Antti Simonen

YRITYKSEN TUOTEKEHITYSPROSESSIN MÄÄRITTELY JA KÄYTTÖÖNOTTO

Yrittäjyyden ja liiketoimintaosaamisen koulutusohjelma
2015

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

YRITYKSEN TUOTEKEHITYSPROSESSIN MÄÄRITTELY JA KÄYTTÖÖNOTTO

Simonen, Antti
Satakunnan ammattikorkeakoulu
Yrittäjyyden ja liiketoimintaosaamisen koulutusohjelma
Kesäkuu 2015
Ohjaaja: Pohjus, Anne & Grönholm, Jukka
Sivumäärä: 135
Liitteitä: 0

Asiasanat: ketterät menetelmät, ohjelmistokehitys, prosessit

Opinnäytetyön aiheena oli tutkia ja kehittää kohdeyrityksen tuotekehityksen prosessia. Tarkoituksena oli kehittää prosessista kaikkien tuotekehitystiimien yhteinen, tehokkaampi ja laadukkaampi tapa toimia. Tutkimus ja kehitystyö toteutettiin laadullisena toimintatutkimuksena kesän 2012 ja kevään 2015 välisenä aikana.

Tutkielman teoreettisessa osiossa käsiteltiin ohjelmistotuotantoa ja eri projektimalleja. Erityisesti keskityttiin ketteriin menetelmiin kuuluviin SCRUM- sekä Kanban -menetelmiin. Teoreettisessa osiossa käsiteltiin myös prosessin kehittämisen menetelmiä ja vaiheita.

Empiirisessä osiossa sovellettiin tutkimuksen teoreettisia menetelmiä kohdeyrityksen tuotekehityksen prosessin kehittämisessä. Osio käsitti sekä SCRUM- että Kanban -menetelmiin perustuvan prosessin kehittämisen, käyttöönoton ja seurannan.

Tuotekehityksen prosessin ongelmat kiteytyivät puutteellisiin tai muuttuviin vaatimusmäärittelyihin, testauksen ylikuormittumiseen sekä ohjelmistoprojektien aikatauluihin. Tutkimuksen tuloksena kumpikaan käytetyistä prosessimalleista ei tarjonnut suoria ratkaisuja kaikkiin kohdattuihin ongelmiin. Tuotekehitystiimeille luotiin yksi yhteinen prosessi, joka mahdollistaa monen ongelman ratkaisemisen. Ongelmien ratkaiseminen vaatii organisaatiossa käytettyä prosessia laajempia muutoksia.

DEFINING AND IMPLEMENTING A COMPANY'S DEVELOPMENT PROCESS

Simonen, Antti
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Entrepreneurship and Business Management
June 2015
Supervisor: Pohjus, Anne & Grönholm, Jukka
Number of pages: 135
Appendices: 0

Keywords: agile methods, processes, software development

The purpose of this thesis was to study and develop a company's software development process. The aim was to develop a more efficient and common process for all the company's development teams. The study was conducted as a qualitative action research between the summer of 2012 and the spring of 2015.

The topics discussed in the theoretical part of the thesis include software development and different kind of project models. The focus of this part was especially in agile methods and SCRUM and Kanban –methodologies. The theoretical part also included discussion of different project development methods and phases.

In the empirical part of the thesis, the theoretical methods were applied in the development of a company's software development process. This part included the development, introduction and follow-up of both SCRUM- and Kanban –based processes.

The biggest problems in the process were inadequate or changing requirements, overloaded testing and project schedules delays. As a result of this study, neither of the used process models provided direct solutions for these problems. A common process was created for all of the development teams that enabled the organization to fix most of the problems. To really fix all of the problems requires larger changes in the organisation than the process that is being used.
.

SISÄLLYS

1 JOHDANTO

Kreikkalainen filosofi Herakleitos sanoi aikanaan, että maailmassa ei ole mitään muuta pysyvää kuin muutos. Näiden sanojen merkitys on totta erityisesti ohjelmistoprojektissa (Chemuturi & Cagley 2010, 157). Nykymaailman ohjelmistot ovat hyvin monimutkaisia ja sisältävät rajapintoja moniin ulkoisiin järjestelmiin. Ohjelmistojen monimutkaisuudesta johtuen projektit ovat usein hyvin pitkäkestoisia ja maailma ehtii muuttua ennen kuin ohjelmistot ennätetään saada valmiiksi. Lopputuloksena on ohjelmisto, joka ei enää vastaa asiakkaan uusiin tarpeisiin. Ohjelmiston muuttaminen projektin loppuvaiheessa on kuitenkin kallista ja vie paljon aikaa. Juuri tästä syystä ohjelmistoprojektit usein viivästyvät ja ylittävät budjettinsa.

Ohjelmistolle alun perin asetetut vaatimukset eivät useinkaan ole enää projektin päätösvaiheessa täysin paikkansapitäviä. Voidaan huomata, että vaatimukset ovat epätäydellisiä, niissä on epäselvyyksiä tai ne ovat keskenään ristiriidassa. Usein tällaiset virheet havaitaan jo ohjelmiston kehitysaikana, mutta joskus virheet vaatimuksissa tulevat esille vasta, kun tuote on toimitettu asiakkaalle. Mitä myöhemmin virheet vaatimuksissa havaitaan, sitä kalliimmaksi niiden korjaaminen tulee. (Saleh 2009, 80.)

Mainitut ongelmat ovat tuttuja myös tämän opinnäytetyön kohdeyrityksessä. Kehitettävät ohjelmistot ovat yrityksen ydinosaamista ja yrityksen menestys on hyvin riippuvainen uusien ohjelmistoversioiden julkaisemista. Yrityksen toiminta edellyttää, että ohjelmiston kehitysprojekteja tulisi pystyä ennustamaan mahdollisimman tarkasti. Viivästymiset aikataulussa ja mahdolliset ongelmat tulisi nähdä hyvissä ajoin, jotta niihin pystytään reagoimaan ennen kuin on liian myöhäistä. Kun vaatimukset voivat muuttua kesken kehitystyön, ennustamisesta tulee hyvin hankalaa. Vaatimuksia on voitu myös ymmärtää väärin, jolloin ohjelmisto toimii eri tavalla, kuin on alun perin haluttu. Yrityksen tulee kuitenkin pystyä hallitsemaan näitä muutoksia ja kehittämään onnistuneita ohjelmistoja, jotka vastaavat asiakkaan tarpeisiin.

Tässä opinnäytetyössä kehitetään ja otetaan käyttöön kohdeyrityksen uusi tuotekehitysprosessi. Uuden prosessin toivotaan ratkaisevan ohjelmistoprojektien suurimmat ongelmat muutoksien ja ennustettavuuden suhteen, sekä lopulta parantavan yrityksen tuotekehityksen tehokkuutta. Uuden prosessin käyttöönoton myötä yrityksen kehittämien ohjelmistojen vaatimuksia voidaan turvallisesti muuttaa kesken projektin. Muutoksista tulee läpinäkyviä koko organisaatiolle ja niiden vaikutus projektiin pystytään määrittelemään. Uusi prosessi tulee myös yhdenmukaistamaan yrityksen eri tuotekehitystiimien toimintatavat. Eri tuotekehitystiimien prosesseista saadaan vertailukelpoisia keskenään, jolloin pystytään vertaamaan tiimien suorituskykyä ja ryhtymään tarvittaviin toimenpiteisiin, kun poikkeamia havaitaan. Uuden prosessin myötä koko yrityksen tuotekehityksestä on tarkoitus saada täysin läpinäkyvä, jotta yritys saa käyttöönsä oikeiden päätösten tekemiseksi vaaditun tiedon.

Opinnäytetyön tuloksena kehitettävä prosessi on osa laajempaa tuotehallinnan prosessia. Opinnäytetyöhön sisällytetään myös uuden prosessin jalkauttamisen yrityksen käytäntöön ja eri tuotekehitystiimien käyttöön. Prosessi tullaan tulevaisuudessa mahdollisesti jalkauttamaan myös osaksi emoyrityksen tuotekehityksen prosesseja. Prosessin jalkauttaminen kohdeyrityksen ulkopuolelle rajataan kuitenkin sen laajuuden ja epävarmuuden vuoksi tämän opinnäytetyön ulkopuolelle.

2 KEHITTÄMISTEHTÄVÄ

2.1 Tutkimustehtävä

Jotta kohdeyrityksen tuotekehitysprosessia voidaan lähteä kehittämään paremmaksi, tulee yritys ja sen nykyinen tuotekehitysprosessi tuntea syvällisesti ja perusteellisesti. Tulee olla selkeää, minkä vuoksi yrityksessä halutaan kehittää uusi tuotekehityksen prosessi, sekä mitä ongelmia uuden prosessin toivotaan ratkaisevan. Tulee myös ymmärtää mistä nämä ongelmat johtuvat, koska usein organisaatiossa eri henkilöillä voi olla eri käsitys ongelmien perimmäisestä syystä. Tutkimus-ongelmaksi muodostuu yrityksen nykyisten olosuhteiden, kulttuurin ja prosessien ymmärtäminen tar-

peeksi syvällisesti, jotta kyseiselle organisaatiolle voidaan onnistuneesti kehittää uusi ja tehokkaampi toimintatapa ja prosessi.

Vielä suuremmassa roolissa yrityksen nykytilan ymmärtäminen on prosessin jalkauttamisessa. Jalkauttaminen todennäköisesti epäonnistuu, jos kehitetty prosessi ei sovi kohdeyrityksen kulttuurille ja jos se vaatii yrityksen työntekijöiden toimivan täysin voimassa olevan kulttuurin vastaisesti.

Tutkimuksen tavoitteena onkin kehittää ja jalkauttaa yritykseen sellainen prosessi, joka saa yrityksen eri tuotekehitystiimit toimimaan yhtenäisen mallin mukaisesti. Opinnäyteyön tavoitteen mukaisesti uuden prosessin pitäisi tuoda yrityksen tuotekehitykseen mahdollisuuden erilaisten KPI-mittareiden käytön. KPI-mittareilla (Key Performance Indicator) tarkoitetaan suorituskykymittareita, joita yritykset usein käyttävät mittaamaan suorituskykyään tietyllä osa-alueella.

Kun yrityksen tuotekehitys toimii tehokkaammin ja halvemmalla, luodaan yrityksen tuottavuuden kasvulle yhä paremmat kasvunäkymät. Tuotekehitykseen tehtävien investointien määrä pienenee ja rahaa säästyy muihin investointeihin. Samalla uusia ohjelmistoversioita voidaan julkaista yhä nopeammin ja asiakkaalle enemmän lisäarvoa tuottavilla ominaisuuksilla. Kehitettävän prosessin pääasiallisena tarkoituksena on ratkaista olemassa olevat ongelmat, mutta samalla mahdollistaa uusien tuoteversioiden julkaiseminen ja asiakkaalle toimittaminen nykyistä nopeammin, jopa muutamien viikkojen välein.

Laadun parantuessa saadaan tulevaisuudessa tuotteen ylläpitotyön määrää pienemmäksi. Virheitä ei tarvitse korjata yhtä paljon kuin aikaisemmin ja voidaan keskittyä vanhojen ominaisuuksien korjaamisen sijaan uusien ominaisuuksien kehittämiseen.

2.2 Tutkimusmenetelmät

Opinnäytetyön tärkeimpinä tutkimusmetodeina käytetään havainnointia ja kyselyä. Havainnointi on tärkein tutkimusaineiston keräämiseen käytettävä tapa. Työskentelen opinnäytetyön kohdeyrityksessä ja olen ollut yrityksessä töissä noin kuusi vuotta.

Toimin yrityksen tuotekehityksessä esimiehenä, sekä useamman tiimin Scrum Masterina. Scrum Master vastaa siitä, että tiimi toimii Scrum-menetelmän oppien mukaan ja hänet voidaankin käsittää eräänlaisena valmentajana tiimille. Scrum Master isännöi kaikkia tiimin palavereja ja muutenkin varmistaa, että muu tiimi pystyy työskentelemään mahdollisimman tuotteliaasti. Minulla on ainutlaatuinen tilaisuus havainnoida kyseisessä roolissa yrityksen tuotekehitystiimin toimintaa, ongelmatilanteita ja sitä miten uusi prosessi tulee vaikuttamaan tuotekehitystiimin päivittäiseen toimintaan. (Beyer 2010, 4.)

Kyselytutkimuksella voidaan nopeasti ja tehokkaasti kerätä laaja tutkimusaineisto. Kyselytutkimuksen heikkona puolena ei pystytä arvioimaan kuinka vakavasti vastaajat ovat kyselyyn vastanneet. Opinnäytetyössä käytetään kyselytutkimusta selvittämään vanhan ja uuden tuotekehitysprosessin toimivuutta. Toteuttamalla kaksi kyselytutkimusta pystytään näiden kahden prosessin eroavaisuuksia vertaamaan. (Ojasalo, Moilanen & Ritalahti 2010, 108.)

Opinnäytetyö on laadullinen toimintatutkimus. Laadullisen tutkimuksen luotettavuutta voidaan lisätä tutkimalla aihetta useammasta eri näkökulmasta. Työskentelen yrityksessä useassa eri roolissa ja pystyn tutkijana tarkastelemaan yrityksen toimintaa useammasta näkökulmasta. Tällainen tarkastelu lisää tutkimuksen triangulaatiota ja parantaa sitä kautta tutkimustulosten luotettavuutta. (Ojasalo, Moilanen & Ritalahti 2010, 94.)

On hyvin haasteellista asettua useampaan rooliin ja tarkastella kohdeorganisaatiota useammasta näkökulmasta. Onnistuminen eri näkökulmien saamisessa, lisää tutkimuksen luotettavuutta ja tuo enemmän syvyyttä koko tarkasteluun, mutta epäonnistuessaan voi puolestaan heikentää tutkimustuloksien luotettavuutta.

Koska kyseessä on toimintatutkimus, pyritään kohdeorganisaation toimintaa ymmärtämään mahdollisimman syvällisellä tasolla. Syvällinen ymmärrys on tärkeää, jotta organisaatiolle kyetään kehittämään toimiva ja tehokas prosessi. Toimintatutkimuksen tavoin työ etenee spiraalimaisesti. Ensin luodaan suunnitelma, joka toteutetaan. Tämän jälkeen havainnoidaan tilannetta, jotta pystytään arvioimaan miten suunni-

telma ja toteutus ovat onnistuneet. Tämän jälkeen reflektoidaan kaikkea opittua ja aloitetaan spiraalin uusi kierros muokatulla suunnitelmalla.

2.3 Tehtävän rajaus

Opinnäytetyön tuloksena kehitettävä ja jalkautettava prosessi kattaa hyvin suuren osan kohdeorganisaation tuotekehityksen kehittämisessä. Varsinkin jalkauttamisen seurannassa voidaan opinnäytetyötä helposti venyttää ajallisesti liian pitkälle. Opinnäytetyö tulee rajata täsmällisesti, jotta siitä saadaan järkevä kokonaisuus.

Kohdeorganisaation tuotekehityksen prosessin kehittäminen kuuluu opinnäytetyöhön. Opinnäytetyön tuloksena kehitetään uusi tuotekehityksen prosessi, joka toimii osana kohdeorganisaation laajempaa PLM-prosessia (Product Lifecycle Management). PLM-prosessilla tarkoitetaan tuotteen elinkaaren hallinnan prosessia. PLM-prosessin kehittäminen tai jalkauttaminen ei kuulu tämän opinnäytetyön piiriin ja on rajattu sen ulkopuolelle.

Opinnäytetyöhön rajataan kuuluvaksi kiinteänä osana kehitetyn prosessin jalkauttaminen kohdeyrityksen tuotekehitystiimeille. Opinnäytetyössä käsiteltävät tiimit on kuvattu luvussa 3.3. Kehitettävä prosessi on mahdollisesti tarkoitus jalkauttaa myöhemmässä vaiheessa osaksi koko konsernissa tapahtuvaa tuotekehitystä, mutta laajempi käyttöönotto ja jalkauttaminen rajataan tämän opinnäytetyön ulkopuolelle.

Kun uusi tuotekehitysprosessi on otettu käyttöön kohdeyrityksen tuotekehityksessä, sen käyttöä seurataan ja prosessia kehitetään jatkuvasti eteenpäin. Ennen uuden prosessin käyttöönottoa kohdeyrityksessä teetetään lähtötilannetutkimus, jolla tutkitaan kuinka hyvin entinen prosessi toimii tuotekehityksen työntekijöiden kannalta. Kun uusi prosessi on otettu käyttöön ja jalkautettu organisaatioon, teetetään toinen vertaileva tutkimus. Näiden kahden tutkimuksen tuloksia vertaamalla saadaan hyvän käsitys siitä, miten uusi prosessi on muuttanut tilannetta kohdeyrityksen tuotekehityksessä.

Vertaileva tutkimus teetetään syksyllä 2014, kun uusi prosessi on ollut jonkin aikaa käytössä. Vertailevan tutkimuksen tuloksia verrataan opinnäytetyön alussa tehdyn tutkimuksen tuloksiin ja analysoidaan prosessin vaikutukset tuotekehityksen seurantaan, suorituskykyyn ja laatuun.

3 TAUSTA

3.1 Kohdeyritys

Opinnäytetyö suoritetaan OpusCapita Group Oy:ssä ja osana sen Tampereen tuotekehitysyksikköä. OpusCapita Group on entinen Itella Informaatio, joka yhdistyi OpusCapita Oy:n kanssa tammikuussa 2014. OpusCapita Group tarjoaa asiakkailleen taloushallinnon ulkoistus- sekä automatisointipalveluja. OpusCapita Group on osa Itella-konsernia, joka sisältää OpusCapitan lisäksi viestinvälityksen ja logistiikan liiketoimintaryhmät.

OpusCapita Groupissa työskentelee nykyään n. 2400 työntekijää yhdeksässä eri Euroopan maassa. Opinnäytetyö suoritetaan erityisesti automatisointipalvelujen tuotekehityksessä, joka on pääasiassa keskitetty Tampereen toimipisteeseen. Tampereen toimipisteen lisäksi tuotekehitystä tehdään myös yrityksen Latvian toimipisteessä, joka sijaitsee Rigassa. Lisäksi tuotekehitystiimeissä työskentelee muutamia alihankkijoita. Tampereen toimipisteessä toimii tuotekehityksen eri tehtävissä n. 40 työntekijää. (OpusCapitan www-sivut.)

OpusCapita Oy perustettiin vuonna 1984 ja se oli erikoistunut taloushallinnon automatisointipalveluihin. Yritys siirtyi osaksi Itella-konsernia, kun Itella Informaatio osti sen kesällä 2011. Ensimmäisen vuoden aikana uusi omistaja ei näkynyt millään tavalla yrityksen arjessa, mutta vuoden 2012 aikana yrityksen toimintoja alettiin integroida emoyrityksen prosesseihin ja tämä on aiheuttanut muutoksia myös tämän opinnäytetyön suorittamiseen. (Itellan Pörssitiedote 23.6.2011.)

Viimeisin muutos tämän opinnäytetyön kohdeyrityksessä tapahtui tammikuussa 2014, kun OpusCapita Oy ja Itella Informaatio yhdistyivät uudeksi OpusCapitaksi. Uuden tuotekehitysprosessin kehitys alkoi n. 100 hengen yrityksessä ja nyt kehitys jatkuu yli 2000 henkilöä työllistävässä OpusCapita Groupissa. Tällainen kasvu tuo myös tuotekehitysprosessille uusia vaatimuksia, jotka vaikuttavat prosessin kehittämiseen.

Yrityksen ydinosaamista ovat erilaiset keskitetyt likviditeetinhallinnan järjestelmät, sekä rahoituksen ja sähköisen laskunkäsittelyn ratkaisut. Tämän lisäksi Itella Informaatio toi mukanaan uusia kehitettäviä taloushallinnon ratkaisuja, mutta koska näitä tuotteita ei vielä kehitetä yrityksen Tampereen toimipisteessä, niiden kehitys rajataan tämän opinnäytetyön ulkopuolelle.

3.2 Tuotteet

OpusCapita Groupin tuotteet muodostuvat taloushallinnon automatisoinnin ohjelmistoista. Yrityksen asiakkaat käyttävät ohjelmistoja automatisoidakseen oman yrityksensä taloushallinnon prosesseja. Automatisoinnilla pyritään lisäämään tehokkuutta, vähentämään virheitä ja saamaan enemmän ajantasaista tietoa yrityksen talouden tilasta nyt ja tulevaisuudessa.

Tässä opinnäytetyössä tuotteista käytetään termiä ohjelmisto. Voidaan myös käsittää, että OpusCapitan tuotteisiin kuuluu myös yrityksen tarjoamat muut palvelut, mutta tämän opinnäytetyön kehityskohteena oleva tuotekehitysprosessi koskettaa vain ohjelmistotuotteiden kehitystä. Tässä opinnäytetyössä ei käsitellä yrityksen muita tuotteita tai niiden tuotannon prosesseja.

Taloushallinto käsittää monta eri osa-aluetta. Näihin osa-alueisiin kuuluu mm. laskujen ja maksujen käsittely, tilien hallinta ja raportointi. OpusCapitan tuotteet pyrkivät kattamaan mahdollisimman suuren osan näistä osa-alueista. Kaikki asiakkaat eivät kuitenkaan tarvitse tai halua ostaa ohjelmistoa kaikkiin taloushallintonsa osa-alueiden tarpeisiin. Tästä syystä OpusCapitan ohjelmisto on jaettu pienempiin osiin. Erilliset osat muodostavat asiakkaalle yhden yhtenäisen ohjelmistokokonaisuuden.

Jokaisella erillisellä ohjelmistolla on oma tuotepäällikkönsä (engl. Product Manager), joka vastaa kyseisen tuotteen kehittämisestä oikeaan suuntaan niin, että se sisältää sellaisia ominaisuuksia, joista asiakkaat ovat valmiita maksamaan. Voidaan siis katsoa, että tuotepäällikkö ohjaa oman tuotteensa tuotekehityksen työtä. Yrityksen tuotteita kehitetään jatkuvasti. Niihin lisätään uusia ominaisuuksia sekä korjataan ilmenneitä ongelmia. Tuotepäällikkö määrittelee oman tuotteensa osalta sisällön jokaiselle uudelle versiolle.

Kun asiakas hankkii ohjelmiston itselleen, se useimmiten asennetaan asiakkaan omalle palvelimelle. Palvelin (engl. Server) on tietokone, joka tarjoaa tietoverkon välityksellä erilaisia palveluita muille tietokoneille eli asiakaskoneille (engl. Client). Kohdeyrityksen ohjelmisto asennetaan tällaiselle palvelimelle, josta sitä käytetään selainpohjaisen käyttöliittymän tai työpöytäsovelluksen avulla. Kohdeyrityksen ohjelmistot ovat alun perin sisältäneet pelkästään työpöytäpohjaisen käyttöliittymän. Jokaiselle ohjelmiston käyttäjälle asennettiin oma asiakassovellus, jonka avulla palvelimelle asennettua ohjelmistoa käytettiin. Ohjelmiston käyttämä tieto sijaitsee vain yhdessä paikassa, jolloin kaikki käyttäjät pystyvät työskentelemään samojen tietojen pohjalta. Tekniikan kehittyessä ohjelmistoon on lisätty mahdollisuus selainpohjaiseen käyttöön, jolloin ohjelmiston käyttäjät ottavat yhteyttä palvelimeen omalla internet-selaimellaan ja käyttävät ohjelmistoa selainpohjaisen käyttöliittymän avulla. Selainkäyttö ei vaadi asennettavaa ohjelmistoa asiakkaan tietokoneelle, joten sen käyttöönotto on perinteistä työpöytäsovellusta helpompaa.

Selainpohjainen käyttöliittymä on vähitellen syrjäyttämässä työpöytäpohjaisen käytön, mutta tuote sisältää yhä paljon toimintoja, jotka ovat saatavilla ainoastaan työpöytäpohjaisessa käyttöliittymässä. Selainpohjaisen käyttöliittymän kehittyessä ja käytön yleistyessä kohdeyritys on alkanut tarjoamaan tuotettaan myös palveluna, jolloin ohjelmistoa ei asenneta enää asiakkaan omalle palvelimelle vaan asiakas käyttää ohjelmistoa kohdeyrityksen omalta palvelimelta internet-selaimen avulla. Tällainen toimintatapa pienentää käyttöönottokustannuksia ja helpottaa koko ohjelmiston käyttöönottoa entisestään. Asiakkaan on helpompaa aloittaa tuotteen käyttö, kun ei tarvita omia laitehankintoja palvelinta varten. Myös tuotteen tekninen ylläpito pysyy kohdeyrityksellä, joka vähentää tarvetta asiakkaan omalle tekniselle tuelle.

Pelkkä ohjelmiston asennus asiakkaan tietokoneelle tai palvelimelle ei riitä tuotteen käyttöönottoon. Ohjelmisto pitää konfiguroida eli määritellä, ennen kuin se voidaan ottaa käyttöön. Ohjelmistoon pitää määritellä muun muassa käytettävät yritykset, tilit ja käyttäjät sekä määritellä yhteydet asiakkaan käyttämiin pankkeihin. Pankkiyhteyksien luominen vaatii sopimuksia pankkien kanssa. Tämä määrittelytyö sisältää paljon teknistä määrittelyä, eikä asiakas kykene useinkaan ottamaan ohjelmistoa käyttöön ilman kohdeyrityksen konsultin apua.

Ohjelmiston asennus ja käyttöönotto ovat usein pitkiä prosesseja. On hyvin tärkeää, että ohjelmisto tukee asiakkaan käyttämiä prosesseja ja siksi näiden vaiheiden aikana myös usein kehitetään asiakkaan omia taloushallinnon prosesseja. Näin asiakas pystyy saamaan ohjelmistosta parhaimman hyödyn. Kun asiakkaalle on asennettu ja määritelty jokin tuotteen versio, ei siihen enää pystytä tekemään ohjelmamuutoksia. Suurempien versiopäivitysten yhteydessä joudutaan ohjelmisto asentamaan uudelleen ja tietyissä tapauksissa uusi versio vaatii uutta määrittelytyötä ennen kuin se voidaan taas ottaa täysin käyttöön.

Koska versiopäivitykset ovat näinkin suuria projekteja, ei kohdeyrityksen ole mielekästä julkaista uusia versioita kovin tiheään tahtiin. Asiakkaat eivät halua päivittää ohjelmistoaan jatkuvasti, vaan versioita käytetään usein jopa vuosien ajan, ennen kuin ohjelmisto päivitetään uudempaan versioon. Uusi versio julkaistaan yleensä 1-2 kertaa vuodessa. Jokaista uutta versiota ei toimiteta jokaiselle asiakkaalle vaan vanhojen asiakkaiden ohjelmistoja päivitetään lähinnä tapauskohtaisesti. Jos uusi versio sisältää jonkin halutun uuden ominaisuuden, niin asiakas on yleensä halukas aloittamaan päivitysprosessin. Monilla yrityksen asiakkailla on käytössään jopa useita vuosia vanhoja versioita. Jos ohjelmisto toimii eikä asiakas tarvitse uudempien versioiden tuomia ominaisuuksia ja hyötyjä, niin versiota ei yleensä lähdetä päivittämään.

Yritys ei pysty julkaisemaan uutta versiota ennen kuin kaikkien versioon kuuluvien tuotteiden tuotekehitys on valmis. Versiota ei pystytä julkaisemaan yhtään aikaisemmin, vaikka joku yksittäinen tuote olisikin valmis etuajassa. Jos taas joku yksittäinen tuote myöhästyy aikataulusta, se tarkoittaa myöhästymistä koko versiolle. Eri

tuotteiden kehitystiimien tulisi työskennellä yhdessä, jotta jokainen tuote saadaan valmiiksi mahdollisimman nopeasti ja samanaikaisesti.

3.3 Tiimit

Kaikkien ohjelmistojen tuotekehitys on erillisten tiimien vastuulla. Tiimien koot ovat hyvin pieniä ja vaihtelevat muutamasta henkilöstä alle kymmeneen henkilöön. Jokainen tiimi toimii hyvin läheisessä yhteistyössä oman tuotteensa tuotepäällikön kanssa. Sen lisäksi, että tiimin jäsenille on kehittynyt syvä tuote-osaaminen, he ovat myös oppineet tuntemaan tuotteensa tuotepäällikön hyvin, joka luo kriittiselle yhteistyölle hyvät mahdollisuudet.

Kaikkia kohdeyrityksen tuotteita ei kehitetä aktiivisesti, vaan ne ovat elinkaarensa siinä vaiheessa, ettei jatkokehitys välttämättä tarjoa asiakkaille enää merkittäviä lisähyötyjä. Usein nämä tuotteet ratkaisevat hyvin erityisiä ongelmia, eivätkä niiden asiakasmäärät ole kovin suuria. Näiden tuotteiden kehitystä ohjaavat markkinoilta tulevat tarpeet ja kehitystä tehdään tapauskohtaisesti ilman, että tuotteelle olisi jatkuvasti resursoitu omaa tuotekehitystiimiä.

Osana kohdeyrityksen tuotekehitystä toimii myös suhteellisen suuri laadunvarmistusorganisaatio (engl. Quality Assurance, QA). Laadunvarmistuksessa työskentelee n. 10 henkilöä, joiden pääasiallinen vastuu on varmistaa oman tuotealueensa laatu. Vaikka tuotekehittäjilläkin on vastuu omista tekemisistään, niin QA:n avulla saadaan varmistettua, että kaikki ohjelmiston eri osat toimivat yhdessä luotettavasti. Laadunvarmistuksen testaajilla on usein hyvin syvä tuoteosaaminen ja hyvä käsitys asiakkaiden tarpeista. He tukevat hyvin paljon asiakaspalvelua ja ratkovat asiakaspalvelun kanssa mahdollisia asiakasongelmia. Tätä kautta muodostuneen tuotetietämyksen avulla he kykenevät testaamaan kehitettävää ohjelmistoa ja havaitsemaan mahdolliset ongelmat tehokkaammin kuin tuotekehittäjät.

Opinnäytetyön alkuvaiheessa tuotekehitystiimit olivat hyvin tuotesidonnaisia. Jokaisella aktiivisesti kehitettävällä tuotteella oli oma dedikoitu tiiminsä ja tuotekehitysresurssinsa. Kyseisten tuotekehitystiimien koko määräsi tuotteisiin tehtävien inves-

tointien koon. Tämä ei ole kuitenkaan ollut kovin tehokas tapa ohjata tuotekehitystä ja kohdeyrityksessä ollaankin nyt luopumassa dedikoiduista tiimeistä ja siirtymässä yrityksen strategian mukaisesti muuttuviin tiimeihin.

Yrityksen strategian mukaisesti eri tuotteet ovat eri ohjelmistoversioissa pääsijalla. Yhdessä versiossa voidaan keskittyä kehittämään yhtä tuotetta, kun taas seuraavassa versiossa jotain toista. Dedikoitujen kehitysresurssien kanssa tällainen työskentely ei ole mahdollista ja tästä syystä tiimien koostumusta on alettu muuttamaan aina haluttujen tarpeiden mukaisiksi.

Tiimit muodostetaan aina jokaisen ohjelmistoversion alkuvaiheessa. Jokaiselle tiimille pyritään asettamaan vain yksi tuote ja tuotepäällikkö, jotta tiimin on helpompi keskittyä yhteen asiaan eikä tiimin sisälle muodostu erillisiä alitiimejä. Jokaisella tuotteella on yhä oma pääkehittäjänsä, joka tuntee kyseisen tuotealueen parhaiten, mutta siirtämällä kehittäjiä tuotteesta toiseen pystytään vastaamaan paremmin yrityksen johdon asettamiin strategisiin tavoitteisiin, kuin myös jakamaan tuotetietoutta eri kehittäjien kesken. Tällä tavalla koko tuotekehityksestä tulee jatkossa entistä ketterämpi.

3.4 Työkalut prosessin tukena

Kohdeyrityksen käyttämät projektihallinnan työkalut ovat tärkeässä osassa tuotekehitysprosessia. Työkalujen ominaisuudet ratkaisevat tietyltä osin miten prosessissa toimitaan. On totta, että käytettävät työkalut pitäisi valita kehitetyn prosessin mukaan, eikä toisinpäin. Tämä ei kuitenkaan aina ole täysin mahdollista kun käytetään yleisiä kolmannen osapuolen kehittämiä ohjelmistoja. Toiminnanohjuksen järjestelmät ovat laajoja ja monimuotoisia ohjelmistoja, jotka juurtuvat usein syvälle yrityksen toimintaan. Tästä syystä valittua järjestelmää ei kovin helposti lähdetä vaihtamaan.

Kohdeyrityksen tuotekehityksen toiminnanohjausjärjestelmänä toimii hyvin pitkälti Microsoftin kehittämä Team Foundation Server (TFS). TFS on erillinen palvelinohjelmisto, joka on suunniteltu ohjelmistokehityksen tiimeille ja niihin kuuluville kehit-

täjille, testaajille, arkkitehdeille, projektipäälliköille, liiketoiminta-analyytikoille ja kaikille muille, jotka jollain tavalla ovat osallisena tuotekehitysprojekteissa. (Blankenship, Woodward & Holliday 2011, 4.)

TFS pitää sisällään versiohallinnan, sekä projektihallinnan toiminnallisuudet. Se sisältää hyvin kattavat raportointitoiminnot, joiden avulla saadaan hyvä kuva ohjelmistoprojektien tilasta lähes reaaliaikaisesti. TFS tarjoaa kaksi erillistä käyttöliittymää, Visual Studio kehitystyökalun, sekä internet-selaimella käytettävän selainkäyttöliittymän. Koska Visual Studio on myös kehitystiimin pääasiallinen työkalu, viitataan TFS:ään usein myös pelkällä Visual Studio -nimellä.

Microsoft julkaisi nykyisen TFS:n ensimmäisen esiversion vuonna 2004. Tällöin tuotteen nimi oli vielä Visual Studio Team System (VSTS) 2005 (koodinimeltään Burton). Siihen asti ohjelmistotiimit olivat seuranneet koodimuutoksia manuaalisesti, sekä pitäneet kirjaa edistymisestä Excel-tiedostoissa. Ensimmäistä versiota markkinoitiin Systems Development Life Cycle (SDLC) ohjelmistona. SDLC pitää sisällään tuotteen kehittämiseen liittyvät toiminnot, kuten määrittelyn, arkkitehtuurin, kehityksen, testauksen ja projektinhallinnan. Seuraavassa versiossa (VSTS 2008) Microsoft muokkasi lähestymistään enemmän Application Lifecycle Managementin (ALM) suuntaan. ALM:llä tarkoitetaan kokoelmaa työkaluja, sekä prosesseja, jotka auttavat organisaatioita hallitsemaan tuotetta sen koko elinkaaren ajan. ALM integroi eri tiimit, ohjelmistoalustat, sekä aktiviteetit yhdeksi kokonaisuudeksi, jonka avulla organisaatio kykenee tuomaan tuotteeseen jatkuvasti liiketoiminnallista lisäarvoa. (Hundhausen 2012, 1303-1322.)

TFS kattaa nykyään koko ohjelmiston elinkaaren aina ensimmäisestä tuoteideasta sen lopettamiseen saakka. Tämä elinkaari pitää sisällään projektin aloittamisen, määrittelyn, suunnittelun, kehittämisen, testaamisen, julkaisemisen, asentamisen ja jopa operoimisen erilaisten monitorointityökalujen avulla. Kohdeyrityksellä on käytössä TFS versio 2012, joka on kolmas merkittävä julkaisu sen jälkeen, kun ALM ominaisuudet ensimmäisen kerran lisättiin tuotteeseen. (Hundhausen 2012, 1322.)

3.4.1 Versionhallinta

Versiohallinnasta voidaan käyttää myös muita termejä, kuten lähdekoodin hallinta (engl. Source Control) tai versionhallintajärjestelmä (engl. Revision Control). Koska nykyaikainen ohjelmisto pitää sisällään paljon muutakin kuin pelkän ohjelmiston lähdekoodin, kutsutaan sitä tässä opinnäytetyössä versiohallinnaksi. Versionhallinta pitää sisällään kaiken kohdeyrityksen tuotekehityksen kehittämän ohjelmakoodin, kuvat, dokumentaation ja muut resurssit. Versiohallinta sisältää myös mahdollisuuden hallita näiden resurssien eri versioita, jotta tiedostojen historiassa voidaan palata haluttuun ajankohtaan. Versiohallinta mahdollistaa myös useamman eri sijainneissa työskentelevän tahon työskentelyn saman lähdekoodin parissa. Nämä kaikki ovat välttämättömyyksiä nykyaikaiselle ohjelmistoprojektille. (Blankenship, Wood-ward & Holliday 2011, 83.)

Versiohallinnan avulla ohjelmakoodiin tehdyt muutokset säilyvät itsenäisinä muutoksina ja niitä voidaan tarkastella erikseen ja tarvittaessa palata vanhempaan versioon. Kun samaa ohjelmistoa kehitetään useamman kehittäjän toimista, on hyvin tärkeää pystyä hallitsemaan muutoksia keskitetysti. Jokainen kehittäjä saa noudettua versiohallinnasta itselleen uusimman version, jonka jälkeen hän voi työskennellä kyseisen ohjelmistokoodin parissa. Versionhallintaan merkataan tieto siitä kuka on milläkin hetkellä tekemässä muutoksia mihinkin tiedostoon ja sitä myötä sen tietävät myös muut yrityksen tuotekehittäjät. Kun tuotekehittäjä on saanut muutoksensa valmiiksi, hän voi siirtää muutokset takaisin versionhallintaan, josta muut tuotekehittäjät voivat halutessaan hakea sen. Versionhallinta tarjoaa myös työkaluja mahdollisten ristiriitatilanteiden ratkaisemiseen. Tällaisissa tilanteissa kohdetiedosto on muuttunut toisen kehittäjän toimesta sillä aikaa, kun toinen kehittäjä on työskennellyt tiedoston parissa.

Versiohallinta ei sinällään ole merkittävässä osassa prosessia ja käytettävä versiohallinta voitaisiin tarvittaessa vaihtaa täysin toiseen tuotteeseen ilman, että se vaikuttaisi käytettävään prosessiin. TFS:n projektinhallinnan osio on kuitenkin hyvin tärkeässä osassa käytettävää prosessia. Projektinhallinta määrittelee tuotekehitysprosessissa hyvin monia asioita ja tuotekehitysprosessin kehittämisen suurimmista haasteista onkin saada kehitettävä prosessi toimimaan käytössä olevan projektinhallinnan ohjel-

miston kanssa yhteen. On hyvin tärkeää, että työkalu tukee prosessia eikä ohjaa prosessin sisäisiä toimintoja väärään suuntaan.

3.4.2 Projektinhallinta

TFS:n sisältämä projektinhallintaohjelmisto mahdollistaa halutun prosessimallin pohjan valinnan. Yleisimmät prosessimallit ovat jonkinlaisia muunnelmia erilaisista ketterän ohjelmistokehityksen menetelmistä. Lisää ketteristä menetelmistä ja SCRUM - prosessimallista luvuissa 4.3 ja 4.4. Suurimmat erot eri prosessimallien kesken TFS:n kannalta, ovat erityyppisten objektien nimissä. TFS sisältää periaatteessa aina samanlaiset artefaktit, mutta näiden nimet ja sisältämät tietokentät eroavat eri prosessimallien välillä. On kuitenkin hyödyllistä valita käytössä olevaa prosessia tukeva malli myös TFS:n prosessimalliksi, jotta termistö pysyy yhtenäisenä ja tuttuna myös TFS:n kanssa työskenneltäessä. Näin työkalu tukee käytössä olevaa prosessia mahdollisimman hyvin. Kun kohdeyrityksen tuotekehityksen prosessia lähdettiin kehittämään, oli kohdeyrityksessä käytössä Microsoft Solutions Framework (MSF) for Agile -prosessimalli.

Vaikka MSF for Agile -prosessimalli perustuu ketteriin menetelmiin ja iteratiiviseen kehitysmalliin, se eroaa huomattavasti Scrum-menetelmästä. MSF for Agile -mallissa suunnitteluvaihe ja dokumentointi ovat hyvin suuressa roolissa eivätkä aivan yhtä joustavia kuin Scrum-menetelmässä. Eräällä tapaa MSF-mallia voidaan pitää vesiputousmallin ja Scrum-menetelmän välimallina ja se sisältää käytäntöjä molemmista malleista. (Resnick, de la Maza & Bjork 2011, 20 – 22.)

TFS:n projektinhallinta pohjautuu erityyppisten artefaktien käyttöön. Päätason artefaktit kuvaavat ohjelmiston ominaisuuksia ja ovat yleensä tuotepäällikön luomia. MSF for Agile -prosessimallissa näitä artefakteja kutsutaan käyttäjäkertomuksiksi (engl. User Story). Jokaisen User Storyn alle linkitetään ominaisuuden kehittämiseen ja testaamiseen liittyvät konkreettiset tehtävät (engl. Task), sekä muut niihin liittyvät asiat, kuten ohjelmistovirheet (engl. Bug) ja testitapaukset (engl. Test Case). Bug kuvaa ohjelmistovirhettä, jolloin ohjelmisto toimii eri tavalla kuin sen pitäisi. Testitapauksien (Test Case) avulla taas kuvataan miten tuotteen sisältämä ominaisuus testa-

taan. Testitapauksen avulla voidaan todentaa, että kyseinen ominaisuus toimii oikein. Usein yhteen User Storyyn liittyy useita taskeja, sekä testitapauksia.

Jokaiselle artefaktille, joka pitää sisällään konkreettista työtä, voidaan määrittää työmääräarvio. Työmääräarvio annetaan usein jäljellä olevan työmäärän muodossa. Arvio tarkentuu jatkuvasti, kun kehittäjä tai testaaja työskentelee tehtävän parissa. Käyttäjäkertomuksille työmäärä arvioidaan usein päivissä tai käyttäen keinotekoista yksikköä, jonka tarkoituksena on vain havainnoida ominaisuuden kokoa suhteessa muihin ominaisuuksiin. Tarkemman tason konkreettisille tehtäville työmäärä arvioidaan usein tunneissa. Annettua arviota tulee tarkentaa koko ajan, jolloin koko projektin jäljellä oleva työmäärä myös tarkentuu.

Edellä mainittujen artefaktien avulla pystytään luomaan hierarkioita, joista nähdään helposti tuotteen kehityksen status halutulla tasolla. Tehtäville annetut työmääräarviot muodostavat jatkuvasti tarkentuvan kuvan koko projektissa jäljellä olevasta työmäärästä. Projektihallinnon eri ryhmissä työskentelevät ihmiset ovat kiinnostuneita eritasoisista asioista. Tuotepäälliköt ovat kiinnostuneita ainoastaan suuremmista kokonaisuuksista (User Story) ja niiden etenemisestä, kun taas tuotekehittäjät ovat useimmiten kiinnostuneita enimmäkseen konkreettisista tehtävistä (Task).

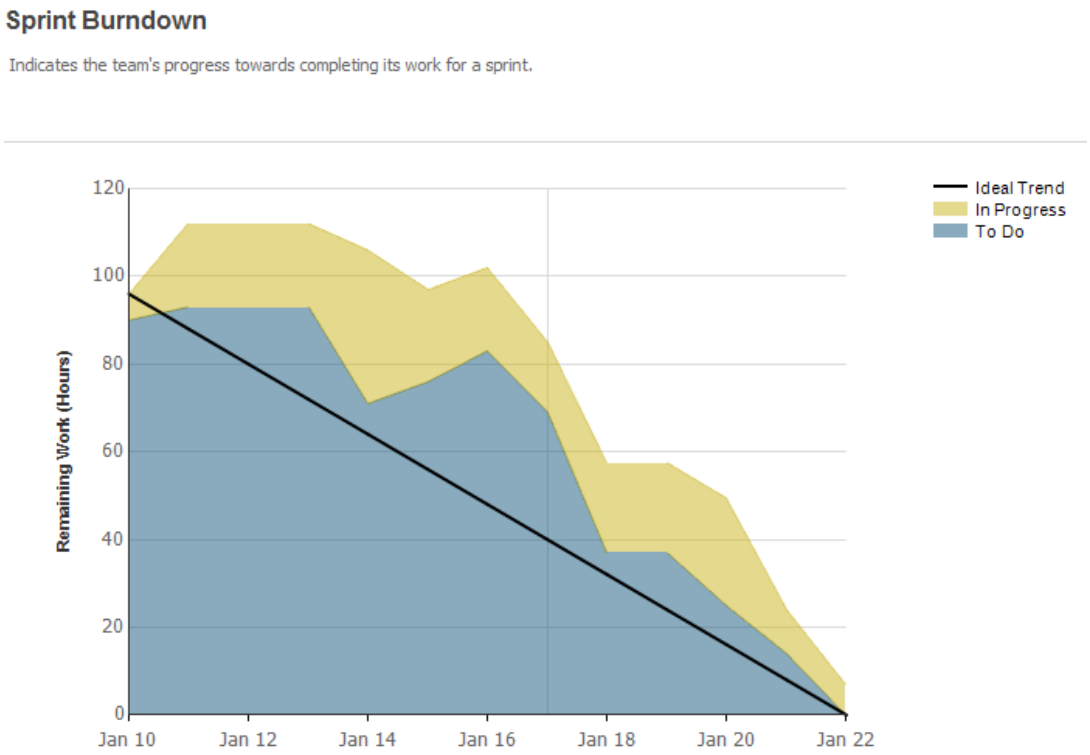
Kaikki TFS:n tarjoamat raportit pohjautuvat näiden artefaktien tietoihin ja tästä syystä on hyvin tärkeää, että artefaktien tiedot täytetään täydellisesti ja ne pidetään ajan tasalla. Jos artefaktien tiedoissa on puutteita, raportit muodostuvat epäluotettaviksi ja koko organisaatiolta katoaa läpinäkyvyys projektin etenemiseen.

