on jo löydettävissä suorat vastaavuudet scrumin menetelmistöstä. (Potter & Sakry 2011.)

CMMI:n yleisten vaatimusten 3. tason vaatimukset eivät suoraan täyty scrumin käytännöin. Kolmannen kypsyystason vaatimuksina on mm. se että projektien tietämystä jaetaan eri projektien kesken yhteisen tietämyskannan välityksellä. (Potter & Sakry 2011.) Suoranaisesti tämä tosin ei ole ketterien menetelmien periaatteiden mukaista, jossa suositaan enemmän kasvokkain kommunikointia ja välitöntä palautetta, kuin organisoidumpaa kollektiivista palautetta (Salo & Abrahamsson 2006, 82-83). Scrum ja CMMI saadaan sovitettua myös 3. tason vaatimusten osalta yhteen, mutta se vaatii scrumin peruskäytäntöjen muokkaamista. (Potter & Sakry 2011.) Näiden vaatimusten täyttäminen voi tehostaa entuudestaan ketterien menetelmien hyötyjä. Vastaavasti CMMI:tä jo entuudestaan hyödyntävässä organisaatiossa ketterät menetelmät voivat puolestaan tuoda tehostusta prosesseihin. (Shelton 2008; Sutherland ym. 2007, 278.) Myös Extreme Programming ja CMMI:n 3. kypsyystason välillä voidaan havaita yhteneväisyyksiä. (Shelton 2008). XP:n menetelmistö pariohjelmoinnin ja testausta painottavan lähestymistavan ansiosta tukee hyvin verifioinnin ja validoinnin prosessialueiden tavoitteita (Fritzsche & Keil 2007, 17-18).

Osa kypsyystaso 3:sta ylöspäin olevista prosessialueista vaatii joiltakin osien ketterien menetelmien periaatteista luopumista. Tämä ei välttämä ole tarkoituksenmukaista ja on ristiriitaista sekä ketterien menetelmien, että myös CMMI:n kannalta. Toisessa tapauksessa joudutaan luopumaan ketteryyden periaatteista ja menetelmistä ja toisessa voidaan joutua luopumaan itse menetelmän tärkeimmästä päämäärästä: prosessien kehittämisestä. Muuttamalla ketterät menetelmät vähemmän ketteriksi, eivät kyseessä enää ole ketterät menetelmät. Ketteriä menetelmiä hyödyntävässä organisaatiossa voi näin ollen olla tarkoituksenmukaista pyrkiä CMMI:n kypsyystasolla ainoastaan tasolle 3 asti. (Fritzsche & Keil 2007, 24.)

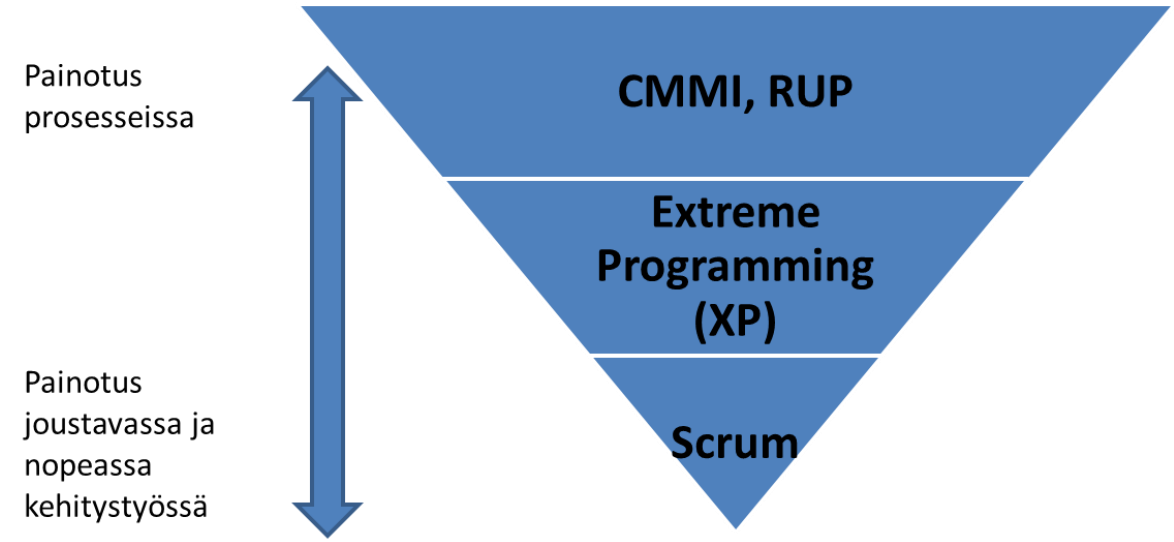
Scrumin lisäksi myös XP:n menetelmistöllä voidaan täyttää useita CMMI:n vaatimuksista. Joiltain osin XP täyttää CMMI:n vaatimukset paremmin kuin scrum, ja päinvastoin. Täydellistä CMMI:n vaatimukset täyttävää yhdistelmään ei saa XP:tä ja scrumia yhdistelemällä, mutta hyvä alku se on CMMI:n jalkauttamista varten ketteriä menetel-

miä hyödyntävässä organisaatiossa. (Fritzsche & Keil 2007, 23.) Taulukko CMMI-DEV:in, XP:n ja scrumin yhteensopivuudesta löytyy liitteestä 5.

CMMI:n ja ketterien menetelmien yhdistämisestä löytyy mielipiteitä sekä yhdistämisen puolesta, että sitä vastaan. Ketterien menetelmien puolestapuhujat voivat nähdä CMMI:n olevan liian raskas ja turhaa työtä sisällään pitävän menetelmän. (Shelton 2008). Ensi vaikutelma CMMI:stä voi olla se, että se vaatii todella paljon dokumentaatiota. Näin ei todellisuudessa välttämättä ole. CMMI ei suoranaisesti kerro mitä tuotoksia täytyy olla vaatimusten saavuttamiseksi. Joitain ehdotuksia se tosin antaa. Organisaation on näin ollen mahdollista itse määrittää mikä on riittävä taso prosessin muodollisuudelle ja dokumentaatiolle CMMI:n yhteydessä. (Baker 2006, 149.) Ongelmia voi tulla vastaan niiden CMMI:n vaatimusten osalta, jotka suorasanaisesti vaativat kirjallista aineistoa vaatimusten läpäisemiseksi. Tämä osaltaan sotii ketterien menetelmien periaatteita vastaan jossa päämääränä on tuottaa toimivia ohjelmistoja ja dokumentaatio pyritään pitämään minimissään ja tuottamaan vasta niin myöhään, kuin on mahdollista. (Pikkarainen & mäntyniemi 2006.)

CMMI:n puolestapuhujat kritisoivat ketteriä menetelmiä siitä, että niistä puuttuu kontrolli ja hallinta, joka johtaa dokumentoimattomiin muutoksiin ja kaoottiseen tilanteeseen. CMMI:n avulla voidaan saada ennustettavuutta ketterään kehitysprosessiin sen ennakoitavasta suorituskyvystä. (Shelton 2008.) Vaikkei CMMI:n käytön päämääränä välttämättä olisikaan CMMI-arviointi ja tietyn kypsyystason saavuttaminen, voidaan ketterien menetelmien ja CMMI:n yhdistämisellä varmistaa, että prosessit ovat tehokkaita (Pikkarainen & Mäntyniemi 2006).

Ketterät menetelmät ja CMMI voidaan suhteuttaa toisiinsa sen mukaisesti, kuinka paljon ne painottavat tiettyjä toimintatapoja ja asettavat sitä kautta tiettyjä rajoituksia toiminnalle. Ketterissä menetelmissä vaatimuksia ja rajoitteita on huomattavasti vähemmän, kuin tarkemmin määritellyillä kehitystyön menetelmillä (Kniberg & Skarin 2010, 7-9). Prosessien ja ennalta asetettujen vaatimusten suhde nopeasti mukautuvaan kehitykseen eri menetelmissä on esitetty kuviossa 15.



Kuvio 15. Ketteryys vs prosessit (Eickmann 2008)

Ketterät menetelmät mahdollistavat prosessin kehittämisen nopeasti ja näkyvästi. Ketterien menetelmien periaatteisiin kuuluu prosessin arviointi ja sen parantaminen tietyin aikavälein. (Salo & Abrahamsson 2007, 81.)

3.5 CMMI ja Lean

Vaikka Lean onkin enemmän ajattelutapa kuinka hallita varastoja ja tehtävää työtä, kuin prosessien kehittämismenetelmä, voidaan sen yhteensopivuutta tarkastella CMMI:n kanssa (Hertneck 2009, 13). Verrattaessa Lean-ajattelun päämääriä CMMI:n prosessialueisiin ja niiden päämääriin voidaan havaita, että niistä löytyy toinen toisiaan tukevia elementtejä. Lean-ajattelussa on asiakaskeskeisyys hyvin tärkeässä osassa ja CMMI:n käsitteistöstä asiakkaan jatkuva osallistaminen prosessiin voisi osua esimerkiksi validoinnin (validation) prosessialueeseen. Samoin myös Lean-ajattelun aikainen verifiointi voidaan saada hoidetuksi CMMI:n ehdottamalla vertaisarvioinnin (peer reviews) avulla. (Hertneck 2009, 18.) Myös muita Lean-ajattelun tavoitteita tukevia prosessialueita on CMMI:n avulla mahdollista kehittää.

CMMI myös tukee Lean-ajattelun mukaista jatkuvaa kehittämistä myös koko organisaation kattavalla tasolla, sekä muilla menetelmillä jotka mahdollistavat ja auttavat kehityksen analysoinnissa. (Hertneck 2009, 23.) CMMI:n mekanismeilla voidaan myös välttää osa Lean-ajattelun mukaisesta jätteestä. Lisäksi CMMI antaa Lean-kehitykseen valmiin

kehyksen prosessien kehittämistä varten, aina projekti- ja tiimitasosta aina koko organisaation kattavalle tasolle. (Hertneck 2009, 30.)

CMMI:n dokumentaatiota prosessikuvauksineen voisi pitää Leanissa jopa ylimääräisinä prosesseina – eli jätteenä – koska ne dokumentaation voitaisiin kokea olevan ylimääräistä turhaa työtä (Poppendieck 2010a, 5-6). Tämä ei kuitenkaan välttämättä pidä paikkaansa. CMMI on joustava sen suhteen, minkälaista dokumentaatiota organisaation tulee tuottaa ja mikä on dokumentaation riittävä taso. Käytännössä organisaatio pystyy hyvin pitkälle itse määrittelemään mikä on organisaatiolle riittävä dokumentaation taso, eikä CMMI:n mukainen dokumentointi ole pakosti ja yksiselitteisesti jätettä. (Baker 2006, 149.)

4 Laatu

4.1 Laadun määritelmä

Laadulla tarkoitetaan ohjelmistotuotteiden yhteydessä toisaalta ohjelmistotuotteen toiminnallisuutta vaatimuksiin nähden. Toisaalta laadun voidaan nähdä myös tarkoittavan sitä kuinka hyvin tuote vastaa asiakkaan toiveita ja odotuksia. Näillä kahdella näkökulmalla on selkeät erot toisiinsa nähden. Ensimmäisen näkökulman mukaan tuotteen määritysten virheellisyyttä, joka johtaa puutteelliseen tai virheelliseen ohjelmiston toimintaan, ei katsoa laatua heikentäväksi tekijäksi. Toisen näkökulman tähdätessä asiakkaan todellisten mahdollisesti määrittelemättömien tarpeiden täyttämiseen voivat kehittäjät joutua panostamaan aikaansa huomattavissa määrin todellisten tarpeiden esiin kaivamiseksi. Tämä voi myös johtaa siihen, että lopullinen tuote on toiminnallisuudeltaan kaukana alkuperäisten määritysten mukaisesta toiminnallisuudesta, kun toiminnallisuus muuttuu ja tarkentuu koko kehitysprojektin ajan. (Galin 2004, 24-25.) Tämän pohjalta asiakasvaatimusten täyttymistä ei voida pitää laadullisen tarkastelun kohteena johtuen siitä, että vaatimukset voivat olla hyvin epämääräisiä ja muuttuvia. Tällaisten vaatimusten täyttäminen on usein hyvin hankalaa. Laatua voidaan siis verrata vain sellaisiin ominaisuuksiin tai toiminnallisuuksiin jotka on selkeästi määritelty ja dokumentoitu. (Schulmeyer 2008, 7.) Käytettäessä tuotteen ominaisuuksien ja toiminnallisuuksien vastaavuutta määrittelyihin laadunvalvonnan kriteerinä on huomioitava, että virheelliset määritykset ovat syyllisiä noin 20% virheellistä toiminnallisuutta (defect). Vakavien virheiden osalta määrä on vielä huomattavasti suurempi, sillä niiden osalta virheet vaatimuksessa johtavat 35%:iin vakavia virheitä. (Jones 2010, 559.)

Laatu on seurausta niistä toiminnoista joita joudutaan tekemään ohjelmistoprojektien yhteydessä projektin loppuun saakka. Näitä toimintoja ovat määrittely, suunnittelu, toteutus eli varsinainen ohjelmointityö sekä testaus. Kaikilla toiminnoilla on vaikutuksensa lopputuloksen laatuun, joskin joidenkin toiminnallisuuksien merkitys on laadun kannalta suurempi. (Schulmeyer 2008, 8.)

Ohjelmistojen laadun määritelmälle voidaan asettaa myös tiettyjä vaatimuksia. Jonesin (2010, 558-559.) mukaan laadulle voidaan asettaa seuraavia vaatimuksia:

- Laadun on oltava ennakoitavissa ennen projektin aloittamista.
- Laatuun tulee sisällyttää lähdekoodin lisäksi kaikki muut toimituksen sisältämät tuotteet.
- Laadun on oltava mitattavissa kehitystyön aikana.
- Laadun on oltava mitattavissa toimituksen jälkeen.
- Laadun on oltava asiakkaan huomattavissa ja tunnustettavissa.
- Laadun on pysyttävä yllä myös julkaisun jälkeen ylläpitovaiheessa.

4.2 Laadunvarmistus

Laadunvarmistuksella tarkoitetaan toimintamallia niille keinoille ja menetelmille, joilla varmistetaan, että ohjelmistotuote täyttää sille asetetut vaatimukset. Tämän lisäksi laadunvarmistus pitää sisällään ohjelmistojen kehitysprosessin evaluoinnin jonka mukaisesti tuotteiden kehitystyötä tehdään. (Galin 2004, 26.)

Laadunvarmistamisen keskeinen päämäärä on virheiden paikantaminen ja virheiden julkaisuversioon päätymisen estäminen. Virheiden paikantamisen merkittävyyttä ei voi liikaa korostaa, sillä virheiden korjaamiseen kuluu suurin osa ajasta (noin 80 % ajasta) ohjelmistoprojektien yhteydessä. (Lazic, Kolašinac & Avdic 2009, 37.)

Laadunvarmistuksessa ohjelmistokehitysprosessien kehittämisen lisäksi merkittävässä roolissa on myös organisaation henkilöstön kehittämispolitiikka. Toisaalta prosessien ja henkilöstön kehittämisen voidaan nähdä olevan osaamista ja sitä kautta laatua parantavana tekijänä, mutta toisaalta organisaatiot voivat nähdä varsinkin prosessien kehittämisen vain ylimääräisenä kustannuksena. (Lazic, Kolašinac & Avdic 2009, 21-22.)

4.2.1 Sisäinen ja ulkoinen laadunvarmistus

Laadunvarmistus voidaan toteuttaa joko sisäisesti tai ulkoisesti. Sisäinen laadunvalvonta tarkoittaa sitä, että tiimi tai yksikkö joka tekee ohjelmistojen toteutustyön, tekee myös laadunvarmistukseen liittyvät toimenpiteet. Ulkoinen laadunvarmistus puolestaan tarkoittaa sitä, että laadunvarmistuksen hoitaa jokin toinen tiimi, yksikkö tai organisaatio.

Sisäisen laadunvarmistuksen ongelmana on se, että toteuttajilla voi olla olettamuksia siitä millainen lopputuotteen pitäisi heidän mielestään olla. Tämän seurauksena ohjelmiston toiminta ei välttämättä vastaa odotettua ja jopa joitain virheitä voi jäädä huomaamatta. Tällöin tuote ei läpäisisi sille laadunvarmistusta varten asetettuja vaatimuksia. Näiden ongelmien vuoksi laadunvarmistuksen tulisi olla eri tiimin, yksikön tai organisaation vastuulla, kuin sen tiimin, jonka vastuulla varsinainen toteutustyö on. (Schulmeyer 2008, 16.) Pitämällä laadunvarmistus ulkoisena voidaan myös välttyä mahdollisilta pakotteilta joita voisi aiheutua laadunvarmistuksen ollessa sulautettuna kehitystiimin toimintoihin. Ulkoisella laadunvarmistuksella voidaan myös säilyttää laadunvarmistus objektiivisena, mikäli koko laadunvarmistus on kehityksestä erillisenä itsenäisenä yksikkönään aina ylimpiin esimiehiin asti. (Jones 2010, 282.)

Vaikka ulkopuolinen laadunvarmistus voisi tuoda objektiivisen näkökulman laadunvarmistukseen, on siinäkin ongelmia. Ulkopuolista laadunvarmistusorganisaatiota käytettäessä ei laadunvarmistusorganisaatio voi varmistua siitä, että laatu on ”sisään rakennettu”. Laatu rakennetaan ohjelmistoihin päivittäisessä kehitystyössä hyvien toimintatapojen ja käytäntöjen avulla. Siihen ei ulkopuolinen organisaatio pysty. Ulkopuolinen organisaatio voi lähinnä arvioida laadun, mutta ei varmistaa sitä. (Kruchten 2011, 194.)

Laadunvarmistuksessa voidaan siis yhtenä ohjaavana ajatuksena pitää sitä, kuinka laatu saadaan sisään rakennettua tuotteeseen eikä sitä, kuinka hyvin laatua voidaan arvioida tai evaluoida. Kun tuote on saatu toimitettua, ylläpitovaihe ei saa heikentää tuotteen laatua. Ylläpitovaiheelle luonteen omaisesti, sen aikana tehdyt virhekorjaukset monesti tekevät ohjelmakoodista ”spaghettikoodia” ja näillä on usein laatua heikentävä vaikutus. Yleisesti laadunvarmistuksen toimenpiteitä ei ole tarpeen toteuttaa kokonaisuudessaan jokaisen virhekorjauksen osalta. Rajana uudelleen varmistukselle on pidetty sitä hetkeä, jolloin muuttuneiden moduulien määrä kohoaa 30%:iin. Mikäli muuttuneiden moduulien määrä kohoaa 67%:iin on aika uudelleen suunnitella tai –kirjoittaa osa ohjelmakoodista. Näin suurella määrällä muutoksia on hyvin suuri mahdollisuus olla laatua heikentävä vaikutus. (Schulmeyer 2008, 9-10.)

5 Muutoksen aikaansaaminen

Muutos ei ole uusia asia. Ihmiskunnan historiassa on tapahtunut muutoksia kautta aikojen ja samalla ihminen on löytänyt keinot suhtautua ja sopeutua niihin. Nykyajan työelämän muutoksessa erona menneisyyteen on se, että muutostahti on nopeutunut huimasti. Nopea muutostahti voi johtaa siihen, ettei todellista muutosta tapahdu ja sopeutuminen jää hyvin näennäiseksi ja pinnalliseksi. Nopeutunut muutostahti näkyy selkeimmin työelämässä, jossa vaatimukset osaamisen suhteen ovat muuttuneet hyvin lyhyessä ajassa. Työelämän kannalta tämä tarkoittaa sitä, että ihmisen on opittava työuransa aikana yhä enemmän uusia tietoja ja taitoja. (Arikoski & Sallinen 2011, 7, 9.) Työelämän muutoksessa on huomioitava se, että vaikka tekninen johtaminen olisi hallittu hyvin, ei henkilöjohtamista sovi missään tilanteessa unohtaa ja jättää huomiotta. Mikäli inhimilliset asiat, kuten muutokseen liittyvät tunnetilat ja niiden käsittely laiminlyödään, voi lopulta koko muutos epäonnistua. (Arikoski & Sallinen 2011, 41.)

Myös jokainen ihminen on kohdannut elämässään muutoksia. Muutoksia ja uudenlaista sopeutumista vaativat tilanteet voivat olla esimerkiksi opiskeluiden alkaminen tai muutto uudelle paikkakunnalle. Muutos ei ole ainoastaan järkeistämällä läpikäyty prosessi vaan muutoksessa ihminen tulee käyneeksi läpi neljä perustunnetta: pelko, viha, suru ja ilo. Käsittelemällä ja käymällä läpi nämä tunteet, voi ihminen sopeutua muutokseen. Nämä tunteet eivät koske ainoastaan yksityiselämän muutoksia, vaan samanlaisia tunteita voivat herättää myös työelämän muutokset. (Arikoski & Sallinen 2011, 41.)

5.1 Prosessien kehittämisen merkittävyys

Ohjelmistokehityksen yhteydessä toimivat prosessit voivat osoittautua merkittävyydeltään yhtä tärkeiksi, kuin muilla liiketoiminta-alueilla. Mikäli ohjelmistokehitystiimillä ei ole määriteltyä prosessia ja toimintatapoja, on toiminta luonteeltaan ad hoc -tyylistä. Tällöin menestys riippuu hyvin pitkälti tiimille ja työn tuloksille omistautuneiden työjäsenten panostukselle ja liki sankarimaiset mittasuhteet saavuttaville urotöille. (Kruchten 2001, 15; Hertneck 2009, 16.) Muutamien henkilöiden omistautumiseen perustuvaa tilannetta ei voi pitää kovin kestävänä tilanteena pitkällä aikavälillä. Yhdenmukaisten toimintatapojen ja sitoutumisen puuttuessa prosessia ei voida luotettavasti tai ennustet-

tavasti toistaa muissa projekteissa. Vielä haastavammaksi tilanne muodostuu, kun kyseessä ovat hyvin monimutkaiset järjestelmät. Määritellyn prosessin puuttuessa voidaan menettää myös yksi merkittävä hyöty: toiminnan kehittäminen. Prosessin määrittely ei itsessään tulisi olla varsinainen päämäärä, vaan prosessia tulisi kehittää jatkuvasti, jotta toiminta ajan myötä kehittyisi aina tulevaisuuden tarpeiden mukaisesti. (Kruchten 2001, 15.) Tarkasteltaessa prosesseja kypsyyden näkökulmasta niin se, että prosessi on epäkypsä, ei välttämättä tarkoita sitä ettei määriteltyä prosessia olisi. Epäkypsä prosessi itsessään voi olla kuvattu, mutta kuvausta ei välttämättä vain noudateta. Tällöin myös prosessin kehittäminen on hankalaa ja kehittäminen onkin luonteeltaan enemmän reaktiivista, kuin proaktiivista. Ongelmakohtia prosessissa ei välttämättä havaita ajoissa tai niihin ei osata puuttua riittävällä vakavuudella ja ongelmat voivat jäädä pinnan alle kytemään. (Hertneck 2009, 16.)

Kruchtenin (2001, 15.) mukaan prosessin kehittämisellä voidaan nähdä olevan seuraavia päämääriä joihin prosessin kehittämisellä voidaan pyrkiä:

- Ohjeistaa kehitystiimiä mitä toimenpiteitä tehdään ja missä järjestyksessä.
- Määritellä minkälaisia tuloksia tulee olla valmiina ja millä aikataululla.
- Ohjata kehitystiimiä ja sen yksittäisiä jäseniä sen suhteen mitä tehtäviä tehdään missäkin vaiheessa kehitystä.
- Asettaa kriteerit minkä mukaan projektin tuloksia tai työskentelyä arvioidaan.

5.2 Muutoksen haasteet

Yksi suurimmista haasteista prosessien kehittämiseen liittyy motivaation puutteeseen. Mikäli johdolla ei ole riittävää motivaatiota toteuttaa muutoksia, ei motivaatiota usein ole muullakaan henkilöstöllä. Kun tehdään muutoksia, on odotettavissa vastarintaa. Vastarinnan syitä voi olla monia. Muutokset saatetaan kokea uhkaavina. Näin ollen on tarpeen tietää mitkä ovat vastarinnan syyt, jotta ne voidaan ylittää. (Fantina 2005, 139.)

Vaikka muutosten taustalla olisikin korkeamman johdon tuki, ovat työntekijät viime kädessä ne jotka ratkaisevat onnistuuko muutosprosessi. Tällöin on tärkeää tietää mitä

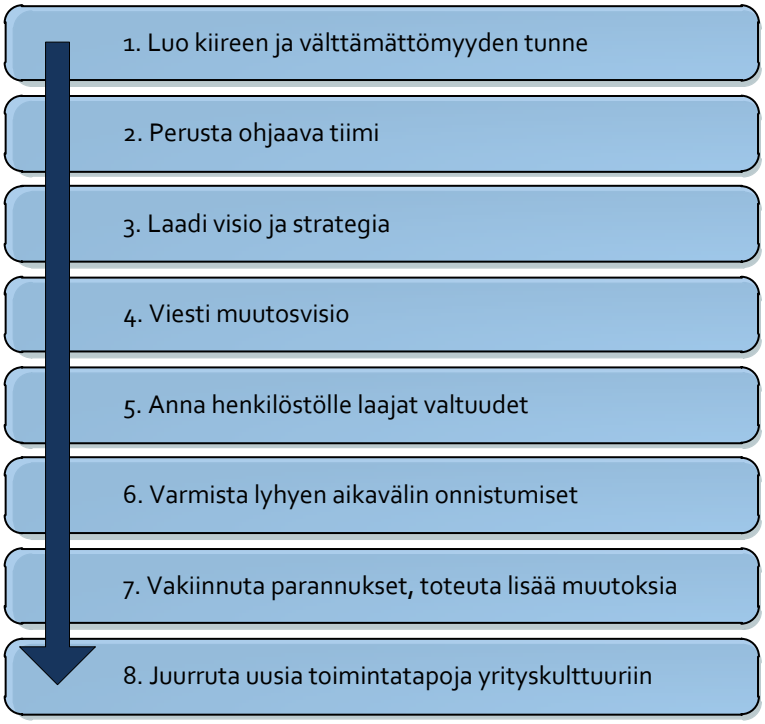
syitä voi olla muutosvastarinnan takana ja kuinka se voidaan hoitaa. Henkilökohtaisia syitä jotka voiva olla muutosvastarinnan (Fantina 2005, 140-146.) takana:

- Kiinnostuksen puute. Mikäli henkilökohtainen työtaakka on kohtuullisen suuri, voi työntekijältä puuttua halu muuttaa toimintatapoja siinä tilanteessa.
- Vaikutusvallan väheneminen. Muutos pienentää työntekijän vaikutusvaltaa tehden työntekijän vaikutusvallan alla olevia asioita helpommiksi tai jopa tarpeettomiksi. Tämä ei kuitenkaan vähennä erityisosaamisalueiden osaajien tarpeellisuutta.
- Tilivelvollisuus. Vaikka prosessin muutokset lisäävätkin seurantaa, se ei kuitenkaan tarkoita, että seurantatyökaluja käytettäisiin kenenkään syyllistämiseen. Virhetilanteista tulee oppia, sen sijaan että etsittäisiin syntipukkia.
- Itsevarmuuden puute. Työntekijä voi kokea, etteivät hänen kykynsä ja taitonsa ole riittäviä muuttuneessa prosessissa. Tällöin tiimin muiden jäsenten tuki sekä mentorointi tulevat tarpeeseen.

Muutoksen edessä on syytä tiedostaa, että edessä on haasteita. Sitoutuminen uusiin toimintatapoihin voi olla haastavaa ja onkin syytä huomioida, että työntekijän sitoutuminen voidaan saavuttaa vain hyväksynnän ja ymmärryksen kautta. Työntekijän on ymmärrettävä ja hyväksyttävä uuden toimintatavat, jotta hän voi sitoutua niihin. Uudet toimintatavat voivat vaatia myös uutta osaamista, jolloin myös koulutus tulee tarpeeseen. Sen lisäksi on huomioitava, että ihmiset suhtautuvat asioihin usein tunneperäisesti, jolloin muutoksen perustelu asia-argumentein ei edesauta sitoutumisen saavuttamiseen. Muutoksen käsittelyyn tarvitaan aikaa sulatella asiaa, sekä mahdollisuus vaikuttaa myös itse muutokseen. Erilaiset ihmisen myös suhtautuvat muutoksiin eri tavalla. Jotkut haluavat pitäytyä vanhassa, tutussa ja turvallisessa toimintatavassa, kun toiset näkevät mahdollisuuksia ja seikkailua uusissa toimintatavoissa. (Laamanen & Tinnilä 2009, 41-42.)

Muutosvastarintaa kohdatessa tarvitaan muutosjohtamista, jossa organisaation johtoasemassa olevat henkilöt ovat avainasemassa. Korkeammassa asemassa olevat henkilöt yleensä määrittävät sen mikä on tärkeää organisaatiossa ja organisaation prosesseissa.

Organisaation johdon tuki muutokselle on tärkeää myös sosiaalisesta näkökulmasta tarkasteltuna. Muutos voi usein tarkoittaa aikaisemmasta poikkeavaa asennetta tai ajattelutapaa. Mikäli valtaapitävät eivät muuta ajattelutapaansa muutoksen suhteen, eivät muutokset jää pysyviksi. Yksi suosituimmista muutosjohtamisen tueksi luoduista malleista on Kotterin malli, jonka vaiheet on esitetty kuviossa n. (Laamanen & Tinnilä 2009, 41.)



Kuvio 16. Kottlerin muutosjohtamisen vaiheet (Laamanen & Tinnilä 2009, 41)

Toimintaa uudistettaessa muutosvalmiutta voidaan parantaa ja kehittää luomalla puitteet osaamisen laajentamiselle. Samalla voidaan laajentaa ryhmän tietoisuutta muiden ryhmän jäsenten osaamisesta. Luontevin tapa tähän on toimia ryhmän yhteisten tehtävien kautta, jossa jokainen voi antaa oman panoksensa ryhmän hyväksi. Samalla ryhmän jäsenet voivat perehtyä muiden tehtäviin. (Arikoski & Sallinen 2011, 16; Toikko & Rantanen 2009, 100.)

6 Tutkimus ja työelämän kehittäminen

Kehittämisenä käsitetään usein sellaista systemaattista toimintaa joka prosessina etenee tiettyjen vaiheiden kautta tietyssä järjestyksessä alusta loppuun. Prosessin suorittamisen jälkeen voidaan onnistumista arvioida vertaamalla saavutettuja tuloksia asetettuihin tavoitteisiin. Kehittämistoiminnassa nähdään usein olevan taustalla selkeä ajatus, johon toiminnalla pyritään. Kehittämistoiminta on kuitenkin monimuotoista, jonka luonne vaihtelee kehitettävän kohteen, laajuuden ja muiden seikkojen mukaan. (Toikko & Rantanen 2009, 14.)

