Markus Ahonen

Tapaustutkimus: Soveltuuko Scrum vesiputousmallin korvaajaksi yrityksen sovelluskehitysprojekteihin?

Elektroniikan, tietoliikenteen ja automaation tiedekunta

Automaatio- ja systeemitekniikan laitos

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 3.5.2010.

Työn valvoja:

Prof. Pirkko Oittinen

Työn ohjaaja:

DI Visa Laasonen

Aalto-yliopisto
Teknillinen korkeakoulu

AALTO-YLIOPISTO
TEKNILLINEN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Markus Ahonen

Työn nimi: Tapaustutkimus: Soveltuuko Scrum vesiputousmallin korvaajaksi yrityksen sovelluskehitysprojekteihin?

Päivämäärä: 3.5.2010 Kieli: Suomi Sivumäärä: 7+68

Elektroniikan, tietoliikenteen ja automaation tiedekunta

Automaatio- ja systeemitekniikan laitos

Professuuri: Viestintätekniikka Koodi: AS-75

Valvoja: Prof. Pirkko Oittinen

Ohjaaja: DI Visa Laasonen

Tässä tapaustutkimuksessa tutkittiin Scrum-prosessimallin soveltuvuutta kohdeyrityksen ulkoistettujen sovelluskehitysprojektien hallintaan. Käytännön kokemuksia mallin sopivuudesta kerättiin Scrum-menetelmää kokeilevassa pilottiprojektissa, jossa tutkija oli osa kehitystiimiä. Kehitystiimin ja tutkijan henkilökohtaisia kokemuksia ja havaintoja kerättiin kehitystyön rinnalla hieman yli vuoden ajan. Kokemuksia kerättiin myös projektin aikana suoritetuista haastatteluista, joihin osallistui yksitoista projektin avainhenkilöä. Aikaisempaa tutkimusta ketteriin menetelmiin liittyvistä kokemuksista etsittiin lisäksi Internetin artikkelikokoelmista.

Aiemmista tutkimusraporteista Scrum-kokemuksista löydettiin yhteyksiä kokemuksiin pilottiprojektissa. Scrum ei ollut pilottiprojektin kannalta pelkkää menestystä, mutta yritys koki päässeensä silti parempaan lopputulokseen kuin vesiputousmallilla olisi ollut mahdollista. Scrumilla koettiin olleen positiivinen vaikutus projektin sisäiseen viestintään.

Avainsanat:
Scrum, ketterät menetelmät, vesiputousmalli, ohjelmistokehityksen prosessimallit, tapaustutkimus

AALTO-UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY

ABSTRACT OF THE
MASTER'S THESIS

Author: Markus Ahonen

Subject of the thesis: Case-study: How does Scrum succeed in replacing waterfall model in software development projects of an organization?

Date: 5/3/2010 Language: Finnish Number of pages: 7+68

Faculty of Electronics, Communications and Automation

Department of Automation and Systems Technology

Professorship: Media Technology Code of professorship: AS-75

Supervisor: Prof. Pirkko Oittinen

Instructor: M.Sc. Visa Laasonen

This case-study was conducted to research the suitability of Scrum process model in managing outsourced software development projects in a target company. Experiences were collected from a pilot project, in which the researcher belonged to the development team. The experiences and observations of the team and the researcher were collected for a little over a year alongside with the development work. Experiences were also collected by interviewing eleven key persons associated with the project. Earlier studies related to experiences from agile methods were also searched from article databases on the Internet.

Earlier reports of Scrum experiences revealed similarities and connections to experiences from the pilot project. When reviewing the results of the pilot project, Scrum-process was successful, but not perfect. According to the company's experiences however, the end result was better than it would have been if waterfall model was used. It was also experienced that Scrum had a positive effect to internal communication in the project.

Keywords:
Scrum, agile methods, waterfall model, software development process, case-study

Esipuhe

Tämä tutkimus ei olisi valmistunut ilman HiQ Finlandin tarjoamaa mahdollisuutta suorittaa diplomityö työn ohessa, kiitokset siitä. Suuri osa näistä kiitoksista kuuluu myös asiakasyritykselle, jolle tämä työ tehtiin toimeksiantona. Kiitän Petriä ja Visaa sekä professori Pirkko Oittista rakentavasta palautteesta. Opiskelun ja työn yhdistäminen on toki ollut ajoittain hankalaa ja väsyttävää, joten kiitos vaimolleni Katariinalle kärsivällisyydestä ja tuesta tämän pitkän urakan aikana.

Haluan kiittää myös kaikkia kollegoitani, erityisesti Mikkoa ja molempia Jukkia, vinkeistä ja avusta. Kiitos kaikille tähän projektiin osallistuneille, on ollut hienoa työskennellä kanssanne!

Markus Ahonen

Espoossa, 3.5.2010

Sisällysluettelo

DIPLOMITYÖN TIIVISTELMÄ II
ABSTRACT OF THE MASTER'S THESIS III
ESIPUHE IV
SISÄLLYSLUETTELO V
INDEKSI KUVISTA VII
1 JOHDANTO 1
2 PROSESSIMALLIT SOVELLUSKEHITYKSESSÄ 3
2.1 Vesiputousmalli 4
2.2 Inkrementaalinen kehitys 5
2.3 Ketterät menetelmät 6
2.4 Scrum 9
2.4.1 Scrumin vaiheet 9
2.4.2 Roolit 11
2.4.3 Product backlog 12
2.4.4 Sprintit 15
2.5 XP 18
2.6 Aiempia kokemuksia Scrumista ja XP:stä 19
2.7 Yhteenveto prosessimalleista 23
3 CASE-YRITYKSEN VANHA SOVELLUSKEHITYSPROSESSI 26
3.1 Motivaatio uuteen prosessimalliin 30
3.2 Siirtyminen uuteen prosessimalliin 32
4 CASE-YRITYKSEN UUSI PROSESSIMALLI – SCRUM KÄYTÄNNÖSSÄ 36
4.1 Tutkimuksen toteutus 36
4.2 Haastatellut henkilöt 37
4.3 Henkilöiden kokemukset prosessista 38
4.3.1 Aiemmat toimintamallit 38
4.3.2 Scrumista saadut kokemukset 39
4.3.3 Viestintä ja suhteet 48
5 TULOSTEN TARKASTELU 50
6 JOHTOPÄÄTÖKSET 52
6.1 Miten Scrum toimi case-projektissa? 52

6.2 Scrumin vaikutukset viestintään 53
6.3 Kehityskohteiden analysointi 54
VIITTEET 61
LIITE A HAASTATTELUIDEN TUKIMATERIAALI 64

Indeksi kuvista

Kuva 1. Vesiputousmalli 4
Kuva 2. Esimerkki inkrementaalisesta mallista 6
Kuva 3. Scrum-prosessin vaiheet 9
Kuva 4. Scrumin eri roolit 11
Kuva 5. Käyttäjätarinoiden luokittelu kustannusten ja arvon mukaan 14
Kuva 6. Esimerkki sprintin burndown-graafista 16
Kuva 7. Esimerkki product backlogin burndown-graafista 17
Kuva 8. BPMN-kaavio vanhasta prosessimallista, osa 1/3 27
Kuva 9. BPMN-kaavio vanhasta prosessimallista, osa 2/3 28
Kuva 10. BPMN-kaavio vanhasta prosessimallista, osa 3/3 29
Kuva 11. BPMN-kaavio case-projektin etenemisestä, osa 1/2 34
Kuva 12. BPMN-kaavio case-projektin etenemisestä, osa 2/2 35
Kuva 13. “There is no cookbook for Scrum.”. 53
Kuva 14. BPMN-kaavio tulevaisuuden prosessimallista, osa 1/3 57
Kuva 15. BPMN-kaavio tulevaisuuden prosessimallista, osa 2/3 58
Kuva 16. BPMN-kaavio tulevaisuuden prosessimallista, osa 3/3 59
Kuva 17. Tavoitteellinen ajankäyttö tulevaisuuden sovelluskehitysprosessissa 60

1 Johdanto

Tämä tutkimus keskittyy analysoimaan ketterien menetelmien, erityisesti Scrumin, sopivuutta erään yrityksen sovelluskehitysprojektien toteutusmalliksi. Kyseisessä yrityksessä ei ole omaa sovelluskehitysosastoa, vaan kehitystyö on suoritettu projektiluontoisesti ulkoisten toimittajien avulla. Tällä tavalla yrityksessä on toteutettu lukuisia sovelluskehitysprojekteja ja kokeiltu erilaisia sovelluskehitysmenetelmiä. Toteutettujen projektien joukossa on ollut sekä useita onnistumisia että useita epäonnistumisia. Yleisimmin käytetty prosessimalli on muistuttanut perinteistä vesiputousmallia.

Yrityksen tavoitteena oli selvittää, voisiko Scrumia käyttämällä saada etuja vesiputousmalliin verrattuna. Keskeisinä tavoitteina yrityksellä olivat sekä kehitysprosessien tehostaminen ja sitä kautta saatavat kustannussäästöt että sovellusten käyttäjäkokemuksen ja laadun parantaminen.

Tutkimuskysymykset

1. Mitä Scrumin tai muiden ketterien menetelmien käytännön soveltamisesta ja kokemuksista kerrotaan kirjallisuudessa ja artikkeleissa?
   a) Löytyykö raportoiduista kokemuksista yhtäläisyyksiä case-projektiin?

2. Miten Scrum toimi case-projektissa?
   a) Miten projektiin osallistuneet tahot suhtautuivat Scrumiin?
   b) Mitkä ovat suurimmat erot verrattuna vanhaan prosessimalliin?
   c) Mitkä asiat osoittautuivat ongelmallisiksi?

3. Millaiseksi asiakkaan ja toimittajan välinen suhde ja viestintä koettiin Scrum-projektissa?

4. Mihin asioihin pitäisi kiinnittää huomiota tulevissa ketteriä menetelmiä käyttävissä projekteissa?

Tutkimuksen rajaukset

Tutkimuksessa keskitytään niihin prosessimalleihin, jotka kohdeyrityksessä ovat olleet tai ovat nyt käytössä, eli Scrumiin ja vesiputousmalliin. Muut prosessimallit ovat toissijaisia, joten niitä käsitellään suppeammin. Myös ohjelmistoprosessien parantamismenetelmät (*Software Process Improvement, SPI*) on rajattu tämän tutkimuksen ulkopuolelle.

Tutkimusmetodit

Ennen tutkimuksen aloittamista pidetyt keskustelut yrityksen sovelluskehityspäällikön kanssa toimivat ponnahduslautana tutkimuksen aloittamiselle ja antoivat suuntaa tutkimuskysymyksille. Projektin aikana käydyissä kahdenkeskisissä keskusteluissa käytiin läpi muun muassa yrityksen historiaa ja kokemuksia aiemmista sovelluskehitysprojekteista. Keskusteluiden avulla selvitettiin yrityksen motivaatiota prosessin kehittämiseen sekä kartoitettiin tavoitteita tulevalle prosessimallille.

Kirjallisesta aineistosta haettiin vastauksia erityisesti tutkimuskysymyksiin 1 ja 4, sekä myös kysymykseen 3. Kokemuksia käytännön soveltamisesta ja yhtäläisyyksistä case-projektiin etsittiin erityisesti tutkimusraporteista ja artikkeleista.

Noin kuusi kuukautta projektin toteutusvaiheen aloittamisen jälkeen projektiin osallistuvia avainhenkilöitä haastateltiin tavoitteena selvittää tutkimuskysymyksiin 2, 3 ja 4 liittyviä asioita.

Aikaisempi tutkimus

Sovelluskehityksen prosessimalleja yleisesti käsittelevää aineistoa on saatavilla runsaasti sekä tutkimusraporteista ja artikkeleista että kirjallisuudesta. Prosessimalleja vertailevaa tutkimusta löytyi suppeammin. Tutkimuksia ja artikkeleita, jotka käsittelevät ketteriä menetelmiä, löytyi Internetin hakujen avulla melko paljon. Haasteena oli tunnistaa löydöksistä relevantit tutkimusraportit, joissa aihetta käsiteltiin tätä tutkimusta hyödyttävällä tavalla.

Tutkimusraportin rakenne

Tämän tutkimusraportin ensimmäisissä kappaleissa käsitellään ensin sovelluskehityksen prosessimallien teoriaa ja selvitetään aiempia kokemuksia ketteristä menetelmistä. Kappaleesta 3 eteenpäin käsitellään tapaustutkimuksen kohteena olleen yrityksen sovelluskehitysprosessia ja tutkimuksen tuloksia.

Case-projekti

Selvitystyö toteutettiin käytännössä uuden web-sovelluksen kehitysprojektissa, johon kohdeyritys oli palkannut ulkopuolisen toimittajan. Tutkimuksen tekijä oli osa projektiin määrättyä kehitystiimiä. Projektin määrittelyvaihe oli toteutettu vesiputousmallin mukaisesti, mutta prosessimalliksi projektin toteutusvaiheeseen oli valittu Scrum. Projektin etenemistä seurattiin aitiopaikalta ja tutkimusaineistoa ja kokemuksia kerättiin sovelluskehityksen ohessa. Tutkimuksen käytännön osassa halutaankin tuoda esille myös henkilökohtaisia kokemuksia, mutta samalla pyritään tarkastelemaan tuloksia objektiivisesti.

2 Prosessimallit sovelluskehityksessä

Tietojärjestelmät ovat keskeinen osa nyky-yhteiskuntaa. Uusia ohjelmistoja kehitetään erilaisiin tarpeisiin jatkuvasti. Monia sovelluksia on kehitetty ja luultavasti kehitetään jatkossakin niin kutsutulla koodaa-ja-korjaa-menetelmällä, eli ilman selkeää prosessia, jolloin hukataan yleensä resursseja. Erityisesti monimutkaisempien ohjelmistojen kehityksen ja suurempien kehitystiimien tueksi tarvitaan hyvin määritelty prosessi. Sen avulla sovelluskehitysprojekti voidaan pitää hallinnassa ja päästä helpommin oikeaan lopputulokseen, eli sovellukseen joka toimii ja myös sisältää halutun toiminnallisuuden.

Sommervillen mukaan ohjelmistokehitysprosessi on joukko toimenpiteitä, jotka johtavat ohjelmistotuotteen valmistumiseen. Hän toteaa myös, että ohjelmisto voidaan kehittää tyhjästä, tai kuten on nykyään yleisemmin tapana, hyväksikäyttäen muita valmiita tuotteita ja komponentteja. Ohjelmistokehityksen prosessimallit ovat abstrakteja malleja, jotka pyrkivät kertomaan kuinka prosessia sovelletaan käytännössä. Tällaisia yleiskäyttöisiä, abstrakteja malleja voidaan laajentaa ja sovittaa muodostamaan jokin tiettyyn ympäristöön ja tarkoitukseen soveltuva prosessimalli. [27]

Perinteiset, niin kutsutut raskaat sovelluskehityksen prosessimallit olettavat, että vaatimukset voidaan aina määritellä etukäteen, eivätkä ne muutu kehityksen aikana. Lisäksi ne olettavat, että käyttäjät tietävät täsmälleen mitä haluavat ennen kuin he ovat nähneet lopputuloksen, ja että ohjelmistokehitys on helposti ennustettava ja toistettavissa oleva prosessi. Tämän tuloksena 31 % ohjelmistoprojekteista, joista suurimmassa osassa käytetään vesiputousmallia tai sen varianttia, keskeytetään ennen valmistumistaan [8]. Lisäksi nopealla tahdilla uudistuvat työkalut ja teknologiat tekevät toteutusstrategiat huonosti ennustettaviksi. [29]

Perinteisissä menetelmissä, joissa sovellukseen tulevia ominaisuuksia ei välttämättä ole edes priorisoitu, toteutetaan usein ensin teknisesti helpoimmat toiminnot. Näin projektin loppuvaiheessa voi olla jäljellä vain vaikeita, suuririskisiä toimintoja, jotka kuitenkin usein ovat asiakkaalle niitä kaikista tärkeimpiä. [18]

Ketterissä menetelmissä toistuvat teemat pyrkivät taklaamaan edellä mainittuja ongelmia: halutut ominaisuudet priorisoidaan, ja hyväksytään, että niihin voi tulla kehityksen aikana muutoksia. Kokonaisuus jaetaan pienempiin osatoimituksiin, joten ohjelmistokehityksen elinkaari käydään läpi useita kertoja. Näin prosessia voidaan kehittää matkan varrella ja voidaan varautua esimerkiksi usein perinteisissä menetelmissä vasta käyttöönottovaiheessa tuleviin yllättäviin ongelmiin. Myös laadunvarmistusta ja testausta suoritetaan koko projektin ajan, eikä vain sen lopussa. [18]

Budjetti, aikataulu ja sovelluksen ominaisuudet muodostavat sovelluskehitysprojektin kolme tärkeintä rajaa. Ihanteellinen tilanne sallii näiden rajojen joustoa eri suuntiin. Jos jokin raja, esimerkiksi aikataulu, halutaan kuitenkin kiinnittää, pitäisi muissa rajoissa pystyä joustamaan enemmän. Jokainen kiinnitetty raja-aita tekee kehitysprojektista vaativamman hallita, ja sitä enemmän kiinnittämättömissä osa-alueissa tulee olla liikkumavaraa. Neljäs, usein tässä yhteydessä unohdettu tai näkymätön, raja-aita on laatu. Sovel-

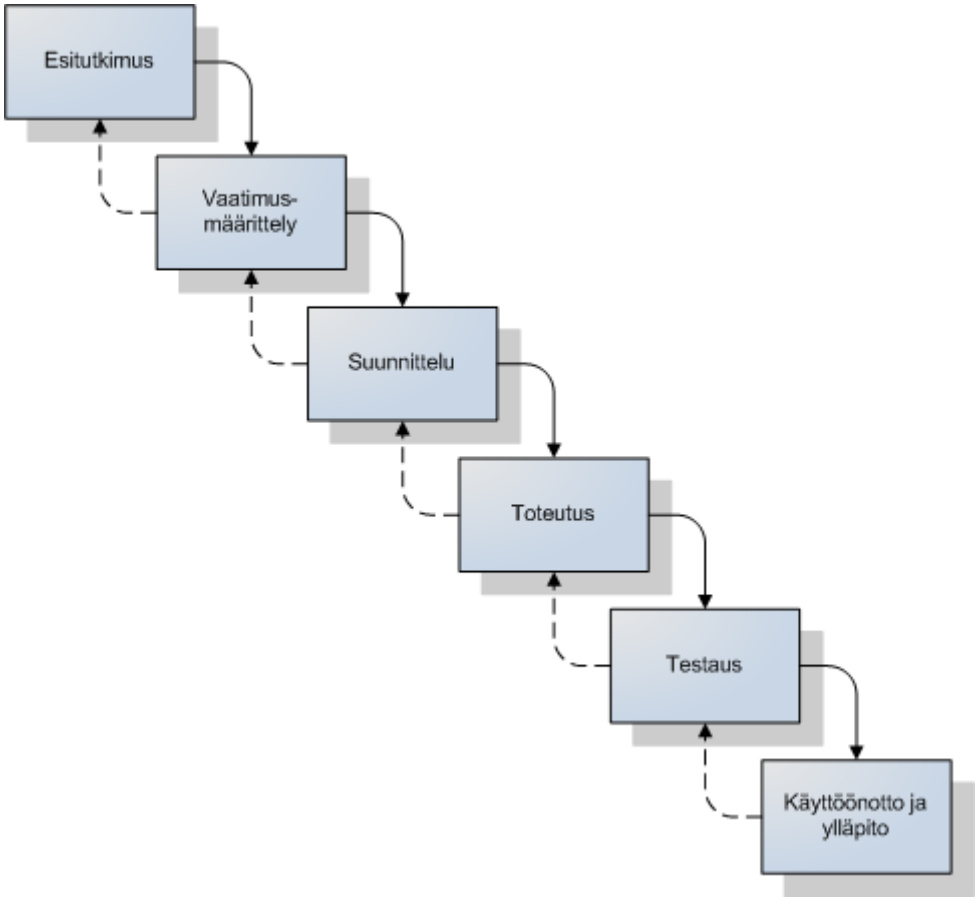
luksen laatu todennäköisesti kärsii, mikäli liikkumavaraa ei muualla ole. [18, 31, s. 9-16]

Tulevissa kappaleissa esitellään tarkemmin neljä prosessimallia: perinteisistä menetelmistä vesiputousmalli ja inkrementaalinen kehitys sekä ketteristä menetelmistä Scrum ja XP. Edellä mainituista vesiputousmalli ja Scrum liittyvät vahvasti case-projektiin ja kaksi muuta ovat myös usein käytettyjä prosessimalleja.

2.1 Vesiputousmalli

Winston Royce esitteli artikkelissaan vuonna 1970 prosessimallin, joka myöhemmin opittiin tuntemaan nimellä vesiputousmalli [27]. Vaikka Royce esitteli mallin viallisena ja käytäntöön soveltumattomana, se omaksuttiin laajalti ja sitä käytettiin yleisesti prosessimallina 80- ja 90-luvuilla ja vielä viime vuosinakin [17].

Vesiputousmalli kuvataan koostuvan – lähteestä riippuen – viidestä–seitsemästä peräkkäisestä vaiheesta; joskus tietyt vaiheet on yksinkertaistettu yhdeksi vaiheeksi tai vaiheet on nimetty hieman eri tavalla, mutta vaiheistuksen periaate on aina sama. Esimerkiksi esitutkimus ja vaatimusmäärittely on joissain kaavioissa esitetty yhtenä vaiheena (ks. esim. [14, s. 25]).



Kuva 1. Vesiputousmalli

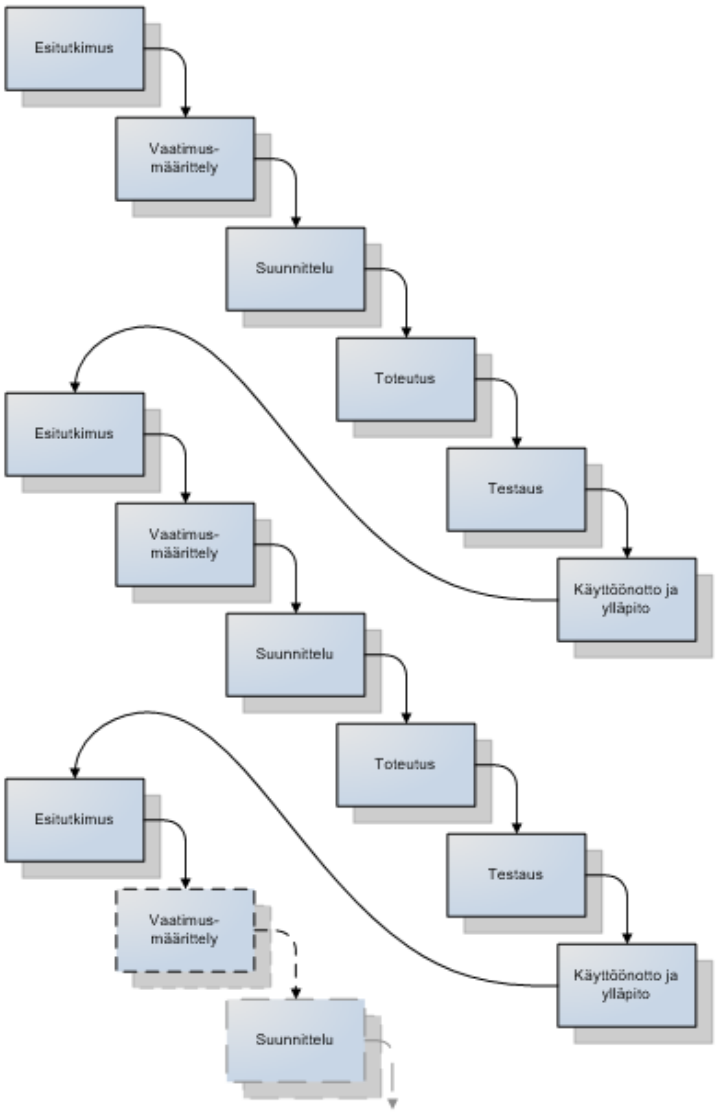
Tässä tutkimuksessa vesiputousmalli kuvataan kuusivaiheisena (Kuva 1) esitutkimusvaiheesta käyttöönotto- ja ylläpitovaiheeseen. Vesiputousmalli lisäsi tiukasti askeleesta toiseen etenevään malliin mahdollisuuden palata takaisin edelliseen vaiheeseen tekemään muutoksia opittujen asioiden perusteella, tätä esittävät katkoviivat kuvassa 1. [14, s. 24–25]

Vesiputousmallia noudatettaessa – tai ainakin virkaintoisten kehittäjien ehkä hieman väärinymmärretysti toteuttamana – seuraavaa vaihetta ei aloiteta ennen kuin edellinen vaihe on saatu täysin valmiiksi. Jos jossain vaiheessa tehdään muutoksia, tulee muutosten heijastaminen taaksepäin toteuttaa askel kerrallaan, eikä vaiheiden ohittamista sallita. Vesiputousmallissa painotetaan tarkkaa dokumentointia, joten jokainen vaihe suunnitellaan ja dokumentoidaan tarkasti.

Vesiputousmallissa asiakas saa arvoa sijoitukselleen vasta aivan projektin lopussa, jolloin kehitetty ohjelmisto toimitetaan. Todella suurissa projekteissa tämä voi tarkoittaa, että käyttäjät pääsevät kokeilemaan ohjelmaa vasta vuosien päästä projektin aloittamisesta [21]. Vesiputousmallin vahvuus – toimintojen järkevä ja kurinalainen vaiheistaminen – kääntyy sen suurimmaksi heikkoudeksi: vaatimusten kiinnittäminen kehitysprojektin heti alkuvaiheessa ja sovelluksen kehittäminen niiden perusteella käyttäjien tarpeiden tai liiketoimintaympäristön muutoksista riippumatta johtaakin usein lopputulokseen, joka ei ole toivottu [14, s. 24–25]. Vesiputousmalli – sellaisena kuten sitä käytännössä on usein harjoitettu – soveltuisikin parhaiten käytettäväksi niissä todella harvinaisissa projekteissa, joissa vaatimukset ovat täysin selkeitä, eivätkä tule muuttumaan.

2.2 Inkrementaalinen kehitys

Inkrementaalisessa kehityksessä kokonaisuus toimitetaan vähän kerrallaan pienissä paloissa. Tällaista mallia on toteutettu käytännössä eri tavoin jo vuosikymmeniä. Inkrementaalinen kehitysmalli voi esimerkiksi sisältää useita peräkkäisiä kokonaisia vesiputouksia (ks. esim. Kuva 2), tai pelkästään inkrementaalisen toteutusvaiheen, jolloin mahdollisimman paljon vaatimusten analysoinnista ja suunnittelusta on tehty yhdellä kertaa etukäteen. [18]



Kuva 2. Esimerkki inkrementaalisesta mallista, joka sisältää peräkkäisiä vesiputouksia

Ensimmäisissä inkrementeissä kehityksen fokus on välttämättömissä ja tärkeimmissä ominaisuuksissa ja ominaisuuksien kirjoa kasvatetaan myöhemmin toimitettavilla lisäyksillä. Viimeisissä inkrementeissä voidaan lopulta tehdä vain viimeistelyä ja haluttaessa lisätä muita vähemmän olennaisia ominaisuuksia. [18, 30]

Inkrementaalinen kehitys on askel ketterämmän kehityksen suuntaan, jossa muutoksiin reagointi on vesiputousmallia helpompaa. [30]

2.3 Ketterät menetelmät

Ketterän kehityksen tärkeimmät arvot ja periaatteet on lueteltu ketterän kehityksen manifestissa [5]. Ketterän kehityksen arvot ovat vapaasti suomennettuna seuraavanlaiset:

Ketterän kehityksen manifesti

Yksilöt ja vuorovaikutus ovat tärkeämpiä kuin prosessit ja työkalut
Toimiva ohjelmisto on tärkeämpi kuin kattava dokumentaatio
Yhteistyö ja kumppanuus ovat tärkeämpiä kuin sopimusneuvottelut
Muutoksiin reagointi on tärkeämpää kuin suunnitelman noudattaminen

Yllä oleva manifesti määrittelee vasemman puolen lihavoidut asiat oikean puolen asioita tärkeämmiksi. Vaikka tärkeitä arvoja korostetaan vastakkainasettelulla, oikean puolen asiat eivät ole merkityksettömiä, eikä niitä pidä jättää kokonaan huomiotta.