3.4.3 Raportointi

TFS tarjoaa kattavat raportoinnin työkalut projektin seurantaan. Valmiita raportteja löytyy yleisimpiin tarkoituksiin ja raportteja voidaan luoda tarvittaessa lisää. Raportit saadaan ajastettua tiettyihin ajanhetkiin, jolloin raportit saadaan automaattisesti halutussa formaatissa valmiina tarkasteltavaksi. Raporttien muodostaminen manuaalisesti erityistarpeiden mukaan on myös mahdollista.

Kohdeyrityksessä käytössä ollut ketteriin menetelmiin pohjautuva prosessimalli perustui lyhyempiin toistuviin iteraatioihin. Iteraatioiden edistymisen seuraaminen on hyvin tärkeää prosessin hallinnan kannalta ja TFS tarjoaa tähän tarkoitukseen oman Burndown Chart -raportin (kuvio 1). Tämä raportti on selkeästi yksi tärkeimmistä raporteista koko prosessin kannalta. Raportti näyttää tiimin edistymisen yhden iteraation sisällä ja se pohjautuu TFS:n tehtäville (Task) annetuille jäljellä olevan työn tuntiarvioille. On selvää, että jos tehtäviä ei ole määritelty oikein tai niiden tuntiarviot ovat puutteellisia, koko raportin pohja katoaa eikä siitä saadaan enää minkäänlaista hyödyllistä tietoa. Lisää iteraatioiden hallinnasta ja ketteristä menetelmistä luvussa 4.3.

Burndown Chart tarjoaa myös tietoa meneillään olevasta työstä näyttäen graafisena kuvaajana meneillään olevan työn määrän. Tämän kuvaajan tulisi olla koko iteraation ajan tasainen, mutta jos meneillään oleva työ kasvaa suhteessa valmistuneen työn määrään, tulee miettiä minkä vuoksi tiimi aloittaa uusia töitä ilman, että edelliset työt on saatu valmiiksi. (Resnick, de la Maza & Bjork 2011, 130.)

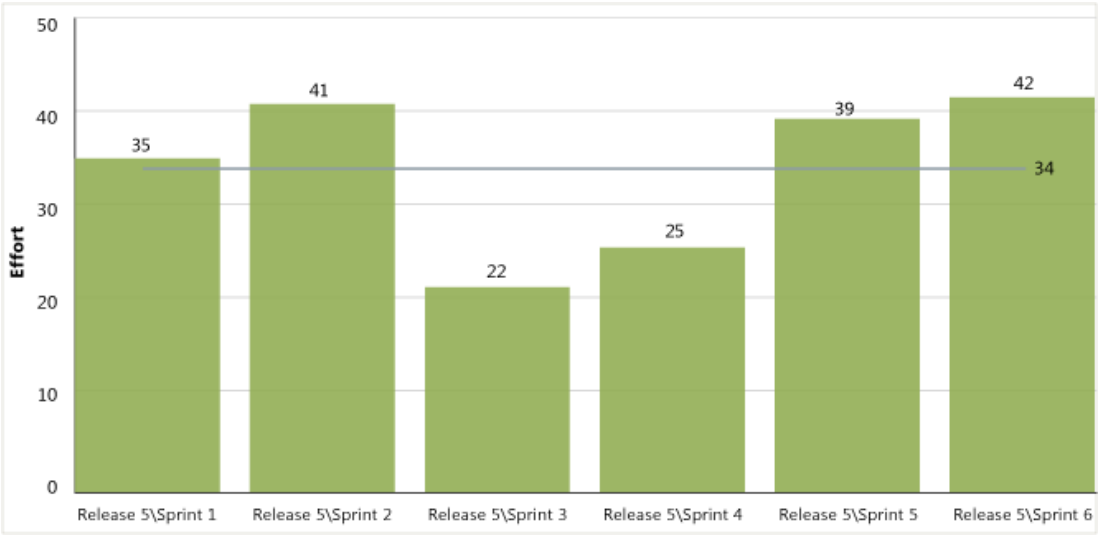


Kuvio 1. Iteraation Burndown Chart -raportti (MSDN keskustelupalstan www-sivut 2013)

Burndown Chart ei ole ainoa raportti, jonka TFS tarjoaa valmiina. Yleisimmin seurattuihin raportteihin kuuluvat myös koko tuoteversion edistymistä kuvaava Release Burndown -raportti, sekä tiimin suorituskykyä kuvaava Velocity-raportti.

Release Burndown -raportti on samankaltainen kuin Sprint Burndown -raportti, mutta tarjoaa näkymän koko tuoteversion kehitykseen, kun taas Sprint Burndown -raportti näyttää edistymisen vain yhden iteraation sisällä. Kun Sprint Burndown -raportti on tarkoitettu ensisijaisesti kehitystiimin käyttöön, Release Burndown -raportti on enemmänkin tuote- ja projektihallinnalle tarkoitettu raportti koko tuotteen kehityksen tilanteesta.

Velocity-raportti näyttää kuinka paljon tiimi sai valmiiksi kussakin iteraatiossa (kuvio 2). Tällä voidaan seurata tiimin suorituskykyä, mutta sitä voidaan myös käyttää arvioimaan kuin paljon tiimi saa aikaiseksi tulevilla sprinteillä. Historiatieto auttaa hyvin paljon seuraavien iteraatioiden suunnittelussa edellyttäen, että tiimin rakenne ja iteraation pituus pysyvät samoina. (Resnick, de la Maza & Bjork 2011, 130.)



Kuvio 2. Velocity-raportti (MSDN:n www-sivut 2013)

Velocity-raportin tarjoamaa tietoa voidaan myös käyttää hyväksi arvioidessa mahdollista tuotteen valmistumispäivää. Kun tiedetään tekemättömän työn määrä ja tiimin työskentelynopeus, voidaan jokaisen iteraation jälkeen tarkentaa arviota tuotteen valmistumisesta.

3.5 PLM-prosessi

Tässä opinnäytetyössä kehitettävä tuotekehityksen prosessi vaikuttaa ohjelmistoprojektissa vain sen kehitysvaiheeseen. Ohjelmistoprojektissa on muitakin työvaiheita, jotka kuvataan ja käsitellään kohdeorganisaation Project Lifecycle Management (PLM) -prosessikuvauksessa. Projektin elinkaari pitää sisällään kaikki työvaiheet koko organisaation kannalta aina projektin vaatimusten määrittelemisestä sen ylläpitoon saakka.

Myös PLM-prosessi on vain yksi osa suurempaa prosessien kokonaisuutta. Kaikki alkaa asiakastarpeista ja niiden täyttämisestä ja päämääränä on asiakastyytyväisyys. Asiakastyytyväisyys syntyy siitä, kun asiakkaalle pystytään toimittamaan valmis ja toimiva palvelu, joka vastaa heidän tarpeisiinsa.

PLM-prosessi pitää sisällään kolme erillistä aliprosessia. Vaatimusten määrittelyn, roadmappingin, sekä itse tuotekehityksen. Vaatimusten määrittelyn prosessi kuvaa organisaation tasolla miten ideat ja ajatukset uusista palveluista kerätään, analysoidaan ja priorisoidaan yhteisellä ja standardoidulla tavalla.

Roadmapping tarkoittaa tuotteen tulevaisuuden suunnittelua. On tärkeää, että jokaisella yrityksen tuotteella on suunnitelma siitä, mihin suuntaan tuotetta halutaan kehittää. Roadmapping tehdään yleensä hyvin yleisellä tasolla, eikä se ole lupaus asiakkaalle tulevista ominaisuuksista, vaan sen hetkinen ajatus siitä, mihin suuntaan tuote on menossa. Roadmapping-prosessi kuvaa miten tuotepäällikön tulee valmistella tuotteensa roadmap ja miten se hyväksytään liiketoimintayksiköissä.

Kolmantena aliprosessina on tuotekehitysprosessi, jonka osana myös tässä opinnäytetyössä kehitettävä prosessi toimii. Tuotekehitysprosessi kuvaa tuotekehityksen kuitenkin korkeammalla tasolla, kuin mitä tässä opinnäytetyössä kehitettävä prosessi tekee. Kehitettävä prosessi on laajemmassa tuotekehitysprosessissa vain yksi alikohta. Näin ollen näillä kahdella prosessilla ei ole päällekkäisyyksiä vaikka molempia kutsutaankin tuotekehityksen prosesseiksi. On kuitenkin tärkeää, että molemmat prosessit tukevat toisiaan, koska ne toimivat toistensa kanssa. Koska näitä kahta proses-

sia kehittää kaksi erillistä tahoa, on tärkeää pyrkiä toimimaan mahdollisimman paljon yhdessä ja ottamaan huomioon mahdolliset muutokset toisessa prosessissa.

Kohdeorganisaatiossa ohjelmistoa kehitetään versioittain. Jokaisen version kehittäminen tehdään omana tuotekehitysprojektina. Projektissa noudatetaan kehitettävää tuotekehityksen prosessia. Projekti saa alkunsa ominaisuuksien määrittelyllä. Kunkin tuotteen tuotepäällikkö määrittelee roadmapin perusteella ominaisuudet, jotka hän haluaa seuraavaan versioon. Roadmapin ulkopuolelta versioon halutaan usein myös asiakasprojekteista tulleita vaatimuksia. Tuotepäällikön lisäksi myös tuotekehitys määrittelee versiolle omat tekniset muutoksensa, jotka se haluaa tehdä, jotta tuote pystytään säilyttämään teknisesti ajan tasalla.

Kaikista näistä vaatimuksista muodostetaan priorisoitu lista kehitettävistä asioista, jonka pohjalta pystytään muodostamaan projektisuunnitelma. Kun projektisuunnitelma on hyväksytty, voidaan tuotteen kehittäminen aloittaa. Kehittäminen tapahtuu tässä opinnäytetyössä kehitettävän prosessin mukaisesti. Kun tuotekehitys on valmis ja tuotteen hyväksyntätestaus on onnistunut, voidaan tuote siirtää eteenpäin muille yksiköille. Tuotekehitysprosessi päättyy siihen, kun tuote on onnistuneesti toimitettu asiakkaalle.

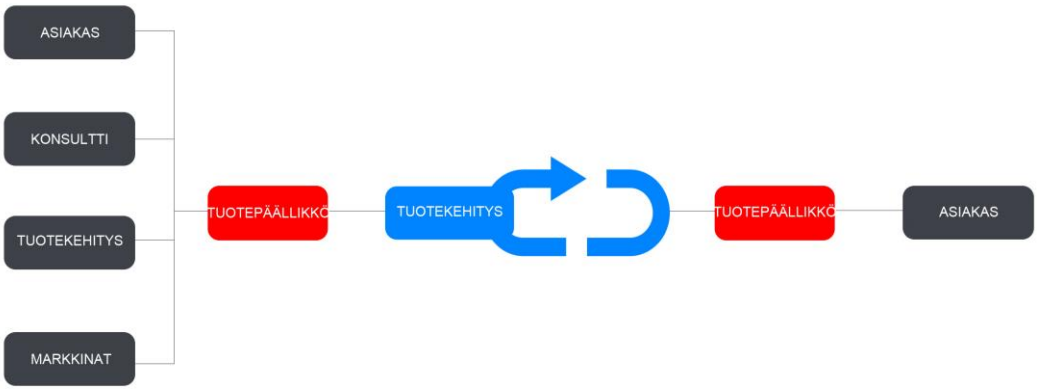
On tärkeää huomata, että osana PLM-prosessia määritelty tuotekehitysprosessi kattaa monen eri yksikön toimintoja. Tuotekehitysyksikköä koskee ainoastaan tuotteen kehittäminen ja konkreettinen tuotekehitys, joka suoritetaan tuotekehitysprosessin mukaisesti. Tämä on yksi syy siihen, että tässä opinnäytetyössä kehitettävä tuotekehitysprosessi on rajattu vain tähän osaan laajempaa PLM-prosessia. Kohdeorganisaatiossa ei tämän opinnäytetyön tiimoilta ole mahdollisuuksia määritellä laajempaa PLM-prosessia tai muuttaa sen osaprosesseja. Myös prosessin jalkauttaminen hankaloituisi huomattavasti, kun puhutaan koko organisaatiota ja eri liiketoimintayksikköjä kattavista prosesseista. Tässä opinnäytetyössä kehitettävä prosessi koskee ainoastaan yhtä yksikköä, jolloin sen onnistunut jalkauttaminen ja käyttöönotto ovat myös opinnäytetyön kannalta mahdollista.

Opinnäytetyön kohdeyksikköä ollaan vähitellen integroimassa osaksi suuremman organisaation prosesseja ja tästä syystä koko PLM-prosessin jalkautus on vielä hyvin

alkuvaiheessa. Kohdeyksikön kannalta muutoksia on hyvin paljon ja niiden ottaminen käytäntöön ja onnistunut jalkauttaminen tulee olemaan hyvin haasteellista. Vaikka PLM-prosessin käyttöönotto ei kuulukaan tämän opinnäytetyön piiriin, tulee se omalta osaltaan hankaloittamaan myös tässä opinnäytetyössä kehitettävän prosessin jalkauttamista.

3.6 Tuotekehitysprosessi ja sen ongelmat

OpusCapitan tuotekehityksellä ei ole ollut ennen tämän opinnäytetyön aloittamista yhtä yhtenäistä mallia, jonka mukaan kaikki tuotekehitystiimit toimisivat. Jokainen tiimi on sopinut keskenään ne pelisäännöt joiden mukaan toimitaan. Osa tiimeistä on noudattanut ketteriin menetelmiin perustuvaa iteratiivista kehitysmallia (kuvio 3) ja toisissa tiimeissä tuotetta kehitettiin ilman minkäänlaista määriteltyä prosessia. Vaikka tuotteiden kehitys ei ole täysin epäonnistunut, niin yrityksellä on ollut suuria ongelmia pitäytyä määritellyissä aikatauluissa. Tuotteiden valmistumista ei ole kyetty seuraamaan ja niitä ei ole pystytty viimeistelemään versioprojektille määritellyssä aikataulussa.



Kuvio 3. Kohdeyrityksen tuotekehitysprosessi

Tuotehallinta määritteli tuotteen uudet ominaisuudet. Tuotehallinta sai toivomuksia uusista ominaisuuksista eri lähteistä; asiakkailta, konsulteilta, myyjiltä ja tuotekehityksestä. PLM-prosessin mukaisesti tuotehallinnan tehtävä oli määritellä ominaisuudet tarkemmalla tasolla ja priorisoida ne niin, että tuotetta kehitettiin kokonaisuuden kannalta järkevään suuntaan ja tuotekehitykseen tehdyistä investoinneista saataisiin

tulevaisuudessa mahdollisimman paljon tuottoa. Tämän jälkeen tuotehallinta kävi ominaisuuksia lävitse tuotekehityksen kanssa, jotta niiden määrittely saatiin tarkennettua riittävälle tasolle. Koko tuotekehitystiimillä tuli olla yhtenäinen näkemys siitä mitä oltiin kehittämässä. Kun uusi ominaisuus oli määritelty tarpeeksi tarkasti, voitiin sen kehitystyö aloittaa.

Riippuen käytössä olevasta prosessista tai sen puutteesta, uusia ominaisuuksia tuotiin tuotekehitykselle joko virallisemmin iteraation suunnittelupalaverissa tai epävirallisemmin ilman määriteltyä ajankohtaa. Tämän jälkeen ohjelmistokehittäjä ohjelmoi uuden ominaisuuden, jonka toimivuuden testaaja varmensi. Lopulta tuotehallinta katselmoi ominaisuuden ja antoi sille lopullisen hyväksynnän.

Vaikka osassa tuotekehitystiimejä oli käytössä iteratiivinen kehitysmalli, se ei prosessina eronnut juuri lainkaan niiden tiimien prosesseista, joissa ei käytetty iteratiivista kehitysmallia. Iteratiivista prosessia käyttävät tiimit eivät useinkaan onnistuneet kehittämään uusia ominaisuuksia iteratiivisesti, vaan heidän työskentelynsä oli hyvin samanlaista muiden tiimien kanssa. Merkittävin ero oli, että heidän työtään hallittiin iteratiivisesti.

3.6.1 Tuotekehityksen vaiheet

Iteratiivisen kehitysmallin päätavoite on säilyttää tuotteen laatu koko kehityksen ajan. Tuotteen tulee olla toimitettavissa asiakkaalle jokaisen iteraation jälkeen. Koska kohdeyrityksessä ei ole täysin onnistuttu tällaisen iteratiivisen kehitysmallin käytössä, on tuotteen kehitys jouduttu jakamaan eri vaiheisiin. Eri vaiheiden tarkoituksena on ilmaista missä vaiheessa uuden version kehitys on. Eri vaiheisiin pätee erilaiset säännöt. Sääntöjen tarkoituksena on parantaa lopullisen tuotteen laatua. Yrityksessä on myös pyritty noudattamaan tiettyjä rajoituksia, minkä perusteella kehityksessä on voitu siirtyä seuraavaan vaiheeseen.

Tiettyjen vaiheiden jälkeen versioiden sisältämä lähdekoodi eriytetään versiohallinnassa omiksi haaroikseen (engl. branch). Eriyttämisen yhteydessä koko versiohallinnan sisältö kopioidaan tietyllä ajanhetkellä toiseksi haaraksi. Uusi haara syntyy siis

aina jonkin toisen haaran kopiona, mutta sen jälkeen uudella haaralla ei ole yhteyttä alkuperäiseen haaraan. Kun versiot sijaitsevat versiohallinnassa omissa haaroissaan, voidaan niihin tehdä muutoksia ilman, että muutokset vaikuttavat toisiin versioihin. Eri haaroihin tehdyt muutokset voidaan sisällyttää myös toiseen haaraan ja haarat voidaan myös sulauttaa myöhemmin toisiinsa. (Somasundaram 2013, 125–126.)

Kohdeyrityksessä uusi versio kopioidaan erilliseksi haaraksi, kun edellinen versio saavuttaa Release Candidate (RC) -vaiheen. Kun uusi versio sijaitsee versiohallinnassa omana haaranaan, voidaan sen kehitys aloittaa ilman, että tehtävät muutokset vaikuttavat edelliseen versioon. Tämä on tärkeää, koska versiot ovat eri kehitysvaiheissa (kuvio 4). Uuteen versioon aletaan kehittää uusia ominaisuuksia, kun taas vanhempi versio pyritään stabiloimaan mahdollisimman vakaaksi. Näin ollen edelliseen versioon ei voida enää lisätä uusia ominaisuuksia, jotka on tarkoitettu vasta seuraavaan versioon. Vanhemman version ohjelmistovirheitä voidaan myös korjata erikseen ilman, että ne vaikuttavat uuteen versioon.

Kuvio 4. Ohjelmistoversion vaiheet

Ohjelmisto voidaan julkaista monella eri tavalla ja julkaisu voi olla sisäinen tai julkinen. Kehitettävän ohjelmiston julkaisut tulisi suunnitella ja olla osa version kehitysprojektia. Ohjelmiston julkaisusuunnitelman tulee tukea yrityksen strategiaa ja julkista imagoa. Jos yrityksen tuotteet ovat tunnettuja korkeasta laadusta, myös niiden julkaisusuunnitelman tulisi tukea korkean laadun saavuttamista. Jos taas yritys on tunnettu nopeasta julkaisemisesta, tulisi suunnitelman tukea nopeita julkaisuja. Eri julkaisuille tulee suunnitella aikataulu, joka voidaan kommunikoida muulle organisaatiolle ja tietyllä tasolla myös asiakkaille. (Testa 2009, 110.)

Kohdeyrityksessä julkaistaan sisäisesti useita versioita päivässä. Jokainen ohjelmamuutos aiheuttaa automatisoidun prosessin, jonka seurauksena muodostuu uusi ver-

siopaketti. Näitä versioita ei julkaista yrityksen ulkopuolelle, vaan niitä käytetään pääasiassa tuotteen laadunvarmennuksessa. Yritys tekee julkisen tuotejulkaisun yleensä kaksi kertaa vuodessa ja sen julkaisuprosessi eroaa merkittävästi sisäisistä julkaisuista.

Kohdeyrityksessä uuden version kehityksen ensimmäinen tekninen vaihe on ns. alpha-vaihe. Alpha-vaiheessa tuotteeseen kehitetään uusia ominaisuuksia. Versioon tehtävillä muutoksilla ei ole alpha-vaiheessa erityisiä rajoituksia. Alpha-vaiheessa tuotteesta tulee useita uusia versioita päivässä, joita kutsutaan alpha-versioiksi.

Alpha-julkaisu on yleensä hyvin aikainen julkaisu vain muutamalle asiakkaalle, jotka voivat testata ohjelmiston konsepteja ja ominaisuuksia. Alpha-julkaisu ei ole ominaisuuksiltaan täydellinen, eikä kaikkia toteutettuja ominaisuuksia ole vielä testattu. (Testa 2009, 113.)

Alpha-versiot sisältävät testaamattomia ja keskeneräisiä ominaisuuksia ja tästä syystä alpha-versiot ovat yleensä vain yrityksen sisäisessä käytössä. Alpha-vaihe on selkeästi pisin vaihe tuotteen kehityksessä. Vaiheen pituus vaihtelee eri versioiden kesken, mutta yleensä vaiheen pituus on n. 75–80 % koko projektin kestosta.

Kun ohjelmistoon on kehitetty kaikki tarvittavat ominaisuudet, voidaan siirtyä beta-vaiheeseen. Beta-vaiheeseen siirryttäessä kaikki tuotteen ominaisuudet tulisi olla kehitettynä, mutta niiden ei tarvitse vielä olla täydellisesti testattuja. Beta-vaiheen aikana ohjelmistoon tehtäviä muutoksia rajoitetaan niin, että vain testauksen aikana löydettyjen ohjelmistovirheiden korjaus on sallittua. Beta-vaihe ei välttämättä eroa käytännössä kovinkaan merkittävästä edeltävästä alpha-vaiheesta, eikä siihen siirtyminen aiheuta erillisiä toimenpiteitä. Voidaankin todeta, että beta-vaihe on vain nimellinen vaihe ohjelmiston kehityksessä, jonka aikana pyritään viimeistelemään uusien ominaisuuksien kehitys. Vaiheen aikana aloitetaan myös tuotteen stabilointi estämällä uusien ominaisuuksien kehityksen aloittaminen.

Beta-julkaisu pitää sisällään kaikki suunnitellut ominaisuudet ja se julkaistaan yleensä muutamille asiakkaille, joiden tarkoitus on testata ja antaa palautetta toteutetuista ominaisuuksista ja ohjelmiston toimivuudesta. Koska julkaisu on valmiimpi kokonai-

suus, asiakas voi käyttää sitä laajemmassa mittakaavassa ja tarjota parempaa palautetta. (Testa 2009, 113.)

Viimeinen julkaisu ennen lopullista julkaisua on yleensä rajoitettu julkaisu muutamille läheisille asiakkaille. Julkaisun tarkoituksena on paljastaa ohjelmistosta mahdolliset vakavat ongelmat ennen, kuin se julkaistaan julkisesti kaikille. (Testa 2009, 113.)

Kohdeyrityksen viimeinen julkaisu ennen lopullista massajakelua on Release Candidate (RC) -versio. RC-vaihe on huomattavasti edellistä beta-vaihetta merkittävämpi vaihe tuotteen kehityksessä. Jotta RC-vaiheeseen voidaan siirtyä, tulisi ohjelmiston olla tarpeeksi laadukas lopullista julkaisua varten. Vaiheen nimen perusteella, kyseessä on eräänlainen kandidaatti ohjelmiston julkaisua varten. Ohjelmistossa ei saisi enää olla merkittäviä virheitä ja tarpeeksi suuri osa ohjelmiston toiminnallisuuksista tulisi olla myös testattu, jotta voidaan olla tarpeeksi varmoja, että ohjelmistoa voidaan toimittaa asiakkaille. RC-vaiheessa ohjelmistoon tehtäviä muutoksia rajoitetaan yhä enemmän. Jokainen muutos kirjataan ylös, jotta versioon tehtäviä muutoksia voidaan seurata yhä tarkemmin.

Kun kehitetään uusia tuotteita, törmätään usein ongelmiin, joita kukaan ei ole tullut ajatelleeksi. Joskus myös ongelmat, joita on pidetty vakavina osoittautuvatkin harmittomiksi. Aina törmätään johonkin odottamattomaan, johon ei ole kyetty valmistautumaan etukäteen. Tätä varten myös ohjelmistotuotannossa uusia versioita pilotoidaan eli koekäytetään. Koekäytöllä pienennetään uuden tuotejulkistuksen riskiä ja varmistetaan, ettei uuden tuotteen kanssa tule odottamattomia ongelmia kun sitä käytetään tosielämän ympäristössä tuotantokäytössä. (Drucker 1999, 105 – 106.)

Kun ohjelmistoversio saavuttaa RC-vaiheen, se annetaan muulle organisaatiolle rajoitettuun jakeluun. Rajoitetussa jakelussa uutta versiota pilotoidaan muutamalla asiakkaalla, jotta nähdään, toimiiko uusi versio täysin todellisissa asiakasympäristöissä. Jos suuria ongelmia ei havaita, voidaan ohjelmistoversiolla aloittaa massajakeluvaihe.

Massajakelun alkaessa, ohjelmisto siirtyy lopulta sen viimeiseen, Release-vaiheeseen. Release-vaiheessa ohjelmisto julkaistaan massajakeluna kaikille asiakkaille ja muulla organisaatiolla on lupa toimittaa sitä haluamilleen asiakkaille. Yleensä viimeistään tässä vaiheessa yrityksen tuotekehitys on jo aloittanut seuraavan version kehittämisen ja siirtynyt seuraavan version alpha-vaiheeseen.

3.6.2 Projektinhallinnan kolme rajoitusta

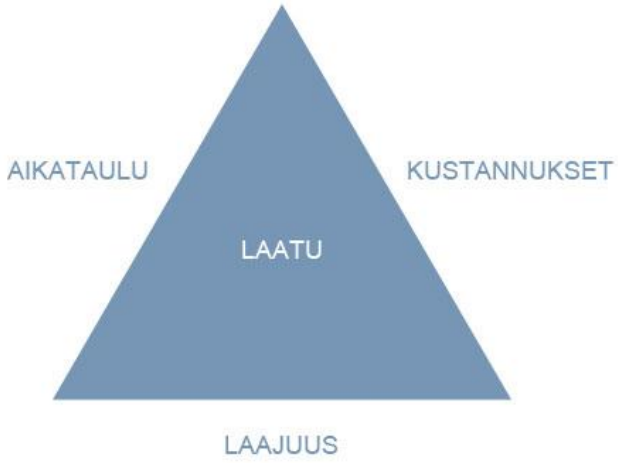
OpusCapitassa jokainen uusi tuoteversio toteutetaan erillisenä projektina. Projektille määritellään resurssit, aikataulu, sisältö ja budjetti. Budjetti ja resurssit liittyvät tiiviisti yhteen, koska lähes kaikki projektin kustannuksista muodostuu henkilöstökuluista. Budjettia myös usein seurataan toteutuneiden työtuntien muodossa. Tästä syystä budjetti ja resurssit ovat myös vähiten merkityksellisiä kohdeyrityksen projekteissa. Pääasiassa projekteissa keskitytään sen aikatauluun ja sisältöön. Projektille määritellään haluttu aikataulu, jonka perusteella yritetään arvioida mahdollisimman tarkasti, kuinka paljon uusia ominaisuuksia tuotteisiin ehditään nykyisillä resursseilla kehittämään.

Projektinhallinnan onnistumista kuvataan yleensä projektinhallinnan kolmiolla (kuvio 5). Kolmion mukaan projektin onnistuminen mitataan sillä, kuinka hyvin projektiryhmä onnistuu hallitsemaan projektia niin, että projektille määritellyt tavoitteet saavutetaan määritellyssä aikataulussa ja määriteltyjen kustannuksien puitteissa. Kolme projektinhallinnan rajoitusta kuvataan kolmiona, jonka sivuina ovat projektin kustannukset, laajuus ja aikataulu. (Newell & Grashina 2003, 8.)

Kolmion keskelle lisätään usein neljäs ulottuvuus; asiakastyytyväisyys. Kaikki kolme rajoitetta vaikuttavat yhdessä asiakastyytyväisyyteen. Saavuttaakseen hyvän asiakastyytyväisyyden, tulee toimittajan pysyä aikataulussa, sekä toimittaa kaikki sovitut ominaisuudet sovitussa aikataulussa. (Newell & Grashina 2003, 8.)

Ohjelmistoprojektin näkökulmasta kolmion keskelle kuitenkin nostetaan usein asiakastyytyväisyyden sijaan laatu. Ohjelmistoprojektin onnistuminen ei välttämättä takaa vielä asiakkaan tyytyväisyyttä, mutta projektin kolme rajoitetta vaikuttaa suoraan

toteutuvaan laatuun. Kohdeyrityksen tuotekehitys kehittää yrityksen sisäistä tuotetta, eikä tuotekehitysprojektissa ole suoraan mukana ulkoista asiakasta. Myös tästä syystä on perusteltua nostaa laatu keskelle projektikolmiota. Toisin, kuin muiden rajoitteiden kohdalla, laadun tulisi olla kiinteästi asetettu, eikä projektiryhmän tulisi sallia toteutuvan laadun heikentyä.



Kuvio 5. Projektikolmio mukailtuna (Newell & Grashina, 2003)

Projektinhallinnan kolmion mukaan, yhtäkään kolmesta rajoituksesta ei voida muuttaa ilman, että se vaikuttaa muihin rajoituksiin. Jos asiakas haluaa projektin toteutettavan puolessa alkuperäisestä ajasta, joudutaan luultavasti projektin laajuutta pienentämään tai lisäämään projektiin resursseja, jolloin taas projektin kustannukset nousevat. Jos vastakkaisia rajoituksia ei muuteta, ohjelmistoprojektin tapauksessa vain laatu voi enää muuttua. Projektiryhmän ja projektipäällikön tärkeimpänä tehtävänä onkin tasapainotella projektissa näiden kolmen rajoitteen välillä. (Newell & Grashina 2003, 10.)

Versioprojektin aikataulu on kohdeyrityksessä tärkein kolmesta rajoituksesta. Aikataulu on usein määritelty jo kauan ennen projektin alkamista. Aikataulu kommunikoidaan myös asiakkaille, joka tekee sen jälkikäteen muuttamisesta hyvin vaikeaa. Asiakkaat haluavat myös tietää tulevista ominaisuuksista ja osa projektin sisältämistä ominaisuuksista voi myös olla mukana asiakkaan allekirjoittamissa sopimuksissa. Kun asiakkaille on luvattu tiettyjä ominaisuuksia tietyssä aikataulussa, on ajauduttu

hyvin hankalaan tilanteeseen, jossa projektinhallinnan kolmen rajoitteen mukaan ainoat muuttuvat asiat voivat enää olla kustannukset sekä laatu.

Pääosa kohdeyrityksen ohjelmistoprojektien kustannuksista ovat projektin henkilöstöresurssit. Kun projektin laajuus ja aikataulu on kiinnitetty, voidaan enää tarvittaessa muuttaa kustannuksia eli projektin resursseja. Ohjelmistoprojektin resurssien lisääminen ei kuitenkaan ole aina mahdollista. Kohdeyrityksen monimutkainen ohjelmisto ja vaativa aihealue tekevät resurssien lisäämisen nopealla aikataululla lähes mahdottomaksi. Ohjelmistoprojekteissa on myös hyvin tiedossa, että kasvattamalla projektin tiimien kokoa, saadaan aikaiseksi lähes päinvastainen vaikutus aikataulun viivästyessä yhä enemmän.

Fred Brooks esitti kirjassaan ”The Mythical Man-Month” vuonna 1975, että jos myöhässä olevaan ohjelmistoprojektiin lisätään henkilöitä, se tulee olemaan yhä enemmän myöhässä. Tämä lause tunnetaan nykyään Brooken lakina. Lisätyt henkilöt hajottavat olemassa olevat tiimit ja aiheuttavat näin projektin myöhästymisen entistä enemmän. Toimivin tapa lisätä resursseja on lisätä kokonaisia tiimejä. Uudet tiimit aiheuttavat myös projektille uusia haasteita lisääntyneen kommunikoinnin ja yhteistyön muodossa, mutta vaikutus ei ole yhtä suuri, kuin jos tiimien kokoa vain kasvatettaisiin. (Goodpasture 2010, 183.)

3.6.3 Ongelmat määrittelyssä

Uuden ominaisuuden määrittely kuuluu kohdeyrityksessä tuotehallinnan vastuisiin. Tuotepäällikkö on ainoa henkilö, joka on lopulta vastuussa tuotteesta. Tuotepäällikköllä tulee olla visio tuotteen kehityssuunnasta ja hänen pitää pystyä kommunikoimaan tuo visio tuotekehitykselle, sekä muille sidosryhmille. Tuotepäällikkö huolehtii siitä, että tuotetta kehitetään vision suuntaisesti ja että tuotteeseen kehitetään aina suurinta lisäarvoa tuovia ominaisuuksia. (Viscardi 2013, 19.)

Tuotteen ominaisuusmäärittely kuvaa liiketoiminnallisen ongelman ja ohjelmistoratkaisun vaatimukset käyttäjän ja liiketoiminnan näkökulmasta. Kuvauksen tulee olla

täsmällinen ja yksityiskohtainen, jotta tuotekehitys tietää sen perusteella mitä heidän tulee tehdä ja miten he voivat arvioida työnsä tuloksen. (Rinzler 2009, 6.)

Ominaisuuden määrittely tehdään kohdeyrityksessä lähes aina kirjallisesti. Tuotepäällikön määrittely halutusta ominaisuudesta tulisi olla liiketoiminnan näkökulmasta, eikä saisi ottaa kantaa ominaisuuden tekniseen toteutukseen. Ominaisuuden tekninen toteutus on tuotekehityksen päätettävissä. Tuotepäällikkö on vastuussa siitä mitä ja miksi tehdään, kun taas tuotekehitys vastaa siitä miten kyseinen ominaisuus toteutetaan. Liiketoiminnallisen määrittelyn lisäksi ominaisuudesta voidaan tehdä teknisiä määrittelyjä, mutta nämä määrittelyt tehdään yhdessä tuotekehityksen kanssa ja ne toimivat pääasiassa työn dokumentoinnin tukena.

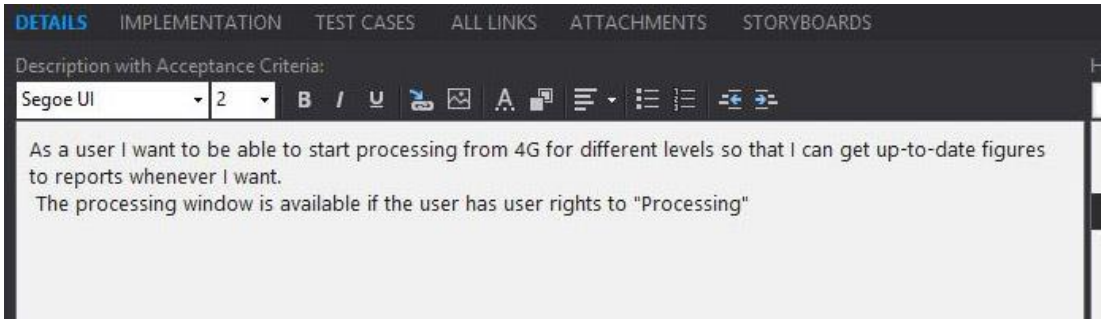
Tuotepäällikön määrittely voi esimerkiksi olla, että uuden käyttäjän tulee pystyä rekisteröitymään ohjelmiston käyttäjäksi eli luomaan itselleen uuden käyttäjätilin järjestelmään. Tuotepäällikkö voi määritellä tämän lisäksi, että käyttäjän tulee pystyä syöttämään sähköpostiosoitteensa sekä nimensä ja antamaan käyttäjäprofiililleen salasanan.

Kun tuotekehitys alkaa toteuttaa edellä mainittua ominaisuutta, herää helposti lisäkysymyksiä: miltä rekisteröitymissivun tulee näyttää? Kuinka pitkän nimen käyttäjä saa määritellä? Onko annetulla salasanalla, jotain vaatimuksia pituuden tai monimutkaisuuden suhteen? Mitä tapahtuu, jos käyttäjä jättää osan kentistä tyhjiksi? Mihin käyttäjä ohjataan, kun hän on täyttänyt lomakkeen? Pitääkö sähköpostiosoitteen todellisuus varmentaa?

Edellä oli vain muutamia kysymyksiä, joita alkuperäinen määrittely jättää auki. Kaikki nämä kysymykset vaikuttavat ominaisuuden toteuttamiseen ja lopulta koko ohjelmiston käyttökokemukseen. Nämä kysymykset vaikuttavat myös ominaisuuden työmäärään. Useimmat uudet ominaisuudet ovat kuitenkin monimutkaisuudeltaan ja vaikutuksiltaan niin laajoja, ettei kukaan pysty etukäteen ottamaan huomioon kaikkia tilanteita, joista monet tulevat esille vasta siinä vaiheessa, kun ominaisuutta toteutetaan.

Esimerkkinä voidaan tarkastella kohdeyrityksessä vuonna 2012 toteutettua ominaisuutta. Ominaisuuden määrittelyssä kuvataan, että käyttäjän tulee pystyä aloittamaan prosessointi ja prosessointi-ikkunan tulee olla saatavilla, jos käyttäjällä on käyttöoikeus prosessointiin (kuva 1).

Vaikka ominaisuuden perusteet ovatkin kuvattuina, jättää määrittely hyvin paljon avonaisia kysymyksiä. Ominaisuuden määrittelyn täydentäminen jälkikäteen kasvattaa ominaisuuden kehitystyön määrää ja aiheuttaa hankaluuksia koko tuotekehitysprosessille. Myöhemmin tullaan huomaamaan miten puutteellinen määrittely kasvatti esimerkkinä olleen ominaisuuden kehitysaikaa.



Kuva 1. Esimerkki puutteellisesta määrittelystä

Tuotekehitys antaa työmääräarvionsa tuotepäällikön määrittelyn perusteella. Näiden työmääräarvioiden perusteella koko ohjelmistoprojektille luodaan alustava sisältö. Kohdeyrityksessä aikataulu on usein kiinnitetty ennen projektin sisällön määrittelemistä. Projektin suunnittelupalaverissa (Release Planning) on tarkoitus arvioida, mitkä ominaisuudet ehditään toteuttamaan määritellyssä aikataulussa. Suunnittelun aikana voidaan muokata tiimien koostumuksia ja priorisoida tuotteen ominaisuuksia uudelleen, jotta suunniteltavasta kokonaisuudesta saadaan järkevä ja toteuttamiskelpoinen.

Ominaisuuksien määrittely jätetään kohdeyrityksessä usein liian myöhäiseen vaiheeseen, jolloin kaikkien haluttujen ominaisuuksien työmäärä ei ole valmis arvioitavaksi projektin suunnittelupalaverissa. Projektin alkamista ei voida kuitenkaan viivästyttää, jolloin projekti aloitetaan puutteellisilla tiedoilla sen lopullisesta laajuudesta.

Työmääräarvioiden eli estimaatioiden suhde toteutuneeseen kalenteriaikaan on projektisuunnitelman kannalta ongelmallista. Tuotekehitys antaa työmääräarvionsa konkreettisena tuntimääränä. Työmäärä ei ole yhtä kuin kuluva kalenteriaika. Tuotekehitystiimin jäsenet tekevät paljon muutakin, kuin vain kehittävät uusia ominaisuuksia. Näitä toimintoja ovat esimerkiksi asiakastuki ja palaverit. Henkilöstä riippuen muihin aktiviteetteihin kuluu eri määrä aikaa. Kohdeyrityksessä käytetään tiettyä kapasiteettikerrointa kuvaamaan, kuinka paljon todellista työaikaa henkilöllä on käytössään uusien ominaisuuksien kehittämiseen. Ohjelmoijilla tämä kerroin on yleensä 0,7 – 0,8, kun taas testaajille se on usein alhaisempi 0,4 – 0,6. Kertoimen avulla saadaan henkilön todelliseksi työajaksi 3h – 6h päivässä. Kertoimen todenmukaisuutta ei seurata kovinkaan tarkasti, mutta sitä on mahdollista muuttaa, jos kerroin vaikuttaa olevan merkittävästi väärin. Kalenteriaikaan vaikuttaa myös ominaisuuden parissa työskentelevien henkilöiden määrä. Lopullista työmäärää ei kuitenkaan voida laskea vain jakamalla arvioitu työmäärä henkilöiden lukumäärällä. Lopullisen kalenteriajan arvioiminen annettujen työmääräarvioiden pohjalta on monen muuttujan summa ja parhaimmillaankin vain hyvä arvaus.

Puutteet määrittelyssä luovat epävarmuutta ominaisuuden toteutukseen, jolla on selkeä vaikutus koko projektin aikatauluun. Puutteet määrittelyssä huomataan usein vasta toteutusvaiheessa. Tuotepäälliköllä on aina päätäntävalta ja lopullinen vastuu omasta tuotteestaan ja tästä syystä vastaukset avonaisiin kysymyksiin tulee saada tuotepäälliköltä. Tuotepäällikön kiireiden vuoksi vastausta voidaan joutua odottamaan jopa päiviä. Tietyissä tapauksissa koko ominaisuuden toteutus pysähtyy, kunnes tuotepäälliköltä saadaan tarvittava vastaus. Tästä syystä tuotepäällikön tulisi olla lähes aina tavoitettavissa vastaamaan avoimiin kysymyksiin, jotka estävät tuotekehityksen etenemisen.

Yhden ominaisuuden toteuttamisessa on mukana useita henkilöitä. Tuotekehittäjä vastaa teknisestä toteutuksesta ja testaaja varmistaa, että ohjelmiston laatu on vaaditulla tasolla. Testaaja varmistaa tuotteen laadun kirjoitetun määrittelyn perusteella. Tuotepäällikkö voi usein kuvata haluamansa ominaisuuden sanallisesti tuotekehittäjälle tai hän voi vastata avonaisiin kysymyksiin sähköpostitse. Määrittelyjen hajanaisuudesta muodostuu ongelma, kun ominaisuuden parissa työskentelee useita henkilöitä. Jos vastaus on kerrottu sähköpostitse tuotekehittäjälle, ei testaaja välttämättä

tiedä, että näin on sovittu. Tästä syystä on hyvin tärkeää, että kaikki ominaisuuden määrittely tehdään kirjallisena yhdessä paikassa, josta kuka tahansa voi nähdä miten ominaisuus on määritelty ja mitä on sovittu tehtäväksi. Kirjallinen määrittely toimii myös usein tuotteen dokumentaationa ja on tärkeää, että tulevaisuudessa voidaan palata takaisin ja tarkistaa miten ominaisuus on alun perin määritelty toteutettavaksi. Kohdeyrityksessä ominaisuuksien määrittelyt kirjoitetaan Team Foundation Serverin (TFS) ominaisuuksia kuvaaviin artefakteihin (Product Backlog Item tai User Story).

Testaajien hyvän erityisosaamisen johdosta, he osaavat usein miettiä toteutettuja ominaisuuksia loppuasiakkaan näkökulmasta. Ominaisuuden määrittelyssä on voitu unohtaa jokin tilanne, jonka testaaja lopulta nostaa esille. Ongelmien pohjalta voidaan ominaisuuden määrittelyä joutua muuttamaan. Määrittelyn muuttaminen kehitystyön jälkeen aiheuttaa suunnittelematonta lisätyötä ja vaikuttaa negatiivisesti projektin aikatauluun. Pahimmassa tapauksessa muutos aiheuttaa suurempia muutoksia toteutettuun ominaisuuteen, jolloin suuri osa tehdystä työstä joudutaan tekemään uudelleen. Tällainen muutos voi moninkertaistaa ominaisuuden alkuperäisen työmääräarvion.