6.1 Kehittämisen näkökulmat

Toikon & Rantasen (2009, 14-16) mukaan kehittämistoiminta voidaan jäsennellä kuudella tavalla:

- Toimintatavan tai toimintarakenteen kehittäminen. Nimensä mukaisesti kehittämiskohteena ovat joko toimintatavat tai –rakenne. Kehittämisen laajuus voi vaihdella huomattavastikin esimerkiksi yhden työntekijän toiminnan kehittämisestä aina koko organisaation laajuiseen rakenteelliseen uudistukseen. Yksinkertaisimmillaan kehittämistoiminta voi olla esimerkiksi työprosessin mallinnusta.
- Yksikkökohtaiset uudistukset tai suuret reformit. Suuret reformit ja yksikkökohtaiset uudistukset voivat olla luonteeltaan hyvin samankaltaisia, mutta kehitystoiminnan laajuudessa on hyvin suuri ero. Suurissa reformeissa voi olla monipuolisemmistakin asioista kyse, mikäli muutokset suuntautuvat esimerkiksi kansainvälisten markkinoiden uudistamiseen.
- Ulkoinen tavoite tai itse määritelty tavoite. Huomattavan usein kehittämisen tavoitteet tulevat kehittämisen kohteena olevien toimijoiden näkökulmasta ulkopuolelta, esimerkiksi organisaation ylemmältä johdolta. Tällöin tavoite on usein valmiiksi määritelty. Mikäli kehittäminen tapahtuu kehityskohteen omasta aloitteesta ja omin ehdoin, ei tavoitteen etukäteen määrittely ole mahdollista. Tällöin kehittämisen tavoite muodostuu usein kehitysprosessin myötä.
- Hankeperustainen tai jatkuva kehittäminen. Kehittämisprojektit ovat usein enemmän hankkeita, kuin jatkuvia toistuvia prosesseja. Kehittämiselle voidaan

näin ollen asettaa tavoitteet hankkeen aikataulun mukaan sen sijaan, että seuranta olisi jatkuvaa.

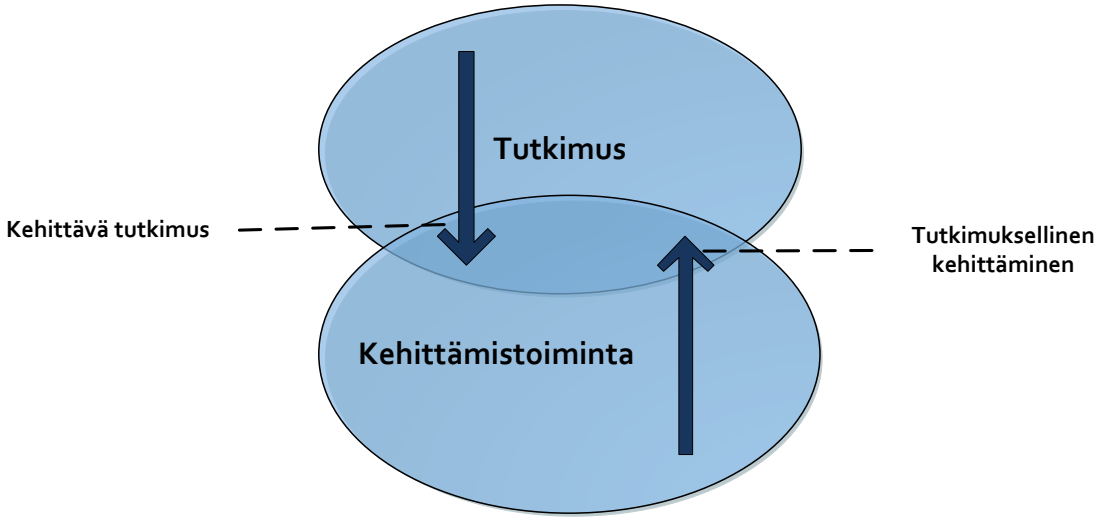
- Innovaatio tai diffuusio. Kehittämistoiminnassa voidaan keksiä uusia asioita, jolloin puhutaan innovaatioista. Toisaalta kehittämistoiminnalla ei välttämättä ole tavoitteena keksiä uusia asioita, vaan levittää hyväksi havaittuja ideoita, jolloin puhutaan diffuusiosta.
- Sisäinen kehitys- tai toimitusprojekti. Kehittämistoiminta voi kohdistua organisaation omaan toimintaan, jolloin kehitystyö on sisäistä. Kehitystoimintaa voidaan myös myydä ja markkinoida asiakkaille, jolloin kehittämistoimintaa voidaan toteuttaa toimitusprojekteina.

6.2 Tutkimus kehittämistoiminnassa

Kehittämistoiminnan ja sen menetelmien yhteydessä usein viitataan tutkimusmenetelmiin. Kehittämismenetelmät ja tutkimusmenetelmät nähdäänkin usein samana asiana, mutta eri näkökulmia asiaan on esitetty. Tutkimusmenetelmillä halutaan ratkaista tutkimusongelmia sekä esitettyjä tutkimuskysymyksiä. Tutkimusmenetelmät myös ovat tieteellisten periaatteiden mukaisen luotettavuusarvioinnin kohteena. Kehittämismenetelmissä lähtökohta on tieteellisen näkökulman sijasta käytännönläheisempi. Kehittämismenetelmissä pohditaan voidaanko menetelmällä saavuttaa haluttuja tuloksia sekä kuinka tulosten saavuttamista voidaan arvioida. Kehittämistoiminnan ja tutkimustoiminnan välisen yhteyden voidaan nähdä olevan siinä, että kehittämistoiminnassa sovelletaan tutkimustietoa. (Toikko & Rantanen 2009, 18-19.)

Perustutkimuksella pyritään tuottamaan uutta tietoa havaintojen kohteena olevista ilmiöistä. Perustutkimuksella ei välttämättä ole suoraan käytännöllistä tavoitetta, vaan päämääränä on ennen kaikkea uuden tiedon tuottaminen. Soveltavassa tutkimuksessa pyritään hyödyntämään uutta tieteellistä tietoa käytännön tasolla. Soveltavassa tutkimuksessa pyritään usein käyttämään perustutkimuksen tuottamaan tietoa. Vaikka kehitystyö ei olekaan samalla tavalla tieteelliseen perustaan sidoksissa kuin perustutkimus tai soveltava tutkimus, hyödynnetään kehitystyössä usein kyseisten tutkimusten saavutuksia. (Toikko & Rantanen 2009, 19-20.)

Tutkimuksen ja kehittämisen yhteistyötä voidaan lähestyä sekä tutkimuksen, että kehittämisen suunnasta. Kehittävässä tutkimuksessa lähtökohtana on tutkimustyön mukainen kysymyksen asettelu joista päästään kohti oikeaa kehittämistoimintaa. Lähtökohtana on siis tutkimus jota suunnataan kehitystyön tarpeisiin, jonka aikana myös tuotetaan uutta tutkimustietoa. Tutkimuksellisessa kehittämisessä käytännön ongelmat asettavat toiminnan reunaehdot. Toiminnassa pääpaino on kehittämisessä ja tietoa tuotetaan käytännön tilanteissa. Tutkimustietoa hyödynnetään tarpeen mukaan, mutta pääpaino on kehittämisessä. Tutkimuksellisella kehittämisellä pyritään aikaansaamaan konkreettisia muutoksia (Toikko & Rantanen 2009, 21-23).



Kuvio 17. Tutkimuksen ja kehittämistoiminnan risteyspaikka (Toikko & Rantanen 2009, 21)

Yksi suuntauksista jolla tutkimusta lähestytään kehittämisen suunnasta, on työelämän tutkimusavusteinen kehittäminen. Siinä toiminta on suunnattu käytännön toiminnan kehittämiseen tutkimustietoa hyödyntämällä. Vaikka toiminnalla pyritään käytännön läheiseen kehittämiseen, pyritään se tekemään tutkimusmielessä perustellusti. Tutkimusavusteisessa kehittämisessä hyödynnetään aikaisempaan tutkimus- ja kokemustietoon perustuvia käsitteellisiä malleja joiden perusteella asetetaan tutkimusongelmat ja hypoteesit. Tutkija testaa hypoteesejaan muutosinterventioiden avulla. Tällöin myös hypoteesit voivat joko muuttua tai tarkentua tutkimusprosessin aikana. Koko prosessin jälkeen arvioidaan kriittisesti saavutettuja johtopäätöksiä sekä mahdollisesti jalostetaan käytettyjä malleja. (Toikko & Rantanen 2009, 33.)

6.3 Toimintatutkimus

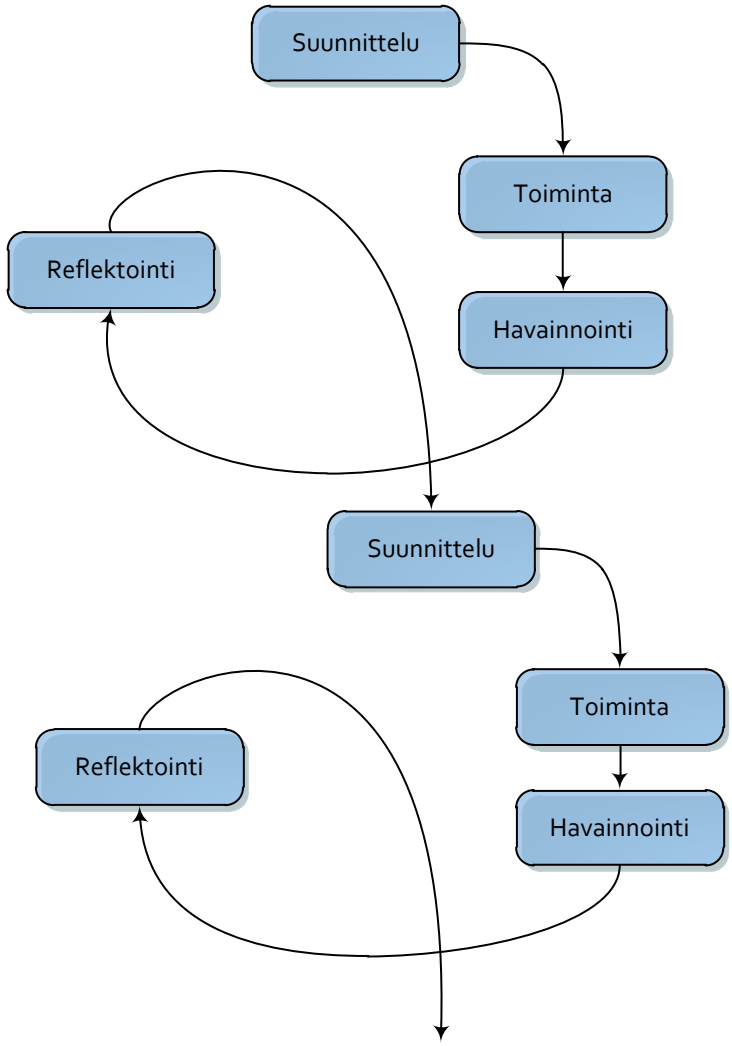
Tutkimuksella on perinteisesti tutkittu asioita ulkopuolisesta näkökulmasta ja pyritty tutkimaan asioita objektiivisesti tietyn välimatkan päästä. Toimintatutkimuksessa tämä asetelma ei päde, vaan tutkija on osa yhteisöä jonka toimintaa hän tutkii ja pyrkii kehittämään käytäntöjä ja menetelmiä muuttamalla. Tutkimuksessa tätä asetelmaan kutsutaan muutosinterventioksi. (Aaltola & Valli 2001, 179; Saaranen-Kauppinen & Puusniekka 2009, 40.) Toimintatutkimuksessa on olemassa myös suuntauksia, joissa tutkija ei ole osallisena tutkimuskohteen toiminnassa, vaan rooli on enemmän ulkopuolisen asiantuntijan kaltainen. Perinteinen suuntaus jossa tutkija on osana tutkittavaa yhteisöä, asettuu suuntauksissa puolivälin paikkeille. Toisessa ääripäässä on suuntaus jossa kaikki jäsenet ovat aktiivisia tutkijoita yhteisössä. (Toikko & Rantanen 2009, 30.)

Toimintatutkimuksen tavoitteena ei ole ainoastaan kuvata ja selittää toiminnassa esiintyviä ongelmia. Toimintatutkimuksen tavoitteena on muuttaa tutkimuksen kohteena olevaa toimintaa ja siinä esiintyviä ongelmia käytännössä. Toimintatutkimus tuottaa myös tietoa toiminnasta, kuten millä tavalla tutkimuksen kohteet voivat muuttua tai miksi ne eivät muuttuisi. (Saaranen-Kauppinen & Puusniekka 2009, 40.)

Toimintatutkimukselle tunnusomaista on toiminnan ja tutkimuksen samanaikaisuus ja tavoite saavuttaa välitöntä hyötyä toimintatutkimuksella. Näin ollen toimintatutkimuksen päämääränä ei ole ainoastaan tutkiminen vaan myös toiminnan samanaikainen kehittäminen. (Aaltola & Valli 2001, 170.) Toiminnan merkitys voidaan käsittää monella eri tavalla, mutta toimintatutkimuksen yhteydessä toiminnalla tarkoitetaan ensisijaisesti sosiaalista toimintaa. Tarkoituksena on siis ensisijaisesti kehittää ihmisten yhteistoimintaa. (Aaltola & Valli 2001, 171.)

Toimintatutkimuksen eräänä lähtökohtana on toiminnan reflektiivinen ajattelu, jonka avulla toimintaa pyritään ymmärtämään uudella tavalla. Tämän pyrkimyksenä on havainnoida nykyisiä toimintatapoja, niiden tarkoitusperiä, sekä miten niitä voitaisiin kehittää. Reflektio onkin hyvin keskeisessä asemassa toimintatutkimuksessa. Toimintatutkimusta hahmotellaan itsereflektiivisyyden kehänä jossa suunnittelu, toteutus, havainnointi ja reflektointi seuraavat toisiaan. Kun syklejä asetetaan peräkkäin, saavute-

taan toimintatutkimuksessa ajassa etenevä spiraali. (Aaltola & Valli 2001, 175-177.) Toimintatutkimus on luonteeltaan kokeilevaa ja sille onkin ominaista, että tutkimuksen eri vaiheet vuorottelevat useita kertoja tutkimuksen aikana. Tämä tarkoittaa usein sitä, ettei tutkimusprosessia pystytä välttämättä suunnittelemaan kovin hyvin etukäteen. Toimintatutkimuksessa käykin niin, että tutkimusprosessin aikana kerätyt tiedot tai havainnot ohjaavat tutkimusprosessia. (Toikko & Rantanen 2009, 30.)



Kuvio 18. Toimintatutkimuksen spiraalimalli (Toikko & Rantanen 2009, 67)

6.4 Kehitysprojektissa käytetyt menetelmät

Tässä kehitysprojektissa pyrittiin kehittämään käytäntöjä ja menetelmiä tutkimustietoa hyödyntäen. Käytäntöjä haluttiin kehittää oikeassa toimintaympäristössä sekä havaintojen, että teoriataustasta löydettyjen ideoiden pohjalta. Siten tutkimuksen ja kehittämis-

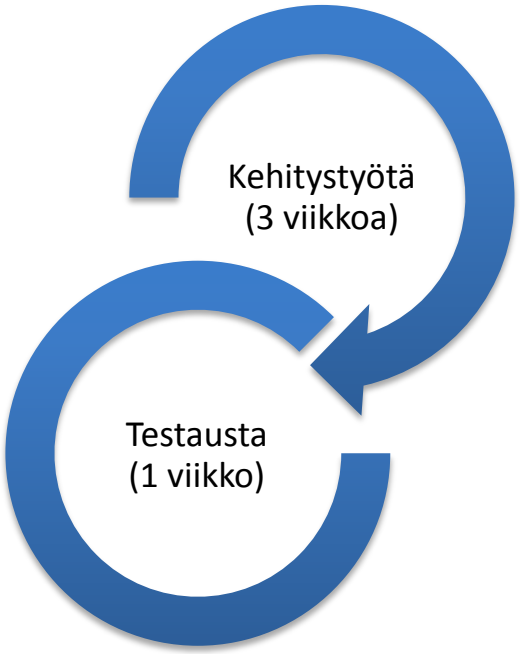
toiminnan lähestymissuunnaksi muodostui tämän projektin yhteydessä työelämän tutkimusavusteinen kehittäminen.

Toimintatutkimuksen teorian ja käytännön menetelmien muutoksen kautta tapahtuvan kehittämisen vuoksi tämä tutkimus toteutettiin toimintatutkimuksena. Muutokset menetelmiin vaikuttivat koko kehitysprojektin ajan sekä tutkijan työhön, tiimin muiden jäsenten työhön ja työmenetelmiin. Toimintatutkimuksen periaatteen mukaisesti sekä tutkija, että tiimin jäsenet olivat aktiivisesti mukana kehitystoiminnassa tutkimuksen ajan. Näiden seikkojen vuoksi, oli toimintatutkimus tutkimusmenetelmänä perusteltu.

Toimintatutkimuksen itsereflektiivisyyden kehistä muodostuva spiraali muodostivat tälle kehitysprojektille sopivan mallin. Ketterien kehitysmenetelmien ollessa luonteeltaan samankaltaisesti syklisiä toimintatutkimuksen kanssa, oli kehityssyklien päätyttyä hyvä hetki reflektoida mennyttä sykliä ja miettiä suuntaviivoja seuraavaa sykliä varten. Kunkin ohjelmistokehityssyklin loppuvaihe ennen seuraavaa kehityssykliä oli siis luonteva hetki käydä dialogia edellisen syklin aikana hyväksi tai huonoksi havaituista menetelmistä tai käytännöistä. Näiden pohjalta voitiin tarpeen tullen ehdottaa mahdollisia uusia suuntia seuraavaa kehityssykliä varten. Vastavuoroisesti tiimin muut jäsenet pääsivät antamaan palautetta ja ideoita tutkijalle kehitysprojektissa seuranneita kehityssyklejä varten.

7 Kehitysprojektin lähtötilanne

Tuotekehitystiimissä työskentelevät kaikki henkilöt tekevät sekä kehitystöitä, että testausta. Kehitysprojektin alkuvaiheessa testaus oli organisoitu siten, että jokainen kehittäjä oli vuorollaan viikon kerrallaan testausvastuussa. Tällöin kehittäjän päävastuuna oli testata ja verifioida valmiiksi merkittyjä tehtäviä. Tiimin muut jäsenet keskittyivät tällöin varsinaiseen kehitystyöhön. (Frends Technology 2011 a.) Tiimin koostuessa neljästä henkilöstä muodostui kehitys-testaussyklistä siten, että kehitystyötä kukin kehittäjä teki 3 viikkoa yhtäjaksoisesti jota seurasi viikon testausjakso, kuten kuviossa n esitetään.



Kuvio 19. Tuotekehitystiimin tuotekehitys- ja testaussykli

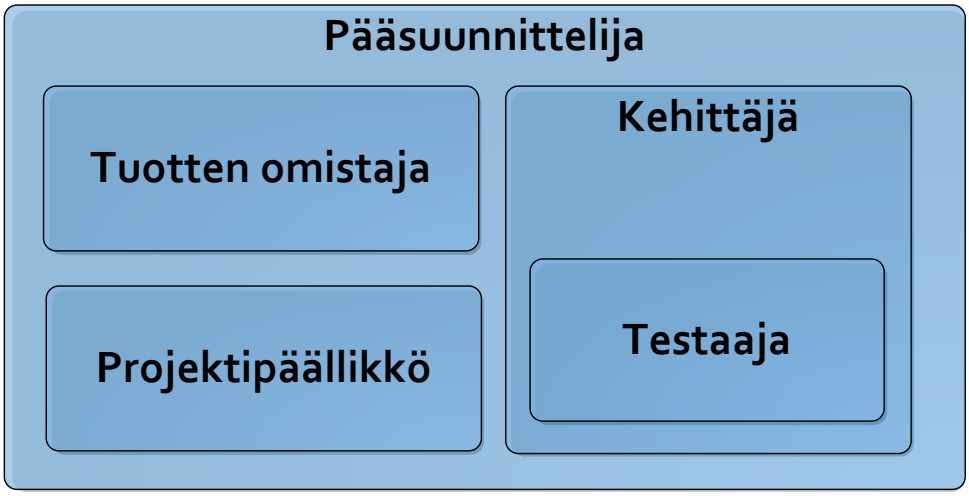
7.1 Tuotekehitystiimin roolit

Tuotekehitystiimin roolit olivat kehitysprojektin alkuvaiheessa hyvin scrum-tyyliset. Kehitysprojektin alkuvaiheen roolit tuotekehitystiimissä on esitetty taulukossa 4.

Taulukko 4. Tuotekehitystiimin kehitysprojektin alkutilanteen roolit (Frends Technology 2011 b)

Rooli	Kuvaus
Projektipäällikkö (Project Manager)	Projektipäällikön rooli tuotekehitystiimissä muistuttaa scrummasterin roolia scrum-kehitystyössä: Projektipäällikön tehtävänä on varmistaa, että päivittäiset tehtävät hoituvat ja tulevat tehdyksi, eikä tiimillä ole ongelmia jotka haittaisivat tai estäisivät töiden tekemisen. Projektipäällikön tehtävänä on myös hallinnoida tehtävälistoja valkotaululla ja JIRA:ssa sekä ylläpitää julkisia dokumentteja sekä tilastoja tehdyistä tehtävistä. Tuotekehitystiimissä projektipäällikön rooli on sulautettu pääsuunnittelijan rooliin.
Tuotteen omistaja (Product Owner)	Henkilön vastuulla on asiakastarpeiden kartoitus ja määritysten tekeminen niiden perusteella. Tehtävien kasautuessa tuotteen omistajan vastuulla on myös tehtävien priorisointi. Tuotekehitystiimin ylläpitämien tuotteiden osalta tuotteen omistajan rooli on sulautettu pääsuunnittelijan rooliin, mutta teknologiajohtaja ohjaa prosessia.
Pääsuunnittelija (Technical Lead, Chief Designer)	Ohjaa suunnittelu- ja toteutustyötä tuotekehitystiimissä.
Kehittäjä (Developer)	Toteuttaa tuotekehitystehtäviä sekä kirjoittaa yksikkö- ja integraatiotestejä ohjelmakoodille. Ottaa osaa myös suunnitteluprosessiin.
Testaaja (Tester)	Suorittaa hyväksymistestaukset tehtäville sekä toisinaan myös integraatiotestejä. Ylläpitää myös testiympäristöjä sekä ottaa osaa suunnitteluprosessiin.

Kuviossa 20 on esitetty tuotekehitystiimin roolit ja niiden väliset suhteet. Kuvassa ulompana oleva rooli pitää sisällään myös sisempänä olevan roolit vastuut ja velvoitteet. Pääsuunnittelijan rooli siis pitää sisällään käytännössä kaikki tuotekehitystiimin roolit. Kehittäjän rooli pitää sisällään oman roolinsa lisäksi testaajan roolin.



Kuvio 20. Tuotekehtitystiimin roolit ja suhteet

7.2 Projektitilastot

Tuotekehitystiimin kehitystoiminnassa ylläpidetään viikkotasolla projektitilastoja. Projektin alkuvaiheessa projektien tilastot eivät olleet täysin yhdenmukaisia kaikkien projektien osalta. Joidenkin projektin osalta tehdyistä töistä ylläpidettiin tilastoja tehtäväpisteiden muodossa. Joidenkin tuotekehitystiimin toteuttamien projektien tilastoja on kerätty vain valmistuneiden, eli suljettujen tehtävien lukumäärän muodossa. Osassa projekteja ei siis kerätty tehtäväpisteitä ollenkaan. Näiden projektien osalta ei siis ole arvioitu tehtävien kokoluokkaa tai työmäärää suhteessa muihin, vaan kaikki tehtävät olivat samanarvoisia.

Projektitilastot on koostettu yhteisesti kaikista projekteista, eikä projektikohtaisia tilastoja ole kerätty. Käytännössä tuotekehitystiimissä kuitenkin pääsääntöisesti työstetään yhtä projektia kerrallaan, joten useimmiten viikkotason tilastot koskevat vain yhtä projektia. Viikkotason tilastoissa saattaa kuitenkin olla mukana tehtäviä useammasta projektista. Viikkotason tilastoissa on valmiiden tehtävien ja tehtäväpisteiden lisäksi seurattu löydettyjen virheiden määrää sekä korjattujen virheiden määrää.

7.3 Tuotekehitysprosessin alkutilanne

Tuotekehitysprosessi on ottanut vaikutteita scrumista, Leanista sekä Kanban:ista. Scrumista perusperiaatteista poiketen, tuotekehitystiimillä ei ole tarkasti määriteltyjä iteraatiojaksoja. (Frends Technology 2011 c.) Käytännön tasolla tuotekehitystiimissä pidetään joka maanantai viikkopalaveri, jossa käydään läpi edellisen viikon tulokset sekä päätetään alkavan viikon päämäärä, eli mitä on tavoitteena saada tehtyä viikon loppuun mennessä (Frends Technology 2011 d). Viikon aikana pidetään päivittäinen palaveri, jossa käydään läpi mitä on tehty päivän aikana, mitä tullaan seuraavaksi tekemään, sekä on tullut ongelmia vastaan töitä tehdessä. Tarvittaessa ongelmat käsitellään ryhmässä ja käsittelyn jälkeen voidaan päättää kuinka jatketaan ongelman kanssa. Mikäli ongelmaa ei saada ratkaistua, voidaan tarvittaessa pohtia tehtävän suunnittelua uudestaan ja tehdä lopulta toteutus toisella tavalla. (Frends Technology 2011 b.) Tuotekehitysprosessi on ottanut vaikutteita myös XP:stä, sillä protoilua käytetään tiimissä usein keinona selvittää

ominaisuuksien toteutusmahdollisuuksia. Myös XP:n menetelmistöön kuuluvaa parioh-jelmointia käytetään silloin tällöin tiimin työskentelyssä, mutta hyvin satunnaisesti.

Käytännön kehitystyön tehtävät puretaan pienemmiksi kokonaisuuksiksi. Pyrkimyksenä on, ettei mikään yksittäinen tehtävänä ole enimmilläänkään viittä työpäivää enempää. Tavoitteena on saada pidettyä tehtävät 1-2 päivän mittaisina kokonaisuuksina ja keskit-tyä yhteen kokonaisuuteen kerrallaan. Tarpeen niin vaatiessa tehtäviä voidaan myös uudelleen priorisoida ja asettaa kehitettävänä olevaan ominaisuuskokonaisuuteen kuu-lumattomia tehtäviä ensisijaisiksi töiksi. (Frends Technology 2011 c.)

Tehtäviä on erityyppisiä, joita kuvataan Kanban-valkotaululla erivärisin lapuin. Tehtävi-en tyypit on kuvattu taulukossa 5.

Taulukko 5. Tuotekehitystiimin tehtävätyypit

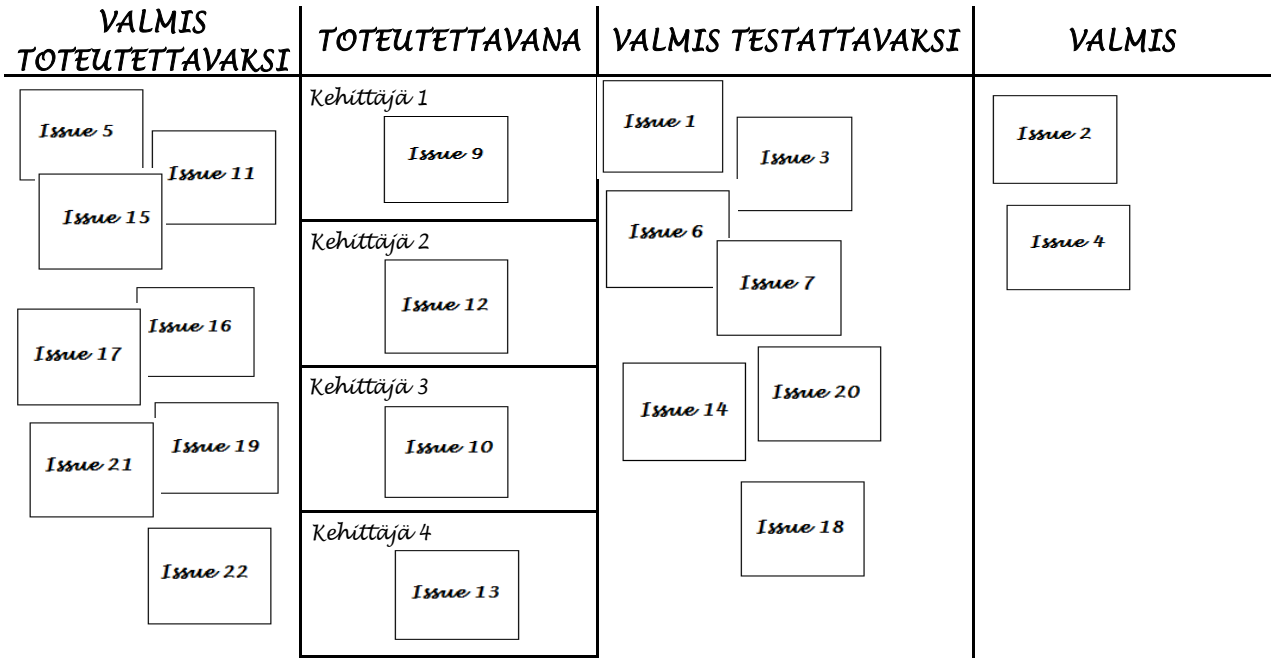
Tyyppi	**Selite**
Ominaisuus (Feature)	Uusi ominaisuus.
Parannus (Improvement)	Olemassa olevan toiminnallisuuden parantaminen. Parantaminen voi olla esimerkiksi toiminnallisuuden muuttamista käyttäjäystävällisesti tai teknisesti toimivammaksi.
Virhe (Bug)	Virhe/puute/muut vika, eli ohjelman virheellinen toiminta.
Tehtävä (Task)	Tehtävä joka voi olla joko dokumentointia, testiympäristöjen konfigurointia, lähdekoodin refaktorointia tai jotain muuta.

Tehtävien tilaa seurataan Kanban-valkotaululle asetetuilla lapuilla, joista kukin lappu edustaa yhtä tehtävää. Tehtävillä on eri tiloja riippuen siitä, missä vaiheessa prosessia ne ovat. Tehtävien tilat valkotaululla, sekä tilojen kuvaukset on esitetty taulukossa 6.

Taulukko 6. Tuotekehitystiimin Kanban-taulun tilat. (Frends Technology 2011 c)

Tila	Selite
Suunniteltavana (In Design)	Tehtävä on tunnistettu, mutta vaatii vielä sekä suunnittelua, että myös hyväksymistestien kriteereiden määrittelyä.
Valmis toteutettavaksi (Ready to Implement)	Tehtävä on tunnistettu ja sille voi olla tehtynä tarkemmat määrittelytkin. Useimmiten määrittelyt on esitetty hyväksymistestien muodossa. Tehtävä on näin ollen valmis toteutettavaksi. Tehtävä voidaan määritellä vielä tarkemmin ennen toteutuksen aloittamista.
Toteutettavana (Implementing)	Tehtävä on toteutettavana. Toteutus pitää sisällään myös yksikkö- ja integraatiotestien tekemisen toteutettaville komponenteille. Myös dokumentointi kuuluu työvaiheen toimenkuvaan, mikäli tehtävän katsotaan tarvitsevan erillistä dokumentointia.
Valmis testattavaksi (Ready for Testing)	Tehtävä on hyväksytysti katselmoitu ja on valmis testattavaksi ja verifioitavaksi.
Valmis (Done)	Tehtävä on toteutettu, testattu ja dokumentoitu.

Tuotekehitystiimin Kanban-taulu oli kehitysprojektin alkuvaiheessa kuvion 21 mukainen.