Tässä tutkimuksessa käsitellään ketteristä menetelmistä tarkemmin ainoastaan Scrumia ja XP:tä (*eXtreme Programming*). Muita tunnettuja ketteriä menetelmiä ovat muun muassa:

- Adaptive software development
- 4 cycles of control
- Lean development
- Kanban
- Crystal

Kehitysmenetelmää voidaan kutsua ketteräksi, jos se on: [1, 8]

- **Iteratiivinen.** Sovellus kehitetään useassa kehityssyklissä.
- **Inkrementaalinen.** Sovellusta ei toimiteta yhtenä valmiina pakettina, vaan sitä kehitetään ja toimitetaan pala kerrallaan.
- **Itseohjautuva.** Kehittäjät määrittävät itse parhaat tavat työskennellä.
- **Yhteistoiminnallinen.** Asiakas ja kehittäjät työskentelevät jatkuvassa yhteistyössä ja pitävät yllä läheistä kommunikointia.
- **Suoraviivainen.** Menetelmä itsessään on helppo oppia ja se on muokattavissa sekä hyvin dokumentoitu.
- **Mukautuvainen.** Viime hetken muutoksien tekeminen on mahdollista.
- **Kasvava ja kehittyvä.** Prosessi, periaatteet ja työtavat kehittyvät projektin aikana sen sijaan, että ne määriteltäisiin projektin alussa.

Boehm ja Turner [8] määrittelevät, että prosessi, joka ei sisällä kaikkia em. ominaisuuksia voi osoittaa ketteryyttä, mutta sitä ei pitäisi kutsua ketteräksi menetelmäksi vaan ”kevyeksi määritellyksi menetelmäksi”. He myös luettelevat olennaisia konsepteja, jotka ovat yleisiä useimmille ketterille menetelmille:

- Muutosten hyväksyminen
- Nopeat kehityssyklit ja tiheä osien toimitus
- Yksinkertainen design
- Refaktorointi – Ohjelmakoodin rakenteen parantaminen ilman toiminnallisuuden muuttamista
- Pariohjelmointi
- Retrospektiivi (*retrospective*) – Katsaus menneeseen iteraatioon, jolla pyritään selvittämään ongelmia ja onnistumisia ja oppimaan niistä
- Hiljainen tieto – Yksilöiden tiedostamattoman, rutiininomaisen tiedon kerääntyminen, joka ohjaa päivittäistä toimintaa ja työskentelyä
- Testivetoinen kehitys (*Test-Driven Development, TDD*)
- Definition of Done – Yhdessä sovittu lista kriteereistä joiden pitää täyttyä, jotta jonkin tehtävän voidaan sanoa olevan valmis

Ketterien menetelmien kannattajat uskovat, että avain menestyksekkääseen sovelluskehitykseen on kehitystiimin ammattitaito ja kyvykkyys. Siksi menetelmästä riippumatta tulisi keskittyä tiimin tukemiseen heidän työskentelyssään ja välttää liiallista huomion kiinnittymistä itse prosessiin. [10]

Kuten ketterän kehityksen manifestissakin todetaan, dokumentointi ei ole olennainen osa ketterää kehitysprosessia. Dokumentointia tulisi tehdä vain sen verran kuin on todella tarpeen, eikä yhtään enempää. Sovelluksesta riippuen, jopa pelkästään hyvin kommentoitu lähdekoodi voi itsessään olla tarpeeksi. Vain dokumentointi, josta on hyötyä sidosryhmille, tai joka huomattavasti helpottaa kommunikointia, tulisi tehdä. Lisääntynyt kommunikaatio vähentää dokumentoinnin tarvetta ja hyödytön dokumentointi vain hidastaa kehitystyötä. [10]

> *"Plans are useless,*
> *but planning is indispensable."*
> *– D.D. Eisenhower*

Erityisesti suunnitelmien yksityiskohtainen dokumentointi kehitysprosessin aikaisessa vaiheessa voi aiheuttaa paljon turhaa työtä, sillä iteratiivisessa kehityksessä suunnitelma voi muuttua merkittävästi toteutuksen aikana [10]. Se ei kuitenkaan tarkoita, että etukäteissuunnittelua ei pitäisi koskaan tehdä. On vain hyvin vaikeaa ennustaa kaikkea, mitä tapahtuu sovelluskehityksen aikana nopeasti muuttuvassa ympäristössä. James Highsmith ehdottaa kirjassaan *Adaptive Software Development* [11], että suunnittelu pitäisi korvata spekulaatiolla. Perinteisestihän poikkeamat suunnitelmasta nähdään virheinä tai erehdyksinä, jotka pitää korjata. Sen sijaan prosessin alussa tehty ”suunnitelma-spekulaatio” on vain oletus siitä, millainen kehitysprosessi ja siitä saatava tuotos tulee olemaan. Kehitystiimin pitäisi pystyä

oppimaan tilanteista, joissa alkuperäinen suunnitelma ei toimikaan ja poikkeamat suunnitelmasta ainoastaan ohjaavat halutun lopputuloksen suuntaan. [11]

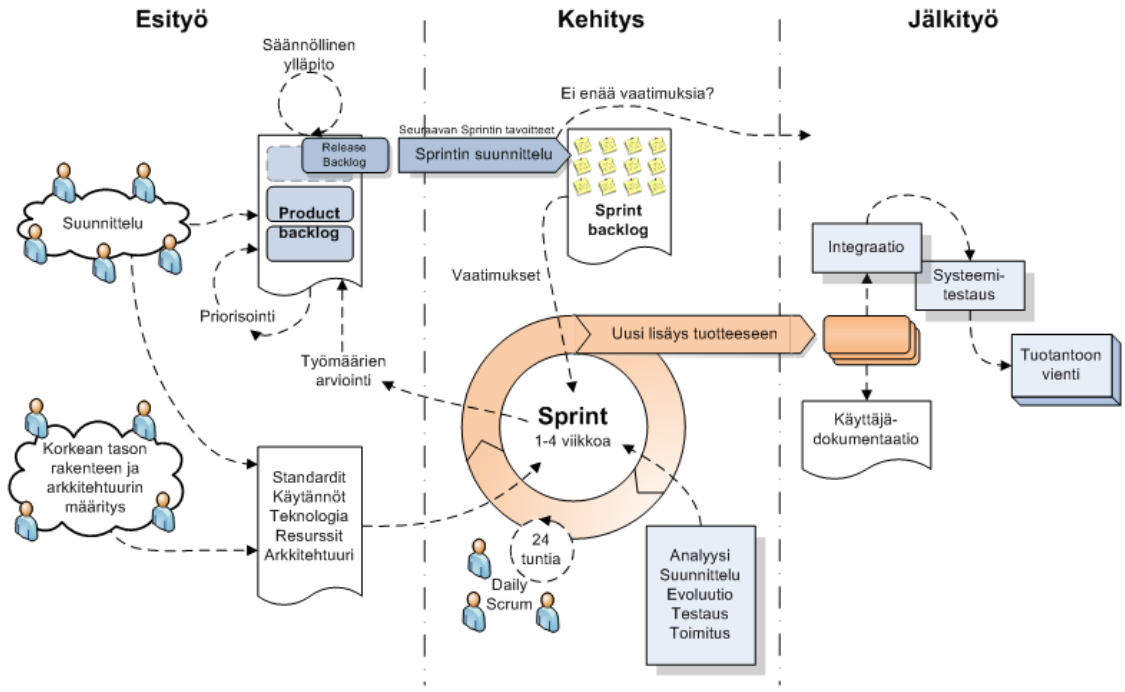
2.4 Scrum

Scrum on ketterä menetelmä, jonka lähtökohtana on ohjelmistokehitysprosessin hallinta muuttuvan ympäristön ja muuttuvien vaatimusten paineessa. Menetelmänä Scrum on yksinkertainen ja suoraviivainen, määritellen ainoastaan pienen, mutta riittävän määrän sääntöjä, sallien toteuttajille paremman mahdollisuuden keskittyä olennaiseen. Ketterille menetelmille ominaisesti Scrumissa määritellään vähemmän kerrallaan ja keskitytään kyseisen asian toteuttamiseen ja mahdollisten esiin tulevien esteiden poistamiseen. Scrumissa pyritään avoimeen suhteeseen asiakkaan ja kehittäjien välillä ilman arkojen asioiden salaamista. [23]

Scrumin tavoitteina on mm. helpottaa projektin seurantaa ja mahdollistaa nopea reagointi muuttuviin vaatimuksiin sekä parantaa kehittäjien tuottavuutta. Scrumissa pyritään myös ylläpitämään tiivistä kommunikaatiota asiakkaan ja kehittäjien välillä.

2.4.1 Scrumin vaiheet

Scrum-prosessi koostuu kolmesta vaiheesta: esityöstä, kehitysvaiheesta ja jälkityöstä.



Kuva 3. Scrum-prosessin vaiheet. Malli on muodostettu Abrahamssonin kaavion pohjalta [1].

Esityö

Esityövaihe koostuu kahdesta rinnakkaisesta osasta, suunnittelusta sekä korkean tason rakenteen ja arkkitehtuurin määrityksestä: [19]

- Suunnittelu
    - Kaikkien toiminnallisten ja ei-toiminnallisten vaatimusten kerääminen eli product backlogin muodostaminen (ks. kappale 2.4.3, jossa product backlog selitetään tarkemmin)
    - Toteutukseen vaadittavien resurssien ja työmäärän karkea arviointi
    - Toteutukseen ja eri vaatimuksiin liittyvien riskien arviointi
    - Product backlogin priorisointi
    - Toteutuksen aikataulutus ja toimituspäivämäärän päättäminen; Jos toteutettava järjestelmä on suuri, on usein tarpeen, että product backlog jaetaan sopiviin kokonaisuuksiin. Yksi kokonaisuus muodostaa tällöin release backlogin, joka toimitetaan erikseen ja sen valmistumiselle asetetaan oma tavoitepäivämäärä. Yhden kokonaisuuden kehityksen aikana tulisi olla riittävä määrä iteraatioita, esimerkiksi neljä tai enemmän.
    - Yhden tai useamman kehitystiimin muodostaminen
    - Tarvittavien työkalujen ja tilojen järjestäminen toteutusta varten
- Korkean tason rakenteen ja arkkitehtuurin määritys
    - Käyttöympäristön ja kohteen analysointi product backlogissa listattujen vaatimusten pohjalta; Tarvittaessa luodaan malleja kuvaamaan suunniteltua ratkaisua sekä helpottamaan kriittisten ja ongelmallisten kohteiden määrittelyä. Voidaan luoda myös prototyyppejä, jotta järjestelmää ja sen tarpeita on helpompi havainnollistaa ja ymmärtää.
    - Järjestelmän arkkitehtuurin määrittely siten, että se palvelisi mahdollisimman hyvin listattuja vaatimuksia halutussa käyttöympäristössä.
    - Product backlogin päivitys tukemaan määriteltyä arkkitehtuuria; Tarvittaessa product backlogiin tehdään lisäyksiä määritetyn arkkitehtuurin toteuttamista varten.

Kehitys

Kehitysvaihe kattaa suuren osan Scrum-prosessista. Kehitysvaihe jakautuu iteraatioihin eli sprintteihin, joiden kunkin tavoite on tuottaa toimiva järjestelmän osa. Sprinteistä kerrotaan tarkemmin kappaleessa 2.4.4.

Jälkityö

Jälkityöhön kuuluvat tyypillisesti perinteiset sovelluksen toimitukseen liittyvät vaiheet: [19]

1) Sprinteissä luotujen sovelluksen osien integrointi
2) Koko järjestelmän testaus
3) Hyväksymistestaus
4) Käyttäjädokumentaation valmistelu

5) Koulutus- ja markkinointimateriaalin luonti
6) Käyttäjien ja ylläpitäjien koulutus
7) Järjestelmän muuntaminen tuotantokelpoiseksi

2.4.2 Roolit

Product owner

Product owner on asiakkaan puolella henkilö, joka tuntee toimialan, yrityksen ja ymmärtää sovelluskehitystä. Hänellä on seuraavat vastuut: [25]

- Vastaa ja määrittelee uuteen sovellukseen tulevat ominaisuudet
- Päättää toimituksen aikataulusta ja sisällöstä
- Vastaa sovelluksen tuottavuudesta
- Priorisoi halutut ominaisuudet (eli product backlogin) niiden arvon mukaan
- Ylläpitää listaa ominaisuuksista ja prioriteeteista
- Hyväksyy tai hylkää kehittäjien työn tulokset

Kuva 4. Scrumin eri roolit. Vitsillä siasta ja kanasta voidaan selittää Scrumin eri rooleja. Scrum-tiimin jäsenet voidaan nähdä sikoina ja käyttäjät kanoina.

Scrum master

Scrum master on osa kehitystiimiä ja hän varmistaa, että Scrumin periaatteita ja käytäntöjä noudatetaan. Näennäisessä johtajan roolissa scrum master auttaa tiimiä tekemään päätöksiä tai hankkimaan tarvittavia resursseja. Hänen tehtävänään on myös huolehtia tiimin tuottavuudesta. Hänen täytyy puuttua asiaan, mikäli esimerkiksi tiimin sisällä syntyy konflikteja. [25]

Kehitystiimi

Kehitystiimi (tai pelkkä tiimi) on joukko toteuttajia, jotka tekevät varsinaisen kehitystyön. Scrumissa luotetaan tiimin aloitekykyyn ja saumattomaan yhteistyöhön. Useita tiimejä voi olla yhtäaikaa työskentelemässä saman product backlogin asioiden parissa. [23]

Tiimit ovat itseohjautuvia ja autonomisia, eli päättävät omista asioistaan. Tiimin työskentelyä rajoittavat ainoastaan organisaation standardit ja käytännöt sekä product backlog. Tiimin päätettäväksi jää, kuinka product backlogista tuleva tehtävä muotoutuu tuotteeseen parannukseksi tai uudeksi ominaisuudeksi. Tiimi ylläpitää listaa tehtävistä, joita kulloinkin meneillään olevan sprintin aikana suoritetaan, tätä listaa kutsutaan nimellä Sprint Backlog. [23]

Scrum-tiimi

Scrum-tiimi on yhteisnimitys joukolle, johon kuuluvat scrum masterin ja kehitystiimin lisäksi myös product owner. Scrum-tiimin kuuluvien henkilöiden voidaan katsoa olevan ”sikoja” kun taas sovelluksen käyttäjät ovat ”kanoja” (ks. Kuva 4). Siat ovat resursseja, jotka ovat sitoutuneet projektiin ja vievät kehitystä eteenpäin. Kanat hyötyvät sikojen työstä ja tuovat projektiin arvoa, mutta niillä ei ole mitään hävittävää.

2.4.3 Product backlog

Product backlog on priorisoitu lista tuotteeseen halutuista ominaisuuksista. Product backlog ei tule koskaan valmiiksi, vaan se elää tuotteen ja kehityksen mukana. Halutuimmat, tärkeimmät tai liiketoiminta-arvoltaan arvokkaimmat asiat priorisoidaan korkeimmalle. Halutut ominaisuudet ja toiminnot voivat tulla product backlogiin mistä tahansa, esim. käyttäjiltä, asiakkailta, myynniltä, markkinoinnilta ja kehittäjiltä. [22]

Product backlogin priorisoinnin hoitaa product owner, ja käytännössä hän päättää, missä järjestyksessä toteutus etenee. Listattujen vaatimusten ja ominaisuuksien tulisi olla käyttäjätarinoiden (*User Story*) muodossa. Käyttäjätarina on lyhyt kuvaus, josta tulisi selvitä käyttäjän *rooli*, ominaisuuden tai toiminnon *tavoite*, sekä mielellään myös *syy*, miksi tavoite halutaan saavuttaa. Lisäksi jokaisen käyttäjätarinan tulisi olla itsenäinen, arvioitavissa, testattavissa sekä kokonaisuudelle merkityksellinen, mutta myös sopivan pieni. [22]

Esimerkki käyttäjätarinasta:

1) *Ylläpitäjänä (rooli) haluan nähdä listan järjestelmään rekisteröidyistä käyttäjistä (tavoite), jotta voin muokata heidän henkilötietojaan (syy).*

Usein sanotaan, että sopivan kokoinen käyttäjätarina on sellainen, jonka toteuttaminen kestää enintään 10 henkilötyöpäivää, mutta käytännössä sopiva koko on projektikohtaista [26]. Usein käyttäjätarinoiden vaativuutta ei kuitenkaan suoraan arvioida työtunteina tai -päivinä, vaan käytetään jonkinlaista pisteytysmenetelmää. Tarvittaessa liian suuria ja epämääräisiä käyttäjätarinoita tulee tarkentaa ja pilkkoa pienempiin ja selkeämpiin osiin.

Edellä mainittu esimerkkikin voitaisiin pilkkoa kahdeksi käyttäjätarinaksi:

1) *Ylläpitäjänä haluan nähdä listan järjestelmään rekisteröidyistä käyttäjistä.*

2) *Ylläpitäjänä haluan voida muokata järjestelmään rekisteröityjen käyttäjien henkilötietoja.*

Yllä olevat esimerkit ovat vain lyhyitä otsikoita halutuista ominaisuuksista. Tarinoiden tulisikin luonnollisesti muodostua ja tarkentua sitä mukaa, mitä lähemmäksi ominaisuuden toteutusajankohtaa mennään. Tarkan kuvauksen lisäksi ominaisuudelle tulisi määritellä mitattavissa olevat hyväksymiskriteerit eli hyväksymistestit. Hyväksymistestit yhdessä käyttäjätarinan selkeän kuvauksen kanssa määrittävät sen, milloin jokin ominaisuus voidaan todeta valmiiksi. [26]

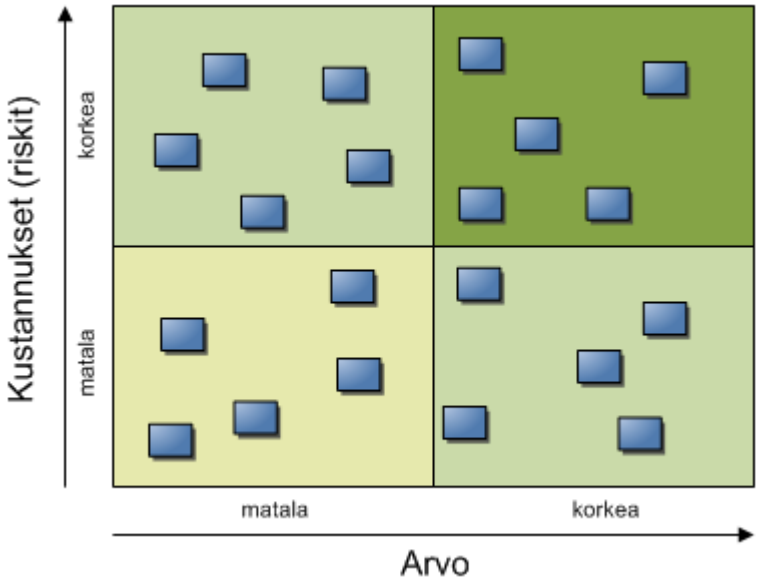
Käyttäjätarinoiden karkeaan priorisointiin voidaan käyttää esimerkiksi MoSCoW-sääntöä:

1. Must have = pakko toteuttaa
2. Should have = pitäisi toteuttaa
3. Could have = voisi toteuttaa
4. (Would be nice, but probably) Won't have = (Olisi kiva, mutta luultavasti) ei toteuteta

Pelkkä MoSCoW-priorisointi ei kuitenkaan ole välttämättä riittävä. Jo kohtuullisen kokoista järjestelmää suunniteltaessa *Must have* -ominaisuuksia voi olla niin iso määrä, ettei niitä kaikkia ehditä toteuttaa pienessä määrässä sprinttejä. Mistä toteutus aloitetaan, jos tärkeimpiä ominaisuuksia ei vielä ole saatu selville?

Sovellusta kehittävän yrityksen tavoitteena on pääsääntöisesti mahdollisimman suuri ROI (*Return On Investment*). ROI:n määrityksen perustana on jostain ominaisuudesta saatava liiketoimintahyöty vastaan ominaisuuden toteutuskustannukset. Toteutuskustannuksissa tulee huomioida myös toteutukseen liittyvät tekniset riskit. Viimeistään tässä vaiheessa, ennen tarkempaa priorisointia, kehittäjätiimin tulee antaa karkea arvio ominaisuuksien toteutuksen vaativuudesta ja mahdollisista teknisistä riskeistä. Jos ominaisuuslista on liian pitkä, voidaan arviot antaa vain minimimäärälle vaadittuja ominaisuuksia, joilla sovellus voidaan toimittaa. Arvioiden perusteella eri ominaisuuksien arvioitua (kustannus)riskiä voidaan suhteuttaa sen oletettuun arvoon ja tehdä näin yksinkertaista luokittelua (ks. Kuva 5). [28]

Jos yksittäinen käyttäjätarina puretaan priorisoinnin jälkeen osiin, tulee osien prioriteetti arvioida uudelleen. Osiin purkamisen jälkeen voidaan huomata jonkin aiheeseen liittyvän komponentin olevan aiemmin luultua tärkeämpi tai päinvastoin.



Kuva 5. Käyttäjätarinoiden luokittelu kustannusten ja arvon mukaan

Usein käyttäjätarinoiden priorisointiin ehdotetaan niiden liiketoiminta-arvon arvioimista product ownerin toimesta jollain numeerisella asteikolla ja tarinat tulisi sitten järjestää saadun arvon mukaan. Muita usein käytettyjä priorisointimenetelmiä ovat: [28]

- **Minimi ominaisuusjoukko, joka myy:** valitaan riittävä määrä välttämättömiä ja myyviä ominaisuuksia, jotta sovellus kiinnostaa käyttäjiä.
- **Vastinetta rahalle:** Tapa muistuttaa läheisesti liiketoiminta-arvon mukaan priorisoimista, mutta tässä mallissa toteutuksen vaativuudella on suurempi painoarvo. Liiketoiminta-arvoltaan samaa suuruusluokkaa olevista ominaisuuksista toteutetaan ensin se, jonka toteuttamiseen on arvioitu kuluvan vähemmän resursseja.
- **Tekniset riskitekijät ensin:** Otetaan haastavimmat ominaisuudet työn alle ensin. Tähän liittyy kuitenkin vaaroja: päädytään ylihienoon ratkaisuun, jolle ei ole käyttäjätarinaa tai jolle on vain matalan arvon tarina, joka kuitenkin vaatii ominaisuuden täydellisemmän ratkaisun. Toisena vaarana on, että yksinkertaisempi ratkaisu voisi löytyä myöhemmin, kun ongelmat ymmärretään paremmin tai rajoitukset on paremmin määritelty.
- **Riskien lykkäys:** Aloitetaan ominaisuuksista, joilla on korkea hyötyodotus ja pienet riskit. Seuraavaksi tulevat korkean riskin ja korkean hyötyodotuksen tehtävät, joita puretaan helpommin hahmotettaviin osiin ja tehdään siten pieni pala kerrallaan.
- **Äänestäminen:** Pyydetään käyttäjiä äänestämään, mitä listan ominaisuuksia he haluaisivat eniten. Tämä toimii hyvin tilanteessa, jossa sovelluksella on jo olemassa oleva käyttäjäjoukko.

2.4.4 Sprintit

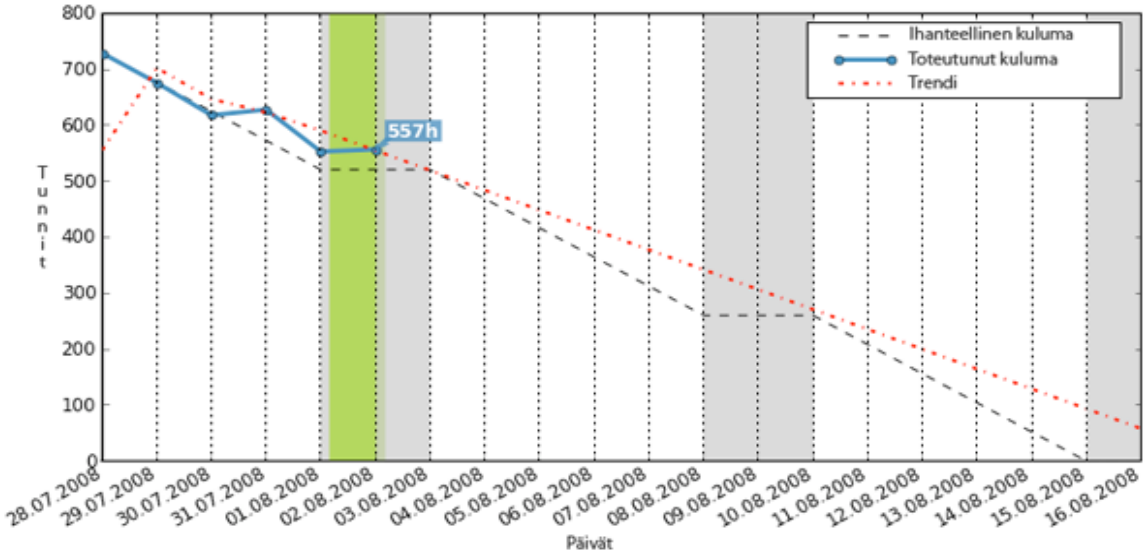
Sprintit ovat vähintään muutaman päivän ja korkeintaan kuukauden pituisia kehityssyklejä. Sprinttiin otetaan product backlogista toteutettavaksi sellainen joukko asioita, jonka tiimi arvioi pystyvänsä toteuttamaan sprintin aikana. Sprintin aikana kehittäjille annetaan työrauha, eikä kenenkään Scrum-tiimin ulkopuolisen tulisi tänä aikana antaa uusia ohjeita, vaan mahdolliset muutokset kehityksen suuntaan otetaan vastaan ja huomioidaan sprinttien välissä.