Muuttuvassa maailmassa on hyvin yleistä, että projektin aikana tulee tarve muuttaa version ominaisuuksia, joko lisäämällä kokonaan uusia ominaisuuksia tai muuttamalla jo toteutettuja ominaisuuksia vastaamaan paremmin asiakkaiden tarpeita. Kohdeyrityksessä nämä tarpeet tulevat usein meneillään olevista asiakasprojekteista, joissa asiakkaalle on luvattu jokin tietty toiminnallisuus eikä kyseisen asiakasprojektin aikataulu anna mahdollisuuksia siirtää kyseisen toiminnallisuuden toteuttamista seuraavaan ohjelmistoversioon. Asiakasongelmista seuraa usein myös suunnittelematonta lisätyötä, jota käsitellään enemmän luvussa 3.6.7. Eri lähteistä tulevien uusien vaatimusten vaikutuksia meneillään olevan ohjelmistoversioprojektin aikatauluun, ei yleensä kuitenkaan oteta huomioon.

3.6.4 Iteraatiopohjaisen työn puuttuminen

Vaikka kohdeyrityksen tuotehallinnan ominaisuusmäärittelyissä on edellä mainittuja ongelmia, voidaan ongelmakohtia löytää myös tuotekehityksestä. Kohdeyrityksen

tuotekehitys on jo pitkään pyrkinyt toimimaan iteraatioittain, ketterien menetelmien mukaisesti. Ongelmana on ollut, että itse tuotekehitystyötä ei ole kuitenkaan tehty iteraatiopohjaisesti.

Yksi iteraatiopohjaisen työskentelyn pääperiaatteista on, että tuote on jokaisen iteraation jälkeen valmis toimitettavaksi asiakkaalle. Jokaisen iteraation tulisi lisätä tuotteeseen jonkinlaista arvoa. Toimimalla tällä tavalla, mahdollistetaan ketterä työskentely, jossa kehityksen suuntaa voidaan muuttaa jokaisen iteraation jälkeen. Jos tuote on iteraation jälkeen keskeneräinen, ei suuntaa pystytä muuttamaan ja hyödyt iteraatiopohjaisesta kehityksestä menetetään lähes täysin.

Vaikka kohdeyrityksessä on noudatettu iteraatiopohjaista työskentelyä, ei työ itsessään ole tukenut tätä toimintamallia. Tuote on usein iteraation jälkeen jäänyt keskeneräiseksi, eikä sitä ole voitu edes teoriassa toimittaa asiakkaalle. Vaikka todellista tarvetta ketterälle suunnanmuutokselle on ollut hyvin harvoin, on tästä muodostunut ongelma versioprojektin loppuvaiheessa. Kun projektin valmistumisen määräaika on lähestynyt ja tuote pitäisi saada valmiiksi toimittamista varten, on huomattu, että ominaisuuksia on yhä kesken. Projektin viimeisteleminen ja päätösvaihe ovat kestäneet suunniteltua kauemmin, joka on usein johtanut projektien viivästymiseen.

Edellisessä luvussa esitelty ominaisuus on tästä hyvä esimerkki. Ominaisuuden kehittäminen aloitettiin 24.11.2011 alkaneella iteraatiolla. Ominaisuus siirrettiin yhteensä neljä kertaa seuraavalle iteraatiolle, koska sitä ei ollut saatu vielä valmiiksi. Ominaisuus saatiin kehitettyä valmiiksi ja siirrettyä testaukseen vasta 5.3.2012 eli neljä kuukautta sen aloittamisen jälkeen. Syitä ominaisuuden kehityksen viivästymiseen oli muitakin, mutta ominaisuus kasvoi liian isoksi kehitettäväksi yhden iteraation aikana. Tuote sisälsi koko tämän ajan kehitteillä olevan version ominaisuudesta, jota kukaan ei vielä voinut täysin käyttää. Ohjelmistoa ei olisi voitu toimittaa tänä aikana asiakkaalle.

Jotta tuotekehitys pystyisi todella toimimaan iteraatioittain, tulee sille määrätyn työn myös tukea iteraatiopohjaista työskentelyä. Tuotehallinnan määrittelemät ominaisuudet ovat usein laajuudeltaan huomattavasti suurempia, kuin mitä tuotekehitystiimi ehtii yhden iteraation aikana toteuttamaan. Tästä syystä, nämä ominaisuusmäärittelyt

tulisi pilkkoa pienemmiksi osakokonaisuuksiksi, joista jokainen ehditään toteuttamaan yhden iteraation aikana ja joista jokainen tuo tuotteeseen jonkinlaista lisäarvoa.

3.6.5 Regressiovirheiden suuri määrä

Ohjelmistotuotannossa tapahtuu aina virheitä. Ohjelmistovirheitä kutsutaan bugeiksi (engl. Bug). Useimmat virheistä on helppo havaita ja korjata, koska ne esiintyvät samassa yhteydessä kehitettävän ominaisuuden kanssa. Tällaiset virheet havaitaan usein jo ensimmäisissä testeissä, kun ominaisuuden laatua varmennetaan.

Regressiolla tarkoitetaan, että muutos ohjelmiston yhteen kohtaan on hajottanut ohjelmiston jostain muualta. Tutkimukset osoittavat, että jopa 50 % tehdyistä ohjelmistomuutoksista aiheuttaa uusia ohjelmistovirheitä. Tällaiset virheet aiheutuvat muutettavan kohdan interaktioista muiden ohjelman osien kanssa. Esimerkki regressiovirheestä voisi olla muutos tietokantamuuttujan tallennuksessa. Regressiovirhe voisi esiintyä kyseisen muuttujan lukemisessa toisessa osassa ohjelmistoa. (Horch 2003, 80.)

Regressiovirheiden esiintymistä ei pystytä myöskään ennustamaan. Kun ominaisuus on testattu, voidaan usein todeta, että siinä ei ole enää jäljellä kovin merkittäviä virheitä. Testaamattomien ominaisuuksien osalta voidaan ottaa huomioon, että niistä saattaa vielä löytyä testauksen aikana virheitä. Regressiovirheiden kohdalla tätä ei pystytä tekemään, koska niiden havaitseminen on hyvin hankalaa. Kehitettävän ominaisuuden yhteydessä havaittujen virheiden korjaus kuuluu kyseisen ominaisuuden kehittämiseen ja sille annettuun työmääräarvioon. Regressiovirheisiin ei pystytä tuotekehityksessä varautumaan ja tästä syystä niiden korjaaminen on ylimääräistä työtä, joka vie aikaa pois itse ominaisuuksien kehittämiseltä.

Regressiovirheiden suuri määrä on usein suoraan verrannollinen kehitettävän ohjelmiston monimutkaisuuteen. Monimutkaisissa ohjelmistoissa regressiovirheiden esiintymistä voidaan hillitä hyvällä teknisellä suunnittelulla ja automaatiotestauksella. Kohdeyrityksen ohjelmisto on hyvin monimutkainen ja monesta erillisestä järjestelmästä muodostuva kokonaisuus. Näiden erillisten järjestelmien välillä on paljon

yhteisiä komponentteja, joten monet ohjelmistomuutokset vaikuttavat useisiin eri ominaisuuksiin. Ohjelmistoa on kehittänyt sen historian aikana lukemattomia eri kehittäjiä ja vanhimmat ohjelmamoduulit ovat n. kymmenen vuotta vanhoja. Tällainen monimuotoisuus aiheuttaa suuren riskin regressiolle, kun nykyiset kehittäjät eivät välttämättä ole edes tietoisia mihin kaikkialle heidän tekemänsä muutos vaikuttaa.

3.6.6 Testauksen ylikuormittuminen

Yrityksen laadunvarmennus (QA, engl. Quality Assurance) varmentaa ohjelmiston toimivuuden ja laadukkuuden. Kun tuotekehittäjä toteuttaa ohjelmistoon uuden ominaisuuden tai korjaa ohjelmistovirheen, QA varmentaa, että tehty muutos toimii. Automaattisten testien lisäksi testaaja käyttää manuaalisesti ohjelmistoa ja varmentaa kaikkien mahdollisten käyttötapausten toimivuuden ja laadun. Manuaalinen testaus ja ohjelman läpikäyminen käsin on hyvin hidasta ja ohjelmiston laajuudesta johtuen vie hyvin paljon aikaa. On olemassa nyrkkisääntö, että testaukseen tuleekin varata yhtä kauan aikaa, kuin ominaisuuden kehittämiseen.

Kohdeyrityksen laadunvarmennus on suhteellisen pieni. Jokaisessa tuotekehitystiimissä on 1-2 testaajaa, joiden tehtävänä on varmentaa kyseisen tiimin vastuulla olevan tuotteen laatu. Tuotekehittäjiä tiimissä on yleensä 3-5. Kun huomioidaan edellä mainittu nyrkkisääntö, että testaukseen kuluu yhtä kauan aikaa, kuin kehittämiseen, tulisi kehittäjiä ja testaajia olla yhtä paljon.

Kehittäjät tuottavat siis enemmän testattavaa, kuin mitä testaajat ehtivät testata. Tästä johtuen testaajien työkuorma kasvaa ja testaamattomien ominaisuuksien määrä kasvaa. Testaaja työskentelee ominaisuuden parissa mahdollisesti hyvinkin kauan sen jälkeen, kun kehittäjä on sen saanut valmiiksi. Jos ohjelmistosta löytyy ominaisuuteen liittyvä virhe, palautuu työ takaisin kehittäjälle. Kehittäjän on hankala palata takaisin vanhaan työhön, koska sen tekemisestä on kulunut kauan aikaa. Virheiden korjaamiseen kuluva aika kasvaa näin entisestään.

Esimerkkinä käytetty ominaisuus vuodelta 2012 toimii hyvänä esimerkkinä myös testauksen ylikuormittumisesta. Kyseinen ominaisuus siirrettiin testaukseen

5.3.2012, mutta sen testaaminen aloitettiin vasta 2.4.2012 eli noin kuukausi tämän jälkeen. Ideaalitapauksessa testaus pystyisi aloittamaan testauksen välittömästi, mutta nyt testaus oli jo noin kuukauden jäljessä tuotekehitystä. Esimerkkinä oleva ominaisuus saatiin suljettua valmiina vasta 25.6.2012. Ominaisuus oli testauksessa lähes kolme kuukautta. Kaksi merkittävintä syytä tähän olivat ominaisuuden puutteellinen määrittely, sekä testauksen aloittamisen viivästyminen.

Koska ominaisuuden määrittely oli puutteellinen, joutui testaaja miettimään miten ominaisuuden haluttiin toimivan. Ilman kirjoitettua määrittelyä, testaaja joutui varmistamaan tuotepäälliköltä lähes jokaisen kohdan ominaisuuden toteutuksesta. Tuotepäällikön käsitys asiasta ei välttämättä ollut aina sama, kuin mitä kehittäjällä oli ollut, jolloin kehittäjä joutui muuttamaan toteutusta ominaisuuteen, jonka hän oli saanut valmiiksi yli kuukausi sitten. Ominaisuuden testauksen viivästyminen ja määrittelyn täydentäminen jälkikäteen aiheuttivat lopulta sen, että ominaisuus saatiin valmiiksi vasta lähes 8 kuukautta sen aloittamisen jälkeen.

Suuremman ongelman laadunvarmennuksen jääminen jälkeen aiheuttaa projektin lopussa. Kun kaikki uuteen ohjelmistoversioon tehtävät ominaisuudet on saatu kehitettyä, pitäisi versio pystyä viimeistelemään suhteellisen lyhyessä ajassa. Laadunvarmennuksen ylikuormittumisen vuoksi, ei kenelläkään ole vielä hyvää käsitystä siitä, miten hyvin kehitetyt ominaisuudet toimivat. Mahdollisia ohjelmistovirheitä löytyy siis huomattava määrä vielä projektin loppuvaiheessa, jolloin koko projektin aikataulu usein viivästyy.

Edellisessä luvussa mainittu projekti myöhästyi yli kaksi kuukautta. Kun RC-vaiheeseen siirtymistä päätettiin siirtää eteenpäin, ohjelmiston regressiotestauksen kattavuus oli vasta 17 %. Tämä kertoo siitä, että ohjelmiston regressiotestaus oli aloitettu liian myöhään eikä sitä ollut ehditty saamaan ajoissa tarvittavalle tasolle. Yrityksellä ei ollut tarvittavaa tietoa tuotteen laadusta, joten RC-vaiheeseen siirtyminen olisi ollut liian riskialtista.

Kohdeyrityksen laadunvarmennus tuntee yrityksen ohjelmistot erittäin hyvin. Testaajilla on paras tuotetuntemus, koko yrityksessä. Tästä johtuen muu organisaatio käyttää testaajia hyvin paljon apunaan ratkaisemassa asiakasongelmia. Asiakastuen ja –

projektien ongelmat työllistävät yrityksen testaajia huomattavan paljon. Testaajien omien arvioiden mukaan, he käyttävät työajastaan n. 40–60 % asiakasongelmien selvittelyyn. Tämä aika on pois heidän omasta työstään, joka pahentaa tilannetta entisestään.

3.6.7 Ylläpitotyö sekoittaa uuden kehittämistä

Kohdeyrityksen asiakkaiden käyttämät vanhemmat tuoteversiot työllistävät yrityksen tuotekehitystä huomattavan paljon. Tuotekehityksen pääasiallinen työ on kehittää aina uutta versiota. Kestää kuitenkin kauan ennen kuin yrityksen asiakkaat saavat uuden version käyttöönsä. Olemassa olevat asiakkaat eivät välttämättä ota uutta versiota lainkaan käyttöönsä, jos se ei tarjoa heille merkittävää lisäarvoa. Asiakkailla on siis käytössä jopa vuosia vanhoja versioita, joihin kohdeyritys on lupautunut tarjoamaan tukea tiettyyn rajaan saakka.

Kohdeyrityksen asiakastuki toimii monella eri tasolla. Ongelmia voidaan eskaloida aina seuraavalle tasolle, kunnes ongelma saadaan ratkaistua. Ensimmäinen taso käsittää asiakastuen eli helpdeskin. Ensimmäinen taso on nimensä mukaisesti asiakkaan ensimmäinen kontakti yrityksen asiakastukeen. Ensimmäinen taso pystyy selvittämään perustason ongelmat aikaisempien ongelmaselvitysten pohjalta. Kun asiakastuki ei kykene ratkaisemaan ongelmaa, ongelman perustiedot pyritään kartoittamaan mahdollisimman hyvin ja ongelma eskaloidaan seuraavalle tasolle. Asiakastuen toisella tasolla osaaminen on teknisesti laajempaa ja siellä kyetään ratkaisemaan teknisesti haastavampia ongelmia. Kohdeyrityksessä asiakastuki eli helpdesk hoitaa sekä ensimmäisen että toisen tason asiakastuen. Kun ongelma joudutaan eskaloimaan kolmannelle tasolle, se siirtyy tuotekehityksen selvitettäväksi. Kohdeyrityksessä ei ole käytössä enempää tasoja, vaan kolmas taso on ylin eskaloinnin taso asiakasongelmien selvittämisessä. (Windley 2002, 3-4.)

Asiakkailla olevia ongelmia selvitetään asiakastuessa jatkuvasti. Tukea yritetään tarjota myös, vaikka käytössä oleva versio olisikin virallisen tuen ulkopuolella. Asiakastuki käyttää ongelmia selvittäessään varsin usein apunaan tuotekehitystä. Varsinkin testaajien tuotetuntemus on usein korvaamatonta asiakkaan ongelmia ratkaistaes-

sa. Virallisesti tuotekehitys on lupautunut vastaamaan yrityksen kolmannen asteen tuesta.

Monet asiakasongelmista selviää ohjeistamalla tai muuttamalla ohjelmiston määrittelyjä. Joskus ongelman aiheuttaa ohjelmavirhe ja ongelman ratkaiseminen vaatii korjauksen ohjelmaan. Tällöin ongelman ratkaiseminen siirtyy kokonaan tuotekehitykselle, joka korjaa ohjelmistovirheen. Korjaus joudutaan tekemään asiakkaalla käytössä olevasta versiosta riippuen useampaan versioon. Korjaus tehdään aina uusimpaan kehitteillä olevaan versioon, mutta asiakas voi haluta korjauksen ennen seuraavan version toimittamista, jolloin korjaus joudutaan tekemään asiakkaan käytössä olevaan versioon. Korjauksen tekeminen vanhempiin versioihin kasvattaa luonnollisesti kehittämiseen kuluvaa aikaa, sekä lisää testaajien työkuormaa, kun sama korjaus joudutaan testaamaan useassa eri ohjelmistoversiossa.

Kohdeyrityksessä seurataan kolmannen tason asiakasongelmien (tikettien) määrää viikoittain. Tilastoista ei käy ilmi käsiteltyjen tikettien määrä, vaan ainoastaan tikettien lukumäärä tietyllä ajanhetkellä. Tilastojen mukaan 2014 jälkimmäisellä puoliskolla tikettien viikoittainen lukumäärä on ollut tuotekehityksessä keskimäärin 51 tikettiä. Tikettien lukumäärä on vaihdellut 35 ja 69 tiketin välillä. Jokainen tiketti merkitsee selvitystyötä tuotekehityksessä. Osa ongelmista pystytään ratkaisemaan nopeasti, mutta toisten ongelmien selvitystyö kestää jopa viikkoja. Kohdeyrityksessä ei ole käytettävissä tilastoa asiakasongelmien ratkaisuajasta. On myös huomioitava, että tikettien kautta tehtävä selvitystyö kattaa ainoastaan yrityksen asiakastuen kautta tulleet ongelmat. Asiakasongelmia tuodaan tuotekehitykselle myös suoraan esimerkiksi palveluorganisaatiolta.

Kohdeyrityksessä seurataan työajan käyttöä eri projektien ja ylläpitotyön välillä. Tämän tilaston mukaan, jokainen tuotekehityksen projekteissa työskentelevä henkilö käytti 2014 heinäkuun ja lokakuun välisenä aikana n. 15 % työajastaan ylläpitotyöhön. Ylläpitotyö ei kuitenkaan jakaudu tasaisesti kaikille, vaan painottuu enemmän testaajille, sekä henkilöille, jotka ovat pääasiallisessa vastuussa asiakasongelmien eteenpäin viemisessä.

Meneillään olevat asiakasprojektit aiheuttavat myös joskus lisätyötä yrityksen tuotekehitykselle. Asiakasprojektissa voi tulla esille tarve, joka vaatii uuden ominaisuuden kehittämistä. Uusi ominaisuus siis määritellään ja toteutetaan uusimpaan versioon, mutta usein se joudutaan toteuttamaan myös vanhempaan versioon. Tämä johtuu siitä, että kehitteillä oleva uusi versio ei välttämättä ole toimitettavissa asiakkaalle projektin aikataulun kannalta tarpeeksi nopeasti. Asiakas ottaa siis käyttöön sillä hetkellä toimituksessa olevan version, mutta tarvitsee siihen myös kyseisen uuden ominaisuuden. Kyseisen ominaisuuden kehittämiseen menevä aika lähes kaksinkertaistuu, kun ominaisuus kehitetään ja testataan kahdessa eri versiossa. Lisäksi uusi ominaisuus ei ollut alun perin mukana kehitteillä olevan ohjelmistoprojektin suunnitelmassa ja täten viivästyttää kyseisen projektin valmistumista.

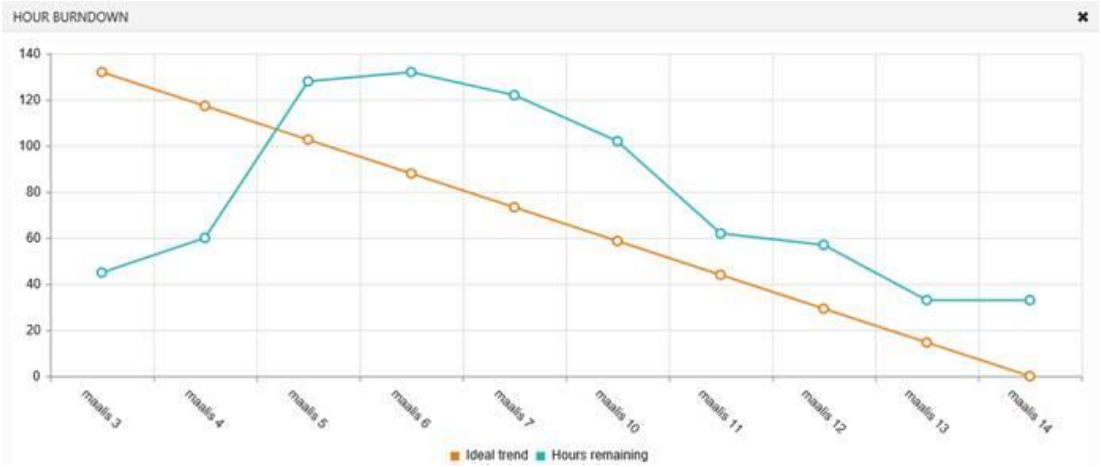
3.6.8 Läpinäkyvyys

Yksi tuotekehitysprojektin tärkeimmistä tukipilareista on läpinäkyvyys. Suunnitelmia voidaan aina muuttaa, mutta ilman läpinäkyvyyttä ei ongelmia nähdä ajoissa eikä niihin ei pystytä reagoimaan. On ensisijaisen tärkeää tietää mitä projektissa on jo tehty ja kuinka paljon työtä on vielä jäljellä. Suoritettujen iteraatioiden perusteella voidaan sen jälkeen ennustaa, kuinka kauan jäljellä olevan työn tekeminen vielä kestää. Läpinäkyvyyden avulla tiedetään tarkalleen missä ollaan ja voidaan tehdä tarvittavia muutoksia, jotta tuotetta saadaan kehitettyä haluttuun suuntaan. (Schwaber, Kubacki & Sutherland 2012, 86.)

Yksi suurimmista kohdeyrityksen ongelmista on ollut läpinäkyvyyden puute. Läpinäkyvyyden puute on hyvin suuri ongelma varsinkin niissä tuotekehitystiimeissä, missä ei ole käytetty minkäänlaista määriteltyä kehitysprosessia. Näiden tiimien työtilannetta ei ole pystytty seuraamaan minkään työkalun avulla. Vaikka tällaisten tiimien tuotteissa oli samalla tavalla huomattavia laadullisia ongelmia, kuin muissakin tiimeissä, niin työn edistyminen ja mahdolliset ongelmat olivat piilossa muulta organisaatiolta. Työn edistymistä voitiin seurata vain kysymällä siitä tiimin jäseniltä. Suullinen työtilanteen seuranta oli parhaimmillaankin epätarkkaa ja puolueellista. Tämä hankaloitti projektin edistymisen seuraamista ja version aikatauluttamista.

Osa kehitystiimeistä on työskennellyt ketteriin menetelmiin kuuluvan Scrum-menetelmän mukaisesti, mutta myös näissä tiimeissä on ollut ongelmia aikataulujen kanssa. Näiden tiimien työtä on pystytty seuraamaan projektinhallintaohjelmiston avulla, mutta edellisissä luvuissa mainittujen ongelmien vuoksi työmääräarviot olivat epätarkkoja ja saattoivat muuttua merkittävästi. Todellinen työmäärä oli luultua suurempi tai pahimmassa tapauksessa oli kehitetty jotain eri tavalla, kuin mitä tuotteen tuotepäällikkö oli alun perin halunnut.

Scrum-menetelmän mukaisesti toimivien tuotekehitystiimien työtä seurattiin pääasiassa Sprint Burndown -raportin avulla (kuvio 6). Alla olevasta tyypillisestä raportista huomataan, että myös Scrum-tiimien työtä oli usein hyvin hankala ennustaa, eikä tiimi useinkaan pystynyt toteuttamaan niitä asioita, mitä se oli alun perin luullut pystyvänsä.



Kuvio 6. Esimerkki kohdeyrityksen tuotekehitystiimin Sprint Burndown -raportista

Raportista havaitaan, että iteraation alussa ei ole vielä tiedetty tai tunnettu kaikkea tehtävää työtä. Jäljellä olevaa työmäärää kuvaava sininen viiva nousee iteraation alussa, kun tuotekehitystiimi antaa iteraatiossa olevalle työlle työmääräarvionsa. Kestää viisi päivää ennen kuin jäljellä oleva työmäärä alkaa ensimmäistä kertaa laskea. Alussa eteneminen on hidasta, mutta työ saadaan suoritettua lähes kokonaan. Ongelmaksi muodostuu jäljelle jäänyt työ. Raportin mukaan tekemättä jäi n. 35 tuntia työtä. Vaikka määrä ei ole suuri, ei keskeneräistä tuotetta voida toimittaa asiakkaalle ja koska kehitettyjä ominaisuuksia ei ole kokonaan testattu, ei voida olla varmoja jäljellä olevan työn määrästä.

3.6.9 Aikatauluongelmat

Edellisissä luvuissa mainitut ongelmat johtivat lähes aina siihen, että versio myöhästyi määritellystä aikataulusta. Versioprojektin aikataulu ei aina riittänyt kaikkien ominaisuuksien toteuttamiseen tai viimeistelemiseen. Kohdeyrityksessä suoritetaan versioprojektien jälkeen katselmus, missä yritetään kartoittaa päättyneen projektin suurimmat onnistumiset ja puutteet. Kartoituksen tarkoituksena on ollut määritellä korjaavia toimenpiteitä, jotta seuraavissa projekteissa onnistuttaisiin paremmin.

Alkuvuodesta 2014 päättyneen projektin kartoituksessa suurimmiksi heikkouksiksi mainittiin muun muassa:

- Läpinäkyvyyden puute
- Epäselvät roolit
- Aikataulun myöhästyminen
- Epäselvät ja/tai puutteelliset ominaisuusmäärittelyt
- Epäonnistuneet projektin aikaiset sisältömuutokset
- Suuri ylläpitotyön määrä

Projektista löydettiin myös hyvin menneitä asioista, kuten:

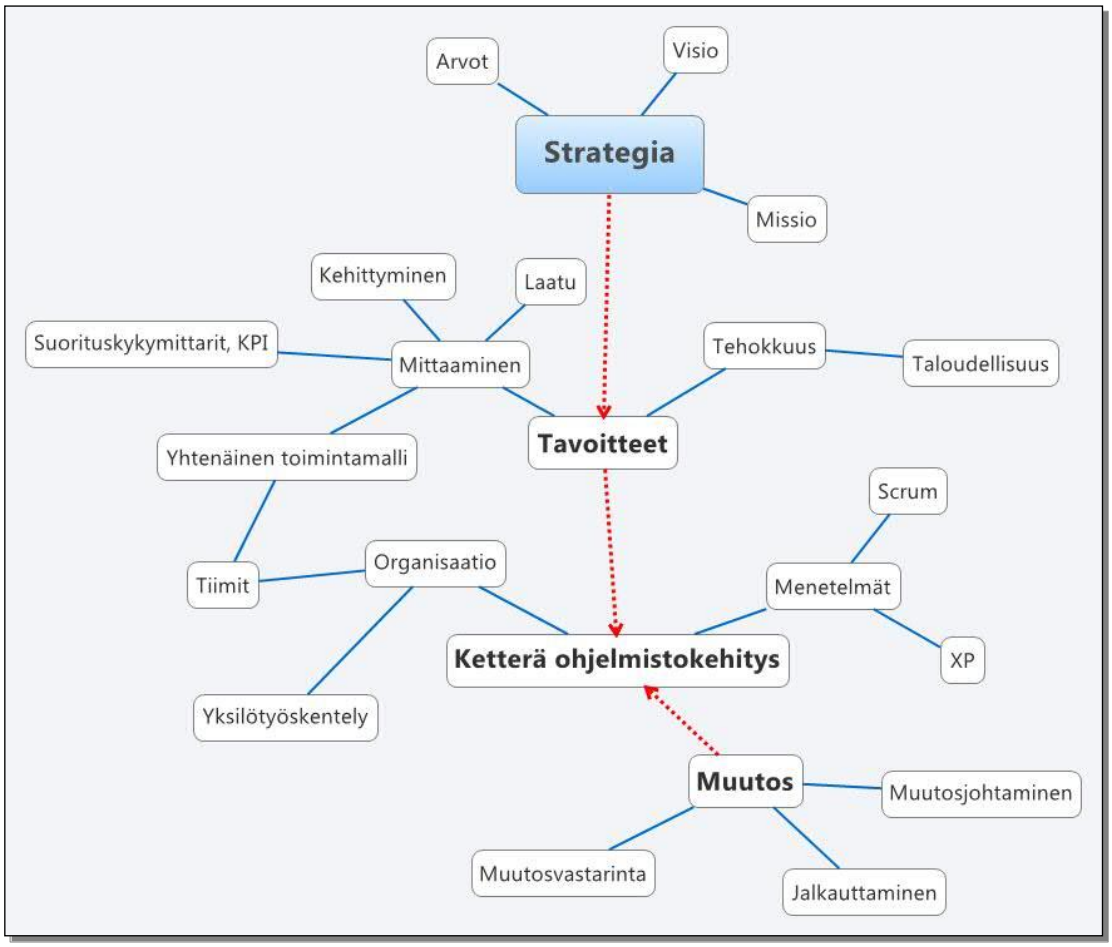
- Tuotteeseen saatiin tehtyä tärkeitä ominaisuuksia ja suorituskykyparannuksia
- Projektissa työskenteli sitoutuneita työntekijöitä
- Ammattimainen testaus, josta päivittäiset raportit
- Osa projektin aikaisista sisältömuutoksista onnistui hyvin

Arvioitu projekti oli hyvin tyypillinen kohdeyrityksen tuotekehitysprojekti. Samat ongelmat tulivat esille lähes joka kerta, mutta kuten huomataan, ei projekti ollut täysin epäonnistunut. Projektin tuloksena saatiin asiakkaille tärkeitä uusia ominaisuuksia ja suorituskykyparannuksia.

Tämän opinnäytetyön tuloksena syntyvän tuotekehitysprosessin tarkoituksena on ratkaista ohjelmiston versioprojektien ongelmia ja mahdollistaa edelleen ne asiat, jotka ovat menneet hyvin. On kuitenkin myös huomioitava, että kaikkia projekteissa esiintyviä ongelmia ei voida ratkaista pelkällä tuotekehitysprosessilla.

Tuotekehityksen käyttämän prosessin tulisi olla läpinäkyvä, ketterä, tehokas ja laadukas. Tuotekehityksen tulisi pystyä ylläpitämään tuottamiaan versioita ilman, että se vaikuttaa negatiivisesti uusien versioiden kehitykseen. Prosessia käytetään virtuaalisilla ja monikansallisilla tiimeillä, jotka yrittävät kehittää uusia innovatiivisia ominaisuuksia useita kymmeniä vuosia vanhan ohjelmiston päälle. Tämä opinnäytetyön toteutettiin osana kohdeyrityksen tuotekehityksen kehitysprosessia määritellä yhteinen tuotekehitysprosessi kaikille yrityksen tuotekehitystiimeille.

3.7 Viitekehys



Halukkuus tuotekehityksen prosessin uudistamiseen pohjautuu yrityksessä määriteltyyn strategiaan ja sen onnistumiseen. Jotta yrityksen tuotekehityksellä on mahdollisuus onnistua määritellyn strategian mukaisesti, sen tulee uudistaa ja yhtenäistää prosessiaan. Kun tässä onnistutaan, saadaan koko tuotekehitys toimimaan tehokkaam-

min ja tuotekehitys pystyy toteuttamaan strategiaa huomattavasti tehokkaammin, kuin aikaisemmin.

Strategiasta muovautuu tuotekehityksen tavoitteet, jotka asettavat omat tavoitteensa myös kehitettävällä prosessille. Prosessin on luotava koko yrityksen tuotekehitykselle yhtenäinen työskentelytapa. Yrityksen on kyettävä mittaamaan eri tiimien tuotekehityksen suorituskykyä. Mittareiden on myös mahdollistettava tulevaisuuden ennustaminen, jotta tuotehallinto pystyy suunnittelemaan tuotteen elinkaarta kauemmas tulevaisuuteen.

Prosessin jalkauttaminen ja uuden prosessin mukanaan tuoman muutoksen hallinta on olennainen osa opinnäytetyötä ja määrittelee sen, miten koko projektissa onnistutaan. Muutosvastarinnan murtaminen ja uuden prosessin onnistunut jalkauttaminen ovat edellytyksiä koko projektin onnistumiselle. Muutosjohtamisen opit ovat hyvin tärkeässä osassa prosessin jalkauttamisessa.

Organisaatiossa työskentelevät ihmiset ovat lopulta ne, ketkä saavat muutoksen onnistumaan tai epäonnistumaan. Projektin jalkauttamisvaiheessa on löydettävä innovatiivisia keinoja mitata, motivoida ja palkita ihmisten hyvää suoritusta. Kun yrityksessä toimitaan tiimeissä, nousevat tiimien esimiehet myös suureen rooliin koko prosessin jalkauttamisessa. (Murthy 2007, 23–24.)

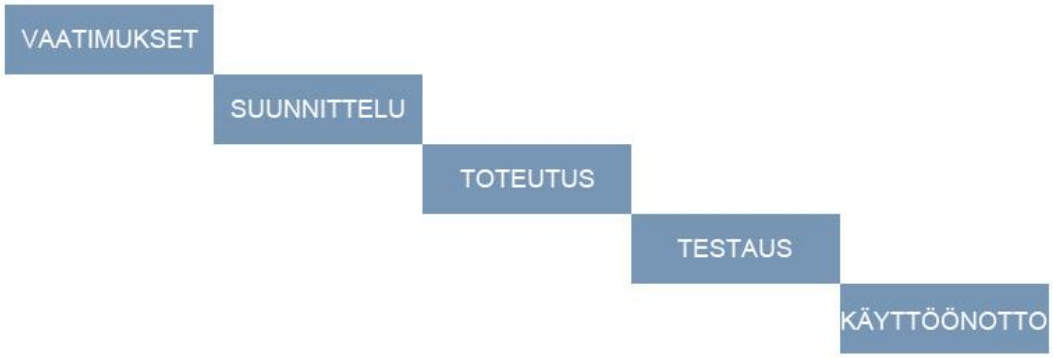
4 OHJELMISTOTUOTANTO

4.1 Projektimallit

Nykyajan ohjelmistotuotanto on hyvin monimutkaista ja sisältää useita eri työvaiheita. Kehitettävät järjestelmät ovat entistä monimutkaisempia ja sisältävät entistä enemmän liittymiä ulkoisiin järjestelmiin. Projektit sisältävät kasvavissa määrin yhä enemmän epätietoisuutta. Tästä johtuen projektit ovat kasvaneet vuosien aikana entistä laajemmiksi. Projekteissa työskentelee suuri määrä ihmisiä ja ne kestävät usein kuukausia tai jopa vuosia. Tämä asettaa ohjelmistotuotannolle entistä enemmän vaa-

timuksia ja tekee nykyajan ohjelmistotuotannon projektien läpiviemisestä entistä haastavampaa.

Ohjelmistoprojekti pitää sisällään useita eri vaiheita. Eri menetelmät suhtautuvat kuitenkin vaiheisiin eri tavalla. Vesiputousmallissa vaiheet suoritetaan järjestyksessä, kun taas ketterissä menetelmissä vaiheita suoritetaan koko projektin ajan lähes samanaikaisesti (kuvio 7). Ohjelmistoprojektin vaiheet eivät eroa kovinkaan merkittävästä minkä hyvänsä muun projektin vaiheista. Ainoastaan vaiheiden sisältö ja tulos ovat erilaisia.



Kuvio 7. Ohjelmistoprojektin vaiheet vesiputousmallissa (Cobb 2011, 169)

Kaikki alkaa suunnitteluvaiheesta. Hyvän suunnittelun tavoitteena on tunnistaa kaikki vaatimukset, jotka kuuluvat alkavaan projektiin. Asiakkaalla on usein hyvin tarkka käsitys siitä, minkä ongelman he haluavat ohjelmistolla ratkaistavan, mutta ei useinkaan minkäänlaista käsitystä siitä, minkälainen ohjelmiston pitäisi olla. Hyvällä suunnittelulla saadaan tämä asiakkaan vaatimus käännettyä konkreettiseksi suunnitelmaksi, jonka pohjalta voidaan ohjelmistoa lähteä suunnittelemaan. Kun ylätason suunnitelmat on saatu valmiiksi, on myös tärkeää rajata kehitettävä ohjelmisto tarkasti, jotta projektista ei muodostu liian suurta ja se saadaan toteutettua aikataulun ja budjetin määrittelemissä puitteissa.

Suunnitteluvaiheeseen liittyy tiiviisti myös ohjelmiston tekninen suunnittelu. Kun ohjelmiston vaatimukset on saatu selville, voidaan näiden vaatimusten pohjalta suunnitella toteutettava sovellus niin, että se täyttää ohjelmistolle asetetut vaatimukset parhaalla mahdollisella tavalla. Kun vaatimusmäärittely oli enemmänkin tuote-

päälliköiden ja ei-teknisten osaajien vastuulla, niin tekninen suunnittelu vaatii jo hyvin paljon teknistä osaamista ja näkemystä. Kohdeyrityksen tapauksessa tuotekehitys tekee teknisen suunnitelman yhdessä tuotepäällikön kanssa.

Kun ohjelmiston vaatimukset ja niistä johdetut suunnitelmat on tehty hyvin, voidaan aloittaa ohjelmiston toteutusvaihe. Toteutusvaiheen aikana ohjelmistoa kehitetään (ohjelmoidaan) suunnitelmien perusteella. Ohjelmiston testaus on työvaiheena lähes päällekkäinen toteutuksen kanssa. Ohjelmistoa voidaan testata hyvin erilaisilla tavoilla. Osa testauksesta on yleensä automatisoitu, mutta hyvin usein testaus riippuu paljon manuaalisesta integraatiotestauksesta. Kun ohjelmisto on saatu kehitettyä suunnitelmien mukaisesti ja kaikki sen toiminnallisuus on testattu halutulla tasolla, voidaan siirtyä käyttöönottovaiheeseen.

Käyttöönottovaiheessa ohjelmisto asennetaan ja otetaan käyttöön tuotantoympäristössä. Usein käyttöönottovaihe sisältää vielä edeltävän testivaiheen (pilotoinnin) lopullisessa ympäristössä, mutta tämä voidaan laskea mukaan projektin käyttöönottovaiheeseen. Ohjelmiston käyttöönottoa varten on hyvä tehdä erillinen käyttöönottosuunnitelma, jossa määritellään miten järjestelmän käyttäjät koulutetaan, miten uudet toimintatavat saadaan hyväksytyksi organisaatiossa, tarvitaanko vanhan ja uuden järjestelmän rinnakkaiskäyttöä ja miten käyttäjäorganisaatio osallistuu hyväksymistesteihin. Kun kaikki edellä mainitut asiat ovat selkeitä ja etukäteen mietitty, säästytään kalliilta yllätyksiltä ja ohjelmiston käyttöönotto on nopeaa. (Lehtimäki 2006, 176.)

Kun ohjelmisto on käyttöönotettu, siirrytään projektinhallinnallisesti ylläpitovaiheeseen. Ohjelmiston ylläpitovaihe ei välttämättä enää kuulu ohjelmistoprojektin piiriin, vaan vastuu ohjelmiston ylläpidosta voidaan siirtää projektin ulkopuoliselle taholle. Usein projekti koetaan päättyneeksi kun ohjelmiston hyväksyntätestaus on suoritettu onnistuneesti ja asiakas on hyväksynyt ohjelmiston ja todennut, että se täyttää kaikki sille määritellyt vaatimukset.

Edellä mainitut vaiheet kuulostavat hyvin yksinkertaisilta ja selkeiltä. Ideaalimaailmassa kaikki onkin yksinkertaista ja ohjelmistoprojektit saataisiin vietyä läpi ilman ongelmia. Projektin alussa tehty vaatimusmäärittely ja sen pohjalta johdettu suunnitelma ovat kuitenkin usein puutteellisia. Vaatimuksista on voitu unohtaa asioita tai

kaikkea ei ole osattu ottaa huomioon projektin alussa. Jokin ohjelmiston ulkopuolinen riippuvuus tai alkuperäiset tarpeet voivat myös muuttua projektin aikana. Kaikki tällainen luo muutostarpeita ohjelmistoon ja niiden hallinta on ohjelmistoprojektin yksi suurimmista haasteista.

4.2 Vesiputousmalli

Vesiputousmalli on perinteinen ohjelmistoprojektin elinkaarimalli. Mallissa on lähdetty siitä olettamuksesta, että projekti onnistuu kun sen vaiheet suoritetaan hyvin ja vaiheittain, eikä seuraavaan vaiheeseen siirrytä ennen kuin edellinen vaihe on täysin valmis. Vesiputousmalli on enää hyvin harvoin käytössä nykyajan ohjelmistoprojekteissa, koska siinä on todettu olevan useita projektin vaarantavia seikkoja.

Vesiputousmallissa siirrytään järjestyksessä projektin vaiheesta toiseen. Edellisen vaiheen lopputuloksesta tulee syöte projektin seuraavaan vaiheeseen. Tästä johtuen vaiheiden lopputuloksena syntyvät dokumentit ovat usein lukittuina muutoksilta. Niiden muuttaminen jälkikäteen romuttaisi koko projektin myöhemmät työvaiheet. Vesiputousmalli on hyvin käytännöllinen, mutta tiukka malli ohjelmistoprojektin etenemisestä. (Mohapatra 2010, 19.)

Vesiputousmalli voidaan kuvata myös V-kirjaimen muotoisena mallina, jossa ensin lähdetään vasemman sakaran yläosasta ja edetään alas, josta jatketaan oikeaa sakaraa ylös. Alaspäin meneminen on tekemistä ja takaisinpäin paluu on testaamista. Tällä tavalla kuvattuna nähdään selkeästi vesiputousmallin keskeisin kritiikin kohde. Mitä alemmas V-mallissa mennään, sitä vähemmän asiakas on mukana tekemisessä. Kun lopulta tullaan takaisin ylös ja asiakas on taas mukana tekemisessä, voi asiakas havaita, että projektissa on menty aivan väärään suuntaan. Korjausten tekeminen ohjelmiston elinkaaren loppuvaiheessa on huomattavasti kalliimpaa kuin niiden tekeminen projektin alkuvaiheessa. (Lehtimäki 2006, 151.)

Toinen suurimmista vesiputousmallin kritiikin kohteista on dokumentoinnin määrä. Malli luottaa siihen, että jokaisesta työvaiheesta tehdään kattava dokumentointi, jonka asiakas hyväksyy. Vasta tämän jälkeen projektissa voidaan siirtyä seuraavan työ-

vaiheeseen. Dokumentoinnin tuottaminen on kallista ja projektin jälkeen suurimmalla osalla näistä dokumenteista, ei ole minkäänlaista arvoa. (Lehtimäki 2006, 152.)

Kun projektin tavoite ja vaatimukset ovat selkeitä ja ne voidaan määritellä tarkasti projektin alussa, voi vesiputousmalli olla sopiva vaihtoehto projektimenetelmäksi. Vesiputousmalli ei kestä hyvin jatkuvia muutoksia, joten muutoksien riskin tulisi olla vesiputousmallin projekteissa hyvin pieni. Perinteistä vesiputousmallia käytetään yleisesti silloin, kun projektin kustannukset ja aikataulu ovat hyvin tärkeitä. Tarkalla muutosprosessilla voidaan myös hallita mahdollisten muutosten vaikutuksia projektin aikatauluun ja kustannuksiin. (Cobb 2011, 169.)

Vesiputousmallilla voidaan saada toteutettua ohjelmistoprojekti aikataulussa ja budjetissa, mutta tuotettu ohjelmisto ei vastaa lainkaan asiakkaan todellista tarvetta. Vesiputousmallin käyttäminen jatkuvasti muuttuvissa ympäristöissä, on osoittautunut lähes mahdottomaksi. On hyväksytty, että vaatimukset muuttuvat ja tällaisiin projekteihin soveltuu joustavampi malli, jossa muutoksia pystytään hallitsemaan paremmin. (Cobb 2011, 7.)

Vesiputousmallin ongelmia on lähdetty korjaamaan kehittämällä iterointiin perustuvia malleja. Näiden mallien katsotaan kuuluvan ketterien menetelmien termin alle. Vaikka useimmissa malleissa on eroavaisuuksia, niin yhteistä niille kaikille on se, että ne perustuvat iteroimiseen. Sen sijaan, että tehtäisiin pitkiä vaiheita peräkkäin, tehdäänkin toimivaa ohjelmistoa lyhyemmissä iteraatioissa. Jokaisen iteraation jälkeen voidaan ohjelmistoa käydä läpi asiakkaan kanssa. Asiakkaan on helpompi kommentoida toimivaa ohjelmistoa, kuin kirjoitettua dokumenttia. Jokaisen iteraation jälkeen on myös helppo vaihtaa projektin suuntaa ja tehdä tarvittavia muutoksia. Jokainen iteraatio on kuin oma projektinsa sisältäen kaikki sen työvaiheet suunnittelusta testaukseen.