Kuvio 21. Tuotekehitystiimin Kanban-taulu kehitysprojektin alussa

Tehtävien tunnistamisesta ja edistymisestä pidetään kirjaa myös JIRAssa (Frends Technology 2011 c.) JIRAssa tehtävien tilat poikkeavat hieman valkotaulun tiloista, mutta sisällöllisesti samat tilat ovat löydettävissä myös JIRAssa ylläpidettävien tehtävien tilois-

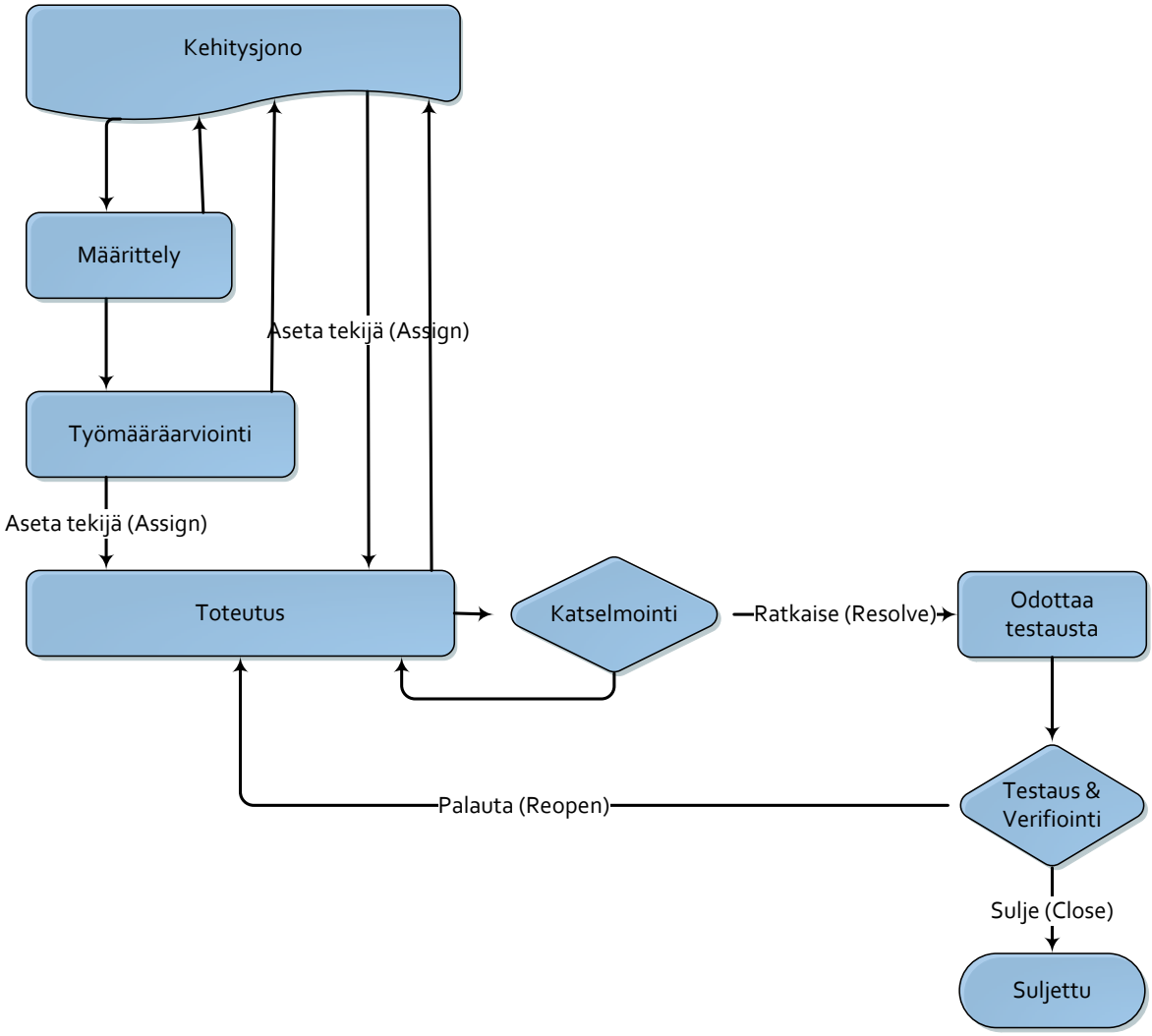
ta. Kanban-taulun tehtävien ja JIRAn tehtävien tilojen vastaavuuden on esitetty taulukossa 7.

Taulukko 7. Tuotekehitystiimin Kanban-taulun tilojen ja JIRA-tilojen vastaavuus

Tila Kanban-taululla	Tila JIRA:ssa	Selite
Toteutettavana (Implementing)	**Assigned**	Tehtävälle on määritetty JIRA:ssa kehittäjä joka toteuttaa tehtävän.
Valmis testattavaksi (Ready for Testing)	**Resolved**	Valmis testattavaksi ja verifioitavaksi.
paluu **Toteutettavana**-tilaan	**Reopened**	Tehtävässä löydettiin virheellistä toimintaa (tehtävä ei joko täytä kaikkia vaatimuksia tai toimii muuten virheellisesti) testausvaiheessa ja vaatii korjauksen ennen verifiointia.
Valmis (Done)	**Closed**	Testattu ja verifioitu.

Kun tehtävän on kirjattu JIRAan, tehtävän tila on Open (Frends Technology 2011 e). Tällöin tehtävien voidaan nähdä olevan valkotaululla valmiina toteutettavaksi, josta tehtävä lähtee etenemään prosessissa.

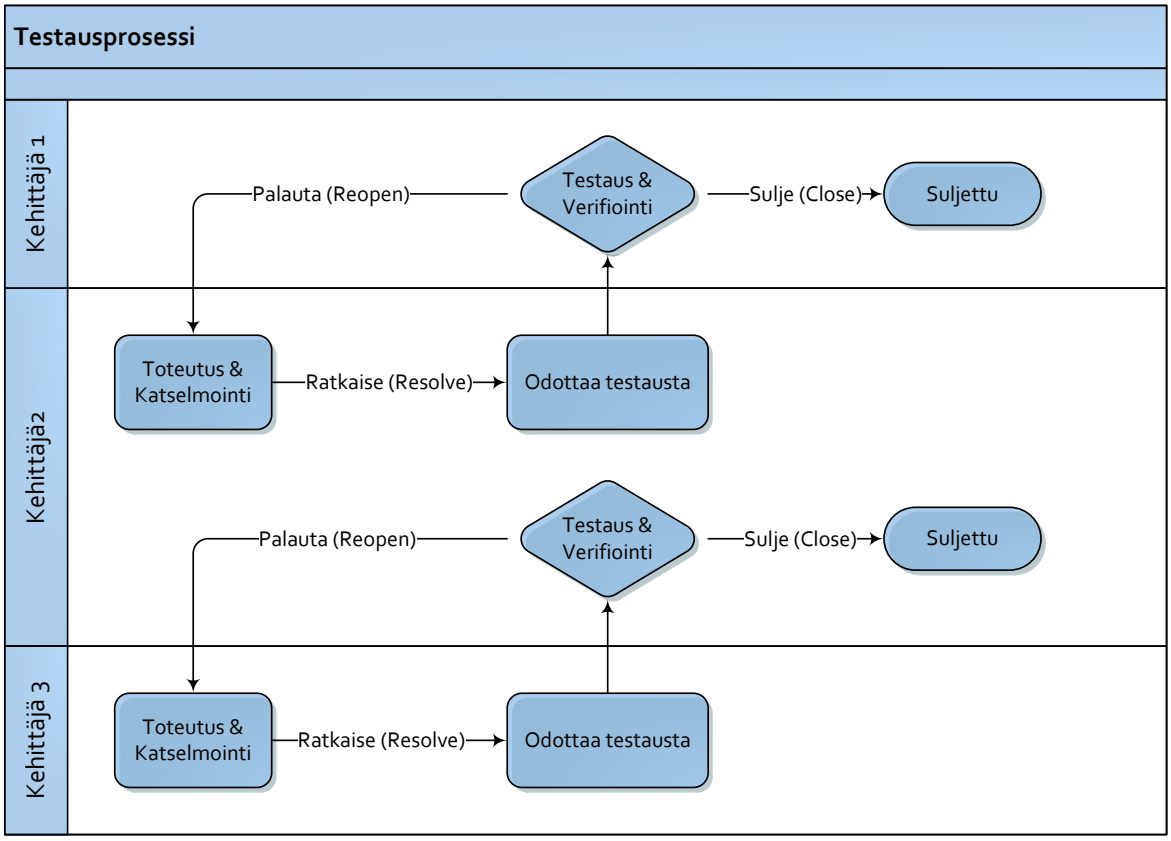
Tehtävien toteutuksen alkutilanne vaihtelee hieman tilanteen mukaan. Ennen toteutusta voidaan tehtävälle tehdä määrittelyitä, mikäli se on tarpeen. Määrittelyitä tehdään, mikäli kyseessä on uusi toiminnallisuus eikä selkeitä määrityksiä ole vielä aikaisemmin tehty. Mikäli kyseessä sen sijaan on esimerkiksi virheen korjaus, on virheellisen toiminnan kuvaus jo riittävä määrittely, kuinka toiminta saadaan korjattua. Toisaalta myös alkuvaiheessa selkeältä näyttänyt tehtävä voi toteutusvaiheessa osoittautua arvioitua monimutkaisemmaksi ja tällöin voidaan tehdä toteutuksen yhteydessä lisämäärittelyitä. Lisämäärittelyiden yhteydessä tehtävä voidaan myös hajauttaa pienemmiksi tehtäviksi, mikäli tehtävä osoittautuu aikaisemmin arvioitua suuremmaksi. Kun tehtävä on toteutettu, merkitään se tehdyksi JIRAssa ja Kanban-taululla tehtävää merkitsevä lappu siirretään testausta odottavien tehtävien sarakkeeseen. Kuviossa 22 on esitetty kuinka tehtävät etenivät tuotekehitystiimin prosessissa kehitysprojektin alkuvaiheessa. Nuolten tekstit kuviossa 22 kuvaavat JIRAn toimintoja joilla tehtävät siirtyvät tilasta toiseen.



Kuvio 22. Tuotekehitysprosessin alkutilanne

7.3.1 Testausprosessi

Vaikka testaus on osa tuotekehitysprosessia, kuvataan testauksen osuus tässä vielä erikseen. Kehitysprojektin alkuvaiheessa vastuu testauksesta oli kullakin viikolla sillä kehittäjällä jonka testausviikko on meneillään (Frends Technology 2011 a). Välillä kuitenkin tulee eteen tilanteita jolloin kaikki kehittäjät tekevät kehitystöitä eikä testausvastuussa oleva ehdi testata kaikkia tehtäviä. Tällöin jää muiden kehittäjien tehtäväksi testata valmiit tehtävät. Perusperiaatteena joka tapauksessa on, että kehittäjä itse ei voi testata ja verifioida omaa tehtäväänsä, vaan jonkun toisen kehittäjän on tehtävä se. Testaus ja muut laadunvarmistuksen toimenpiteet on hoidettu tuotekehitystiimissä sisäisesti.



Kuvio 23. Testausprosessi

7.4 Prosessin ongelmakohdat

Kehitysprojektin alkutilanteessa tiimissä vastuut olivat kertyneet liikaa yhdelle henkilölle. Käytännössä projektipäällikön ja pääsuunnittelijan rooleja ja niiden mukanaan tuomia vastuista vastaa tuotekehitystiimissä yksi henkilö: pääsuunnittelija. Useimpien tuotteiden tapauksessa pääsuunnittelijalla on myös tuotteen omistajan rooli. Näiden lisäksi pääsuunnittelijan rooli pitää sisällään myös kehittäjän ja sen mukaisesti myös testaajan roolin. Kehittäjien vastuualue on ymmärrettävästi pienempi. Myös tuotteiden demojen esittely on kuulunut pääsuunnittelijan vastuulle. Käytännössä tämä on ajoittain johtanut tilanteisiin joissa demo on jäänyt pitämättä pääsuunnittelijan ollessa esimerkiksi lomalla. Knibergin (2007, 65.) mukaan demon esitteleminen sprintin päätteeksi on kuitenkin esiarvoisen tärkeää jo senkin vuoksi, että demo-tilaisuuksissa tiimi voi saada palautetta työstään.

Testauksen osalta ongelmana on ollut motivaation puute testausta kohtaan. Tuotekehittäjät ovat huomattavasti motivoituneempia tekemään varsinaisia kehitystöitä, kuin testausta. Tämän vuoksi testausvastuu on ollut tiimissä viikoittain kiertävä.

Ongelmana alkutilanteessa nähtiin myös prosessin paikalleen pysähtyneisyys. Toisaalta stabiili prosessi on helppo tiimin jäsenille. Toisaalta se myös tarkoittaa, ettei prosessi ole muuttunut pitkään aikaan, eikä näin ollen myöskään kehittynyt. Täysin varmasti ei myös voida sanoa, että nykyistä prosessia sellaisenaan kuin se on kuvattu, edes noudatettaisiin. Osasyy prosessin pysähtyneeseen tilanteeseen saattaa juontaa juurensa keskittyneistä vastuista.

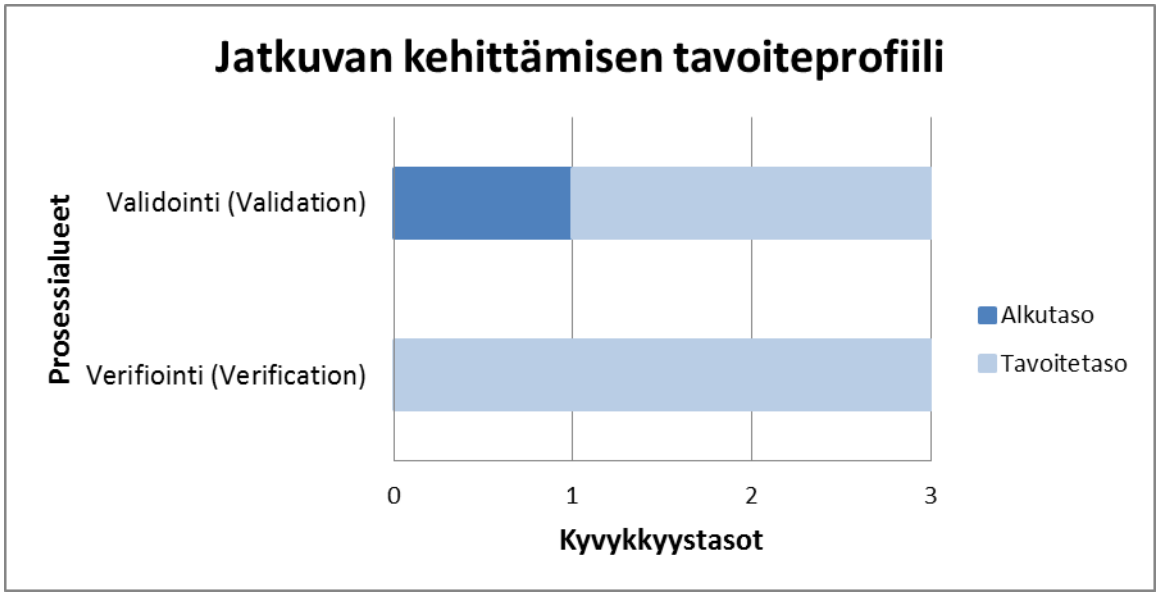
7.5 Prosessin alkutilanteen arviointi

Kehitysprojektissa ei suoritettu varsinaista virallista SCAMPI-auditointia CMMI-DEV:in osalta. Virallista auditointia ei pidetty tarpeellisena, koska organisaatiossa ei ole kokonaisuudessaan tarkoituksena ottaa CMMI-DEV:iä käyttöön. Tämän vuoksi myöskään kehitystiimissä ei hankittu auditointikoulutusta CMMI-DEV:iä varten. Arviointi toteutettiin itsearviointina. SCAMPI-arvioinnissa arviointitulos asettuisi luultavasti B- tai C-tasolle.

Koska tässä opinnäytetyönä tehdyssä kehitysprojektissa keskityttiin laadunvarmistukseen, valikoituivat kehitettäviksi prosessialueiksi validointi (validation) ja verifiointi (verification). Myös muista prosessialueista on löydettävissä keinoja laadunvarmistuksen kehittämiseen, mutta tämän projektin puitteissa valitut kaksi prosessialuetta muodostuivat tärkeimmiksi.

Validoinnin ja verifioinnin prosessialueet liittyvät toisiinsa siten, että verifiointi vastaa enemmän kysymykseen: Tehtiinkö asiat oikein? Validointi puolestaan vastaa kysymykseen: Tehtiinkö oikeita asioita? (Software Engineering Institute 2010, 393.)

Kehitettäväksi valittujen prosessialueiden valinnan jälkeen suoritettiin prosessialueille arviointi lähtötason määrittelemiseksi. Validoinnin taso osoittautui olevan jo aikaisemmin tuntemattomasta CMMI-DEV-viitekehyksestä huolimatta tasoa 1. Verifioinnin taso oli 0. Prosessien alku- ja lopputasoihin liittyvät kuvaukset löytyvät prosessien lopputilanteen arvioinnista luvusta 9.4.4.



Kuvio 24. Valittujen prosessialueiden alku- ja tavoitekyvykkyystasot

8 Muutosten toteutus

Kehitysprojektin ensimmäisessä vaiheessa päätettiin kartoittaa ja kuvata sen hetkinen tuotekehitysprosessi ja etsiä prosessin ongelmakohdat. Kun prosessi olisi kuvattu ongelmakohtineen, olisi helpompi osoittaa prosessien heikot kohdat seuraavia tuotekehityssyklejä ajatellen. Kehitysprojektin alkutilanne on kuvattu luvussa 7.

Prosessin muutosehdotusten käsittely päätettiin tehdä aina tuotekehitystiimin kuukausittaisen retrospektiivin aikana. Tähän päädyttiin sen vuoksi, että silloin kaikilla jäsenillä oli mahdollisuus keskustella avoimesti mahdollisista tulevista muutoksista. Pienen tiimin ollessa kyseessä, oli myös mahdollista kuunnella kaikkia asianosaisia. Tällä pyrittiin helpottamaan muutosten aikaansaamista ja saamaan tiimin jäsenet osallisiksi muutokseen muutosjohtamisen yleisten hyvien periaatteiden mukaisesti.

Prosessin nykytilanteen kartoitus dokumentaation ja käytännön perusteella osoitti dokumentaation jääneen monilta osin käytännön toteutuksesta jälkeen. Perusperiaatteiltaan prosessi toimi kuten dokumentaatio antoi ymmärtää, mutta poikkeuksia sekä muutoksia dokumentaatioon verrattuna oli havaittavissa.

8.1 Tuotekehitystiimin uudet roolit

Projektin alkuvaiheessa tuotekehitystiimissä määritellyt roolit olivat prosessissa lähtökohtaisesti hyvin scrumimaiset. Toiminta itsessään oli Kanbanista ja Leanista vaikutteita ottavaa. Rooleissa oli ongelmana se, että niiden ja varsinkin niihin liittyvien vastuiden huomattiin kertyneen hyvin paljon vain yhdelle henkilölle. Sen vuoksi scrumimaisia rooleja muokattiin paremmin sopivaksi nykyiseen Kanban-tyyliseen prosessiin. Tällä haluttiin myös jakaa vastuita ja siten tasata työtaakkaa. Muutoksilla haluttiin saada aikaan se, ettei yksi rooli pitäisi välttämättä enää sisällään niin paljon vastuita kuin ennen. Näiden asioiden seurauksena tuotekehitystiimissä otettiin kehitysprojektin aikana käyttöön uusia rooleja.

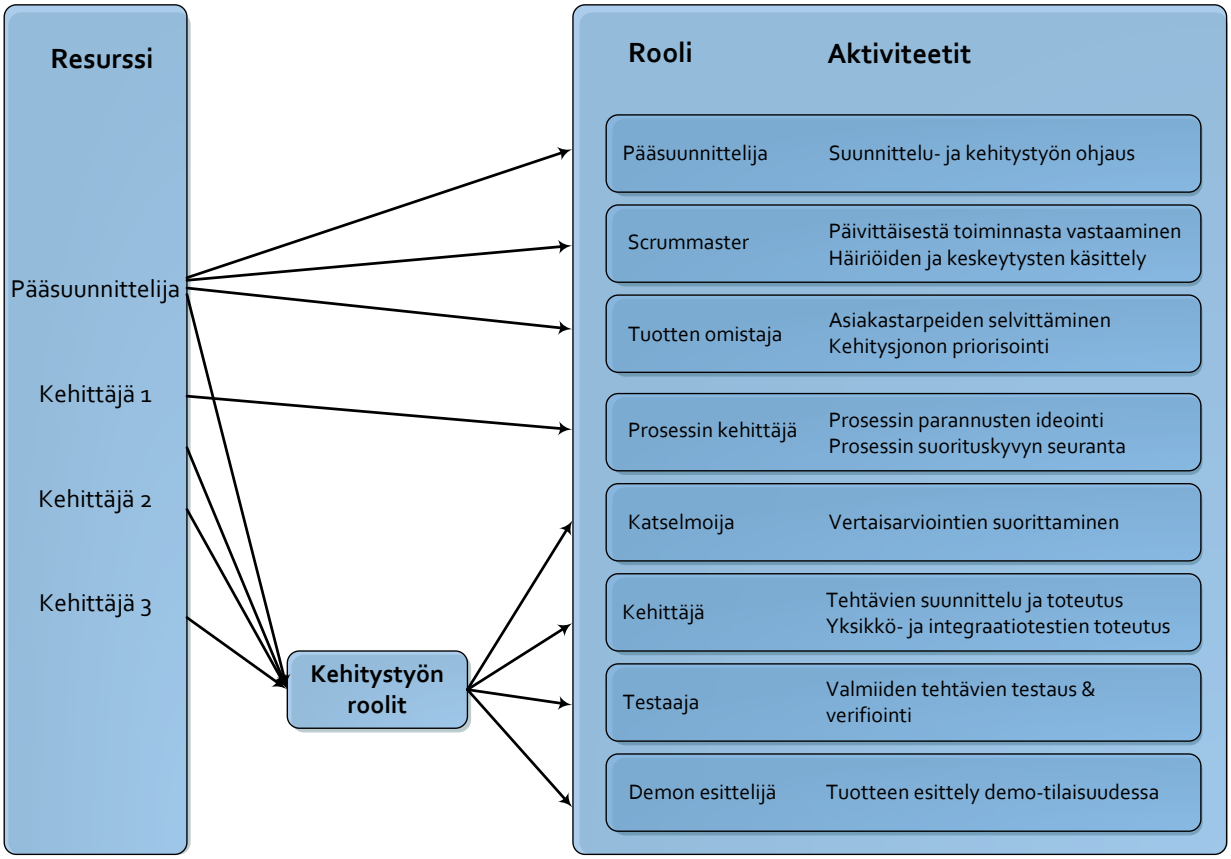
Roolien kuvauksessa käytettiin RUP:in mukaista tapaa esitellä toimenkuvia ja niihin liittyviä aktiviteetteja. Kuvaustavan avulla pyritään esittämään vastuualueet hieman hie-

nojakoisemmalla tavalla, kuin aikaisemmin. Työyhteisössä ei kuitenkaan käytetä RUP:in terminologian mukaista Worker-termiä vaan roolia (role).

Pääsuunnittelijan vastuisiin aikaisemmin kuuluneet tuotekehitystiimin retrospektiivien pitämiset päätettiin siirtää kehitysprojektin yhteydessä opinnäytetyönä tehdyn kehitysprojektin projektipäällikölle. Tämä oli luonteva vaihtoehto siinä mielessä, että projektipäällikkö oli vastuussa prosessin kehittämisestä projektin ajan. Retrospektiivit ovat nimenomaan prosessin jatkuvaa kehittämistä silmällä pitäen pidettyjä tilaisuuksia, joten se puolsi päätöstä täysin. Lisäksi projektipäällikön vastuulle annettiin mittareiden seuraaminen ja mittausdatan kerääminen kaavioita varten. Merkittävänä muutoksena retrospektiiveihin liittyen päätettiin myös seurata, mitä retrospektiiveissä oli päätetty ja kuinka päätöksissä oli pystytty pitäytymään. Tämä vastuu asettui luonnollisesti retrospektiivin vetäjälle. Tämän seurauksena luotiin tiimiin projektin aikana ensimmäisenä uutena roolina prosessin kehittäjä (Process Developer)-rooli. Rooli on käytännössä pääsuunnittelijan roolista eriytetty osa, jossa osa pääsuunnittelijalle kuuluvista vastuista on jaettu omalle uudelle roolilleen. Roolin vastuisiin kuuluu tuotekehitysprosessin kehitys- ja parannusehdotusten ideoiminen. Lisäksi roolin vastuisiin kuuluu tiimin retrospektiivien isännöiminen sekä myös prosessin suorituskyvyn seuranta prosessin seurantaan käytettävien mittareiden avulla.

Prosessissa yksi pääsuunnittelijalle suurelta osin keskittynyt vastuu oli vertaisarviointien suorittaminen. Projektin edetessä vastuuta vertaisarviointien suorittamisesta haluttiin jakaa koko tiimille. Tämän seurauksena tiimiin tuli uusi rooli: Katselmoija (Reviewer).

Uutena roolina tuotiin lisäksi vielä demon esittelijä (Demo Presenter). Roolin ajatuksena on se, että sprintin päätteeksi demon pitää rooliin asetettu henkilö. Erona aikaisempaa tässä on se, että demon pitäminen kuului pääsuunnittelijan vastuulle, eli pääsuunnittelija esitteli demon joka kerta. Demo Presenter on myös yksi vastuunjakorooli uudessa prosessissa. Ajatuksena on se, että kenen tahansa tiimin jäsenistä olisi kyettävä esittelemään demo jollakin tasolla joko kokonaan tai osittain demon yhteydessä.



Kuvio 25. Tuotekehitystiimin uudet roolit

8.2 Mittarit

Mittaamista voidaan tehdä monenlaisilla apuvälineillä. Perinteisesti mittaamiseksi mielletään pituuden tai painon mittaus. Tärkeää on tällöin määritellä mitä asiaa tai suuretta mittarilla voidaan mitata. Käyttötarkoituksesta usein riippuu mitä mittaria voidaan käyttää. Esimerkiksi painon mittaamiseen ei voida käyttää samaa mittaria, kuin pituuden mittaamisen ja päinvastoin. Samanlainen alkuasetelma koskee myös tieteellisiä mittareita. Myös tieteellisille mittareille tulee määritellä mitä asiaa ne mittaavat, eli mittarille on määriteltävä konkreettinen ilmiö jota mitataan. Mittauksessa voidaan käyttää joko valmiita mittareita tai tarpeen vaatiessa ne voidaan kehittää myös itse. (Yhteiskuntatieteellinen tietoarkisto 2011).

Mittareiden valinnassa tai kehittämisessä tulee kiinnittää huomiota myös mittarin luotettavuuteen. Mittarin tulee olla sellainen, että se mittaa aina samaa asiaa mittausajankohdasta riippumatta. Mittaustuloksiin eivät saa vaikuttaa satunnaisvirheet tai olosuhteet kuten esimerkiksi mielialan vaihtelut. Mittarin luotettavuus on tärkeässä asemassa

myös silloin jos kerätystä mittausaineistosta halutaan tehdä esimerkiksi yhteenvetoja. Tällöin kaiken mittaristolla kootun aineiston tulee olla samojen kriteeristöjen mukaisesti tuotettua jotta yhteenveto on mahdollista tehdä. (Yhteiskuntatieteellinen tietoarkisto 2011).

Laadullisen mittarin voidaan määritellä tuottavan määrällistä tietoa siitä, mikä on jonkin toiminnon laadullisen ominaisuuden aste. Ohjelmistojen ollessa kyseessä tämän määritelmän voitaisiin todeta olevan myös prosessi, joka saa syöttötietona tietoa ohjelmistosta ja antaa tuloksena tiedon siitä, mikä on ohjelmiston laadullinen aste tietyn laadullisen ominaisuuden osalta. (Galin 2004, 413.)

Usein ohjelmistokehityksen yhteydessä on käytetty laadullisena mittarina virheiden lukumäärä per 1000 riviä koodia (KLOC). KLOC-mittarin käyttö ei ole kovin suotavaa sillä mittarin heikkouksia ovat mm. se, että mittarin antamiin tuloksiin voivat vaikuttaa laadun kannalta epäolennaiset seikat, kuten ohjelmoijan tyyli kirjoittaa koodia tai kommenttirivien määrä lähdekoodissa. (Galin 2004, 436.)

Laadullisiin indikaattoreihin perustuvat mittarit voivat hyödyntää jo olemassa olevaa tietoa, sekä olla riippumattomia käytettävistä teknologioista. Näitä indikaattoreita ovat (Schulmeyer 2008, 395-396.) esimerkiksi:

- Edistyminen. Kuvaa kehityssyklin aikana tehdyn työn määrä, kuten kuinka paljon ominaisuuksia on saatu toteutettua ja kuinka paljon testejä on tehty.
- Testien kattavuus. Määrittää kuinka hyvin tehdyt ohjelmistokomponentit on testattu. Kuvaa sitä määrää kuinka monta ohjelmistohaaraa käydään läpi automaattisessa testauksessa.
- Virheiden havaitsemistehokkuus. Ilmaisee kuinka suuri osa virheistä on löydetty kussakin kehitysvaiheessa.
- Virheiden korjaustehokkuus. Kuinka suuri osa virheistä on saatu korjattua ajan myötä eri vaiheissa kehitystyötä.
- Monimutkaisuus. Kuvaa lähdekoodin suorituspolkujen lukumäärää ja siten monimutkaisuutta.

Bundschuh, Ebert, Dumke & Schmietendorf (2005, 29-30.) esittävät seuraavanlaisia kriteereitä ohjaamaan mittareiden valinnassa:

- Pysyvä (sustainable). Mittausdatan on oltava käyttökelpoista nyt sekä myös tulevaisuudessa. Kerätystä datasta on pystyttävä tekemään vertailuja myös pitkän ajan jälkeen.
- Ajankohtaista (timely). Datan on oltava raportointihetkellä ajantasaista ja yhdenmukaista. Vanhentuneen datan perusteella ei pystytä tekemään päätöksiä nykyhetkestä tulevaisuutta varten.
- Merkityksellistä (meaningful). Datan on tarjottava sitä tietoa mitä halutaan. Datan on myös oltava hyödynnettävissä päätöksen teon tukena.
- Tavoitteeseen sidottu (goal-oriented). Mittarit on sidottava konkreettisiin tavoitteisiin, jotta niitä voidaan oikeasti hyödyntää päätöksen teossa.
- Tasapainoinen (balanced). Mittarin valinnassa tulisi ottaa huomioon prosessien rajapinnat laajemmalla tasolla. Tarkoituksena ei ole kehittää prosessia erillisenä muista liiketoimintaprosesseista vaan siitä näkökulmasta mikä on tärkeää liiketoiminnan kannalta nyt tai tulevaisuudessa.

8.2.1 Mittareiden valinnan haasteet

Mittareiden valinta ei ole yksikäsitteisesti helppo ja yksinkertainen tehtävä. Mittareiden tulisi tuottaa dataa päätöksen teon tueksi. Tämän vuoksi mittareita varten kerätyn datan tulee olla yksiselitteisesti virheetöntä ja hyödynnettävissä päätöksenteossa. Mittareiden tulisi näin ollen olla sidottuja tavoitteisiin. Se mitä halutaan mittauksella ja seurannalla kontrolloida, määrää sen mitä mitataan ja minkälaisia mittareita käytetään. (Bundschuh ym. 2005, 11-12.)

Oikein määritellyillä mittareilla ja oikein kerätyllä datalla voidaan saavuttaa mittauksessa merkittäviä (Bundschuh ym. 2005, 11.) hyötyjä:

- Läpinäkyvyyttä projektin ja prosessin suorituskyvyssä, sekä parempaa ennustettavuutta.

- Tavoitteiden tarkempi seuranta ja resurssien tehokkaampi allokointi.
- Vähemmän päällekkäisyyksiä ja turhia projekteja.
- Tehokkaampi kehitystoiminta, parempi tuottosuhde kehitystyössä suhteessa investointeihin.
- Parempi kommunikaatio liiketoiminnan ja tuotekehityksen välillä.
- Tulevien tuotekehitysinvestointien tukeminen.