Scrumissa tavoite on, että jokaisen sprintin jälkeen syntyisi valmis ja toimiva osa tuotetta, jota voidaan jo käyttää ja siten kehitys voidaan periaatteessa lopettaa minkä tahansa sprintin jälkeen. Asiakas voi siis itse päättää milloin sovellus kelpaa käyttöön. Kehitys voidaan lopettaa esimerkiksi jo silloin, kun 80 prosenttia product backlogissa olevista asioista on tehty. Päätös voi johtua esimerkiksi siitä, että asiakkaan mielestä loput 20 prosenttia eivät tuo riittävästi hyötyä kustannuksiin nähden, sovellus halutaan tuotantoon aikataulusyistä tai sovelluksen kehitykseen varattu budjetti on käytetty. Tuotteen arkkitehtuuri ja suunnittelu *(design)* kehittyy useiden sprinttien aikana sen sijaan, että se toteutettaisiin kokonaan ensimmäisissä sprinteissä. [23, s. 8-9]

Sprint planning

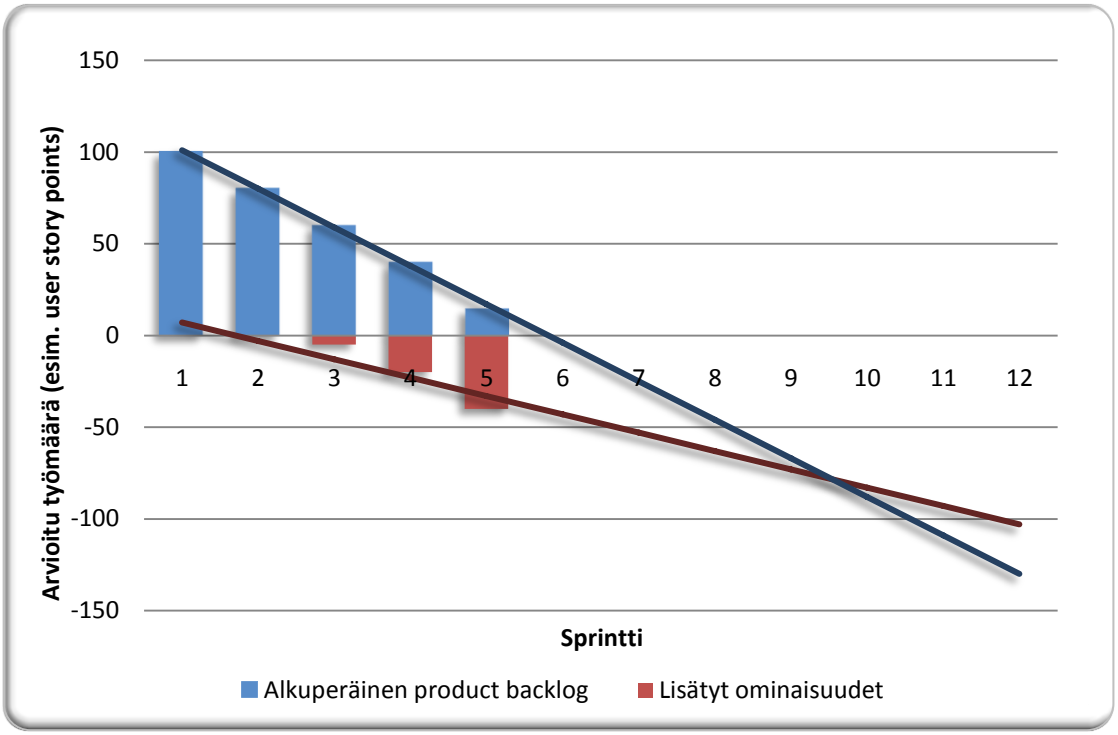
Jokaisen sprintin alussa pidetään suunnittelupalaveri, jossa määritetään tavoite tulevalle sprintille. Vähintään kehitystiimi(e)n ja product ownerin tulisi osallistua palaveriin. Mahdollisesti myös käyttäjät, johto tai muut asiakkaan edustajat voivat osallistua. Käytännössä sprintin tavoitteen määräävät ja havainnollistavat product backlogista otettavat tehtävät (käyttäjätarinat). Tehtäviä pilkotaan palaverissa tarpeen mukaan pienempiin osiin, joille tehdään työmääräarvio henkilötyötunneissa. Kukin tehtävä tulisi olla arvioidulta työmäärältään korkeintaan noin 15 tuntia. Näistä tehtävistä muodostuu Sprint Backlog. [19]

Tehtävien pilkkomisella tavoitellaan sitä, että kaikki Scrum-tiimin jäsenet ymmärtävät sprintin tavoitteet ja siihen sisältyvät tehtävät. Pienempien tehtävien avulla seuranta sprintin sisällä on helpompaa. Tehtävien jäljellä olevia arvioituja työmääriä ylläpidetään päivittäin ja siten sprintin etenemistä ja aikataulun pitävyyttä voidaan helposti hahmottaa *burndown*-graafilla (Kuva 6). Burndown-graafi esittää arviota sprintin jäljellä olevasta työajasta.



Kuva 6. Esimerkki sprintin burndown-graafista

Myös koko product backlogin (tai release backlogin) kattava burndown-graafi, johon kunkin sprintin alussa lasketaan product backlogissa jäljellä oleva kokonaistyömäärä, voi olla hyödyllinen. Sellaisen graafin avulla voidaan arvioida, kuinka monta sprinttiä nykyisellä projektin etenemisnopeudella kestää ennen kuin kaikki vaaditut sovelluksen ominaisuudet ovat valmiina. Graafista saatava ennuste on epävarma ja se voi vaihdella jokaisen sprintin jälkeen. Ennusteesta saadaan tarkempi, kun siinä huomioidaan erikseen uusien käyttäjätarinoiden ilmestyminen (ks. Kuva 7). Kuvan kaltainen graafi toimii yksinkertaisena ja helposti hahmotettavana työkaluna, josta voi olla hyötyä projektin seurannassa.



Kuva 7. Esimerkki product backlogin burndown-graafista. Graafiin piirrettyjen trendiviivojen leikkauskohta ennustaa, että nykyisellä työtahdilla ja uusien tehtävien ilmestymisnopeudella kaikki ominaisuudet tulisivat valmiiksi sprintin 9 jälkeen.

Daily Scrum

Scrum-tiimi tapaa päivittäin lyhyessä tilannekatsauksessa, jota kutsutaan Daily Scrumiksi. Daily Scrumissa tarkastellaan toteutuksen etenemistä ja mahdolliset hidasteet tai esteet otetaan esille, jotta scrum master voi ryhtyä toimenpiteisiin niiden poistamiseksi.

Sprint review eli katselmointi

Sprintin lopuksi tiimi kerääntyy yhteen hallinnon ja muiden sidosryhmien kanssa katselmointitapaamiseen tarkastelemaan saavutettuja kehitysaskelia. Tässä vaiheessa päätetään, jatketaanko lisäaskeleiden ottamista tehdyn toteutuksen varaan. Useimmiten näin käykin, mutta periaatteessa voidaan myös päättää, että toteutetut asiat eivät vastaa tarpeita ja tehty toteutus hylätään kokonaan. Ainoa asia, joka tällöin hävitään, on sprintin verran tehtyä työtä.

Katselmoinnin jälkeen hallinto usein järjestelee product backlogin uudelleen ottaakseen hyödyn irti tiimin kehitystyöstä. Product backlogilla on enemmän merkitystä, kun sitä tarkastellaan osittain toteutetun tuotteen perspektiivistä. Hallinto voi huomata, että tuotteesta on toteutettu niin paljon ominaisuuksia, että ominaisuuksien lisääminen ei lisää ROI:ta. Tällöin voidaan päättää, että kehitysvaihe lopetetaan ja siirrytään valmistelemaan sovelluksen tuotantoonvientiä aikataulusta edellä.

Retrospektiivi

Retrospektiivi on tiimin mahdollisuus tutkia kuluneen sprintin onnistumisia ja epäonnistumisia, ja oppia niistä. Retrospektiivi on epämuodollinen tilaisuus, joka kestää yleensä 1-2 tuntia. Siihen osallistuu yleensä vain kehitystiimi ja scrum master, ja se pidetään jokaisen sprintin päätteeksi. Siellä keskustellaan erityisesti asioista, joissa tiimin mielestä olisi kehitettävää tai muista asioista, jotka vaikuttavat tiimin työtehoon. Asiat voivat koskea työssä viihtyvyyttä, työtapoja, työvälineitä, henkilökemiaa, ja niin edelleen.

Retrospektiivin päätteeksi tiimi voi valita esille nousseista ongelmista tai kehityskohteista muutaman tärkeimmän, joihin tiimi erityisesti keskittyy panostamaan. Seuraavan retrospektiivin aluksi voidaan katsoa taaksepäin ja analysoida kuinka hyvin kyseisissä asioissa on kehitytty.

2.5 XP

XP eli *eXtreme Programming* on prosessimalli, joka keskittyy siihen, kuinka ohjelmistokehitystä tehdään käytännössä. XP kokoaa yhteen joukon jo pitkään tunnettuja ohjelmistokehityksen parhaita käytäntöjä ja nimensä mukaisesti vie ne äärimmilleen. [6, 30]

XP:hen kuuluvia keskeisiä käytäntöjä ovat:

- Pariohjelmointi
- Minimaalisen järjestelmän toimittaminen tuotantoon
- Jatkuva kommunikointi
- Jatkuva (automaattinen) testaus
- Yksinkertainen design
- Refaktorointi

Ihanteellisesti kaikki ohjelmointi tehdään pareittain. Toinen koodaa, toinen katsoo ja kommentoi ja rooleja vaihdetaan aika ajoin. Ongelmat aukeavat keskustelemalla ja molempien tietämys kasvaa. Kattava yksikkötestaus sekä myös testilähtöinen kehitys on olennaista. Järjestelmäintegraatiota tehdään parhaimmillaan useita kertoja päivässä ja koko järjestelmää testataan jatkuvasti.

Koodi pidetään mahdollisimman yksinkertaisena, johon liittyy myös se, että koodataan jotain vasta kun tarve on välttämätön. Sitten, kun jokin ominaisuus tarvitaan, se toteutetaan helpoimmalla ja yksinkertaisimmalla tavalla siten, että se toimii. Ennakoivaa koodausta vältetään, sillä vaatimukset voivat muuttua myöhemmin. Myös ymmärrys ongelma-alueesta kasvaa ajan kanssa. Jotta jatkokehitystyö voidaan pitää joustavana ja design yksinkertaisena, koodia refaktoroidaan ahkerasti aina tarvittaessa ja usein jo heti halutun toiminnon valmistuttua.

Järjestelmän kehittyessä siitä toimitetaan pieniä paloja kerrallaan nopeilla toimitussykleillä. Viestintä on jatkuvaa sekä kehitystiimin sisällä että asiakkaan kanssa. Näin asiakas voi myös vaikuttaa kehityksen suuntaan nopeasti.

Monet XP:n käytännöt mainitaan usein Scrumin yhteydessä, vaikka ne eivät siihen suoranaisesti kuulu. Aiemmin on suositeltu, että kaikkia XP:n käytäntöjä hyödynnetään, mutta nykyään suositaan myös lähestymistapoja, joissa käyttöön otetaan vain yksi tai muutamia XP:n käytännöistä [30]. XP:n käytäntöjä ja Scrumia voidaan käyttää yhdessä. XP:n säännöt koskevat prosessin käytäntöjä ja toimintatapoja, Scrum taas on prosessin hallintaan tähtäävä menetelmä, jolloin yhdessä käytettynä ne täydentävät toisiaan [24].

2.6 Aiempia kokemuksia Scrumista ja XP:stä

Aiempia kokemuksia tätä tutkimusta varten kerättiin etsimällä Scrumia ja muita ketteriä menetelmiä käsitteleviä artikkeleita pääasiassa internetin elektronisista kokoelmista käyttäen erilaisia yhdistelmiä esimerkiksi hakusanoista *scrum, agile, extreme programming, case study, experiences, practices ja enterprise.* Artikkeleita löytyi paljon, ja haasteena olikin löytää siitä massasta relevantteja, todellisia kokemuksia käsitteleviä raportteja. Sopivia artikkeleita jäljitettiin poimimalla ensin vain kiinnostavimmat otsikot. Näistä artikkeleista luettiin kursorisesti esipuhe sekä yhteenveto, ja mikäli artikkeli vaikutti lupaavalta, sisältöön tutustuttiin tarkemmin. Jäljempänä tässä kappaleessa nostetaan esille kiinnostavia huomioita, joihin näissä tarkempaan tarkasteluun päässeissä artikkeleissa törmättiin.

Dybå ym. tekivät vuonna 2009 tutkimuksen, jossa he kartoittivat ketteriin menetelmiin liittyvää tutkimuskenttää. Yli tuhannesta tutkimusraportista he analysoivat tarkemmin 36 raporttia, jotka he totesivat riittävän luotettaviksi ja relevanteiksi. Suurimmassa osassa näistä tutkimuksista käsiteltiin XP:tä. He toteavatkin, että lisäselvitystä Scrumiin liittyviin kokemuksiin kaivataan. Dybå ym. mukaan ketteristä menetelmistä löytyy sekä hyviä että huonoja kokemuksia, mutta suosituksia ketterien menetelmien käyttökohteista erilaisissa ympäristöissä on vaikeaa antaa vahvan näytön yhä puuttuessa. Sen sijaan he suosittelevatkin perehtymistä käsittelemiinsä tutkimuksiin ja etsimään raportteja, joiden tutkimuskohteet vastaavat mahdollisimman hyvin omaa ympäristöä. [9]

Sutherland havaitsi ensimmäisessä johtamassaan Scrum-projektissa, että kehittäjät alkoivat aktiivisesti pienentää toistensa työtaakkaa. Kun tiimin jäsenet kuulivat ja näkivät päivittäin toistensa työtilanteen, yksittäinen kehittäjä saattoi päätyä muuttamaan tekemäänsä koodia säästääkseen monen päivän työn toiselta kehittäjältä. [29]

Rayhan ja Haque tekivät tutkimuksen ulkoistetun Scrumin vaiheistetusta käyttöönotosta. He kertovat tiimin havainneen vasta yli vuoden jälkeen, että menestyäkseen ketterä projekti vaatii parhaiden ketterien käytäntöjen hyödyntämistä. Projektissa hyödynnettyjä ketteriä käytäntöjä olivat muun muassa CI *(Continuous Integration)*, TDD *(Test Driven Development)* eli testilähtöinen kehitys, refaktorointi, testien automatisointi, ketterä QA (eli laadunvarmistus samassa sprintissä kuin kehitys) ja arkkitehtuurin määrittäminen iteratiivisesti. [20]

Babinet ym. toteavat, että Scrum-prosessilla kehitettäessä aikataulun venyessäkin käyttäjät saavat silti käsiinsä tuotteen, jota voi käyttää tai kokeilla ja johon päivittyy uusia ominaisuuksia tasaisin väliajoin. Tämä vähentää turhautumista mahdollisissa toimituksen viivästymistilanteissa. He tutkivat myös riippuvuuksien hallintaa ja Scrumin käyttöä

tapauksessa, jossa saman sovelluksen kehittämiseen osallistui yhtä aikaa useampi kehitystiimi. Babinet ym. mukaan riippuvuudet voivat johtaa konflikteihin tiimien kehitystyössä. Esimerkiksi, jos toisen tiimin eteenpäin pääseminen vaatii työtä toiselta tiimiltä, tulisi Scrum tiimien product ownereiden neuvotella töiden prioriteeteista. Toisen tyyppisiä konflikteja ovat muutokset, jotka aiheuttavat työtä toisille tiimeille. Esimerkiksi jonkin tiimin tekemät arkkitehtuuriset muutokset voivat vaatia muita tiimejä muokkaamaan tekemäänsä koodia vastaamaan tehtyjä muutoksia tai muuttamaan kehitystapojaan. [4]

Babinet ym. mukaan riippuvuuksien hallinnassa Scrum tarjosi merkittävää ohjausta yksittäisille tiimeille, mutta ei juurikaan apua tiimien väliseen koordinointiin. Tutkitussa projektissa riippuvuudet tiimien välisessä kehitystyössä saatiin minimoitua, mutta siitä huolimatta itse sovelluksen koodi ei muuttunut riittävän lyhyessä ajassa. Siten kehitystiimit olivat edelleen riippuvaisia toisistaan välillisesti. [4]

Lindvall ym. tekivät tutkimuksen ketterien menetelmien, erityisesti XP:n, käytöstä suurissa yrityksissä. Tutkimukseen osallistuivat muun muassa Motorola ja Nokia. Tutkimuksen aikana kaikki organisaatiot havaitsivat, että XP:n käytäntöjä täytyi räätälöidä jopa projektikohtaisesti, jotta metodit soveltuisivat organisaatioiden käyttöön. Syinä XP:n käytäntöjen räätälöinnin tarpeeseen olivat, että projektit usein riippuvat toisista projekteista ja sidosryhmistä sekä työ yleensä jakaantuu useille tiimeille. Projektin käytännöille täytyi myös saada yhteensopiva rajapinta organisaation käytäntöihin. Tutkimuksessa päädyttiin johtopäätökseen, että ketterät menetelmät soveltuvat hyvin sovellettavaksi suurissa organisaatioissa, kun työryhmät ovat pieniä ja sijaitsevat toistensa läheisyydessä. [15]

Marchenko ja Abrahamsson tutkivat case-tutkimuksessaan, kuinka Scrumia voidaan käyttää ympäristössä, jossa on meneillään useita projekteja yhtä aikaa. Tutkimuksen pohjalta he tunnistivat 10 haastetta: [16]

1. Tiimin keskittyminen liikaa pelkästään Scrum-prosessiin ja sen käytäntöihin vaikeuttaa normaalia sovelluskehitystä ja itsenäistä ongelmanratkaisua.
2. Scrum masterin liiallinen keskittyminen yksilöihin ja vuorovaikutukseen haittaa prosessin ylläpitoa.
3. Johdolta puuttuu selkeä visio, tavoitteet tai odotukset.
4. Liian suuri määrä ylläpitotyötä ja virheiden korjausta heikentää tiimin tuottavuutta ja työmoraalia.
5. Scrumin ja lyhyiden iteraatioiden sovittaminen tutkimuspainotteiseen tiimityöskentelyyn ei onnistu.
6. Johonkin aiheeseen erikoistuneet kehittäjät vähättelevät yhteistyön merkitystä.
7. Yli-individualismi eli joidenkin henkilöiden piittaamattomuus tiimityöskentelyä kohtaan vaikuttaa negatiivisesti koko tiimin menestymiseen.
8. Tiimi sitoutuu liian suureen työmäärään sprintin ajaksi.

9. Projektin etenemistä on vaikeaa seurata ja seurannan tuloksia ei onnistuta hyödyntämään.
10. Johto sekaantuu liikaa päivittäiseen työntekoon.

Edellä lueteltujen kymmenen haasteen katsottiin johtuvan kolmesta haasteesta, jotka ovat samalla tärkeimmät tutkimuksesta opitut seikat:

- **Siirtyminen henkilöorientoituneesta asiantuntijapohjaisesta kehittämisestä moniosaaviin tiimeihin:** Kehittäjäyksikkö on tiimi yhden henkilön sijaan. Voi olla haastavaa mukautua yksilöorientoituneesta kehittämisestä osaksi tiimiä, jolla on yhteiset ongelmat.
- **Johdon roolin muuttuminen:** Johdon täytyy päättää, mitkä ovat korkean tason tavoitteet ja todelliset prioriteetit. Tutkimuksen mukaan priorisointi oli yksi kaikkein vaikeimmista asioista, sillä johdon voi olla vaikeaa myöntää, että yhtäaikaisia projekteja voi olla liian useita.
- **Sitkeyden ja päättäväisyyden puute jatkuvaan kehittymiseen:** Ensimmäisten parannusten jälkeen tiimi ja sidosryhmät voivat menettää kiinnostuksensa prosessin ja käytäntöjen ylläpitoon. Se voi johdattaa tiimin käyttämään vanhoja ja tuttuja menetelmiä.

Myös muissakin tutkimuksissa on havaittu, että siirtyminen perinteisestä sovelluskehityksestä ketterämpään kehitykseen ei ole ollut ongelmatonta. Monissa tapauksissa menetelmiä on jouduttu räätälöimään organisaation tarpeisiin, sillä suurten yritysten jäykkä byrokratia ei ole otollisin ympäristö ketterien menetelmien käyttämiselle. Siltikin suurimmassa osassa raportoiduista tapauksista tulokset ovat olleet myönteisiä ja yritykset ovat havainneet siirtymisen uuteen prosessiin hyväksi ratkaisuksi. [12]

Tämän tutkimuksen case-yrityksen tavoitteita olivat muun muassa sovellusten parempi laatu ja käyttäjäystävällisyys (yrityksen tavoitteisiin paneudutaan tarkemmin kappaleessa 3.1). Scrum, joka tähtää prosessin hallintaan, ei pysty suoraan tarjoamaan niihin ratkaisuja.

Käyttäjäystävällisyyttä voidaan parantaa hyödyntämällä käyttäjäkeskeisiä menetelmiä sovelluskehityksessä. Tiivistelmässään ketterästä käyttäjäkeskeisyydestä Scott W. Ambler kuitenkin toteaa, että käyttäjäkeskeisten ja ketterien menetelmien yhteensovittamiseen liittyy haasteita: Ketterissä menetelmissä on tavoitteena toteutuksen nopea edistäminen ja hyvä järjestelmän tekninen laatu. Käyttäjäkeskeisyyden ystävät taas haluavat painottaa sovelluksen suunnittelua siten, että käyttäjäkokemus on korkeatasoinen ja kaikkien käyttäjien tarpeet tulevat tyydytetyksi. Ketterissä menetelmissä pyritään huomioimaan kaikkien sidosryhmien tarpeet laajemmalla tasolla. Käyttäjäkeskeisten menetelmien huomion keskipiste on kuitenkin käyttäjässä, ja tavoitteena on mahdollisimman helppokäyttöisen käyttöliittymän suunnittelu ja rakentaminen. Yhteistyökykyinen organisaatio, sen eri tilanteisiin mukautuva strategia sekä itseohjautuvat ja moniosaavat tiimit ovat olennaisia ketterille menetelmille. Käytettävyyden ammattilaiset ovat sen sijaan organisaatioissa usein yhtenä joukkona, ja heidän hajauttamisensa ketteriin tiimei-

hin voi olla haastavaa johtamisen kannalta. Lisäksi, ketterissä menetelmissä suunnittelua pyritään tekemään pienissä erissä toteutuksen yhteydessä. Käyttäjäkeskeisissä menetelmissä käyttöliittymien suunnittelu halutaan yleensä hoitaa kokonaan alta pois ennen toteutuksen aloittamista. [2]

Ambler korjaa käyttäjäkeskeisten menetelmien harjoittajien on väärinymmärryksiä ketteristä menetelmistä: [2]

> ”Ketterissä menetelmissä ei mallinneta asioita.”

Väärin. Mallinnusta tehdään, mutta laajaa etukäteissuunnittelua ei suosita vaan mallintamisen käytetään ketterämpiä keinoja.

> ”Ketterissä menetelmissä sovellukset siirretään aina suoraan tuotantoon.”

Joskus näin tehdään, mutta se ei ole mikään sääntö. Yleistä on toimivan sovelluksen toimittaminen säännöllisesti omaan ympäristöönsä tai käyttäjätestaukseen muutaman viikon välein. Tuotantoonmeno tapahtuu paljon pidemmillä aikaväleillä.

> ”Käytettävyysasiantuntijalle ei ole roolia ketterissä menetelmissä.”

Rooli kyllä löytyy, mutta ketterissä menetelmissä käytetään yleisempiä rooleja, kuten kehittäjä tai asiakas.

> ”Käyttöliittymiä ei pitäisi refaktoroida.”

Käyttöliittymien refaktoroinnissa pelätään, että käyttäjät eivät pysty reagoimaan jatkuviin muutoksiin käyttöliittymässä ja käyttäjäkeskeisyys unohdetaan refaktoroitaessa. Todellisuudessa käyttöliittymän designia voidaan turvallisesti ja hitaasti muokata paremmaksi, kun käytettävyysasiantuntijat osallistuvat refaktorointiin. Käyttöliittymä muuttuu, mutta jatkuva muutos luultavasti näkyy vain käyttäjätesteihin osallistuvalle pienelle osalle käyttäjiä.

Ketteriä menetelmiä suosiva yhteisö myös kärsii käyttäjäkeskeisyyttä koskevista väärinymmärryksistä: [2]

> ”Hyvän käyttäjäkokemuksen saavuttamiseksi riittää sopiva määrä käyttöliittymää koskevia suuntaviivoja.”

Se on hyvä alku, mutta yhtenäinen käyttöliittymä ei vielä takaa käyttäjäystävällisyyttä.

> ”Läheinen työskentely sidosryhmien kanssa riittää.”

Tämä on myös hyvä alku, mutta Jokela ja Abrahamsson [13] huomasivat, että tämäkään ei vielä ensinkään takaa hyvää käytettävyyttä.

> ”Käyttäjäkeskeisyys koskee ainoastaan käyttöliittymäsuunnittelua.”

Käyttöliittymäsuunnittelu on selkeä osa käyttäjäkeskeisiä menetelmiä, mutta menetelmiin kuuluu myös käyttäjien ymmärtäminen. Hyvän käytettävyyden saavuttamiseksi täytyy ymmärtää käyttäjien tavoitteet ja tavat, joilla he käyttävät järjestelmää. Se vaatii hyvää yhteistyökykyä ja taitoa mallintaa asioita.

> ”Käyttäjäkeskeisyys vaatii kattavan etukäteissuunnittelun.”

Tästä asiasta on eri mielipiteitä, mutta useimmin sanotaan, että oikeanlaisen käyttöliittymän luominen vastaa oikeanlaisen arkkitehtuurin määrittelyä. Se usein vaatii hieman etukäteissuunnittelua, mutta ei tarkoita, että kehitystyön eri vaiheet tulisi laittaa jonoon, sillä iteraatioitakin tarvitaan.

Edellä mainittuihin haasteisiin ei ole helppoja ratkaisuja, mutta Ambler antaa joitain neuvoja menetelmien yhteensovittamiseen: [2]

- Valittuun ketterään menetelmään tulisi sisällyttää käyttäjäkeskeisten menetelmien peruskomponentteja, kuten persoonien, skenaarioiden ja käyttötapausten muodostaminen.
- Käyttäjäkeskeisten menetelmien tekniikat pitää sovittaa ketterän menetelmän elinkaareen:
  - Käyttöliittymän olennaisia osia tutkitaan ja mallinnetaan etukäteen
  - Uusien ominaisuuksien käyttöliittymät suunnitellaan useimmiten juuri ennen toteutusta. Käyttöliittymistä tehdään karkeita prototyyppejä, joita testataan käytettävyystesteillä.
  - Käyttäjätarinat muodostetaan käyttäjäkokemukseen keskittyen. Esimerkiksi tietyn ominaisuuden käyttöliittymä on pohjana yhdelle käyttäjätarinalle tai vaatimukselle.

2.7 Yhteenveto prosessimalleista

Vesiputousmalli on jäykkä perinteinen prosessimalli, joka antaa staattisen näkökulman sovelluskehitykseen. Todellisuudessa sovellusta ei rakenneta vaan sitä kehitetään, jolloin sovellus ikään kuin kasvaa [30]. Monissa lähteissä on todettu, että perinteiset mallit sopivat projekteihin, joiden vaatimukset tunnetaan, eivätkä ne muutu (ks. esim. [3, 30]). Iteratiivinen malli voidaan käsittää esimerkiksi peräkkäisinä pieninä vesiputouksina ja on askel ketterämpään suuntaan sovelluskehityksessä. Scrum on ketterä, iteratiivinen prosessimalli, jonka peruselementteihin kuuluvat: product backlog, joka on priorisoitu lista sovellukseen halutuista ominaisuuksista, potentiaalisesti valmiita sovelluksen osia tuottavat lyhyet kehityssyklit eli sprintit, sekä projektin etenemistä seuraavat burndown graafit. XP on myös ketterä menetelmä, joka kokoaa yhteen vahvan joukon sovelluskehityksen jo vanhojakin hyväksi havaittuja käytäntöjä.

Alla olevan taulukon (Taulukko 1) avulla pyritään avaamaan prosessimallien soveltuvuutta erilaisiin tarpeisiin. Taulukkoon on koottu aikaisemmissa kappaleissa käsiteltyihin prosessimalleihin liittyviä soveltuvuuskriteereitä. Erilaisille kriteereille on annettu arvosana jokaisen prosessin näkökulmasta. Arvosanat ovat asteikolla 1-4, jossa 1 on huonoin ja 4 paras. Aiemmissa kappaleissa käsiteltyjen lähteiden lisäksi taulukon muo-

dostamiseen käytettiin Robert Wysockin prosessimalleja laajasti vertailevaa kirjaa *Effective Project Management: Traditional, Agile, Extreme* [31] sekä pyydettiin asiantuntija-arvioita kolmelta case-projektiin osallistuneelta kokeneelta sovelluskehittäjältä. Taulukon arvosanoista ei voida suoraan laskea esimerkiksi keskiarvon mukaista paremmuutta, sillä jotkin kriteereistä ovat lähes toistensa vastakohtia.