4.3 Ketterät menetelmät

Ketterät menetelmät ovat vastaisku perinteistä ja jäykkää projektimallia vastaan, jossa työvaiheet ovat pitkiä, byrokraattisia ja joissa ihmisresurssit ovat korvattavissa

olevaa työvoimaa, kuten muussakin tehdastyössä. Ketteriä menetelmiä on käytetty satunnaisesti jo vuosikymmeniä sitten, mutta vasta viime aikoina niiden käyttö on yleistynyt räjähdysmäisesti. (Schuh 2004, 2.)

Vesiputousmalli on rakennettu niin sisään lähes jokaiseen projektipäällikköön, että on hyvin vaikeaa muuttaa ajattelua ketterämpien menetelmien suuntaan. Vesiputousmallia on käytetty yli 25 vuoden ajan. Sitä opetetaan yliopistoissa ja käsitellään tuhansissa kirjoissa. Vesiputousmallin käytännöt ovat juurtuneet hyvin syvälle ihmisten mieliin. Kun perinteisessä mallissa vaatimusten määrittelyyn ja suunnitteluun voidaan hyvinkin käyttää jopa puolet projektin kestosta, ketterissä menetelmissä tilanne on täysin päinvastainen. Ihmisiä on hyvin hankala saada avoimiksi tämänkaltaisille muutoksille. (Schwaber 2007, 686.)

Ketterän ohjelmistokehityksen kehittäjät ovat yhdessä laatineet manifestin, jolla viestitään niitä arvoja, joita kaikki ketterät menetelmät noudattavat. Manifesti kuuluu seuraavasti:

Löydämme parempia tapoja tehdä ohjelmistokehitystä, kun teemme sitä itse ja autamme muita siinä. Kokemuksemme perusteella arvostamme:

Yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja
Toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota
Asiakasyhteistyötä enemmän kuin sopimusneuvotteluja
Vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa

Jälkimmäisilläkin asioilla on arvoa, mutta arvostamme ensiksi mainittuja enemmän. (Agile Manifeston www-sivut 2013.)

Ketteriä menetelmiä on useita erilaisia, joista kaikki perustuvat samoihin toimintaperiaatteisiin. On tärkeää huomata, että itse ketterä menetelmä ei ole konkreettinen menetelmä vaan ajatusmalli, jonka alla toimii useita erilaisia menetelmiä, jotka kaikki noudattavat ketterien menetelmien periaatteita. (Schuh 2004, 2.)

Parhaiten tunnettuja ketteriä menetelmiä ovat (Schuh 2004, 2):

- Adaptive Software Development (ASD)
- The Crystal Methodologies

- Dynamic Systems Development Method (DSDM)
- Extreme Programming (XP)
- Feature-Driven Development (FDD)
- Lean Software Development
- Scrum

Ketterät menetelmät perustuvat siihen, että pitkien työvaiheiden sijasta tehdään lyhyempiä iteraatioita. Jokaisen iteraation päätteeksi saadaan aikaiseksi toimiva ohjelmisto. Ohjelmisto kasvaa näin vähitellen kohti päämäärää, mutta kuitenkin niin, että jokaisen iteraation jälkeen asiakas voi nähdä uusien arvoa tuottavien ominaisuuksien toimivan. Asiakkaan on näin huomattavasti helpompi kommentoida projektin etenemistä, kuin lukemalla kirjoitettua dokumentaatiota.

Ketterien menetelmien suurin ajuri on jatkuvien muutosten hallinta. Ketterien ohjelmistojen kehittämisprosessien on kyettävä reagoimaan ja mukautumaan jatkuviin ja nopeisiin muutoksiin. Pelkkä mukautuminen ja reagointi eivät kuitenkaan riitä vaan prosessien on edelleen kyettävä tuottamaan laadukkaita ohjelmistoja kustannustehokkaasti. (Holcombe 2008, 2.)

Onnistunut ketterä ohjelmistoprojekti pystyy muokkaamaan ohjelmiston kehitystä asiakkaan muuttuneeseen ongelmaan. Kehitettyä ohjelmistoa on pystyttävä jatkokehittämään helposti uusien tarpeiden täyttämiseksi. On pystyttävä varmistamaan, että ohjelmisto on laadukasta ja se toimii määritellyllä tavalla. Ketterän projektin on vähennettävä byrokratiaa ja dokumentointia. Kaikenlainen turha dokumentaatio tulisi pyrkiä karsimaan. Viimeisenä vaatimuksen onnistuneelle ketterälle ohjelmistoprojektille on ihmisresurssien hallinta. Kun ohjelmiston kehitysprosessi pystyy täyttämään kaikki nämä vaatimukset, se luultavasti onnistuu minkälaisessa ohjelmistoprojektissa tahansa. (Holcombe 2008, 2.)

4.3.1 Historia

1990-luvulla ohjelmistoteollisuutta rasittivat epäonnistuneet ohjelmistoprojektit. Ohjelmistoprojektit tulivat kuuluisiksi myöhästymisistä, ylittyneistä budjeteista, toimi-

mattomista lopputuotteista ja tyytymättömistä asiakkaista. Kourallinen ohjelmistoalan johtavia kehittäjiä rajasivat epäonnistumisten tärkeimmiksi tekijöiksi ylisuunnittelun, riittämättömän kommunikoinnin, sekä kaikki-kerralla ajattelun. (Cooke 2012, 32.)

Epäonnistuneiden ohjelmistoprojektien taustalla oli tietysti myös muita syitä, mutta organisaatiot pystyivät vaikuttamaan suoraan näihin kolmeen asiaan. Näiden kolmen ongelman pohjalta lähdettiin kehittämään erilaisia toimintatapoja, jotka nykyään tunnetaan paremmin ketterinä menetelminä. Ketterät menetelmät saivat lopulta oman nimensä, kun agile manifesto sai syntynsä helmikuussa 2001. (Koch 2004, 3.)

Ketterien menetelmien kehittäneet vaikuttajat kokoontuivat helmikuussa 2001 keskustelemaan siitä, mitä yhteistä heidän kehittämillään menetelmillä on. He huomasivat, että menetelmillä oli paljon yhteistä. Yhteisistä asioista koottiin ketterien menetelmien runko, joka konkretisoitui ketterän manifestin (agile manifesto), sekä 12 perusperiaatteen muodossa. (Koch 2004, 4.)

Ketterien menetelmien 12 periaatetta ovat (Agile Manifesto 2001):

1. Tärkein tavoitteemme on tyydyttää asiakas toimittamalla tämän tarpeet täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti.
2. Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.
3. Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein ja suosimme lyhyempää aikaväliä.
4. Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.
5. Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jonka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.
6. Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.
7. Toimiva ohjelmisto on edistymisen ensisijainen mittari.

8. Ketterät menetelmät kannustavat kestävään toimintatapaan. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.
9. Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.
10. Yksinkertaisuus - tekemättä jätettävän työn maksimointi.
11. Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä.
12. Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan ja mukauttaa toimintaansa sen mukaisesti.

Kaikki ketterät menetelmät perustuvat näihin periaatteisiin. Jokainen menetelmä määrittelee kuitenkin omat tarkemmat sääntönsä ja tapansa.

Agile Manifeston muodostaneet ja allekirjoittaneet 17 alkuperäistä henkilöä olivat Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Stephen J. Mellor, Ken Schwaber, Jeff Sutherland ja Dave Thomas. (Koch 2004, 3.)

4.3.2 Mukautuminen muuttuvaan asiakasongelmaan

Perinteisen vesiputousmallin mukaisesti ohjelmistoprojektissa muodostetaan suunnitelmat asiakkaalta kerättyjen vaatimusten pohjalta. Suunnitteluvaihe voi olla hyvinkin pitkä ja kun sen lopuksi palataan asiakkaan luokse luotujen dokumenttien kanssa, voidaan hyvin huomata, että asiakkaan vaatimukset ovat muuttuneet. Perinteinen malli ei tue vaatimusten muuttumista kovinkaan hyvin ja on löydettävä uusia toimintatapoja. On pystyttävä luomaan jatkuva kommunikointi asiakkaan kanssa, sekä kanava tuotekehitykseen, joka ei pidä sisällään pitkää ja byrokraattista suunnitteluvaihetta. Muutokset on saatava tehtyä nopeasti ja tehokkaasti. Ketterät menetelmät ratkaisevat tämän ongelman jatkuvalla kehittämisellä ja asiakkaan ottamisella mukaan prosessiin. Asiakas saa ohjelmiston nähtäväkseen jokaisen iteraation jälkeen. Palau-

tekierto on nopeaa ja projektin suuntaa voidaan tarvittaessa muuttaa nopeasti. (Holcombe 2008, 4.)

Muutoksia voi tulla kahdesta eri suunnasta. Muutos voi olla projektin ulkoinen tai tulla projektin sisältä. Ulkoisia muutoksia voivat olla esimerkiksi lainsäädännöllinen muutos, ekonominen muutos tai muutos markkinatilanteessa. Vaikka ulkoisia muutoksia voidaan yrittää ennustaa, niin usein niihin pystytään ainoastaan reagoimaan. Onkin tärkeämpää miten käsittelemme oppimisen seurauksena syntyneet sisäiset muutokset. (Koch 2004, 142.)

Sisäisiä muutoksia voi tulla kahdesta eri lähteestä; asiakkaan tai kehittäjän oppimisen tuloksena. Asiakkaalla ei yleensä ole tarkkaa kuvaa siitä, mitä he haluavat ja tästä syystä alkuperäiset vaatimusmäärittelyt voivat sisältää kaiken mahdollisen. Asiakas on oppinut, että projektin alussa tehtävä vaatimusmäärittely on ainoa paikka määritellä kehitettävän ohjelmiston vaatimukset. Ketterät menetelmät pyrkivät muuttamaan tämä sallimalla vaatimuksien muuttumisen projektin kuluessa. Asiakas oppii projektin kuluessa ja pyrkii muuttamaan vaatimuksia siihen mitä he todella tarvitsevat, kuin mitä he vain saattavat tarvita. (Koch 2004, 143.)

Samalla, kun asiakas oppii projektin aikana, myös kehittäjät oppivat. Kehittäjät voivat oppia, että kaksi vaatimusta ovat toisensa poissulkevia tai että valitulla arkkitehtuurilla on odottamattomia rajoitteita. Kun asiakkaan oppiminen johtaa uusiin tai muuttuneisiin toiminnallisiin vaatimuksiin, sisäinen oppiminen vaikuttaa teknisiin vaatimuksiin. Muutoksilla teknisiin vaatimuksiin on usein vaikutuksia myös asiakkaan toiminnallisiin vaatimuksiin, aivan kuten toiminnallisilla vaatimuksilla on vaikutuksia teknisiin vaatimuksiin. (Koch 2004, 143.)

Kun tiedetään, että suunnitelmat tulevat muuttumaan, tulee suunnitelmia tehdä usein. Ketterät menetelmät tarjoavat mahdollisuuden tehdä muutoksia sen perusteella, mitä asiakas ja toimittaja oppivat. Jokaisen iteraation alussa projektin suunnitelmat käydään läpi ja niitä voidaan muuttaa opitun perusteella. Ketterät menetelmät estävät suuret yllätykset, kun projekti ja suunnitelma käydään läpi usein. Muutosten vaikutukset projektin aikatauluun, kustannuksiin ja toiminnallisuuteen voidaan sopia jokaisen iteraation jälkeen. (Koch 2004, 145.)

4.3.3 Jatkuvasti kehittyvä ohjelmisto

Vaikka ohjelmistoprojektissa onnistuttaisiin tuottamaan ohjelmisto, joka ratkaisee asiakkaan nykyisiä ongelmia, se ei välttämättä tee niin enää tulevaisuudessa. Maailma jatkaa muuttumistaan myös projektin päätyttyä ja ohjelmistoa on kyettävä kehittämään edelleen, jotta se pystyy täyttämään sille asetettuja uusia vaatimuksia.

Tämän vaatimuksen täyttäminen on hyvin paljon teknisen toteutuksen määrittelemä. Kun olemassa olevaa ohjelmistoa lähdetään jatkokehittämään, on hyvin tärkeää tietää mitä ohjelmisto tekee. Ketteriin menetelmiin kuuluu dokumentoinnin vähentäminen, joten ohjelmiston toiminnasta ei välttämättä ole tarjolla paljoakaan dokumentteja. Ohjelmiston jatkokehittäjä voi olla myös aivan eri taho, kuin ohjelmiston alkuperäinen kehittäjä. Jotta voidaan todella ymmärtää miten ohjelma toimii, joudumme lopulta tarkastelemaan ohjelman lähdekoodia. (Holcombe 2008, 5-6.)

Ohjelmiston lähdekoodin tulee olla selkeää ja ymmärrettävää myös ulkopuolisten tahojen kannalta. Ohjelman toimintalogiikan tulee välittyä pelkällä lähdekoodin tarkastelemisella. Jos ohjelma on kehitetty ketterien menetelmien ohjaamalla tavalla, lisäämällä toiminnallisuutta pienissä paloissa, voidaan myös lähdekoodista erottaa nämä pienet lisäykset, jolloin myös ominaisuuksien muuttaminen ja lisääminen on jatkossa huomattavasti helpompaa. (Holcombe 2008, 5-6.)

4.3.4 Ohjelmiston laatu

Ohjelmiston laatuvirheet voidaan jakaa kahteen eri kategoriaan; Vaatimusvirheisiin, sekä toiminnallisiin virheisiin. Vaatimusvirheissä tietokonetta pyydetään tekemään vääriä asioita. Tämänkaltaisten virheiden suurin lähde on ketterien menetelmien tärkein kohta eli muutos ohjelmiston toiminnallisissa vaatimuksissa. Toiminnallisissa virheissä, tietokone ei tee sitä mitä pyydämme sen tekemään. (Holcombe 2008, 6.)

Vaatimusvirheiden välttämiseksi ketterät menetelmät ovat kehittäneet oman toimintamallinsa, joka kuvataan tarkemmin kohdassa 4.3.2. Operatiivisten virheiden välttäminen on vaikeampaa, eikä käytettävä prosessi pysty vaikuttamaan siihen kuin hy-

vin rajallisesti. Samat ongelmat esiintyvät kaikissa prosessimalleissa. Operatiivisten virheiden vähentämiseksi ketterissä menetelmissä käytetään jonkinlaista katselmointia. Lähestulkoon kaikki projektiin liittyvä katselmoidaan asiakkaan ja muiden sidosryhmien toimesta. Katselmointi on tehokas ja nopea keino varmentaa ohjelmiston toiminta halutulla tavalla. Katselmointi paljastaa myös vaatimusvirheet kehitetyissä ominaisuuksissa. Kehittäjien vastuulla on korjata kaikki katselmoinnissa esiin tulleet asiat. Katselmoinnin avulla voidaan tarkastella yksittäisiä ominaisuuksia ja varmentaa niiden toiminta helposti ja nopeasti ilman turhaa byrokratiaa. Kun jokainen yksittäinen toiminto on katselmoitu, ollaan jo hyvin paljon lähempänä laadukasta ohjelmistoa. Katselmointi ei kuitenkaan poista testauksen tarvetta vaan ainoastaan täydentää sitä ja tuo mukaan kaikkien sidosryhmien näkökulman.

Ohjelmistokehityksessä voidaan laadun mittarit määritellä hyvinkin tarkasta. Tekninen operaatio joko onnistuu tai ei onnistu. Ohjelmistoalan erikoistilanteesta johtuen näitä testejä voidaan automatisoida, jolloin saadaan lähes reaaliaikainen palaute tehdyn työn laadusta. Ketterään ohjelmistokehitykseen kuuluu jatkuva kehittyminen. Jatkuva kehittyminen sisältää myös kuvattujen automaattitestien kattavuuden lisäämisen. Tavoitteena on päästä tilanteeseen, jossa vain nappia painamalla saadaan täydellinen kuva kehitettävän tuotteen laadusta. (Cooke 2010, 316.)

Todellinen laatu ei ole mitään mitä voi saavuttaa sattumalta. Todellinen laatu vaatii koko organisaatiokulttuurin muutosta. Yksikään prosessin muutos ei yleensä auta parantamaan laatua. Tärkein laatua parantava tekijä on kommunikaation lisääminen. Ketterät menetelmät korostavat kommunikaation merkitystä ja kasvavan kommunikaation kautta myös laatu paranee. Tarkoituksena on saada kaikki toimimaan oikein jo ensimmäisellä kerralla. (Cooke 2010, 304-305.)

4.3.5 Dokumentoinnin vähentäminen

Perinteisessä ohjelmistoprojektissa luotetaan pitkään ja yksityiskohtaiseen suunnitteluun, jonka pohjalta luodaan hyvin tarkat suunnitelmat siitä miten ohjelmisto tulee toteuttaa. Nämä suunnitelmat muodostetaan yleensä dokumentteina, diagrammeina ja muistioina. (Holcombe 2008, 9.)

Vaatimusdokumenttien tekeminen voi kestää jopa kuukausia, jolloin muodolliset dokumentit ovat jo valmistuessaan vanhentuneita. Nykyajan organisaatioilla ei ole varaa työskennellä vanhentuneen tiedon kanssa tai tehdä asioita useampaan kertaan. Tietyntasoista dokumentaatiota tarvitaan aina, mutta vaatimusten määrittelyyn kasvotusten käytävä keskustelu ja minimaalinen dokumentointi ovat parhaimmat menetelmät. (Cooke 2010, 252.)

Ketterien menetelmien tärkeimpiä kohtia on muuntaa suunnitteluvaihe jatkumaan koko projektin keston ajan, jolloin voidaan vastata ketterästi muuttuviin asiakasvaatimuksiin. Kuten aikaisemmin on jo todettu, suunnitelmat yleensä muuttuvat kesken projektin ja perinteinen, hyvin yksityiskohtainen ja pitkäkestoinen suunnitteluvaihe, ei sovi tällaiseen projektiin. Kun suunnitelmat muuttuvat kesken projektin, vaikeasti ylläpidettävistä dokumenteista tulee epäluotettavia.

Ketterissä projekteissa tulee olla avoin mahdollisille muutoksille ja jättää dokumentointi tästä näkökulmasta vähemmälle. Tulee kuitenkin muistaa, että lähdekoodin ja testitapausten dokumentointia ei voida unohtaa, koska sen avulla voidaan jälkikäteen selvittää, miten ohjelma toimii. Tämän dokumentoinnin tulee olla selkeää ja ajantasaista, jokaisessa projektin vaiheessa. (Holcombe 2008, 9.)

4.3.6 Ohjelmiston ihmisresurssien hallinta

Ohjelmistoprojektin ihmisresurssit muodostuvat usein hyvin luovista henkilöistä. Jos projektin suunnitteludokumentit ovat hyvin yksityiskohtaisia tai kattavia, voivat tällaiset ihmiset usein tuntea olonsa kahlituiksi ja täyden hyödyn saaminen irti heistä, muodostuu hankalaksi. (Holcombe 2008, 10.)

Ohjelmistoprojektit ovat usein hyvin ihmiskeskeisiä projekteja. Projektin onnistuminen on suora seuraus sen parissa työskentelevien ihmisten ja tiimien taidoista, motivaatiosta, sekä hyvinvoinnista. Jotta saisimme kaiken hyödyn irti projektin arvokkaimmasta resurssista, projektien tulee käsittää sen parissa työskentelevät ihmiset älykkäinä ja osaavina henkilöinä. On tärkeää ottaa tiimin jäseniä mukaan päätöksen-

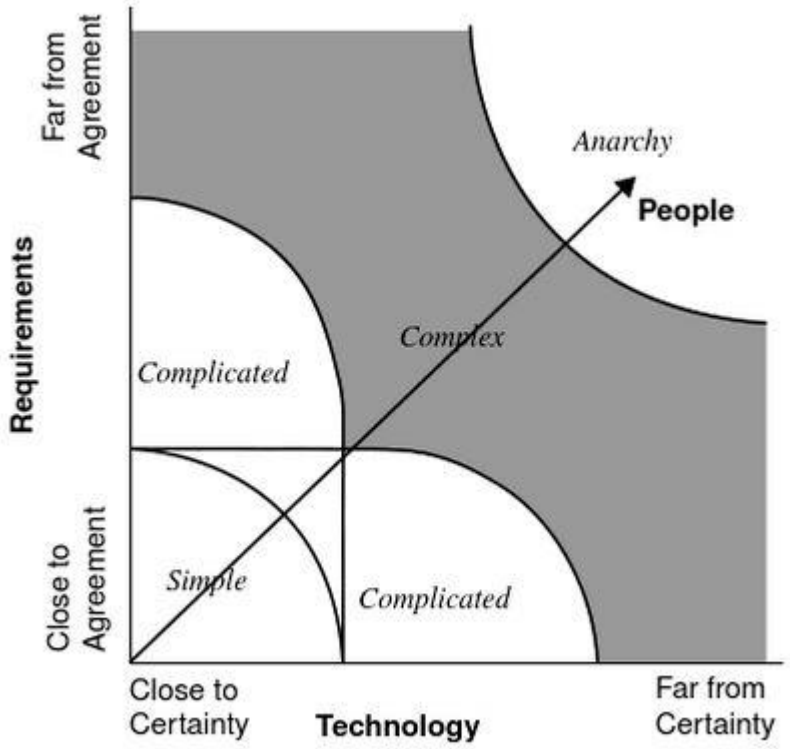
tekoon mahdollisimman paljon. Tällöin ihmiset kokevat saavansa vaikuttaa päätöksiin ja antavat kaiken osaamisensa projektille. Projektin henkilöstön on pystyttävä reagoimaan muutoksiin nopeasti ja ilman, että he tuntevat oloaan uhatuiksi. Tämä ei tapahdu ylhäältä päin käskemällä, vaan tiimit pitää ottaa mukaan päätöksentekoon mahdollisimman aikaisessa vaiheessa. (Holcombe 2008, 10-11.)

Ketterän ohjelmistokehityksen julistuksessa mainitaan, että yksilöitä ja kanssakäymistä arvostetaan enemmän kuin prosesseja ja työkaluja. Monet ketterän ohjelmistokehityksen puolestapuhuja uskovat, että kun tiimi muodostetaan oikein ja sen hyvinvoinnista huolehditaan tarpeeksi, se voi saavuttaa mitä tahansa. Työkaluilla ja prosesseilla ei ole merkitystä, jos projektissa työskentelevät ihmiset eivät toimi yhdessä. (Koch 2004, 53.)

Ketterissä projekteissa ei tule myöskään unohtaa asiakasta, joka on toinen henkilöstöresurssi projektissa. Asiakas on yleensä huolissaan projektin etenemisestä, koska on usein vastuussa projektin onnistumisesta omassa organisaatiossaan. Kommunikointi on tärkeässä osassa myös asiakkaan suuntaan. Sen sijaan, että teetettäisiin statusraportteja ja dokumentteja, voidaan asiakas ottaa mukaan kunkin iteraation lopuksi tapahtuvaan katselmointiin. Kun asiakas näkee kehityksen ja etenemisen omin silmin tasaisin väliajoin, hän saa parhaimman käsityksen projektin tilasta. Katselmointi luo myös kommunikointikanavan kehitystiimin ja asiakkaan välillä ilman turhaa byrokratiaa. (Holcombe 2008, 11.)

4.3.7 Empiirinen prosessimalli

Ohjelmistotuotannon monimutkaisuutta ja ennustettavuutta voidaan arvioida nk. Staceyn kaaviolla (kuvio 8). Kaavio kuvaa eri dimensioiden varmuutta suhteessa niiden ennustettavuuteen. Ohjelmistotuotannossa mallia on käytetty esittämään kolme ohjelmistotuotannon osa-aluetta; vaatimuksia, teknologiaa ja ihmisiä. (Schwaber, Kubacki & Sutherland 2012, 12.)



Kuvio 8. Staceyn kaavio (Schwaber, Kubacki & Sutherland 2012, 11)

Kaaviosta voidaan nähdä, että ohjelmistotuotanto on parhaimmillaankin monimutkaista ja pahimmillaan jopa kaoottista. Teknologia on harvoin täysin tunnettua ja ymmärrettyä. Kehitettävä ohjelmisto on usein liitoksissa muihin ohjelmistoihin eritasoisten liittymien kautta. Teknologialla on niin monia riippuvuuksia muihin järjestelmiin, ettei sitä voida mitenkään pitää riskittömänä. Vaatimukset eivät koskaan ole täysin selkeitä ilman minkäänlaista riskiä muutoksista. Ohjelmistoa kehittävät ihmiset vaikuttavat myös työn kaoottisuuteen omilla mielipiteillään, yhteistyökyvyllään ja mielialoillaan. Kaikkien kolmen dimension epävarmuus vaikuttaa ohjelmistoprojektin työn monimutkaisuuteen. (Schwaber, Kubacki & Sutherland 2012, 11.)

Perinteinen ennustava prosessimalli, kuten vesiputousmalli sopii vain yksinkertaiseen, toistettavaan työhön, jossa ratkaisut ovat jo etukäteen tiedossa. The Standish Group on tehnyt tutkimusta perinteisen prosessimallin projekteista ja todennut onnistuvuusprosentin olevan näillä projekteilla n. 14 %. Ketteriä menetelmiä käyttävissä projekteissa on saavutettu huomattavasti korkeampia onnistumisprosentteja. (Schwaber, Kubacki & Sutherland 2012, 12.)

Empiirisessä prosessimallissa tietoa hankitaan enemmän havainnoimalla, kuin ennustamalla. Tiedämme myös, että empiirinen prosessimalli sopii parhaiten projekteihin missä on enemmän asioita, joita emme tunne, kuin asioita mitä tunnemme. Empiirinen prosessi nojaa kahteen vaatimukseen; Tarkastamiseen ja mukautumiseen, sekä läpinäkyvyyteen. (Schwaber, Kubacki & Sutherland 2012, 18.)

Empiirisessä prosessissa tulee säännöllisesti tarkastaa nykytilanne, jotta voidaan tehdä päätöksiä jatkosta ja saavuttaa paras lopputulos. Jotta voimme tarkastella todellista tilannetta, meillä pitää olla täysi läpinäkyvyys tilanteeseen. Tarkastuksien aikaväli riippuu siitä, kuinka paljon riskejä halutaan ottaa. Mitä enemmän tuntemattomia asioita, sitä enemmän voimme edetä väärään suuntaan ja sitä enemmän suunnan korjaus maksaa. (Schwaber, Kubacki & Sutherland 2012, 18.)

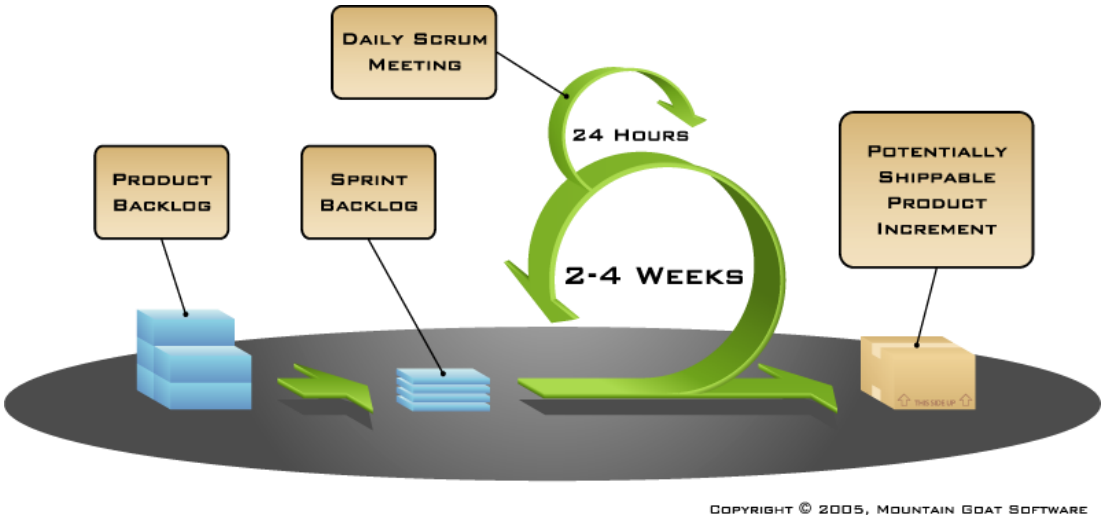
4.4 SCRUM

Scrum on yksi ketteristä menetelmistä. Opinnäytetyön kohdeorganisaatiossa Scrum on valittu ohjelmistoprojekteissa käytettäväksi menetelmäksi jo useita vuosia sitten. Ongelmana on ollut menetelmän käyttöönotto ja jalkautus organisaatiossa. Kohdeorganisaatiossa on osattu ottaa käyttöön kaikki Scrumiin kuuluvat konkreettiset toimenpiteet, mutta perusperiaatteiden jalkautus on epäonnistunut ja jäänyt hyvin vähälle huomiolle. Organisaatio on saanut kaikki Scrumin huonot puolet, ilman sen hyviä puolia.

Jeff Sutherland kehitti Scrumin vuonna 1993. Virallisen kuvauksensa Scrum sai vuonna 1995, kun Ken Schwaber yhdessä Jeff Sutherlandin kanssa, kehittivät menetelmän lopulliseen muotoonsa. Scrum on nopeimmin kasvava ketterä menetelmä ja nykyään n. 50 % kaikista ketteriin menetelmiin nojautuvista projekteista käyttää Scrum-menetelmää. (Rico, Sayani & Sone 2009, 25-26.)

Scrum on iteratiivinen menetelmä (kuvio 9). Menetelmä on hyvin yksinkertainen ja muodostuu kolmesta roolista, kolmesta artefaktista ja viidestä tapahtumasta. Scrum sitoo nämä asiat yhteen yhteisillä säännöillä ja käytännöillä. Ihmisiä, jotka kehittävät ohjelmistoa kutsutaan Scrum tiimiksi (Scrum Team). Tiimi muodostuu henkilöstä,

joka haluaa ohjelmiston kehitettävän (Tuoteomistaja / Product Owner), managerista (Scrum Master), sekä tiimistä. Useampi tiimi voi kehittää samaa ohjelmistoa, mutta yhdessä tiimissä voi olla ainoastaan yksi tuoteomistaja ja Scrum Master. (Schwaber, Kubacki & Sutherland 2012, 57.)



Kuvio 9. Scrum-menetelmän kuvaus

Kaikki tapahtumat Scrumin sisällä ovat ajallisesti rajattuja. Tarkoitus on säilyttää tehokkuus. Scrumin sisällä on tarkoitus työskennellä, ei miettiä työskentelemistä. (Schwaber 2004, 477.)

4.4.1 Suunnittelupalaveri eli Sprint Planning

Kuten ketterien menetelmien periaatteisiin kuuluu, myös Scrum perustuu lyhyissä iteraatioissa tapahtuvaan kehitykseen. Scrumissa iteraatioita kutsutaan sprinteiksi (suom. pyrähdys). Nämä sprintit ovat maksimissaan 30 päivän pituisia ja jokaisen sprintin lopussa saavutetut tulokset katselmoidaan yhdessä sidosryhmien kanssa. Näin tuotteen tilasta saadaan todellinen kuva ja voidaan päättää paras mahdollinen suunta, mihin seuraavassa sprintissä pyritään. (Schwaber, Kubacki & Sutherland 2012, 18.)

Jokainen sprintti alkaa suunnittelupalaverilla (Sprint Planning), jossa koko tiimi on paikalla. Tarkoituksena on suunnitella, mitä seuraavassa sprintissä tehdään. Jotta suunnitelma voidaan tehdä, tulee kaikkien tietää mihin ohjelmistolla pyritään. Mil-

loin ohjelmisto on halutun kaltainen tai milloin se on epäonnistunut. Tuoteomistajan vastuulla on välittää tiimille tämä visio tuotteesta. Tuoteomistaja esittää tiimille priorisoitua listaa halutuista vaatimuksista, joista tiimi valitsee tehtäväksi niin paljon, kuin he arvioivat ehtivänsä tekemään tulevan sprintin aikana. Tiimi arvioi, kuinka kauan kunkin vaatimuksen täyttämiseen kuluu aikaa ja muodostaa vaatimuksista listan ominaisuuksia, jotka tullaan kehittämään tulevan sprintin aikana. Suunnittelupalaveri voidaan päättää, kun kaikki tiimissä tietävät mitä tulevan sprintin aikana tullaan tekemään. (Schwaber, Kubacki & Sutherland 2012, 58.)

Sprint planning voidaan jakaa kahteen eri osaan. Tiimin tavoitteena ensimmäisen osan aikana, on valita sellaiset vaatimukset tuoteomistajan esittämältä listalta, jotka he uskovat kykenevänsä kehittämään tulevan sprintin aikana. Suunnittelupalaverin toinen osio tapahtuu välittömästi ensimmäisen osion jälkeen ja sen aikana tiimi päättää miten se kääntää valitun vaatimukset sellaisiksi ominaisuuksiksi, joka tuovat lisäarvoa tuotteeseen. Toisen osion tuloksena tulee olla sprintin tehtävälista (engl. Sprint Backlog), lista konkreettisista toimista, joilla tiimi pääsee tavoitteeseensa, toimitettavaan lisäarvoa sisältävään tuotteeseen. (Schwaber 2004, 4126.)

4.4.2 Iteraatio eli Sprint

Tiimi aloittaa määriteltyjen ominaisuuksien kehittämisen välittömästi suunnittelupalaverin jälkeen. Tuoteomistaja lupaa tiimille, etteivät tuotteen vaatimukset muutu kuluvan sprintin aikana. Vaikka Scrum kuuluu ketteriin menetelmiin, joissa juuri vaatimusmuutoksien hallinta on yksi tärkeimmistä asioista, eivät vaatimukset saa muuttua kesken sprintin. Jos tuoteomistaja näkee tarpeen muuttaa vaatimuksia, voidaan se tehdä seuraavan sprintin suunnittelupalaverissa tai äärimmäisissä tapauksissa, voidaan kuluva sprintti keskeyttää ja aloittaa uusi uudella suunnittelupalaverilla.

Tiimi pitää sprintin aikana päivittäin maksimissaan 15 minuuttia kestävän päiväpalaverin (engl. Daily Scrum), jossa he kertovat muille tiimin jäsenille mitä he ovat tehneet viime palaverin jälkeen ja mitä tulevat tekemään seuraavaan mennessä. Myös mahdolliset esteet tekemiselle tulee tuoda esiin tässä palaverissa. Daily Scrumin aikana tiimi synkronoi tekemisensä keskenään. Palaverin avulla tiimin jäsenet kom-

munikoivat jatkuvasti keskenään ja ovat tietoisia siitä mitä muut ovat tekemässä. Jokaisen tiimin jäsenen tulee sitoutua siihen mitä hän lupaa tehdä ennen seuraavaa palaveria. (Schwaber, Kubacki & Sutherland 2012, 59.)

Tiimin jäsenillä on ainoastaan kaksi hallinnollista velvollisuutta sprintin aikana. Heidän tulee osallistua päivittäiseen päiväpalaveriin, sekä pitää sprintin tehtävälista ajan tasalla. Näiden kahden velvollisuuden lisäksi tiimillä ei ole muita velvollisuuksia. Tiimillä pitää olla täysi työrauha suunnittelupalaverissa päätettyjen asioiden tekemiseen, eikä heitä saa häiritä muilla tehtävillä. (Schwaber 2004, 4186.)

Tiimi työskentelee keskeytyksettä koko sprintin keston ajan. Scrum Master ohjaa tiimiä työskentelemään Scrumin periaatteiden mukaisesti ja pitää huolen, ettei tiimiä häiritä kesken sprintin. Jotta tiimi saa työskentelyrauhan koko sprintin ajaksi, vaatii se koko organisaatiolta sitoutumista noudattamaan Scrumin periaatteita. Scrum Master ei ole korkeammassa asemassa muihin tiimin jäseniin nähden, vaan hän ainoastaan ohjaa tiimiä toimimaan Scrumin periaatteiden mukaisesti ja toimii järjestävänä tahona kaikissa Scrumin tapahtumissa.

4.4.3 Sprinttikatselmus eli Sprint Review

Sprintin päätyttyä järjestetään katselmointitilaisuus (engl. Sprint Review), jossa koko tiimi ja sidosryhmien edustajat katselmoivat sprintin tulokset. On siis hyvin tärkeää, että sprintin aikana tuotetaan toimivia kokonaisuuksia. Ominaisuuksia voidaan jatkaa seuraavilla sprinteillä, mutta keskeneräistä toiminnallisuutta ei voida katselmoida, joka rikkoo scrumin periaatteita. (Schwaber, Kubacki & Sutherland 2012, 59.)

Katselmoinnin aikana nousee usein esiin uusia vaatimuksia ja ideoita. Pelkästään toiminnallisuuden näkeminen käytännössä herättää usein uusia ideoita. Uudet ideat menevät tuotteen kehitysjonoon (engl. Product Backlog), joka pitää sisällään kaikki tuotteelle asetetut vaatimukset. Tuoteomistaja priorisoi tämän listan vaatimuksista ja esittää ne halutessaan sprintin suunnittelupalaverissa. (Schwaber, Kubacki & Sutherland 2012, 59.)

Jokaisessa katselmoinnissa voidaan päättää jatkaa kehitystä seuraavalla sprintillä tai keskeyttää kehitys kokonaan. Jos kehitystä päätetään jatkaa seuraavalla sprintillä, järjestetään ensin sprint retrospektiivi (engl. Sprint Retorspective), jossa tiimi keskittyy parantamaan prosessia. (Schwaber, Kubacki & Sutherland 2012, 59.)

4.4.4 Retrospektiivi eli Sprint Retrospective

Scrumin periaatteisiin kuuluu prosessin jatkuva parantaminen. Tätä varten, jokaisen sprintin päätteeksi järjestetään retrospektiivi, jossa tiimi kerääntyy yhteen pohtimaan miten prosessia ja tiimin toimintaa voitaisiin seuraavalla sprintillä parantaa.

Parannusehdotukset voivat liittyä työmääräarvioiden tekemiseen tai kommunikointiin tiimin sisällä. Lopuksi tiimi koostaa yhteen asiat, joita he aikovat muuttaa seuraavalla sprintillä. Tällä tavoin tiimillä on mahdollisuus jatkuvaan työnsä ja työskentelynsä parantamiseen. (Schwaber, Kubacki & Sutherland 2012, 60.)

On tärkeää huomata, että Scrum Master ei ole palaverissa mukana tarjoamassa tiimille valmiita vastauksia, vaan ainoastaan palaverin järjestäjän roolissa. Palaverissa päätetyt asiat voidaan lisätä seuraavan sprintin kehitysjonoon ei-funktionaalisina asioina. Retrospektiivit, jotka eivät johda muutoksiin, ovat turhauttavia koko tiimille. (Schwaber 2004, 4226.)

4.5 KANBAN

Sana Kanban tulee Japanin kielestä ja tarkoittaa kirjaimellisesti taulua tai kylttiä. Kanbanin katsotaan saaneen alkunsa 1950-luvulla Toyotan tuotantojärjestelmästä. Kanban ei saavuttanut maailmalla kovinkaan suurta suosiota, kuin vasta 1970-luvulla. Taiichi Onho kehitti Kanban-järjestelmän hallitakseen Toyotan tuotantoa eri prosessien välillä ja ottaakseen käyttöön Just-in-time (JIT) tuotannonohjauksen. Kanbanin avulla Toyotan tuotannossa kyettiin minimoimaan samanaikaisesti työn alla olevien työtehtävien määrä ja sitä kautta myös varastointikulut. JIT-tuotannonohjaus pyrkiikin juuri minimoimaan varastointia ja tuottamaan tuotteita vain kysynnän mukaan. Vaikka Toyota käytti Kanban-järjestelmää alun perin vähen-

tämään tuotannon kustannuksia ja parantamaan koneiden käyttöastetta, se käyttää järjestelmää nykyään myös paljastamaan tuotannon pullonkaulat ja mahdolliset optimoinnin kohteet. (Gross & McInnis 2003, 1-2.)

Kanban-järjestelmässä operaattorit käyttävät visuaalisia signaaleja osoittaakseen työn etenemisen. Näistä signaaleista nähdään välittömästi koko tuotantolinjan tila tietyllä ajanhetkellä. Kanban-järjestelmässä operaattorit tuottavat tuotteita, ennusteen sijasta, todellisen käytön perusteella. Jotta järjestelmää voidaan kutsua todelliseksi Kanban-järjestelmäksi, sen hallinnoiman prosessin pitää täyttää seuraavat kriteerit. (Gross & McInnis 2003, 3.):

- Toimita tuotteita vain korvataksesi se minkä asiakas on kuluttanut
- Toimita tuotteita vain, kun niitä tilataan

Kanban-järjestelmää voidaan käyttää missä tahansa prosessissa, missä halutaan rajoittaa prosessissa olevien asioiden samanaikaista määrää. Kanban-järjestelmässä on vain tietty määrä kortteja (Kanban). Järjestelmässä olevien korttien määrä on yhtä kuin järjestelmän kapasiteetti. Uusi työ voidaan aloittaa vain, kun järjestelmässä on vapaa kortti. Kortti liikkuu työn mukana läpi prosessin ja lopulta vapautuu, kun työ on saatu päätökseen. Kortin vapauduttua voidaan aloittaa uusi työ. Tätä kutsutaan veto-systeemiksi. Uutta työtä vedetään prosessiin, kun sille löytyy kapasiteettia, sen sijaan, että prosessiin työnnetään työtä kysynnän mukaan. Veto-mallia ei ole mahdollista ylikuormittaa kunhan järjestelmän kapasiteetti on asetettu oikein. (Anderson 2010, 511.)

Myös ohjelmistotuotannossa voidaan käyttää Kanban-järjestelmää. Ohjelmistotuotannon Kanban-järjestelmää kutsutaan usein virtuaaliseksi Kanban-järjestelmäksi. Järjestelmässä käytetään kortteja kuvaamaan signaalien sijasta, itse työtä. Järjestelmässä ei ole fyysisiä kortteja signaaleilla, josta syystä järjestelmää kutsutaankin virtuaaliseksi Kanbaniksi. Signaali vetää uutta työtä prosessiin päätellään visuaalisesta työkorttien määrästä, joka vähennetään prosessin kapasiteetistä. (Anderson 2010, 536.)

Ohjelmistotuotannossa ja ketterässä kehityksessä käytettävässä Kanban-metodissa on viisi perusominaisuutta, joiden avulla organisaatiossa saadaan muokattua ketterä organisaatio;

- Visualisoi työnkulku
- rajoita samanaikaisen työn määrää
- mittaa ja hallinnoi työnkulkua
- tee eksplisiittiset prosessikäytännöt sekä
- käytä malleja löytääksesi parannuskohteita.

Nämä viisi ominaisuutta ovat olleet mukana kaikissa onnistuneissa ohjelmistotuotannon Kanban-sovellutuksissa. (Anderson 2010, 561.)

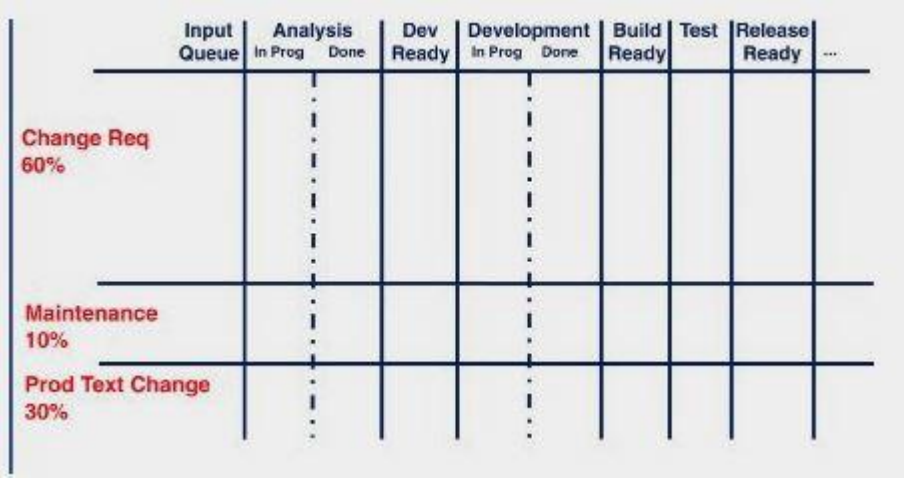
4.5.1 Visuaalinen työnkulku

Kanban-järjestelmän tavoitteena ei ole keksiä uutta, vaan parantaa nykyistä prosessia. Kanbanin tärkein työväline onkin visuaalinen työnkulku, joka kuvastaa todellista tilannetta ja sitä miten yrityksessä tehdään asioita. On tärkeää, että prosessin visualisoinnissa ei kuvata teoreettista ideaalitilannetta vaan todellinen työnkulku, mitä yrityksessä sillä hetkellä noudatetaan. Työntekijät eivät pysty käyttämään työnkulkua, jonka mukaan he eivät kuitenkaan toimi. (Anderson 2010, 1347.)