Mittaaminen itsessään on helppoa ja myös väärin määritellyillä ja väärin käytetyillä mittareilla voidaan saada tuotettua data aikaiseksi. Tällainen data ei välttämättä kuitenkaan kerro yhtään mitään, eikä auta päätösten teossa. Tällaiseen tilanteeseen saatetaan ajautua, jos mitataan asioita jotka eivät ole sidottuja mihinkään organisaation tavoitteisiin. Huomionarvoista on myös se seikka, että vaikka mittarit olisivat sidottuja tavoitteisiin, tavoitteet eivät saa olla epärealistisia. Tavoitteiden on aina oltava saavutettavissa, jottei organisaatio joudu toimimaan jatkuvasti suorituskykynsä äärirajoilla. Toinen ongelmakohta tulee vastaan, mikäli kerättyä dataa ei analysoida millään tavoin. Kerätystä datasta on helppoa tehdä kaavioita ja taulukoita, mutta ne ovat täysin hyödyttömiä, mikäli niitä ei analysoida toiminnan parantamisen näkökulmasta. Vaikka datan kerääminen mittausmielessä onkin tärkeää, on vältettävä keräämästä likaa dataa. Liiallinen datan keräys johtaa ylimääräiseen byrokratiaan ja hallinnointityöhön ja voi olla täyttä ajan hukkaa. Mittaus siis ei ole erillinen toiminto organisaatiossa, vaan enemmänkin sisäänrakennettu osa nykyistä toimintaa olennaisesti siihen liittyen. (Bundschuh ym. 2005, 3-4.)

Väärin valituilla mittareilla voidaan aiheuttaa ylimääräistä työtä. Huonoilla mittareilla voidaan myös tahallisesti tai tahattomasti saada piilotettua dataa, joka olisi oikeasti tarpeellista ja tärkeää. Suurempi vaara piilee kuitenkin siinä, että virheellisillä mittareilla kerättyä dataa halutaan käyttää totuuden piilotteluun ja tosiasioiden kaunisteluun. (Bundschuh ym. 2005, 13.)

Mittareiden valinnan haasteet ja niihin liittyvät vaikeudet korostuvat silloin, mikäli kehitystä näyttäisi joidenkin mittareiden mukaan tapahtuvat. Vaikka mittarit lopulta osoittautuisivatkin vääriksi, voivat ne silti näyttää kehitystä tapahtuneen. Oikeasti mittarit eivät kuitenkaan kerro mitään niistä asioista, jotka ovat oikeasti merkityksellisiä vaan

antavat mittausajankohdan tilanteessa hyvin ruusuisen ja optimisten kuvan. Tällaisia mittareita voidaan kutsua myös turhuusmittareiksi (vanity metric). Tämän vuoksi mittareiden tulisi mitata niitä asioita joilla on merkitystä. (Ries 2011, 128-130.)

Mittareiden osalta on tärkeää, että mittareilla pystytään esittämään selvä syy-seuraussuhde. Esimerkiksi internet-sivujen latausmäärien kasvun ei välttämättä voida osoittaa johtuvan vain yhdestä asiasta. Sivujen latausmäärä on myös yksi esimerkki turhuusmittareista. Turhuusmittareille tunnusomaista on se, että mittausarvon noustessa, moni taho on valmis ottamaan siitä kunnian itselleen. Mittausarvon laskiessa puolestaan kukaan ei halua ottaa vastuuta siitä. Tämän vuoksi turhuusmittarit ovatkin hyvin hämmennystä aiheuttavia ja niiden käyttöä tulisi välttää. (Ries 2011, 143-144.)

Olipa mittari lopulta mikä vain, tulee mittausdatasta tehtyjen raporttien olla riittävän yksinkertaisia. Yksinkertaisuus on raportin kannalta tärkeää siksi, että kaikki jotka haluavat raportin lukea, ymmärtävät sen. Kuten aikaisemmin on mainittu, on raportoitavan datan oltava ehdottomasti luetettavaa ja virheetöntä. Tarvittaessa data on myös voitava jälkikäteen validoida, mikäli tarvetta ilmenee. (Ries 2011, 144-147.)

8.2.2 Kehitysprojektissa käytetyt mittarit

Kehitysprojektin aika asetettiin tavoitteeksi selvittää kuinka tehtävien läpimenoaikoja (cycle time, lead time) saataisiin pienennettyä ja siten tehtävien läpimenoa nopeutettua. Läpimenoaikaa ei seurattu aikaisemmin mitenkään. Projektin aikana päätettiin alkaa seuraamaan läpimenoaikaa kahdella eri tavalla. Ensiksi aloitettiin sen seuranta, kuinka kauan tehtävillä kestää siirtyä ready for testing-tilasta closed-tilaan. Tämä haluttiin tehdä sen vuoksi koska havaittiin, että usein tehtävillä oli tapana jäädä odottamaan testausta hyvinkin pitkäksi aikaa. Pitkä testauksen odotusaika puolestaan aina viivästyttää virheiden ja muiden ei-toivottujen toiminnallisuuksien havaitsemista. Myöhäisempi virheiden havaitseminen on myös laadunvarmistuksen kannalta ongelma, joten siihen haluttiin saada parannusta. Sen vuoksi se oli perusteltu valinta yhdeksi ensimmäisistä uusista mittareista. Tehtävien aika testausjonossa oli laskettavista takautuvasti myös vanhoille, jo toteutetuille tehtäville joten ajan kehittymistä pystyttiin seuraamaan ja kehitysprojek-

tissa tehtyjen muutoksien vaikutuksia arvioimaan. Kyseiset mittaustulokset ovat myös vertailukelpoisia uusien ja tulevien vastaavien mittaustulosten kanssa.

Projektin edetessä läpimenoaikaa alettiin lisäksi seurata Kanbanin mukaisesti (Kniberg & Skarin 2010, 3-5.), eli koko prosessin alusta loppuun saakka. Tämä tarkoitti siis sitä, että siitä hetkestä kun tehtävä on valittu tehtäväksi, aletaan sille laskea läpimenoaikaa. Loppuaika läpimenolle tulee tässäkin tapauksessa sillä hetkellä, kun tehtävä saavuttaa closed-tilan. Ongelmana alkutilanteessa oli se, ettei ollut määriteltävissä yksikäsitteisesti sitä hetkeä jolloin tehtävä olisi valittu tehtäväksi. Tämän vuoksi päädyttiin lisäämään jokaiselle tehtävälle uusi kenttä joka kertoisi hetken jolloin tehtävä on valittu tehtävien töiden joukkoon. Koska kyseessä oli kokonaan uusi tietokenttä, ei läpimenoaikaa voitu laskea takautuvasti vanhemmille tehtäville. Myöhemmin kehitysprojektin aikana päivämääräkenttä korvautui työjonolla, jolloin läpimenoaikalaskenta tehtäville alkoi tehtävän työjonoon saapumisesta.

Vuoden 2012 alussa tehtiin lisäys mittareihin tiimin omien havaintojen pohjalta. Ajoittain testausta odottavien tehtävien lukumäärä kasvoi huomattavan suureksi. Osaksi se johtui siitä, ettei kaikkia tehtäviä jotka odottivat testausta, oltu visualisoitu omilla lapuillaan Kanban-taululla. Osaksi se johtui siitä, ettei asiaa seurattu. Tämän vuoksi päädyttiin lisäämään seurattaviin mittareihin testausta odottavien tehtävien lukumäärä aina työviikon lopussa. Läpimenoajan seuraaminen kertoo saman asian viiveellä, mahdollisesti vasta viikkojen kuluttua, että yksi tai useampi tehtävä odotti testausta huomattavan kauan. Pyrkimyksenä päätettiin olevan se, ettei työviikon päätteeksi jää yhtään tehtävää odottamaan testausta, vaan työviikko voidaan aloittaa aina ns. puhtaalta pöydältä. Käytännössä tämä osoittautui hieman liian tiukaksi rajoitteeksi sillä on mahdollista että tehtävä on voitu saada valmiiksi hyvin myöhäisessä vaiheessa työviikkoa. Tällöin ei ole realistista tai perusteltua odottaa että se voitaisiin vielä saada testattuakin työviikon loppuun mennessä, etenkin jos kyseessä on ollut monimutkaisempi tehtävä. Testaamattomien tehtävien määrä voitiin laskea takautuvasti, joka paljasti testauksessa olleen jonkinasteisesta kontrollin puutetta tiimin historiassa.

Tiimin projekteissa oli myös eroavaisuuksia sen suhteen mitä tietoja tehtävistä kerätään. Merkittävimpänä erona oli se, kerätäänkö tehtävistä tehtäväpisteitä vauhdin laskemista

varten vai ei. Kun joistakin projekteista kyseistä tieto kerättiin ja joistakin ei, eivät tilastot eri työviikkojen välillä olleet vertailukelpoisia. Sen vuoksi vuoden 2012 alussa päätettiin, että kaikista tehtävistä kerätään myös tehtäväpisteitä, jotta tilastoista saadaan vertailukelpoista tietoa.

Yhdenmukaistamalla eri tuotekehitystiimin projektien mittarit, saadaan myös vertailukelpoista mittausdataa eri projektien välillä. Samalla myös mittausdata on vertailukelpoista myös myöhemmin.

8.3 Tuotekehitysprosessin muutokset

Prosessi oli lähtötilanteessa scrumista, Leanista ja Kanbanista vaikutteita ottava. Tähän ei haluttu muutosta, vaan prosessin parannuksia lähdettiin miettimään nykyisen prosessin pohjalta. Projektissa haluttiin etsiä keinoja hyödyntää käytössä olevia menetelmiä paremmin ja tehokkaammin sekä löytää myös uusista aikaisemmin käytössä olevista menetelmistä hyödynnettävää tiimin prosessin kehitystarpeisiin.

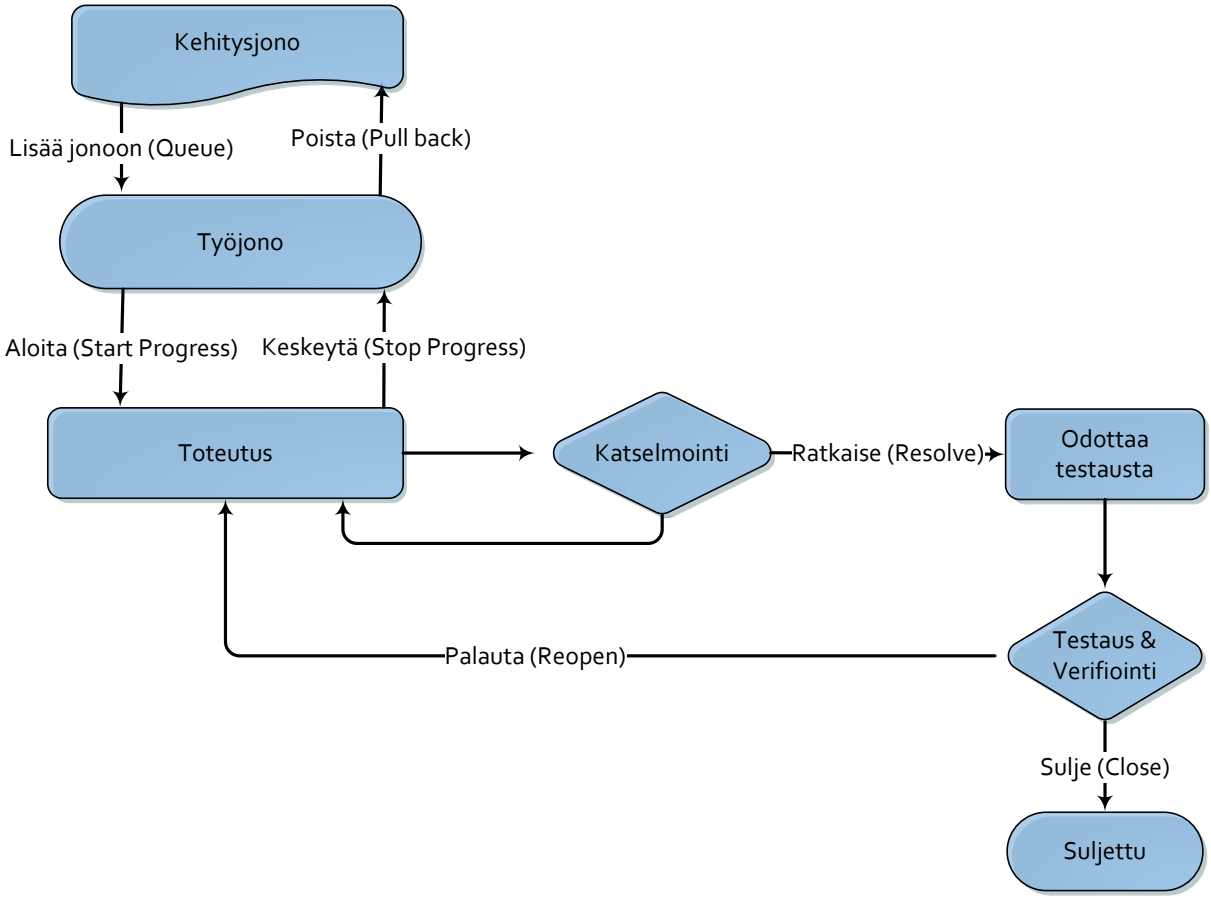
Prosessin kehittämiseen haettiin kehitysprojektissa ideoita CMMI-DEV 1.3:ta. CMMI-DEV valittiin prosessin kehitystoimintaa ohjaavaksi sen vuoksi, että se ei rajaa liian tiukkaan sitä miten tai millä keinoin mallin vaatimukset on täytettävä. Sen lisäksi CMMI-DEV:in valintaa puolsi mallin yhteensopivuus tiimissä käytettyjen ketterien menetelmien: scrumin, Kanbanin sekä Leanin kanssa.

Koska kyseessä oli projektin yhteydessä kuitenkin pienen tiimin prosessikehitys, oli prosessin kehityksessä käytettävän viitekehyksen oltava myös riittävän kevyt ja moniin tarkoituksiin mukautuva. Koska projektin aikana ei varsinaisesti haluttu alkaa jalkauttamaan pieneen tiimiin raskasta tai monia muutoksia vaativaa viitekehystä, päädyttiin kehitystä ohjaamaan valita tarkoitukseen sopivat prosessialueet. Valittujen prosessialueiden osalta päädyttiin kehittämään prosessialueiden kyvykkyyttä. Kypsyyden kehittämisen nähtiin olevan liian raskas kehitysprojektin yhteydessä, eikä sitä olisi ollut realistisesti mahdollista toteuttaa.

Projektin keskittyessä yhden tiimin prosessin kehittämiseen, päätettiin heti aluksi rajata pois kaikki ne prosessikategoriat ja alueet jotka koskivat kehittämistä koko organisaation tasolla. Näin ollen koko Process Management- sekä myös Project Management-kategoriat rajattiin ulos. Koska kehitysprojektin yhteydessä haluttiin pohtia sekä laadunvarmistusta, että prosessin kehittämistä, valittiin kehitettäviksi prosessialueiksi toisiinsa liittyvät prosessialueet validointi (validation) ja verifiointi (verification). Kuten Potter & Sakry (2011) toteavat, että 3 kypsyystason prosessit, joihin validointi ja verifiointi kuuluvat, ovat niitä joiden vaikutusalalla scrum käytännössä toimii. Tämäkin seikka puoltaa kyseisten prosessialueiden valintaa kehittämiskohteiksi.

8.3.1 Tehtävien kulku prosessissa

Tuotekehitystiimissä työstettävien tehtävien kulku ja siten päivittäistä kehitystoimintaa ohjaava prosessi muotoutui kehitysprojektin kuluessa kuvion 26 kaltaiseksi. Kuviosta 26 käy ilmi yksi merkittävä eroavaisuus tuotekehitysprosessin alkutilanteeseen verrattuna. Kehitysprojektin alkuvaiheessa prosessia käytännössä pyöritettiin tuotteiden kehitysjonon kautta. Toisin sanoen, teknisessä mielessä koko kehitysjono oli kokonaisuudessaan syötteenä vanhalle tuotekehitysprosessille. Uuden prosessin myötä syötteet prosessille tulevat työjonon kautta. Kehitysprosessia on siis hieman virtaviivaistettu ja yksinkertaistettu. Nuolten tekstit kuviossa 26 kuvaavat JIRAn workflow-toimintoja, eli toimintoja joilla tehtävät siirtyvät tilasta toiseen.



Kuvio 26. Uusi tuotekehitysprosessi

8.3.2 Tiedon jakaminen

Tuotekehitystiimissä haluttiin kehittää tiedon ja osaamisen jakamista. Tämä oli seurausta siitä, että tiimin jäsenet tekivät ajoittain töitä eri asioiden, teknologioiden ja välineiden kanssa. Tiimin jäsenten tällä tavoin keräämä tieto ja osaaminen haluttiin saada jaettua myös muiden tiimin jäsenten kesken. Tämän vuoksi prosessiin päätettiin ottaa mukaan tiedon jakamista ja uuden oppimista palveleva tiimin sisäinen koulutussessio, jota kutsuttiin milk&cookies-nimellä. Lähtökohtaisesti tavoitteena oli pitää kyseisiä tiedon jakamiseen perustuvia sessioita aina tiimin retrospektiiveissä, mutta niitä päätettiin pitää mahdollisuuksien ja tarpeiden mukaan muulloinkin. Periaatteena oli se, että jokainen tiimin jäsen voisi pitää haluamastaan asiasta kevyen koulutussession muille tiimin jäsenille. Lähtökohtana pidettiin kuitenkin vapaaehtoisuutta, ketään ei pakotettu pitämään sessiota mistään aiheesta, mutta vapaavalintaisten aiheiden toivottiin lisäävän innokkuutta asian selvittämiseen ja niiden muille opettamiseen.

8.3.3 Työmääräarviointi

Työmääräarviointia muutettiin projektin aikana tuotekehitysprosessissa vapaammaksi verrattuna lähtötilanteeseen. Työmääräarvioinneissa päätettiin luopua yhteisistä työmääräarvioiden tekemisestä. Vaihe näin ollen poistettiin tiimin prosessista. Sen sijaan jokainen kehittäjä asettaa työmäärän, eli tehtäväpisteet, itsenäisesti toteuttamalleen tehtävälle. Tiimissä on ajan myötä konsensus kuinka monta tehtäväpistettä erilaiset tehtävät keskimäärin ovat historiassa olleet. Näin voidaan säästää erillisen työmääräarviointipalaveriin kulunut aika, joka sitoi jokaisen tiimin jäsenen aikaa, ja käyttää säästynyt aika tuottavammin.

8.3.4 Testaus ja laadunvarmistus

Testausta koskien tehtiin muutos lähtötilanteeseen verrattuna. Alkutilanteessa käytössä olleesta 3 viikkoa kehitystyötä jota seurasi 1 viikko testausta -toimintamallista, luovuttiin. Ratkaisuun päädyttiin sen vuoksi, että testaustaakka ei ollut kovin tasaisesti jakautunut, eikä myöskään tuottanut toivottuja lopputuloksia. Testausroolista tuli lopulta enemmän virtuaalinen rooli, johon ketään tiimin jäsentä ei suoraan osoiteta. Testausta tekevät tiimin jäsenet, kun kehitystyöltä ehtivät, joko yksi tai useampi henkilö kerralla tarpeen niin vaatiessa.

Osin eräs testausta koskeva muutos koski myös läpimenoaikamittausta. Prosessissa päätettiin painottaa enemmän sitä, että jokaisen on testattava tehtäviä ennen kuin alkaa uuden toteuttamisen. Tämä tehtiin sillä ehdolla, etteivät testattavat tehtävät olleen kyseisen kehittäjän toteuttamia. Tämä kirjoitettiin myös Kanban-taululle muistuttamaan kehittäjiä muuttuneesta toimintojen priorisoinnista prosessista. Tavoitteena oli saada tehokkaammalla testauksella ja verifioinnilla tehtävien testauksen odotusaikaa pienennettyä ja sitä kautta pienennettyä myös läpimenoaikaa.

Testauksen painottaminen ennen kehittämistä oli vain yksi vaihe testauksen kehittämisessä. Lisäksi havaittiin että tiimin käytössä olevalla koontipalvelimella oli jatkuvasti virhetilanteita. Virhetilanteita olivat siis ne että koonti (build) ei mennyt täysin hyväksytysti läpi sillä joko yksikkötestien tai integraatiotestien suoritukset eivät menneet läpi. Sen vuoksi päätettiin prosessiin lisätä myös painotus sille, että ennen testausta, on ke-

hittäjän tarkistettava onko koonti mennyt läpi ja jos ei, niin korjata koonti jälleen läpi meneväksi. Myös tästä lisättiin Kanban-taululle muistutus.

Projektin alkutilanteen mukaisesti laadunvarmistukselliset toimenpiteet myös säilyivät tiimin sisäisinä aktiviteetteina. Vaikka esimerkiksi Schulmeyerin (2008, 16) mukaan sisäinen laadunvarmistus voisikin asettaa ongelmia laadunvarmistuksen objektiivisuudelle, haluttiin tiimissä pitäytyä siinä linjassa, että laatu ”rakennetaan tuotteisiin sisään” ilman ulkopuolista testausorganisaatiota. Ulkoisen laadunvarmistuksen hankkiminen olisi myös voinut olla hyvin haastavaa esimerkiksi aikataulullisesti tämän projektin aikana. Ulkopuolisenkin laadunvarmistuksen organisoiminen vie aikaa, eikä tällöin välttämättä olisi edes ehditty arvioida ulkopuolisen laadunvarmistuksen vaikutuksia tämän kehitysprojektin aikana.

8.3.5 Muuttunut Kanban-taulu

Tuotekehitystiimin Kanban-taulu koki muutoksia kehitysprojektin edetessä eri seikoista johtuen. Muutoksia tehtiin tauluun aina sitä mukaa, kun tiimissä havaittiin taulussa olevan puutteita tai siihen keksittiin parannuksia.

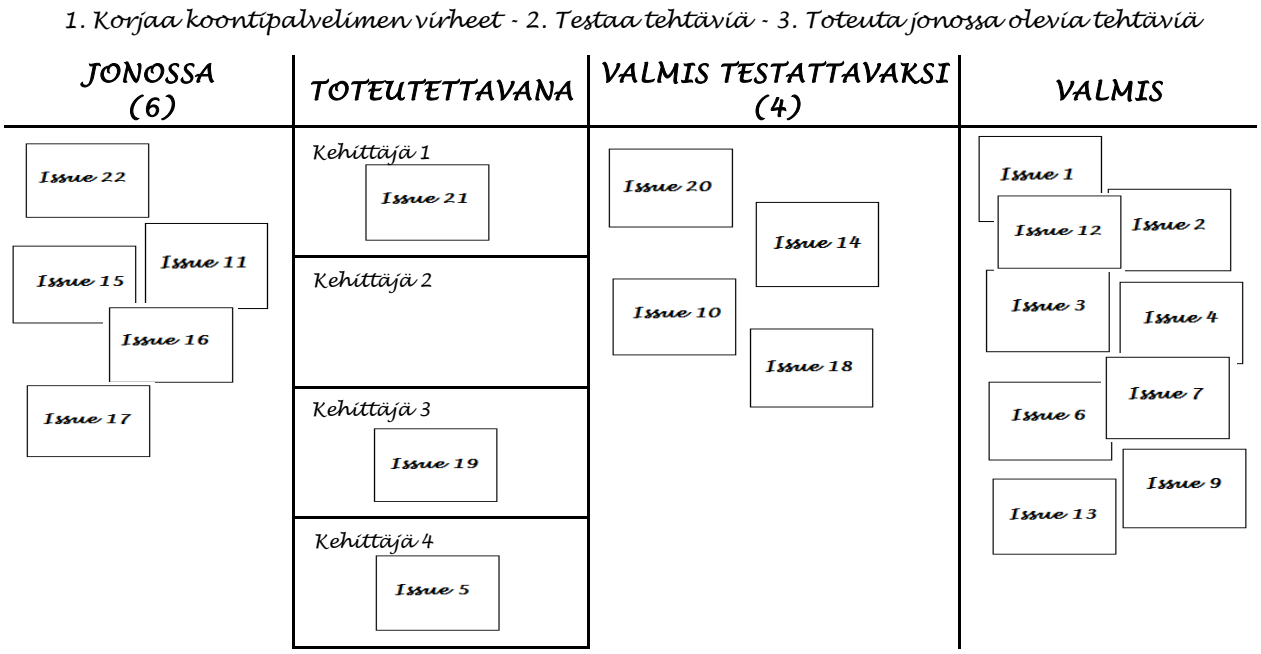
Yhtenä muutoksena päätettiin rajoittaa kehitysprojektin aikana käyttöönotetun työjonon kokoa. Tämä oli seurausta siitä, että kehitysprojektin aikana havaittiin, ettei tehtävien läpimenoaika kaikkien tehtävien osalta kehittynyt halutulla tavalla työjonon käyttöönoton jälkeen. Joidenkin tehtävien läpimenoaika oli huomattavan pitkä verrattuna muihin tehtäviin tai niiden läpimenoajan keskiarvoon. Osasyyksi havaittiin se, että työjonon maksimikoko oli liian suuri. Tämä johtui siitä kun jonossa oli tilaa lisätä uusia tehtäviä, niin myös tehtiin. Tällöin jonossa jo aikaisemmin olleet tehtävät, jotka olivat pienemmällä prioriteetilla, jäivät edelleen odottamaan vuoroaan työjonoon. Tällöin jonossa oloaika kasvoi joidenkin tehtävien osalta hyvinkin pitkäksi, mikä omalta osaltaan kasvatti myös läpimenoaikaa. Työjonon käyttöönoton alkuvaiheessa työjonon koko oli rajattu 10:een, mutta myöhemmin työjono rajoitettiin lopulta 6:een tehtävään. Tavoitteena oli, ettei työjonoon jäisi tehtäviä odottamaan vuoroaan pitkäksi aikaa edellä kuvatulla tavalla. Uuden rajoituksen myötä vähemmän tärkeät tehtävät poistettiin työjonosta

mikäli korkeamman prioriteetin tehtäviä tuli tehtäväksi. Myöhemmin jonosta poistetut tehtävät voitiin lisätä uudestaan työjonoon tilanteen niin vaatiessa.

Työjonon rajoittamisen lisäksi tiimissä havaittiin tarpeelliseksi rajoittaa testausta odottavien tehtävien lukumäärää, sen lisäksi että mittareiden avulla jo seurattiin sitä kuinka monta tehtävää jäi odottamaan testausta työviikon lopuksi. Ajoittain testausta odottavien tehtävien määrä kasvoi huomattavan suureksi, mikä osaltaan johti siihen että tehtäviä myös jäi odottamaan testausta seuraavaa työviikkoa varten. Tavoitteena oli kuitenkin saada kaikki testausta odottavat tehtävät testattua työviikon loppuun mennessä. Testausta odottavien tehtävien lukumäärä asetettiin 4:ään, jottei testausta odottavien tehtävien jono pääsisi kasvavaan liian suureksi. Rajoituksen tarkoituksena on myös ohjata tiimin jäseniä automaattisesti testaamaan tehtäviä heidän huomatessaan testausta odottavien määrän lähestyessä sallittua raja-arvoa ennen uuden tehtävän työstämistä.

Rajoitukset eri tiloille asetettiin huhti-toukokuussa 2012. Koska rajoitukset ovat tilastojen osalta olleet vasta huomattavan vähän aikaa käytössä, ei niiden vaikutusta läpimenoaikaan pystytä vielä tässä vaiheessa arvioimaan kovin luotettavasti.

Kanban-taulu muotoutui lopulta kuvion 27 kaltaiseksi kehitysprojektin edetessä. Kanban-taululla ovat mukana rajoitukset eri sarakkeille, sekä muistutukset tiimille tehtäviä koskevien aktiviteettien tärkeysjärjestyksestä.



Kuvio 27. Tuotekehitystiimin Kanban-taulu kehitysprojektin lopussa

9 Tulokset

Tässä luvussa käydään läpi kehitysprojektin tuloksia. Ensin käydään läpi projektien tilastotietojen muodossa kehitysprojektin aikana tehtyjen muutosten vaikutuksia. Tilastojen jälkeen esitellään CMMI-DEV:in muutosten vaikutusta tuotekehitysprosessiin, sekä arvioidaan vapaamuotoisesti kehitettäväksi valittujen prosessialueiden kyvykkyyttä ennen ja jälkeen kehitysprojektia.

9.1 Projektitilastovertailun tulokset

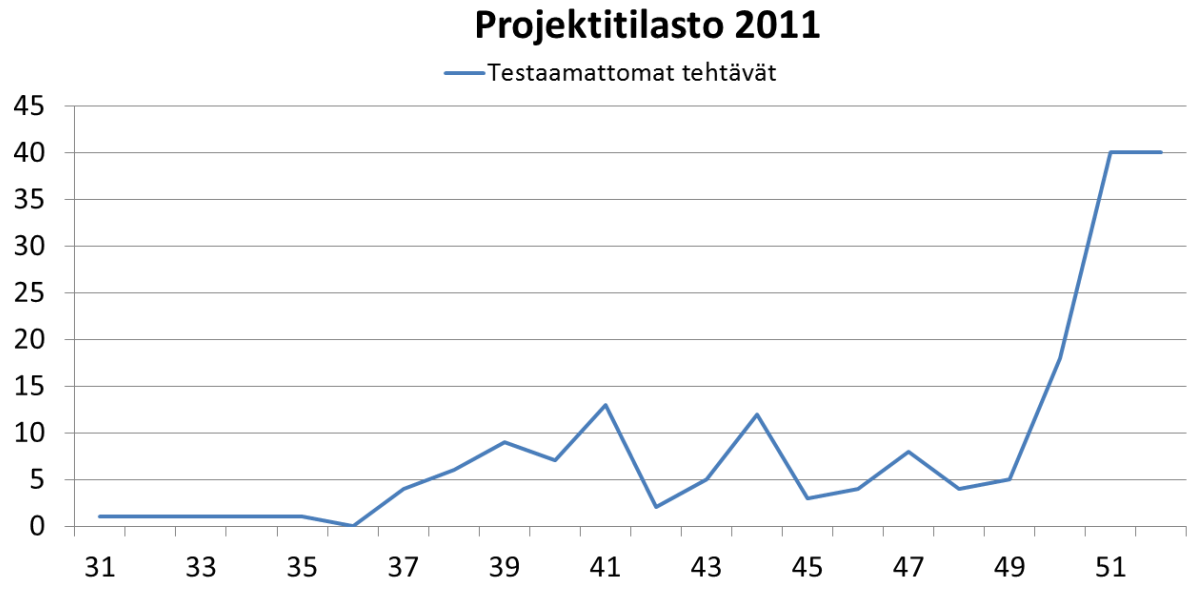
Tämän kehitysprojektin osalta tilastotiedot on jaettu kahteen eri osaan. Ensimmäinen osa on vuoden 2011 elokuusta joulukuun loppuun asti. Toinen osa on vuoden 2012 tammikuusta elokuun loppuun asti. Kokonaisuudessaan tilastotietoja on kerätty hieman yli vuoden ajalta. Raja kahden eri tilastojakson välillä on vedetty vuoden vaihteeseen sen vuoksi, että siinä vaiheessa tehtiin päätös alkaa seuraamaan testaamattomien tehtävien lukumäärää työviikon päätteeksi, sekä kannustamaan aktiivisesti testaamaan tehtäviä, kun testattavaa oli. Samanaikaisesti myös päätettiin yhdenmukaistaa projekteista kerättävät tilastot, aloittamalla keräämään tehtäväpisteistä jokaisesta tehtävästä jokaisessa projektissa. Aloittamalla uusi mittausjakso kyseisestä hetkestä voidaan nähdä muutoksen vaikutukset tilastojen muodostumiseen. Tilastot esitetään viikkotasolla.

Vuoden 2011 ja 2012 tilastoissa on se ero, ettei vuoden 2011 tilastoissa ole mukana tehtäväpisteitä. Se johtuu aikaisemmin mainituista eroista projektien välillä jonka mukaan kaikilla projekteilla ei ollut tehtävillä käytössä tehtäväpisteitä. Tämän vuoksi ne on jätetty vuodelta 2011 kokonaan pois. Eräänlaisena vauhtimittarina käytetään sen sijaan valmiiksi ja loppuun asti saatettujen tehtävien (closed issues) lukumäärää.