Taulukko 1. Käsiteltyjen prosessimallien ominaisuuksia on arvioitu asteikolla yhdestä neljään. Suurempi arvosana on parempi.

Kriteeri	Vesiputous-malli	Iteratiivinen kehitys	Scrum
Soveltuu suuriin ojelmistoprojekteihin	3	3	4
Soveltuu monimutkaisiin tai kriittisiin järjestelmäprojekteihin	3	4	3
Soveltuu pieniin ohjelmistoprojekteihin	2	3	4
Soveltuu keskisuurille tai suurille (4 henkilöä tai enemmän) kehitystiimeille	4	4	4
Soveltuu pienille (alle 4 henkilöä) kehitystiimeille	2	3	2
Muutoksiin reagointi on helppoa	1	3	4
Tukee tilannetta, jossa useita kehitystiimejä työskentelee samassa projektissa	3	2	3
Soveltuu projekteihin, joissa vaatimukset ovat epäselvät	1	2	4
Tuottaa asiakkaalle näkyviä tuloksia nopeasti	1	3	4

Prosessimallien vertailu ei ole aivan yksiselitteistä. On hankalaa luoda yleispätevää mittaristoa tai kriteeristöä, joka sopisi vertailuun perinteisten ja ketterien menetelmien välillä. Myös esimerkiksi Scrumin ja XP:n vertailu on vaikeaa, sillä vaikka molemmat ovat ketteriä menetelmiä, ne tähtäävät sovelluskehitysprojektien hallintaan eri tavoin. Tämän vuoksi, XP:n ollessa prosessimalli joka tukee sovelluskehityksen käytäntöjä eikä niinkään prosessin hallintaa, sitä ei ole sijoitettu yllä olevaan taulukkoon.

Prosessimallien soveltuvuudesta erilaisiin tilanteisiin kiistellään, ja erityisesti yllä olevan taulukon kaksi ensimmäistä riviä ovat monitulkintaisia. Esimerkiksi vesiputousmalli soveltuu hyvin monimutkaisten tai toimintakriittisten (*mission-critical*) sovellusten kehitykseen, mikäli vaatimukset ovat hyvin selvillä. Scrumin soveltuvuudesta toimintakriittisen sovellusten kehitykseen kiistellään Internetin keskustelupalstoilla, eivätkä asiantuntija-arvioijatkaan päässeet aiheesta yksimielisyyteen. Kiistan taustalla on seuraavia väitteitä: Ketterissä menetelmissä keskitytään helposti liikaa yksittäisten tehtävien toteuttamiseen pystymättä näkemään kokonaiskuvaa, joka johtaa ongelmiin kehityksessä sovelluksen kasvaessa ja monimutkaistuessa. Scrumia puolustetaan esimerkiksi sillä, että jo jokaisen sprintin päätteeksi saadaan hyvin toimiva tuotteen osa, koska testausta ja

laadunvarmistusta tehdään jatkuvasti ja refaktoroimalla koodi pidetään jatkokehityskelpoisena.

Monien muidenkin kriteerien kohdalla on ollut vaikeaa löytää arvioinnin tueksi yksiselitteistä tukea alan kirjallisuudesta. Taulukon arvosanat ovat siten osittain subjektiivisia ja taulukon tarkoitus onkin antaa yleiskuva käsiteltyjen prosessimallien ominaisuuksista. Sopivin prosessimalli tulisi aina analysoida tapauskohtaisesti [31].

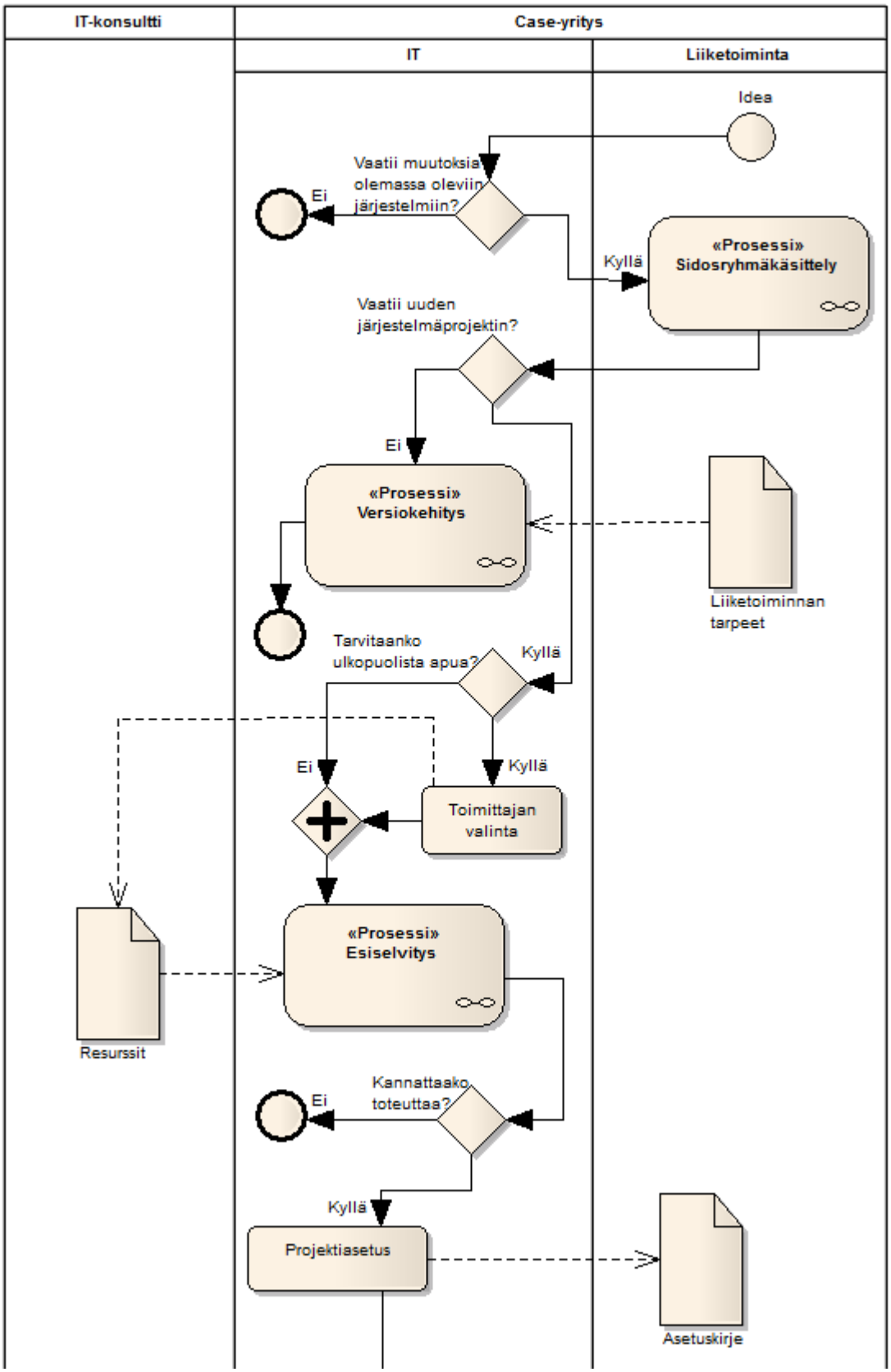
3 Case-yrityksen vanha sovelluskehitysprosessi

Yrityksen vanhaa sovelluskehitysprosessia analysoitiin product ownerin kanssa käytyjen keskusteluiden perusteella. Samoissa keskusteluissa selvitettiin yrityksen motivaatiota lähteä tekemään projekteja ketterillä menetelmillä ja taustoja siitä, miksi päädyttiin Scrumiin. Osa product ownerin kanssa käydyistä keskusteluista myös nauhoitettiin ja litteroitiin. Litteroiduista keskusteluista on nostettu esille sitaatteja product ownerin kommenteista.

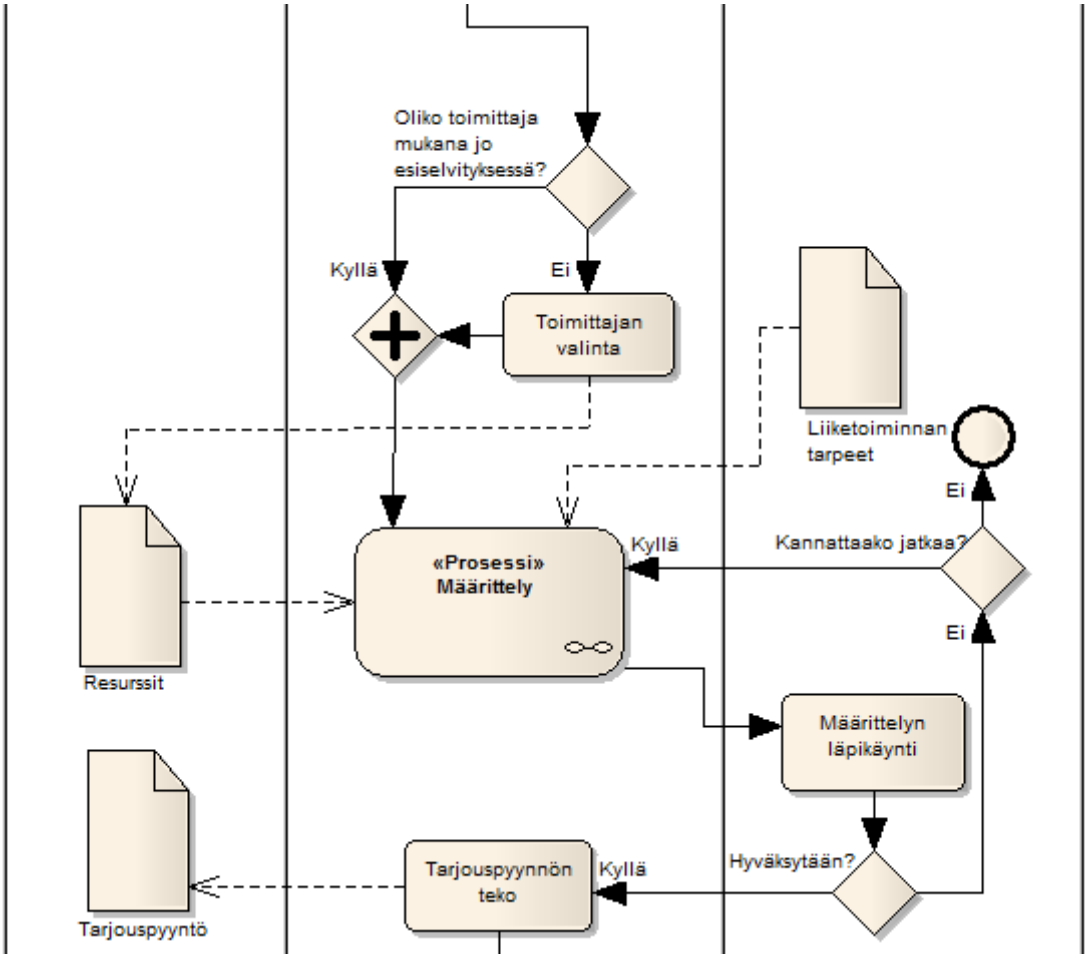
Keskusteluissa myös käytiin läpi yrityksen sovelluskehitysprosessin etenemistä. Keskusteluista tehdyistä muistiinpanoista, sekä käyttöön luovutetuista olemassa olevista dokumenteista yrityksen sovelluskehitysprosesin olennaisia vaiheita hahmoteltiin prosessikaavioiksi helpottamaan prosessien hahmottamista. Kaaviot noudattavat BPMN-notaatiota.

Suurin osa yrityksen sovelluskehitysprojekteista on noudattanut pääpiirteittäin vesiputousmallia. Vuosia sitten yrityksessä oli kokeiltu myös eräänlaista evoluutiomallia, jossa sovellusta kehitettiin inkrementaalisesti iteroiden, eli lyhyissä sykleissä sovellukseen lisättiin ominaisuuksia vähän kerrallaan. Käytetty malli ei kuitenkaan ollut tarkasti määritelty. Kokeilun tulokset eivät silloin vakuuttaneet ja vesiputousmallin käyttöä jatkettiin.

Yrityksen vanha kehitysprosessi on mallinnettu BPMN-kaaviona, joka etenee ylhäältä alaspäin ja on jaettu useammaksi kuvaksi (ks. kuvat 8-10). On hyvä huomata, että kaavioissa oleva *IT-konsultti*-sarake kuvaa ulkopuolisen konsultin aktiviteetteja, mutta konsultti ei ole aina välttämättä sama vaan se voi vaihtua eri aktiviteettien välillä.



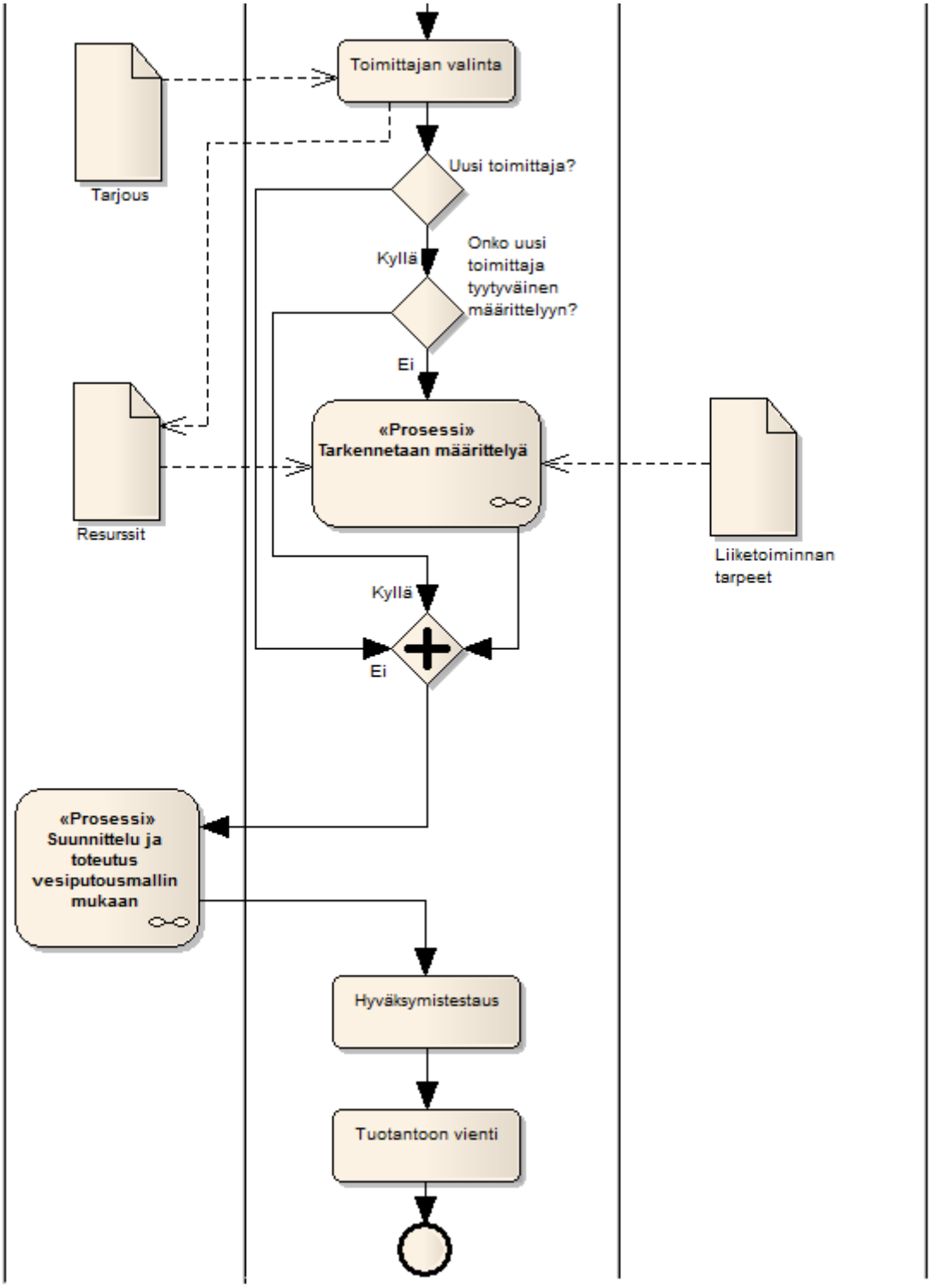
Kuva 8. BPMN-kaavio vanhasta prosessimallista, osa 1/3



Kuva 9. BPMN-kaavio vanhasta prosessimallista, osa 2/3

Alla on yhteenveto eri vaiheista, jotka kuvaavat yrityksen vanhaa tapaa tehdä uusia järjestelmiä:

1. Lyhyissä sessioissa käyttäjien ja liiketoiminnan kanssa on ensin määritelty tarve järjestelmälle
2. Tarvekuvauksen perusteella järjestelmätoimittaja on kirjoittanut ratkaisukuvausdokumentin
3. Kun ratkaisukuvaus on läpikäyty, varsinainen toteutustyö on aloitettu
4. Kun toteutus on kokonaan tehty, integrointi- ja hyväksymistestit on aloitettu



Kuva 10. BPMN-kaavio vanhasta prosessimallista, osa 3/3

3.1 Motivaatio uuteen prosessimalliin

Yrityksen liiketoimintaympäristö on innovatiivinen ja kehittyvä, siellä tapahtuu nopeita muutoksia ja uusia huomioitavia asioita ilmestyy usein. Järjestelmäkehityksen on ollut vaikea pysytellä mukana tässä tahdissa, joten yrityksessä tahdottiin etsiä keinoja parantaa järjestelmäkehitysprojektien mukautuvuutta uusiin tilanteisiin. Aikaisemmissa kehitysprojekteissa on monesti käsitelty liian suuria kokonaisuuksia, jolloin projektit ovat usein myöhästyneet ja toimitettu sovellus ei välttämättä ole enää vastannut nykyhetken tarpeisiin.

Alla olevassa listassa on product ownerin kanssa käydyistä keskusteluista johdettuna muita aiemmissa projektissa havaittuja ongelmia sekä Scrumin käyttämisestä tavoiteltavia hyötyjä. Listattujen tavoitteiden lisäksi asiakkaan tavoitteena on myös lueteltujen ongelmien ratkaisu. Tiivistetysti keskeisinä tavoitteena on tehostaa kehitysprosessia parantamalla prosessinhallintaa, tuottaa parempia sovelluksia liiketoiminnan tarpeisiin ja panostaa ihmisten huomiointiin kehitysprosessissa.

1. Ongelma: Väärin ymmärretty määrittely

   ”Oikeasti sen asian voi olla tekijä käsittänyt ihan väärin. Tekijä voi sulkeutua kammioon tekemään ohjelmaa ilman että sitä tehtyä asiaa katselmoidaan koko ajan – tekijällä voi olla ihan väärä käsitys siitä mitä käyttäjä on halunnut.”

2. Ongelma: Huono käytettävyys

   ”Sellaisia ongelmia on ollut paljon, että tekijä on käsittänyt oikein ja toteuttanut oikein, mutta käyttöliittymä on tehty tekijän näkemyksen mukaan, joka ei silloin ole vastannut ollenkaan sitä mitä käyttäjä on odottanut, eli odotukset toiminnoille ovat olleet erilaisia mitä lopputulos – lopputulos on toiminut oikein, mutta ei ole ollut käytettävä.”

3. Ongelma: Liian pitkä aika määrittelystä valmiiseen sovellukseen
4. Ongelma: Liiketoiminnan tai ympäristön muutoksiin reagointi on vaikeaa

   ”Kun versiopaketit kasvaa isoiksi, niin siinä saattaa olla tosi iso aika alkuperäisen tarvemäärittelyn, ratkaisukuvauksen kirjoittamisen ja valmiin tuotteen välillä. Tällöin ihan joka kerta tulee jotain sellaista, että käyttäjät eivät olekaan ihan varmoja siitä, että oliko tämä sellaista mitä me haluttiin. Ehkä isoimpana ongelmana tuollaisessa perinteisessä toimintamallissa ja varsinkin tämän meidän bisneksen yhteydessä on se, että tää bisnes saattaa muuttua siinä välissä.”

   ”Jos mennään niihin perusasioihin, niin mulla on monta sellaista projektia takana – lukematon määrä sellaisia projekteja itse asiassa, joissa on tehty määrittelyt, tehty vaatimusmäärittelyt, on mietitty miten se tehdään ja sit on tehty. Ja yhtään projektia ei oo sellaista, missä se lopputulos olis ollut täysin sellainen mitä oltiin alun perin ajateltu ja hahmoteltu. Siis siinä vaiheessa kun on menty hyväksymistestiin – eli kun toimittaja on käsistään luovuttanut sen, että tässä on nyt se, mitä on sovittu tehtävän – aina on ollut se tilanne, että asiakas on joutunut joustamaan siitä lopullisesta tavoitteesta.”

5. Ongelma: Puutteet toiminnallisuudessa
6. Ongelma: Huono laatu

> ”On ollut puuttuvia ominaisuuksia joita on jouduttu karsimaan sen takia ettei aika ole riittänyt. On ollut myös niin paljon virheitä, että on joutunut palauttaa sen tehtäväksi sillä tavalla että se ohjelma ei kaadu.”

7. Ongelma: Ratkaisumallin vaihtaminen kesken projektin on liian vaikeaa

> ”Yks iso juttu on se, että kehittämistä web-puolelle ei ole paljoa ollut. On ollut yksi projekti ja se tehtiin vesiputousmallin mukaan – se POC [Proof Of Concept] ei onnistunut. Jos se oltaisiin tehty Scrummaten me oltaisiin onnistuttu. Jos se olisi tehty Scrummaten, me oltaisiin aiemmin voitu sanoa, että ei tehdä näin koska tämä tapa tehdä vie ihan liikaa aikaa. Me oltais voitu luopua joistain jo tehdyistä päätöksistä paljon aikaisemmin, ongelmat olisivat tulleet esille paljon aikaisemmin. [...] Vanhanaikaisessa mallissa on tosi vaikeaa vaihtaa suuntaa nopeasti.”

8. Tavoite: Nopeampi palaute

> ”Iso syy miksi mennään tähän on se, että tekeminen pyritään pilkkomaan pienempiin aikaväleihin, jossa tarkistuspisteitä on useammin.”

9. Ongelma: Virheitä ei havaita ajoissa
10. Ongelma: Hukataan resursseja, kun korjauksia tehdään vasta myöhäisessä vaiheessa

> ”Jos huomataan pitkän koodaamisen jälkeen, että järjestelmässä on suorituskykyongelma, on siinä iso homma lähteä korjaamaan suorituskykyä siitä isosta koodimäärästä mikä on tehty. Se johtaa sellaisiin purkkaliimaratkaisuihin, että paikkaan yhtä paikkaa ja koska paikkaan sitä paikkaa joudun paikkaamaan toista paikkaa. Osan siitä mietitystä ratkaisusta joutuu heittämään roskakoriin.”

11. Ongelma: Tehdyn työn seuranta on hankalaa

> ”Toinen ongelma on se, että sitten kun se projekti on niin pitkä, niin ei sitä erkikään seuraa. En mä pysty niin pitkän ajan jälkeen seuraamaan, että millaista jälkeä se toimittaja on tehnyt. […] Jos sä otat tollaisen kolmen kuukauden pätkän ja vaikka siellä on vaan yksi resurssi, niin sä et millään pysty sanomaan [mitä on tehty]. Jos sä kysyt siltä ihmiseltä, että mitä sä oot tehny kolmen kuukauden aikana, niin ei se pysty kertomaan. Se on liian pitkä aika hahmottaa. Tämä on työmäärien seurantaan ja toimituksen laatuun liittyvä asia.”

12. Ongelma: Riittämätön kommunikointi
13. Ongelma: Urautuminen vanhoihin toimintatapoihin
14. Ongelma: Ei välttämättä havaita henkilöihin liittyviä ongelmia

> ”Organisaatioiden kehittyminen puuttuu vanhasta mallista. Hyvin harvalla yrityksellä on omaa IT-osastoa mikä tekee. Vaikka ollaan näennäisesti kumppaneita, että on olemassa se strateginen kumppani jolta sä aina ostat, niin se ei tuuletu. Vaikka ollaan hyvin tiukasti toisissa kiinni, niin ollaan silti myyjä-asiakas-

suhteessa. Niin silti ei mennä välttämättä samaan maaliin, koska sitä kanssakäymistä ei ole riittävästi.”

”Ja sit jos sitä ajatellaan ihmisten kannalta, niin silloin ei välttämättä näe sitä, että mikä on vialla ihmisissä. Että voi oikeasti kommunikoida ihmisten kanssa. Että yksittäisten resurssien kanssa voi oikeasti olla sellaisia ongelmia, mitkä ei paljastu tuollaisella tiellä, jos puhutaan kahdesta asiasta: hyvinvoinnista ja jaksamisesta ja sit siitä että miten saada joku projekti valmiiksi. Meillä oli yksi projekti aiemmin: yksi projekti myöhästyi kolme kuukautta sen takia kun yksi resurssi ei pystynyt toteuttamaan alun perin suunniteltua asiaa. Me ei oltu tilanteen päällä, koska ei oltu kommunikaatioyhteydessä.”

15. Ongelma: Asiakas joutuu luottamaan siihen, että henkilö, jota ei ole koskaan välttämättä nähnyt, tekee työnsä hyvin

    ”Mua on jossain yhteistyössä häirinnyt se, että on tekijöitä joita mä en tunne ja tapaa ollenkaan. Ne on jossain kammiossa tekemässä ja mun pitäis sit hyväksyä mitä ne tekee. Sellaisen ihmisen työtä pitäis hyväksyä mitä mä en oo välttämättä koskaan nähnyt. Eikä se oo minusta mistään kotoisin. Musta ihminen pitää tavata ja nähdä, että sulla on jonkinlainen kuva siitä miten sen ajatus juoksee.”

16. Tavoite: Projektiin osallistuvien henkilöiden motivointi ja osallistaminen

    ”Osallistaminen ja motivointi on isoja juttuja. ... meillä ollut aiemminkin projekteja missä liiketoiminta on ollut vahvasti mukana, mutta siltikin ne on olleet sellaisia IT-projekteja. [...] Monet näistä järjestelmistä mitä on tänne rakennettu on tehty siten, että me ollaan käyty käyttäjien kanssa ja sitten mä toimitan version. Käyttäjät on olleet mukana alussa ja lopussa, ja sitten ne käyttää [sovellusta] sen kummemmin osallistumatta siihen [kehitykseen].”

17. Tavoite: Liiketoimintaprosessien mallinnus osaksi uuden järjestelmän määrittelyä

    ”Uutena asiana tuli, että lähdettiin prosessien kautta kehittämään uutta järjestelmää. Sitä [määrittelyä] tehtiin eripituisissa workshopeissa: lyhyissä, joissa määriteltiin asioita ja pidemmissä, joissa piirrettiin prosessikuvauksia. Näissä workshopeissa oli käyttäjät mukana ja silloin todettiin, että tää on paras tapa tehdä määrittelyitä. […] Määrittelyvaiheen aikana tuli ehdotus, että eikö toteutusprojekti kannattaisi tehdä vähän samalla tavalla. Sitten tutustuttiin Scrumiin käymällä läpi Scrumin dokumentaatiota ja meille pidettiin lyhyt demo siitä, että mitä Scrum on ja mitä agile-menetelmät ylipäätään on.”