Työnkulkua kuvatessa on tärkeää löytää prosessin alku- ja loppupiste, sekä määritellä ne liittymäpisteet, missä prosessi liittyy muuhun organisaatioon. Työnkulku pystytään näin rajaamaan tarkasti, eikä muun organisaation tarvitse muuttaa omia toimintatapojaan. Kanbanin perusajatuksena on tuoda mahdollisimman vähän muutoksia, jotta vaadittavat, tärkeät muutokset, saadaan tehtyä. Muutos ei tule onnistumaan, jos organisaatiossa pyritään muuttamaan kaikki kerralla. (Anderson 2010, 1362.)

Kun työnkulun alku- ja loppupiste ovat selvillä, voidaan määritellä tehtävän työn eri tyypit. Tyyppejä voivat esimerkiksi olla ohjelmistovirheet ja uudet ominaisuudet. Tyypit voidaan eritellä visuaalisesti toisistaan, jos tällainen nähdään tarpeelliseksi, mutta on tärkeää tietää erityyppisten töiden työvaiheet. Kaikentyyppiset työt eivät välttämättä tarvitse samoja työnkulun vaiheita. Jokaiselle tyypille voidaan myös allokoida erillinen kapasiteetti järjestelmässä ja tämä voidaan kuvata työnkulussa.

Kanbanin työnkulku voidaan kuvata lukemattomilla eri tavoilla (kuvio 10). Onkin tärkeää aloittaa jostain ja tehdä jatkuvasti pieniä parannuksia. (Anderson 2010, 1384-1416.)



Kuvio 10. Esimerkki visuaalisesta työnkulusta ja miten erityyppisille töille on määritelty kapasiteetti (Anderson 2010, 1457)

4.5.2 Samanaikaisen työn rajoitukset

Yksi Kanbanin tärkeimmistä säännöistä on asettaa jokaiseen työnkulun vaiheeseen rajoitus samanaikaisen työn määrästä. Rajoituksen määrittelemiseksi on useita eri tapoja. On kuitenkin suositeltavaa noudattaa periaatetta, että jokaisella tiimin jäsenellä ei saisi olla työn alla, kuin yksi asia kerrallaan. On myös tärkeää sopia rajoista yhteisesti muiden asianomistajien kanssa. Yhteisellä päätöksellä säästytään ongelmatilanteissa syyttelyltä. (Anderson 2010, 2145.)

Useissa työnkuluissa tarvitaan jonkinlaisia jonoja tai puskureita huolehtimaan työn sujuvuudesta. Puskureita tarvitaan etenkin silloin, kun samanaikaisen työn rajoitus on asetettu hyvin pieneksi. Ilman puskureita työnkulku sisältäisi liikaa pysähdyksiä työn normaalin vaihtelun vuoksi. Kaikki työ ei ole täsmälleen yhtä suuritöistä, eivätkä kaikki työntekijät työskentele täsmälleen samalla tahdilla. Puskureille tulee myös asettaa rajoitukset, kuten muillekin työnkulun vaiheille. Puskureiden rajoituksien tulisi olla mahdollisimman pieniä, jotta ne eivät salli järjestelmän väärinkäyttöä. Puskureita tarvitaan kuitenkin prosessin pullonkaulojen vuoksi. Vaikka puskurit kasvat-

tavatkin työn läpikulun aikaa, ne kuitenkin mahdollistavat työn ennustettavuuden. (Anderson 2010, 2177.)

Ohjelmistotuotannossa tuotepäällikkö tuo prosessiin lisää työtä. Työnkulun sisääntulevan työn jonon tulee myös sisältää määrärajoituksia. Optimaalisesti sisääntulevan työn rajoitus on sellainen, että sisääntuleva jono ehditään lähes tyhjentämään, kunnes tuotepäällikkö lisää siihen uutta työtä. Tärkeintä on kuitenkin se, ettei sisääntuleva jono ehdi koskaan tyhjentyä täysin. Tällöin prosessiin ei kyetä ottamaan lisää työtä vaikka kapasiteettia olisikin vapaana. Pahimmassa tapauksessa koko työnkulku siis pysähtyy. On jokaisen tiimin tehtävä sopia tuotehallinnan kanssa paras tapa pitää huolta sisääntulevan työn määrästä. (Anderson 2010, 2193.)

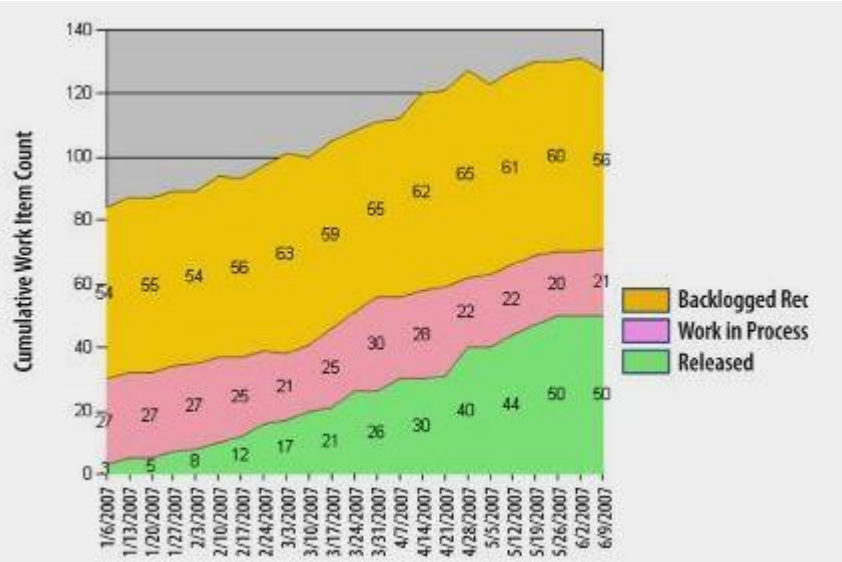
Kuten muutkin ketterät menetelmät, myös Kanban on empiirinen prosessimalli. Työnmäärän rajoituksia ei tule miettiä loputtomasti. On tärkeää päättää aloitus ja tehdä sen jälkeen jatkuvia pieniä muutoksia, kunnes samanaikaisen työn rajoituksista on saatu optimaalisia kyseisen yrityksen prosessille. Tästä syystä onkin tärkeää pystyä mittaamaan työnkulkua ja miten työ etenee prosessissa. (Anderson 2010, 2193.)

4.5.3 Työnkulun mittaaminen

Työnkulun mittaamisen avulla työnkulkua voidaan kehittää jatkuvasti paremmaksi. Mittaustuloksia käytetään usein myös raportoinnissa. Koska Kanban muuttaa tiimin tapaa tehdä yhteistyötä organisaation muiden tahojen kanssa, myös raportointi on hieman erilaista, kuin perinteisessä projektimallissa. Kanban-järjestelmässä tehdään työtä jatkuvana virtana ja tästä syystä mielenkiintoista ei ole se, onko projekti aikataulussa tai noudatetaanko tiettyä suunnitelmaa. Raportoinnin tulee osoittaa, että järjestelmä toimi, että järjestelmä on ennustettava ja että järjestelmää kehitetään jatkuvasti paremmaksi. (Anderson 2010, 2567.)

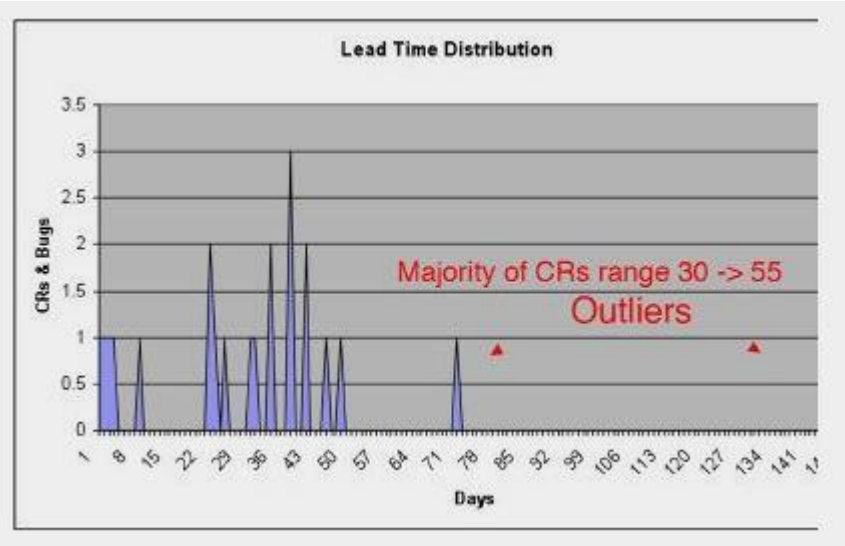
Kanban-järjestelmän tärkein mittari on kumulatiivinen työnkulun raportti (kuvio 11). Raportti osoittaa sen, kuinka hyvin Kanban toimii. Raportti näyttää ajan myötä samanaikaisen työn määrän jokaisessa työnkulun vaiheessa. Ideaalitilanteessa raportti nousee tasaisesti ja niin, että jokainen vaihe pysyy jatkuvasti samankokoisena. Kun

raporttia luetaan horisontaalisesti, saadaan myös toinen tärkeä Kanban-järjestelmän tunnusluku; läpimenoaika. Läpimenoaika kertoo, kuinka nopeasti työ kulkee läpi koko prosessin. Läpimenoaikaa voidaan käyttää prosessin ennustettavuudessa. Jatkuvasti kehittyvän organisaation läpimenoaika lyhenee ajan kuluessa, jolloin työtä saadaan valmiiksi entistä nopeammin. On hyödyllistä raportoida toteutunut läpimenoaika suhteessa ennustettuun aikaan. (Anderson 2010, 2583-2589.)



Kuvio 11. Esimerkki kumulatiivisesta työnkulun raportista (Anderson 2010, 2583)

Muut mittarit ja raportit riippuvat käytössä olevasta työnkulusta ja käytössä olevista työn tyypeistä. On tärkeää, että mittarit ja raportit muodostetaan niin, että ne auttavat paljastamaan mahdollisia ongelmakohtia. Näin raporteista saadaan paras mahdollinen hyöty. Läpimenoajan raportista saadaan hyödyllisempi näyttämällä töiden lukumäärä suhteessa eri läpimenoaikoihin (kuvio 12). Näin ongelmalliset työt nähdään välittömästi ja niihin voidaan pureutua tarkemmin perimmäisen syyn selvittämiseksi. Pelkkä läpimenoajan keskiarvon seuraaminen ei vielä osoita mahdollisia ongelmakohtia. (Anderson 2010, 2588.)



Kuvio 12. Esimerkki toteutuneiden läpimenoaikojen raportista (Anderson 2010, 2588)

4.6 Tuotteen elinkaaren hallinta (PLM)

Ketterät menetelmät ja tuotekehitysprosessi vaikuttavat ohjelmistoprojektissa vain sen kehitysvaiheeseen. Ohjelmistoprojektissa on useita muitakin työvaiheita, jotka kuvataan ja käsitellään kohdeorganisaation PLM-prosessikuvauksessa. Tuotteen elinkaari pitää sisällään kaikki työvaiheet koko organisaation kannalta, aina projektin vaatimusten määrittelemisestä, sen ylläpitoon saakka.

Myös PLM-prosessi on vain yksi osa suurempaa prosessien kokonaisuutta. Kaikki alkaa asiakastarpeista ja niiden täyttämisestä ja päämääränä on asiakastyytyväisyys, kun asiakkaalle pystytään toimittamaan valmis ja toimiva palvelu, joka vastaa heidän tarpeitaan. PLM-prosessin lisäksi kohdeyrityksen prosesseista löytyy yrityksen strategian määrittelyyn oma prosessinsa. Kaikki yrityksen toiminta saa alkunsa yrityksen visiosta ja strategiasta. Kaikkien prosessien on ohjattava yritystä yhteiseen suuntaan, joka on määritelty yrityksen strategiassa ja visiossa.

PLM-prosessin lisäksi prosesseista löytyvät toisiinsa linkittyneet prosessit myynnille, tuotannolle, asiakasimplementaatioille, sekä mm. laskutukselle. Osa prosesseista eroaa sen mukaan, minkä liiketoimintayksikön prosessista on kyse, mutta PLM-prosessi on kaikille yksiköille sama.

PLM-prosessi pitää sisällään kolme erillistä aliprosessia. Vaatimusten määrittelyn, roadmappingin, sekä itse tuotekehityksen. Vaatimusten määrittelyn prosessi kuvaa organisaation tasolla, miten ideat ja ajatukset uusista palveluista kerätään, analysoidaan ja priorisoidaan yhteisellä ja standardoidulla tavalla.

Roadmapping tarkoittaa, tuotteen tulevaisuuden suunnittelua. On tärkeää, että jokaisella yrityksen tuotteella on suunnitelma siitä, mihin suuntaan tuotetta halutaan kehittää. Roadmapping tehdään yleensä hyvin yleisellä tasolla, eikä se ole lupaus asiakkaalle tulevista ominaisuuksista, vaan sen hetkinen ajatus siitä, mihin suuntaan tuote on menossa. Roadmapping-prosessi kuvaa sen, miten tuotepäällikön tulee valmistella tuotteensa roadmap ja miten se hyväksytään liiketoimintayksiköissä.

Tuotteen roadmapin muodostaminen aloitetaan tunnistamalla tuotteen tarpeet. Tunnistetuista tarpeista valitaan tärkeimmät ja kiireellisimmät. Tämän jälkeen voidaan tehdä suunnitelma, minkä avulla tarpeet saadaan parhaiten täytettyä. Tarpeita voivat olla asiakkaiden, markkinoiden, lain vaatimia tai teknisiä tarpeita. (Hunt & Killen 2008, 41.)

Kolmantena aliprosessina on tuotekehitysprosessi, jonka osana myös tässä opinnäytetyössä kehitettävä prosessi toimii. PLM-tuotekehitysprosessi kuvaa tuotekehityksen kuitenkin korkeammalla tasolla, kuin mitä tässä opinnäytetyössä kehitettävä prosessi tekee. Kehitettävä prosessi on laajemmassa tuotekehitysprosessissa vain yksi alikohta. Näin ollen näillä kahdella prosessilla ei ole päällekkäisyyksiä, vaikka molempia kutsutaankin tuotekehityksen prosesseiksi. On kuitenkin tärkeää, että molemmat prosessit tukevat toisiaan, koska ne toimivat toistensa kanssa. Koska näitä kahta prosessia kehittää kaksi erillistä tahoa, on tärkeää pyrkiä toimimaan mahdollisimman paljon yhdessä ja ottamaan huomioon mahdolliset muutokset toisessa prosessissa.

Kohdeorganisaatiossa ohjelmistoa kehitetään versioittain. Jokaisen version kehittäminen tehdään omana tuotekehitysprosessissa. Tuotekehitysprosessi saa alkunsa roadmapping-prosessin tuloksista, jolloin tiedetään mitä seuraavassa ohjelmiston versiossa halutaan kehitettävän. Tämän jälkeen muodostetaan tuoteversiolla priorisoitu kehitysjono, jonka pohjalta pystytään muodostamaan julkaisusuunnitelma. Kun

julkaisusuunnitelma on hyväksytty, voidaan aloittaa tuotteen kehittäminen. Kehittäminen tapahtuu tässä opinnäytetyössä kehitettävän prosessin mukaisesti. Kun tuotekehitys on valmis ja tuotteen hyväksyntätestaus on onnistunut, voidaan tuote siirtää eteenpäin muille yksiköille. Tuotekehitysprosessi päättyy siihen kun tuote on onnistuneesti toimitettu asiakkaalle.

On tärkeää huomata, että osana PLM-prosessia määritelty tuotekehitysprosessi kattaa toimintoja monesta eri yksiköstä eikä pelkästään tuotekehitysprosessista. Tuotekehitysyksikköä koskee ainoastaan tuotteen kehittäminen ja konkreettinen tuotekehitys, joka suoritetaan tuotekehitysprosessin mukaisesti. Tämä on yksi syy siihen, että tässä opinnäytetyössä kehitettävä tuotekehitysprosessi on rajattu vain tähän osaan suurempaa PLM-prosessia. Suuressa kohdeorganisaatiossa ei tämän opinnäytetyön tiimoilta ole mahdollisuuksia määritellä laajempaa PLM-prosessia tai muuttaa sen osaprosesseja. Myös prosessin jalkauttaminen hankaloituisi huomattavasti, kun puhutaan organisaation ja eri liiketoimintayksikköjen laajuisista prosesseista. Tässä opinnäytetyössä kehitettävä prosessi koskee ainoastaan yhtä yksikköä, jolloin sen onnistunut jalkauttaminen ja käyttöönotto ovat myös opinnäytetyön kannalta ja aikataulussa mahdollista.

Opinnäytetyön kohdeyritystä ollaan vähitellen integroimassa osaksi suuremman organisaation prosesseja ja tästä syystä koko PLM-prosessin jalkautus on hyvin alkuvaiheessa. PLM-prosessi on myös itsessään muuttumassa tietyiltä osin lähiaikoina. Kohdeyrityksen kannalta muutoksia on hyvin paljon ja niiden ottaminen käytäntöön ja onnistunut jalkauttaminen tulee olemaan hyvin haasteellista. Vaikka PLM-prosessin käyttöönotto ei kuulukaan tämän opinnäytetyön piiriin, tulee se omalta osaltaan hankaloittamaan myös tässä opinnäytetyössä kehitettävän prosessin jalkauttamista.

5 PROSESSIN KEHITTÄMINEN

5.1 Prosessin nykytilan kuvaus

Tuotekehitysprosessin kehittäminen ei poikkea muunlaisten prosessien kehittämisestä. Kehitettävä prosessi on pystyttävä rajaamaan, jotta tiedetään mistä prosessi alkaa ja mihin se päättyy. Yhtä tärkeää on myös tietää miten prosessin onnistumista mitataan. Mistä yrityksessä tiedetään, kun prosessi toimii hyvin? Prosessin laajuuden, sekä mittareiden määrittely ovat prosessin kehittämisen ensimmäiset vaiheet. (Page 2010, 51.)

Kehitettäessä uutta prosessia, on hyödyllistä mallintaa nykyinen prosessi. Nykyisen prosessin mallintaminen auttaa ihmisiä näkemään, miten nykyinen prosessi todella toimii. Kehitystyö saadaan sidottua käytäntöön huomattavasti paremmin, kun ymmärretään miten yrityksessä sillä hetkellä toimitaan. Uusi prosessi jää helposti etäiseksi, jos sen kehitystyössä ei ole lainkaan huomioitu yrityksen nykyisiä toimintamalleja. (Page 2010, 78.)

Prosessin mallintamisen lähtökohta on aina prosessin alkupiste. Alkupiste on toiminto, josta koko kuvattava prosessi saa alkunsa. Prosessi kuvataan näin vaihe vaiheelta, kunnes saavutetaan prosessin rajattu päätepiste. Prosessin kuvantamiselle on useita eri tapoja. Prosessi voidaan kuvata yleisellä tai hyvin yksityiskohtaisella tasolla riippuen tapauksen vaatimuksista. Prosessi voidaan kuvata myös useammalla eri tasolla. Yleisempi taso voi sisältää ainoastaan prosessin kriittisimmät toiminnot, kun taas tarkemmalla tasolla on kuvattu kaikki prosessin vaiheet. Eritasoisia malleja voidaan käyttää eri tarkoituksiin. Toteutettua prosessikaaviota voidaan myös hyödyntää toimintaohjeena sekä uusien työntekijöiden perehdytyksessä. (Page 2010, 82–106. Harvard Business Press 2010, 30.)

Prosessin kuvaaminen on helpompaa, kun prosessissa seurataan ensin yhtä polkua loppuun saakka. Jokaisessa vaiheessa on tärkeää kuvata se mitä todella tapahtuu, eikä sitä mitä ohjesääntö sanoo. Kartoittaminen tulee tehdä todellisten työntekijöiden kanssa, koska ainoastaan he tietävät, mitä prosessissa todella tapahtuu. Prosessikar-

tassa tulee kuvata kaikki prosessin vaiheet; virhetilanteet, tarkastukset, poikkeukset, odotukset, sekä kaikki muut mahdolliset tilanteet. Prosessin vaiheiden ajoittaminen voi myös olla hyödyllistä, varsinkin prosessin kehittämisen myöhemmissä vaiheissa. Ajoittamiseen ei kuitenkaan tule käyttää liikaa aikaa ja aika-arviot voivat olla suuntaa antavia. (Flanigan & Scott 1995, 63.)

Prosessia kuvattaessa on tärkeää ottaa huomioon kaikki prosessin suorittamiseen liittyvä tieto. Nämä tiedot eivät suoranaisesti liity prosessin tehokkuuteen tai suorittamiseen, mutta ovat hyödyllisiä arvioitaessa prosessin kehittämisen kokonaisvaikutuksia. Tällaisia tietoja ovat esimerkiksi tieto siitä kuinka useasti prosessi suoritetaan tai kuinka monta tapahtumaa prosessi pitää sisällään. (Harvard Business Press 2010, 28.)

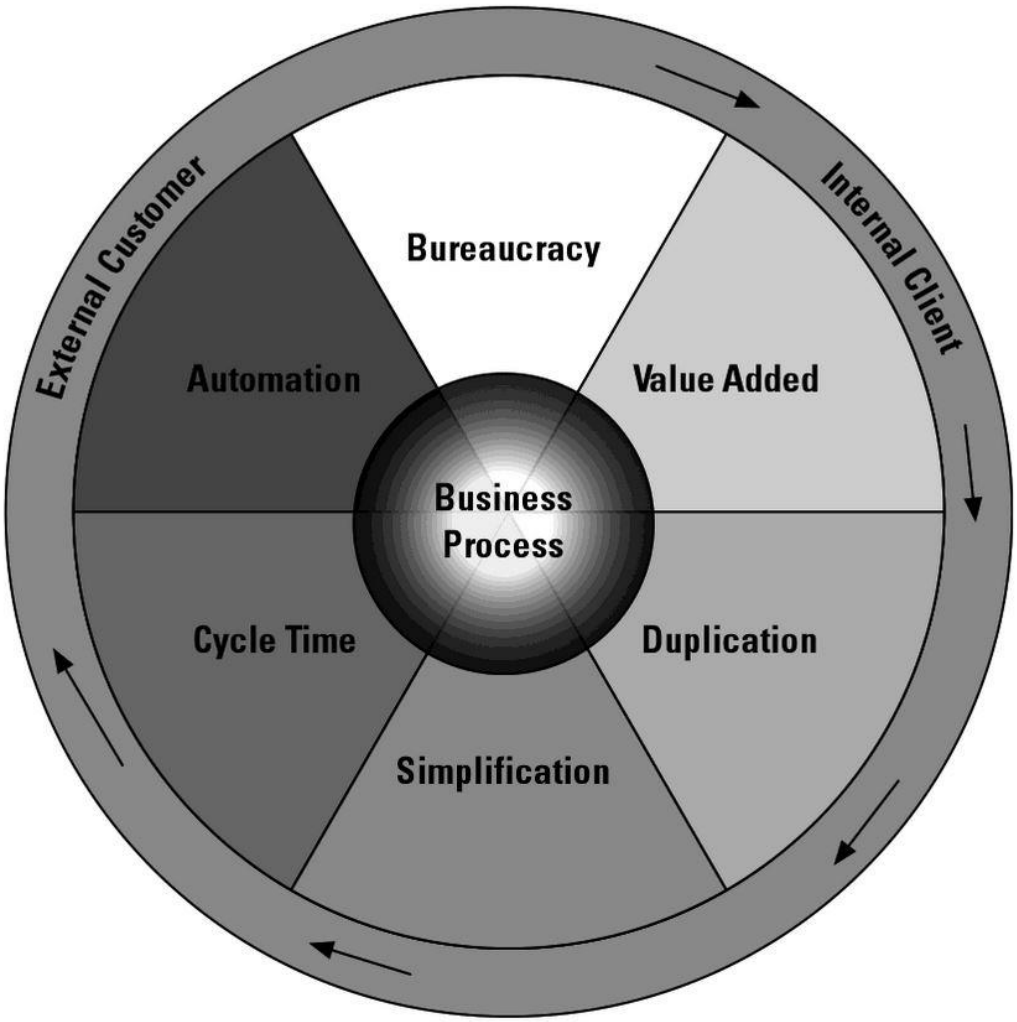
Yhtä tärkeää, kuin prosessin kuvaaminen, on myös sen mittaaminen. Organisaation tulee pystyä mittaamaan prosessin toimintaa, jotta se pystyy kehittämään prosessia. Mittareiden tulee olla johdettuja prosessin perimmäisestä tarkoituksesta ja niiden tulee tukea tämän tarkoituksen täyttymistä. Mittareiden määrittelemisessä tulee olla varovainen ja miettiä mittarin hyödyllisyyttä suhteessa mittausdatan keräämisen kustannuksiin. (Page 2010, 187.)

Ennen kuin voidaan analysoida prosessin suorituskykyä, tulee ymmärtää sen tarkoitus. Tämän jälkeen voidaan luoda mittareita, jotta tiedetään, kuinka hyvin tässä tarkoituksessa suoriudutaan. Lopuksi mittareiden avulla pystytään luomaan ja kehittämään toimintatapoja, jotka parhaiten palvelevat perimmäisen tarkoituksen täyttymistä. Kehittämisprosessin aloittaminen mittareista johtaa suoraan vääristyneeseen tarkoitukseen ja tätä kautta virheellisiin toimintatapoihin. (Armson 2011, 260.)

5.2 Prosessin kehittäminen

Prosessin kehittämisessä on useita eri tekniikoita. Potentiaalisia parannuspaikkoja huomataan usein jo nykyisen prosessin kuvaamisen yhteydessä. Prosessin tarkoitus ja sitä tukevat mittarit ovat kehittämisen perusta, minkä pohjalta prosessia lähdetään kehittämään. Prosessin kehittämisen tekniikkapyörästä nähdään prosessin kehittämi-

sen erilaiset menetelmät (kuvio 13). Prosessipyörässä prosessi sijaitsee keskellä, asiakas ulkorenkaalla ja eri menetelmät prosessin ympärillä. Menetelmiä sovellettaessa tulisi aloittaa keskeltä ylhäältä ja liikkua myötäpäivään, kunnes päästään automatisointiin saakka. Järjestys on tärkeää, koska ei haluta automatisoida tehotonta prosessia. Automatisointiin tulisi siirtyä vasta, kun prosessista on saatu muilla menetelmillä mahdollisimman tehokas. (Page 2010, 142-143.)



Kuvio 13. Prosessin kehittämisen menetelmärengas (Page 2010, 142)

Kuten muukin työ ketterissä menetelmissä, myös ketterän tuotekehitysprosessin kehittäminen tapahtuu iteroiden. Tarkoituksena ei ole tehdä kerralla lopullista prosessia, vaan kehittää prosessia jatkuvasti. Tehtyjen muutoksien vaikutukset nähdään iteraation aikana ja prosessiin voidaan tehdä uusia muutoksia seuraavaa iteraatiota var-

ten. Tällä tavalla toimimalla muutoksien vaikutuksista saadaan todellista tietoa, jonka perusteella voidaan tehdä oikeanlaisia päätöksiä prosessin kehittämisen suunnasta.

5.2.1 Turhan byrokratian poistaminen

Liikemaailman prosesseissa byrokratia vaatii ihmisiä toimimaan monimutkaisesti, muuten yksinkertaisissa ja tehokkaissa prosesseissa. Byrokratia tekee prosesseista joustamattomia ja usein ilman näkyvää syytä. Turhan byrokratian syyksi paljastuu usein halu kontrolloida prosessia, virheiden pelko ja oman selustan turvaaminen. Vaikka kukaan ei myönnä haluavansa byrokratiaa, voi sen poistamisen yrittäminen luoda odottamatonta vastarintaa. (Page 2010, 144.)

Byrokratian tunnistaminen prosessissa on haastavaa, koska useimmat ovat tottuneet siihen. Prosessin jokainen vaihe tulee kyseenalaistaa ja yrittää tunnistaa byrokraattiset työvaiheet. Jokaisen prosessin työvaiheen tulisi lisätä arvoa prosessiin. Byrokraattiset toiminnot eivät useinkaan lisää arvoa ja ne voidaan näin tunnistaa. Poistamalla byrokratiaa, parannetaan myös prosessin läpimenoaikaa. Näin voidaan huomata, että useimmat prosessin kehittämisen menetelmät ovat hyvin tiukasti yhdessä toisiinsa. Byrokratiaa voidaan vähentää, kun toiminto ei tuota lisää arvoa prosessin tarkoituksen täyttymiseksi ja työvaiheita vähentämällä saavutetaan nopeampi läpimenoaika. (Page 2010, 145–147.)

5.2.2 Lisäarvon tuottaminen

Jokainen prosessin vaihe lisää lopputuotteen työmäärää ja kustannuksia ja siksi on tärkeää arvioida tuottaako työvaihe lisää arvoa asiakkaalle. Olisiko prosessin asiakas valmis maksamaan työvaiheesta? Jotta pystytään arvioimaan tuoko prosessin työvaihe lisäarvoa asiakkaalle, tulee pystyä määrittelemään, mitä asiakas pitää arvokkaana. Asiakkaalle voi olla arvokasta; nopeus, tehokkuus, parempi tarkkuus, pienemmät kustannukset tai vain yksi kontaktipiste yritykseen. (Page 2010, 148. Harward Business Press 2010, 45.)

Poistamalla lisäarvoa tuottamattomia työvaiheita, vähennetään myös samalla prosessin byrokratiaa. Joskus kuitenkin työvaiheiden poistamista pystytään paremmin perustelemaan niiden tuottaman lisäarvon kautta, kuin pelkän byrokratian vähentämisen avulla. (Page 2010, 149.)

Vaikka prosessin työvaihe ei välttämättä tuotakaan lisäarvoa asiakkaan näkökulmasta, se voi olla kriittinen yrityksen toiminnan kannalta. Organisaatio on voitu rakentaa niin, että poistamalla tietty lisäarvoa tuottamaton työvaihe, tehdään jonkin toisen organisaation osan toiminta mahdottomaksi. Tällaisissa tapauksissa tulisi kuitenkin pyrkiä minimoimaan työvaiheen kustannukset pienentämällä sen vaikutuksia prosessin kulkuun ja tehokkuuteen. (Page 2010, 149.)

Vaikeimpia työvaiheita poistaa, ovat sellaiset, joita organisaatio pitää tärkeinä omista syistään. Houkutus väittää niiden lisäävän arvoa, on hyvin suuri. Prosessikaaviota läpikäytäessä onkin syytä kyseenalaistaa jokainen työvaihe ja pohtia prosessin tarkoituksen perusteella maksaisiko asiakas tästä työvaiheesta? (Page 2010, 150.)

5.2.3 Toiston eliminoiminen

Toistoa syntyy eniten silloin, kun prosessi rikkoo organisaatiorajoja toisistaan erillään toimivien yksiköiden välillä. Jokainen taho kerää ja ylläpitää omaa versiotaan prosessissa käsiteltävästä tiedosta. Näin tapahtuu useista eri syistä. Joskus toinen taho ei ymmärrä, mitä toinen taho on tekemässä. Toinen taho ei välttämättä luota toisen tahon tekemiseen tai pahimmassa tapauksessa tahot kilpailevat keskenään. (Page 2010, 152.)

Tiedon siirtäminen taholta toiselle voi helposti johtaa virheisiin ja toistoon. Prosessissa tulee tarkastella tarkasti niitä toimintoja, joissa tietoa siirretään taholta toiselle. On arvioitu, että tiedon siirtämisen on yksi suurimmista prosesseihin liittyvistä ongelmista. (Page 2010, 152.)

Helpoin tapa löytää toistoja on tunnistaa tilanteet, joissa useammalla taholla on samalla hetkellä kopio samasta tiedosta. Tiedon kopioiminen aiheuttaa kustannuksia

työmäärässä, sekä kasvavassa tilan tarpeessa. Tiedon useampi kopio aiheuttaa myös ongelmia tiedon luotettavuudessa. Kaikki kopiot eivät välttämättä ole ajan tasalla. Tiedetäänkö mikä kopioista sisältää oikean tiedon? (Page 2010, 154.)

Kun prosessista pyritään poistamaan kaikki toistoa aiheuttavat toiminnot, tulee kiinnittää erityistä huomiota seuraaviin asioihin (Page 2010, 154).

- Luo tiedolle vain yksi lähde
- Poista työvaiheet, joissa useampi työntekijä tekee samaa työtä, kuten luo samanlaisen raportin
- Poista työvaiheet, joissa useampi työntekijä ylläpitää samaa tietoa
- Poista työvaiheet, joissa sama tieto syötetään järjestelmiin useampaan kertaan
- Minimoi tallennustilan tarve

5.2.4 Yksinkertaistaminen

Prosessin yksinkertaistaminen tarkoittaa monimutkaisuuden vähentämistä tai poistamista prosessista, jotta siitä tulee ymmärrettävämpi ja tehokkaampi. Yksinkertaisen prosessin ylläpitäminen on helpompaa ja prosessi joustaa paremmin asiakkaan tarpeisiin. (Page 2010, 154.)

Prosesseista tulee monimutkaisia, kun niihin lisätään organisaation liiketoiminnan vaatimuksista uusia toimintoja. Yksinkertaisista prosesseista kasvaa näin vähitellen monimutkaisia ja byrokraattisia prosesseja. Monimutkaisuus liittyy usein tiedon rakenteeseen ja siksi onkin syytä tarkastella, minkälaista tietoa prosessin eri vaiheissa käsitellään. Käsiteltävä tieto tulisi pitää sisällään vain ehdottomasti tarvittavan tiedon. (Page 2010, 155.)

Prosessin yksinkertaistamisessa huomataan jälleen eri kehittämismenetelmien suhde toisiinsa. Yksinkertaistamalla prosessia, prosessista tulee vähemmän byrokraattinen. Kun prosessista poistetaan toistoa, siitä tulee yksinkertaisempi. Vaikka eri menetelmien välinen ero onkin häilyvä, on suositeltavaa käydä ne läpi yksitellen. Eri menetelmien soveltaminen aiheuttaa prosessin tarkastelua hieman eri näkökulmista. (Page 2010, 156.)

5.2.5 Läpimenoajan parantaminen

Kuten luvussa 5.1 mainittiin, nykyisen prosessin kartoituksessa on hyödyllistä mitata prosessin työvaiheiden ja koko prosessin kesto käytännöllisellä tasolla. Läpimenoajan parantaminen on yksi prosessin kehittämisen tärkeimmistä tavoitteista. Prosessin läpimenoaika on tärkeää prosessin asiakkaalle, koska he havaitsevat sen konkreettisesti, kuinka kauan he joutuvat odottamaan prosessin tuloksia. Organisaation näkökulmasta prosessin kehittäminen on tärkeää, jotta resursseja saadaan vapautettua nopeammin muihin toimintoihin. (Page 2010, 157.)

Jokainen prosessin kehittämisen edeltävä vaihe, on omalta osaltaan parantanut prosessin läpimenoaikaa. Tässä vaiheessa tulee tarkastella prosessin työvaiheita ja tunnistaa kauimmin kestävimmät työvaiheet. Näitä työvaiheita nopeuttamalla saavutetaan suurimpia hyötyjä koko prosessin läpimenoaikaan. (Page 2010, 157.)

Suurin syy prosessin työvaiheiden suureen kestoon ovat odotukset. Tästä syystä on tärkeää mitata työvaiheen konkreettinen kesto kalenteriajassa, sekä aika kuinka kauan prosessi odottaa jotain muuta toimintoa. Eliminoimalla odottamiset pystytään helposti nopeuttamaan koko prosessin läpimenoaikaa ilman prosessin toimintojen muuttamista. Optimoimalla läpimenoaikaa tulisi keskittyä seuraaviin seikkoihin. (Page 2010, 159.)

- Vähennä tiedon siirtoja (engl. handoff)
- Optimoi työvaiheita, jotka tuovat prosessiin lisäarvoa
- Poista työvaiheita, jotka eivät tuo prosessiin lisäarvoa
- Nopeuta kauan kestäviä työvaiheita tai yritä poistaa ne kokonaan
- Suorita eri työvaiheita samanaikaisesti
- Yhdistä useampia työvaiheita

5.2.6 Automatisointi

Viimeisimpänä prosessin kehittämisen tekniikkana on automatisointi. Automatisointi on viimeisin työvaihe, siksi koska prosessi on saatu edellisillä työvaiheilla optimoi-

tua niin tehokkaaksi, kuin mahdollista. Ei ole järkeä automatisoida tehotonta prosessia.

Automatisointia mietittäessä ei ole järkevää aloittaa organisaatiossa uutta valtavaa tietojärjestelmäprojektia. Automatisoinnilla pyritään automatisoimaan prosessia työvälineillä ja ohjelmistoilla, jotka organisaatiolla on jo käytössään. Automatisointi pitää myös tehdä prosessin näkökulmasta. Muuten voidaan päätyä automatisoimaan sellaista työvaihetta, joka olisi pitänyt poistaa kokonaan prosessista. (Page 2010, 161.)

6 UUSI TUOTEKEHITYSPROSESSI

6.1 SCRUM

Scrum-menetelmä on ollut käytössä kohdeyrityksessä jo lähes 10 vuotta. Kun uutta tuotekehitysprosessia lähdettiin kehittämään, ei Scrum-menetelmän käyttöä haluttu kyseenalaistaa. Tuotekehitysprosessin kehitystiimin mielestä yrityksen tuotekehityksen ongelmat eivät johtuneet itse Scrum-menetelmästä, vaan menetelmän puutteellisesta käytöstä organisaatiossa ja organisaation muista ongelmista. Tuotekehitysprosessin lähtökohtana oli vahvasti selkeyttää Scrum-menetelmän käytäntöjä, sekä tapaa miten menetelmää käytettiin kohdeyrityksessä.

Tuotekehityksen prosessin kehittäminen aloitettiin keväällä 2012. Projektin nimenä oli OpusCapita Way of Scrum, joka kuvaakin hyvin projektin tarkoitusta. Kehitysprojekti aloitettiin osana tätä opinnäytetyötä. Tarkoituksena oli määritellä se, mitä Scrum kohdeyrityksessä tarkoittaa ja miten sitä pitää toteuttaa. Määrittely tehtäisiin kuvaamalla toimintatavat käytännön tasolla, sekä mitä eri rooleissa toimivilta ihmisiltä odotetaan.

Prosessin kehitys aloitettiin pienellä neljän hengen tiimillä. Tiimiin kuului tuotekehitystiimin esimies, tuotekehityksen vetäjä, QA-tiimin esimies, sekä tuotekehittäjä ja Scrum Master. Kaikki henkilöt toimivat osana tuotekehitysprosessia hieman erilaisis-

sa rooleissa, joten katsottiin, että kyseisellä joukolla saataisiin katettua kaikki näkökulmat ja aikaiseksi paras lopputulos. Kehitysprojektin pääasiallisina vastuuhenkilöinä olivat tuotekehitystiimin esimies, sekä allekirjoittanut.

Tarkoituksena oli aluksi käydä Scrum-menetelmän prosessi läpi alusta loppuun ja peilata sitä yrityksessä aikaisemmin käytössä olevaan prosessiin. Tiedettiin, että prosessi olisi pääpiirteittäin samanlainen, koska yrityksessä oli aikaisemminkin käytössä Scrum-menetelmän mukainen tuotekehitysprosessi. Tiedettiin myös, että nykyisessä toteutuksessa oli ongelmakohtia ja näihin haluttiin tarttua heti ensimmäisen läpikäynnin yhteydessä.

Osa vanhan prosessin ongelmista oli tiedossa jo ennen kehitysprojektin alkua. Ongelmia haluttiin kuitenkin kartoittaa lisää ja tästä syystä päätettiin kohdeyrityksen tuotekehityksessä toteuttaa lähtötasokysely nykyisestä tuotekehitysprosessista. Haluttiin myös pystyä vertaamaan kehitysprosessin tuloksia vanhaan prosessiin ja toteutettava kysely toimikin jatkossa uuden prosessin vertailukohtana.

6.2 Kyselytutkimus

Kyselytutkimus tuotekehityksen nykyisestä prosessista toteutettiin toukokuussa 2012. Lomaketutkimuksen vastaanotti yhteensä 35 henkilöä, jotka koostuivat seuraavasti: 14 kehittäjää, 9 testaajaa, 5 esimiestä, 5 tuotehallinnan edustajaa, sekä 2 UX suunnittelijaa. Kyselyyn vastasi yhteensä 24 tuotekehityksen eri rooleissa toimivaa henkilöä. Kaikki vastanneet vastasivat lomakekyselyn jokaiseen kysymykseen. Vastanneiden tarkkaa profiilia ei pystytä määrittelemään, koska vastaukset annettiin anonyymisti. Kaikki lomaketutkimuksen vastaanottaneet olivat kohdeyrityksen työntekijöitä, eikä alihankkijoita ollut kyselyssä mukana.

Kyselyssä oli kolme avointa kysymystä ja yksi numeerinen kysymys. Kyselyn kysymykset olivat:

1. Tuotekehitysprosessin kouluarvosana?
2. Mistä pidät eniten nykyisessä tuotekehitysprosessissa?
3. Mistä pidät vähiten nykyisessä tuotekehitysprosessissa?

4. Mitkä muutokset parantaisivat eniten nykyistä tuotekehitysprosessia?

Kyselytutkimus koostui pääasiassa avoimista kysymyksistä eli kyselytutkimuksen tuloksia tulkitaan laadullisen analyysin avulla. Laadullisessa analyysissä aineistoa tarkastellaan usein kokonaisuutena. Kvalitatiivinen analyysi vaatii tilastollisesta tutkimuksesta poikkeavaa absoluuttisuutta, jossa kaikki selvitettävään asiaan liittyvät seikat tulee kyetä selvittämään siten, että ne eivät ole ristiriidassa esitetyn tulkinnan kanssa. (Alasuutari 1993, 21.)

Laadullinen tutkimus koostuu kahdesta vaiheesta; havaintojen pelkistämisestä ja arvoituksen ratkaisemisesta. Pelkistämisessä aineistoa tarkastellaan aina vain tietystä näkökulmasta käsin. Pelkistämällä aineisto pelkistyy hallittavammaksi määräksi ”raakahavaintoja”. Pelkistämisen toisessa vaiheessa karsitaan havaintomäärää yhdistämällä havaintoja. Tarkoituksena on löytää havaintojen takaa yhteisiä nimittäjiä, joissa havainnot ovat näytteitä samasta ilmiöstä. (Alasuutari 1993, 23.)

Laadullisessa tutkimuksessa ei aineistossa saa olla yhtään poikkeusta. Laadullinen tutkimus poikkeaa tässä tilastollisesta analyysistä. Yksikin poikkeus kumoaa säännön ja osoittaa, että asia pitää miettiä uudelleen. Havaintojen perusteella tuotetun säännön pitää päteä poikkeuksetta kaikkiin raakahavaintoihin. (Alasuutari 1993, 25.)

Laadullisen analyysin toinen päävaihe on tutkimustulosten tulkitseminen. Tulkitsemin merkitsee sitä, että tuotettujen johtolankojen ja käytettävissä olevien vihjeiden pohjalta tehdään merkitystulkinta tutkittavasta ilmiöstä. (Alasuutari 1993, 27.)

Vaikka lomaketutkimus ja laadullinen tutkimus eroavat hieman toisistaan, myös lomaketutkimuksessa voidaan erottaa kaksi laadullisen tutkimuksen vaihetta. Lomaketutkimuksessakin pyritään rajoittamaan havaintojen määrää halittavammaksi kokonaisuudeksi. Usein tämä tapahtuu jo lomakkeen kysymyksiä suunniteltaessa. Laadullisen tutkimuksen ja lomaketutkimuksen suurin ero on raakahavaintojen yhdistämisvaiheessa. Lomaketutkimuksessa havaintoja yhdistetään soveltamalla tilastollisia menetelmiä. Toisin kuin laadullisessa tutkimuksessa, myös poikkeustapaukset sallitaan. (Alasuutari 1993, 34–35.)

Lomaketutkimuksen vastauksissa esiintyvät ilmiöt ja teemat kirjattiin ylös ja niiden esiintymismäärät laskettiin. Näin saatiin tulokseksi eniten esiintymiä saaneet teemat, joita ei ollut, kuin yksi tai kaksi per kysymys.

Tuotekehityksen prosessin yleisarvosanaksi saatiin vastaajien kesken 7 (tarkalleen 7,04). Suurin osa annetuista vastauksista oli 6 ja 8 välillä, niin että vain yksi vastaaja oli antanut tuotekehitysprosessille arvosanan 5. Tämä osoittaa sen, että suurin osa vastaajista piti prosessia tyydyttävänä. Prosessia ei koettu erityisen hyvänä eikä huonona. 18 vastaajaa antoi prosessille arvosanan 7 tai 8, kun taas vain 6 vastaajaa antoi sille arvosanan 5 tai 6.