9.1.1 Vuoden 2011 projektitilastot

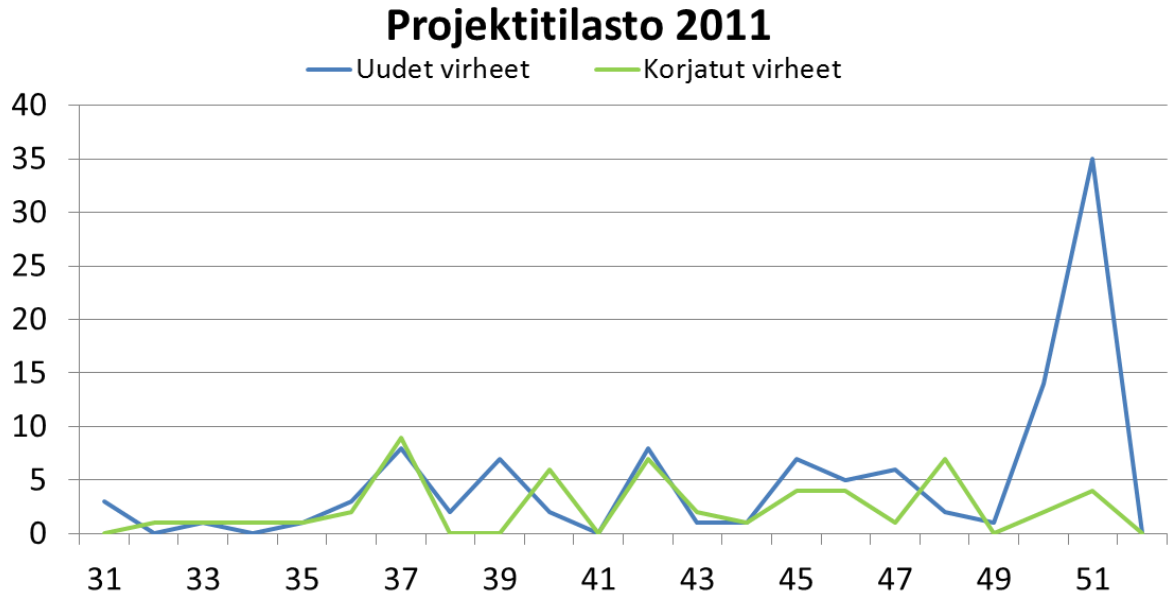
Vuoden 2011 testaamattomien tehtävien tilastosta (kuvio 28) voidaan havaita, että testausjonossa (untested issues) on ollut koko ajan tehtäviä odottamassa testausta. Usein testaamatta jääneiden tehtävien jono on viikon loppuun mennessä kasvanut yli 5:n ja ajoittain jopa yli 10:n tehtävän mittaiseksi. Vuoden loppua kohti jono kasvoi kaikkien

aikojen huippuunsa 40:een tehtävään. Määrä on ollut hämmästyttävän suuri ja osoittaa, että testauksen suhteen on ollut pahoja puutteita.



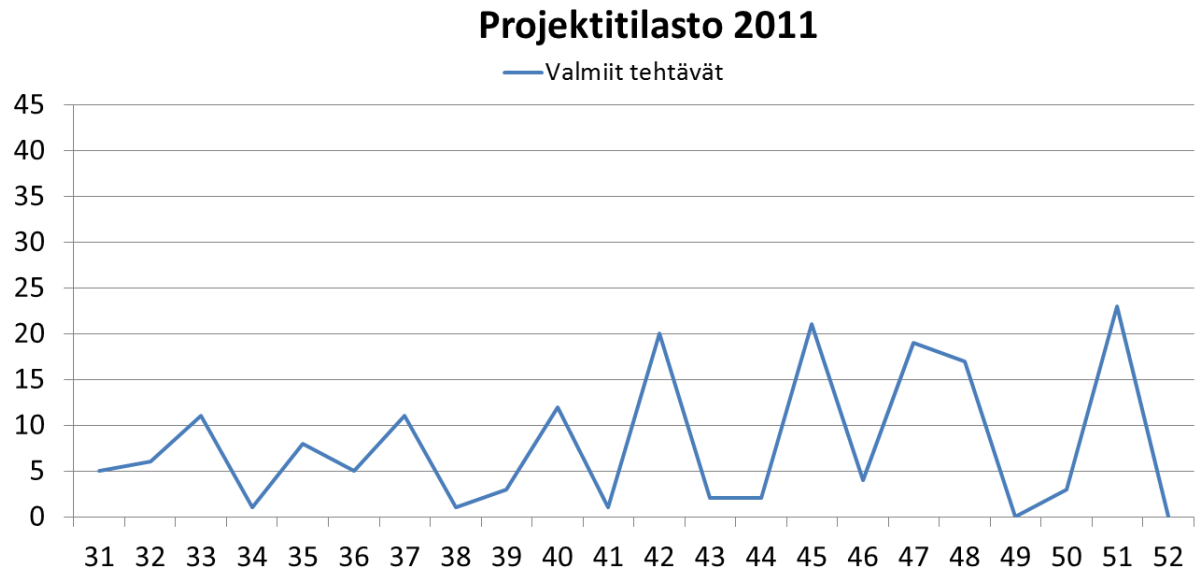
Kuvio 28. Testausjonon koko vuonna 2011

Merkittävä ongelma on myös ollut vuoden 2011 lopussa löydettyjen uusien virheiden (new bugs) valtava määrä. Uusien virheiden määrä on noussut uuteen huippuunsa samaan aikaan testaamattomien tehtävien kanssa, ollen huikeat 35 vuoden 2011 lopussa, kuten kuvio 29 esittää. Periaatteessa tavoitteena on saada enemmän virheitä korjattua, kuin mitä uusia on havaittu. Se ei kuitenkaan ole onnistunut vuoden 2011 aikana.



Kuvio 29. Uusien virheiden ja korjattujen virheiden määrä vuonna 2011

Mitä tulee vauhtiin valmiiksi tehtyjen tehtävien lukumäärän osalta, niin sen suhteen on havaittavissa suurta vaihtelua eri viikkojen välillä. Koska prosessin suorituskyky on lopulta kiinni siitä kuinka monta tehtävää saadaan valmiiksi viikon loppuun mennessä, voidaan tämän perusteella todeta, ettei suorituskyky ole ollut ainakaan kovin tasainen vuoden 2011 aikana. Kuvio 30 esittää vaihtelun valmiiden tehtävien lukumäärässä vuoden 2011 aikana.



Kuvio 30. Valmiiden tehtävien määrä vuonna 2011

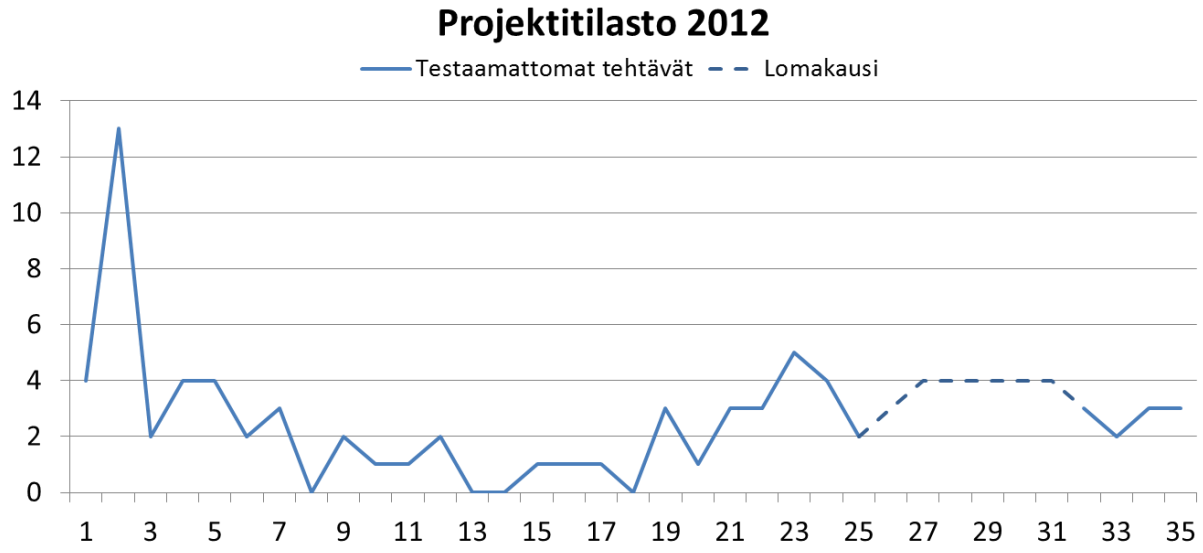
Vuoden 2011 valmistuneiden tehtävien tilastoista voidaan havaita, että useimmiten suuret selkeästi muusta linjasta poikkeavat arvot ovat seurausta aikaisemmin testaamattomien tehtävien kasautumisesta. Kun testausjonoa on saatu purettua, on tämä näkynyt myöhemmin suurena määränä valmistuneita tehtäviä.

Vuoden 2011 lopussa tuotekehitystiimille tuli suuria aikataulupaineita kiireellisen projektin muodossa. Aikataulupaineiden merkitys oli huomattavan suuri tilastoissa esiintyvään piikkiin testaamattomien tehtävien osalta vuoden 2011 lopussa ja vuoden 2012 alussa. Tämä vahvistaa teoriassa (Software Engineering Institute 2010, 27; Chrissis, Konrad & Shrum 2011, 42) esitetyn väitteen normaalien käytäntöjen hylkäämisestä aikataulupaineiden edessä, kun määritellyt prosessit ja niiden seuranta ovat puutteellisia.

9.1.2 Vuoden 2012 projektitilastot

Vuoden 2012 projektitilastoissa on mukana myös kesälomakausi, joka on maininnan arvoinen asia siinä mielessä, että sen vaikutus näkyy tilastoissa selkeästi. Kesälomakaudella viikosta 25 viikolle 32 asti on ollut tiimin miehitys vajaata, vaihdellen aina yhdestä henkilöstä kolmeen. Tilastot kyseiseltä ajanjaksolta ovat kuitenkin samalla tavalla kerättyjä.

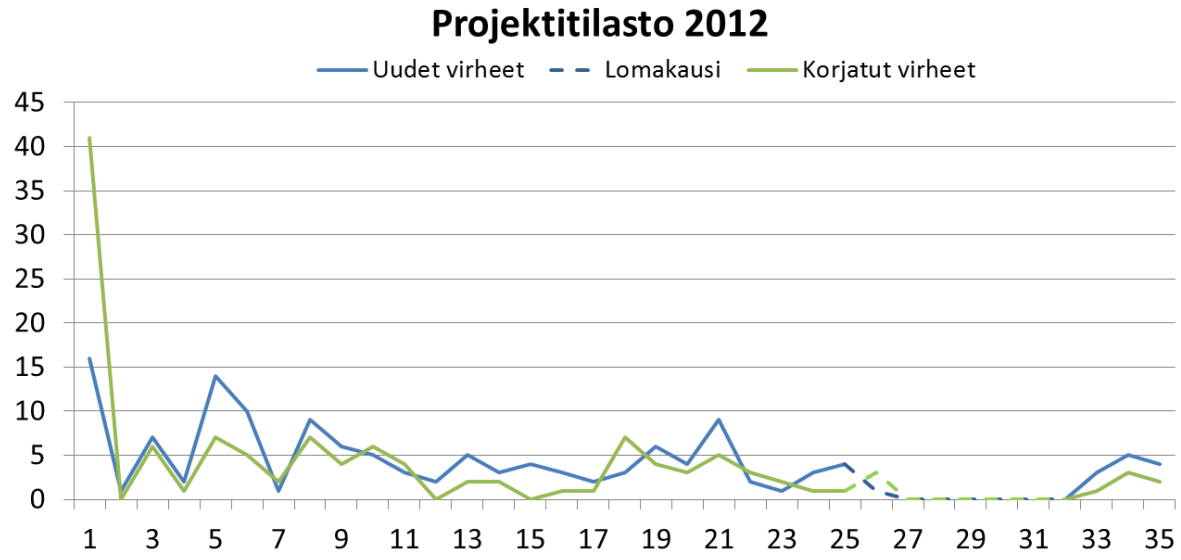
Vuoden 2012 tilastoissa testaamattomien tehtävien osalta voidaan huomata, että kun alkuvuonna on saatu purettua suuri testaamattomien tehtävien jono, ei jono ole uudestaan kasvanut missään vaiheessa lähellekään 2011 vuoden huippulukemia. Korkeimmillaan jono on kasvanut 5:een. Joinakin viikkoina on päästy jo tavoitteeseenkin, eli 0:aan. Testausjonon koko vuonna 2012 on esitetty kuvoissa 31. Testausjonon kasvu on suuren osan ajasta saatu pidettyä kohtuullisen hyvin kurissa. Lomakauden aikana tilanne on pysynyt täysin muuttumattomana tiimin jatkuvasta, joskin vaihtelevasta, miehityksestä huolimatta.



Kuvio 31. Testausjonon koko vuonna 2012

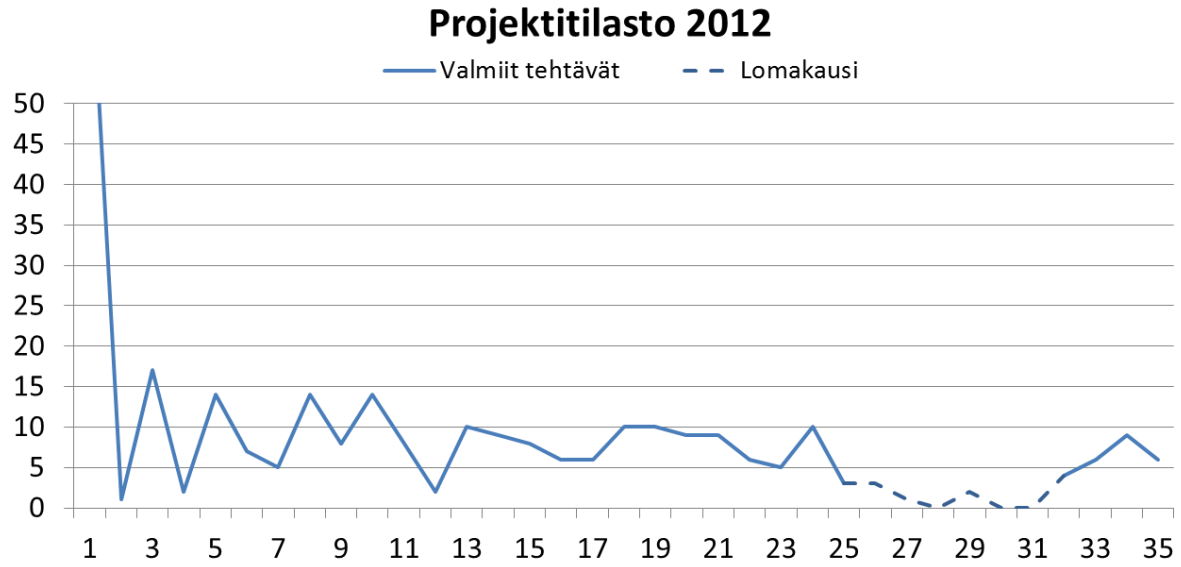
Vuonna 2012 uusien virheiden osalta esiintyy vaihtelua eri viikkojen välillä. Yhtä korkeaa tilastopiikkiä ei kuitenkaan esiinny kuin vuoden 2011 tilastoissa. Lomakauden aikana ei uusia virheitä ole löytynyt. Vaikka lomakauden yleisesti ottaen ovatkin hiljaisempia

ajanjaksoja työyhteisöissä, voi tässä tapauksessa kyse olla kuitenkin siitä, ettei laadunvarmistusta ole kyseisenä ajanjaksona tehty. Uusien virheiden lukumäärä vuonna 2012 on esitetty kuviossa 32. Myös vuoden 2012 osalta korjattujen virheiden lukumäärä on useiden viikkojen osalta pienempi, kuin mitä on ollut uusien virheiden määrä.



Kuvio 32. Uusien virheiden ja korjattujen virheiden määrä vuonna 2012

Vuoden 2012 valmiiden tehtävien tilaston osalta voidaan havaita, että valmistuneiden tehtävien lukumäärässä ei ole läheskään yhtä suurta vaihtelua, kuin vuonna 2011 oli. Tilasto myös osoittaa selvästi suorituskyvyn tasaantumista valmiiden tehtävien suhteen viikosta 13 eteenpäin lomakauden alkuun saakka. Valmiiden tehtävien tilasto on esitetty kuviossa 33.



Kuvio 33. Valmiiden tehtävien määrä vuonna 2012

Vuonna 2012 kaikille tehtäville alettiin laskea vauhtia tehtäväpisteiden muodossa. Kuviossa 34 on esitetty mikä on ollut tuotekehitystiimin vauhti tehtäväpisteillä laskettuna vuonna 2012. Lomakauden aikana vauhdissa on luonnollisesti tapahtunut selkeä notkahdus. Notkahdusta lukuun ottamatta vauhti on pysynyt pääsääntöisesti 12 ja 23 tehtäväpisteen välillä muutamaa korkeampaa piikkiä lukuun ottamatta.

Kuvio 34. Tehtäväpisteiden määrä vuonna 2012

9.1.3 Yhteenveto projektitilastoista

Verrattaessa sekä vuoden 2011, että vuoden 2012 tilastoista niitä arvoja, jotka molemmista tilastoista löytyvät, vuonna 2012 vaihteluväliä on saatu tasattua ja vuonna 2011 esiintyneet terävimmät kärjet leikattua pois. Näin ollen voidaan sanoa, että tilastojen valossa prosessi on saatu ainakin hieman paremmin hallintaan vuoden 2012 aikana, joka on johtanut vaihteluiden pienenemiseen. Vuoden 2012 tilastoissa on ennen lomakautta jakso jossa on selvää tasaantumista arvojen vaihtelussa, varsinkin valmiiden tehtävien osalta (kuvio 33). Tämä kertoo sen, että muutoksille tulee antaa aikaa, jotta niiden vaikutukset voidaan huomata sillä ne voidaan havaita mahdollisesti vasta jonkin ajan kuluttua muutosten toteutuksesta. Tämän projektin tapauksessa suuret muutokset tapahtuivat vuoden 2012 alkupuolella, kun muutos haluttiin ajaa sisään välittömästi siltä osin, että testausjono saadaan lyhennettyä ja mahdollisesti siivottua kokonaan.

Kuten vuoden 2011 tilastoista (kuvio 28) voidaan nähdä, niin testausta odottavien tehtävien testaaminen ei ratkennut tiimissä nimetyllä testaajalla, jolloin sellainen oli tiimin käytännöissä vielä mukana. Nimetystä testausroolista tiimin sisällä luovuttiin vuoden 2012 alussa. Tilastojen valossa voidaan kuitenkin todeta, että painottamalla testausta päivittäin ja fokusoimalla siihen, on saatu aikaan parannusta prosessissa testausjonon lyhenemisen muodossa, joka käy ilmi vuoden 2012 tilastosta (kuvio 31).

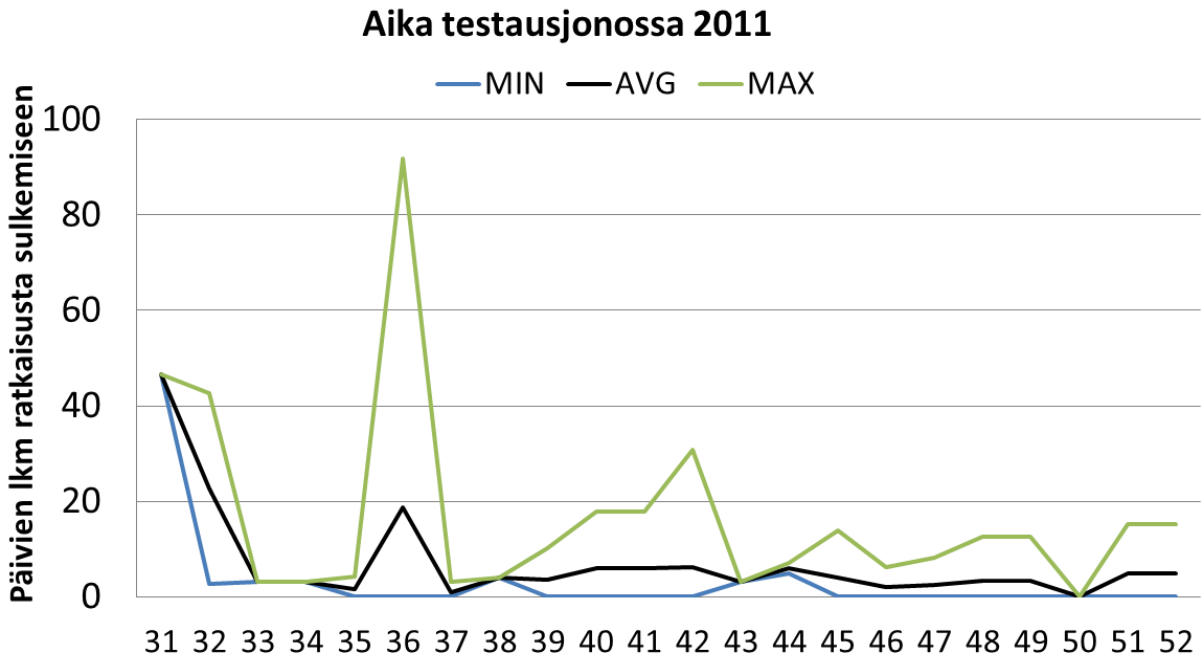
Mielenkiintoisena seikkana molempien vuosien tilastoissa se, kuinka valmiiden tehtävien ja uusien virheiden käyrät mukailevat monin osin toisiaan. Tämä johtunee siitä seikasta, että silloin kuin tehtäviä valmistuu, silloin on myös testattu. Testaus puolestaan on tärkein keino virheiden havaitsemiseen. Testauksella on tässä yhteydessä nimenomaan kyse manuaalisesta testauksesta ja verifioinnista. Vaikka automaattisella testauksellakin on merkittävä roolinsa laadunvarmistuksessa, on manuaaliseen testaukseen syytä panostaa laadunvarmistuksen kehittämisessä.

9.2 Testausaikamittauksen tulokset

Osana läpimenoaikamittausta haluttiin kehitysprojektin alkuvaiheessa saada pienennettyä sitä aikaa, jonka testattavat tehtävät odottavat testausta testausjonossa. Näin ollen ensin läpimenoaikaa laskettiin siitä hetkestä kun tehtävä on ratkaistu (resolved), siihen hetkeen kun se on valmistunut, eli suljettu (closed).

Testausaikamittauksessa päätettiin seurata minimi- maksimi- ja keskiarvoa sille, kuinka kauan tehtävät ovat testausjonossa odottaneet testausta. Minimi- ja maksimiarvojen osalta on huomioitava se, että niiden osalta on aina kyse yksittäisestä arvosta, eli vain yhden tehtävän testausjonossa oloaika. Yksittäiset minimi- ja maksimiarvot eivät ole kokonaisuuden kannalta tärkeitä, mutta antavat käsitystä siitä mikä on suurin vaihteluväli ajassa jona tehtävät ovat olleet testausjonossa kullakin viikolla. Keskiarvo on kokonaisuuden ja prosessin kokonaishallinnan kannalta merkittävämpi ja mielekkäämpi mittari, joka tasoittaa terävimmät kärjet sekä minimi- että maksimiarvojen suhteen. Pyrkimyksenä on pitää keskiarvo suhteellisen tasaisena viikosta toiseen.

Kuviossa 35 vuoden 2011 tilastot osoittavat huomattavaa vaihtelua, sekä äärimmäisen korkeita maksimiodotusaikoja. Vaikka otetaan huomioon, että odotusajasta ei ole poistettu viikonloppuja, on joidenkin tehtävien odotusaika ollut huomattavan pitkä. Vuoden 2011 osalta on myös havaittavissa se, että usean viikon kohdalla minimiarvo on 0 tai lähellä 0:aa. Usean viikon kohdalla onkin saatu testattua nopeasti yksittäisiä tehtäviä, jolloin ajaksi testausjonossa on tullut 0,1 päivää. Tilastokuvan suuresta skaalavaihtelusta johtuen tilastosta ei pysty erottamaan kyseisiä viikkoja niistä, joiden minimiarvona on 0. Kun minimiarvo on 0, se tarkoittaa käytännössä sitä, että tehtävää ei ole muistettu viedä ratkaista, jolloin tehtävän sulkemisen yhteydessä sekä ratkaisuhetki, että valmistumishetki saavat saman aikaleiman. Tällaiset unohdukset laskevat viikon keskiarvoa alaspäin ja todellisuudessa testausajan keskiarvo näiden viikkojen kohdalla olisi korkeammalla. Kyseisen kaltaisia viikkoja on kokonaisuuden kannalta hyvin vähän, mutta niiden olemassaolo on hyvä tiedostaa, sillä se osoittaa prosessissa olleen puutteita prosessin noudattamisessa tehtävien käsittelyn suhteen.

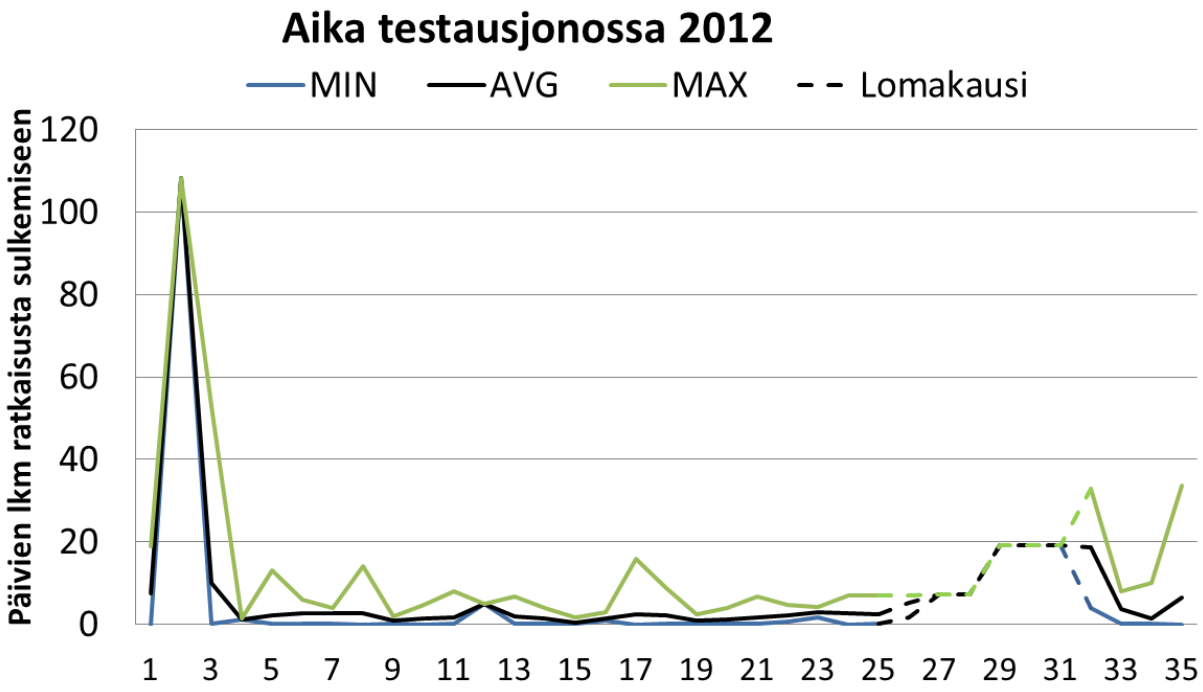


Kuvio 35. Tehtävien odotusaika testausjonossa vuonna 2011

Vuoden 2012 odotusaikatilastoissa on havaittavissa samankaltaisuuksia kuin projektitilastojen osalta. Vuoden 2012 alkupuolella on saatu purettua vuonna 2011 kertynyt testausjono, jolloin viimeiset korkeat piikit esiintyvät tilastoissa. Vaihtelua esiintyy vuoden 2012 tilastoissakin, mutta vaihtelu on pienempää, kuin vuoden 2011 tilastoissa. Näin

ollen voidaan sanoa, että testauksen osalta prosessi on ollut paremmin hallinnassa vuonna 2012, kuin mitä se on ollut vuonna 2011.

Lomakausi näkyy ensin nousuna ja sen jälkeen tasaisena muuttumattomana tilanteena. Muuttumattoman ajanjakson aikana tehtävät ovat odottaneet testausjonossa ja periaatteessa odotusajan tulisi kasvaa myös silloin. Aika testausjonossa lasketaan kuitenkin vain valmiille tehtäville. Mikäli testausjonossa on ollut tehtäviä työviikon päätteeksi, mutta yhtään ei ole testattu, ei tilanne ole kehittynyt tilastojen kannalta mihinkään suuntaan ja arvot pysyvät samoina. Merkittävää lomakauden osalta on kuitenkin se, että tilastot osoittavat olleen puutteita prosessin noudattamisessa lomakauden aikana. Vaikka lomakaudellakin on tiimissä ollut jatkuva miehitys, ei normaaleja käytäntöjä ole noudatettu testauksen suhteen ja aika testausjonossa on kasvanut selvästi lomakauden aikana. Vuoden 2012 odotusaikatilasto on esitetty kuviossa 36.



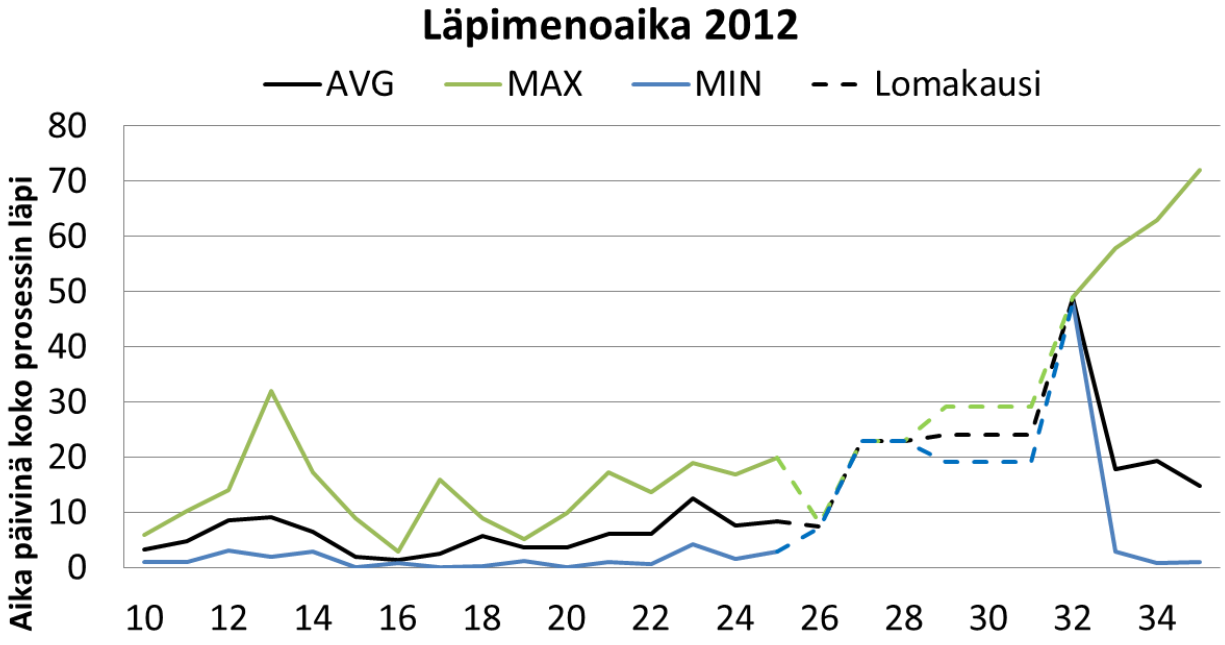
Kuvio 36. Tehtävien odotusaika testausjonossa vuonna 2012

9.3 Läpimenoaikamittauksen tulokset

Kehitysprojektin yhtenä tavoitteena oli saada tehtävien läpimenoaikaa pienennettyä. Luonnollisesti tämä tarkoitti sitä, että läpimenoaikaa oli alettava seuraamaan, koska ai-

kaisemmin tehtävien läpimenoaikoja ei seurattu. Läpimenoajan mittaus koko prosessin alusta loppuun aloitettiin vuoden 2012 viikolla 10.