3.2 Siirtyminen uuteen prosessimalliin

Erään sovelluksen vesiputousmallin mukaisen laajan määrittelytyön yhteydessä oli tullut esille ajatus sovelluksen toteuttamisesta Scrumilla. Yritys päättikin ottaa Scrumin koekäyttöön kerralla uuden sovelluksen toteutusvaiheessa ilman, että Scrum-mallin käytäntöjä olisi tuotu prosessiin vähitellen. Toteutusprojekti oli siis pilottikokeilu ja mahdollisten positiivisten kokemusten seurauksena Scrum-mallia voisi alkaa käyttämään muissakin tulevissa kehitysprojekteissa.

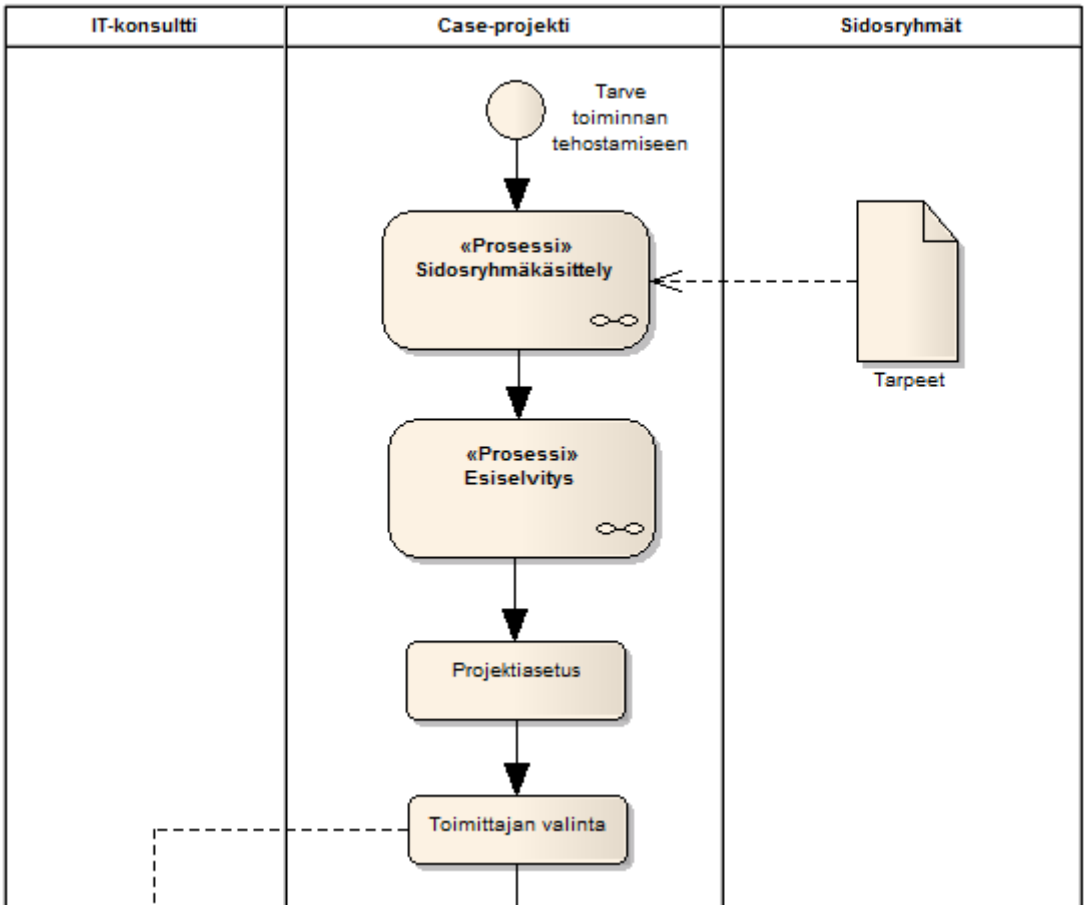
Toteutusprojekti käynnistettiin nopeasti. Uutta prosessimallia siirryttiin toteuttamaan kiireisellä aikataululla sekä asiakkaan että kehittäjien yhä tuntematta Scrum-prosessin kaikkia yksityiskohtia tarkasti. Asiakkaan ja toimittajan välillä ei ollut vielä yhteistä historiaa, eikä olemassa olevia pelisääntöjä. Lähtötilanteessa myös kehittäjien toimialatuntemus oli vähäistä. Näillä seikoilla on varmasti ollut vaikutusta projektin vaihtelevaan etenemisnopeuteen.

Yrityksen tekemä vaatimusmäärittely perustui prosessikuvauksiin ja niistä muodostettuihin käyttötapauksiin, joista varsinaisten järjestelmävaatimuksien yksiselitteinen tulkinta oli kehittäjille vaikeaa. Määrittelyn ja vaatimusten laatu oli useiden keskusteluiden aiheena ja siitä ilmeni näkemyseroja asiakasyrityksen ja kehittäjien välillä. Kehittäjät kritisoivat määrittelyitä useimmiten juuri niiden epämääräisyyden takia. Osa näkemyseroista voi selittyä asiakkaan käsityksestä siitä, mitä vaatimukset tarkoittavat. Muun muassa haastatteluista saadun ymmärryksen mukaan (ks. kappale 4.3) asiakas on käsittänyt vaatimukset lähinnä laveina rajoina sovelluksen ominaisuuksille. Kehittäjät taas puolestaan ymmärtävät vaatimukset tarkkoina kuvauksina erilaisista sovelluksen toiminnallisista ja teknisistä rajoituksista ja halutuista ominaisuuksista. Määrittelyt oli tehty toimialan ja liiketoiminnan tarpeet ennestään hyvin tuntevien henkilöiden toimesta, jolloin itsestään selviltä tuntuvia asioita oli selvästi jäänyt dokumentoimatta.

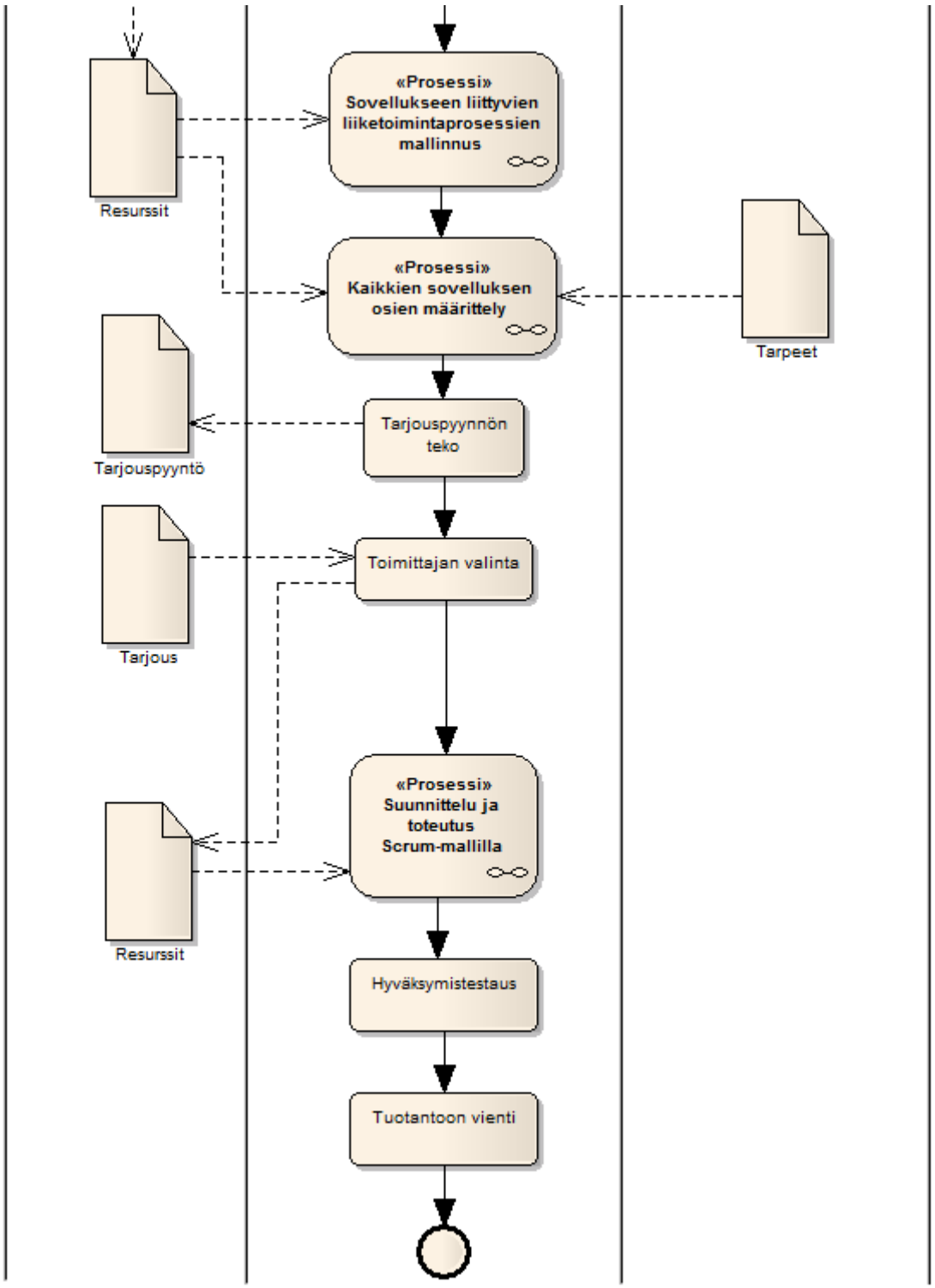
Laaja etukäteismäärittely osoittautui osittain myös turhaksi: hyvin monien ominaisuuksien määrittelyihin jouduttiin tekemään muutoksia tai vähintään tarkennuksia ennen toteutuksen aloittamista sekä vielä toteutuksen aikana. Tähän vaikutti myös toimialan ja toimintatapojen muuttuminen nopeatempoisesti; liiketoimintaympäristön elämisestä johtuvia muutoksia perustason vaatimuksiin havaittiin jo toteutusprojektin alkuvaiheessa.

Yksinkertainen malli case-projektin elinkaaresta on esitelty kuvissa 11-12. Määrittelyvaiheen loppumisen ja toteutusprojektin alkuun saattamisen välillä aikaa ehti kulua useita kuukausia, sillä varsinaisen järjestelmän toteutusta edelsi kattavan selaimella käytettävän prototyypin kehittäminen, jonka perusteella määrittelyä vielä hienosäädettiin. Tätä vaihetta kuvaan 12 ei ole merkitty, vaan sen voidaan katsoa tässä yhteydessä kuuluvan osaksi *Kaikkien sovelluksen osien määrittely* -prosessia. Yhteensä aikaa määrittelyn aloittamisen ja toteutuksen aloittamisen välillä kului lähes vuosi, mikä on aivan liian pitkä aika yrityksen nopeasti muuttuvassa liiketoimintaympäristössä.

Varsinaisen järjestelmän toteutuksen aloittamisen jälkeen Scrumin mukaisen product backlogin vajavuus vaikeutti projektin ennustettavuutta ja seurantaa pitkän aikaa. Backlogiin täydennettiin loput jo tiedossa olleet vaatimukset käyttäjätarinoina vasta seitsemän kuukautta projektin toteutusvaiheen aloittamisesta.



Kuva 11. BPMN-kaavio case-projektin etenemisestä, osa 1/2



Kuva 12. BPMN-kaavio case-projektin etenemisestä, osa 2/2

4 Case-yrityksen uusi prosessimalli – Scrum käytännössä

4.1 Tutkimuksen toteutus

Saadaksemme vastauksia tutkimuskysymyksiin *”Miten Scrum toimi kehitysprojektissa?”* ja *”Millaiseksi asiakkaan ja toimittajan välinen suhde ja viestintä koettiin Scrum-projektissa?”* sekä *”Mihin asioihin pitäisi kiinnittää huomiota tulevissa ketteriä menetelmiä käyttävissä projekteissa?”,* haluttiin projektiin osallistuneilta henkilöiltä kerätä heidän kokemuksiaan projektista.

Kvalitatiiviset tutkimusmenetelmät pyrkivät vastaamaan kysymykseen ”*Miten?*” kun kvantitatiiviset menetelmät pyrkivät vastaamaan esimerkiksi kysymyksiin ”*Mitä?*” ja ”*Kuinka paljon?*”. Siten tämän tutkimuksen kysymyksiin pääteltiin saatavan parhaiten vastauksia kvalitatiivisella tutkimusmenetelmällä. Toiseksi projektiin osallistuneiden henkilöiden määrä ei olisi ollut aivan riittävän suuri luotettavan kvantitatiivisen datan keräämiseen. Tietoja päätettiin kerätä haastattelemalla projektin avainhenkilöitä. Teemahaastattelut suoritettiin noin 7 kuukautta projektin aloittamisen jälkeen.

Teemahaastatteluihin varattiin aikaa noin tunti henkilöä kohti. Kaikki haastattelut pidettiin tämän aikarajan sisällä. Kaikki suunnitellut aihealueet saatiin käytyä läpi jokaisella kerralla, mutta haastateltavan käytännössä valitessa keskustelun aiheet, jotkin aiheet käsiteltiin toisissa haastatteluissa suppeammin tai laajemmin kuin toisissa.

Haastatteluiden tueksi oli haastattelijalla mukana kysymysrunko (Liite A), mutta haastattelut etenivät haastateltavan ehdoilla. Jos haastateltava nosti esille jonkin suunnitellun aihealueen, ohjautui keskustelu käsittelemään muitakin kyseiseen aihepiiriin suunniteltuja kysymyksiä. Haastateltavalta saatujen vastauksien päähuomioita kirjoitettiin haastattelun aikana ylös. Muistiinpanojen avulla haastattelija pystyi pitämään kirjaa jo käsitellyistä aiheista ja palaamaan kysymysrungon mukaiseen järjestykseen.

Kaikki haastattelut myös nauhoitettiin haastateltavien luvalla ja litteroitiin jälkikäteen pääpiirteittäin kysymysrungon mukaiseen taulukkoon. Taulukoista pystyttiin tämän jälkeen poimimaan tiettyä aihepiiriä koskevat vastaukset yhteen joukkoon. Vastausjoukoista voitiin havaita niissä usein esiintyvät tai kiinnostavat kommentit ja tehdä niiden avulla vastausten luokittelua. Luokittelun perusteella haastattelut purettiin tuloksiksi.

Lisäksi product ownerin kanssa pidettiin yksi erillinen keskustelu noin neljä kuukautta projektin toteutusvaiheen aloittamisen jälkeen, jolloin voitiin käsitellä Scrumista siihen mennessä opittuja asioita. Tämä keskustelu kuitenkin käsiteltiin osana varsinaisia haastatteluja.

Haastatteluaineiston tueksi kokemusten analysointiin tulivat mukaan myös tutkijan omat kokemukset projektin ajalta. Haastatteluista saadut kommentit tuettuina projektiin osallistumisen aikana saaduilla henkilökohtaisilla kokemuksilla antoivat tuloksiin enemmän syvyyttä.

4.2 Haastatellut henkilöt

Haastateltavia avainhenkilöitä oli yhteensä 11:

- 4 kehitystiimin jäsentä
- 3 käyttäjää
- 3 ohjausryhmän jäsentä
- product owner.

Sekä haastateltavien henkilöiden että yrityksen anonymiteetti haluttiin säilyttää ja tämän vuoksi henkilöiden nimiä ei mainita. Haastateltavat jaettiin yllä mainittuihin ryhmiin, jotta kommentit voidaan helpommin käsittää oikeassa kontekstissa. Haastattelut suoritettiin kuitenkin aina henkilökohtaisesti.

Product owner muodostaa poikkeuksen ryhmittelyssä, sillä hänet on ryhmitelty yksin. Hänen mielipiteensä menettäisivät merkittävyyttään tai kiinnostavuuttaan, mikäli ne jouduttaisiin kertomaan jonkin muun ryhmän kontekstissa. Tästä johtuen hänen anonymiteettinsa ei ole yhtä vahva kuin muilla haastateltavilla, mutta tästä asiasta sovittiin yhteisymmärryksessä. Product owneria lukuun ottamatta haastatellut henkilöt on numeroitu ryhmittäin ja sitaatit on koodattu kahdella kirjaimella ja henkilön numerolla. Eri sitaateista voidaan siten tietää ovatko ne saman henkilön lausumia. Esimerkiksi kehittäjät: (ke1), käyttäjät: (kä1) ja ohjausryhmäläiset: (oh1).

Kehitystiimin jäsenet

Haastateltavat kehitystiimin jäsenet ovat varsinaista sovelluskehitystä projektissa tekevät henkilöt. Kaksi haastatelluista henkilöistä oli tullut mukaan kesken projektin ja he olivat haastatteluvaiheessa työskennelleet projektissa vasta noin 2 kk. Kaikilla haastateltavilla kehittäjillä oli usean vuoden kokemus sovelluskehityksestä erilaisista, pääosin vesiputousmallilla toteutetuista, projekteista. Kaikki kehittäjät tunsivat ennestään Scrumin teorian, osalla oli jonkin verran käytännön kokemustakin. Projektin scrum master laskettiin mukaan tähän joukkoon, vaikka hän oli samalla sekä projektipäällikkö toimittajan puolelta että ohjausryhmän jäsen.

Käyttäjät

Projektissa on ollut määrittelyvaiheen alusta asti mukana pieni avainjoukko yrityksen sisäisiä käyttäjiä, jotka ovat osallistuneet kehitysprosessiin aktiivisesti. Kaikki haastatellut käyttäjät olivat aiemmin osallistuneet jollain tavalla johonkin aikaisempaan yrityksen sovelluskehitysprojektiin, lähinnä vaatimusten määrittelyyn ja testaukseen käyttöönoton yhteydessä. Käyttäjillä ei ollut aiempaa Scrum-kokemusta.

Product owner

Product owner on ollut projektissa mukana koko sen elinkaaren ajan esitutkimusvaiheesta asti. Ennen Scrum-vaiheen aloittamista ja product ownerin roolin omaksumista hän on toiminut projektipäällikkönä asiakkaan puolella useissa erilaisissa sovelluskehitysprojekteissa. Product ownerilla ei ollut aiempaa Scrum-kokemusta.

Ohjausryhmän jäsenet

Product ownerin, kehittäjien ja käyttäjien lisäksi haastateltiin myös kolmea ohjausryhmän jäsentä. Ohjausryhmä muodostui joukosta esimies- tai johtotason henkilöitä, joiden intressinä on ollut seurata projektin etenemistä. Ohjausryhmälle on projektin aikana säännöllisesti raportoitu projektin tila erillisissä tilannepalavereissa. Ohjausryhmän jäsenten haastatteluista pyrittiin selvittämään, miltä kehitysprosessi on vaikuttanut etäämpää seurattuna. Myös scrum master ja product owner kuuluivat ohjausryhmään, vaikka heitä ei siihen ryhmitelläkään tässä yhteydessä.

4.3 Henkilöiden kokemukset prosessista

4.3.1 Aiemmat toimintamallit

Vastaajat toivat aiemmista toimintamalleista esille lähinnä ongelmia. Positiivisia asioita aiemmista malleista ei juuri tuotu haastatteluissa esille.

Haastateltavat olivat käyttäneet aiemmissa projekteissaan joko vesiputousmallia, iteratiivista mallia tai jotain epäselvää mallia, joka ei noudattanut mitään määriteltyä prosessia. Osalla kehittäjistä oli myös ennestään vähän työkokemusta Scrumista ja XP:stä. Keskeisimpinä ongelmina olivat olleet aikataulun venyminen, epäolennaisiin asioihin keskittyminen tai liian suurien kokonaisuuksien tekeminen kerralla, jolloin liiketoiminnan tarpeet olivat usein ehtineet muuttua ennen sovelluksen valmistumista.

Projektien onnistumisen aiemmalla mallilla koettiin olevan vahvasti riippuvaista kehittäjien henkilökohtaisesta osaamisesta. Joitain projekteja ei oltu saatu vietyä koskaan loppuun asti. Product owner kommentoi yrityksen vanhaa toimintamallia näin:

> ”Enemmän tai vähemmän ne on aina ollut niin että on tehty jonkinlainen esitutkimus ja sit on ollut decision point jolloin päätetään jatketaanko, ja sitten on tehty vaatimusmäärittely ja määrittely. Sitten vasta kun kaikki on määritelty niin sitten on lähdetty suunnittelemaan ja toteuttamaan. Sitten on monesti ollut useamman kuukaudenkin tauko, että [asiakasyrityksen] ihmiset eivät ole osallistuneet projektiin. Se on ollut toimittajakeskeistä eli toimittaja on tehnyt suunnittelua ja toteutusta. Siellä on ollut sekä onnistuneita projekteja että epäonnistuneita projekteja. Eniten on mennyt pieleen sellaiset että on tehty isoja kokonaisuuksia ja sitten on nähty tulos vasta puolen vuoden päästä ja sitten on alettu testaa ja on todettu aika usein että eihän tää nyt ole sinne päinkään.”

Eräs ohjausryhmäläinen oli sitä mieltä, että vanhassa toimintamallissa saatiin aikataulun ja budjetin näkökulmasta luotettavammin määrittelyjä vastaavia asioita. Hänen mukaansa se ei kuitenkaan riitä: kun sovellus on saatu valmiiksi, on ollut jo tiedossa asioita, jotka ovat muuttuneet tai jotka pitäisi saada lisättyä sovellukseen. Tällöin on jouduttu tekemään erinäisiä korjailuja ja muutoksia.

4.3.2 Scrumista saadut kokemukset

Kun eri vastaajat arvioivat projektissa käytettyä prosessia asiakkaan näkökulmasta, esille tuli, että Scrumia käytettäessä aikataulu ja budjetti eivät tuntuneet olevan hyvin ennakoitavissa. Asiakas koki myös projektin tilanteen hahmottamisen vaikeaksi, koska siinä joutui vain luottamaan pitkälti projektin vetäjiin. Toinen Scrumiin liittyvistä käsityksistä liittyi ihmisiin. Ensiksi, käyttäjien mukanaoloa pidettiin positiivisena asiana, mutta negatiivisena asiana mainittiin, että käyttäjät eivät olleet riittävän varhaisessa vaiheessa mukana. Toiseksi vaatimukset tulivat käyttäjiltä vain vähän kerrallaan, mikä vaikeutti aikataulun ja kokonaisuuksien suunnittelua.

Hyvin tärkeänä nähtiin se, että projektissa pitää olla oikeat ihmiset mukana. Monet henkilöt olivat sitä mieltä, että käytetystä prosessista riippumatta minkä tahansa sovelluskehitysprojektin onnistuminen on enemmän kiinni vain oikeista henkilöistä.

Erityisesti kehittäjät olivat sitä mieltä, että Scrumia ei toteutettu projektissa puhtaasti. Vaikka vesiputousmallin mukaista määrittelyvaihetta ei huomioitaisi, prosessi nähtiin hybridimallina, johon oli sekoittunut vesiputousmallin käytäntöjä tai muita yrityksen käytäntöjä vanhoista sovelluskehitysprojekteista, kuten esimerkiksi erilliset tilanneraportit ohjausryhmälle. Vaikutusta varmasti oli myös sillä, että kaikki kehittäjätkään eivät olleet kokeneita Scrummaajia. Opittavaa oli siis puolin ja toisin.

Useimmat käyttäjät eivät hahmottaneet järjestelmien vanhan kehitystavan prosessia. Heidän kosketuspintansa sovelluskehitykseen oli tähän asti ollut lähinnä viime vaiheen käyttöliittymätestaus.

Positiivisia kokemuksia

Kaikki ohjausryhmän jäsenet olivat sitä mieltä, että katselmoinnit ja muu käyttäjien mukanaolo projektissa parantaa tilannetta vanhaan kehitysmalliin nähden. Sekä kehittäjien että product ownerin mielestä parhaimpia Scrumin puolia oli prosessin tuoma läpinäkyvyys.

Kaikki haastateltavat olivat tyytyväisiä menetelmässä jo havaittuun ketteryyteen ja nopean reagoinnin mahdollisuuteen, sillä yrityksen toimialueella muutoksia tapahtuu todella nopeasti. Samalla esille tuli kuitenkin pelko siitä, että budjetti tai aikataulu karkaa käsistä. Eräs ohjausryhmän jäsen (oh1) totesi, että vanhalla menetelmällä saadaan luotettavasti sitä mitä on määritelty, mutta se mitä saadaan, ei välttämättä olekaan sitä mitä haluttiin:

> ”Vanhassa mallissa saadaan luotettavammin määrittelyjä vastaavia asioita aikataulussa ja budjetissa, mutta negatiivisena puolena sitten kun ohjelma saadaan, on jo valmiiksi tiedossa asioita, jotka ovat muuttuneet tai jotain asioita mitä pitäisi saada lisää. Joudutaan kuitenkin tekemään erinäisiä korjailuja tai muutoksia, koska määrittelytkään harvoin ovat täydellisiä.”

Vastaajat kokivat, että Scrumin avulla sovellus voidaan todennäköisesti saada helpommin halutunlaiseksi. Ohjausryhmään kuuluvan henkilön (oh3) näkemys:

> ”...käyttäjien tietous ja tahtotila saadaan paremmin samantien. Perinteisessä menetelmissä on tärkeää että projektissa on mukana henkilöitä joilla on jo hyvin tiedossa tahtotila ja ne henkilöt ovat mukana isossa roolissa. Perinteinen projekti on kaikkein huonoin kun on tietotekniikkaorientoitunut projektipäällikkö, joka ei yhtään tiedä miten asiat käytännössä menevät bisneksessä ja käyttäjillä ei ole kykyä, kokemusta tai halua osallistua projektiin niin sitten järjestelmästä tulee vain vähän sinne päin oleva. Toki sama ongelma voi olla Scrumissakin, mutta jos käyttäjät ovat hyvin mukana niin tämä riski saadaan minimoitua.”

Lyhyen aikajänteen työn tulosten seurantaan oltiin projektissa tyytyväisiä ja product owner kommentoi asiaa näin:

> ”Jos mä haluaisin tässä projektissa seurata esimerkiksi mitä te ootte tehneet viimeisen kolmen viikon aikana, niin mä saan sen. Mun on helppo hahmottaa sellainen kokonaisuus, vaikka teitä on useampikin henkilö. Mä tiedän ennen sitä kolmea viikkoa mitä piti tehdä ja kolmen viikon jälkeen mä nään mitä on valmiina. Ja sit periaatteessa pystyn ihan suoraan kertomaan mitä on tehty.”

Yleisemmällä tasolla tämä liittyy prosessin läpinäkyvyyteen, joka koettiin projektissa hyväksi. Product ownerin:

> ”Tässä kun mennään näin tiiviissä tahdissa niin tätä ei voi tehdä falskisti – periaatteessa molempiin suuntiin. Ne paljastuu aika nopeasti, että mä en oo saanu riittävästi tietoa tän asian tekemiseen ja päinvastoin toimittaja ei ehdi oikeasti jättää huonoa työtä. Huono työ tulee esiin jo ryhmän paineesta. Melkein vois sanoa niin, että sä et voi Scrummia tehdä jos on vaan yksi tekijä.”

Prosessin kyky kehittyä projektin edetessä nähtiin erittäin hyvänä asiana. Muun muassa retrospektiivit koettiin hyödyllisinä sekä tiimihenkeä ja työmotivaatiota kasvattavina että mahdollisuutena havaita ja korjata projektin ongelmakohtia.