Kyselyn tulos oli positiivinen. Tunnetuista ongelmista johtuen odotettiin heikompaa tulosta. Tulos kertoo sen, että kaikki havainnoidut ongelmat, eivät kosketa kaikkia tuotekehityksen työntekijöitä. Esimerkiksi läpinäkyvyyden puute on enemmänkin tuotekehityksen sidosryhmien ongelma, eikä näy niin vahvasti tuotekehityksen sisällä.

Kyselyn tuloksien pohjalta on kuitenkin tärkeää huomioida, että prosessi toimii monelta osin hyvin ja se koetaan toimivaksi. Uuden prosessin kehityksessä tulee varmistaa, ettei hyvin toimivia asioita muuteta uudessa prosessissa huonommaksi.

Vanhan tuotekehityksen noudattaman prosessin pidetyimmistä seikoista nousi selkeästi esille kaksi asiaa. Selkeimmin esille nousi itse prosessin olemassaolo ja selkeät käytännöt eri tilanteisiin. Kolmannes vastaajista kertoi pitävänsä prosessista sen vuoksi, että siinä oli päivittäiset tapaamiset, suunnittelupalaverit ja muut säännölliset tapahtumat. Lisäksi pidettiin siitä, että prosessi tarjosi selkeät toimintamallit eri tilanteisiin.

Lähes yhtä moni vastaajista kertoi pitävänsä prosessin tarjoamasta vapaudesta hallita omia töitään. Samaan teemaan liittyi myös moni kommentti byrokratian vähyydestä. Vastauksista päätellen prosessilla oli onnistuttu löytämään hyvä tasapaino palaverien ja määriteltyjen toimintamallien, sekä vapauden välillä.

Prosessi ja sen tarjoama vapaus sisältyi yli 60 %:iin vastauksista. Näiden tulosten perusteella uuden prosessin kehityksessä on hyvin tärkeää pyrkiä säilyttämään mahdollisimman kevyt byrokratia ja vapaus hallita omia töitään mahdollisimman paljon. Vanhassa prosessissa oli myös hyvin paljon asioita, mistä ei pidetty. Niiden korjaaminen ja muuttaminen paremmiksi oli yksi uuden prosessin tärkeimmistä tavoitteista.

Kyselyyn vastanneiden mielestä suurin puute prosessissa on ominaisuusmäärittelyjen vajavaisuus tai niiden suoranainen puute. 50 % vastanneista mainitsi vanhan prosessin huonoksi puoleksi heikon ominaisuusmäärittelyn. Ominaisuuksien määrittely oli vajavaista tai se muuttui toteutuksen aikana. Varsinkin muuttuvat määrittelyt aiheuttavat prosessissa huomattavia ongelmia.

Moni vastaaja (29 %) koki myös Scrum-menetelmän toteutuksen huonoksi. Itse Scrum-menetelmää ei niinkään kritisoitu vaan sitä, miten sitä toteutetaan tuotekehityksessä. Tuotekehityksessä koettiin, että osa prosessin vaiheista tehdään vain sen vuoksi, kun Scrum-menetelmä niin määrittelee, eikä toiminnoista saada niiden todellisia hyötyjä irti. Scrum-menetelmän käytäntöjen selkiyttäminen ja käytännön toimien kuvaaminen on vastausten perusteella hyvin ajankohtaista ja tärkeää.

Muita mainittuja negatiivisia asioita olivat alihankinnan käyttö, prosessin noudattamattomuus, sekä yleisen informaation puute. Nämä asiat koettiin ongelmiksi kuitenkin huomattavasti harvemmin, kuin kaksi muuta syytä. Vain 6 vastaajaa koki yhden tai useamman näistä asioista ongelmalliseksi.

Lomaketutkimuksen viimeisimpänä kysymyksenä kysyttiin parannusehdotuksia, joista vastaaja kokisi olevan eniten hyötyä prosessin parantamisessa. Ei ole yllätys, että tärkeimmäksi parannuskohteeksi nousi vaatimusmäärittelyjen parantaminen. Vaatimusmäärittelyt koettiin suurimmaksi heikkoudeksi, joten on myös loogista, että niiden parantaminen on tärkeimpien parannuskohteiden listan kärkipäässä. Jopa 58 % vastaajista mainitsi vaatimusmäärittelyt tärkeimmäksi parannuskohteeksi.

Toisena parannuskohteena mainittiin tuotekehitystiimien välisen yhteistyön parantaminen. Vastausten perusteella tiimit toimivat erillään toisistaan eikä tiimien välillä

ole tarpeeksi yhteistyötä. Kun eri tiimit kuitenkin kehittävät yhtä yhteistä tuoteperhettä, koettiin yhteistyön kehittämisen olevan tärkeää.

Vastaajat mainitsivat myös joukon muita parannusehdotuksia, kuten prosessin joustavuuden lisääminen, resurssien keskittämisen ja selkeiden toimintaohjeiden määrittelemisen. Näitä parannusehdotuksia ei kuitenkaan ehdottanut kuin yksittäiset vastaajat.

Havaintojen kerääminen ja yhdisteleminen lomaketutkimuksen vastauksista tuotti erilaisia vastauksia mitä-kysymyksiin. Tutkimuksen tarkoitus on löytää miksi-vastaus näihin kysymyksiin. Lomaketutkimuksella kerätyn aineiston perusteella pyritään löytämään ratkaisu näihin kysymyksiin. (Alasuutari 1993, 175-176.)

Lomaketutkimuksen vastausten perusteella nykyisen prosessin selkeästi suurimmaksi ongelmaksi nousi vaatimusmäärittelyjen heikko laatu. Vaatimusmäärittely nousi esiin sekä prosessin heikoimpana osa-alueena että tärkeimpänä kehityskohteena. Vaatimusmäärittely on pääasiassa tuotehallinnan vastuulla, joten voidaan olettaa, että lomaketutkimuksen vastaajista suurin osa oli tuotekehitystiimien jäseniä eli kehittäjiä ja testaajia. Ei ole uskottavaa, että kukaan tuotehallinnasta kritisoisi omaa työtään prosessin suurimpana heikkoutena. Täyttä varmuutta asiasta ei kuitenkaan, vastaajien anonymiteetin vuoksi ole.

Miksi kehittäjät ja testaajat kritisoivat vaatimusmäärittelyjen heikkoa tasoa? Tuotekehitystiimeissä on havainnoitu muutamia syitä, jotka nousivat esiin myös lomaketutkimuksen avoimissa vastauksissa. Kun ei tiedetä tarkkaan, mitä pitäisi tehdä, ei pystytä arvioimaan työhön kuluvaa aikaa. Väärät työmääräarviot vaikuttavat iteraatioiden suunnitteluun ja johtavat usein niiden epäonnistumiseen, kun kaikkea suunniteltua työtä ei saada ajoissa valmiiksi. Työn myöhästymistä ei kuitenkaan koeta merkittävä ongelmana tuotekehitystiimien sisällä. Tuotekehittäjä tai testaaja ei useinkaan pääse vaikuttamaan versioprojektin aikatauluun tai sen laajuuteen, josta syystä projektiin sitoutuminen jää usein heikoksi.

Havaintojen perusteella merkittävämpi ongelma puutteellisissa määrittelyissä on se, että osa työstä joudutaan tekemään useaan kertaan. Ensin tehdään, jotain minkä luul-

laan olevan oikein, mutta mikä joudutaan myöhemmin muuttamaan toisenlaiseksi. Työ voidaan joutua tekemään useitakin kertoja uudelleen. Myös tämä vaikuttaa heikentävästi alkuperäisen työmääräarvion tarkkuuteen, mutta ennen kaikkea se tekee työn tekemisestä uuvuttavaa. Tuotekehittäjä joutuu tekemään saman työn useaan kertaan eikä testaaja tiedä onko tehty sitä mitä halutaan. Suurimpana ongelmana tästä seuraa erimielisyyksiä ja ristiriitatilanteita tuotekehittäjän ja testaajan välillä.

Ei olekaan yllätys, että toiseksi tärkeimmäksi kehityskohteeksi osoittautui yhteistyön parantaminen. Tiimissä on hyvä tehdä töitä, kun yhteistyö tiimin jäsenten välillä sujuu hyvin. Lisäksi yhteistyö eri tiimien ja jopa yksikköjen välillä on tärkeää. Mahdolliset ongelmat saadaan aina ratkaistua, kun ne ratkaistaan yhdessä kaikkien osapuolten kanssa. Kun yhteistyö ei suju, syntyy riitatilanteita, jotka vähentävät eri tahojen välistä luottamusta. Henkilökohtaiset ristiriitatilanteet vaikuttavat eniten yksittäisen työntekijän arkeen ja uskon, että tästä syystä yhteistyön kehittäminen ja sitä heikentävien asioiden parantaminen ovat vanhan prosessin tärkeimpiä kehityskohteita.

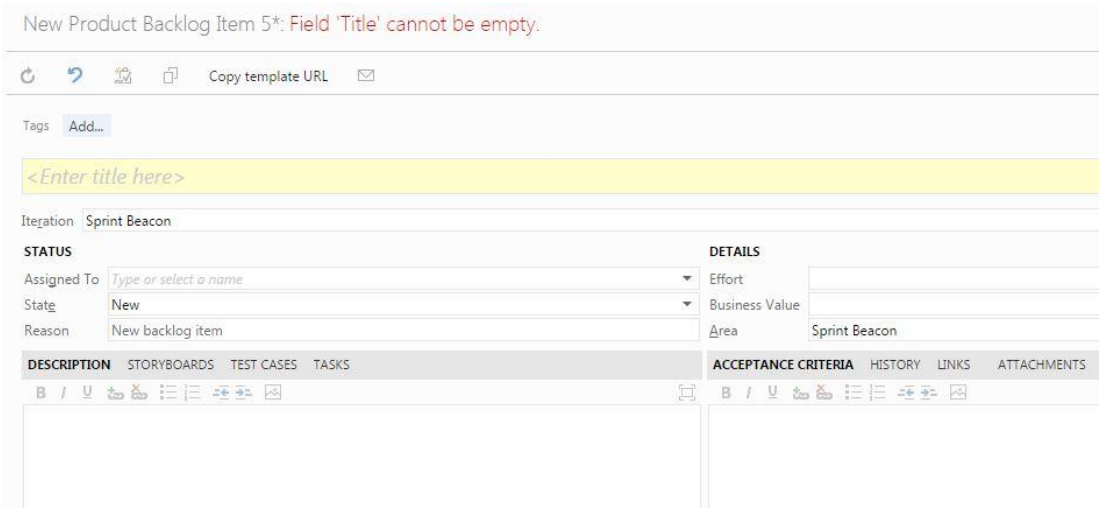
Vaatimusmäärittelyjen laatu ja hyvä yhteistyö eivät suoraan ole määriteltävän prosessin vaikutuspiirissä. Lomaketutkimuksen tulosten perusteella on kuitenkin selvää, että jo prosessin määrittelyssä tulee ottaa käyttöön kaikki mahdolliset apukeinot, jotta tilanne saadaan paremmaksi. Prosessin avulla voidaan määritellä tiettyjä toimintoja ja työvaiheita, joilla pyritään varmistumaan siitä, että mahdolliset ongelmat saadaan ratkaistua ennen kuin niistä tulee todellisia ongelmia. Näin prosessin avulla voidaan omalla osaltaan vaikuttaa ongelmien ratkeamiseen.

6.3 Uuden tuotekehitysprosessin kehittäminen

Prosessin kehittäminen aloitettiin kuvaamalla prosessin nykytila. Prosessi on myös kuvattu yleisemmällä tasolla luvussa 3.6. Tuotekehitys tuottaa kahdentyyppisiä asioita. Prosessin läpi kulkee uusia ominaisuuksia (Product Backlog Item, PBI tai Feature) ja ohjelmistovirheiden korjauksia (bug fix). Tuotekehitysprosessia ei käytetä muunlaisen työn tekemiseen. Ideaalitilanteessa molemmat työt kulkevat prosessin läpi täsmälleen samalla tavalla ja samojen vaiheiden kautta. Ohjelmistovirheet halu-

taan kuitenkin usein korjattavaksi mahdollisimman nopeasti ja tästä syystä niiden prosessi eroaa käytännössä siitä miten uusia ominaisuuksia kehitetään.

Uusi ominaisuus ja ohjelmistovirhe ovat tässä merkityksessä TFS-järjestelmässä sijaitsevia tietotyyppejä (Work Item Type). Molemmilla on järjestelmässä tieto sen tilasta, kenelle työ on määritetty, sekä tehtävän kuvaus ja muita tehtävän työn kannalta olennaisia tietoja. Työn siirtyminen prosessin vaiheesta toiseen määritellään sen iteraation (sprint), henkilön (assigned to), sekä tilan (state) avulla. Konkreettinen tehtävään liittyvä työ on useimmiten TFS versiohallintaan tuotettava ohjelmakoodi. Tieto tehdyistä ohjelmakoodin muutoksista tallentuu automaattisesti TFS-kortille (kuva 2).

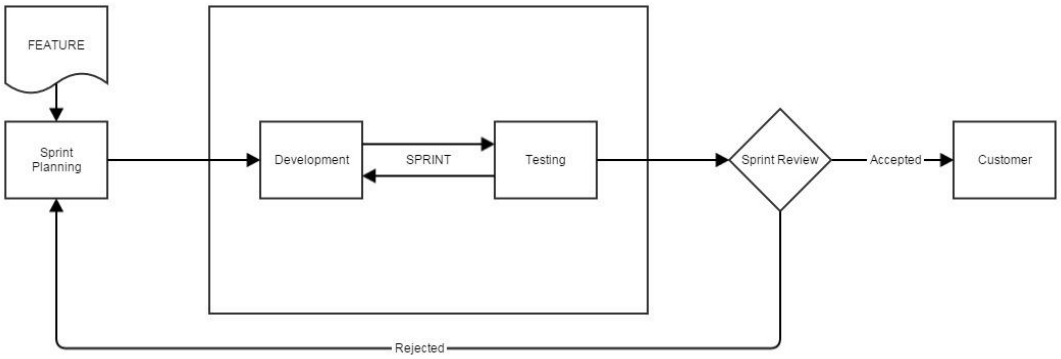


Kuva 2. Esimerkki TFS-kortista

Prosessi alkaa iteraation suunnittelupalaverista ja päättyy sen katselmukseen (kuvio 14). Suunnittelupalaverissa tehtävä työ esitellään tiimille ja katselmuksessa tehty työ katselmoidaan ja joko hyväksytään tai hylätään. Tämän lisäksi retrospektiivin katsotaan olevan tärkeä osa tuotekehitysprosessia, mutta koska se ei suoranaisesta vaikuta prosessin tuotoksiin, sitä ei ole merkattu osaksi prosessia.

Retrospektiivin avulla mahdollistetaan kehitystiimin itsereflektio ja prosessin jatkuva kehittäminen. Jatkuva kehitys on olennainen osa ketteriä menetelmiä. Uuden tuotekehitysprosessin kehitystyössä haluttiin määritellä myös prosessin ulkoisia toiminto-

ja, joiden tuotokset vaikuttavat prosessin kulkuun ja jotka ovat ketterän ohjelmistokehityksen olennaisia osia.



Kuvio 14. Tuotekehitysprosessi uuden ominaisuuden (feature) näkökulmasta

Uuden prosessin kehitystyötä tehtiin samalla, kun vanhaa prosessia mallinnettiin. Jokainen prosessin vaihe käytiin järjestyksessä läpi ja halutut muutokset määriteltiin tiedossa oleviin ongelmakohtiin. Kunkin toiminnon kehittämisessä pyrittiin käyttämään eri prosessin kehittämisen menetelmiä, jotka on kuvattu luvun 5.2 aliluvuissa. Kullekin toiminnolle määriteltiin tarvittavat syötteet, sekä vaaditut tuotokset. Tämän lisäksi määriteltiin kaikki toiminnossa mukana olevat henkilöt rooleittain. Näin jokainen pystyi tarkastelemaan määriteltyä prosessia myös rooliperusteisesti.

Uuden prosessin käytäntöjen kehityksessä tuotettiin myös paljon erityyppistä prosessin tukimateriaalia, kuten vinkkejä ja ratkaisuja yleisimpiin ongelmatilanteisiin. Poikkeustilanteet ovat yleisimpiä kohtia, missä prosessia lähdetään rikkomaan ja näille tilanteille haluttiin määritellä selkeät toimintaohjeet, jotta kaikki toimisivat samalla tavalla myös mahdollisissa poikkeustilanteissa.

6.3.1 Julkaisun suunnittelupalaveri (Release Planning)

Julkaisun suunnittelupalaveri ei ole Scrum-menetelmän virallinen osa. Ensimmäinen askel vaatimusmäärittelyjen parantamisessa on suunnitella koko julkaistava versio paremmin. Uuden prosessin yhteydessä haluttiin ottaa käyttöön myös julkaisun suunnittelupalaverin, jonka tarkoituksena oli saada kirjattua ylös koko julkaisun sisältämät ominaisuudet, sekä suurimmat riskit. Julkaisun suunnittelulla saadaan mää-

riteltyä versioprojektille jonkinlainen aikataulu, sekä budjetti. Tärkeimpänä tavoitteena oli saada tuotehallinta muodostamaan haluamistaan ominaisuuksista kokonaisuus, joka tyydyttää asiakkaita ja mahdollistaa parhaimman mahdollisen tuotteen kehittämisen aikataulun ja resurssien puitteissa.

Suunnitelman ei ole tarkoitus olla lopullinen. Ketterien menetelmien mukaisesti tuotehallinta voi muuttaa suunnitelmaa jokaisen sprintin jälkeen ja ohjata tuotteen kehitystä toivottuun suuntaan. Suunnitelman muuttamisen mahdollisuus on erittäin tärkeää ja jopa suositeltavaa. Luvussa 4.3.2 on käsitelty muutoksien lähteitä, sekä sitä miten ketterät menetelmät suhtautuvat näihin muutoksiin.

Määrittelimme julkaisun suunnittelupalaverin syötteiksi:

1. Tuotteen visio
2. Järjestetty tuotteen kehitysjono

Tuotteen visiolla tarkoitetaan käsitystä tuotteen tavoitteesta ja mitä tuotteella halutaan tavoittaa. Kun tuotteen visio on selkeänä kaikkien tiedossa, pystytään tuotteen kehitystä ohjaamaan haluttuun suuntaan. Tehtävien päätöksien tulisi myös tukea tuotteen vision määrittelemää tavoitetta.

Toiseksi syötteeksi määriteltiin tuotteen kehitysjono. Kehitysjonon tulisi pitää sisällään halutut ominaisuudet ja niiden tulisi olla tärkeysjärjestyksessä. Tuotteen kehitysjonon sisältämien ominaisuuksien tulisi pitää myös sisällään karkeat työmääräarviot, jotta voidaan alustavasti ottaa kantaa julkaisun aikatauluun.

Julkaisun suunnittelupalaverin tulosteina tulisi syntyä:

1. Sitoutuminen julkaisuun
2. Selkeä julkaisun tavoite
3. Tuotteen priorisoitu kehitysjono
4. Suunniteltu aikataulu, sekä kustannusarvio
5. Suurimmat riskit, sekä tekniset tuntemattomuudet
6. Mahdollisten riippuvuuksien määritteleminen
7. Valmiin määritelmä (Definition of Ready)
8. Tehdyn määritelmä (Definition of Done)

9. Mahdolliset kehitysehdotukset seuraavan julkaisun suunnittelupalaveriin

Hyvin suunniteltu julkaisu nopeuttaa ja helpottaa sprinttien suunnittelua ja oikeiden päätösten tekemistä muutoksiin reagoitaessa. Julkaisuprojektin ensimmäinen sprintti voidaan aloittaa julkaisun suunnittelupalaverin jälkeen.

6.3.2 Suunnittelupalaveri (Sprint Planning)

Prosessi alkaa siitä, kun tuotepäällikkö esittelee uuden ominaisuuden iteraation suunnittelupalaverissa. Suunnittelupalaveri aloittaa uuden iteraation (sprint), jossa kyseinen ominaisuus on tarkoitus kehittää ja testata. Suunnittelupalaveri on kuvattu tarkemmin luvussa 4.4.1. Tuotekehittäjä aloittaa uuden ominaisuuden kehittämisen välittömästi suunnittelupalaverin jälkeen.

Suunnittelupalaverin syötteiksi määriteltiin:

1. Tuotteen priorisoitu kehitysjono
2. Scrum-tiimin tiedossa olevat poissaolot ja
3. Edellisen sprinttikatselmuksen ja retrospektiivin tulokset

Tuotteen priorisoitu kehitysjono on syötteistä tärkein. Tuotteen kehitysjono on tuotepäällikön vastuulla. Kuten luvussa 4.4.1 kuvattiin, sprintin suunnittelupalaverissa tuotepäällikkö esittelee tärkeimmät ominaisuudet, jotka hän haluaa seuraavaksi tuotteeseen kehitettävän. Kun tuotteen kehitysjono on valmiiksi priorisoitu, ei ominaisuuksien priorisointia tarvitse alkaa miettimään suunnittelupalaverissa.

Tuotteen kehitysjonon kärkipään ominaisuudet tulisi myös olla suunniteltu tarpeeksi yksityiskohtaiselle tasolle, jotta Scrum-tiimin on nopeampi arvioida, kuinka paljon he pystyvät tekemään seuraavan sprintin aikana. Scrum-tiimin on myös tarkoitus aloittaa ominaisuuksien kehitys välittömästi suunnittelupalaverin jälkeen, joka ei ole mahdollista, jos jokin kehitettävistä ominaisuuksista on vielä liian epäselvä.

Scrum-tiimin tiedossa olevat poissaolot vaikuttavat Scrum-tiimin kapasiteettiin seuraavan sprintin aikana. Suunnittelemattomille poissaoloille, kuten sairaslomille, ei

luonnollisesti voida mitään, mutta tiedossa olevat poissaolot auttavat tiimiä arvioimaan oman kapasiteettinsa ja varmistamaan, etteivät he ota sprintille liikaa työtä.

Edellisen sprinttikatselmuksen ja retrospektiivin tulokset vaikuttavat mahdollisesti tulevan sprintin sisältöön. Sprinttikatselmuksessa on voitu hylätä jokin ominaisuus, jolloin kyseinen ominaisuus tulee hyvin mahdollisesti jatkokehitettäväksi seuraavalla sprintillä. Myös retrospektiivissä on voitu sopia uusista toimintatavoista, jotka on hyvä ottaa huomioon sprintin suunnittelun aikana.

Suunnittelupalaverin tuotoksiksi määriteltiin:

1. Sprintin tavoite/teema
2. Lista Scrum-tiimin jäsenistä
3. Sprintin tehtävälista
4. Sprinttikatselmuksen aika ja paikka
5. Päiväpalaverin aika ja paikka
6. Suunnitteludokumentti tiedoksi sidosryhmille

Sprintin tavoite on tärkeä tilanteissa, missä tiimin jäsenellä ei välttämättä ole mitään tehtävää sprintin tehtävälistalta. Kun sprintin tavoite on kaikkien tiedossa, voivat tiimin jäsenet omatoimisesti tehdä töitä tavoitteen täyttämiseksi. Tällaista tilannetta ei käytännössä esiinny kovin usein, vaan sprintin tehtävälistalta löytyy usein jokaiselle tehtävää koko sprintin ajaksi.

Sprintin tehtävälista on suunnittelupalaverin tärkein tulos. Scrum-tiimi sitoutuu parhaansa mukaan toteuttamaan kaikki sprintin tehtävälistan ominaisuudet. Sprintin tehtävälista voi sisältää myös tiedossa olevien ohjelmistovirheiden korjauksia. Suurin osa suunnittelupalaverin ajasta käytetään sprintin tehtävälistan muodostamiseen, kun taas muut tuotokset ovat lähinnä tiedoksiantoja Scrum-tiimille ja muille sidosryhmille.

Suurimpina ongelmina vanhassa tuotekehitysprosessissa havaittiin jo syötteenä vaadittu tuotteen priorisoitu tehtävälista. Listaa ei useinkaan ole ollut olemassa tai se on ollut hyvin yleisellä tasolla. Tämän seurauksena suunnittelupalaverissa jouduttiin suunnittelemaan kehitettäviä ominaisuuksia. Epäselvien ominaisuuksien työmäärää

ei ole myöskään ollut mahdollista arvioida. Tämä on johtanut palaverin venymiseen ja tehnyt usein sprintin suunnittelusta hyvin hankalaa. Epärealistisesti suunniteltu sprintti on taas johtanut sprintin epäonnistumiseen, töiden kasaantumiseen ja kiireeseen.

Tuotteen heikkolaatuinen kehitysjono oli yksi tärkeimmistä asioista, jota halusimme parantaa uudella prosessilla. Tilannetta helpottaaksemme, otimme käyttöön kehitysjonon työstöpalaverit ja valmiin ominaisuuden määritelmän (Definition of Ready).

6.3.3 Työstöpalaveri (Backlog Grooming)

Työstöpalaveri ei ole Scrumin virallinen osa, mutta se on yleisesti käytössä oleva käytäntö, jonka aikana kehitysjonoa työstetään yhdessä tuotepäällikön kanssa. Tarkoituksena on valmistella kehitysjono seuraavaa suunnittelupalaveria varten. Tämä pitää sisällään ominaisuuksien priorisointia, tarkennusta ja työmäärien arviointia.

Koska tuotepäällikkö vastaa tuotteen kehitysjonosta, hän vastaa myös työstöpalaverien järjestämisestä. Tarkoituksena on työstää tuotteen kehitysjonon tärkeimmät ominaisuudet niin, että ne täyttävät valmiin ominaisuuden määritelmän (Definition of Ready, DoR). Valmiin ominaisuuden määritelmä on myös osa uutta tuotekehitysprosessia. Kuten työstöpalaveri, myöskään DoR ei ole Scrumin virallinen osa, mutta se haluttiin ottaa käyttöön, jotta varmistutaan siitä, että tuotteen kehitysjonon ominaisuudet ovat vaaditulla tasolla ennen sprintin suunnittelupalaveria.

Definition of Ready määrittelee mitä tietoja ominaisuuden tulee pitää sisällään, jotta se voidaan esitellä sprintin suunnittelupalaverissa. Näitä tietoja ei lähdetty määrittelemään osana prosessin kehitystyötä, vaan jokaiselle Scrum-tiimille haluttiin antaa vapaus luoda oma valmiin ominaisuuden määritelmä. Määritelmä pitää esimerkiksi sisällään tiedon vaatimusmäärittelyn tasosta, testitapausten määrästä ja että ominaisuudella tulee olla kehitystiimin antama työaika-arvio.

Valmiin ominaisuuden määritelmään lisättiin myös tieto tyytyväisyyden vaatimuksista (Conditions of Satisfaction, CoS). Scrum-tiimin tulee määritellä jokaiselle ominai-

suudelle ne vaatimukset, joiden perusteella voidaan sanoa ominaisuuden kehityksen olleen onnistunut. Monet vaatimuksista (esim. laatu) ovat samoja jokaiselle kehitettävälle ominaisuudelle, mutta jokaisen ominaisuuden kehittämisessä tulee ottaa huomioon myös muita vaatimuksia kuten budjetti, aikataulu ja suorituskyky. Monet näistä vaatimuksista ovat ns. ei-toiminnallisia vaatimuksia.

Kehitysjonon työstöpalavereilla ja ominaisuuden valmiusmääritelmällä pyrittiin prosessin osalta varmistamaan, ettei sprintin suunnittelupalaverissa esitellä ominaisuuksia, jotka eivät ole vielä valmiita kehitettäväksi. Valmiit ominaisuudet ovat selkeitä koko Scrum-tiimille ja niiden kehitys voi alkaa välittömästi suunnittelupalaverin jälkeen, jolloin seuraavan sprintin työt alkavat.

6.3.4 Sprint

Sprintin aikana kehitystiimi tekee sprintin tehtävälistalla lueteltuja töitä. Prosessin näkökulmasta ensin tuotekehittäjä tekee tarvittavat ohjelmistomuutokset, jotka testaaja tämän jälkeen testaa varmistaen, että tehty ominaisuus on toteutettu sen vaatimusmäärittelyn mukaisesti. Kun tuotekehittäjä saa yhden tehtävän valmiiksi, hän alkaa tehdä seuraavaa. Näin jatketaan kunnes sprintti päättyy.

Uuden tuotekehitysprosessin yhteydessä sprintin kestoksi määriteltiin 4 viikkoa. Koettiin, että tällöin Scrumin määräämät palaverit ja muut aktiviteetit eivät vie liikaa aikaa ja prosessin byrokratia on oikeassa suhteessa sprintin kestoon. 4 viikkoa antaa myös kehitystiimille tarpeeksi aikaa saada laajempiakin ominaisuuksia valmiiksi eikä ominaisuuksia jouduta pilkkomaan liian pieniksi.

Sprintin aikana kehitystiimillä on täysi rauha tehdä sprintin tehtävälistan tehtäviä. Scrum Master pitää huolen, että tiimin jäseniä ei häiritä muilla tehtävillä. Tiimillä ei ole sprintin aikana päiväpalaverin lisäksi muita palavereja. Scrum Masterin tehtävä on poistaa kaikki mahdolliset esteet tiimin työn etenemiseltä, joka voi pitää sisällään myös palaverien järjestämisen tuotepäällikön kanssa tai esimerkiksi IT-ongelmien ratkaisemista.

Ainoan poikkeuksen edellä mainittuun tekee päiväpalaveri, joka pidetään joka päivä sovittuna ajankohtana. Palaveriin osallistuvat kaikki kehitystiimin jäsenet. Palaverin kesto on rajattu 15 minuuttiin ja kaikki seisovat palaverin aikana. Seisomisella pyritään estämään palaverin tahaton venyminen. Palaverin tarkoitus on auttaa tiimiä synkronoimaan oma toimintansa. Palaverissa jokainen tiimin jäsen kertoo mitä on tehnyt edellisen palaverin jälkeen ja erityisesti mitä aikoo tehdä ennen seuraavaa palaveria. On myös tärkeää kertoa mahdollisista esteistä (engl. Impediment), jotka estävät tai hidastavat oman työn etenemistä. Scrum Masterin tehtävä on poistaa nämä esteet mahdollisimman nopeasti. Palaverin avulla jokainen tiimin jäsen tietää mitä muut tekevät ja pystyy suunnittelemaan oman työnsä yhdessä muiden kanssa. Tämä on varsin hyödyllistä useassa eri toimipisteessä työskentelevissä tiimeissä.

Uuden prosessin kehitystyön yhteydessä määriteltiin, että jokaisen tiimin tulee pitää päiväpalaverit eikä niistä voi olla pois kukaan ilman pätevää syytä. Tässä vaiheessa ei nähty syytä tehdä muita muutoksia sprintin käytäntöihin.

Kuten kaikki muutkin tapahtumat Scrum-menetelmässä, myös sprintti on aikarajattu. Sprintti päättyy aina sille varatun ajan jälkeen, vaikka sille suunnitellut työt olisivatkin vielä kesken. Valmistuneille ominaisuuksille otettiin käyttöön tehdyn määritelmä (Definition of Done).

Jotta ominaisuus tai jonkin sen osan voitiin sanoa olevan täysin toteutettu, tuli toteutuksen täyttää kaikki tehdyn määritelmän kohdat. Määritelmä on tiimikohtainen ja sen tarkoitus on saada aikaiseksi yhteinen ymmärrys siitä, mitä täysin toteutettu pitää sisällään. Ideaalitilanteessa määritelmä pitää sisällään kaikki organisaation toimenpiteet ominaisuuden ohjelmoinnista siihen, kun se voidaan toimittaa asiakkaalle. Jokaisen tiimin tulisi pyrkiä kehittämään omaa määritelmäänsä lisäämällä siihen jatkuvasti vaadittuja asioita. Määritelmä asetetaan aina sprintin suunnittelupalaverissa.

6.3.5 Sprinttikatselmus (Sprint Review)

Iteraation päätyttyä pidetään Sprint Review -tapahtuma, missä kehitystiimi esittelee uuden ominaisuuden tuotepäällikölle ja muille sidosryhmille. Tapahtuman tarkoituk-

sena on selvittää, mitä sprintin aikana kehitettiin, jotta voidaan tehdä oikea päätös siitä, mitä kehittää seuraavaksi.

Määrittelimme sprinttikatselmuksen syötteiksi:

1. Sprintin tavoitteen/teeman
2. Sprintin tehtävälistan

Sprinttikatselmuksen tuloksina tulisi olla:

1. Eri sidosryhmien palaute toteutetuista ominaisuuksista
2. Toteutettujen ominaisuuksien hyväksyntä tai hylkäys
3. Sprinttikatselmuksen dokumentti

Tuotepäällikkö esittelee sprinttikatselmuksen aluksi edellisen sprintin tavoitteen, sekä tehtävälistan ja esiteltävät ominaisuudet. Tämän jälkeen kehitystiimi esittelee valmiiksi saadut ominaisuudet, jonka jälkeen paikallaolijat voivat keskustella ominaisuuden toteutuksesta. Tuotepäällikkö voi tehdä sidosryhmien palautteen pohjalta uusia kehitysideoita, jotka hän voi priorisoida tuotteen kehitysjonossa suhteessa muihin kehitettäviin asioihin.

Lopuksi tuotepäällikkö hyväksyy tai hylkää esitellyn ominaisuuden toteutuksen, jonka perusteella ominaisuus merkataan valmiiksi tai otetaan jatkokehitettäväksi seuraavalla sprintillä. Tuotepäällikkö voi myös päättää, että kyseinen ominaisuus tiputetaan tuotteesta kokonaan pois. Jos ominaisuus vaatii jatkokehitystä, se esitellään uudelleen seuraavan sprintin suunnittelupalaverissa, missä tuotepäällikkö esittelee haluamansa muutokset. Jos ominaisuus hyväksytään, se merkataan kokonaan valmiiksi. Ominaisuuden näkökulmasta prosessi päättyy siihen, kun tuotepäällikkö hyväksyy ominaisuuden toteutuksen. Koska tuotekehitystyötä tehdään kuitenkin asiakkaan vuoksi, haluttiin asiakas saada mukaan myös prosessikuvaukseen. Asiakas saa uuden ominaisuuden käyttöönsä tulevien versiopäivitysten yhteydessä. Monessa tapauksessa ominaisuuden hyväksymisestä on voinut kulua useita kuukausia siihen, kun asiakas pääsee lopulta käyttämään kyseistä ominaisuutta.

Sprinttikatselmukseen on usein myös kutsuttu muita yrityksen sisäisiä sidosryhmiä, kuten muiden tuotteiden kehitystiimin jäseniä. Tilaisuus on kaikille avoin mahdolli-

suus nähdä käytännössä mitä kyseisen tuotteen tuotekehitystiimi on saanut edellisen sprintin aikana valmiiksi. Näin kyseinen palaveri toimii myös tiedonjakelun kanavana muulle organisaatiolle.

Uuden tuotekehitysprosessin yhteydessä määriteltiin, että kaikkien tuotekehitystiimien sprinttikatselmukset pidettäisiin samana päivänä. Useimmissa tapauksissa tämä olisi sprintin viimeisen viikon keskiviikko. Tämä helpottaisi sidosryhmien osallistumista kyseisiin tapahtumiin. Suurempana tarkoituksena oli synkronoida kaikkien tuotekehitystiimien sprintit samaan sykliin, jolloin myös suunnittelupalaverit olisivat aina samana päivänä.

Määriteltiin myös, että sprinttikatselmuksessa tulee käyttää tuotteen julkaisuversiota (Release Version), eikä tuotekehittäjän omaa tietokonetta saa käyttää. Tämä pakottaa tiimin todella viimeistelemään esittelemänsä ominaisuudet ja hyväksytyt ominaisuudet voidaan todella merkitä kokonaan tehdyiksi.

Scrum Master muodostaa katselmuksesta dokumentin, mistä nähdään kuka palaverissa oli paikalla, mitä ominaisuuksia esiteltiin ja mitkä niistä hyväksyttiin tai hylättiin. Myös esille tulleet kehitysideat on hyvä kirjata ylös kyseiseen dokumenttiin. Dokumentti jaetaan tämän jälkeen koko yritykselle.

Katselmuksen jälkeen tuotepäällikkö päättää, jatketaanko tuotteen kehitystä vai lopetetaanko koko tuotteen kehitys. Kohdeyrityksen tuotekehityksessä kehitetään uusia versioita olemassa olevasta tuotteesta, eikä käytännössä ole muita vaihtoehtoja, kuin jatkaa tuotteen kehitystä. Kun tuotteen kehitystä päätetään jatkaa, koko kehitystiimin tulee miettiä mahdollisia keinoja miten he pystyvät parantamaan toimintaansa. Tätä varten järjestetään sprintin retrospektiivi.

6.3.6 Retrospektiivi (Sprint Retrospective)

Uuden prosessin kehityksen yhteydessä määriteltiin, että jokaisen Scrum-tiimin tulee pitää sprintin päätyttyä retrospektiivi. Retrospektiivi tulisi pitää sprinttikatselmuksen jälkeen, mutta ennen seuraavan sprintin suunnittelupalaveria. Retrospektiivi suositel-

tiin pidettäväksi sprintin viimeisen viikon torstaina. Koska retrospektiiviin osallistuu vain kyseinen Scrum-tiimi, ei jokaisen tiimin retrospektiivin tarvitse olla samana päivänä.

Retrospektiivin syötteiksi määriteltiin:

1. Sprintin tehtävälista
2. Sprinttikatselmuksen dokumentti
3. Edellinen retrospektiividokumentti

Sprintin tehtävälista sekä sprinttikatselmuksen tulokset auttavat tiimiä keskustelemaan siitä mitä he saivat valmiiksi sprintin aikana. Näin tiimi voi paremmin reflektoida ja kehittää omaa toimintaansa. Edellisen retrospektiividokumentin perusteella tiimi voi palauttaa mieleen, mitä edellisellä kerralla sovittiin. Näin tiimi voi pohtia onko sovittuja toimenpiteitä tehty ja tulisiko niitä mahdollisesti muuttaa.

Retrospektiivin tulosteiksi määriteltiin vastaukset seuraaviin kysymyksiin:

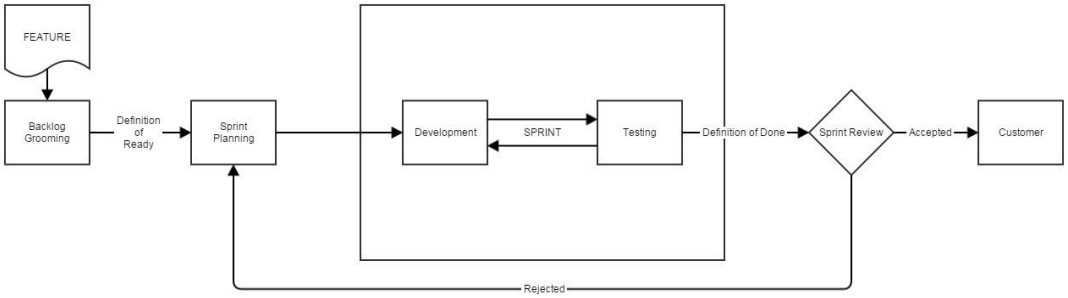
1. Mitä asioita tiimin tulisi aloittaa?
2. Mitä tiimin tulisi lopettaa?
3. Mitä tiimin tulisi jatkaa?

Vastaamalla näihin kysymyksiin pidetään keskustelu niissä asioissa mihin tiimi pystyy itse suoranaisesti vaikuttamaan, sekä saadaan tulokseksi konkreettisia toimenpiteitä, minkä perusteella tiimi pyrkii kehittämään toimintaansa. Scrum Master kirjaa nämä tulokset retrospektiividokumenttiin, joka jaetaan muulle organisaatiolle. Jos tiimin toiminnan kehittämiselle on tiimin ulkoisia esteitä tai hidasteita, tulisi Scrum Masterin pyrkiä poistamaan nämä esteet mahdollisimman nopeasti.

6.3.7 Uusi tuotekehitysprosessi

Kehitetty uusi tuotekehitysprosessi on kuvattuna kuviossa 15. Pääpiirteittäin prosessia ei muutettu paljon. Täysin uusina asioina otettiin käyttöön työstöpalaverit, sekä käsitteet Definition of Ready ja Definition of Done. Näiden tarkoituksena oli paran-

taa puutteellisia vaatimusmäärittelyjä. Lomaketutkimus paljasti vanhassa prosessissa myös muita puutteita, mutta päätettiin keskittyä vain yhteen kerrallaan.



Kuvio 15. Uusi tuotekehitysprosessi

Uusien ominaisuuksien kehittäminen ja kulkeminen prosessin läpi oli nyt selkeää. Prosessin eri vaiheille oli kehitetty selkeät säännöt ja mitä keneltäkin vaadittiin. Aikaisempi prosessi oli kuitenkin osoittautunut haastavaksi ohjelmistovirheiden korjauksien osalta ja ohjelmistovirheiden käsittely päätettiin määritellä tarkemmalla tasolla.

6.3.8 Ohjelmistovirheiden käsittely

Ohjelmistovirheiden korjauksessa noudatettiin samaa tuotekehityksen prosessia, kuin uusien ominaisuuksien kehityksessä. Kehitettävien ominaisuuksien testauksessa löytyneet ohjelmistovirheet tuli korjata saman sprintin aikana, jolloin sprintin päätyttyä ominaisuudet eivät sisältäneet enää tunnettuja virheitä. Tarkoitus oli tuottaa jokaisen sprintin aikana virheettömiä ja kokonaisia ominaisuuksia.

Ohjelmistovirheitä tuli tuotekehityksen tietoon useasta eri paikasta, jotka voidaan jakaa kahteen eri kategoriaan ulkoisiin ja sisäisiin. Ulkoisia lähteitä ovat asiakkaat, jotka raportoivat kohtaamistaan ongelmista. Sisäisiä lähteitä ovat pääasiassa tuotekehityksen omat testaajat.

Ohjelmistovirheen alkuperää tärkeämpää on tietää, missä tuotteen versiossa ohjelmistovirhe on olemassa ja mihin versioihin ohjelmistovirhe halutaan korjattavaksi. Asiakkaan raportoimat virheet korjataan sisäisiä useammin aikaisempiin versioihin

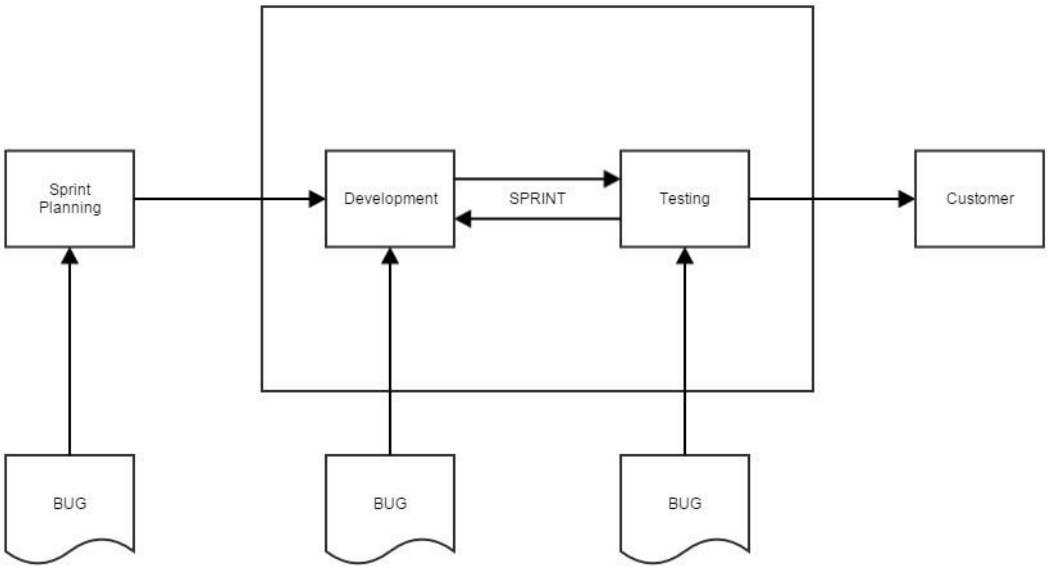
(Maintenance Version). Kun virhe on raportoitu, se kirjataan ylös kyseisen tuotteen kehitysjonoon.

Tuotepäällikkö esittelee kehitysjonosta korjattavaksi haluamansa ohjelmistovirheet sprintin suunnittelupalaverissa. Ohjelmistovirheitä korjataan ja niiden korjaus testataan sprintin aikana. Ainoana erona uusien ominaisuuksien kehitykseen on, ettei ohjelmistovirheiden korjauksia esitellä sprinttikatselmuksessa, vaan niiden katsotaan olevan valmiita, kun testaaja on hyväksynyt niiden korjauksen. Sovittiin kuitenkin, että korjatut ohjelmistovirheet mainitaan sprinttikatselmuksesta muodostettavassa dokumentissa.