Kuviossa 37 on esitetty tilasto tehtävien läpimenoajasta koko prosessin läpi, alkaen siitä hetkestä kun tehtävä on tullut työjonoon, aina siihen hetkeen kun tehtävä on saatu valmiiksi. Läpimenoaikamittauksessa minimi- ja maksimiarvot ovat vain yksittäisten tehtävien kokonaisläpimenoaikoja. Keskiarvossa on kohtalaista vaihtelua läpimenoajan ollessa 2 päivän ja 13 päivän välillä. Lomakausi esiintyy läpimenoajassakin selkeänä nousujohteisena ajanjaksona. Syy sille on se, että ennen lomakautta työjonoon on lisätty riittävä määrä tehtäviä kattamaan lomakauden työtarpeet. Siitä seuraa se, että osa tehtävistä odottaa jo jonossa hyvin pitkän aikaa ennen toteutusta, joka myös kasvattaa läpimenoaikaa. Tällainen menettely on vastoin pyrkimystä pitää läpimenoaika mahdollisimman pienenä. Lomakauden poikkeuksellisuuden huomioiden mm. resursoinnin osalta, on tämä ratkaisu tehtävien etukäteen jonoon laittamisesta tiimissä kuitenkin päätetty tehdä.



Kuvio 37. Tehtävien läpimenoaika vuonna 2012

9.4 Prosessin lopputilanteen arviointi

Validoinnin erityistavoitteet kuvauksineen on esitetty liitteessä 3 ja verifioinnin liitteessä 4, joiden perusteella arvioinnit on tehty.

9.4.1 Arviointi validoinnin erityistavoitteiden täyttymisestä

Tason 1 erityistavoitteiden arviointi

Tarkasteltaessa tuotteita jotka valitaan validoinnin piiriin (SP 1.1) tuotekehitystiimin kannalta, voidaan todeta että kaikki tuotteet ovat samassa asemassa. Näin ollen kaikki tuotteet myös kuuluvat validoinnin piiriin ja niiden työprosessissa noudatetaan validoinnin ohjeistuksia. Täysin samanlaisina validoinnin toimenpiteet eivät toistu kaikkien tuotteiden osalta.

Validointiympäristön valmistelu (SP 1.2) ei vaadi käytännössä nykyiseen ympäristöön muutoksia. Koontipalvelin on olemassa ja sitä käytetään edelleen ja hyödynnetään validoinnissa. Sen lisäksi validoinnissa käytetään eri työasemia ja palvelimia joilla voidaan validoida tuotteita erilaisissa toimintaympäristöissä.

Validoinnin kriteerit (SP 1.3) on määritelty tuotteesta riippuen osin samoin ja osin eri tavalla. Tuotteiden osalta vaihtelee mm. se kuinka kattavat julkaisutestit on tehtävä, sekä minkälaisia komponentteja ja tilanteita on testattava. Kaikkia tuotteita koskevat integraatiotestit ja yksikkötestit ovat osa validointia ja täyttävät myös osan validointikriteereistä tuotekehitystiimin vastuulla olevissa tuotteissa. Testit on usein johdettu oikeista käyttötapauksista, jolloin testin onnistunut suoritus vahvistaa toiminnallisuuden. Joillekin tuotteille on myös erillisiä dokumentoituja julkaisutestejä, jotka on suoritettava hyväksytysti ennen tuotejulkistusta.

Tason 2 erityistavoitteiden arviointi

Validointia suoritetaan (SP 2.1) tuotekehitystiimissä sekä automaattisesti että manuaalisesti. Tuotekehitystiimissä validoinnista on esimerkiksi integraatiotestien suorittamisesta seurauksena joko onnistunut suoritus, jolloin mitään toimenpiteitä ei tarvita. Mikäli suoritus sen sijaan on virheellinen, saadaan virheellisesti suoritetuista yksittäisistä testeistä raportti josta voidaan havaita mitkä testitapaukset eivät ole toimivia ja reagoida niihin asianmukaisesti. Muutoin esimerkiksi hyväksymistestauksen yhteydessä ei erillistä

raporttia generoida onnistuneista suorituksista. Mikäli virheitä tulee vastaan hyväksymistesteissä, tehdään niistä virheraportti joka sitten käsitellään prosessin mukaisesti. Tarvittaessa hyväksymistestit suoritetaan uudestaan. Jokaisen tehtävän yhteydessä suoritetaan validointi myös manuaalisen testauksen muodossa, jotta voidaan varmistua oikeanlaisesta toiminnallisuudesta.

Validoinnin tuloksia analysoidaan (SP 2.2) tuotekehitystiimissä ja esimerkiksi integraatiotestien suorittamista seurataan jatkuvasti sekä niihin liittyviin ongelmatilanteisiin reagoidaan saman tien, kun ne esiintyvät. Muutoin odottamattomat virhetilanteet käsitellään ryhmässä yhteisesti.

9.4.2 Arviointi verifioinnin erityistavoitteiden täyttymisestä

Tason 1 erityistavoitteiden arviointi

Tuotekehitystiimissä kaikki tuotteet ovat verifioinnin osalta samassa asemassa ja näin ollen ne kaikki kuuluvat verifioinnin (SP 1.1) piiriin. Kaikkien tuotteiden osalta verifiointi suoritetaan automaattisesti integraatio- ja yksikkötestien muodossa, että myös manuaalisesti verifioiden jokainen yksittäinen tehtävä.

Tuotekehitystiimissä on ympäristöt (SP 1.2) valmiina verifiointia varten. Automaattiset integraatiotestit suoritetaan omalla koontipalvelimellaan, jolla suoritetaan myös automaattiset yksikkötestit. Vertaisarvioinnit puolestaan suoritetaan tuotekehitystiimin työpisteillä.

Tuotekehitystiimissä verfioinnin kriteerit (SP 1.3) tulevat suurelta osin kattavista yksikkö- ja integraatiotesteistä. Yhtenä verifioinnin onnistumisen kriteerinä on niiden onnistunut suoritus. Muuten toiminnalliset vaatimukset tulevat määrityksistä joiden perusteella on varmistuttava että tuotteet toimivat kuten niiden on luvattu tai haluttu toimivan.

Tason 2 erityistavoitteiden arviointi

Tuotekehitystiimissä ei ole nähty tarpeelliseksi erikseen aikatauluttaa vertaisarviointeja. Vertaisarviointeja tehdään aina tarvittaessa ja yhtenä pääsääntöjä onkin, ettei sellaista ohjelmakoodia saa viedä versiohallintaan jota ei ole vertaisarvioitu. Projektin alkutilanteessa vertaisarvioinnin vastuu oli suureksi osaksi pääsuunnittelijan vastuulla, mutta projektin edetessä päätettiin vastuu jakaa koko ryhmälle. Tällöin vertaisarvioinnin hyväksymisen kriteerit saattoivat hieman vaihdella arvioijan mukaan, mihin asioihin kukin kiinnitti huomiota.

Kehitysprojektin alkuvaiheessa tuotekehitystiimillä oli määriteltynä kevyt lista ohjaamaan (SP 2.1) vertaisarviointeja. Listaa ei kuitenkaan missään vaiheessa ollut otettu käyttöön juuri ollenkaan edes kokeilumielessä. Projektin aikana tiimin vertaisarviointeja ohjaava muistilista katselmoitiin ja otettiin käytettäväksi vertaisarvioinneissa.

Tuotekehitystiimissä suoritettiin vertaisarviointia jo kehitysprojektin alkuvaiheessa. Koska vertaisarvioinnin ohjeistusta ei ollut käytössä, vertaisarviointi ja kontrolli (SP 2.2) mihin vertaisarvioinnissa kiinnitetään huomiota luonnollisesti arvioijan mukaan. Vertaisarvioinnissa esiin tulleet ongelmat ja muut asiat kirjaa jokainen ylös omatoimisesti tarvittaessa, sekä toimii niiden edellyttämällä tavalla.

Vertaisarviointien osalta päätettiin tiimissä kiinnittää huomiota siihen, minkälaisia asioita vertaisarvioinneissa nousee esiin, jotta voitaisiin huomata mistä suurimmat ongelmat vertaisarvioinnissa johtuvat. Tavoitteena on ajan myötä saada vähennettyä mahdollisia ohjelmakoodiin liittyviä ongelmakohtia jo ennen vertaisarvioinnin suorittamista. Näin voitaisiin myös ohjata kehitystyötä parempaan ja laadukkaampaan suuntaan, mikäli saadaan osa ongelmista eliminoitua kokonaan uusien käytäntöjen avulla. Näin myös voitaisiin saada vertaisarviointia nopeammaksi. Vertaisarvioinneissa esiin nousseita asioita päätettiin myös käsitellä (SP 2.3) yhdessä. Luontevin hetki vertaisarviointien käsittelylle oli tiimin retrospektiivi.

Tason 3 erityistavoitteiden arviointi

Verifioinnin toimenpiteet (SP 3.1) ovat tuotekehitystiimin päivittäisiä aktiviteettejä. Nämä aktiviteetit pitävät sisällään sekä automaattiset testit, manuaaliset testit että myös vertaisarvioinnit.

Verifioinnin tuloksia analysoidaan (SP 3.2) sen mukaisesti täyttyivätkö verifioinnin odotukset esimerkiksi hyväksymistestien tai suorituskykytestien osalta. Mikäli korjattavaa löytyi, ryhdytään tiimissä korjaavien toimenpiteiden suorittamiseen.

9.4.3 Arviointi yleisten tavoitteiden täyttymisestä

CMMI-DEV 1.3:ssa kaikkiin prosessialueisiin liittyvät olennaisesti yleiset tavoitteet, joiden tavoitteiden täyttymistä arvioidaan seuraavaksi. Yleiset tavoitteet on kuvattu liitteessä 2. Samassa liitteessä on myös kuvattu mitä kukin tavoite tarkoittaa validoinnin ja verifioinnin kannalta, mikäli niihin liittyy jotain erityistä.

Organisatorisen menettelytavan asettamisen (GP 2.1) osalta mainittakoon, että suurimmaksi osaksi vaatimukset verifioinnin ja validoinnin tuotoksille asettaa tuotekehitystiimi itse pääsuunnittelijan johdolla. Organisaation ylempi johto voi puuttua tilanteeseen, mikäli tarvetta ilmenee.

Prosessin suunnittelun (GP 2.2) osalta verifioinnin ja validoinnin toimenpiteitä suoritetaan jatkuvasti. Molempien prosessialueiden toiminnot ovat sisäänrakennettuja tuotekehitystiimin käytäntöihin, joiden avulla prosessialueiden tavoitteet tulevat täytetyiksi. Resursoinnin (GP 2.3) osalta voidaan todeta, että tiimillä on ollut ja on edelleen käytössään tarvittavat resurssit ja välineet prosessialueiden käytäntöjen suorittamista varten.

Kehitysprojektin alkuvaiheessa tiimin vastuut (GP 2.4) oli määritelty, mutta niihin tarvittiin muutoksia. Sen seurauksena vastuita ja rooleja määriteltiin uudestaan vastaamaan nykyisiin tarpeisiin. Koulutuksen (GP 2.5) osalta vaatimukset tulevat osin täytetyksi tiimin päiväpalaverien kautta. Päiväpalaverien aikana ei sen lyhyen keston vuoksi kuitenkaan ehditty käydä asioita läpi riittävällä tarkkuudella, jotta tietämys olisi levinnyt tiimissä. Sen vuoksi projektin yhteydessä aloitettiin milk&cookies-sessiot, joiden nimenomaisena tarkoituksena on kouluttaminen ja tiedon jakaminen.

Tuotosten hallinnan (GP 2.6) osalta ei tehdä eroa tuotekehitystiimin tekemien tuotteiden ja projektien suhteen. Kaikki projektit ja tuotteet ovat validoinnin ja verifioinnin piirissä. Tiimin käytännöt, kuten vertaisarvioinnin tarkistuslista, pitävät osaltaan huolen siitä että tuotokset täyttävät tiimin laadulliset kriteerit.

Asianosaisten osallistamisessa (GP 2.7) tuotekehitystiimin näkökulmasta asia on säilynyt käytännössä muuttumattomana. Tuotekehitystiimin ulkopuolisia asianosaisia ja sidosryhmiä ei validoinnin ja verifioinnin aktiviteetteihin liittyen useimmiten käytännössä ole. Käytännön tasolla molemmat tavoitteet tulevat täytetyiksi prosessien suorittamisella. Ketterien menetelmien mukaisesti loppukäyttäjät otetaan tarvittaessa mukaan prosessiin ja lopputulosten validointiin sekä verifiointiin.

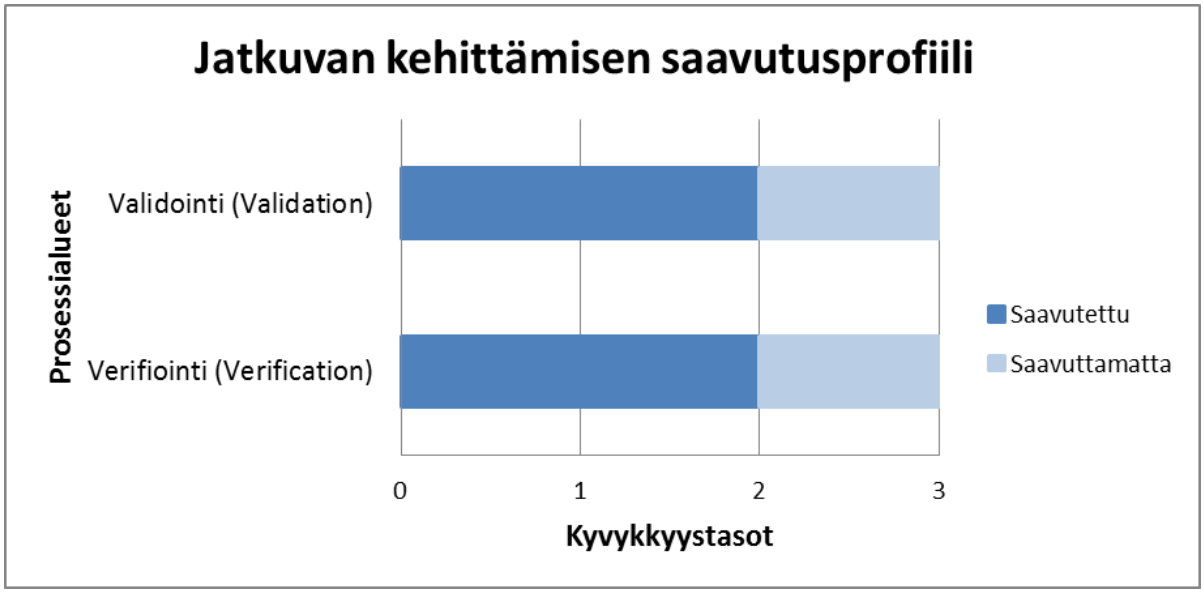
Prosessin hallinnoinnin ja valvonnassa (GP 2.8) voidaan todeta seurannan olevan riittävää siltä osin, että seurannalla voidaan havaita syntyneitä virhetilanteita sekä reagoimaan niihin. Tuotekehitystiimissä seurataan viikkotasolla uusien virheiden määrää, sekä myös korjattujen virheiden määrää.

Prosessin noudattamisen (GP 2.9) ja katselmoinnin (GP 2.10) osalta voidaan todeta, että tuotekehitystiimissä kaikki prosessiin ja siihen liittyvät muut dokumentit, kuten tilastotiedot, ovat organisaatiossa kaikille julkisia. Näin ollen kaikki joita prosessin suorituskyky tai prosessin noudattaminen kiinnostavat, saavat nämä tiedot halutessaan. Päivittäisellä tasolla tiimi itse huolehtii prosessin mukaisesta toiminnasta.

Prosessin määrittelyn (GP 3.1) lähtökohtana ovat organisaation standardit prosessit, joista on räätälöity jokaiselle prosessialueelle omat prosessinsa. Lähtökohta standardiprosessien kehittämiselle ja edelleen räätälöinnille on Organizational Process Definition (OPD)-prosessialueen-toteutus. Koska kyseisen prosessialueen kyvykkyyden kehittäminen ei kuulunut tämän projektin piiriin, ei käytännön vaatimusten voida katsoa täyttyneen. Kokemusten kerääminen prosesseista (GP 3.2) organisaatiotasolla prosessisuunnittelun tueksi organisaation muissa yksiköissä tai tasoilla ei tule toteutuneeksi tuotekehitystiimin käytännöissä. Näin ollen myös tämän käytännön vaatimukset jäävät täyttymättä.

9.4.4 Yhteenveto prosessin arvioinnista

Validoinnin kyvykkyystaso oli kehitysprojektin lähtötilanteessa tasolla 1 ja verifioinnin tasolla 0. Muutosten myötä sekä validoinnin, että verifioinnin kyvykkyystasot nousevat tasolle 2. Kyvykkyystaso 3 jää molempien prosessialueiden osalta saavuttamatta, koska yleisten käytäntöjen 3. tason vaatimuksia ja käytäntöjä ei pystytä täyttämään. Yleisten vaatimusten 3. tasolla edellytettäisiin koko organisaation kattavia käytäntöjä joita hyödynnettäisiin myös validoinnin ja verifioinninkin osalta. Koko organisaation kattava toiminta oli kuitenkin rajattu tämän kehitysprojektin ulkopuolelle, eikä 3. tason saavuttaminen ollut siten tavoiteltava tilanne.



Kuvio 38. Valittujen prosessialueiden saavutusprofiili

Taulukoissa 8, 9 ja 10 on vertailtu mikä oli yleisten tavoitteiden ja kehitettyjen prosessialueiden tavoitteiden tilanne kehitysprojektin alku- ja loppuvaiheessa. Taulukoista käy ilmi, että moni tavoite oli itse asiassa jo itsearvioinnin näkökulmasta täytetty kehitysprojektin alkuvaiheessa.

Taulukko 8. Vertailu CMMI-DEV yleistavoitteiden täyttymisestä.

Yleistavoitteiden saavuttaminen validoinnin ja verifioinnin prosessialueiden osalta			Projektin alku	Projektin loppu
GG 1	Saavuta erityiskäytäntöjen tavoitteet			
	GP 1.1	Suorita erityiskäytännöt	-	+
GG 2	Vakiinnuta hallittu prosessi			
	GP 2.1	Aseta organisatorinen menettelytapa	+	+
	GP 2.2	Suunnittele prosessi	+	+
	GP 2.3	Tarjoa resurssit	+	+
	GP 2.4	Osoita vastuut	-	+
	GP 2.5	Kouluta henkilöstö	-	+
	GP 2.6	Hallitse tuotoksia	+	+
	GP 2.7	Tunnista ja osallista asianosaiset	+	+
	GP 2.8	Valvo ja hallitse prosessia	+	+
	GP 2.9	Arvioi prosessin noudattamista	+	+
	GP 2.10	Katselmoi tilanne korkeamman johdon kanssa	+	+
GG 3	Vakiinnuta määritelty prosessi			
	GP 3.1	Määrittele prosessi	-	-
	GP 3.2	Kerää kokemuksia prosessista	-	-
Tavoite saavutettu (+) **Tavoite saavuttamatta (-)**				

Taulukko 9. Vertailu CMMI-DEV verifioinnin erityistavoitteiden täyttymisestä.

Erityistavoitteiden saavuttaminen verifioinnin osalta			Projektin alku	Projektin loppu
SG 1	Valmistaudu verifiointiin			
	SP 1.1	Valitse verifioitavat tuotteet	+	+
	SP 1.2	Valmistele verifiointiympäristö	+	+
	SP 1.3	Määrittele verifioinnin kriteerit	+	+
SG 2	Suorita vertaisarviointeja			
	SP 2.1	Valmistaudu vertaisarviointeihin	-	+
	SP 2.2	Suorita vertaisarvioinnit	+	+
	SP 2.3	Analysoi vertaisarvioinnin tuloksia	-	+
SG 3	Verifioi valitut tuotteet			
	SP 3.1	Suorita verifiointi	+	+
	SP 3.2	Analysoi verifioinnin tulokset	+	+
Tavoite saavutettu (+) **Tavoite saavuttamatta (-)**				

Taulukko 10. Vertailu CMMI-DEV validoinnin erityistavoitteiden täyttymisestä.

Erityistavoitteiden saavuttaminen validoinnin osalta			**Projektin alku**	**Projektin loppu**
SG 1	Valmistaudu validointiin			
	SP 1.1	Valitse validoitavat tuotteet	+	+
	SP 1.2	Valmistele validointiympäristö	+	+
	SP 1.3	Määrittele validoinnin kriteerit	+	+
SG 2	Suorita validointi tuotteille tai komponenteille			
	SP 2.1	Suorita validointi	+	+
	SP 2.2	Analysoi validoinnin tulokset	+	+
Tavoite saavutettu (+) **Tavoite saavuttamatta (-)**				

10 Yhteenveto

10.1 Tulosten arviointia

Kehitysprojektin tavoitteena oli löytää kehitysideoita paikalleen pysähtyneeseen prosessiin. Lisäksi tavoitteena oli löytää keinoja kehittää ohjelmistojen laadunvarmistusta sekä prosessien, että muiden käytäntöjen avulla. Kehittymisen aste verrattuna alkutilanteeseen oli myös pystyttävä arvioimaan kehitysprojektissa.

Tutkimuksen ensimmäisenä tavoitteena oli löytää vastaus sille mitkä ovat kehitysprojektin alkutilanteessa tuotekehitysprosessin ongelmakohdat. Prosessissa havaittiin olevan sekä konkreettisia ongelmakohtia, että myös kehityskohteita. Projektin alkutilanne sekä sen ongelmakohdat on kuvattu luvussa 7. Ensimmäiseen tutkimuskysymykseen löydettiin siis vastaus.

Kehitysprojektin toisena tavoitteena oli selvittää tuotekehitysprosessin kehittämismahdollisuuksia sekä keinoja arvioida sen suorituskykyä. Teoriataustan avulla prosessissa päätettiin hyödyntää tuotekehitystiimissä jo käytössä olevien menetelmien ns. ”käyttämättömiä” osa-alueita. Käytössä olevien menetelmien käyttöä siis syvennettiin. Lisäksi prosessia päätettiin kehittää CMMI-DEV:in ohjeistuksen mukaisesti. CMMI-DEV tarjosi itsessään jo mittarin kyvykkyyden arviointiin ja tuotekehitysprosessiin tuotiin lisäksi mittareita muista käytössä olevista menetelmistä, sekä keksittiin havaintojen pohjalta myös itse uusia.

CMMI-DEV osoittautui hyväksi suuntaviivoja antavaksi viitekehykseksi, kun prosessien kehittämistä tarkasteltiin tuotekehitystiimissä kyvykkyyden kannalta. Vaikkei CMMI-DEV asettanutkaan tarkkoja ehtoja sen suhteen miten asiat tulisi organisaatioissa toteuttaa, ovat useat mallin esimerkit ja ehdotukset hyvin käytännöllisiä. CMMI-DEV:in suunnittelussa on selvästi otettu vaikutteita ohjelmistokehitystyön parhaista käytännöistä ja se näkyy mallissa hyvällä tavalla. Tämä on havaittavissa tutkimalla ohjelmistokehitykseen liittyviä prosessialueita ja niiden tavoitteita sekä niihin liittyviä esimerkkejä CMMI-DEV:in dokumentaatiosta.

Tämän kehitysprojektin puitteissa päätettiin CMMI-DEV:iä hyödyntää nimenomaan kyvykkyyden kehittämisen mukaisesti. Valinta osoittautui oikeaksi. Projektin hyvin rajallisilla resursseilla ei olisi ollut mitään mahdollisuuksia lähteä kehittämään kypsyyttä. Kypsyyden kehittäminen olisi jo alkutilanteessa vaatinut huomattavasti enemmän resursseja. Tuotekehitystiimin osalta olisi luultavasti pärjätty käytössä olleilla resursseilla, mutta kypsyyden kehittäminen olisi vaatinut runsaasti resursseja myös muualta organisaatiosta. Tällöin vaarana olisi myös ollut, että varsinainen projektin tavoitteena ollut kehitystyö olisi jäänyt tekemättä ja tuloksena olisi voinut olla vain organisaation prosessien ja kypsyystasojen kartoitus, sekä kehitysmahdollisuudet.

Tarkasteltaessa tuotekehitystiimin projektitilastoja laadunvarmistuksen kannalta ei tilastoista käy täysin suoraan ilmi laadunvarmistuksen tehokkuus. Tilastoissa ei eritellä niitä virheitä, jotka on havaittu tuotekehitystiimin omassa laadunvarmistuksessa, niistä virheistä jotka ovat esiintyneet asiakasympäristöissä. Virheiden löytäminen toisaalta kertoo sen, että laadunvarmistuksen toimenpiteitä on tehty jossain vaiheessa, mutta tilastot eivät kerro missä vaiheessa. Mittareiden valossa laadunvarmistukselliset toimenpiteet voivat olla aivan riittäviä sen suhteen, ettei virheitä esiinny asiakasympäristöissä merkittävissä määrin. Tilanne voi olla myös täysin päinvastainen. Mittaamalla virheiden lukumäärää nykyisellä tavalla ei voida esittää suoraa syy-seuraus-suhdetta laadunvarmistuksen toimenpiteiden tehokkuudesta.

Lähtökohtaisena ajatuksena kehitysprojektin alussa oli saada uusien virheiden lukumäärää pienennettyä. Pelkästään tavoite uusien virheiden pieneen määrään on tarkoitukseltaan hieman kyseenalainen, sillä päämäärään päästään helposti suorittamalla laadunvarmistus puutteellisesti. Virheiden löytyminen itsessään ei ole välttämättä huono tai välteltävä asia, vaan huomiota olisi kiinnitettävä yllä mainitun kaltaisella tavalla siihen missä kehitysvaiheessa ja missä ympäristössä virhe on esiintynyt.

Löydettyjen virheiden lisäksi tilastoja kerätään tuotekehitystiimissä myös korjattujen virheiden lukumäärästä. Nykyisellään korjattujen virheiden lukumäärä ei suoraan kerro virheiden korjauksen tehokkuudesta (Schulmeyer 2008, 395-396.), sillä niiden osalta on kerätty yhteen kaikki eri projekteissa ja eri kehitysvaiheissa korjatut virheet. Tilanne on

niiden osalta siis sama, kuin uusien virheiden osalta, eli korjattu virhe on voinut koskea joko kehitysvaiheessa olevaa tuotetta tai tuotantoympäristössä olevaa.

Kanbanissa käytetty (Kniberg & Skarin 2010, 3-5.) läpimenoaikamittaus herätti myös ajatuksia. Tehtävien läpimenoaikaan voivat vaikuttaa useat eri asiat. Läpimenoajan kannalta eri tavalla prosessin läpi suoriutuvat esimerkiksi virheet ja kokonaan uudet ominaisuudet. Virheiden osalta on usein tarkkaan tiedossa, kuinka virhe tapahtuu ja siten myös mihin muutos tulee tehdä, jotta virhe saadaan korjatuksi. Käytännössä korjaus voi tapahtua hyvinkin nopeasti. Uudet ominaisuudet sen sijaan voivat viedä huomattavan paljon kauemmin aikaa kulkiessaan prosessin läpi, varsinkin mikäli alkutilanteessa määritykset ovat ketterien menetelmien mukaisesti väljiä ja tarkentuvat kehityksen edetessä. Läpimenoaikamittaus soveltuu luultavasti hyvin valmistavan teollisuuden prosesseihin, joissa prosessin toiminnot toistuvat kerta toisensa jälkeen täsmälleen samanlaisina ja tuottavat täsmälleen samanlaisia tuotoksia. Ohjelmistokehityksessä asia ei välttämättä ole aivan näin yksinkertainen yllä mainituista seikoista johtuen. Kun erilaisilla tehtävillä on hyvinkin eri pituisia läpimenoaikoja, ei keskiarvoa voida välttämättä pitää kovin hyvänä ohjenuorana tai vertailukohtana. Yhtenä vaihtoehtona voisi olla laskea erikseen läpimenoaikaa erityyppisille tehtäville. Suuret erot erityyppisten tehtävien läpimenoajoissa voivat toisaalta myös kertoa siitä, että tehtäviä ei ole pilkottu riittävän pieniksi osakokonaisuuksiksi, jotta ne olisivat läpimenoaikamittauksessa vertailukelpoisia. Tulevaisuudessa olisi siis kiinnitettävä aikaisempaa enemmän huomiota tehtävien pilkkomiseen sopivan kokoisiksi kokonaisuuksiksi.

Mittarien osalta testaamattomien tehtävien seuranta sen sijaan osoittaa tehostusta tapahtuneen laadunvarmistuksen osalta. Mittari osoittaa testausta tehdyn aktiivisemmin kuin ennen. Testaamattomien tehtävien osalta käy ilmi se, että muutos tapahtui lopulta hyvin pienillä toimenpiteillä. Käytännössä tiimissä sovittiin, että tehtäviä on testattava mikäli mahdollista ja tämä myös kirjoitettiin kaikkien nähtävillä olevalle Kanbantaululle. Testauksen odotusajan mittari osoittaa saman asian odotusajanpienenemisen muodossa.

Toiseenkin tutkimuskysymykseen löydettiin vastaus. Keinoja prosessin ja laadunvarmistuksen kehittämiseen ja suorituskyvyn mittaamisen löydettiin, vaikka mittarit kaipaavat vielä kehittämistä.

Vaikka projektin aikataulu olikin kohtuullisen pitkä, on silti huomattava että muutosten vaikutuksia ei välttämättä havaita välittömästi muutosten jalkautusten jälkeen. Kuten luvun 9 tuloksista käy ilmi, niin vakiintumista prosessissa on havaittavissa vasta n. 13 (esimerkiksi valmiiden tehtävien tilasto) viikon kuluttua siitä, kun muutoksia on prosessiin otettu käyttöön. Kokonaisuudessaankin tilastoja vertailemalla monin osin vaihteluvälit ovat vuonna 2012 pienempiä, kuin vuonna 2011. Tilastojen valossa prosessi on siis saatu jossain määrin paremmin hallintaan vuoden 2012 aikana, vaikka kehittämistä vielä löytyykin.

10.2 Työprosessin arviointia

Projektin ympäristö ja tavoitteet pysyivät monilta osin samoina, kuin mitä ne olivat silloin, kun kehitysprojektia ensimmäisiä kertoja ideoitiin. Luonnollisesti tarkempi fokus ja kehityssuunta kuitenkin tarkentuivat projektin edetessä. Kehitysprojekti jatkui alkuvaiheen ideointivaiheen jälkeen teoriataustan keräämisellä. Teorian keräämisessä kävi ilmi, että ohjelmistojen ja tuotekehitysprosessien kehittämisestä on lukuisia artikkeleita ja malleja, joiden mukaan kehitystyötä voidaan lähestyä ja kehitystavoitteita saavuttaa. Tarjonnan paljous olikin projektin alkuvaiheissa huomattava ongelma siinä mielessä, että kuinka löytää sopivin malli tämän kehitysprojektin tarkoituksiin. Teoriataustaa lähdettiinkin aluksi keräämään aivan liian laajalla skaalalla. Alkuperäisenä tavoitteena oli kerätä tietoa monista eri menetelmistä ohjelmistokehityksen prosessien kehittämiseen ja sen jälkeen yhdistellä niistä kohdeympäristön kannalta parhaat ideat. Kaikkien eri mallien hyvien puolien ja artikkeleiden ideoiden yhdisteleminen olisi lopulta ollut hyvin hankalaa. Samalla olisi luultavasti kadonnut täysin ajatus siitä, minkä mukaan prosessia kehitetään, jos jokainen eri prosessin osa on lopulta peräisin aivan eri ajatuksesta.