Prosessissa korostuva tiivis yhteistyö asiakkaan ja toimittajan ja erityisesti kehittäjien ja käyttäjien välillä koettiin erittäin positiivisena. Projektin edetessä kehittäjien liiketoimintaosaaminen kasvoi ja käyttäjät kokivat alkavansa ymmärtämään sovelluskehitystä paremmin. Vastaajien mielestä tämän seurauksena yhteistoiminta oli helpottunut ja työn laatu parantunut: kehittäjät osasivat tehdä parempia työmääräarvioita tietäessään enemmän ja käyttäjät osasivat paremmin tuoda esille toiveitaan ja kokemuksiaan sovelluksesta. Product ownerin mielestä jatkuva kanssakäyminen tuo myös esille mahdolliset ongelmat projektiryhmän henkilökemioissa, työmotivaatiossa tai yhteishengessä. Tällaisille vesiputousmallissa asiakkaalle näkymättömille ongelmille voidaan tehdä jotain, kun ne ovat helpommin havaittavissa.

Negatiivisia kokemuksia

Vaikka prosessin ketteryys koettiin pääosin erittäin positiivisena, herätti se myös negatiivisia tunteita: kehittäjät kokivat, että muutoksia tulee liikaa. Järjestelmän arkkitehtuurin ja kokonaisuuksien rakentaminen koettiin hankalaksi. Keskeisiinkin jo toteutettuihin asioihin jouduttiin tekemään muutoksia, mitkä hidastivat kehitystyötä. Kehittäjät koki-

vat projektin ketteryydessä olevan kehitettävää erityisesti (automaattisessa) testauksessa ja laadunvarmistuksessa.

Product owner koki roolinsa raskaammaksi verrattuna asiakkaan projektipäällikkönä toimimiseen aikaisemmissa vesiputousmallin mukaisesti toteutetuissa projekteissa. Hän ei ollut osannut varautua siihen, mitä product ownerina oleminen tarkoittaa. Hänen täytyi olla sitoutunut projektiin huomattavan paljon, joten aikaa muille töille jäi paljon vähemmän. Product ownerin rooli onkin Scrumissa kaikkein vaativin ja voidaan sanoa, että se vaatii melkein sataprosenttisen panostamisen kyseiseltä henkilöltä. Jos product ownerilta ei riitä aikaa projektille, on sen eteenpäinvieminen Scrum-mallilla hankalaa, jopa mahdotonta.

Tässä projektissa tiimin tai scrum masterin harteille lankesi joissain vaiheissa projektia osa product ownerin vastuista, esimerkiksi toivottujen ominaisuuksien lisäys product backlogiin ja jopa backlogin priorisointi. Jos product owner ei tunne product backlogia todella hyvin ja ole huolehtinut tehtävien priorisoinnista liiketoimintanäkökulmat huomioon ottaen, ei silloin projektissa ole välttämättä kenelläkään hyvää kuvaa siitä, millaista sovellusta ollaan kokonaisuudessaan tekemässä.

Useimmat kokivat, ettei itse Scrum-prosessilla ollut pääosin helpottavaa tai haittaavaa vaikutusta omaan työhön. Kehittäjillä sekä product ownerilla oli ollut opittavaa Scrumin käytännöistä ja prosessin pyörittämisestä. Oltiin sitä mieltä, että projektissa ei ollut käytössä parhaita mahdollisia työkaluja Scrumin ylläpitämiseen. Sovelluksen toteutuksen kannalta useimmat kehittäjät kokivat hankalaksi sopivien kokonaisuuksien pilkkoutumisen useammalle sprintille. Tehtävien priorisointiin tulisi kiinnittää tässä suhteessa huomiota. Kehittäjän (ke2) kommentti:

> ”Ei haitannu sikäli ole, mutta en kokis että juuri helpottanutkaan. Joissain asioissa haitannut, just niissä että mielellään tekis jonkin loogisen kokonaisuuden valmiiksi, mutta nyt pitää sitten katsoo että mitä taskeja siellä on ja tehdä niitä**.”**

Jotkin kehittäjistä kokivat stressiä tai paineita onnistumisesta saadessaan uusia tehtäviä, jotka olivat heidän osaamisalueensa laitamilla tai jopa ulkopuolella. Vaikka Scrumissa kannustetaankin moniosaavien tiimien muodostamiseen, ei se käytännössä ole välttämättä yksinkertaista. Yksi kehittäjistä (ke4) kommentoi asiaa näin:

> ”Ehkä se on enemmän stressiä mitä syntyy kun on aikataulupaineita [...] erityisesti jos on paljon sellaisia asioita, missä henkilö joutuu tekemään asiaa, mikä ei kuulu vahvimpaan osaamisalueeseen. Esimerkiksi jos jollain tietokantaihmisellä on hirveä kasa tietokantaan liittyviä toteutusasioita niin hän kuitenkin helposti tuntee olevansa tilanteen herra, koska tietää mitä tarvitsee tehdä vaikka töitä olisikin paljon, eikä se aiheuta silloin niin paljon stressiä.”

Scrumissa usein toteutetulla satunnaisemmalla tehtäväjaolla toimittaessa tiimillä on suuri vastuu toistensa perehdyttämisessä ja auttamisessa. Kaikki kehittäjät eivät kuitenkaan ole välttämättä halukkaita jakamaan tietojaan ja osaamistaan. Myöskään asiakas ei vält-

tämättä ole halukas maksamaan niin sanotusti kehittäjien kouluttautumisesta. Miksi maksaa kokemattomamman kehittäjän ylimääräisestä ajankäytöstä tiettyyn tehtävään jos voidaan olla lähes varmoja, että toinen aiheeseen perehtyneempi henkilö tekisi saman tehtävän esimerkiksi kolme kertaa nopeammin? Kannattaako maksaa, jos voidaan olettaa seuraavan saman aihealueen tehtävän sujuvan samalta noviisilta ehkä enää vain puolet hitaammin kuin kokeneemmalta kehittäjältä? Yhtenä perusteluna voidaan käyttää henkilöiden mahdollisuutta korvata paremmin toisiaan esimerkiksi sairaustapauksissa. Käytännössä mielivaltainen tehtävänjako todettiinkin tässä projektissa huonoksi. Toteutustyö vaati erikoisosaamista niin usealla alueella, että koettiin tehokkaammaksi jakaa tehtäviä pääasiassa olemassa olevan osaamisen mukaisesti.

Vastaajat kokivat, ett' projektinhallintaan käytettiin enemmän resursseja kuin oli odotettu. Esimerkiksi suuri osa scrum masterin työajasta kului projektissa hallinnollisiin tehtäviin ja raportointiin. Kehittäjien näkökulmasta raportointiin ja statistiikan keräämiseen kulutettiin liikaa resursseja.

Projektin seurattavuus koettiin huonoksi. Syynä tähän voi olla tottumus perinteisempään raportointiin ja se, että Scrumiin olennaisesti liittyviä burndown-graafeja ei projektissa hyödynnetty. Perinteisemmät ohjausryhmän kokouksissa esitetyt raportit eivät olleet aina riittäviä. Pelkkä prosentuaalisten tietojen esittely jonkin osa-alueen valmiudesta ei ole kertonut välttämättä tarpeeksi ja todellisesta etenemisestä oltiin epätietoisia ja huolissaan. Ohjausryhmän jäsenen (oh1) kommentti:

> ”Katselmoinnit ja käyttäjien mukanaolo parantaa tilannetta huomattavasti. Se ei varmastikaan tuo kuitenkaan ohjausryhmän tai laskun maksajan kannalta sitä turvaa, että ollaanko saavuttamassa nyt niitä päämääriä sillä budjetilla mitä suunniteltiin. Se on tällaisen menetelmän haittapuoli, että tämä helposti riistäytyy käsistä – aikataulut venyy tai budjetit venyy siitä mitä on ajateltu.”

Projektin aikataulu venyikin alkuperäisen vesiputousmallin mukaan tehdystä arviosta. Siihen vaikutti selvästi projektin edetessä määrittelyihin tulleet muutokset ja tarkennukset, joita ei mitenkään voitu ennustaa projektin alussa.

Toteutettavaksi valittu kokonaisuus koettiin liian suureksi. Haastattelujen aikaan product owner koki jäljellä olevan suuren työmäärän lannistavana:

> ”Nyt kun tätä katsoo taaksepäin, niin tää on ihan liian pitkä projekti Scrummattavaksi kokonaisuutena. […] Se on liian kuluttavaa. Se mitä mä oon kattonu, että ei pitäisi olla noin pitkää listaa tehtäviä tehtävänä. Backlogissa kun on yli kolmesataa itemiä ja siellä on sellaisia tehtäviä, mitkä on kymmenen henkilötyöpäivää. Siinä tulee sellainen henkinen olotila, että sitä ei enää hahmota ja tulee epäilys, että saadaankohan tää valmiiksi.”

Samanhenkistä keskustelua oli käyty myös kehittäjien keskuudessa erityisesti hetkinä, jolloin jo ennestään suuritöisestä toiminnallisuudesta paljastui uusia ongelmia. Ratkaisuna tähän voisivat olla osatoimitukset eli *releaset*. Product backlogista lohkaistaisiin jokin selkeä kokonaisuus omaksi release backlogikseen, johon kiinnitettäisiin huomio

osatoimituksen kehityksen ajaksi ja loput product backlogista niin sanotusti unohdettaisiin väliaikaisesti. Näin osatoimitusten kehitystä voitaisiin käsitellä erillisinä projekteina.

Projektin eteneminen

Scrum-vaiheen ulkopuolella tehty vanhan prosessimallin mukainen määrittely oli jättänyt joitakin asioita huomioimatta tai kuvaamatta riittävällä tarkkuudella. Määrittelyistä oltiin yleisesti sitä mieltä, että ne olivat matkan varrelle huomattavasti tarkentuneet. Samalla vaatimuksiinkin oli tullut joitakin muutoksia. Ohjausryhmäläisen (oh1) kommentti:

> ”Tässäkin projektissa on käynyt se, että ruokahalu kasvaa syödessä. Ehkä jossain kohtaa oltiin menossa vielä isompaan, mutta sitä pikkuisen saatiin vedettyä takaisin päin – pakostikin. Tämä on Scrumin riski, että lisäykset ja kehitys- ja ’tää olis kiva’ -jutut saadaan pidettyä hanskassa, että voidaan keskittyä olennaiseen.”

Product owner:

> ”Me ollaan prosessien kautta lähdetty määrittelemään ja me ollaan tehty liian pitkälle se määrittely valmiiksi ja se on tullut tässä projektissa erilaisina muutoksina ja lisätöinä vastaan.”

Tyytyväisiä oltiin kuitenkin siihen, että määrittelyihin tulleisiin muutoksiin tai tarkennuksiin oli prosessin ketteryyden ansiosta pystytty reagoimaan todella hyvin. Toisaalta muutoksien huomiointi ja niiden vaatima määrittelyiden tarkennus ja suunnittelu, puhumattakaan toteutuksesta, venyttivät aikataulua. Kehittäjän (ke3) kommentti:

> ”Mun mielestä se on toiminut hyvin, että tiettyjä asioita käsitellään projektin aikana. [...] On järkevä tehdä myös sellaisia projekteja, joissa tiettyjä asioita käsitellään vasta projektin aikana eikä oleteta että asiakas olisi pystynyt määrittämään kaiken tarkkaan ennalta. Miten se sitten heijastuu laskutusasioihin, niin se on oma ongelmansa.”

Tavasta aikatauluttaa projektin eteneminen sprintteihin pidettiin, ja koettiin hyvänä, että tehtävänä on pieniä selkeitä osuuksia, joille on deadline. Siihen oltiin tyytyväisiä, että katselmoinneissa voidaan havainnollistaa mihin asti sovelluksen kehitys on edennyt. Kokeilluista kahden ja kolmen viikon pituisista sprinteistä todettiin, että kolme viikkoa on sopiva aika ja kaksi viikkoa liian lyhyt. Kahdessa viikossa ei kehittäjien mielestä ehditty riittävästi paneutua toteutukseen.

Yleisesti oltiin sitä mieltä, että sovelluksen kehittäminen aloitettiin väärästä päästä eli vähiten tärkeästä toiminnallisuudesta. Syinä tähän olivat ainakin kehittäjien puutteellinen toimialatuntemus ja product backlogin puutteellinen priorisointi projektin alussa. Keskeisintä toiminnallisuutta päästiin toteuttamaan vasta usean sprintin jälkeen. Product backlogin viimeistely ja huolellisempi suunnittelu ennen ensimmäistä sprinttiä olisi voinut ehkäistä nämä ongelmat.

Kehittäjät kokivat yhteistyön product ownerin kanssa toimineen, mutta löysivät myös kehitettävää. Osa kehittäjistä sekä product owner itse olivat sitä mieltä, että product ownerin pitäisi olla helpommin saavutettavissa esimerkiksi läheisessä huoneessa, jolloin epäselvistä asioista olisi helpompi keskustella suoraan kasvotusten ja tehdä nopeita päätöksiä. Toiset kehittäjistä taas pitivät sähköpostia ja puhelinta pääsääntöisesti riittävinä yhteydenpitovälineinä, eikä product ownerin sijainti hieman kauempana haitannut heitä. Erään kehittäjän (ke4) kommentti:

> ”Esimerkkinä yksi aiempi projekti missä olin mukana, missä asiakas oli käytännössä lähes samassa tilassa ja oli ihan kävelymatkan päässä. Siellä sai aina käydä suoraan keskustelemassa, kysymässä ja varmistamassa ja esittämässä ratkaisuja, jos tuntui vaan siltä. Se auttoi huomattavasti siinä, että sitten kun iteraation lopussa katsottiin tuloksia, niin ne oli aika pitkälti sen asiakkaan mieltymysten mukaisia.”

Jotkin kehittäjistä olivat sitä mieltä, että product ownerin pitäisi huolehtia enemmän roolinsa mukaisista tärkeistä tehtävistä, kuten product backlogin ylläpidosta ja priorisoinnista. Se olisi kuitenkin ollut ajankäytöllisesti haastavaa, koska product ownerilla oli projektiroolinsa lisäksi myös paljon muita työvelvoitteita.

Sprinttiin kuuluvien tehtävien työmääräarviointien kanssa product owner oli projektin alusta lähtien tiukkana. Liian suurilta näyttäviä työmääräarvioita kutistettiin ylioptimistisesti pienemmiksi paineen alla. Tavoitehintaisen sopimuksen rajoissa piti yrittää pysyä, vaikka kehittäjille alkoi tulla tunne, että alkuperäiset arviot monien ominaisuuksien toteutuksen vaativuudesta olivat alimitoitettuja. Lisäksi oli varsin pian havaittu, että monien ominaisuuksien lisääntyvät tai tarkentuvat vaatimukset kasvattivat todellista työmäärää väistämättä. Esimerkiksi integraatio vanhoihin järjestelmiin oli alkuperäisessä määrittelyssä sivuutettu lähes kokonaan. Pian ei oltukaan enää tekemässä uutta itsenäistä järjestelmää, vaan oltiin myös muokkaamassa ja rakentamassa yrityksen keskeisten tietojärjestelmien infrastruktuuria.

Prosessin kehittyminen

Kehittäminen oli muuttunut projektin edetessä suunnitelmallisemmaksi siten, että tarkempaa suunnittelua tehtiin vielä juuri ennen toteutusta, usein käyttäjien kanssa. Myös osa käyttäjistä ja ohjausryhmän jäsenistä koki asioiden kehittyneen, mutta he eivät olleet varmoja johtuiko se vain siitä, että eri osapuolten ymmärrys käsiteltävistä asioista oli parantunut.

Eräät kehittäjistä mainitsivat, että sprintteihin otettiin haastattelujen aikaan mukaan kohtuullisempi määrä tehtäviä kuin projektin alussa. Näiden vastaajien mukaan ajankäyttö oli parantunut: turhia palavereita oli pystytty karsimaan pois ja kehittäjiltä ei tuhlautunut aikaa sprinttien suunnitteluun sprint planning-vastuun siirryttyä lähes pelkästään product ownerille ja scrum masterille. Tällä tosin saattoi olla negatiivisia seurauksia projektin myöhempiin vaiheisiin: sprintit alkoivat pääsääntöisesti epäonnistua siinä suhteessa, että tehtäviä jäi usein paljon tekemättä tai kesken. Kehittäjien itsensä suorittama tarkempi sprinttien suunnittelu otettiinkin uudestaan taas käyttöön myöhemmin, kun projektissa saatiin hengähdystauko ja ongelmiin pyrittiin löytämään ratkaisuja.

Työkalut ja käytännöt

Daily Scrum -palavereihin kehittäjät suhtautuivat erittäin positiivisesti. Niiden koettiin parantavan kehitystiimin sisäistä viestintää ja koettiin, että Daily Scrumeista kiinni pitämällä tiimin jäsenet pysyvät hyvin tilanteen tasalla ja selvillä muiden vastuulla olevien ominaisuuksien etenemisestä. Kehittäjän (ke1) kommentti:

> ”Ne tuo kommunikaatioo, keskustelua ja helpotusta muiden töihin. Sieltä sitä keskustelua viriää ja autetaan. Jos ei olis daily scrumeja niin projektipäällikön vastuulla olis katsoa mitä kukin tekee. Nyt se menee vapaamuotoisemmin mutta samalla säilyy jokin kontrolli.”

Sprinttien katselmointeja pidettiin hyvänä asiana, erityisesti kun esiteltävänä oli jotain käyttöliittymään liittyvää. Katselmointeihin oli joidenkin vastaajien ehkä varattu liian vähän aikaa, ja käyttäjien voisi olla hyvä varautua katselmointiin paremmin perehtymällä sprintin aikana toteutettuihin asioihin etukäteen mikäli mahdollista. Toivottiin myös, että katselmoinnit pystyttäisiin hoitamaan tehokkaammin ilman sivuraiteille ajautumista.

Sekä kehittäjät että product owner pitivät retrospektiivejä todella tärkeänä osana Scrum-prosessia. Suurin osa kehittäjistä ei pitänyt joidenkin retrospektiivien aikana tehtyä, inspiroivaksi tarkoitettua askartelua kovinkaan hyödyllisenä, mutta ongelmista keskustelua, hyvien ja huonojen puolien analysointia ja siitä seuraavaa määrätietoista prosessin ja käytäntöjen kehitystä pidettiin erittäin arvokkaana. Lisäksi koettiin, että tiimin vapaa yhdessäolo parantaa yhteishenkeä ja sitoutumista projektiin. Sekä tiimi että product owner olivat sitä mieltä, että projektin kaltaisessa myyjä-asiakas-suhteessa on parempi, jos retrospektiiviin osallistuvat ainoastaan kehitystiimin jäsenet, jolloin esimerkiksi ongelmista keskustelu on vapautuneempaa.

Wiki-muotoinen dokumentaatio koettiin erittäin hyväksi ja dokumentoinnin määrä koettiin pääosin sopivaksi. Siihen oltiin tyytyväisiä, että dokumentointia tapahtui projektin koko elinkaaren aikana, eikä se kasautunut pelkästään projektin loppuun, kuten perinteisiä menetelmiä käytettäessä.

Erityisesti käyttäjät kokivat prototyyppien käytön suunnittelun ja määrittelyn apuna hedelmälliseksi. Useat käyttäjistä olivat sitä mieltä, että karkeammankin tason prototyypeistä on apua uusia ominaisuuksia hahmoteltaessa. Prototyyppien tekoon ei muun kehityksen aikana haluttu käyttää liikaa resursseja, mutta siihen oli haastateltavien mielestä löydetty hyvä käytäntö. Onnistuneita prototyyppejä voitiin käyttää myös nk. määrittelydokumenttina varsinaista toteutusta tehtäessä. Kehittäjän (ke2) kommentti:

> ”Määrittelytyötähän kuitenkin tarvitsee tehdä, sitten jos sen prototyypin tekeminen ei siihen määrittelyyn kuluvaa aikaa muuta niin se on ehdottomasti hyvä asia. Prototyyppi kertoo toteuttajalle paljon enemmän kuin joku lista ranskalaisia viivoja. Jos siihen kuitenkin kuluu paljon aikaa niin se on sitten eri asia, aikahan se on joka asiassa joka ratkaisee.”

Muita havaittuja ongelmia tai haasteita

Tässäkin projektissa oli mukana useammanlaisia sovelluskehittäjiä: vanhoja ja nuoria, kokeneita ja kokemattomia. Kullakin kehittäjällä on omat periaatteensa sovelluskehitykseen. Usein erityisesti kokeneet sovelluskehittäjät ovat tottuneet tiettyyn tapaan kehittää sovelluksia ja pitävät usein näistä tavoista ja periaatteista kiinni tiukastikin.

Projektiryhmän sisällä Scrumin hyödyt ja haitat herättivät antoisia, jopa kiivaitakin, keskusteluita. Scrumin kannalta pohdittiin muun muassa sitä, että kun asioita tehdään hyvin pienissä paloissa ilman kattavaa etukäteissuunnittelua, sovelluksesta jää helposti puuttumaan kunnollinen design. Eräs kehittäjä (ke3) kommentoi asiaa haastattelussa näin:

> ”Ongelma on monta kertaa ollut siinä että ne tehtävät, taskit aika helposti tuntuu Scrumissa pilkkoutuvan sillä tavalla että se ei ole kauhean ideaalista sen ohjelmistokehityksen kannalta. Jos lähtee ideaalisemmin kehittämään ohjelmistoa, niin keskeistä on se että tehdään design. Ei niinkään toiminnallisuuden kannalta vaan siitä että millainen design syntyy.”

> ”Keskitytään liikaa toiminnallisuuteen ja se toiminnallisuus tulee sellaisena silppuna. Toki se toiminnallisuus pitää tehdä, mutta se toiminnallisuus ei tule sellaisina kokonaisuuksina siihen tuotekehitysprosessiin että se olis ideaalista kehityksen kannalta vaan se tulee sellaisena hirveänä silppuna ja ne ei muodosta sellaista kokonaisuutta että niitä olisi helppoa niputtaa tuotekehityksen kannalta järkeviin kokonaisuuksiin. [...] Tehdään sitä softaa sillä tavalla että rakennetaan yks asia sinne ja toinen tänne sen sijaan että yritettäs rakentaa moduuleja, palveluita, ja niin edelleen sinne järjestelmän sisään, jotka tietyllä tavalla ovat autonomisia kokonaisuuksia. Niitä ei synny, koska kaikki on on-demand. Siinä on myöskin se jatkuvuusongelma, että designia ei synny senkään takia, että tiettyä toiminnallisuusaluetta, joka olisi luonteva moduuli, niin sitä käy ensin yks kaveri tekemässä, sitten toinen, kolmas ja neljäs ja se design jää tekemättä. [...] Luultavasti mitään visiota toiminnallisuudesta ei synny vaan sitä asiaa katsotaan liikaa sen pienen toiminnallisuuden kannalta.”

Edellä esitellyissä kommenteissa on mielenkiintoinen periaatteellinen näkökulma sovelluskehitykseen. Yhden kehittäjän mielestä tärkeintä kehityksessä on hyvä design ja sovelluksen arkkitehtuuri, jotta tuotteen kehittäminen etenee järkevästi ja loogisesti. Toisen kehittäjän mielestä taas tärkeintä voi olla se, että edetään käyttäjäkeskeisesti pitäen halutut toiminnallisuudet etusijalla. Nämä näkökulmat ovat melko kaukana toisistaan, mutta kummankin on käytännössä pakko ottaa toinen näkökulma huomioon.

Yrityksen yhtenä ongelmana vanhoja menetelmiä käytettäessä oli ollut resurssien hukkaantuminen ja ”purkkaratkaisujen” tekeminen eli kustannusten kasvaminen, jos korjauksia on jouduttu tekemään myöhäisessä vaiheessa projektia. Tämä voidaan yhdistää Scrumissa kritisoituun hyvän designin haasteeseen.

Kysymyksellä sovelluksen designin laadusta lähestytään kysymystä sovelluskehityksen kurista. Scrumissakin on mahdollista, että aukkoja paikataan purkalla eli tehdään hätäi-

nen toteutus ja otetaan design-velkaa, jota mahdollisesti maksetaan myöhemmin jatkokehitys- tai ylläpitovaiheissa. Scrumissa on käytännössä tarpeellista, että koodia refaktoroidaan ja muutetaan tarvittavan suurelta alueelta. Se voi maksaa sillä hetkellä enemmän, mutta tulla pitkällä aikavälillä halvemmaksi, koska myöhemmissä muutoksissa ei tarvitse tuskastella huonojen design-ratkaisuiden aiheuttamien ongelmien kanssa.

Etukäteissuunnittelullakin voidaan säästää kustannuksissa ja parantaa designia, mutta se ei tapahdu väistämättä. Esimerkiksi kattavaan etukäteissuunnitteluun nojautuvaan BDUF-mallin (*Big Design Up Front*) toimivuutta on kyseenalaistettu vastaväitteellä, että kehittäjien ei ole mahdollista nähdä kehityksessä tulevia ongelmatilanteita ennalta ja huomioida niitä suunnittelussa. BDUF-mallin viedessä etukäteissuunnittelun äärimmäisyyteen voidaan kuitenkin todeta, että Scrumissakin etukäteissuunnittelua voi ja pitää tehdä. Kysymys on kustannustasapainosta: etukäteissuunnittelun kustannukset eivät saa ylittää koodin korjaamisen ja refaktoroinnin kustannuksia. Tässä projektissa designhaasteet eivät välttämättä johdu Scrumista, vaan siitä millaisia kokonaisuuksia kunkin sprintin sisältöön on päätynyt. Kritiikki on siis osuvaa: sekä Product backlogin ylläpito ja oikea priorisointi että huolellinen suunnittelu sprintti kerrallaan eteenpäin ovat Scrumissa hyvin tärkeitä.

Kun projektin aloittamisesta oli kulunut neljä kuukautta, oli asiakkaalla sellainen käsitys, että pelkkä katselmointi riittää sovelluksen ongelmien havaitsemiseen ja riittävän käyttäjäpalautteen saamiseen. Tämä kuitenkin osoittautui vääräksi luuloksi. Myöhemmin, vasta ensimmäisten hyväksymistestien aikana konkreettisesti havaittiin, että käyttäjien saama kosketus sovelluksen kehitysversioon oli ollut liian vähäistä. Heti hyväksymistestien alussa testiraportteihin jouduttiin kirjoittamaan useita muutostoiveita käyttäjiltä saatujen kommenttien mukaan. Huomattiin, että pelkkä katselmointi ja sivusta katsominen ei riitä, vaan käyttäjien pitää oikeasti käyttää sovellusta, jotta he saavat käytettävyydestä ja ominaisuuksista oikean kuvan.

Sekä kehittäjät että käyttäjät kokivat, että käyttäjien mukanaoloa projektissa voisi parantaa. Käyttäjille ei ollut varattu riittävästi aikaa testaukseen ja uuden sovelluksen kokeiluun heidän varsinaisilta työtehtäviltään. Käyttäjien (kä1, kä3) kommentteja:

> ”Työt ei ole antaneet hirveesti periksi siihen, että olisi ehtinyt työn ohessa testata ja pyöritellä kehitysympäristöä itse. Sit se olis ollu vähän vielä valmiimpaa meidän puolesta aina kun ollaan nähty. [...] Pitäisi olla varattu aikaa pyörittelyyn, mielellään jossain ihan muualla - ettei esim. puhelin ole siinä vieressä jatkuvasti soimassa tai vastaavaa.”

> ”Käyttäjä pitäisi sitouttaa järjestelmän kehitykseen, jotta käyttäjä ymmärtää että uusi järjestelmä on hyvä asia.”