Ohjelmistovirheiden korjaus ei useinkaan voi odottaa seuraavan sprintin suunnittelupalaveria. Asiakas voi odottaa pikaista korjausta tai ohjelmistovirhe voi estää muiden töiden jatkumisen, jonka vuoksi se on korjattava mahdollisimman nopeasti. Varsinkin kriittisemmät ohjelmistovirheet ovat hyvin aikariippuvaisia.

Ohjelmistovirheiden korjauksen aikariippuvuudesta johtuen, niiden prosessi on käytännössä ajautunut sekavammaksi (kuvio 16). Osa virheistä tulee työn alle kesken sprintin, joko suoraan kehittäjälle tai testaajan kautta. Kun tulevaa korjausta ei ole kyetty ottamaan huomioon sprintin suunnitelmissa, osa sprintin suunnitelluista töistä jää tekemättä.

Suunnittelemattomien töiden vaikutusta on pyritty minimoimaan sillä, että tiimin jäsenille varataan sprintin suunnittelupalaverissa tietty määrä ylimääräistä aikaa. sprintin työmäärä pyritään siis mitoittamaan todellista kapasiteettia pienemmäksi, jotta tiimille jää aikaa tehdä myös suunnittelemattomia töitä. Suunniteltu kapasiteetti nojaa vahvasti kehitysjonon ominaisuuksien työmääräarvioihin. Ohjelmistokehityksen luonteesta johtuen työmääräarviot voivat olla hyvinkin vääriä, josta syystä myös kapasiteetti on ylimitoitettu.



Kuvio 16. Vanha tuotekehitysprosessi ohjelmistovirheen (bug) näkökulmasta

Uuden prosessikehityksen yhteydessä sovittiin, että kaikki uudet ohjelmistovirheet osoitetaan kyseisen tiimin Scrum Masterille. Scrum Master pystyy tarkastamaan raportoidun virheen ja että raportti sisältää kaiken tarvittavan tiedon. Lisäksi Scrum Master voi korjauksen kiireellisyydestä riippuen ohjata ohjelmistovirheen tuotteen kehitysjonoon tai sopivalle kehittäjälle niin, että virheen korjaus häiritsee muuta tiimin työtä mahdollisimman vähän.

Sovittiin myös, että sprintin jäsenille muodostetaan yksilöllinen kerroin, jonka avulla voidaan paremmin allokoida kyseisen henkilön työaika suunniteltujen ja suunnittelemattomien töiden välillä. Toteutuneita työaikoja seurattaisiin ja käytettävä kerroin tarkennettaisiin toteutuneiden työmäärien mukaisesti. Näillä toimenpiteillä pyrittiin pienentämään ohjelmistovirheiden korjaamisesta aiheutuvia ongelmia, kunnes ohjelmistovirheet vähenisivät niin paljon, ettei niiden suunnittelemattomaan korjaamiseen tarvitsisi enää varata erikseen aikaa.

7 UUDEN PROSESSIN KÄYTTÖÖNOTTO

7.1 Uuden tuotekehitysprosessin käyttöönotto

Uusi tuotekehitysprosessi otettiin käyttöön kohdeyrityksen tuotekehitystiimeissä portaittain. Ensimmäinen tiimi otti prosessin käyttöön syksyllä 2012 ja sen käyttöä laajennettiin asteittain, niin että kaikki tuotekehitystiimit käyttivät uutta prosessia vuoden 2013 kevään aikana.

Prosessin jalkauttamisen toteuttivat tiimien Scrum Masterit. Prosessi ja siihen liittyvät käytännöt olivat kuvattuina yrityksen sisäisessä wikissä, jossa sen sisältö oli tarkasteltavissa sekä tapahtumien, että eri roolien kautta. Näin jokainen pystyi tarkastelemaan esimerkiksi suunnittelupalaveria kokonaisuutena tai pelkästään oman roolinsa kautta. Tämän uskottiin helpottavan kirjoitetun määritelmän käyttöä.

Prosessin kehittämistä jatkettiin koko käyttöönoton ajan. Tiedettiin, ettei kaikkia mahdollisia ongelmia oltu ratkaistu eikä oltu varmoja siitä, että kaikki esitetyt ratkaisut toimisivat todellisissa tilanteissa. Ketterien menetelmien teeman mukaisesti haluttiin ottaa prosessin käyttöön mahdollisimman nopeasti ja jatkaa sen kehittämistä saadun käytännön palautteen perusteella.

Itse toimin Scrum Masterina tiimissä, jossa otettiin ensimmäisenä käyttöön uusi prosessi. Prosessi otettiin käyttöön uuden versioprojektin alkaessa syksyllä 2012. Scrum Masterin roolissa pyrin selventämään prosessin käyttöä jokaisessa tiimin kohtaamassa tilanteessa, sekä viemään eteenpäin mahdollisia ongelmakohtia, jotta niihin pystyttiin yhdessä löytämään paras mahdollinen ratkaisu.

Vertaileva kyselytutkimus toteutettiin loppusyksystä 2012. Yksi tiimi oli tähän mennessä käyttänyt jo prosessia, kun taas osa ei. Haluttiin kuitenkin saada jo palautetta määritellystä prosessista, joten pyydettiin palautetta kirjallisen määritelmän perusteella, jos henkilöllä ei vielä ollut käytännön kokemusta uudesta tuotekehitysprosessista.

7.2 Vertailevan tutkimuksen toteutus ja analysointi

Vertaileva tutkimus toteutettiin marraskuussa 2012 ja se toteutettiin lomaketutkimuksena. Tutkimus lähetettiin yhteensä 22 henkilölle ja tällä kertaa lomaketutkimukseen vastaaminen ei ollut anonyymiä. Haluttiin pystyä seuraamaan kuka tutkimukseen on vastannut, jotta saataisiin mahdollisimman paljon vastauksia. Tutkimuksen vastaanotti 14 tuotekehittäjää, 6 testaajaa ja 1 esimies. Tutkimusta ei lähetetty tuotehallinnan edustajille, koska prosessi haluttiin esitellä tuotehallinnalle vasta sitten, kun kaikki tuotekehitystiimit alkaisivat toimia sen mukaisesti.

Kyselyyn vastasi 15 (65,2 %) henkilöä, joista 9 (60 %) henkilöä toimi tuotekehittäjän roolissa ja 6 (40 %) testaajana. Huomionarvoista oli, että kaikki testaajat vastasivat kyselyyn, kun taas tuotekehittäjistä vastasi vain hieman yli puolet (64,2 %).

Tutkimuksen kysymykset olivat kohdistettu määriteltyihin Scrumin tapahtumiin. Lisäksi kysyttiin yleisesti koko prosessista, sekä kuinka pitkiä sprinttien tulisi olla. Kysymykset sisälsivät linkin yrityksen sisäiseen wikiin, josta vastaaja pystyi tutustumaan aiheeseen, jos hänellä ei vielä ollut prosessista käytännön kokemusta.

Lomaketutkimuksen kysymykset:

1. Lue WIKI-artikkeli julkaisun suunnittelupalaverista. Kerro mielipiteesi prosessin julkaisun suunnittelupalaveriosiosta. Puuttuuko prosessin tästä osiosta jotain? Pitäisikö jotain muuttaa? Kerro mielipiteesi siitä, millainen julkaisun suunnittelupalaveri meillä tulisi olla?
2. Lue WIKI-artikkeli kehitysjonon työstöpalaverista. Kerro mitä mieltä olet prosessin työstöpalaverin osiosta? Puuttuuko prosessin tästä osiosta jotain? Tulisiko jotain muuttaa? Kerro mielipiteesi siitä, millainen kehitysjonon työstöpalaveri meillä tulisi olla?
3. Lue WIKI-artikkeli sprintin suunnittelupalaverista. Kerro mitä mieltä olet prosessin suunnittelupalaverin osiosta? Puuttuuko prosessin tästä osiosta jotain? Tulisiko jotain muuttaa? Kerro mielipiteesi siitä, millainen sprintin suunnittelupalaveri meillä tulisi olla?
4. Lue WIKI-artikkeli sprintin päiväpalaverista. Kerro mitä mieltä olet prosessin päiväpalaverin osiosta? Puuttuuko prosessin tästä osiosta jotain? Tulisiko jo-

tain muuttaa? Kerro mielipiteesi siitä, millainen sprintin päiväpalaveri meillä tulisi olla?

5. Lue WIKI-artikkeli sprinttikatselmuksesta. Kerro mitä mieltä olet prosessin sprinttikatselmuksen osiosta? Puuttuuko prosessin tästä osiosta jotain? Tulisiko jotain muuttaa? Kerro mielipiteesi siitä, millainen sprinttikatselmus meillä tulisi olla?
6. Lue WIKI-artikkeli retrospektiivistä. Kerro mitä mieltä olet prosessin retrospektiivin osiosta? Puuttuuko prosessin tästä osiosta jotain? Tulisiko jotain muuttaa? Kerro mielipiteesi siitä, millainen retrospektiivi meillä tulisi olla?
7. Lue OpusCapita Way of Scrum WIKI -sivut ja anna meille palautetta sen sisällöstä ja rakenteesta.
8. Olemme tällä hetkellä suunnittelemassa neljän viikon pituisia sprinttejä. Kuinka pitkiä sprinttien tulisi mielestäsi olla?

Kaikki kyselyyn vastanneet vastasivat jokaiseen kysymykseen. Kysymykset esitettiin Englanniksi, koska vastaanottajien joukossa oli Suomea osaamattomia henkilöitä. Vastata sai sekä Suomeksi että Englanniksi.

Lomaketutkimuksen vastaukset käsiteltiin ja analysoitiin vastaavalla tavalla, kuin aikaisempi tutkimus. Vastauksissa esiintyneet teemat kirjattiin ylös ja niiden esiintymismäärät laskettiin. Näin saatiin selville vastauksien yleisimmät teemat, jotka esitellään seuraavissa luvuissa.

Julkaisun suunnittelupalaveri oli täysin uusi asia organisaatiolle. Siitä pidettiin yleisesti ja yleisin kommentti olikin, että se on ajatuksena hyvä ja ehdottomasti kokeilemisen arvoinen asia. Vastauksista ei noussut mitään asiaa ylitse muiden, vaan käsitellyt teemat hajautuivat hyvin laajasti.

Ainoa teema, jota käsitteli useampi, kuin yksi vastaus, oli testauksen tuominen esiin paremmin. Testaajat ja testaus haluttiin selkeästi mukaan julkaisun suunnitteluun. Suunnitelmaan haluttiin mukaan myös testisuunnitelma, joka sisältäisi mm. vaaditun laadun määritelmän ja testattavat asiat (toiminnalliset ja ei-toiminnalliset). Kaikki vastaajat, jotka nostivat testausta esiin, olivat yrityksen testaajia.

Muita mainittuja asioita olivat mm. kehitysjonoon liittyviä. Haluttiin, että kehitysjono toimitettaisiin etukäteen tutustuttavaksi kaikille suunnittelupalaveriin osallistuville. Haluttiin myös, että kehitysjono olisi aina helposti saatavilla ja sen edistymistä pystyttäisiin helposti seuraamaan.

Kehitysjonon työstöpalaveri koettiin pääasiassa hyvänä asiana. Kolmannes vastaajista piti työstöpalaveria hyvänä asiana. Muita esiin nousseita teemoja olivat työstöpalaverien ajankohtaan, sekä valmiin määritelmään liittyviä kommentteja ja kysymyksiä.

20 % vastaajista kommentoi työstöpalaverien ajankohtaa ja erityisesti sitä, kuinka usein palavereja pidettäisiin. Eniten oltiin sitä mieltä, että ohjearvona oleva tunti viikossa olisi liikaa, varsinkin jos koko kehitystiimi osallistuisi palaveriin. Huolena oli myös se, miten varmistuttaisiin, että palavereja todella pidettäisiin.

20 % vastaajista nosti esiin valmiin määritelmän ja mitä se pitäisi sisällään. Määriteltiin, että jokainen tiimi voisi muodostaa oman valmiin määritelmänsä. Kaikki kyseiseen teemaan liittyvät vastaukset liittyivät valmiin määritelmän sisältöön.

Muut vastaukset hajaantuivat useampiin teemoihin, joista yksikään ei esiintynyt useammassa, kuin yhdessä vastauksessa. Vastaukset käsittelivät mm. tiedon jakamista ja pidemmän aikavälin suunnittelua.

Sprintin suunnittelupalaverin yleisimmäksi teemaksi nousi ohjelmistovirheiden käsittely. Jopa 40 % vastauksista käsitteli bugien hallintaa sprintin suunnittelupalaverissa ja sprintin aikana. Useimmat näistä vastauksista ottivat kantaa esitettyyn bugikynnykseen ja miten bugien korjaus otettaisiin huomioon sprintin suunnittelussa.

Palaverin kesto mainittiin myös parissa vastauksessa. Vastauksissa oltiin epäileväisiä siitä, riittäisikö esimerkkinä mainittu aikataulu koko sprintin suunnitteluun. Myös työmääräarvioiden tekeminen (estimointi) sprintin töistä mainittiin kahdessa vastauksista. Muut teemat hajaantuivat yksittäisiin esiintymisiin, eikä muita teemoja noussut selkeästi esiin.

Työmääräarvioiden tekeminen koettiin vaikeaksi. Estimointi on laajempi ongelma ja sen tekemiseen on useita eri käytäntöjä. Osa muiden yritysten tuotekehitystiimeistä ovat luopuneet estimaatioista tai he ovat estimoineet abstrakteissa yksiköissä, jolloin estimoidaan ainoastaan ominaisuuksien suhdetta toisiinsa. Suurin ongelma on kuitenkin vaatimusmäärittelyjen puutteellisuus. Vaatimusmäärittelyjä tulee saada kattavammiksi ja sitä varten prosessissa onkin otettu käyttöön kehitysjonon työstöpalaverit.

Päiväpalaverin vastauksien perusteella voidaan sanoa, että päiväpalaveri oli määritelty uudessa prosessissa liian tarkasti. Määritelmästä oli tullut liian jäykkä ja virallisen oloinen, kun taas itse määriteltävä palaveri on hyvin epävirallinen ja pienimuotoinen.

Vastauksissa selkeästi yleisimmäksi teemaksi nousi päiväpalaverin ajankohta. Uuden prosessin määritelmässä korostettiin sitä, että kaikkien tulee olla paikalla ajoissa eikä myöhästymisiä tulla sallimaan. Myöhästymisen seurauksia kommentoitiin myös vastauksissa. Moni oli myös sitä mieltä, että päiväpalaverin ei tarvitsisi olla joka päivä, vaan joka toinen päivä riittäisi. Monessa tiimissä on myös sen verran vähän jäseniä, että heidän on helppo kommunikoida tekemisensä myös ilman päivittäistäkin palaveria.

Päiväpalaverin sisältö nousi toiseksi yleisimmäksi teemaksi. 27 % vastauksista käsitteli päiväpalaverin sisältöä. Erityisesti oltiin huolissaan siitä, miten muiden tekeminen ei välttämättä ole merkityksellistä kaikille tiimin jäsenille.

Muita mainittuja teemoja olivat osallistujien roolitus, sekä paikka missä päiväpalaveri pidettäisiin. Nämä teemat mainittiin kuitenkin vain kerran, eivätkä siten nousseet esiin merkittävinä asioina.

Sprinttikatselmuksen määritelmää pidettiin yleisesti ottaen hyvänä. Eniten vastauksissa esiintyi keskeneräisten asioiden demoaminen. Kolmannes vastauksista käsitteli tätä teemaa.

Uuden prosessin määritelmän mukaan vain valmiin määritelmän täyttävät ominaisuudet voidaan esitellä sprinttikatselmuksessa. Suurin osa vastaajista epäili, että täl-

laisella määrityksellä ei pystytä useinkaan esittelemään mitään, koska usein ominaisuuksia saadaan kehitettyä vasta sprintin loppupuolella, jolloin niitä ei ehditä testaamaan tai viimeistelemään ennen sprinttikatselmusta.

Toisaalta haluttiin myös, että pystyttäisiin esittelemään keskeneräisiä töitä, koska tällöin saataisiin arvokasta tietoa, jonka perusteella voitaisiin vielä tehdä muutoksia kehityksen alla olevaan ominaisuuteen. Määritellyn prosessin mukaan tällainen ei ole mahdollista. Jokainen ominaisuus tulisi saada kokonaan valmiiksi sprintin aikana, eikä Scrum-menetelmä mahdollista muunlaista kehitysmallia.

Muut vastauksissa esiintyneet teemat olivat yksittäisiä tapauksia. Näissä esimerkiksi haluttiin, että katselmointiin osallistuisi mahdollisimman kattavasti henkilöitä eri rooleista. Näin saataisiin hyödyllisiä kommentteja eri näkökulmista.

Huomionarvoista on myös se, että yksi vastaaja ehdotti sprinttikatselmuksen yhdistämistä julkaisun suunnittelupalaveriin. Vastaus osoittaa sen, ettei kyseinen vastaaja ole ymmärtänyt prosessin tarkoitusta. Sprinttikatselmuksen tarkoitus on jäänyt myös hyvin epäselväksi.

Retrospektiivin vastauksista yksi teema nousi selkeästi ylitse muiden. Lähes puolet (47 %) vastauksista käsitteli retrospektiivissä päätettyjen parannusten toteuttamista. Kun otetaan huomioon, että osa vastauksista ei sisältänyt teemaa, niin parannusten toteuttamisen teema oli aiheena lähes 80 %:ssa vastauksista. Suurimmalla osalla vastaajista oli kokemuksia, ettei retrospektiivissä tunnistettuja ongelmakohtia ole kuitenkaan saatu korjattua ja samat ongelmat esiintyvät retrospektiiveissä toistuvasti. Tästä johtuen ehdotettiin, että retrospektiivi pidettäisiin harvemmin tai sitä ei pidettäisi lainkaan.

Toinen mainittu teema käsitteli palaveriin osallistujia. Koska palaverissa on tarkoitus myös puhua asioista, jotka eivät sujuneet hyvin, toivottiin että palaveriin osallistuisi ainoastaan kehitystiimi, eikä ulkopuolisia osallistujia sallittaisi.

Yleinen palaute oli pääosin positiivista, mutta myös odottavaista. Vastauksista tuli esiin epäilys siitä, miten prosessi saataisiin otettua käyttöön. Kahdessa vastauksessa

nostettiin esiin mahdollinen ongelma prosessin jäykkyyden suhteen. Uusi prosessi on määritelty hyvin tarkasti ja osittain ehkä liiankin tarkkaan. Tämä on mahdollisesti antanut kuvan liian jäykästä prosessista.

Muissa vastauksissa haluttiin prosessin määritelmään tarkempaa kuvausta tietyistä tilanteista. Muita yleisiä teemoja ei noussut esiin ja vaikutti siltä, että prosessi otettiin pääosin positiivisesti vastaan.

Viimeisenä lomaketutkimuksen kysymyksenä kysyttiin vastaajilta mikä heidän mielestään olisi paras sprintin pituus. 50 % vastaajista (7 kpl) vastasi, että 4 viikon sprintti olisi paras pituus. 3 viikon sprintin puolesta oli kuitenkin lähes yhtä paljon (5 kpl). Yksi vastaaja ehdotti myös kahden viikon ja viiden viikon pituisia sprinttejä.

Lomaketutkimuksena toteutetun tutkimuksen tulokset olivat pääasiassa positiivisia. Suurin osa vastaajista piti määriteltyä prosessia hyvänä. Osa vastaajista toi myös esiin omia toiveitaan prosessin muuttamiseksi. Tuloksista oli välillä hankala löytää perimmäisiä syitä, joiden perusteella muutoksia ehdotettiin. Pyrittiin miettimään, mitkä nämä perimmäiset syyt olivat ja vasta sen jälkeen miettimään mahdollisia muutoksia määriteltyyn prosessiin. Suurin osa tiimeistä ei ollut vielä päässyt käytännössä käyttämään uutta prosessia eikä tästä syystä haluttu tehdä vielä tämän tutkimuksen perusteella suuria muutoksia määriteltyyn prosessiin.

Osa vastaajista oli ymmärtänyt määritellyn prosessin hieman liian kirjaimellisesti. Scrum-menetelmässä kaikki palaverit ja tapahtumat ovat aina aikarajattuja. Osa vastaajista oli selkeästi olettanut, että palaverien tulee aina kestää ilmoitettu aika. Esimerkiksi sprinttikatselmuksen määritelmässä mainittiin, ettei demoihin valmistautuminen saisi kestää kauempaa, kuin 2 tuntia. Osa vastaajista oli ymmärtänyt tämän niin, että demoihin tulee valmistautua kaksi tuntia. Yhdessäkään palaverissa ei ole tarkoitus viettää ylimääräistä aikaa ja siksi jokaiselle palaverille ja tapahtumalle määriteltiin vaaditut tulosteet, jotta palaveri saadaan pidettyä tuottavana ja päätettyä heti, kun vaaditut tulosteet ovat selvillä.

Määrittelyiden yhteyteen lisättiin myös esimerkki palaverin aikataulusta. Osa vastaajista oli selkeästi olettanut, että aikataulu oli sääntö, eikä vain yksi esimerkki palave-

rin kulusta. Tämänkaltaiset vastaukset olivat tulleet henkilöiltä, jotka eivät vielä olleet käyttäneet uutta prosessia käytännössä. Koettiin, että tämänkaltaiset väärinymmärrykset korjaantuisivat käytännön kautta, kun ihmiset huomaisivat, miten palaverit käytännössä sujuvat.

Julkaisun suunnittelupalaveri oli uusi käytäntö koko yritykselle eikä sitä ollut kokeiltu käytännössä vielä yhdessäkään yrityksen projekteista. Palaveri koettiin pääasiassa hyvänä eikä siksi lähdetty tekemään siihen tässä vaiheessa muutoksia.

Kehitysjonon työstöpalaverien tarkoitus oli parantaa ominaisuuksien määrityksien laatua ja ne koettiinkin pääasiassa hyvänä. Myös tuotehallinnalta saatiin positiivista palautetta työstöpalavereista ja haluttiin ehdottomasti jatkaa niiden pitämistä. Tutkimuksessa esiin nousseiden teemojen pohjalta ei lähdetty tekemään muutoksia työstöpalaverin käytäntöihin.

Osa vastaajista halusi, että vain osan tiimistä tarvitsisi osallistua kehitysjonon työstöpalaveriin, koska palaverin aiheen ei koeta aina koskettavan kaikkia tiimin jäseniä. Havaintojen perusteella tämä johtuu enemmänkin siitä, ettei tiimi koe olevansa tiimi, jolla on yksi yhteinen tavoite. Kohdeyrityksen tuotekehityksessä osa tuotekehittäjistä on erikoistunut tiettyihin ohjelman osiin. Kehittäjät kokevat, etteivät muut ohjelman osat välttämättä kosketa heitä, jolloin he eivät myöskään ole kiinnostuneita näiden muiden alueiden ominaisuuksien työstöpalavereista. Tulisikin keskittyä enemmän tiimin rakentamiseen ja yhteistyön lisäämiseen tiimin sisällä, kuin edesauttaa tiimin jäsenien eristäytymistä määrittelemällä, ettei kaikkia tiimin jäseniä kutsuta kaikkiin työstöpalavereihin. Tästä syystä työstöpalaverin käytäntöihin ei tehty muutoksia.

Sprintin suunnittelupalaverin vastauksissa yleisin teema oli ohjelmistovirheiden käsittely. Tähän haluttiin puuttua uuden prosessin jatkokehityksessä ja sitä varten määriteltiin tarkempia ohjeita ohjelmistovirheiden käsittelyyn. Määriteltiin yksityiskohtaisesti miten ohjelmistovirheitä tulee käsitellä. Ohjelmistovirheiden käsittelyn prosessi on sprintin sisäisten käytäntöjen määrittelyä ja siksi se rajataan pois opinnäytetyön piiristä. Se ei vaikuta tämän opinnäytetyön tuloksena kehitetyn prosessin kulkuun. Tarkoituksena oli yhtenäistää ohjelmistovirheiden käsittelyä ja saada prosessi määriteltyä niin, ettei sen noudattamisessa tule ongelmatilanteita.

Päiväpalaverin vastauksien taustalla saattoi olla samaa ongelmaa, kuin kehitysjonon työstöpalaverin vastauksissa. Tiimi ei toimi yhdessä yhden tavoitteen eteen, vaan jokaisella on omat työnsä eivätkä muiden tekemisistä kiinnosta kuunnella. Tässäkin tapauksessa päätettiin keskittyä enemmän tiimin rakentamiseen, kuin päiväpalaverin käytäntöjen muuttamiseen. Ymmärrettiin kuitenkin, että jos kyseessä oli hyvin pieni tiimi, joka työskenteli tiiviisti yhdessä, niin päiväpalavereja ei välttämättä tarvittu niin usein. Näiden tiimien kanssa sovittiin, että päiväpalavereja voitaisiin pitää harvemmin.

Pelkällä prosessilla ei pystytä tekemään toimimattomista tiimeistä toimivia ja onkin syytä huomata, että prosessin toimivuuden kannalta on tärkeää, miten ihmiset tekevät yhteistyötä keskenään. Prosessia kehitettäessä tämä ongelma tiedostettiin ja sitä pyrittiin parantamaan, kun mietittiin tiimien koostumuksia ja tuotevastuita.

Sprinttikatselmuksen vastauksista tuli hyvin esille huoli siitä, miten hyvin ominaisuuksia saataisiin todella valmiiksi sprintin aikana. Scrum-menetelmän mukaisesti katselmoinnissa ei saa demota keskeneräisiä töitä, koska silloin tiimi ikään kuin pakotetaan tekemään ominaisuudet täysin valmiiksi. Sprinttikatselmuksen tärkein tavoite on saada hyvä käsitys siitä, missä tilassa tuote on, jotta tuotehallinta pystyy tekemään oikeita päätöksiä tuotteen kehityksen jatkosta. Sallimalla keskeneräisten töiden demoaminen aiheuttaisi ominaisuuksien jäämisen kesken eikä tuotehallinnalla olisi todellista kuvaa siitä, mitä on vielä tekemättä. Kuva tuotteen nykytilasta ei olisi todenmukaisen, eikä sen perusteella pystyttäisi tekemään päätöksiä tuotteen kehityksen jatkosta.

Vastauksissa toivottiin myös mahdollisimman laajaa osallistujakuntaa. Tämä kommentti koettiin hyväksi ja tämän jälkeen sprinttikatselmuksen kutsu lähetettiin lähes koko yritykselle. Haluttiin, että kaikki voisivat halutessaan osallistua katselmukseen, jossa he näkisivät tuotteen uusia ominaisuuksia. Sprinttikatselmus muutettiin myös online-kokoukseksi, jotta kaikkien halukkaiden olisi mahdollista osallistua palaveriin. Scrum-tiimi osallistui kuitenkin palaveriin fyysisesti, kuten ennenkin.

Retrospektiivin ongelmat oli tiedostettu jo aiemmin. Ongelmana oli parannusehdotusten toteuttaminen, joka oli usein jäänyt tekemättä. Tämä johtui havaintojen perusteella siitä, että parannusehdotukset koskivat usein tiimin ulkoisia asioita, kuten tuotehallinnan vaatimusmäärittelyjen puutteellisuutta. Scrum-tiimi ei itse pysty parantamaan tätä asiaa ja siitä syystä parannuksia on jäänyt toteuttamatta. Jatkossa Scrum Masterin haluttiin ohjaavan keskustelua tiimin sisäisiin asioihin, joita tiimillä oli täysi mahdollisuus muuttaa. Tiimin ulkoiset ongelmat kirjattiin myös ylös ja kommunikoitiin tarvittaville tahoille.

Vastauksissa haluttiin, että tietoa retrospektiivien tuloksista jaettaisiin muille tiimeille. Tämä oli hyvä idea ja toimintaa muutettiin niin, että retrospektiividokumentti jaettaisiin jatkossa kaikille Scrum-tiimeille.

Vastauksissa toivottiin myös, ettei retrospektiiviin kutsuttaisi tuotehallintaa. Retrospektiivi on pääasiassa tarkoitettu tiimin sisäiseksi tavaksi parantaa ja tehostaa työskentelytapojaan. Kyselyn vastauksien perusteella ainakin osa tiimin jäsenistä kokee tilanteen hankalaksi, jos tuotehallinta on läsnä palaverissa. Määriteltiin, että jokainen tiimi voi keskenään päättää kutsuvatko he tuotehallintaa retrospektiiviin vai eivät. Retrospektiivin tulokset jaetaan myös tuotehallinnalle, jotta myös he ovat tietoisia tiimin mahdollisista ongelmista ja pystyvät omalta osaltaan tehostamaan tiimin toimintaa.

Sprintin pituudessa päädyttiin pitäytymään määritellyssä neljän viikon ajassa. Vastauksien perusteella harkittiin myös lyhyempää sprinttiä, mutta tässä vaiheessa ei nähty syytä muuttaa käytäntöjä. Jälkikäteen ajateltuna kysymys sprintin pituudesta oli johdatteleva. Kysymyksessä mainittiin, että ollaan suunnittelemassa neljän viikon pituisia sprinttejä. Useimmat vastaajat vastasivat haluavansa neljän viikon sprintin, muttei voida tietää vääristikö kysymyksen asettelu tuloksia vai ei. Sprintin pituuksia voidaan helposti muuttaa myöhemminkin, joten suuresta ongelmasta ei ollut kyse.

Tutkimuksen perusteella ei nähty tarvetta prosessin kannalta merkittäviin muutoksiin. Muutoksia tehtiin vain palaverien käytäntöihin, mutta prosessia itsessään ei lähdetty muuttamaan. Suurin osa tutkimuksen paljastamista ongelmista liittyi tiimien toimintaan ja yhteistyöhön tuotehallinnan ja kehitystiimien välillä. Tähän ei pystytä

puuttumaan prosessin tasolla, vaan pyrittiin parantamaan tilannetta muilla toimenpiteillä. Koettiin myös, että jos prosessi saadaan toimimaan niin, että se on kaikille selkeää, se myös parantaisi luottamusta ja yhteistyötä eri tahojen välillä.

7.3 Työskentely uuden prosessin kanssa

Kohdeyrityksessä siirryttiin vaiheittain uuden prosessin käyttöön. Kaksi tuotekehitystiimiä oli toiminut prosessin mukaisesti jo syksystä 2012 alkaen. Muut kehitystiimit siirtyivät käyttämään uutta prosessia vuoden 2013 alusta alkaen. Kohdeyrityksessä aloitettiin uuden tuoteversion kehitysprojekti vuoden 2013 alussa ja oli luonnollista siirtyä siinä vaiheessa uuden prosessin käyttöön kaikissa tiimeissä. Myös tiimien koostumuksia muutettiin versiokehityksen alkaessa.

Prosessin toimivuutta voidaan parhaiten arvioida tarkastelemalla tuotekehitystiimien retrospektiivien tuloksia. Kohdeyrityksessä toimittiin uuden tuotekehitysprosessin mukaisesti yli vuoden ajan ja tältä ajalta tarkastellaan tässä yhteydessä neljää retrospektiiviä.

Ensimmäinen tarkasteltava retrospektiivi on kolmannen sprintin retrospektiivi (28.3.2013) ja toinen on lähes vuotta myöhemmin olleen uuden versioprojektin sprintin 8 retrospektiivi (17.3.2014).

Kaksi muuta tarkasteltavaa retrospektiiviä ovat toisen tiimin retrospektiivit. Ensimmäinen retrospektiivi on maaliskuulta 2013 ja toinen lähes vuotta myöhemmin tammikuussa 2014 pidetyltä retrospektiiviltä.

Retrospektiivejä tarkastellaan kahdessa ryhmässä. Ensin vuoden 2013 retrospektiivejä, jonka jälkeen voidaan verrata tuloksia vuonna 2014 pidettyihin retrospektiiveihin. Tämän jälkeen voidaan tarkastella kuinka hyvin uusi tuotekehitysprosessi on ratkaissut vanhan tuotekehitysprosessin ongelmia, jotka on mainittu luvussa 3.6.

Kuten voidaan huomata, suunnittelupalavereja, katselmointeja tai retrospektiivejä ei pystytty pitämään samanaikaisesti kaikkien tiimien kanssa. Poissaolot ja muut tapah-

tumat aiheuttivat tiimeissä muutoksia, joiden vuoksi samanaikaisista sprinteistä ja palavereista jouduttiin luopumaan. Tämän ei kuitenkaan koettu haittaavan prosessin toimivuutta.

7.3.1 Retrospektiivit vuonna 2013

Ensimmäisen tiimin retrospektiivin mukaan tiimi sai melkein koko sprintin tehtävälistan toteutettua. Myös tiimin sisäistä yhteistyötä pidettiin hyvänä. Tiimin jäsenet vaihtoivat tehtäviä keskenään, sen mukaisesti kenellä oli eniten aikaa toteuttaa jokin asia. Onnistumisena mainittiin myös ohjelmistovirheiden nopea ja laadukas korjaaminen. Myös kaikki toteutetut ominaisuudet hyväksyttiin tuotehallinnan puolelta katselmoinnin yhteydessä. Tiimi pystyi myös keskittymään sprintin tehtävälistan tehtäviin, eivätkä suunnittelemattomat ylläpitotehtävät häirinneet tiimin työskentelyä.

Ongelmiksi tiimi mainitsi vaatimusmäärittelyt. Sprintin aikana vaatimusmäärittelyjä muutettiin alkuperäisestä tai ne puuttuivat kokonaan. Tästä johtuen tiimin jäsenet joutuivat etsimään vastauksia avonaisiin kysymyksiin ja siihen miten ominaisuus tulisi toteuttaa. Muita tiimin kokemia ongelmia olivat julkaisutestauksen kestäminen odotettua kauemmin, sekä läpinäkyvyys sprintin töiden edistymiseen.

Parannusehdotuksina tiimi ehdotti teknisten asiantuntijoiden ottamista mukaan uusien ominaisuuksien konseptisuunnitteluun. Ominaisuuksien toteutukseen liittyvien asioiden tulisi myös olla saatavilla suoraan ominaisuuden TFS-kortilta.

Tiimi halusi myös puuttua tiimin rakenteeseen. Samassa tiimissä työskenteli eri tuotemoduulien kehittäjiä. Tiimillä oli kuitenkin yksi yhteinen tehtävälista. Käytännössä kehittäjät ottivat listalta kehitettäväksi vain oman moduulinsa tehtäviä, jolloin yhteinen tehtävälista menetti merkityksensä ja hankaloitti tiimin toimintaa. Tästä johtuen tiimi halusi erilliset listat eri moduuleille, jolloin kehittäjien ja testaajien olisi helpompi seurata oman alueensa tehtäviä. Tällaiset näkymät päätettiin tehdä seuraavaksi sprintiksi.

Toisen kehitystiimin retrospektiivissä onnistuneiksi asioiksi mainittiin työmääräarvioiden tarkkuudet, pidetyt kehitysjonon työstöpalaverit sekä tiimin sisäinen yhteistyö. Myös toinen tiimi onnistui lähes kokonaan toteuttamaan sprintin tehtävälistan ominaisuudet.

Työstöpalavereja ei ollut järjestetty kaikista toteutettavista ominaisuuksista ja se koettiin ongelmaksi. Samoin ongelmaksi koettiin muutoksien toteuttaminen sprintin keskellä. Osa ominaisuuksien vaatimusmäärityksistä oli ristiriitaisia ja muuttui kesken sprintin. Ominaisuuksien hyväksyntäkriteerien mukainen kehittäjän oma testaus tehtiin kehittäjän omassa ympäristössä, joka koettiin ongelmalliseksi.

Ongelmakohtien ratkaisuksi tiimi ehdotti kehittäjän tekemän testauksen tekemistä QA-ympäristössä, jossa ohjelmiston virallinen julkaisuversio oli asennettuna. Näin varmistutaan siitä, että ominaisuus on todella viimeistelty ja toimitettavissa asiakkaalle. Toisena parannusehdotuksena tiimi ehdotti testitapausten tekemisen yhdessä testaajan ja kehittäjän kanssa. Näin testitapauksiin saataisiin myös kehittäjän mielipide ja myöhemmin vältyttäisiin mahdollisilta ristiriidoilta. Viimeisimpänä parannusehdotuksena tiimi ehdotti sprintin aikana tehtävien muutosten parempaa hallintaa.

Verrattaessa molempien tiimien retrospektiivejä voidaan helposti huomata samoja ongelmia. Molemmat tiimit kokivat ongelmia vaatimusmäärittelyjen kanssa. Myös sprintin aikana tehtyjen muutosten hallinta koettiin ongelmalliseksi molemmissa tiimeissä. Muutokset johtuivat pääasiassa vaatimusmäärittelyjen muuttumisista.

Molemmat tiimit kuitenkin onnistuivat sprinteissään hyvin saaden lähes kaiken valmiiksi. Tätä voidaan pitää erinomaisena saavutuksena, kun otetaan huomioon epämääräiset vaatimusmäärittelyt ja kesken sprintin tehdyt muutokset. Huomionarvoista on, että molemmat tiimit kokivat sprintin aikana muutoksia ominaisuuksien vaatimusmäärittelyissä.

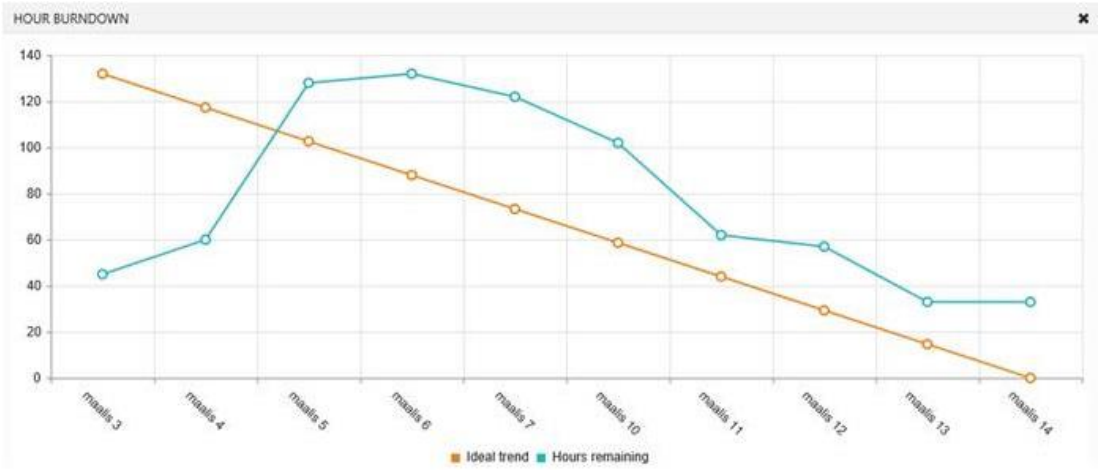
Tiimien ehdottamat parannusehdotukset erosivat hieman toisistaan. Ensimmäinen tiimi ehdotti tuotekehittäjien läsnäoloa jo ominaisuuden konseptisuunnittelussa. Tiimi pyrki tällä parantamaan ominaisuuksien suunnittelua, niin että siinä on otettu

huomioon tuotteen tekninen toteutus. Näin ominaisuuden määrittely olisi teknisesti toteutuskelpoinen ja mahdollisilta muutoksilta vältyttäisiin.

Toinen tiimi ehdotti työstöpalaverien lisäämistä ja tämän avulla varmistamaan vaatimusmäärittelyjen kattavuus. Tiimi oli kokenut, että ne ominaisuudet, joista oli järjestetty työstöpalaveri, olivat tarkemmin määriteltyjä kuin ne, mistä palavereja ei ollut järjestetty.

7.3.2 Retrospektiivit vuonna 2014

Ensimmäisen tiimin maaliskuussa 2014 pitämä retrospektiivi käsitteli silloisen ohjelmistoversion kahdeksatta sprinttiä. Sprintin sisältö oli pääasiassa edellisen version viimeistelyä, joka vaikutti myös sprintin suunnitteluun. Sprintin koko sisältö ei ollut vielä suunnittelupalaverissa täysin tiedossa. Tämä näkyy myös tiimin burndown chart -raportissa (kuvio 17).



Kuvio 17. Sprintin 8 burndown chart –raportti

Raportista on selkeästi nähtävissä, että tehtävä työ selvisi tiimille vasta ensimmäisen kolmen päivän aikana. On myös huomionarvoista, että tässä vaiheessa kohdeyrityksen tuotekehityksessä oli siirrytty kahden viikon pituisiin sprintteihin. Kahden viikon sprintteihin siirryttiin, koska haluttiin pitää suunnittelupalaverit entistä kevyempinä.

Tiimi sai kuitenkin toteutettua lähes kaiken sprintille suunnitellun työn. Sprintin aikana tiimi kuitenkin kohtasi useita ongelmia, jotka aiheuttivat sen, ettei tiimi saanut kaikkea suunniteltua työtä valmiiksi. Tiimillä oli yhä hankaluuksia saada tarpeeksi kattavia vaatimusmäärittelyjä halutuista ominaisuuksista. Puutteelliset vaatimukset hidastivat työn etenemistä, kun ominaisuuksia jouduttiin täydentämään myöhemmin paljastuneiden vaatimusten perusteella.

Suunnittelemattomien töiden määrä oli sprintin aikana huomattava. Tämä johtui pääasiassa meneillään olevasta asiakasprojektista. Tiimi joutui antamaan paljon tukea kyseiselle asiakasprojektille, joka aiheutti ylimääräistä työtä. Tämä ongelma koettiin kuitenkin vain väliaikaiseksi ja tilanne palautuisi takaisin normaaliksi seuraavan sprintin aikana.

Tiimi koki ongelmaksi myös regressiovirheiden suuren määrän. Ohjelmistovirheiden korjaus aiheutti usein jonkin muun asian hajoamisen ja tilanteen stabiloiminen oli hankalaa. Ohjelmiston monimutkaisuus ja automaatiotestauksen puute hankaloitti ohjelmiston toiminnallisuuksien verifiointia. Puutteista johtuen osa korjauksista aiheutti uusia ongelmia, joita ei kyetty havainnoimaan ajoissa.

Toisen tiimin sprintti oli myös hyvin onnistunut. Kaikki suunnitellut ominaisuudet saatiin ajoissa valmiiksi. Myös toinen tiimi viimeisteli silloista ohjelmistoversiota ja saikin version omalta osaltaan valmiiksi kyseisen sprintin aikana. Tiimi koki myös TFS-toiminnanohjausjärjestelmän uudistuksen onnistuneen hyvin. Kohdeyrityksessä oli otettu käyttöön uusi versio TFS-ohjelmistosta ja tiimi koki uuden version hyvänä.

Toisella tiimillä oli sprintin aikana myös ongelmia. Uuden TFS-version käytössä oli hieman epäselvyyksiä ja osaa tehdyistä ominaisuuksista ei ehditty testaamaan sprintin aikana. Tämä kerrytti testivelkaa seuraavalle sprintille, joka on ollut aikaisemminkin kohdeyrityksen tuotekehityksen ongelma.

Kuten ensimmäinenkin tiimi, myös toinen tiimi koki, että heidän tiiminsä sisällä oli useita eri alitiimejä. Eri henkilöt kehittivät ja testasivat eri tuotteen osia ja näistä henkilöistä muotoutui tiimin sisälle alitiimejä, joilla jokaisella oli omat tavoitteensa ja tehtävänsä.

Tällä sprintillä aloitettiin myös uuden ohjelmistoversion kehitys. Tiimi koki ongelmaksi, että testiympäristö ei ollut vielä kyseiselle versiolle valmis. Tiimi otti yhdeksi parannuskohteekseen testiympäristön tekemisen jatkossa ajoissa ennen kuin uuden version kehitys alkaisi. Toiseksi kehityskohteekseen tiimi valitsi TFS-käytäntöjen selventämisen kaikille. Uusi TFS-versio oli osittain muuttunut aikaisemmasta eivätkä vanhat käytännöt olleet enää sovellettavissa uuden version kanssa. Viimeisimpänä kehityskohteena tiimi päätti kokeilla päiväpalaverien jakamista useammaksi palaveriksi, joissa paikalla olisi vain tietyn tuoteosan kehittäjät ja testaajat. Tämä ratkaisisi alitiimien aiheuttamia ongelmia, joissa muiden tuoteosioiden kehittäminen ei välttämättä kiinnosta kaikkia tiimin jäseniä.

Molemmat tiimit onnistuivat ongelmista huolimatta sprinteissään hyvin. Koetut ongelmat olivat kuitenkin hyvin samankaltaisia, kuin edellisenä vuonna. Vaatimusmäärittelyn ongelmat olivat yhä toisen tiimin ongelma ja toinen tiimi ei saanut testattua kaikkea sprintin työtä, muodostaen testivelkaa seuraavalle sprintille. Muut tiimien kokemat ongelmat tuntuivat olevan yksittäistapauksia, jotka tiimi pystyisi ratkaisemaan eivätkä ne toistuisi enää myöhempien sprinttien aikana.