Projektin aikataulu asetettiin lähtökohtaisesti hyvin väljäksi, jotta mahdolliset yllättävät aikataulumuutokset eivät sotkisi projektin lopullista aikataulua ja saattaisi projektin valmistumista vaaraan. Ratkaisu osoittautui siltä osin oikeaksi, että odottamattomia kii-

reitä ilmaantui sekä muiden opintojen muodossa, että myös työkiireiden muodossa. Kiireitä oli kuitenkin enemmän, kuin mitä oli osattu odottaa. Projektin aikataulu ja työmäärä oli alun perin ajateltu jakautuvan tasaisesti koko projektin ajalle, mutta niin ei lopulta käynyt. Odottamattomat kiireet aiheuttivat sen, että työmäärä jakautui lopulta hyvin epätasaisesti eri kuukausille. Tämä myös johti siihen, että osa projektin välituloksista ei valmistunut ajallaan ja projektille alkuvaiheessa aikataulutettuja seurantakokouksia peruttiin. Tarpeen tullen kommunikointi, niin viivästyksistä kuin muutoinkin, hoidettiin sähköpostitse.

Tutkijan roolissa ollut opiskelija työskenteli koko kehitysprojektin ajan kehitysprojektin kohteena olevan tiimin jäsenenä. Tämän seurauksena useinkin projektin tilanne ja projektissa toteutetut muutokset olivat usein hyvin kyseisen opiskelijan sekä tiimin tiedossa. Tällöin jossain määrin unohtui aktiivinen tiedottaminen ja kommunikointi oppilaitoksen ohjaajan kanssa. Vaikka ideoista ja muutoksista ohjaajalle tiedotettiinkin, olisi yhteydenpidon tullut olla oppilaitoksen suuntaan huomattavasti aktiivisempaa.

10.3 Jatkokehitysehdotukset

Tämän projektin yhteydessä hyödynnettyjen CMMI-DEV:in prosessialueista löydettyjen hyvien kokemusten vuoksi, voi toiminnan kehittämisen kannalta olla hyödyllistä tarkastella myös muita prosessialueita. Ohjelmistokehitystä tekevässä tiimissä kannattanee ensin tarkastella myös muita ohjelmistokehityskategorian prosessialueita uusien kehitysideoiden saamiseksi. Laadunvarmistuksen kannalta asiaa mietittäessä, voisi muista prosessialueista löytyä toiminnan kehittämisen ideoita, joilla nimenomaan saadaan tuotteiden laatua parannettua. Laadunvarmistuksessa ei kuitenkaan lopulta ole kyse vain testauksesta, vaan myös siitä kuinka ohjelmistot rakennetaan. Kehitystyöhön liittyviin toimintoihin CMMI-DEV:ista löytyy hyvin todennäköisesti hyviä ideoita. Mikäli muiden prosessialueiden osalta halutaan myös tehdä kyvykkyysarviointia, on oletettavaa, että osa CMMI-DEV:in kyvykkyys- ja kypsyystasojen vaatimuksista täyttyvät jo nykyisellään kohdeorganisaation tuotekehitystiimissä käytössä olevilla menetelmillä, kuten asia oli validoinnin ja verifioinninkin suhteen.

Vaikkei jatkossa päätettäisikään kehittää toimintaa jonkin tietyn CMMI-DEV -kyvykkyystason saavuttamiseksi, voivat kyvykkyystason kaltaiset tavoitteet kuitenkin auttaa kehittämisen päämäärien saavuttamisessa, kun tavoite on jollakin tavalla konkretisoitu. Mikäli CMMI-DEV:iä ei päätettäisikään jatkossa hyödyntää kohdeorganisaatiossa, on tulevaisuuden kehitystarpeita mietittäessä pohdittava käytettävä viitekehys sen mukaisesti, mikä soveltuu laajasti käytettäväksi korkeamman tason viitekehysten, kuten Cobit:in kanssa.

Mittarit kaipaavat vielä kehittämistä, jotta mittaustulosten perusteella voidaan esittää selviä syy-seuraus-suhteita. Erottamalla mittauksessa virheiden määrät esimerkiksi sisäisessä laadunvarmistuksessa havaittuihin virheisiin ja asiakasympäristössä esiin tulleisiin virheisiin, saataisiin tarkempi käsitys sisäisten laadunvarmistuksen toimenpiteiden riittävyydestä. Myös korjattujen virheiden osalta voitaisiin eritellä virheet sen mukaisesti mitä tuotetta ne koskevat ja missä vaiheessa virheet on korjattu.

Mittaustulosten perusteella voidaan myös tulevaisuutta varten asettaa raja-arvot joiden sisällä mittaustulosten tulee pysyä. Kuten kehitysprojektin mittaustuloksista käy ilmi, niin vaihtelua esiintyy luonnollisesti mittareissa. Merkittävää vaihteluissa on se kuinka paljon sitä esiintyy. Sen vuoksi raja-arvojen asettaminen auttaisi keskittymään kehitystyössä olennaisiin asioihin, mikäli mittaustulokset ovat sallituissa rajoissa. Raja-arvojen ylittyessä tulisi reagoida asiaan kuuluvalla tavalla ja arvioida syitä poikkeuksiin ja keinoja niiden ratkaisuun sekä ehkäisyyn. Raja-arvojen asettamisessa tulisi kuitenkin huomioida erilaiset poikkeustilanteet, kuten esimerkiksi puutteet käytettävissä olevissa resursseissa esimerkiksi lomakausien aikana. Lomakausien osalta on tulevaisuudessa kiinnitettävä myös huomiota siihen, että normaaleissa prosessien mukaisissa toiminnoissa ja tehtävissä pitäydytään myös lomakausien aikana. Lomakaudet eivät saisi aiheuttaa poikkeuksia normaaliin prosessiin.

Yhtenä keinona hallita prosessia tilastojen kannalta olisi pohtia jonkinlaisen fokus-kertoimen (Kniberg 2007, 26-27.) käyttöä. Fokus-kerroin voisi auttaa suorituskyvyn ja raja-arvojen asettamisessa sekä normaalitilanteessa, että myös poikkeustilanteissa. Fokus-kertoimen tulisikin olla nimenomaan olosuhteisiin suhteutettu indikaattori mikä on suorituskyvyn oletettu suunta kaikki tarpeelliset seikat huomioiden. Tekemällä tilastois-

ta fokus-kerroin –korjattuja, saadaan esimerkiksi lomakausien tilastoista täysin vertailukelpoista dataa normaalitilanteeseen verrattuna.

Lähteet

Aaltola, Juhani & Valli, Raine (toim.). 2001. Ikkunoita tutkimusmetodeihin I. Metodin valinta ja aineiston keruu: virikkeitä aloittelevalle tutkijalle. PS-kustannus. Gummerus kirjapaino Oy. Jyväskylä.

Arikoski, Juha & Sallinen, Mikael. 2011. Vastarinnasta vastarannalle – johda muutos taitavasti. 1.-2. painos. Johtamistaidon opisto JTO. Keuruu. ISBN 978-951-802-766-2.

Baker, Steven W. 2006. Formalizing Agility, Part 2: How an Agile Organization Embraced CMMI. IEEE Computer Society. s. 147-154.

Benefield, Gabrielle & Deemer, Pete & Larman, Craig & Vodde, Bas. 2010. The Scrum Primer. Version 1.2.

Bundschuh, Manfred & Ebert, Christof & Dumke, Reiner & Schmietendorf, Andreas. 2005. Best Practices in Software Measurement. How to user metrics to improve project and process performance. Springer-Verlag Berling. ISBN 3-540-20867-4.

Chrissis, Mary Beth & Konrad, Mike & Shrum, Sandy. 2011. CMMI for Development. Guidelines for Process Integration and Product Improvement. Third Edition. Pearson Education, Inc. Boston. ISBN 978-0-321-71150-2.

Dumke, Reiner R & Kunz, Martin & Schmietendorf, Andreas. 2008. How to Measure Agile Software Development. Software Process and Product Measurement. International Conference, IWSM-Mensura 2007. Palma de Mallorca, Spain, November 5-8, 2007 Revised Papers. ISSN 0302-9743. s. 95-101.

Eickmann, Marion. 2008. Keep It Lean And Agile ("KILAA"). http://www.agile42.com/en/blog/2008/12/10/php-magazine-article/. Luettu 21.9.2012.

Ergin, Lemi Orhan. 2011. Do we need Technical Leads in Scrum? http://www.flyingtomoon.com/2011/06/do-we-need-technical-leads-in-scrum.html. Luettu 28.11.2011.

Fantina, Robert. 2005. Practical Software Process Improvement. Artech House, Inc. Norwood. ISBN 1-58053-959-9.

Foegen, Malte. 2010. Scrum and CMMI – Does it fit together? http://www.wibas.com/e20/e9138/e9324/e9344/e9348/downloads9349/wibas_Scrum_CMMI_en.pdf. Luettu 11.6.2012.

Forss, Anna. 2009. Scrum Roles – an Usolvable Puzzle? Methods and Tools. ISSN 1661-402X. Summer 2009. Volume 7 – number 2. s. 32 - 47.

Frends Technology. 2011 a. Circulatory testing duty. Intranet. Teams. Iss. R&D. Process guidelines. Luettu 17.6.2011.

Frends Technology. 2011 b. Daily meeting. Intranet. Teams. Iss. R&D. Process guidelines. Luettu 17.6.2011.

Frends Technology. 2011 c. How the day-to-day-process works. Intranet. Teams. Iss. R&D. Process guidelines. Luettu 17.6.2011.

Frends Technology. 2011 d. Status and planning meeting each Monday. Intranet. Teams. Iss. R&D. Process guidelines. Luettu 17.6.2011.

Frends Technology. 2011 e. Testing Process. Intranet. Teams. Iss. R&D. Process guidelines. Luettu 17.6.2011.

Fritzsche, Martin & Keil, Patrick. 2007. Agile Methods and CMMI: Compatibility or Conflict. e-Informatica Software Engineering Journal. Volume 1, Issue 1, 2007.

Galin, Daniel. 2004. Software Quality Assurance. From theory to implementation. Pearson Education Limited. Essex. ISBN 0201 70945 7.

Glazer, Hillel & Dalton, Jeff & Anderson, David & Konrad, Mike & Shrum, Sandy. 2008. CMMI or Agile: Why Not Embrace Both! Carnegie Mellon University.

Hertneck, Christian. 2009. Lean Thinking with CMMI. http://www.sei.cmu.edu/library/assets/presentations/1480_hertneck.pdf. Luettu 19.6.2012.

Jones, Capers. 2010. Software Engineering Best Practices. Lessons from Successful Projects in the Top Companies. McGraw-Hill. New York. ISBN 978-0-07-162162-5.

Kniberg, Henrik & Skarin, Mattias. 2010. Kanban and Scrum –making the most of both. InfoQ Enterprise Software Development Series. C4Media. United States of America. ISBN 978-0-557-13832-6.

Kniberg, Henrik. 2007. Scrum and XP from the Trenches. How we do Scrum. InfoQ Enterprise Software Development Series. C4Media. United States of America. ISBN 978-1-4303-2264-1.

Kroll, Per & Kruchten Philippe. 2003. The rational unified process made easy: a practitioner's guide to RUP. Pearson Education, Inc. Boston. ISBN 0-321-16609-4.

Kruchten, Philippe. 2001. The rational unified process: An introduction. Second Edition. Pearson Education, Inc. Boston. ISBN 0-201-70710-1.

Laamanen, Kai & Tinnilä, Markku. 2009. Prosessijohtamisen käsitteet. 4. uudistettu painos. Teknologiatelollisuus Oy. Espoo. ISBN 978-952-238-000-5.

Larman, Craig & Vodde, Bas. 2009. Lean Primer. Version 1.5. http://www.leanprimer.com/downloads/lean_primer.pdf. Luettu 29.5.2012.

Lazic, Ljubomir & Kolašinac, Amel & Avdic, Dzenan. 2009. The Software Quality Economics Model for Software Project Optimization. WSEAS TRANSCATIONS on COMPUTERS. Issue 1. Volume 8. s. 21-47.

Lekman, Lare. 2009. Mikä Ihmeen Kanban? http://larelekman.com/2009/09/26/mika-ihmeen-kanban/. Luettu 13.6.2011.

Milunsky, Jack. 2009. The Significance of Story points. http://agilesoftwaredevelopment.com/blog/jackmilunsky/significance-story-points. Luettu 14.8.2011.

Murphy, Craig. 2004. Adaptive Project Management Using Scrum. Methods And Tools. Winter 2004. Volume 12 – number 4. ISSN 1023-4918.

Pikkarainen, Minna & Mäntyniemi, Annukka. 2006. An Approach for Using CMMI in Agile Software Development Assessments: Experiences from Three Case Studies. SPICE 2006 Conference, Luxemburg.

Poppendieck, Mary & Tom. 2010a. Lean Software Development. An Agile Toolkit. 15th Printing. Pearson Education, Inc. Boston. ISBN 0-321-43738-1.

Poppendieck, Mary & Tom. 2010b. Leading lean software development: results are not the point. 2nd Printing. Pearson Education, Inc. Boston. ISBN 0-321-62070-4.

Potter, Neil & Sakry Mary. 2011. Implementing Scrum (Agile) and CMMI Together. http://www.scrumalliance.org/articles/334-implementing-scrum-agile-and-cmmi-together. Luettu 10.6.2012.

Ries, Eric. 2011. The Lean Startup. How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business. Random House, Inc. New York. ISBN 978-0-307-88789-4.

Saaranen-Kauppinen, Anita & Puusniekka, Anna. 2009. Menetelmäopetuksen tietovaranto KvaliMOTV. Kvalitatiivisten menetelmien verkko-oppikirja. Yhteiskuntatieteellinen tietoarkisto. Tampere.

Salo, Outi & Abrahamsson, Pekka. 2007. An Iterative Improvement Process For Agile Software Development. Software Process: Improvement and Practice, Vol. 12. s. 81-100.

Schulmeyer, G. Gordon (editor). 2008. Handbook of software quality assurance. Fourth edition. Artech house, Inc. Norwood. ISBN-13:978-1-59693-186-2.

Schwaber, Ken & Sutherland, Jeff. 2011. The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game. http://www.scrum.org/storage/scrumguides/Scrum_Guide%202011.pdf. Luettu 4.6.2012.

Scrum Alliance. 2011. Scrum Is An Innovative Approach To Getting Work Done. http://www.scrumalliance.org/pages/what_is_scrum. Luettu 13.6.2011.

Shelton, Cindy. 2008. Agile and CMMI: Better Together. http://www.scrumalliance.org/articles/100-agile-and-cmmi-better-together. Luettu 10.6.2012.

Singleton, Andy. 2010. Agile, the next generation: Three ways to go beyond Scrum. http://blog.assembla.com/assemblablog/tabid/12618/bid/14064/Agile-the-next-generation-Three-ways-to-go-beyond-Scrum.aspx. Luettu 28.11.2011.

Sutherland, Jeff & Jakobsen, Carsten Ruseng & Johnson, Kent. 2007. Scrum and CMMI Level 5: The Magic Potion For Code Warriors. Agile 2007. IEEE Computer Society. ISSN 0-7695-2872-4/07. s. 272-278.

Software Engineering Institute 2012a. CMMI Compatibility. http://www.sei.cmu.edu/cmmi/compatibility/. Luettu 10.6.2012.

Software Engineering Institute 2012b. CMMI Appraisal Classes. http://www.sei.cmu.edu/cmmi/solutions/appraisals/classes.cfm. Luettu 2.9.2012.

Software Engineering Institute. 2010. CMMI for Development, Version 1.3. CMMI-DEV, V1.3. http://www.sei.cmu.edu/reports/10tr033.pdf. Luettu: 7.1.2012.

Sommerville, Ian. 2006. Software Engineering. Eighth Edition. Pearson Education, Inc. Boston. ISBN 7-111-19770-4.

Srinivasan, Vibhu. 2007a. What is a story point? http://agilefaq.wordpress.com/2007/11/13/what-is-a-story-point/. Luettu 14.8.2011.

Srinivasan, Vibhu. 2007b. What is velocity in a scrum team. http://agilefaq.wordpress.com/2007/11/03/what-is-velocity-in-a-scrum-team/. Luettu 14.8.2011.

The Agile Management Company. 2012. Velocity. http://www.versionone.com/Agile101/Agile-Scrum-Velocity/. Luettu 8.1.2012.

Toikko, Timo & Rantanen, Teemu. 2009 Tutkimuksellinen kehittämistoiminta. Näkökulmia kehittämisprosessiin osallistamiseen ja tiedontuotantoon. Tampereen Yliopistopaino Oy. Tampere. ISBN 978-951-44-7732-4.

Trindale, Francisco. 2009. What's the Tech Lead doing anyway? http://blog.franktrindade.com/2009/08/11/whats-the-tech-lead-doing-anyway/. Luettu 28.11.2011.

Yhteiskuntatieteellinen tietoarkisto. 2011. KvantiMOTV - Menetelmäopetuksen tietovaranto. http://www.fsd.uta.fi/menetelmaopetus. Päivitetty 28.4.2011. Luettu 17.9.2012.

Liitteet

Liite 1. CMMI-DEV 1.3 Prosessialueet ja tavoitteet

Prosessien hallinta (Process Management)

Kategoria pitää sisällään organisaation yleiset aktiviteetit liittyen prosessien määrittelyyn, suunnitteluun, suorittamiseen, mittaukseen, arviointiin ja prosessien kehittämiseen liittyen. Prosessialueiden kehittäminen mahdollistaa organisaation dokumentointikäytäntöjen yhdenmukaistamisen, parhaiden käytäntöjen jakamisen ja tiedon jakamisen koko organisaatiossa. (Software Engineering Institute 2010, 39; Chrissis, Konrad & Shrum 2011, 60.)

Organizational Process Definition (OPD)
Prosessialueen tarkoituksena on luoda edellytykset organisaation prosessien suoritukselle ja tiimien ohjeistuksille. Yhdenmukaiset prosessikäytännöt organisaatiossa luovat organisaatiolle hyötyä pitkällä aikavälillä tukien organisaation oppimista ja prosessien kehittämistä. Prosesseissa voi olla projektikohtaisia eroja ja organisaation standardiprosessit tulee suunnitella niin, että ne tukevat prosessien kustomointeja projekteittain niin tarvittaessa. (Software Engineering Institute 2010, 191.)
SG 1 Establish Organizational Process Assets · SP 1.1 Establish Standard Processes · SP 1.2 Establish Lifecycle Model Descriptions · SP 1.3 Establish Tailoring Criteria and Guidelines · SP 1.4 Establish the Organization's Measurement Repository · SP 1.5 Establish the Organization's Process Asset Library · SP 1.6 Establish Work Environment Standards · SP 1.7 Establish Rules and Guidelines for Teams

Organizational Process Focus (OPF)
Prosessialueen tarkoituksena on suunnitella organisaation prosessien kehittäminen organisaation vahvuuksiin ja heikkouksiin perustuen. Vahvuuksia ja heikkouksia kartoitettaessa huomioon otetaan kaikkien projektien suorittamat standardit ja kustomoidut prosessit. Mm. erilaisten kyselyiden ja benchmarkauksen avulla voidaan kerätä tietoa organisaation prosesseista ja niiden ongelmakohdista joiden perusteella prosessien kehittämistoimia voidaan suunnitella. (Software Engineering Institute 2010, 203.)
SG 1 Determine Process Improvement Opportunities · SP 1.1 Establish Organizational Process Needs · SP 1.2 Appraise the Organization's Processes · SP 1.3 Identify the Organization's Process Improvements SG 2 Plan and Implement Process Actions · SP 2.1 Establish Process Action Plans · SP 2.2 Implement Process Action Plans SG 3 Deploy Organizational Process Assets and Incorporate Experiences · SP 3.1 Deploy Organizational Process Assets · SP 3.2 Deploy Standard Processes · SP 3.3 Monitor the Implementation · SP 3.4 Incorporate Experiences into Organizational Process Assets

Organizational Performance Management (OPM)
Prosessialueen tarkoituksena on saada organisaation suorituskyky vastaamaan organisaation liiketoimintatavoitteita. Tavoitteena on hallita suorituskykyä proaktiivisesti ja iteratiivisesti analysoiden toistuvasti organisaation suorituskyvystä kerättyä dataa ja verrata sitä organisaation liiketoimintatavoitteisiin. Organisaatiosta riippuen liiketoimintatavoitteet voivat pitää sisällään esimerkiksi tuotteiden laadun parantamisen, organisaation tuottavuuden parantamisen tai tuotteiden jakeluketjun toiminnan tehostaminen. (Software Engineering Institute 2010, 217.)
SG 1 Manage Business Performance · SP 1.1 Maintain Business Objectives · SP 1.2 Analyze Process Performance Data · SP 1.3 Identify Potential Areas for Improvement SG 2 Select Improvements · SP 2.1 Elicit Suggested Improvements · SP 2.2 Analyze Suggested Improvements · SP 2.3 Validate Improvements · SP 2.4 Select and Implement Improvements for Deployment SG 3 Deploy Improvements · SP 3.1 Plan the Deployment · SP 3.2 Manage the Deployment · SP 3.3 Evaluate Improvement Effects

Organizational Process Performance (OPP)
Prosessialueen tarkoituksena on kerätä ja hallita määrällistä dataa organisaation standardiprosesseista. Tavoitteena on seurata suorituskykytavoitteiden saavuttamista ja muodostaa datan perusteella organisaation suorituskyvyn lähtötaso projekteja varten. Kerätyn datan avulla voidaan mm. määritellä suoriutuvatko prosessit vakaasti vai esiintyykö joissain prosesseissa poikkeuksellisen paljon ongelmia. Samalla data myös auttaa parantamaan organisaation standardiprosesseja paremmaksi. (Software Engineering Institute 2010, 235-236.)
SG 1 Establish Performance Baselines and Models · SP 1.1 Establish Quality and Process Performance Objectives · SP 1.2 Select Processes · SP 1.3 Establish Process Performance Measures · SP 1.4 Analyze Process Performance and Establish Process Performance Baselines · SP 1.5 Establish Process Performance Models

Organizational Training (OT)
Prosessialueen tarkoituksena on kehittää henkilöstön tietämystä ja kykyä, jotta he pystyvät suoriutumaan rooliensa asettamista vaatimuksista. Prosessialue keskittyy sen kaltaiseen koulutukseen, joka tukee organisaation liiketoimintatavoitteita, eikä näin ollen pidä sisällään yksittäisten projektien vaatimaa koulutusta. Käytännössä tämä voisi tarkoittaa esimerkiksi organisaation koulutussuunnitelman luomista tai mentorointiohjelman aloittamista organisaatiossa. Pääpaino prosessialueen koulutuksessa on kuitenkin niissä taidoissa ja tiedoissa joita tarvitaan organisaation standardiprosessien suorittamiseen. (Software Engineering Institute 2010, 247.)
SG 1 Establish an Organizational Training Capability · SP 1.1 Establish Strategic Training Needs · SP 1.2 Determine Which Training Needs Are the Responsibility of the Organization · SP 1.3 Establish an Organizational Training Tactical Plan o SP 1.4 Establish a Training Capability SG 2 Provide Training · SP 2.1 Deliver Training · SP 2.2 Establish Training Records · SP 2.3 Assess Training Effectiveness

Projektien hallinta (Project Management)

Prosessikategoria sisältää projektien hallinnoimiseen liittyvät käytännöt koskien projektien suunnittelua, hallintaa ja seurantaa. Tavoite on mahdollistaa projektin tavoitteiden täyttäminen ja aikataulussa pysyminen sekä tarjota keinoja ongelmatilanteiden hallintaan. (Software Engineering Institute 2010, 43; Chrissis, Konrad & Shrum 2011, 64.)

Integrated Project Management (IPM)
Prosessialueen tarkoituksena on projektin perustaminen ja sen hallinta siten, että kaikki asianosaiset huomioidaan. Näin on oltava myös silloin kun prosessi on kustomoitu organisaation standardiprosesseista. Tavoitteena on hallita käytännössä projektien kaikkia osa-alueita, kuten riskien, kustannusten, aikataulun ja resursoinnin hallinta. (Software Engineering Institute 2010, 157.)
SG 1 Use the Project's Defined Process · SP 1.1 Establish the Project's Defined Process · SP 1.2 Use Organizational Process Assets for Planning Project Activities · SP 1.3 Establish the Project's Work Environment · SP 1.4 Integrate Plans · SP 1.5 Manage the Project Using Integrated Plans · SP 1.6 Establish Teams · SP 1.7 Contribute to Organizational Process Assets SG 2 Coordinate and Collaborate with Relevant Stakeholders · SP 2.1 Manage Stakeholder Involvement · SP 2.2 Manage Dependencies · SP 2.3 Resolve Coordination Issues

Project Monitoring and Control (PMC)
Prosessialueen tarkoituksena on tarjota näkemys projektin tilanteesta, jotta tarvittaviin toimenpiteisiin voidaan ryhtyä ongelmatilanteiden ilmaantuessa. Käytännön tasolla tämä tarkoittaisi aikataulun, kulujen, työmäärän yms jatkuvaa seurantaa ongelmien havaitsemiseksi. (Software Engineering Institute 2010, 271.)
SG 1 Monitor the Project Against the Plan · SP 1.1 Monitor Project Planning Parameters · SP 1.2 Monitor Commitments · SP 1.3 Monitor Project Risks · SP 1.4 Monitor Data Management · SP 1.5 Monitor Stakeholder Involvement · SP 1.6 Conduct Progress Reviews · SP 1.7 Conduct Milestone Reviews SG 2 Manage Corrective Action to Closure · SP 2.1 Analyze Issues · SP 2.2 Take Corrective Action · SP 2.3 Manage Corrective Actions

Project Planning (PP)
Prosessialueen tarkoituksena on kuvata projektien toiminnot ja ylläpitää niitä. Kuvakset kertovat miten organisaatiossa tehdään projektisuunnitelma, kuinka kommunikoidaan kaikkien asianosaisten kanssa, kuinka suunnitelmaan sitoudutaan ja kuinka suunnitelmat ylläpidetään. Käytännön tasolla prosessialueen prosessin suorittamisesta syntyy projektisuunnitelma sisältäen projektisuunnitelman olennaiset osat kuten riskianalyysin, projektiaikataulun jne. (Software Engineering Institute 2010, 281.)
SG 1 Establish Estimates · SP 1.1 Estimate the Scope of the Project · SP 1.2 Establish Estimates of Work Product and Task Attributes · SP 1.3 Define Project Lifecycle Phases · SP 1.4 Estimate Effort and Cost SG 2 Develop a Project Plan · SP 2.1 Establish the Budget and Schedule · SP 2.2 Identify Project Risks · SP 2.3 Plan Data Management · SP 2.4 Plan the Project's Resources · SP 2.5 Plan Needed Knowledge and Skills · SP 2.6 Plan Stakeholder Involvement · SP 2.7 Establish the Project Plan SG 3 Obtain Commitment to the Plan · SP 3.1 Review Plans That Affect the Project · SP 3.2 Reconcile Work and Resource Levels · SP 3.3 Obtain Plan Commitment

Quantitative Project Management (QPM)
Prosessialueen tarkoituksena on määrällisen datan perusteella varmistua projektin laadullisten tavoitteiden ja prosessin suorituskyvyn saavuttamisesta. Tarkoituksena on siis määritellä mitkä ovat laadulliset tavoitteet ja mitkä ovat tavoitteet suorituskyvylle. Sen lisäksi päätetään millä mittareilla kyseisiä asioita seurataan. Projektistatistiikan tilastollisen analyysin perusteella voidaan arvioida kuinka projektin tavoitteet saavutettiin. (Software Engineering Institute 2010, 307.)
SG 1 Prepare for Quantitative Management · SP 1.1 Establish the Project's Objectives · SP 1.2 Compose the Defined Process · SP 1.3 Select Subprocesses and Attributes · SP 1.4 Select Measures and Analytic Techniques SG 2 Quantitatively Manage the Project · SP 2.1 Monitor the Performance of Selected Subprocesses · SP 2.2 Manage Project Performance · SP 2.3 Perform Root Cause Analysis

Requirements Management (REQM)
Prosessialueen tarkoituksena on hallita projektin lopputuloksena syntyviä tuotteita ja tuotekomponentteja sekä varmistua, että ne ovat vaatimusten ja projektisuunnitelman mukaisia. Tuote voi olla ohjelmiston lisäksi esimerkiksi palvelu tai muu palveluun suoraan liittyvä asia. Tarkoituksena on varmistua siitä, että kaikista projektiin liittyvistä vaatimuksista vallitsee yhteisymmärrys eikä väärinkäsityksiä esiintyisi. Vaatimusten hallintaan kuuluu lisäksi myös vaatimusten sekä muutosten dokumentointi jäljitettävyyden vuoksi. (Software Engineering Institute 2010, 341.)
SG 1 Manage Requirements · SP 1.1 Understand Requirements · SP 1.2 Obtain Commitment to Requirements · SP 1.3 Manage Requirements Changes · SP 1.4 Maintain Bidirectional Traceability of Requirements · SP 1.5 Ensure Alignment Between Project Work and Requirements

Risk Management (RSKM)
Prosessialueen tarkoituksena on havaita mahdolliset ongelmatilanteet ennen niiden syntymistä, jotta niihin voidaan reagoida etukäteen sovitulla tavalla. Riskien hallinta on jatkuvaa toimintaa ja tärkeä osa projektinhallintaa, jottei projektille tule ylitsepääsemättömiä esteitä riskien toteutumisvaiheessa. Tarkoituksena on että riskit jotka vaarantavat projektin onnistumisen, havaitaan etukäteen. Organisaatiolla tulisi olla suunnitelma kuinka tunnistetaan ja hallitaan riskit ja kuinka yhteistyö eri asianosaisten välillä toimii riskitilanteissa. (Software Engineering Institute 2010, 349.)
SG 1 Prepare for Risk Management · SP 1.1 Determine Risk Sources and Categories · SP 1.2 Define Risk Parameters · SP 1.3 Establish a Risk Management Strategy SG 2 Identify and Analyze Risks · SP 2.1 Identify Risks · SP 2.2 Evaluate, Categorize, and Prioritize Risks SG 3 Mitigate Risks · SP 3.1 Develop Risk Mitigation Plans · SP 3.2 Implement Risk Mitigation Plans

Supplier Agreement Management (SAM)
Prosessialueen tavoitteena on hallita tuotteiden ja palveluiden hankintaa ulkopuolisilta tahoilta. Prosessialue käsittää sen kuinka hankitaan tuotteita tai palveluita joita voidaan edelleen tarjota organisaation omille asiakkaille joko sellaisenaan tai osana organisaation omaa tuotetta tai palvelua. Prosessialue pitää sisällään toimittajien valinnasta lähtien myös toimittajasopimusten teon ja ylläpidon aina tuotteen tai palvelun päätymisen loppuasiakkaalle. (Software Engineering Institute 2010, 363.)
SG 1 Establish Supplier Agreements · SP 1.1 Determine Acquisition Type · SP 1.2 Select Suppliers · SP 1.3 Establish Supplier Agreements SG 2 Satisfy Supplier Agreements · SP 2.1 Execute the Supplier Agreement · SP 2.2 Accept the Acquired Product · SP 2.3 Ensure Transition of Products

Ohjelmistokehitys (Engineering)

Käytännön ohjelmistokehitystyöhön, mukaan lukien ylläpitotyöt, kuuluvat aktiviteetit ja toimenpiteet (Software Engineering Institute 2010, 47; Chrissis, Konrad & Shrum 2011, 68).