Käyttäjiä voi myös olla vaikea saada osallistumaan, koska aiemmin yrityksen IT-osasto on käytännössä hoitanut kaiken uusiin sovelluksiin liittyvän. Tässä projektissa tällaista ongelmaa ei kuitenkaan tuntunut olevan, luultavasti, koska käyttäjät olivat mukana projektissa alusta asti. Product ownerin kommentti:

> ”Se vaatii enemmän osallistumista, vaatii asiakkaalta enemmän. […] Kun aiemmin järjestelmäkehitys on tehty siten, että IT huolehtii että asiat tulee valmiiksi. Nyt kun liiketoiminnalle tulee tehtäviä, mihin heidän pitää oikeasti osallistua ja tehdä, sitä voi olla joissain kohtaa vaikeaa myydä. Onneksi nyt on sellainen projekti mistä ne on innoissaan, me tehdään oikeasti käyttöliittymää ja jotain mitä ne voi käyttää eikä vaan jotain mikä on pellin alla.”

4.3.3 Viestintä ja suhteet

Asiakkaan ja toimittajan suhde koettiin avoimeksi ja välittömäksi. Käyttäjät kokivat kehittäjien olevan enemmän osa omaa organisaatiota kuin ulkopuolista yritystä. Projektin sisäisen viestinnän koettiin toimineen pääasiassa hyvin.

Viestintä käyttäjien ja kehitystiimin välillä koettiin toimivaksi ja kehittäjien fyysisen paikallaolon asiakkaan tiloissa koettiin helpottavan viestintää. Product ownerin rooli välikätenä lähes kaikissa asioissa mainittiin hankalana, sillä se saattoi synnyttää turhia viiveitä. Toisaalta product ownerin täytyy Scrumin tärkeimpänä linkkinä pitää suurin osa langoista käsissään ja pysyä parhaiten selvillä projektin tilasta. Product owner itse näki roolinsa sellaisena, että hänen tehtävänsä on viestiä kaikkien kanssa ja huolehtia siitä, että viestintä myös toimii hyvin joka suuntaan. Hyväksi koettiin, että oli löydetty yhteinen kieli ja kehittäjien viestintä on ollut ymmärrettävää. Tämä liittyy luultavasti enemmän projektin henkilöiden henkilökohtaisiin ominaisuuksiin, eikä sinänsä liity prosessiin. Prosessin eduksi nähtiin hankalan ”sähköpostichatin” väheneminen.

Yleisesti koettiin, että koko projektiin osallistuvat ihmiset sekä asiakkaan että toimittajan puolelta muodostivat kuukausien aikana oman yhteisönsä, mikä helpotti kanssakäymistä. Product ownerin kommentti:

> ”Koko ajan, kun tehdään katselmointeja ja muuta, sä opit tuntemaan ihmiset, keskustelukulttuurin, termistön ja kaiken muun – ihan selkeästi tässä tulee sellainen yhteisöllisyys mukaan, mikä puuttuu perinteisestä vesiputousmallista. Perinteisessä mallissa lähellä on projektipäällikkö ja ehkä määrittelijä, mutta toteuttaja voi olla kokonaan ulkona. Se ei kuulu siihen yhteisöön, se vaan tekee sellaisen ratkaisun mitä on kirjoitettu.”

Kasvokkaisviestintä koettiin lähes aina paremmaksi kuin sähköpostit. Kasvokkaisviestintää ja spontaaneja viestintätilanteita toivottiin enemmän. Jatkuva yhteydenpito koettiin hyvänä ja sillä saatiin minimoitua rikkinäinen puhelin -efekti. Product owner kommentoi viestintää näin:

> ”Sellaista fiilistä, että me tehdään järjestelmää mulle ja meille on paljon helpompi rakentaa, kun sä oot niitten kanssa koko ajan tekemisissä. Hyvänä esimerkkinä on nää käyttöliittymäkuviot, missä käyttäjät on alusta pitäen olleet mukana ja tehty Scrummaamalla. Ne on tehty hyvällä draivilla, käyttäjät on olleet innoissaan siitä, että ne pääsee meidän kanssa kattomaan. [...] Nyt on jo tullut käyttäjiltä asioita, mitä ne pystyy projektiryhmästä kertomaan asioita ja havainnoimaan ihmisiä, kehonkieltä, ja niin edelleen.”

Kehittäjien puolelta haastatteluissa ilmeni huoli, että loppukäyttäjiä ei oltu huomioitu riittävästi. Käyttäjien kommenteista taas ilmeni, että käyttäjien näkemykset siitä, miten he työssään haluavat toimia voi poiketa siitä, miten heidän esimiehensä näkevät miten heidän pitäisi toimia. Työntekoa yrityksessä pyrittiin ohjaamaan määrittelemällä ja uudistamalla liiketoimintaprosesseja, jotka olivat perustana myös uutta sovellusta määriteltäessä.

5 Tulosten tarkastelu

Tässä kappaleessa tarkastellaan tutkimuksen tulosten luotettavuutta ja pohditaan tavoitteiden saavuttamista, avoimeksi jääneitä kysymyksiä sekä yrityksen sovelluskehitysprosessien kehittämisen kannalta mahdollisesti tärkeitä jatkotutkimuskohteita.

Aiempien Scrum-kokemusten selvitys jäi hieman suppeaksi. Löydetystä aineistosta tehtiin kiinnostavia erillisiä havaintoja, mutta olisi ollut mielekästä saada muodostettua selkeämpi kokonaiskuva tutkimuskentästä. Vaikka yhtäläisyyksiä tähän projektiin aiemmista tutkimustuloksista löytyikin, on kirjallisuusselvityksen suppeuden vuoksi vaikeaa tehdä laajasti yleistettävissä olevia johtopäätöksiä.

Vaikka yleistettävissä oleva vertailu tämän ja aiempien tutkimusten välillä oli hankalaa, onnistui kokemusten kerääminen teemahaastatteluilla hyvin ja tutkijan läheinen osallistuminen projektiin helpotti havaintojen tekemistä. Samankaltaisten, nopeasti muuttuvassa liiketoimintaympäristössä toimivien yritysten Scrum-kokeiluihin voi tämän projektin kokemuksista ottaa hyvin oppia. Tuloksista löytyy paljon asioita, joihin kannattaa kiinnittää erityistä huomiota uusissa projekteissa.

Mahdolliset tulevaisuuden tutkimuskohteet

Yrityksen sovelluskehityksen tulevaisuutta ajatellen tutkimus jätti useita kysymyksiä auki ja myös loi joukon uusia kysymyksiä. Esimerkiksi seuraavista asioista voisi tehdä jatkotutkimuksia yrityksen sovelluskehitysprosessien kehittämiseksi:

- Toimituksen jälkeisen ylläpitovaiheen toteuttaminen ketterillä menetelmillä
  - Miten ylläpitoprosessin voisi mallintaa?
  - Liittyykö ketterien menetelmien käyttöön haasteita ylläpitovaiheessa?
- Scrumin räätälöinti
  - Onko yrityksen toimintatavoissa jotain sellaista, minkä vuoksi esimerkiksi joitain Scrumin toteutusvaiheen käytännöistä pitäisi poistaa käytöstä kaikissa projekteissa?
  - Miten XP:n käytäntöjä voitaisiin parhaiten hyödyntää yrityksen Scrum-prosessissa?
- Työkalut Scrumin käyttämiseen
  - Mitä valmiita sovelluksia tai työkaluja on olemassa, joilla voitaisiin tukea ja tehostaa Scrumin käyttöä yrityksessä?
- BDD (*Behaviour Driven Development*) eli käytöskeskeinen kehitys
  - Kehitystyö case-projektissa oli hyvin käyttöliittymäkeskeistä. Muutkin projektin käytännöt osuvat lähelle BDD:n perusperiaatteita. Voisiko yrityksessä yrittää hyödyntää BDD:tä hallitusti?
- ATDD (*Acceptance Test Driven Development*) eli hyväksymistestilähtöinen kehitys
  - Yrityksen itse tekemät ja suorittamat hyväksymistestit ovat keskeinen osa yrityksen laadunvarmistusta. Voitaisiinko yrityksessä siirtyä hyväksymistestilähtöiseen sovelluskehitykseen?

- Regressiotestaus
    - Regressiotestaus tulee kriittiseksi viimeistään sovelluksen ylläpitovaiheessa. Mitä keinoja tai työkaluja yrityksessä voitaisiin käyttää, jotta regressiotestaus saadaan tehokkaaksi, toimivaksi ja mahdollisesti automaattiseksi?
- SPI-menetelmät
    - Onko olemassa sopivia SPI-menetelmiä, joilla yrityksen sovelluskehitysprosesseja voitaisiin kehittää?
- Sopimusmallit
    - Ketterä kehitys korostaa vilpittömän yhteistyön merkitystä ja kumppanuutta. Yrityksenkin tavoitteena on pitkäaikainen kumppanuus toimittajiensa kanssa. Millaiset sopimusmallit soveltuvat parhaiten projekteihin, joissa käytetään ketteriä menetelmiä?

6 Johtopäätökset

Tutkimuksen aikana kerättyjen kokemusten ja tehtyjen haastattelujen perusteella Scrum-projekteissa törmätään usein samoihin haasteisiin. Suurimpana haasteena tässä projektissa oli Scrum-prosessin ylläpitäminen, mutta prosessin edetessä määritellyllä tavalla koettiin onnistumisia. Scrumin ei voida sanoa olevan ratkaisu kaikkiin sovelluskehitysprojekteihin, mutta se on hyvä vesiputousmallin korvaaja esimerkiksi kehitysprojektissa, jonka aikana voidaan odottaa tapahtuvan paljon muutoksia. Voidaan todeta, että Scrumiin siirtyminen voi myös parantaa toimittajan ja asiakkaan välistä viestintää vesiputousmalliin nähden. Tämän tutkimuksen tuloksia voidaan jossain määrin yleistää, mutta varauksella. Tulokset on saatu kokemuksista yhdestä projektista, johon osallistui suhteellisen pieni määrä henkilöitä.

6.1 Miten Scrum toimi case-projektissa?

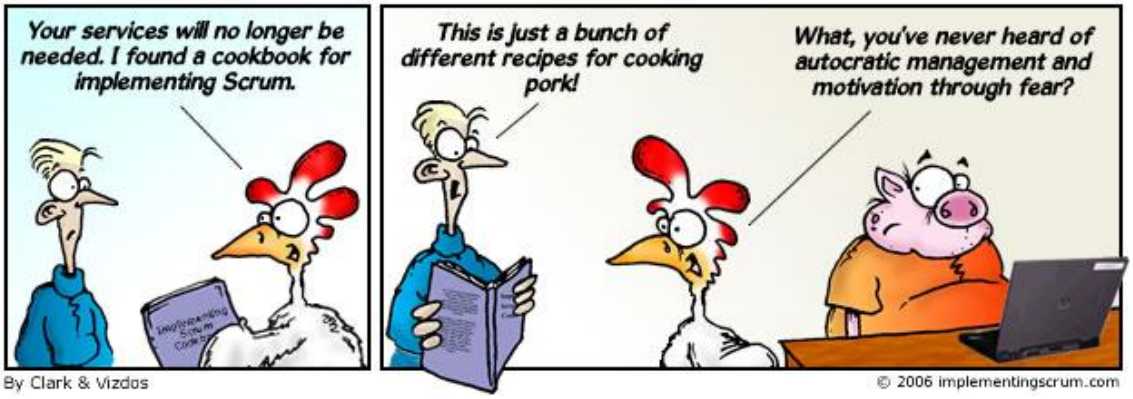
Scrumin toteuttaminen onnistui projektissa vaihtelevasti. Pääosin Scrum-mallilla kehittämiseen oltiin tyytyväisiä, mutta ongelmilta ei vältytty. Yhtäläisyyksiä vanhoihin tutkimustuloksiin Scrum-kokemuksista koettiin: esimerkiksi kaikki Marchenkon ja Abrahamssonin tekemän tutkimuksen kolmesta päähaasteesta (ks. kappale 2.6) havaittiin projektin aikana, näistä ehkä selvimpänä *”sitkeyden ja päättäväisyyden puute jatkuvaan kehittymiseen”*.

Scrum-prosessin hyödyntäminen ja ylläpitäminen vaatii kuria sekä kehittäjiltä että asiakkaalta. Projekti lopulta venyi alkuperäisestä aikataulustaan yli kolminkertaiseksi ja budjettikin kaksinkertaistui, joten projektin edetessä aikataulupaineet kasvoivat ajoittain hyvin suuriksi. Ehkä osittain näiden seikkojen seurauksena kurin ylläpitäminen välillä epäonnistui ja monet Scrumin perusrakenteet horjuivat, esimerkiksi:

- Kuluvan sprintin tehtävälistaa ei jäädytetty ja tehtäviin tuli muutoksia kesken sprinttien
- Kehittäjät eivät osallistuneet sprinttien suunnitteluun ja työmääräarviointi oli ylimalkaista
- Tuntikirjanpito tehdyistä tehtävistä oli olematonta, joten burndown-graafeista ei voitu nähdä sprinttien tai projektin edistymistä
- Käyttäjätarinoiden vaativuuksia ei enää arvioitu; yhdessä tuntikirjanpidon puuttumisen kanssa ei ollut mitään statistiikkaa jonka avulla olisi voitu ennustaa ajankohtaa välttämättömien vaatimusten täyttymiselle sovelluksen tuotantoonvientiä varten tai käyttää tietoja projektin muuhun seurantaan
- Välillä retrospektiivejä jätettiin tai jäi pitämättä, joten prosessin ongelmien kanssa vaan elettiin
- Huolimatta siitä, että käyttäjien kanssa tehtävät katselmoinnit oli koettu hyödyllisinä, myös niistä myöhemmässä vaiheessa projektia jostain syystä luovuttiin. Syy jäi tutkijalle hämärän peittoon, mutta seurauksia voidaan analysoida: Viestintään katselmointien puuttumisella tuntui ainakin olevan vaikutusta – kehittäjät pohtivat, oliko kehitys menossa oikeaan suuntaan vai ei, sillä käyttäjien kommentteja tai kokemuksia tuli kehittäjien tietoon välillä hyvin vähän.

Osa edellä luetelluista poikkeamista tiukasti noudatettuun Scrumiin oli product ownerin ja scrum masterin tietoisia valintoja, osa lienee projektiväsymyksestä johtuvia lipsahduksia Scrum-prosessin ylläpitämisessä. On vaikeaa nähdä, mitä kokonaisvaltaisia seurauksia kaikilla näillä poikkeamisilla määritellyistä Scrumin menetelmistä oli. Prosessi kuitenkin kehittyi ja muokkautui lähes jatkuvasti yrityksen ja projektin tilanteen mukaan. Esimerkiksi tuotantoon siirtymisen aikana ei noudatettu sprinttejä ollenkaan.

Tavoitteena on kuitenkin jatkaa sovelluksen kehittämistä edelleen Scrum-mallilla, ja tätä kirjoitettaessa osaan yllä olevista listan asioista on jo puututtu. Yrityksen täytyykin oppia toteuttamaan Scrumia sille sopivalla tavalla, jopa projektikohtaisesti (ks. Kuva 13).



Kuva 13. "There is no cookbook for Scrum." – Ei ole olemassa yleistä ohjekirjaa, jossa kerrottaisiin kuinka Scrumia tulee toteuttaa.

Huolimatta projektissa koetuista pienistä epäonnistumisista asiakas oli erittäin tyytyväinen Scrum-prosessiin ja siitä saatuihin kokemuksiin. Asiakas koki, että kappaleessa 3.1 esiteltyihin tavoitteisiin päästiin ja suurimpaan osaan aiempien projektien ongelmista saatiin vähintään parannuksia. Asiakas epäili, että vesiputousmallilla lopputulos olisi varmasti ollut huonompi. Epäily voi olla oikea, sillä on perusteltua olettaa (ks. kappale 2.7), että esimerkiksi projektin aikana tullutta suurta muutoksien määrää ei olisi pystytty vesiputousmallilla käsittelemään. Vesiputousmallilla sovellus olisi alkuperäisten määrittelyiden perusteella saatu toimitettua varmasti aiemmin, mutta toimituksen jälkeen olisi edessä ollut epäilemättä hyvin työläs muutostenkäsittelyprosessi. Yrityksen aiempien kokemusten perusteella se olisi voinut johtaa lopulta huonompaan ja kalliimpaan lopputulokseen.

6.2 Scrumin vaikutukset viestintään

Yksi tutkimuksen pääkysymyksistä oli selvittää Scrum-prosessin vaikutuksia sovelluskehitysprojektin viestintään, sillä sujuva ja riittävä viestintä on tärkeää niin liiketoiminnassa kuin sovelluskehityksessäkin.

Sovelluskehityksessä viestintä jää usein vaille panostuksia ja myöskään yksittäisten sovelluskehittäjien viestintätaitoihin ei välttämättä kiinnitetä riittävästi huomiota. Tähän pureutuen yksi ketterien menetelmien pääajatuksista on viestintään panostaminen.

Scrumiin liittyvät yhteiset tapahtumat, yhteinen työtila, asiakkaan ja kehitystiimin läheinen sijainti sekä jatkuva kasvokkain tapahtuva keskustelu lisäävät tiimihenkeä, saavat ihmiset tottumaan toisiinsa ja helpottavat avointa kommunikaatiota. Vaikka viestinnän laatu tässä projektissa oli vaihtelevaa, koettiin Scrum-mallin mukaisen prosessin parantaneen viestintää vanhaan menetelmään nähden. Scrum-mallista saatuja oppeja ja hyviä kokemuksia, kuten esimerkiksi daily scrumeja, oli alettu hyödyntämään yrityksen muissakin projekteissa jo tämän pilottiprojektin aikana.

6.3 Kehityskohteiden analysointi

Viimeisen tutkimuskysymyksen tavoitteena oli pohtia, mitä asioita tulevissa kehitysprojektissa tulisi huomioida. Tässä kappaleessa pohditaan tutkimuksesta esille nousseita tärkeimpiä ongelmakohtia.

Vaatimusmäärittely

Suoranaisia teknisiä vaatimuksia ei case-projektia varten oltu ennen toteutusvaihetta määritelty kuin muutamia melko yleisellä tasolla. Asiakkaalla oli monista asioista selkeä tahtotila siitä, mitä todella halutaan, mutta dokumentointi ja vaatimusmäärittely tästä tahtotilasta olivat jääneet epämääräiseksi. Esimerkiksi toisiin järjestelmiin liittyvien rajapintojen vaatimuksia ei määrittelyvaiheessa oltu mietitty juuri ollenkaan. Scrumin kannalta tämä ei olisi ollut ongelmallista. Kuviossa vaikuttivat kuitenkin myös sopimustekniset erot ennalta määriteltyjen asioiden ja uusien toiminnallisuuksien toteutuksen kustannuksissa. Tällöin vaadituissa toiminnallisuuksissa törmättiin näkemyseroihin ja päädyttiin kiistelemään esimerkiksi siitä onko jokin asia toimittajan virhe vai puute määrittelyissä.

Vaatimusmäärittelyn ja käyttöliittymäsuunnittelun voisi tehdä järjestelmällisemmin esim. käyttäjäkeskeistä metodiikkaa käyttäen:

1. Analysoidaan suunnittelun lähtökohdat ennen vaatimusmäärittelyä esimimerkiksi PACT-analyysin (*People*, *Activities, Context*, *Technologies*) avulla (ks. esim. [7]).
2. Kuvataan toiminnoista esimerkkiskenaarioita
3. Kerätään ja luokitellaan vaatimukset
    - Korkean tason vaatimukset
    - Matalan tason vaatimukset
    - Vaatimusten luokittelu toiminnallisiin ja ei-toiminnallisiin
4. Luodaan prototyyppejä
    - Tarkistetaan vaatimukset

Projektin seuranta

Projektin seurannan vuoksi tulisi huolehtia riittävällä tarkkuudella ja huolellisuudella

- käyttäjätarinoiden pisteytyksestä

- product backlogin priorisoinnista
- sprinttien suunnittelusta
- tuntikirjanpidosta sprinttien aikana.

Näiden tietojen avulla saataisiin muodostettua burndown-graafit sekä kustakin sprintistä että koko product backlogista. Niistä projektin tilannetta voitaisiin paremmin seurata ja ennustaa.

Testaus

Kehittäjän näkökulmasta sovelluksen testauksessa olisi luultavasti eniten parannettavaa tässä projektissa. Projektin aikana yksikkötestejä ei saatu rullaamaan halutulla tavalla teknisistä ongelmista johtuen. Näistä ongelmista ja aikataulupaineista johtuen myös testien automatisointi hyllytettiin. Tämän vuoksi myös varsin usein ketterissä menetelmissä käytettyä TDD:tä ei tässä projektissa paljon hyödynnetty. Tämä vaikeutti sovelluskoodin refaktorointia ja hidasti siten kehitystä sovelluksen monimutkaistuessa.

Eniten päänvaivaa sekä projektin aikana että varsinkin käyttöönoton yhteydessä tuntui aiheutuvan integraatio-ongelmista muihin yrityksen järjestelmiin. Integraatiotestien suorittamiseen ei yrityksellä ollut selkeää prosessia.

Käyttäjätarinoiden hyväksymistestit olivat monissa tapauksissa melko yksinkertaisella tasolla määrittelyiden ja vaatimusten ollessa ripoteltuna ympäri projektin teknistä wikidokumentaatiota. Projektin edetessä ja sovelluksen kasvaessa monimutkaisemmaksi, alkoi yhä useammin tulla tilanteita, joissa vaatimusten selvittäminen jonkin ominaisuuden toteutusta tai muutosta varten vaati eräänlaista salapoliisityötä.

Pelkät hyväksymistestit voisivat toimia kehittäjien referenssinä vaatimuksista, jolloin voidaan helpommin selvittää, milloin jokin käyttäjätarina tai vaatimus saadaan täytettyä. Tällöin käyttäjätarinoihin liittyvien hyväksymistestien ylläpitoa ja tarkkuutta tulisi parantaa sekä huolehtia käyttäjien ja liiketoiminnan osallistumisesta hyväksymistestien määrittelyyn ja kirjoittamiseen. Tarkkojen hyväksymistestien määrittelyn tulisi aina pysyä kehityksen edellä, jotta kehittäjät eivät ole pelkästään product ownerin tietojen (tai luulojen) varassa toteuttaessaan uusia ominaisuuksia.

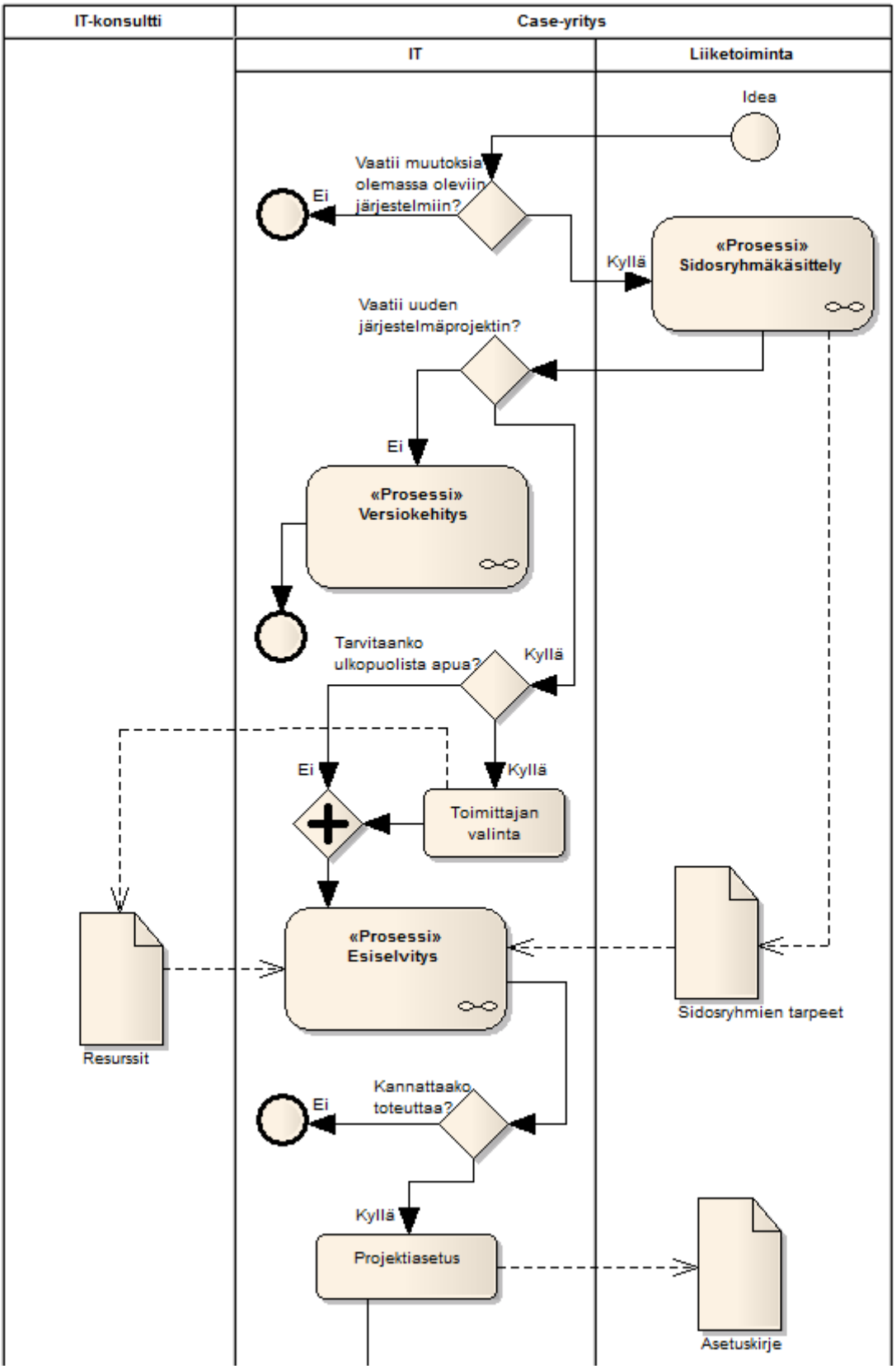
Käytettävyyteen ja käyttäjätestaukseen panostaminen

Käytettävyydestä puhuttaessa kuulee usein vihjeet *”testaa ajoissa, testaa usein”* ja *”mikäli mahdollista, testaa käyttäjien kanssa”*. Tässä projektissa kehittäjien huomio oli usein siinä, miten ominaisuudet saadaan toteutettua riittävälle tasolle mahdollisimman nopeasti ja edullisesti. Kehitystyötä pitäisi pystyä välillä jarruttamaan ja järjestää aikaa myös käytettävyyteen keskittymiseen uusien ominaisuuksien toteuttamisen kustannuksella. Kehittäjien joukossa tulisi olla käytettävyyteen erikoistuneita henkilöitä tai käytettävyyttä voitaisiin auditoida ulkoisten käytettävyysasiantuntijoiden avulla.