Sprinttien toteuttaminen onnistui vuoden harjoittelun jälkeen suhteellisen hyvin ja suunnitellut työt saatiin hyvin lähelle valmiiksi. Yksittäisten sprinttien retrospektiivien tarkastelu ei kerro kuitenkaan koko totuutta tuotekehityksestä. Ohjelmistoprojektien onnistuminen vaatii paljon muutakin, kuin yksittäisten sprinttien onnistumisen.

7.3.3 Uuden prosessin ohjelmistoversio ja päätelmät

Retrospektiivien perusteella uusi prosessi ei ole ratkaissut kaikkia tuotekehityksen ongelmia. Ongelmat vaatimusmäärittelyssä ovat yhä edelleen ongelmana. Tilanne on parantunut osittain työstöpalaverien avulla, mutta niitä ei ole pidetty tarpeeksi tai ne ovat olleet liian yleisellä tasolla, jotta ongelma olisi saatu ratkaistua laajemmin.

Työmääräarviot ja sprinttien suunnittelu ovat kuitenkin parantuneet huomattavasti uuden prosessin myötä. Työstöpalaverit ovat kuitenkin parantaneet ominaisuuksien

määrittelyjen laatua, jolloin myös työmääräarviot ovat tarkentuneet. Ei kuitenkaan ole tiedossa lisätäänkö työmääräarvioihin tarkoituksella varoaikaa mahdollisten muutosten vuoksi, vai ovatko kehitystiimin jäsenet parantuneet ominaisuuksien ymmärtämisessä ja arvioiden tekemisessä. Työmääräarvioiden alittamisestakaan ei ole tullut raportteja tai havaintoja, joten voidaan olettaa, että työmääräarviot ovat yleisesti ottaen tarkentuneet. Kohdeyrityksessä ei kuitenkaan mitata toteutuneita aikoja, joten tarkkaa tietoa tästä ei ole saatavilla. Sprinteille suunnitellut työt on saatu toteutettua lähes kokonaan. Vaikka sprinttien valmistuminen ei olekaan onnistunut aivan täydellisesti, on uusi prosessi parantanut tilannetta verrattuna entiseen.

Läpinäkyvyyden puute on myös yhä edelleen ongelma. Sprinttien tilanteesta saadaan tarkkaa tietoa burndown chartin -raportin perusteella, mutta tuotekehityksellä ei ole raporttia, joka kertoisi ominaisuuksien tilanteen. Burndown chart -raportti on tarkoitettu vain tiimin käyttöön, jotta he saavat tarvittavan informaation sprintin etenemisestä ja pystyvät tarvittaessa tekemään muutoksia. Sprinttikatselmus on tuotehallintaa ja muita ulkoisia sidosryhmiä varten, mutta koska osa ominaisuuksista ei ole vielä sprintin aikana valmis, katoaa läpinäkyvyys helposti. Tämä johtuu osittain liian suurista yksittäisistä ominaisuuksista, joita ei ehditä toteuttamaan yhden sprintin aikana. Työstöpalaverien aikana olisi tarkoitus pilkkoa liian suuret ominaisuudet pienemmiksi, jotta tältä ongelmalta vältyttäisiin.

Tuotekehityksen prosessin onnistumista on kuitenkin tarkasteltava laajemmassa mittakaavassa. Vuoden 2013 aikana kehitetyn ohjelmistoversion jälkeisessä julkaisutason retrospektiivissä havainnoitiin useita onnistumisia ja ongelmia. Suurimmat ongelmat olivat läpinäkyvyyden puute ja aikataulun viivästyminen. Aikataulun viivästymiselle löydettiin useita syitä, joista epäselvät vaatimusmäärittelyt ja versioprojektin sisällön muutokset olivat merkittävimpiä. Projektissa oli myös ongelmia resursoinnin kanssa, joka pahensi aikatauluongelmia.

Osa ominaisuuksien vaatimusmäärittelyistä oli kuitenkin tehty hyvin ja se havainnoitiin myös koko projektin tasolla positiivisena asiana. Suurin osa projektin sisällöstä saatiin toteutettua ajoissa, mutta testauksen viivästyminen aiheutti ohjelmistovirheiden löytymisen hyvin myöhään projektissa. Löydettyjen virheiden korjaaminen viivästytti osaltaan version stabilointia ja aikataulua. Myös asiakasprojektit ja ylläpito-

tehtävät veivät paljon projektin resurssien aikaa, jonka seurauksena projekti myöhästyi.

Uusi tuotekehitysprosessi ei ollut ratkaissut yrityksen tuotekehityksen ongelmia. Vaatimusmäärittelyt olivat parantuneet, mutta niissä oli yhä suuria ongelmia. Toteutettavat ominaisuudet olivat osin yhä liian suuria toteutettavaksi yhden sprintin aikana, jonka seurauksena testaus jäi jälkeen sprintille suunnitelluissa töissä. Ylläpitotyö vei yhä liikaa tuotekehityksen aikaa eikä sitä osattu ottaa tarpeeksi huomioon sprinttien suunnittelussa. Tuotekehityksen etenemiseen ei myöskään ollut tarpeeksi läpinäkyvyyttä, jotta ongelmat voitaisiin nähdä ja niihin voitaisiin puuttua aikaisemmin. Kaikki mainitut ongelmat olivat kuitenkin ratkaistavissa ilman prosessin muuttamista. Ongelmat olivat prosessin ulkoisia ongelmia, jotka prosessi toi hyvin esille. Scrum-menetelmästä sanotaankin, että se paljastaa organisaation ongelmat hyvin kivuliaalla tavalla (Schwaber 2011, 381).

2014 kesällä alkaneessa uudessa versioprojektissa päätettiin muutaman tuotekehitystiimin kanssa kokeilla erityyppiseen ketterään menetelmään perustuvaa tuotekehitysmallia. Näiden tiimien kehitysmalliksi valittiin Kanbaniin perustuva malli. Pääasiassa muutos päätettiin tehdä sen vuoksi, että muodostetut tiimit olivat liian pieniä (2-3 henkilöä) täysipainoisen Scrum-prosessin pyörittämiseen. Toinen syy kehitysmallin vaihtamiseen oli, että Kanbanissa ei olisi iteraatioita eikä niiden suunnittelupalavereja, jotka vaativat onnistuakseen hyvin määritellyt ominaisuudet, joiden työmäärä pystytään arvioimaan tarkasti.

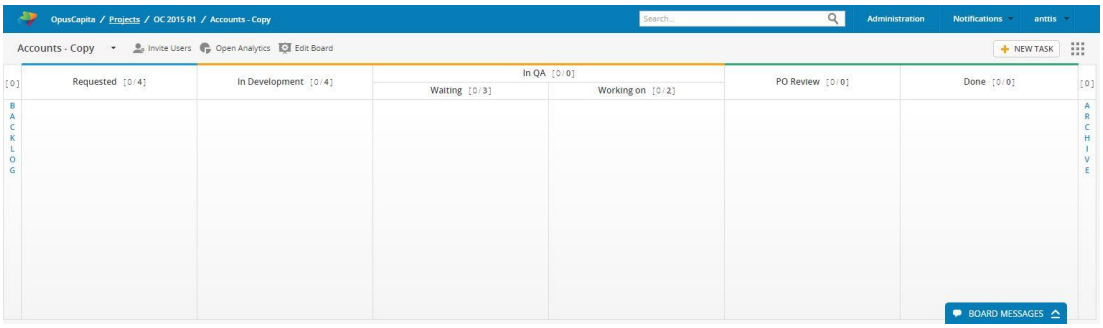
Ohjelmistoversio kehitettiin kokonaan käyttäen Kanbania ja seuraavassa luvussa tarkastellaan, miten kehitysmallin vaihtaminen vaikutti kohdeyrityksen tuotekehityksen toimintaan.

7.4 Kanban

Tuotekehityksen tiimejä järjestettiin uudelleen kesällä 2014. Tiimeistä muodostettiin aikaisempia pienempiä, jotka keskittyivät vain yhden tuotteen kehittämiseen. Osittain tiimien pienestä koosta johtuen neljä tiimiä aloitti uuden ohjelmistoversion kehityk-

sen käyttäen Kanbaniin perustuvaa kehitysmallia. Kanban valittiin myös siitä syystä, ettei tuotekehityksen töitä tarvinnut suunnitella ja lukita kokonaisen sprintin ajaksi. Ylläpitotyöt ja meneillään olevista asiakasprojekteista tuli kehitystiimeille työtä hyvinkin nopealla varoitusajalla. Tuotekehityksessä haluttiin myös nähdä toimisiko Kanbaniin perustuva malli tällaisessa ympäristössä Scrum-menetelmää paremmin.

Kohdeyrityksessä mallinnettiin ensin sen nykyinen prosessi Kanbaniin pohjautuvaksi tauluksi, jossa jokainen työvaihe oli kuvattu omana sarakkeenaan. Jokaiselle työvaiheelle oli asetettu kapasiteetin mukainen raja, kuinka monta keskeneräistä työtä kyseisessä työvaiheessa sai olla (WIP-raja). Kohdeyrityksen käyttöön hankittiin kolmannen osapuolen Kanban-työkalu nimeltä Kanbanize. Määritelty prosessi määriteltiin kolmannen osapuolen ohjelmiston avulla (kuva 3). Jokaisella kehitystiimillä oli työkalussa oma työtilansa, josta tiimi näki vain omalle tiimilleen kuuluvat työt. Jokaisen tiimin ”taulua” eli työvaiheita voitiin muokata erikseen.



Kuva 3. Kanbanize-työkalu ja prosessin eri työvaiheet

Koska yrityksen tuotekehityksen pääasiallinen toiminnanohjausjärjestelmä oli TFS, piti työn hallinnointi kahdessa eri järjestelmässä hoitaa jotenkin. Tehtiin pieni ohjelman, joka automatisoi uusien töiden lisäämisen Kanbanize-ohjelmistoon suoraan TFS:stä. Automatisoimalla prosessin työläin työvaihe, saavutettiin huomattavia parannuksia prosessin tehokkuudessa. Samalla varmistuttiin, että kaikki Kanbanizessa liikkuvat kortit oli määritelty samalla tavalla.

Kohdeyrityksen Kanbaniin pohjautuvassa prosessissa oli 5 eri vaihetta. Prosessi alkaa siitä, kun tuotepäällikkö asettaa ”Requested”-sarakkeeseen uuden työn. Kehitystiimi otti uutta työtä prosessiin ”requested”-sarakkeesta sitä mukaan, kun prosessista vapautuu kapasiteettiä. WIP-rajat määrittelevät prosessin kapasiteetin jokaisen työ-

vaiheen osalta. Jos kapasiteetti tulee jossain työvaiheessa täyteen, kaikki työ prosessin aikaisemmissa vaiheissa pysähtyy. Kyseisestä prosessin vaiheesta muodostuu prosessin pullonkaula. Järjestelmällisesti poistamalla kaikki muodostuvat pullonkaulat, tiimi pystyy asteittain tehostamaan prosessin läpimenoaikaa.

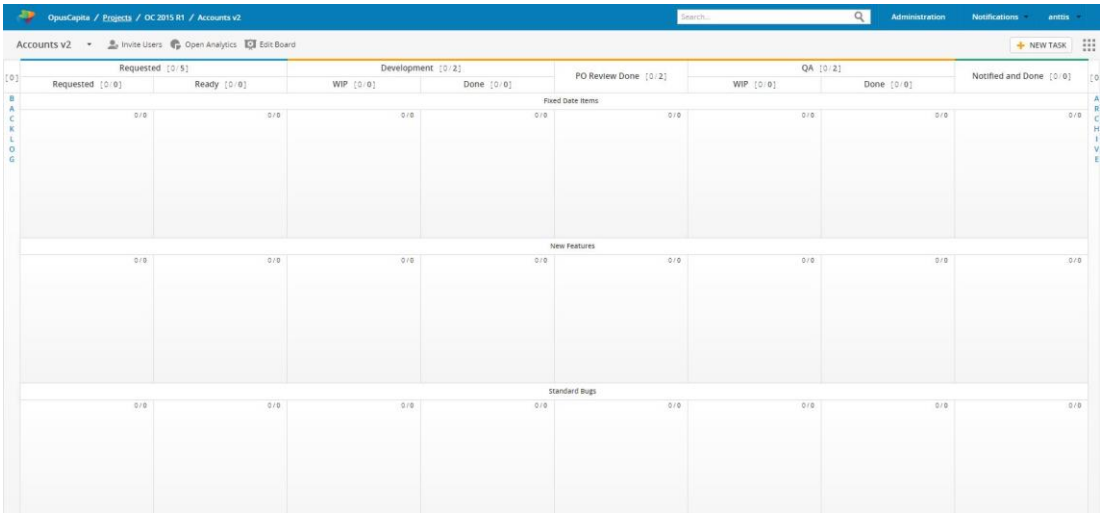
Ensimmäinen tuotekehityksen työvaihe on nimeltään ”In Development”. Tässä vaiheessa kehittäjä toteuttaa kyseisen ominaisuuden tai korjaa ohjelmistovirheen. ”Requested”-jono voi sisältää siis uusia ominaisuuksia tai ohjelmistovirheitä, jotka halutaan korjattavaksi. Kun ominaisuus on valmis, kehittäjä siirtää työn odottamaan testausta. Testauksen työvaiheet on jaettu kahteen eri vaiheeseen; ”Waiting” ja ”Working On”. Waiting-vaiheessa ovat kaikki testausta odottavat työt, josta testaaja vetää itselleen uutta työtä sitä mukaan, kun häneltä vapautuu kapasiteettia.

Testauksen jälkeen työ siirretään odottamaan tuotepäällikön hyväksyntää. Ohjelmistovirheiden korjaukset siirtyvät suoraan valmiiden töiden listaan sarakkeessa nimeltä ”Done”. Uudet ominaisuudet odottavat tuotepäällikön hyväksyntää ennen, kuin ne voidaan merkitä valmiiksi. Tuotepäällikön hyväksynnän työvaihe ”PO Review” ei ollut prosessissa alusta alkaen, vaan se lisättiin yhden kehitystiimin toiveesta loppusyksystä 2014.

Työ kulkee Kanban-prosessissa vasemmalta oikealle, eikä koskaan toiseen suuntaan. Tarkoituksena on keskittyä saamaan asioita valmiiksi sen sijaan, että aloitettaisiin uusia tehtäviä. Työvaiheille sovitut rajat oli sovittu hyvin väljiksi ja samalla henkilöllä sallittiin työtehtävistä riippuen olla 2-4 samanaikaista työtehtävää. Kanban-mallissa ei ole mitään ohjesääntöä rajalle, mutta yleisesti ottaen pienempi on aina parempi. Kohdeyrityksessä päädyttiin käyttämään suurempia rajoja useammasta eri syystä. Rajoja ei myöskään asetettu estäviksi, vaan ainoastaan informatiivisiksi.

Epäselvät vaatimusmäärittelyt aiheuttivat työn pysähtymisen ja vastauksia jouduttiin joskus odottamaan kauankin. Tällä aikaa pystyttiin edistämään jotain toista työtä, josta syystä samanaikaisen työn rajaa jouduttiin nostamaan. Nopeat ohjelmistovirheiden korjauspyynnöt ja muut ulkoiset seikat aiheuttivat myös työn pysähtymistä eri vaiheissa. Kanbaniin pohjautuvassa mallissa asiasta ei tullut ongelmaa, vaan tiimi pystyi edistämään sillä aikaa muita tehtäviä.

Kehitystiimit työskentelivät edellä esitetyn prosessin mukaisesti aina alkuvuoteen 2015 saakka, jolloin prosessia muutettiin yhdellä tiimillä (kuva 4). Samanaikaisen työn rajoista haluttiin ehdottomat, eikä niiden rikkomista enää sallittaisi. Prosessiin tehtiin muutenkin muutamia pieniä muutoksia, jolloin prosessista saatiin enemmän Kanbanin veto-mallin mukainen. Prosessin muutoksilla haluttiin myös puuttua siihen mennessä esiintyneisiin ongelmakohtiin, kuten puutteellisiin vaatimusmäärittelyihin ja läpinäkyvyyteen.



Kuva 4. Kanban-prosessin uudempi versio

Uudessa prosessissa Kanban-taulu jaettiin kolmeen eri kaistaan, työn laadusta riippuen. Ylimmäisenä kaistassa olivat määräajan sisältävät työt, jotka olivat usein asiakasongelmiin liittyvien ohjelmistovirheiden korjauksia. Toisessa kaistassa olivat uudet ominaisuudet ja viimeinen kaista oli varattu normaaleille ohjelmistovirheiden korjaukselle.

Jakamalla erityyppinen työ eri kaistoihin, voitiin prosessin kapasiteetti asettaa työn tyypin mukaisesti. Näin tuotekehityksen tiimeille voitiin asettaa tietty kapasiteetti ylläpitotyölle ja uusille ominaisuuksille. Kapasiteetin asettaminen tapahtui samanaikaisen työn rajan asettaminen kaistakohtaisesti. Määräajan sisältävät työt saivat ylittää rajan väliaikaisesti, mutta tällä kaistalla ei saanut milloinkaan olla enempää, kuin yksi työ kerrallaan.

Uudessa prosessissa ”Requested”-vaihe oli jaettu kahteen erilliseen vaiheeseen, joista jälkimmäisessä olivat kehitystiimin hyväksymät työt. Prosessiin tuleville töille haluttiin tarkastuspiste, jossa niiden vaatimusmäärittelyt voitiin tarkistaa, ennen työn aloittamista. Tuotepäällikkö lisäsi uusia töitä ensimmäiseen työvaiheeseen, mutta ne voitiin merkitä valmiiksi (Ready) vasta, kun kehitystiimi oli hyväksynyt niiden vaatimusmäärittelyn kattavuuden ja tarkkuuden. Tällä pyrittiin estämään mahdollisia ongelmia prosessin myöhemmissä vaiheissa.

Myös tuotekehityksen (Development) ja testauksen (QA) työvaiheet jaettiin kahteen erilliseen vaiheeseen; WIP (Work in progress) ja Done. Tällä päästiin eroon testauksen ”Waiting”-työvaiheesta ja prosessista tuli selkeämmin veto-mallinen, jossa työtä vedettiin prosessin seuraavaan vaiheeseen.

”PO Review”-työvaihe siirrettiin kehityksen ja testauksen väliin. Tällä haluttiin, että tuotepäällikkö katselmoi kehitetyn ominaisuuden mahdollisimman varhaisessa vaiheessa. Ei ole järkeä testata ja verifioida ominaisuutta, joka ei ole sellainen mitä halutaan. Prosessi etenee testaukseen vasta, kun tuotepäällikkö on hyväksynyt ominaisuuden toteutuksen. Ohjelmistovirheiden osalta prosessi ei sisällä tätä työvaihetta, vaan ohjelmistovirheiden käsittely etenee suoraan kehityksen ”Done”-sarakkeesta testauksen ”WIP”-vaiheeseen.

Viimeisimpänä muutoksena uuteen prosessiin lisättiin Notify-työvaihe, jonka tarkoituksena oli jakaa tietoa kehitettävistä asioista muulle organisaatiolle. Ohjelmistovirheiden kohdalla tämä tarkoitti pääasiassa asiakastukea, joka pystyi informoimaan asiakasta korjauksen saatavuudesta. Uudet ominaisuudet informoitiin tuotepäällikölle ja tarvittaessa myös muulle organisaatiolle.

Uuden prosessin muutokset eivät olleet suuria ja pääpiirteittäin Kanban-malli pysyi hyvin paljon aikaisemman kaltaisena. Vain yksi tuotekehitystiimi alkoi käyttää uudistettua Kanban-mallia. Uudessa mallissa huomattiin heti ongelmia WIP-rajojen kanssa. Rajoja jouduttiin kasvattamaan, jotta työstä ei tullut liian katkonaista. Vaatimusmäärittelyjen tasokaan ei noussut, koska tiimille ei ollut selvää millä perusteilla työ voitiin siirtää ”Ready”-sarakkeeseen. Kuten luvussa 4.5 on kuvattu, olisi ollut tärkeää muodostaa eksplisiittiset prosessikäytännöt, jolloin kaikille olisi ollut selvää

milloin työ voidaan siirtää prosessin vaiheesta toiseen. Nyt työtä siirtyi Ready-sarakkeeseen, vaikka sen vaatimusmäärittely ei ollut vaaditulla tasolla. Tämä johtui pääasiassa siitä, että prosessissa oli kapasiteettiä vetää lisää työtä, mutta Ready-sarakkeessa ei ollut valmiita asioita otettavaksi työn alle. Jotta ongelma ei pysäyttäisi tuotekehitystä, tiimi veti prosessiin asioita suoraan Requested-vaiheesta.

Kanban-pohjaisen mallin mukainen työskentely ei parantanut kohdeyrityksen perimmäisiä ongelmia, mutta se teki työskentelystä helpompaa ja vähemmän alttiimpaa niiden aiheuttamille ongelmille. Scrum-mallin aikarajoitetut sprintit ovat hyvin alttiita määrittelypuutteiden aiheuttamille ongelmille, kun taas Kanbanin jatkuvan työskentelyn malli ei epäonnistu heti, vaikka jokin työ jäisikin odottamaan tuotepäällikön panosta.

Molemmat kehitysmallit paljastivat yrityksen ongelmat. Kanbanissa WIP-rajojen riittämättömyys oli selkeä merkki siitä, että työ pysähtyi aivan liian usein. Työn pysähtymisen syy oli useimmiten määrittelyjen tarkennus tuotepäälliköltä. Myös Kanban-mallissa oli ongelmia läpinäkyvyyden kanssa. Tämä johtui pääasiassa liian suurista yksittäisistä töistä. Yksittäiset työt seisoivat liian kauan Kanban-taulun yhdessä työvaiheessa, jolloin läpinäkyvyys prosessin etenemiseen katosi. Kuten Scrum-menetelmässä, myös Kanbanissa työ tulisi pilkkoa tarpeeksi pieniksi, jotta prosessin edistyminen olisi jatkuvasti havaittavaa. Toisin, kuin Scrum-menetelmässä, yksittäisen työn ei tarvitse tuottaa lisäarvoa tuotteeseen, joten suuremman ominaisuuden pilkkominen voidaan tehdä helpommin ja suoraviivaisemmin.

Kehitystiimit kuitenkin pitivät Kanban-pohjaisesta mallista enemmän, kuin Scrum-menetelmästä. Työskentely oli sujuvampaa, eikä kehitystiimin aikaa kulutettu suunnittelupalavereihin tai muihin määriteltyihin tapahtumiin. Kanban-mallissa tämä kaikki tapahtui silloin, kun sille oli tarvetta ja vain sillä laajuudella, kuin oli tarve (Just-in-time). Käytännössä tämä tarkoitti vapaampaa työskentelyä sekä tuotehallinnalle että tuotekehitykselle.

Kohdeyrityksen tuotekehityksessä oli yritetty kahta erilaista prosessia; Scrum-menetelmää sekä Kanban-malliin pohjautuvaa prosessia. Molemmat prosessit paljas-

tivat samoja ongelmia kohdeyrityksen organisaatiossa. Kumpikaan prosesseista ei yksistään pystynyt ratkaisemaan kyseisiä ongelmia.

8 TUTKIMUKSEN ARVIOINTI

Tutkimuksellisen kehitystyön viimeisenä työvaiheena on tehdyn työn arviointi. Vaikka arviointia käytetään työn aikana ohjaamaan työn etenemistä, suunnitelmallinen arviointi tapahtuu vasta työn valmistuttua. Suunnitelmallisen arvioinnin avulla voidaan määriteltyihin kriteereihin vertaamalla arvioida kehitystyön vaikutuksia. Arviointi tapahtuu yleensä kehittämistyön panostuksen, muutosprosessin, sekä lopputuloksen perusteella. Arvioinnissa tarkastellaan usein kehittämistyön suunnittelua, tavoitteiden selkeyttä ja niiden saavuttamista. Myös kehittämistyön menetelmien, toiminnan johdonmukaisuuden, sekä vuorovaikutuksen onnistumista tulee voida arvioida. (Ojasalo, Moilanen & Ritalahti 2009, 47.)

Opinnäytetyön tarkoituksena oli kehittää kohdeyritykselle uusi tuotekehityksen prosessi, joka auttaisi ratkaisemaan tuotekehityksessä havaittuja ongelmia. Merkittävimmät ongelmat liittyivät puutteellisiin ja muuttuviin vaatimusmäärittelyihin, joista osittain seurasi iteraatiopohjaisen työn puuttuminen ja regressiovirheiden suuri määrä. Edellisten versioiden ylläpitotyö sekoitti uuden tuotteen kehittämistä, josta seurasi testauksen ylikuormittuminen ja aikatauluongelmat. Havaittujen ongelmien ratkaiseminen ajoissa oli hyvin vaikeaa ilman läpinäkyvyyttä tuotekehityksen edistymiseen. Aikatauluongelmien vuoksi ohjelmistoversiosta jouduttiin karsimaan ominaisuuksia, eikä kaikkia ominaisuuksia ehditty aina viimeistelemään versioaikataulun rajoissa.

Havainnointi oli opinnäytetyön tärkein tutkimusmenetelmä. Toimin koko kehittämistyön ajan yhden tai useamman kehitystiimin vetäjänä ja pystyin hyvin havainnoimaan kehitystiimien päivittäistä työskentelyä ja työn tuloksia. Toimin myös tiiviissä yhteistyössä muiden kehitystiimien vetäjien kanssa ja pystyin täten seuraamaan myös muiden tiimien työtä kuulemalla mahdollisista ongelmista ja onnistumisista. Muiden

tiimien kokemuksiin tulee kuitenkin suhtautua hyvin kriittisesti, koska niitä ei ole pystytty havainnoimaan omatoimisesti. Näistä havainnoista on kuitenkin hyötyä omien havaintojen validiteetin vahvistajana.

Havainnointi kehitystiimien sisällä sisältää omat riskinsä. Havainnoija on lähellä tekemistä ja havainnot saattavat vääristyä omakohtaisen kokemuksen kautta. Havainnoija toimii osana kehitystiimejä ja vaikuttaa niiden toimintaan. Tällainen aktiivinen havainnointi saattaa vääristää havainnoinnin tuloksia ja tästä syystä myös muiden tiimien ottaminen mukaan tutkimukseen oli ensisijaisen tärkeää. Näiden tiimien avulla saatiin vahvistusta muille havainnoille.

Lomaketutkimuksina toteutetut kyselytutkimukset antoivat myös yhtenäisiä tuloksia, joista oli selkeästi havaittavissa yhteiset teemat. Lomaketutkimuksien otos oli pieni, mutta suhteutettuna prosessin vaikutusjoukkoon, voidaan otosta pitää tarpeeksi laajana tarjoamaan luotettavia tuloksia. Ainoastaan tuotepäällikköjen heikko osallistuminen tai puuttuminen kokonaan vääristi osaltaan tulokset koskemaan ainoastaan tuotekehityksen henkilöstöä.

Havaintojen perusteella voidaan todeta, että kehitetty prosessi paransi tuotekehityksen toimintaa, mutta muutos ei ollut tarpeeksi merkittävä, jotta se olisi johtanut täysin onnistuneisiin versioprojekteihin. Kehitystiimien retrospektiivien tulokset eri vuosina olivat hämmästyttävän samankaltaisia. Sprintit onnistuvat suhteellisen hyvin ja lähes kaikki suunniteltu työ saatiin tehdyksi. Sprinttien onnistuminen ei kuitenkaan kerro koko totuutta ja versioprojekti myöhästyi edelleen suunnitellusta aikataulusta.

Sprinttien onnistuminen mitataan siitä, kuinka hyvin tiimi saa toteutettua sprintille suunnitellut työt. Suunnitelman tekemisessä käytetään tiimin omia työaika-arvioita. Tästä johtuen tiimin suorituskykyä arvioidaan tiimin omien arvioiden perusteella. Tällainen arviointi ei anna todellista kuvaa prosessin suorituskyvystä, eikä välttämättä johda onnistuneeseen projektiin. Tuotekehitysprosessin onnistumista tulisikin mitata tuotteeseen tuotetun lisäarvon perusteella. Arvon tuottamisen arviointi on kuitenkin hyvin hankalaa ja tästä syystä arvioinnin perusteena jouduttiin käyttämään olemassa olevia mittareita. Sprinttiä voidaan kuitenkin pitää myös onnistuneena, kun sen päätteeksi tiedetään tuotteen tilanne. Uuden tuotekehitysprosessin tarjoaman lä-

pinäkyvyyden ja sprinttikatselmuksien avulla, tuotteen tila tiedetään huomattavasti aikaisempaa tarkemmin.

Uskoisin, että tutkimuksen tulokset ovat hyvin yleistettävissä muihin tuotekehitysorganisaatioihin. Kohdeorganisaation ongelmat eivät ole ainutlaatuisia. Henkilöt, jotka ovat vastuussa vaatimusmäärittelyistä, eivät useinkaan ole tuotekehittäjiä eivätkä välttämättä pysty ymmärtämään millaisia tarpeita tuotekehityksellä on vaatimusten tason suhteen. Kun organisaatiossa ylläpidetään useita vanhoja versioita ja kehitetään samaan aikaan uusia ominaisuuksia, joudutaan väkisinkin tasapainoilemaan eri tehtävien välillä. Tästä tasapainoilusta johtuen päädyttiin siihen johtopäätökseen, että Kanban sopisi Scrum-menetelmää paremmin tuotekehityksen prosessin perustaksi.

Scrum-menetelmä tuli kuitenkin valittua tuotekehityksen menetelmäksi kohdeyrityksen halusta. Kohdeyritys oli vaihtelevalla menestyksellä. käyttänyt jo vuosia Scrum-menetelmään perustuvaa prosessia. Organisaation yleinen kanta oli, että ongelmat olivat Scrumin toteutuksessa, eivätkä itse Scrum-menetelmässä. Scrum oli valittu uuden tuotekehitysprosessin pohjaksi jo etukäteen, eikä muita vaihtoehtoja todella harkittu kehitystyön alkuvaiheessa. Kanbaniin perustuvan prosessin käyttöönotto tuli mukaan vasta myöhemmässä vaiheessa. Vaikka Kanbaniin perustuvassa prosessissa oli osittain samoja ongelmia, kuin Scrum-menetelmään perustuvassa prosessissa, voitaisiin Kanbanin avulla jatkossa saavuttaa Scrumia parempia tuloksia.

Kanban tarjoaa valmiit mekanismit erityyppisten töiden kapasiteetin hallintaan ja helpottaisi työn hallintaa jatkuvasti muuttuvassa ympäristössä. Scrum-menetelmä toimii paremmin puhtaasti uusien ominaisuuksien kehityksessä, kun tuotekehitys voi täysin keskittyä uuden tuotteen kehittämiseen ilman muita vastuualueita. Scrum vaatii kuitenkin suunnitelmallisuutta, jota kohdeorganisaation toimintaympäristö ei aina mahdollista.

Kehitystyötä voidaan pitää onnistuneena, koska kaikki kohdeorganisaation kehitystiimit käyttävät nyt samaa tuotekehitysprosessia. Prosessi tarjoaa hyvät mahdollisuudet onnistua projekteissa. Kohdatut ongelmat ovat prosessin käyttöönoton ja käytön ongelmia, eivätkä johdu itse prosessista. Kun kohdeorganisaatiossa opitaan toimi-

maan ketterien menetelmien mukaisesti, voidaan päästä eroon kohdatuista ongelmista.

9 JOHTOPÄÄTÖKSET JA JATKOKEHITYS

Sekä Kanban, että Scrum paljastivat kohdeyrityksessä samat ongelmat. Useimmat kohdatuista ongelmista olivat sellaisia, ettei niitä voida täysin korjata pelkällä prosessin määrittelyllä. Ratkaisut vaativat laajempia muutoksia eri organisaation roolien työskentelyssä. Jatkuvan työskentelyn Kanban-malli sallii enemmän häiriöitä ja on siksi mielestäni paremmin soveltuva kohdeyrityksen tuotekehityksen prosessiksi. Myös tuotekehitystiimien jäsenet pitivät Kanban-mallin mukaisesta työskentelystä enemmän.

Kohdeyrityksen tuotekehityksen suurin ongelma on yrityksen ominaisuuksien vaatimusmäärittely. Vaatimusmäärittelyt ovat liian puutteellisia tai niitä muutetaan kesken kehityksen. Molemmat aiheuttavat ongelmia prosessin myöhemmissä vaiheissa. Scrum-mallin työstöpalaverit ja Kanbanin Ready-sarake auttavat tilannetta, mutta eivät korjaa kaikkia ongelmia. Kanban-mallissa määrittelyjen puutteellisuus ei aiheuta yhtä suuria ongelmia, kuin Scrum-menetelmässä.

Ainoa keino ratkaista vaatimusmäärittelyjen ongelma, on allokoida tuotehallinnalle enemmän aikaa tehdä yhteistyötä tuotekehityksen kanssa. Jos tuotepäällikkö toimisi täysipäiväisenä tiimin jäsenenä, tilannetta saataisiin parannettua. Vaatimusmäärittelyssä tulisi varmasti yhä olemaan puutteita, koska täydellistä määrittelyä ei ole olemassakaan, mutta jatkuvasti läsnä oleva tuotepäällikkö olisi aina valmiina vastaamassa kysymyksiin ja työ voisi jatkua ilman pidempiä keskeytyksiä. Lähempänä tuotekehitystä toimiva tuotepäällikkö pysyisi jatkuvasti ajan tasalla kehityksen tilanteesta ja voisi paremmin ohjata tuotteensa kehitystä oikeaan suuntaan. Ketterissä menetelmissä puhutaan termistä tuoteomistaja. Kohdeorganisaatiossa ei ole henkilöitä tällaisessa roolissa, vaan ainoastaan perinteisiä tuotepäälliköitä, jotka toimivat enemmän yrityksen asiakasrajapinnassa, kuin tuotekehityksen kanssa.

Molemmat prosessimallit kärsivät iteraatiopohjaisen työn puuttumisesta. Scrum-menetelmä vaatii, että jokaisen iteraation tulisi tuottaa tuotteeseen lisäarvoa, jolloin laajempien ominaisuuksien pilkkominen pienempiin osiin on haastavampaa. Kanbanissa pilkkominen voidaan tehdä suoraviivaisemmin esimerkiksi teknisten toteutusten pohjalta (tietokanta, käyttöliittymä, jne..). Toteuttamalla pienempiä kokonaisuuksia, myös prosessin läpinäkyvyys parantuisi. Eteneminen olisi jatkuvasti havaittavaa, jolloin myös työn etenemisen ennustaminen parantuisi. Paremmalla ennustamisella olisi suora positiivinen vaikutus yrityksen projektien aikataulujen pitämiseen.

Testauksen ylikuormittuminen ei korjaantunut kummassakaan prosessissa. Ketterät menetelmät eivät toimi kovinkaan hyvin, jos tuotekehityksen prosessi vaatii erilliset ja peräkkäiset työvaiheet kehitykselle ja testaukselle. Kohdeyrityksen testauksen tulisi lisätä huomattavasti enemmän automatisoitua testausta, kuten yksikkötestausta. Näin ominaisuuksien verifiointi voitaisiin tehdä automatisoidusti ja koko tuotteen integraatiotestaus voitaisiin toteuttaa manuaalisesti testaamalla. Eritasoisen testauksen automatisointi parantaisi myös regressiovirheiden havainnointia ja parhaimmassa tapauksessa estäisi kokonaan niiden muodostumisen.

Scrum-menetelmä sopii parhaiten uusien tuotteiden kehittämiseen. Kohdeyrityksen suuri ylläpitotyön määrä ei sovi Scrum-menetelmän rakenteeseen. Kanbanin avulla ylläpitotyö saadaan luontevaksi osaksi prosessia, jolle voidaan määritellä myös erillinen kapasiteetti. Tämä ei ole täysin mahdollista Scrum-menetelmän kanssa, jossa sprintin aikana tehtävä työ pitäisi olla suunniteltavissa etukäteen.

Kohdeorganisaatiossa ollaan tällä hetkellä siirtymässä takaisin Scrum-menetelmän mukaiseen prosessiin, jokaisen tuotekehitystiimin osalta. Prosessia tullaan kehittämään jatkuvasti eteenpäin työryhmässä, jossa ovat mukana kaikki organisaation projektipäälliköt ja Scrum Masterit. Organisaation ongelmat ovat tiedossa ja niihin pyritään löytämään kaikkia sidosryhmiä tyydyttäviä ratkaisuja. Uuden version kehitysprojektin on tarkoitus alkaa kesällä 2015, jonka aikana organisaatiossa tullaan näkemään miten sillä kertaa toteutetut uudistukset onnistuvat parantamaan tilannetta.

LÄHTEET

Agile Manifeston www-sivut. 2013. Viitattu 29.5.2013. http://agilemanifesto.org.

Alasuutari, P. 1993. Laadullinen tutkimus. Jyväskylä:Osuuskunta Vastapaino. Viitattu 21.2.2015

Anderson, D. J. 2010. Kanban - Successful Evolutionary Change for Your Technology Business. Sequim, WA: Blue Hole Press. Viitattu 22.1.2015.

Arikoski, J. & Sallinen, M. 2007. Vastarinnasta vastarannalle – johda muutos taitavasti. Helsinki : Työterveyslaitos. Viitattu 26.5.2013.

Armson, R. 2011. Growing Wings on the Way : Systems Thinking for Messy Situations. Axminster, Devon, GBR: Triarchy Press. Viitattu 7.2.2015. http://www.ebrary.com

Beyer, H., 2010. Synthesis Lectures on Human-Centered Informatics : User-Centered Agile Methods. San Rafael, CA, USA: Morgan & Claypool Publishers. Viitattu 12.5.2013.

Blankenship, E., Woodward, M. & Holliday, G. 2011. Professional Team Foundation Server 2010. Hoboken : Wrox. Viitattu 29.5.2013.

Boonstra, J. J. 2004. Dynamics of Organizational Change and Learning. Hoboken : Wiley. Viitattu 28.5.2013.

Chemuturi, M. & Cagley , T. M., 2010. Mastering Software Project Management : Best Practices, Tools and Techniques. Ft. Lauderdale, FL, USA: J. Ross Publishing Inc.. Viitattu 15.4.2013.

Cobb, C. 2011. Making Sense of Agile Project Management : Balancing Control and Agility. Hoboken : Wiley. Viitattu 29.5.2013.

Cooke, J. 2010. Agile Principles Unleashed : Proven Approaches for Achieving Real Productivity Gains in Any Organisation. Cambs : IT Governance. Viitattu 29.5.2013.

Cooke, J. L. 2012. Everything you want to know about Agile : How to get Agile results in a Less-than-Agile Organization. Cambridge, GBR: IT Governance. Viitattu 11.1.2015. http://www.ebrary.com

Drucker, P.F. 1999. Johtamisen haasteet. Helsinki : WSOY. Viitattu 26.5.2013.

Flanigan, E., & Scott, J. 1995. Process Improvement : Enhancing Your Organization's Effectiveness. Boston, MA, USA: Course Technology Crisp. Viitattu 5.2.2015. http://www.ebrary.com

Genero, M., Piattini, M. & Calero, C., 2005. Metrics for Software Conceptual Models. London, GBR: Imperial College Press. Viitattu 9.5.2013.

Goodpasture, J. C. 2010. Project Management the Agile Way : Making It Work in the Enterprise. Ft. Lauderdale, FL, USA: J. Ross Publishing Inc.. Viitattu 19.11.2014. http://www.ebrary.com

Gross, J. M., & McInnis, K. R. 2003. Kanban Made Simple : Demystifying and Applying Toyota's Legendary Manufacturing Process. New York, NY, USA: Amacom. Viitattu 22.1.2015. http://www.ebrary.com

Hamel, G. & Breen, B. 2007. The Future of Management. Harvard Business School Press. Viitattu 28.5.2013.

Harvard Business School Press. 2010. Improving Business Processes (Pocket Mentor). Harvard Business Review Press. Kindle Edition. Viitattu 10.2.2015.

Holcombe, M. 2008. Running an Agile Software Development Project. Hoboken NJ USA : Wiley. Viitattu 18.5.2013.

Holmström, B. & Tirole, J. 2011. Inside and Outside Liquidity. Cambridge : MIT Press. Viitattu 29.5.2013.

Horch, J. 2003. Practical Guide to Software Quality Management (Second Edition). Norwood, MA, USA: Artech House. Viitattu 24.11.2014. http://www.ebrary.com

Hundhausen, R. 2012. Professional Scrum Development with Visual Studio 2012. Microsoft Press. Viitattu 16.2.2014.

Hunt, R., & Killen, C. P.. 2008. International Journal of Quality and Reliability Management, Volume 25, Number 1 : Best Practice Project Portfolio Management. Bradford, GBR: Emerald Group Publishing Ltd. Viitattu 25.1.2015. http://www.ebrary.com

Itellan Pörssitiedote 23.6.2011. Viitattu 29.4.2013. http://www.itella.fi/group/tiedotteet/2011/20110623_itella_ostaa_opuscapita_groupin.html

Koch, A. 2004. Agile Software Development : Evaluating the Methods for Your Organization. Norwood : Artech House. Viitattu 29.5.2013.

Lehtimäki, T. 2006. Ohjelmistoprojektit käytännössä. Jyväskylä : Gummerus Kirjapaino Oy. Viitattu 18.5.2013.

Mohapatra, P. K. J. 2010. Software Engineering. Daryaganj : New Age International. Viitattu 29.5.2013.

MSDN keskustelupalstan www-sivut. 2013. Viitattu 29.5.2013. https://social.msdn.microsoft.com

MSDN:n www-sivut. 2013. Viitattu 29.5.2013. https://msdn.microsoft.com

Murthy, C.S.V. 2007. Change Management. Mumbai, IND: Global Media. Viitattu 12.5.2013.

Newell, M. W. & Grashina, M. N. 2003. Project Management Question and Answer Book. Saranac Lake, NY, USA: Amacom. Viitattu 17.11.2014. http://www.ebrary.com

Ojasalo, K., Moilanen, T. & Ritalahti, J., 2010. Kehittämistyön menetelmät. Helsinki: WSOYPro Oy. Viitattu 12.5.2013.

OpusCapitan www-sivut. 2013. Viitattu 29.4.2013. http://www.opuscapita.fi/media/40049/opuscapita_fast_facts_fi.pdf

Page, S. 2010. Power of Business-Process Improvement : 10 Simple Steps to Increase Effectiveness, Efficiency, and Adaptibility. Saranac Lake, NY, USA: Amacom. Viitattu 2.2.2015. http://www.ebrary.com

Resnick, S., de la Maza, M. & Bjork, A. 2011. Professional Scrum with Team Foundation Server 2010. Hoboken : Wrox. Viitattu 29.5.2013.

Rico, D., Sayani, H. & Sone, S. 2009. Business Value of Agile Software Methods : Maximizing ROI with Just-In-Time Processes and Documentation. Ft. Lauderdale : J. Ross Publishing Inc. Viitattu 29.5.2013.

Rinzler, B. 2009. Telling Stories : A Short Path to Writing Better Software Requirements. Hoboken, NJ, USA: John Wiley & Sons. Viitattu 23.11.2014. http://www.ebrary.com

Saleh, K. A. 2009. Software Engineering. Ft. Lauderdale : J. Ross Publishing Inc. Viitattu 29.5.2013.

Schaeffer, M. 2004. Accounts Payable : A Guide to Running an Efficient Department. Hoboken : Wiley. Viitattu 29.5.2013.

Schuh, P. 2004. Integrating Agile Development in the Real World. Hingham, MA, USA : Charles River Media / Cengage Learning. Viitattu 19.5.2013.

Schwaber, K., Kubacki, K. & Sutherland, J. 2012. Software in 30 Days : How Agile Managers Beat the Odds, Delight Their Customers, and Leave Competitors in the Dust. Hoboken, NJ, USA : Wiley. Viitattu 19.5.2013.

Schwaber, K. 2004. Agile Project Management with Scrum. Microsoft Press. Viitattu 19.5.2013.

Schwaber, K. 2007. The Enterprise and Scrum. Microsoft Press. Viitattu 20.5.2013.

Schwaber, K., Kubacki, R., & Sutherland, J. 2012. Software in 30 Days : How Agile Managers Beat the Odds, Delight Their Customers, and Leave Competitors in the Dust. Hoboken, NJ, USA: John Wiley & Sons. Viitattu 10.12.2014. http://www.ebrary.com

Somasundaram, Ravishankar. 2013. Git: Version Control for Everyone. Birmingham: Packt Publishing. Viitattu 14.8.2014.

Testa, Louis. 2009. Growing Software : Proven Strategies for Managing Software Engineers. San Fransisco: No Starch Press. Viitattu 14.8.2014.

Viscardi, S. 2013. Professional ScrumMaster's Handbook. Olton, Birmingham, GBR: Packt Publishing. Viitattu 23.11.2014. http://www.ebrary.com

Windley, P. 2001 – 2002. Delivering High Availability Services Using a Multi-Tiered Support Model. Utah.gov. Viitattu 3.12.2014. http://www.windley.com/docs/Tiered%20Support.pdf