Product Integration (PI)
Prosessialueen tarkoituksena on varmistaa, että tuotteen eri komponentit toimivat yhdessä kuten niiden on tarkoitus toimia. Tarkoituksena on siis integroida tuotteen tai palvelun komponentit toisiinsa. Kriittisintä on varmistua komponenttien ulkoisten ja sisäisten rajapintojen toimivuudesta ja varmistaa niiden yhteensopivuus. Komponenttien integrointi on luonteeltaan jatkuvaa ja toistuu useita kertoja projektin aikana. (Software Engineering Institute 2010, 257.)
SG 1 Prepare for Product Integration · SP 1.1 Establish an Integration Strategy · SP 1.2 Establish the Product Integration Environment · SP 1.3 Establish Product Integration Procedures and Criteria SG 2 Ensure Interface Compatibility · SP 2.1 Review Interface Descriptions for Completeness · SP 2.2 Manage Interfaces SG 3 Assemble Product Components and Deliver the Product · SP 3.1 Confirm Readiness of Product Components for Integration · SP 3.2 Assemble Product Components · SP 3.3 Evaluate Assembled Product Components · SP 3.4 Package and Deliver the Product or Product Component

Requirements Development (RD)
Prosessialueen tarkoituksena on tunnistaa mitkä ovat tuotteelle asetetut vaatimukset ja kuka on asiakas jota tuote voisi kiinnostaa. Prosessialue käsittää kuvaukset siitä kuinka täytetään asiakastarpeet sekä tuotteelle ja tuotekomponenteille asetetut vaatimukset. Lopulta prosessialueen toimenpiteiden seurauksena tulisi syntyä liki täydelliset vaatimukset minkälainen tuotteen pitää olla kuinka asianosaiset huomioidaan ja kuinka tuotteen elinkaari hallitaan. (Software Engineering Institute 2010, 325.)
SG 1 Develop Customer Requirements · SP 1.1 Elicit Needs · SP 1.2 Transform Stakeholder Needs into Customer Requirements SG 2 Develop Product Requirements · SP 2.1 Establish Product and Product Component Requirements · SP 2.2 Allocate Product Component Requirements · SP 2.3 Identify Interface Requirements SG 3 Analyze and Validate Requirements · SP 3.1 Establish Operational Concepts and Scenarios · SP 3.2 Establish a Definition of Required Functionality and Quality Attributes · SP 3.3 Analyze Requirements · SP 3.4 Analyze Requirements to Achieve Balance · SP 3.5 Validate Requirements

Technical Solution (TS)
Prosessialueen tarkoituksena on kertoa kuinka tuotteet suunnitellaan ja toteutetaan vaatimusten mukaisesti. Prosessialueen toiminnot ohjaavat käytännön tason suunnittelu- ja ohjelmointityötä. Käytännön tasolla toiminnot voivat pitää sisällään esimerkiksi prototyyppien tekoa riittävän tietämyksen saamiseksi vaatimusten pohjalta. Prosessialueen toiminnot pitävät sisällään toimintaohjeet tuotteen koko elinkaaren ajalle aina ylläpitovaiheeseen saakka. (Software Engineering Institute 2010, 373-378.)
SG 1 Select Product Component Solutions · SP 1.1 Develop Alternative Solutions and Selection Criteria · SP 1.2 Select Product Component Solutions SG 2 Develop the Design · SP 2.1 Design the Product or Product Component · SP 2.2 Establish a Technical Data Package · SP 2.3 Design Interfaces Using Criteria · SP 2.4 Perform Make, Buy, or Reuse Analyses SG 3 Implement the Product Design · SP 3.1 Implement the Design · SP 3.2 Develop Product Support Documentation

Validation (VAL)
Tarkoituksena on varmistaa, että tuote tai palvelu täyttää sille asetetut vaatimukset. Käytännössä kyseiseen prosessiin liittyvät toiminnot voivat olla toimintoja monelta osa-alueelta. Validation voi sisältää esimerkiksi koulutus-, valmistus- ja tukipalveluiden toiminnan varmistamisen. Validation voi näin ollen pitää sisällään esimerkiksi tuotteen testauksen ja toisaalta myös kehitysvaiheen protoilun. Tavoitteena on saada validoinnin piiriin määritellyt asiat mahdollisimman aikaisessa vaiheessa kehityssykliä mukaan, jotta voidaan varmistua prosessin tavoitteiden saavuttamisesta. (Software Engineering Institute 2010, 393.)
SG 1 Prepare for Validation · SP 1.1 Select Products for Validation · SP 1.2 Establish the Validation Environment · SP 1.3 Establish Validation Procedures and Criteria SG 2 Validate Product or Product Components · SP 2.1 Perform Validation · SP 2.2 Analyze Validation Results

Verification (VER)
Prosessialueen tavoitteena on varmistaa, että synnytetyt tuotokset täyttävät niille asetetut määritykset. Verifioinnilla pyritään varmistamaan, että tuotteet ja palvelut täyttävät niille asetetut vaatimukset kaikilta osin ja kaikki näkökulmat huomioiden. Verifiointi on luonteeltaan inkrementaalinen prosessi, joka toistuu koko tuotteen tai palvelun elinkaaren ajan alkaen verifiointitarpeiden määrittelystä jatkuen aina valmiin tuotteen tai palvelun verifiointiin. (Software Engineering Institute 2010, 401.)
SG 1 Prepare for Verification · SP 1.1 Select Work Products for Verification · SP 1.2 Establish the Verification Environment · SP 1.3 Establish Verification Procedures and Criteria SG 2 Perform Peer Reviews · SP 2.1 Prepare for Peer Reviews · SP 2.2 Conduct Peer Reviews · SP 2.3 Analyze Peer Review Data SG 3 Verify Selected Work Products · SP 3.1 Perform Verification · SP 3.2 Analyze Verification Results

Tukiprosessit (Support)

Tuotekehityksen tukemiseen ja ylläpitoon liittyvät aktiviteetit. Tukiprosessien tavoitteena on nimenmukaisesti tukea muiden prosessiealueiden prosessien suoritusta. (Software Engineering Institute 2010, 51; Chrissis, Konrad & Shrum 2011, 77.)

Causal Analysis and Resolution (CAR)
Prosessialueen tarkoituksena on tunnistaa ne syyt jotka aiheuttivat prosessin epätoivotun lopputuloksen ja tehdä tarvittavat toimenpiteen suorituskyvyn parantamista varten. Prosessialueen toiminnoilla parannetaan prosessin laatua ja tuottavuutta estämällä virheet. Prosessialueen tehokkuutta voidaan parantaa sillä, että jaetaan vastaavan prosessialueen tietämystä ja kokemuksia eri projektien välillä, jotta muut voivat oppia toisten projektien kohtaamista ongelmista. (Software Engineering Institute 2010, 127.)
SG 1 Determine Causes of Selected Outcomes · SP 1.1 Select Outcomes for Analysis · SP 1.2 Analyze Causes SG 2 Address Causes of Selected Outcomes · SP 2.1 Implement Action Proposals · SP 2.2 Evaluate the Effect of Implemented Actions · SP 2.3 Record Causal Analysis Data

Configuration Management (CM)
Prosessialueen tarkoituksena on ylläpitää tuotteiden eheyttä tunnistamalla minkälaisen konfiguraation tuotteet vaativat. Konfiguraation hallinnassa päätetään sekä konfiguraation perustaso, että se minkälaisia muutoksia perustasoon tarvitsee tehdä. Konfiguraation hallinta voi kertoa esimerkiksi tuotteen tietokoneelle asettamat tekniset vaatimukset. Se voi pitää sisällään myös esimerkiksi tuotteen asennusohjeet tai käyttöohjeet. (Software Engineering Institute 2010, 137-138.)
SG 1 Establish Baselines · SP 1.1 Identify Configuration Items · SP 1.2 Establish a Configuration Management System · SP 1.3 Create or Release Baselines SG 2 Track and Control Changes · SP 2.1 Track Change Requests · SP 2.2 Control Configuration Items SG 3 Establish Integrity · SP 3.1 Establish Configuration Management Records · SP 3.2 Perform Configuration Audits

Decision Analysis and Resolution (DAR)
Prosessialueen tarkoituksena on analysoida mahdollisia erilaisia ratkaisuvaihtoehtoja muodollisen arvioinnin avulla. Tarkoituksena on määritellä millaiset asiat vaativat muodollista arviointia sekä suorittaa kyseisen kaltainen arviointi. Itse arviointiprosessissa pohditaan erilaisia ratkaisuvaihtoehtoja, päätetään kuinka eri vaihtoehtoja arvioidaan sekä tehdään suositukset vaihtoehtojen pohjalta. (Software Engineering Institute 2010, 149.)
SG 1 Evaluate Alternatives · SP 1.1 Establish Guidelines for Decision Analysis · SP 1.2 Establish Evaluation Criteria · SP 1.3 Identify Alternative Solutions · SP 1.4 Select Evaluation Methods · SP 1.5 Evaluate Alternative Solutions · SP 1.6 Select Solutions

Measurement and Analysis (MA)
Prosessialueen tavoitteena on tuottaa mittaustietoa suorituskyvystä organisaation johdolle päätöksen teon tueksi. Prosessialueen aktiviteettien tehtävänä on määritellä tavoitteet, mitä mittaustietoa kerätään, kuinka sitä analysoidaan, varastoidaan ja raportoidaan sekä kuinka siitä pystytään antamaan palautetta. Vaikka mittaus tapahtuu pääsääntöisesti projektitasolla, voidaan mittaustietoa hyödyntää koko organisaation tasolla. (Software Engineering Institute 2010, 175.)
SG 1 Align Measurement and Analysis Activities · SP 1.1 Establish Measurement Objectives · SP 1.2 Specify Measures · SP 1.3 Specify Data Collection and Storage Procedures · SP 1.4 Specify Analysis Procedures SG 2 Provide Measurement Results · SP 2.1 Obtain Measurement Data · SP 2.2 Analyze Measurement Data · SP 2.3 Store Data and Results · SP 2.4 Communicate Results

Process and Product Quality Assurance (PPQA)
Prosessialueen tarkoituksena on tarjota objektiivinen näkymä prosesseihin ja niihin liittyviin tuotoksiin. Käytännön tasolla tämä tarkoittaa sitä, että prosessia evaluoidaan objektiivisesti prosessikuvausten mukaan ja tarkistetaan toimitaanko prosessikuvausten mukaisesti. Lisäksi prosessista pyritään tunnistamaan poikkeavuudet ja varmistamaan että niihin reagoidaan asianmukaisesti. Lopullisena päämääränä on tukea korkealaatuisten tuotteiden valmistusta ja samalla tarjota läpinäkyvyyttä prosessiin sekä tarjota palautekanava prosessia ja sen tuotoksiin liittyen. (Software Engineering Institute 2010, 301.)
SG 1 Objectively Evaluate Processes and Work Products · SP 1.1 Objectively Evaluate Processes · SP 1.2 Objectively Evaluate Work Products SG 2 Provide Objective Insight · SP 2.1 Communicate and Resolve Noncompliance Issues · SP 2.2 Establish Records

Liite2. CMMI-DEV 1.3 Yleistavoitteet validoinnin ja verifioinnin kannalta

GG = Generic Goal, GP = Generic Practice

GG1: Suorita prosessialueen erityistavoitteet (Achieve Specific Goals)

GP 1.1: Suorita erityiskäytännöt (Perform Specific Practices)
Tarkoitus on varmistua siitä, että prosessialueen erityiskäytännöt suoritetaan haluttujen tuotteiden tai palveluiden tuottamiseksi. Tällä tasolla erityiskäytäntöjä ei tarvitse välttämättä tarvitse suorittaa minkään ohjeistuksen mukaisesti. Erityiskäytäntöjen suoritus voi käytännön tasolla vaihdella paljonkin sen mukaan, ketkä prosessia suorittavat ja hallinnoivat. (Software Engineering Institute 2010, 68.)

GG2: Vakiinnuta hallittu prosessi (Institutionalize a Managed Process)

GP 2.1: Aseta organisatorinen menettelytapa (Establish an Organizational Policy)
Organisatorisen menettelytavan asettamisella on tarkoitus määritellä mitä prosessin suorittamisella odotetaan syntyvän sekä tehdä nämä määritykset näkyviksi kaikille asianosaisille. Prosessialueesta riippuen se voi tarkoittaa eri asioita. Validoinnin ja Verifioinnin yhteydessä kyse on siitä, minkälaisia organisaatiotason odotuksia on sille, kuinka validointi ja verifiointi tehdään millekin tuotteelle ja millä kriteereillä. (Software Engineering Institute 2010, 69-71.)

GP 2.2: Suunnittele prosessi (Plan the Process)
Prosessin suunnittelun tarkoitus on kertoa mitä toimintoja on suoritettava haluttujen tulosten saavuttamiseksi. Vaatimuksen täyttäminen voi käytännössä tarkoittaa esimerkiksi prosessikuvauksen tekemistä. Validoinnin ja verifioinnin yhteydessä kyse on sen kuvauksesta kuinka kyseisten prosessialueiden tavoitteet täytetään projektien yhteydessä. (Software Engineering Institute 2010, 72-76.)

GP 2.3: Tarjoa resurssit (Provide Resources)
Resurssien tarjoamisessa on nimensä mukaisesti kyse siitä, että riittävät henkilöresurssit ovat käytettävissä prosessin suorittamista varten. Validoinnin ja verifioinnin osalta kyseeseen tulee myös riittävien työkalujen varmistaminen toimenpiteitä varten. Työkalut voivat olla esimerkiksi testien hallinnointiin tai simulointiin liittyviä välineitä, joilla validoinnin ja verifioinnin toimenpiteitä suoritetaan. (Software Engineering Institute 2010, 76-81.)

GP 2.4: Osoita vastuut (Assign Responsibility)
Myös vastuiden määrittelyllä on tarkoitus varmistua siitä, että prosessin suoritukselle ja sen tuloksille on määritelty vastuuhenkilöt. Vastuut on voitu määritellä joko staattisesti työnkuvausten perusteella tai dynaamisemmin prosessikuvauksen yhteydessä. (Software Engineering Institute 2010, 82.)

GP 2.5: Kouluta henkilöstö (Train People)
Henkilöstön koulutuksella on tarkoitus varmistua siitä, että prosessia suorittavilla henkilöillä on riittävät tiedot ja taidot prosessin suorittamista varten ja tarvittaessa myös kouluttaa henkilöstöä. (Software Engineering Institute 2010, 82.)

GP 2.6: Hallitse tuotoksia (Control Work Products)
Tuotosten hallinnassa on tarkoitus varmistua ja kontrolloida sitä, että tuotokset ovat laadultaan yhdenmukaisia koko tuotosten elinkaaren ajan. Erilaiset tuotteet voivat vaatia erilaista ja eritasoista kontrollia. Samalla myös kontrolli ja sen taso voi vaihdella myös yhden tuotteen kohdalla sen elinkaaren eri vaiheissa. Validoinnin ja Verifioinnin kohdalla kyse esimerkiksi niistä määrityksistä ja niiden hallinnasta joilla kerrotaan mille tuotteille ja millä kriteereillä verifioinnin ja validoinnin toimenpiteet suoritetaan. Mukaan luetaan myös validoinnin ja verifioinnin tuloksena syntyneet raportit. (Software Engineering Institute 2010, 88-93.)

GP 2.7: Tunnista ja osallista asianosaiset (Identify and Involve Relevant Stakeholders)
Asianosaisten tunnistamisessa ja osallistamisessa on tarkoitus tunnistaa ne sidosryhmät jotka ovat merkityksellisiä prosessin suorittamisen kannalta. Samalla tarkoitus on varmistua niistä keinoista joiden avulla esimerkiksi kommunikointi sidosryhmän kanssa hoidetaan. Validoinnin ja verifioinnin yhteydessä on lopulta kyse siitä mitä tuotoksia validoidaan ja verifioidaan ja minkä sopimusten mukaisesti. Tarvittaessa voidaan sidosryhmäksi ottaa mukaan myös loppukäyttäjät, joiden avulla voidaan varmistua hyväksytyistä tuotoksista prosessien suorittamisessa. (Software Engineering Institute 2010, 93-99.)

GP 2.8: Valvo ja hallitse prosessia (Monitor and Control the Process)
Prosessin hallinnoinnin ja valvonnassa on nimensä mukaisesti kyse prosessin päivittäisen suorituksen seurannasta ja sen kontrolloinnista. Jatkuvalla seurannalla pitäisi havaita tilanteet joissa on tarpeen ryhtyä korjaaviin liikkeisiin tilanteen niin vaatiessa. Validoinnin ja Verifioinnin kohdalla seuranta kohdistuu suoritettuihin validointi- ja verifiointitoimenpiteisiin ja esiin tulleisiin virheraportteihin, sekä niiden trendeihin (esim. uudet virheet vs. korjatut virheet). (Software Engineering Institute 2010, 100- 106.)

GP 2.9: Arvioi prosessin noudattamista (Objectively Evaluate Adherence)
Prosessin noudattamisessa on tarkoitus varmistua siitä, että prosessi suoritetaan kuten se on suunniteltu. Samalla tulee varmistua siitä että myös prosessin tuotokset täyttävät niille asetetut vaatimukset. Tyypillisesti prosessin ulkopuoliset henkilöt arvioivat prosessin suorituksen sekä tuotosten lopullisen laadun. Validoinnin ja verifioinnin yhteydessä on käytännössä tarkoitus varmistua siitä, että kyseisten prosessialueiden tehtävät tulevat hoidetuksi. (Software Engineering Institute 2010, 106-113.)

GP 2.10: Katselmoi tilanne korkeamman johdon kanssa (Review Status with Higher Level Management)
Prosessin katselmoinnissa korkeamman johdon kanssa on tarkoitus varmistua siitä, että ylemmällä johdolla on riittävä näkyvyys prosessiin. Korkeammalla johdolla on saatava riittävät tiedot prosessista ja sen suorituksesta päätöksen teon tueksi. (Software Engineering Institute 2010, 113.)

GG3: Vakiinnuta määritelty prosessi (Institutionalize a Defined Process)

GP 3.1: Määrittele prosessi (Establish a Defined Process)
Prosessin määrittelyn lähtökohtana ovat organisaation standardit prosessit, joista on räätälöity jokaiselle prosessialueelle omat prosessinsa. Lähtökohta standardiprosessien kehittämiselle ja edelleen räätälöinnille on Organizational Process Definition (OPD)-prosessialueen-toteutus. (Software Engineering Institute 2010, 115.)

GP 3.2: Kerää kokemuksia prosessista (Collect Process Related Experiences)
Prosessin liittyvien kokemusten keräämisen tarkoituksena on sanamukaisesti kerätä kokemuksia ja informaatiota käytännön toimista. Kokemusten ja informaation tarkoituksena on palvella niitä, jotka suunnittelevat oman organisaation standardiprosesseista räätälöidyn samankaltaisen prosessin toteuttamista. (Software Engineering Institute 2010, 115-116.)

Liite 3. CMMI-DEV 1.3 Validoinnin erityistavoitteet

SG = Specific Goal, SP = Specific Practice

SG 1: Valmistaudu validointiin (Prepare for Validation)

Valmistautumisaktiviteetteihin kuuluvat mm. niiden tuotteiden tai tuotekomponenttien valinta, joille validointi suoritetaan. Lisäksi aktiviteetit pitävät sisällään myös ne toimenpiteet joilla validointi mahdollistetaan kuten tarvittavien ympäristöjen valmistelu. (Software Engineering Institute 2010, 394.) Ohjelmistojen integraatiotestit voivat käytännössä pitää sisällään validointiin liittyviä toimenpiteitä varmistamalla toistuvasti ohjelmistojen toimivuuden. (Chrissis, Konrad & Shrum 2011, 533.)

SP 1.1: Valitse validoitavat tuotteet (Select Products for Validation)
Koska validointi voi olla kustannuksiltaan hyvinkin kallista toteuttaa, on validoinnille syytä määritellä missä laajuudessa se tullaan toteuttamaan. Lähtökohtana validoinnissa tulee olla loppukäyttäjätarpeiden täyttäminen ja määritellä laajuus sen mukaan. Validoinnin kohteena voivat olla esimerkiksi tuotteen toiminnallisuus, käyttöohjeet, koulutusmateriaalit jne. Validoinnin laajuutta mietittäessä on tarpeellista myös määritellä kuinka validointi toteutetaan. Validointimenetelminä voidaan käyttää esimerkiksi prototyyppejä, toiminnallisia demonstraatioita, loppukäyttäjien kanssa kommunikointia, pilotointia jne. Tavoitteena on lopulta kertoa mitä validoidaan, miten validoidaan ja millä kriteereillä validointi suoritetaan. (Software Engineering Institute 2010, 395-396; Chrissis, Konrad & Shrum 2011, 533-534.)

SP 1.2: Valmistele validointiympäristö (Establish the Validation Environment)
Vaatimukset validointiympäristölle asettavat käytännössä ne vaatimukset joita validoitavat tuotteet asettavat toimintaympäristönsä suhteen. Validointiympäristö voi validointia varten sisältää myös esimerkiksi erilaisia testityökaluja tai simulaattoreita validoinnin helpottamista varten. (Software Engineering Institute 2010, 397; Chrissis & Konrad & Shrum 2011, 536.) Validointiympäristönä voidaan käyttää myös tuotteen integrointiympäristöjä (Chrissis, Konrad & Shrum 2011, 536).

SP 1.3: Määrittele validoinnin kriteerit (Establish Validation Procedures and Criteria)
Määrittelemällä validoinnille kriteerit voidaan varmistua tuotteen toimivuudesta sille asetetun käyttötarkoituksen mukaisesti ja sellaisessa ympäristössä kun on tarkoitettu. Validoinnissa voidaan hyödyntää eri testitapauksia ja hyväksymistestejä, jotka omalta osaltaan varmistavat toiminnallisuutta määritysten mukaisesti. (Software Engineering Institute 2010, 398.)

SG 2: Suorita validointi tuotteille tai komponenteille (Validate Product or Product Components).

Keinot joilla varmistetaan tuotteiden ja tuotekomponenttien toimivuus kohdeympäristössä koko tuotteen elinkaaren ajan (Software Engineering Institute 2010, 399).

SP 2.1: Suorita validointi (Perform Validation)
Suorita validointitoimenpiteet kuten ne on määritelty. Mahdolliset poikkeavuuden tulee huomioida kulloiseenkin tilaanteeseen sopivalla tarkoituksenmukaisella tavalla. Validointitoimenpiteiden suorittamisesta voidaan tuottaa esimerkiksi tulosraportti, kuinka onnistuneesti validointi suoritettiin. (Software Engineering Institute 2010, 398-399.)

SP 2.2: Analysoi validoinnin tulokset (Analyze Validation Results)
Validoinnin, tarkastuksen tai demonstraatioiden tuloksia verrataan asetettuihin validointikriteereihin. Tulokset kertovat täytettiinkö vaatimukset ja jos ei, niin missä tapahtui virhe. Virheiden esiintyessä merkittävää on myös arvioida syy minkä vuoksi virhe syntyi. (Software Engineering Institute 2010, 399.)

Liite 4. CMMI-DEV 1.3 Verifioinnin erityistavoitteet

SG = Specific Goal, SP = Specific Practice

SG 1: Valmistaudu verifiointiin (Prepare for Verification)

Etukäteen on syytä varmistua että verifioinnin ehdot tulevat täytetyksi automaattisesti tuotekehityksen yhteydessä. Käytännössä nämä käytännöt voivat olla testausta, vertaisarviointeja, tuotedemostraatioita jne. (Software Engineering Institute 2010, 403.)

SP 1.1: Valitse verifioitavat tuotteet (Select Work Products for Verification)
Valitaan tuotteet jotka verifioidaan ja joille suoritetaan verifoinnin mukaiset toimenpiteet. Verifioinnissa käytettäviä menetelmiä voivat olla esimerkiksi automaattisten testien kattavuus, suorituskykytestaus, hyväksyntätestaus jne. Lopputuloksena tulee olla kuvaus siitä mitä tuotteita verifioidaan ja millä tavalla. (Software Engineering Institute 2010, 403-404.)

SP 1.2: Valmistele verifiointiympäristö (Establish the Verification Environment)
Verifioinnin suorittamista varteen tulee olla ympäristö, jossa aktiviteetit voidaan suorittaa. Ympäristö voi vaihdella sen mukaan, minkälaisesta verifioinnista on kyse. Vertaisarviointi vaatii erilaisen ympäristön, kuin esimerkiksi integraatiotestit. (Software Engineering Institute 2010, 405.)

SP 1.3: Määrittele verifioinnin kriteerit (Establish Verification Procedures and Criteria)
Määritellään verifioinnin kriteerit joilla varmistetaan tuotteiden toimivuus vaatimusten mukaisesti. Kriteerit voivat tulla esimerkiksi tuotteiden määrityksistä tai suoritettavista testeistä ja niille asetetuista parametreista. (Software Engineering Institute 2010, 405.)

SG 2: Suorita vertaisarvioinnit (Perform Peer Reviews)

Suoritetaan vertaisarvioinnit valituille tuotoksille. Vertaisarvioinnin ovat metodologinen tapa tunnistaa ja poistaa virheitä tuotteissa. (Software Engineering Institute 2010, 406.) Vertaisarviointia voidaan myös hyödyntää tiedon jakamiseen tiimin sisällä. Usein vertaisarviointia suoritetaankin samassa projektissa työskentelevien kesken. (Chrissis, Konrad & Shrum 2011, 547)

SP 2.1: Valmistaudu vertaisarviointeihin (Prepare for Peer Reviews)
Vertaisarviointiin valmistautuessa tunnistetaan ketkä ovat osallisia vertaisarvioinnissa. Vertaisarviointiin valmistautumisessa voivat olla hyödyksi esimerkiksi etukäteen tehdyt ja ylläpidettävät vertaisarvioinnin muistilistat, hyväksymiskriteerit, hylkäyskriteerit jolloin tarvitaan myöhemmin uusittava vertaisarviointi, vertaisarvioinnin aikataulu ja tuotteet joiden kehitykseen sovelletaan vertaisarviointia. Vertaisarvioinnin muistilista voi sisältää esimerkiksi vaatimukset sille, että ohjelmakoodi on ylläpidettävää, toteutuksessa on noudatettu sovittua ohjeistusta eikä siinä ole tunnettuja virheitä tai puutteita. (Software Engineering Institute 2010, 406-407.)

SP 2.2: Suorita vertaisarvioinnit (Conduct Peer Reviews)
Vertaisarvioinnin tarkoituksena on löytää ja poistaa virheitä mahdollisimman aikaisessa vaiheessa. Vertaisarvioinnin lähtökohtana tulisi olla se, että arviointiin on voitava varautua riittävästi, arviointi suoritetaan kontrolloidusti ja yhdenmukaisesti ja arvioinnin tulokset sekä tarpeelliset toimenpiteet kirjataan ylös. (Software Engineering Institute 2010, 408.)

SP 2.3: Analysoi vertaisarvioinnin tuloksia (Analyze Peer Review Data)
Vertaisarviointiin liittyvän datan analysointi, joka pitää sisällään kaiken vertaisarviointiin liittyvän datan. Arviointi tulee tehdä vertaisarvioinnin ohjeistuksille, kuin tuloksillekin. Datasta voidaan analysoida esimerkiksi vertaisarvioinnin ajankäyttöä, minkä tyyppisiä virheitä esiintyi ja minkä vuoksi joitain virheitä esiintyi vertaisarvioinneissa. (Software Engineering Institute 2010, 409.)

SG 3: Verifioi tuotokset (Verify Selected Work Products)

Verifioinnilla varmistetaan että tuotteet täyttävät niille asetetut vaatimukset suorittamalla niille määritellyt toimenpiteet (Software Engineering Institute 2010, 409).

SP 3.1: Suorita verifiointi (Perform Verification)
Suoritetaan tuotteille ne toimenpiteet, jotka verifiointiprosessiin on määritelty kullekin tuotteelle kuuluvaksi sekä kirjataan tulokset. Näihin toimenpiteisiin kuuluvat myös vertaisarvioinnit. (Software Engineering Institute 2010, 409.) Usein verifioinnilla tarkoitetaan testausta. Testaus on osa verifiointia, mutta kuten aikaisemmin on mainittu, kuuluu verifiointiin muitakin aktiviteetteja. (Chrissis, Konrad & Shrum 2011, 551.)

SP 3.2: Analysoi verifioinnin tulokset (Analyze Verification Results)
Verrataan testien tuloksia odotettuihin tuloksiin, jotta voidaan varmistua tuotteelle asetettujen vaatimusten täyttämisestä. Näihin tuloksiin lasketaan mukaan myös vertaisarvioinnin tulokset sekä muut raportit kuten esimerkiksi tulokset suorituskyky- ja rasitustesteistä. Tulosten perusteella analysoidaan onko tarpeen ryhtyä vielä korjaaviin toimenpiteisiin vai onko validointi suoritettu hyväksytysti. (Software Engineering Institute 2010, 410.)

Liite 5. CMMI-DEV ja ketterien menetelmien yhteensopivuus

Ketterien menetelmien ja CMMI-DEV yhteensopivuusvertailu on lähteestä: Fritzsche, Martin & Keil, Patrick. 2007. Agile Methods and CMMI: Compatibility or Conflict. e-Informatica Software Engineering Journal. Volume 1, Issue 1, 2007.

Yhteensopivuusvertailussa menetelmän kattavuus suhteessa CMMI-DEV:in prosessialueen tavoitteisiin on käytännössä nelitasoinen, 0:sta kolmeen plussaan. Konflikti tarkoittaa sitä, ettei mainittua ketterää menetelmää voida hyödyntää luopumatta jostain menetelmän periaatteista. Taulukoissa ei oteta kantaa siihen onko prosessialueen päämäärä saavutettavissa nimenomaisesti prosessialueen määritysten ja tavoitteiden mukaisesti vai jollakin CMMI-DEV:in sallimalla vaihtoehtoisella tavalla.

Taulukko prosessialueiden ja ketterien menetelmien yhteensopivuudesta

Process area	**XP**	**Scrum**
Causal analysis and resolution (CAR)	**0**	**0**
Configuration management (CM)	**+++**	**0**
Decision analysis and resolution (DAR)	**-**	**-**
Integrated project management (IPM)	**++**	**+++**
Measurement and analysis (MA)	**+**	**+++**
Organizational process definition (OPD)	**0**	**0**
Organizational process focus (OPF)	**-**	**-**
Organizational process performance (OPP)	**-**	**-**
Organizational training (OT)	**++**	**+**
Process and product quality assurance (PPQA)	**+**	**0**
Product integration (PI)	**+++**	**0**
Project monitoring and control (PMC)	**+++**	**+++**
Project planning (PP)	**+++**	**+++**
Quantitative project management (QPM)	**-**	**-**
Requirements development (RD)	**++**	**++**
Requirements management (REQM)	**+++**	**+++**
Risk management (RSKM)	**+++**	**+++**
Supplier agreement management (SAM)	**0**	**0**
Technical solution (TS)	**+++**	**0**
Validation (VAL)	**+++**	**+++**
Verification (VER)	**+++**	**0**
Conflicting (–) Not addressed (0) Partially supported (+) Supported (++) Largely supported (+++)		

Taulukko yleistavoitteiden ja ketterien menetelmien yhteensopivuudesta

Generic practice	**XP**	**Scrum**
Establish an organizational policy	**0**	**0**
Plan the process	**0**	**0**
Provide resources	**+**	**+++**
Assign responsibility	**+++**	**+++**
Train people	**+++**	**+++**
Control work products	**++**	**0**
Identify and involve relevant stakeholders	**+++**	**+++**
Monitor and control the process	**++**	**++**
Objectively evaluate adherence	**+**	**0**
Review status with higher level management	**++**	**+++**
Establish a defined process	**0**	**0**
Collect process related experiences	**-**	**-**
Conflicting (–) Not addressed (0) Partially supported (+) Supported (++) Largely supported (+++)		