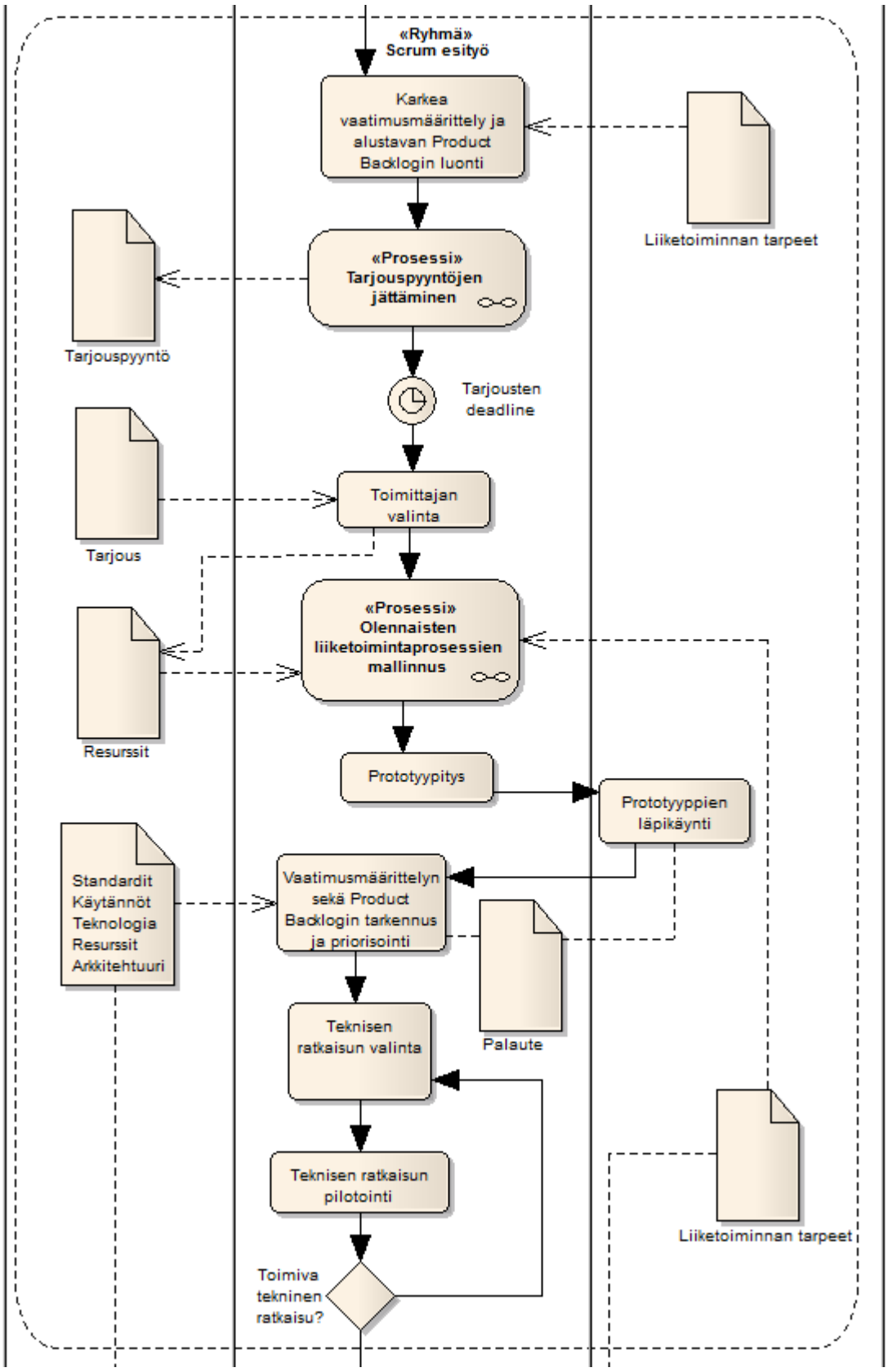
Ketterät menetelmät mahdollistavat esimerkiksi yhden viikon omistamisen käyttäjätesteille jopa kesken kehityksen. Tätä mahdollisuutta kannattaa hyödyntää potentiaalisesti vaikeiden ominaisuuksien määrittelyiden ja vaatimusten tarkentamiseen esimerkiksi (paperi)prototyyppien avulla ennen niiden toteutusta.

Ehdotus tulevaisuuden prosessimalliksi

Yrityksen toiveena oli saada ehdotus prosessimallista tulevaisuuden sovelluskehitysprojekteihin. Kuviin 14-16 on mallinnettu ehdotus yrityksen tulevaisuuden prosessimalliksi huomioiden tästä projektista saatuja kokemuksia. Ehdotuksessa sovelluskehityksen varsinainen työvaihe toteutettaisiin Scrumilla. Scrum-prosessin toteutus- ja jälkityövaiheita kuvaavia prosesseja ei ole kaavioon erikseen avattu. Scrum-prosessin esityövaihe on malliin hahmoteltu, koska siihen on sisällytetty liiketoiminnan kannalta tärkeät vaiheet eli tarjouspyyntö- ja tarjousprosessi.

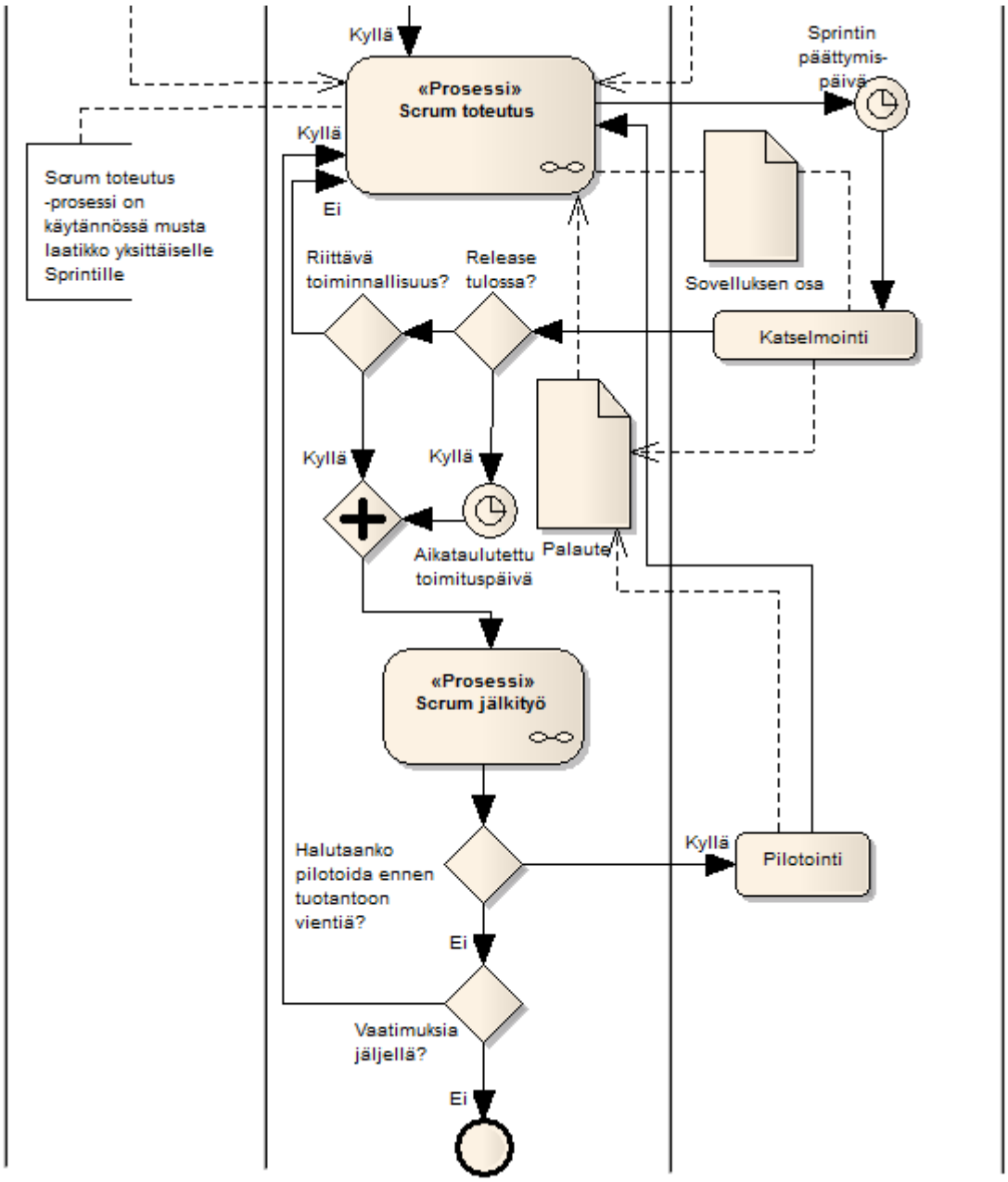


Kuva 14. BPMN-kaavio tulevaisuuden prosessimallista, osa 1/3



Kuva 15. BPMN-kaavio tulevaisuuden prosessimallista, osa 2/3

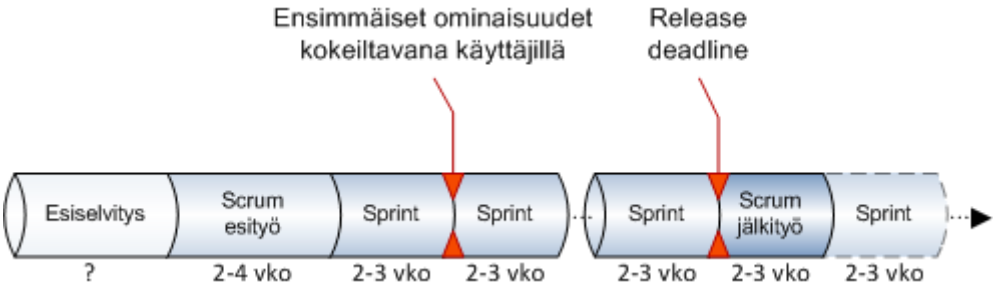
Yrityksen tavoitteena on pyrkiä kestäviin kumppanuussuhteisiin ohjelmistotoimittajien kanssa, joten usein varsinainen tarjouspyyntöprosessi pyritäänkin pitämään kevyenä. On mahdollista pyytää ainoastaan yhdeltä tai kahdelta jo hyväksi todetulta toimittajalta RFI (*Request For Information*) tai mahdollisesti RFP (*Request For Proposal*) ja toimittajan valinta hoidetaan ilman raskasta tarjouspyyntökierrosta. Saman toimittajan käyttäminen sovelluskehitysprojektin eri vaiheissa on resurssien käytönkin kannalta tehokkaampaa, eikä tietoa hukata mahdollisissa toimittajan vaihdoissa esim. määrittelyvaiheesta toteutusvaiheeseen siirryttäessä.



Kuva 16. BPMN-kaavio tulevaisuuden prosessimallista, osa 3/3

Jotta vältytään liialliselta etukäteismäärittelyltä, jolla on riski vanhentua jo ennen toteutusvaihetta, täytyy yrityksen ja toteuttajaehdokkaiden käydä asioita läpi tarkemmin ja luultavasti entistä enemmän tarjouspyyntöprosessin aikana.

Prosessien mallinnus tässä tutkimuksessa loppuu sovelluksen toimitukseen. Mahdollista ylläpitovaihetta ja sen etenemistä ei ole analysoitu.



Kuva 17. Tavoitteellinen ajankäyttö tulevaisuuden sovelluskehitysprosessissa

Kuvassa 17 esitellään tulevaisuuden prosessin eri vaiheita aikajanalle asetettuna. Kuvan mukaan tulevaisuuden tavoitteena on, että ensimmäiset sovelluksen ominaisuudet olisivat parhaassa tapauksessa käyttäjien nähtävissä ja kokeiltavissa jo kuukauden kuluttua esiselvitysvaiheen päättymisestä.

Edellä esitelty malli sopii pääpiirteissään luultavasti useimpiin yrityksen tuleviin sovelluskehitysprojekteihin. Mallia mahdollisesti käyttöön otettaessa kannattaa kuitenkin muistaa, että mikään malli ei sovi täydellisesti kaikkiin tapauksiin. Siksi projektin mahdolliset erityistarpeet kannattaa pohtia ennen prosessimallin valintaa.

Viitteet

[1] Abrahamsson, P., Salo, O., Ronkainen, J. ja Warsta J. *Agile software development methods – review and analysis.* Espoo, VTT Publications, 2002.

[2] Ambler, S. W. Introduction to agile usability: User experience activities on agile development projects. Teoksessa: E. Law, E. Hvannberg and G. Cockton, *Maturing Usability: Quality in Software, Interaction and Value*. Human-Computer Interaction -sarja, Lontoo, 2008, tiivistelmä kappaleesta 4. Viitattu 7.4.2010. Saatavissa: http://www.agilemodeling.com/essays/agileUsability.htm

[3] Augustine, S., Payne, B., Sencindiver, F. ja Woodcock, S. Agile project management: steering from the edges. *Communications of the ACM*, 2005, vol. 48, nro 12, s. 89. Viitattu 2.8.2009. DOI: 10.1145/1101779.1101781

[4] Babinet, E. ja Ramanathan, R. Dependency Management in a Large Agile Environment. Teoksessa: *Agile, 2008. AGILE ´08 Conference*, Toronto, 4.-8.8.2008, s. 401-406, 2008. Viitattu 2.8.2009. DOI: 10.1109/Agile.2008.58

[5] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. ja Thomas, D. Manifesto for Agile Software Development. Verkkodokumentti. Viitattu 3.7.2009. Saatavissa: http://agilemanifesto.org

[6] Beck, K. ja Andres, C. *Extreme Programming Explained: Embrace Change.* 2. painos. Boston, Addison-Wesley, 2004.

[7] Benyon, D., Turner, P. ja Turner, S. *Designing Interactive Systems : People, Activities, Contexts, Technologies.* Harlow, Pearson Education Limited, 2005.

[8] Boehm, B. and Turner, R. B. *Balancing Agility and Discipline: A Guide for the Perplexed.* Boston, Addison-Wesley, 2003.

[11] Dybå, T. What Do We Know about Agile Software Development? *IEEE Software*, 2009, vol. 26, nro 5, s. 6-9. Viitattu 3.8.2009. DOI: 10.1109/MS.2009.145

[10] Fowler, M. The new methodology. Verkkodokumentti. Päivitetty 13.12.2005. Viitattu 2.12.2009. Saatavissa: http://martinfowler.com/articles/newMethodology.html

[11] Highsmith, J. A. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York, Dorset House Publishing, 2000.

[12] Jakobsen, C. R. ja Sutherland, J. Scrum and CMMI. Teoksessa: *2009 Agile Conference*, Chigago, 24.-28.8.2009, s. 333-337, 2009. Viitattu: 4.8.2009. DOI: 10.1109/AGILE.2009.31

[13] Jokela, T. ja Abrahamsson, P. Usability assessment of an extreme programming project: Close co-operation with the customer does not equal to good usability. Lecture Notes in Computer Science, vol. 3009, s. 393, 2004.

[14] Leffingwell, D. ja Widrig, D. *Managing Software Requirements: A use Case Approach*. Boston, Addison-Wesley, 2003.

[15] Lindvall, M. Agile software development in large organizations. *Computer*, 2004, vol. 37, nro 12, s. 26-34. Viitattu 2.12.2009. DOI: 10.1109/MC.2004.231

[16] Marchenko, A. ja Abrahamsson, P. Scrum in a multiproject environment: An ethnographically-inspired case study on the adoption challenges. Teoksessa: *Agile, 2008. AGILE ´08 Conference*, Toronto, 4.-8.8.2008, s. 15-26. Viitattu 2.12.2009. DOI: 10.1109/Agile.2008.77

[17] Norfolk, D. Over the waterfall. Verkkodokumentti. Päivitetty 23.2.2006. Viitattu 2.12.2009. Saatavissa: http://www.theregister.co.uk/2006/02/23/waterfall_method/page2.html

[18] Poimala, S., Heikniemi, J. ja Blåfield, H. Ketterät käytännöt. Verkkodokumentti. Viitattu 7.3.2010. Saatavissa: http://www.ketteratkaytannot.fi

[19] Ramsin, R. ja Paige, R. F. Process-Centered Review of Object Oriented Software Development Methodologies. *ACM Computing Surveys*, vol. 40, nro 1, s. 1-89, 2008. Viitattu 7.3.2010. DOI: 10.1145/1322432.1322435

[20] Rayhan, S. H.ja Haque, N. Incremental adoption of scrum for successful delivery of an IT project in a remote setup. Teoksessa: *Agile, 2008. AGILE ´08 Conference*, Toronto, 4.-8.8.2008, s. 351-355. Viitattu 6.3.2010. DOI: 10.1109/Agile.2008.98

[21] Schach, S. R. *Object-Oriented and Classical Software Engineering*. Boston, McGraw-Hill, 2005.

[22] Schwaber, K. *Agile Project Management with Scrum*. USA, Redmond, Microsoft Press, 2004.

[23] Schwaber, K. ja Beedle, M. *Agile Software Development with Scrum*. Upper Saddle River, Prentice-Hall, 2002.

[24] Schwaber, K. ja Mar, K. Scrum with XP. Verkkodokumentti. Päivitetty 22.3.2002. Viitattu 27.11.2009. Saatavissa: http://www.informit.com/articles/article.aspx?p=26057

[25] Scrum alliance. Scrum roles. Verkkodokumentti. Viitattu 28.2.2010. Saatavissa: http://www.scrumalliance.org/pages/scrum_roles

[26] Shore, J. Beyond Story Cards: Agile Requirements Collaboration. Verkkodokumentti. Päivitetty 21.3.2005. Viitattu 2.12.2009. Saatavissa: http://jamesshore.com/Presentations/Beyond%20Story%20Cards.html

[27] Sommerville, I. *Software Engineering*. 7. painos. Harlow, Essex, Iso-Britannia, Pearson Education Limited, 2004.

[28] Stevens, P. Prioritizing the product backlog. Verkkodokumentti. Päivitetty 8.10.2008. Viitattu 2.12.2009. Saatavissa: http://agilesoftwaredevelopment.com/blog/peterstev/prioritizing-product-backlog

[29] Sutherland, J. Agile can scale: Inventing and reinventing SCRUM in five companies. *Cutter IT Journal*, verkkolehti, joulukuu, nro 14, s. 5-11, 2001. Viitattu 27.11.2009. Saatavissa: http://codespeak.net/svn/pypy/tag/funding-final/agile_can_scale.pdf

[30] van Vliet, H. *Software Engineering: Principles and Practice*. 3. painos. Wiley Publishing, 2008.

[31] Wysocki, R. *Effective Project Management: Traditional, Agile, Extreme*. 5. painos. Indianapolis, Wiley Publishing, 2009.

Liite A Haastatteluiden tukimateriaali

Tässä liitteessä on esitelty materiaali, joka tutkijalla oli tukenaan jokaisessa haastattelussa. Ensimmäisenä on aloituspuhe, jossa tutkija pohjusti haastateltavalle haastattelun taustaa. Toisena on taulukko aiheista ja kysymyksistä, joita tutkija käytti muistilistanaan haastatteluissa. Taulukon oikeassa reunassa olevat sarakkeet kuvaavat haastateltavia ryhmiä. Tummalla pohjalla oleva numero 1 haastateltavan ryhmän sarakkeessa tarkoitti, että rivillä oleva aihe oli tarkoitus käydä läpi haastateltavan kanssa.

Aloituspuhe

Minun diplomityössäni tutkitaan täällä [yrityksessä], [tässä] projektissa sovelluskehityksessä käytettyjä menetelmiä ja sovelluskehitysprosessia. Teen kirjallisuustutkimuksen ja omien havaintojeni lisäksi tällaiset haastattelut useille projektiin osallistuneille henkilöille. Haastattelujen avulla on tarkoitus selvittää, miltä projektiin osallistuneista ihmisistä uudet menetelmät ovat oikeasti tuntuneet ja sitä kautta mahdollisesti löytää kehitettäviä asioita ja vaikka uusia ideoitakin. Siitä ei tarvitse huolestua, jos jotkin kysymistäni kysymyksistä kuulostavat vaikeilta – minultakin voi kysyä, että mitä tarkoitan, jos esimerkiksi käytän jotain termejä mitä et ymmärrä.

Tarkoitukseni on nauhoittaa tämä keskustelu, jotta voin myöhemmin poimia tarkemmin olennaiset asiat. Haastatteluaineistoa ei tule käsittelemään kukaan muu kuin minä. Kaikki mitä tulee paperille tulee täysin anonyymisti. Sopiiko nauhoitus?

Toivon, että puhut mahdollisimman vähän tämän haastattelun sisällöstä muille haastateltaville, kunnes olen saanut kaikki henkilöt haastateltua.

Kysymysrunko ja aiheideat

				Kenelle kysymys voidaan esittää?				
Järjestys	TK #	Aihealue	Kysymys	Kehittäjä	Scrum Master	Product Owner	OhRy jäsen	Käyttäjä
0	0	Alkukysymykset	Kerro lyhyesti mitä teet työksesi?	1	1	1	1	1
0	0	Alkukysymykset	Kuinka kauan olet ollut näissä tehtävissäsi?	1	1	1	1	1
0	0	Alkukysymykset	Miten usein olet osallistunut tällaisiin sovelluskehitysprojekteihin?	1	1	1	1	1

Prosessi, Scrum								
1	2	Prosessi	Kuvaile lyhyesti (oma näkemyksesi) vanha toimintamalli sovelluskehitysprojekteissanne.			1	1	1
2	2	Scrum, Prosessi	Kuvaile käsityksesi tässä projektissa käytettävästä prosessimallista, Scrumista.	1	1	1	1	1
3	2	Scrum	Kuinka Scrum-malli on mielestäsi toiminut?	1	1	1	1	1
3,1	2	Scrum, Prosessi	Mitkä mielestäsi ovat Scrumin suurimmat hyödyt? (Oliko mielestäsi asioita, joita oltaisiin voitu tehdä enemmän/paremmin Scrumin periaatteiden mukaisesti?)	1	1	1	1	1
3,11	2,4	Scrum	Entä Scrumin huonot puolet? Onko projektissa ollut Scrumiin liittyviä käytäntöjä, mitkä olisi mielestäsi ollut parempi unohtaa tai muokata ne sopivammiksi omaan käyttöön?	1	1	1	1	
4	2	Prosessi	Mikä tässä toimintamallissa on ollut parasta, jos vertaat aikaisempiin sovellusprojekteihin? (Miltä käyttäjien mukanaolo projektissa on tuntunut? Käyttäjäystävällisyys, käyttäjäkeskeisyys?)	1	1	1	1	1
5	2	Prosessi	Mitkä asiat ovat vanhassa toimintamallissa olleet mielestäsi paremmin?	1	1	1	1	1
5,1	2	Prosessi	Oletko kokenut Scrumin haittaavan tai helpottavan omaa työtäsi?	1	1	1	1	1
5,11	2	Prosessi	Onko työnteko ollut tällä mallilla joustavaa? (Onko eroa aikaisempiin projekteihin?)	1	1	1		
5,12	2	Prosessi	Onko projekti edennyt mielestäsi oikeassa järjestyksessä? (backlogien priorisointi)	1	1	1	1	1
5,5	2,4	Käytäntö, Suunnittelu	Mitä mieltä olet suunnittelupalavereista (Backlog, Sprint)? (ajankäyttö, hyödyllisyys, mukavuus, onko kehitystä tapahtunut?, parannusehdotukset)	1	1	1		
5,6	2,4	Scrum	Mitä mieltä olet Backlogien käytöstä? (Hyödyllisyys? Onko backlogien käsittely ollut helppoa? Onko työkaluja ollut helppo käyttää? Miten backlogien käsittelyä voisi parantaa? Pitäisikö käyttää vain yhtä työkalua backlogien hallintaan? Kannattaisiko ”paperilaput seinällä” -mallia kokeilla kunnolla?)	1	1	1	1	

Projektinhallinta								
5,7	2	Projektinhallinta	Onko projektinhallinta toiminut projektissa? (Onko projektinhallintaan kulunut normaalia enemmän resursseja? Voisiko asialle tehdä jotain?)	1	1	1	1	
5,71	2,4	Scrum	Onko projektin etenemistä ja aikataulussa pysymistä ollut helppoa seurata? (Toimivatko burndown graafit? Miten asioita olisi voitu tehdä paremmin?)	1	1	1	1	1
5,8	4	Prosessi, Käytäntö	Ovatko projektin vaatimukset olleet mielestäsi selkeät? (Onko vaatimuksiin tullut paljon muutoksia projektin aikana? Ovatko vaatimukset nyt mielestäsi selkeät?)	1	1	1		
5,81	2,4	Scrum, Prosessi	Onko vaatimusten muutoksiin pystytty reagoimaan projektissa hyvin? (Kuvaile käsityksesi muutoksien hallinnasta tässä projektissa? Onko muutokset käsitelty aina oikein / em. tavalla? Onko sinulla parannusehdotuksia muutoksien hallintaan?)	1	1	1	1	
Sprintit								
6	2	Sprintit	Mitä mieltä olet Sprinteistä? (periaate, että tuotetaan ja toimitetaan jatkuvasti toimivia palikoita tasaisin väliajoin -> onko tapa hyvä?)	1	1	1		1
6,1	2,4	Scrum, Sprintit	Ovatko Sprintit olleet sopivan pituisia? Mikä olisi mielestäsi sopivin pituus Sprinteille?	1	1	1		1
6,1	2,4	Scrum	Mitä mieltä olet Daily Scrumeista? (Kevyitä/raskaita? Käsitelläänkö oikeita asioita? Pituus? Hyödyllisyys?)	1	1			
7,0	2,4	Katselmoinnit, Sprintit	Mitä mieltä olet Sprinttien katselmoinneista? (hyödyllisyys, hyvät puolet, huonot puolet, parannusehdotukset)	1	1	1		1
7,5	2,4	Scrum, Retrospectivet	Mitä mieltä olet retrospektiiveistä? (Mikä parasta, Hyödyllisyys, Product Ownerin osallistuminen)	1	1	1		

Käytäntö								
8,9	2,4	Scrum, Käytäntö	Millaiset kokemukset / tuntuma sinulla on Product Ownerin roolista tässä projektissa?	1	1	1		
9	2,4	Prosessi, Käytäntö	Onko Product Owner ollut riittävästi käytettävissä projektiin liittyvissä asioissa?	1	1	1		1
10	2,4	Prosessi	Onko prosessissa tai menetelmissä tapahtunut projektin aikana mielestäsi kehitystä tai muutoksia? (Millaista? Missä asioissa?)	1	1	1	1	1
10,1	2,4	Prosessi, Käytäntö	Onko jotain, mitä projektissa olisi pitänyt tehdä eri tavalla tähän mennessä?	1	1	1	1	1
13	2,4	Käytäntö	Mitkä projektin käytännöt olet kokenut kaikista hyödyllisimmiksi? Miksi?	1	1	1	1	1
13	2,4	Käytäntö	Mitkä projektin käytännöt ovat olleet hankalimpia, hyödyttömimpiä tai ikävimpiä? Miksi?	1	1	1		1
14,5	4	Käytäntö, Dokumentaatio	Mitä mieltä olet projektin dokumentoinnista/dokumentaatiosta? (Oletko tutustunut? Toimivuus, määrä, mikä oli turhaa, mikä hyödyllistä, olisiko jotain tarvinnut lisää?)	1	1	1		1
15	4	Käytäntö, Dokumentaatio	Oletko käyttänyt projektinhallintaan käytettävää Trac-työkalua? Miten? (Onko Trac ollut hyödyllinen?)	1	1	1	1	1
15	4	Käytäntö, Dokumentaatio	Entä Tyyliopasta, mitä mieltä olet Tyylioppaasta? (Täydentävätkö Trac ja Tyyliopas toisiaan? Pitäisikö toinen poistaa ja käyttää vain toista?)	1	1	1		
15	4	Käytäntö, Prototyypit	Mitä mieltä olet prototyypeistä? (hyödyllisyys, hyvät puolet, huonot puolet, parannusehdotukset)	1	1	1	1	1
15,1	4	Käytäntö, Työtilat	Onko sinulla jotain kommentteja projektitilasta? (Tilan koko, sijainti, työkalut, muut mahdollisuudet, parannusehdotuksia?)	1	1			

Kommunikaatio									
16	3,4	Kommunikaatio, tiedottaminen	Onko kommunikaatio toiminut projektin aikana? (Tiimin sisällä, asiakkaan ja tiimin välillä, käyttäjien ja tiimin välillä, käyttäjien ja product ownerin välillä?)	1	1	1	1	1
16,1	3,4	Kommunikaatio, tiedottaminen	Tunnetko saavasti riittävästi tietoa projektin etenemisestä ja sovelluksen tilasta? (Ymmärrätkö mitä projektissa on meneillään? Mistä haluaisit saada lisää tietoa? Mitä osaat sanoa projektin tämänhetkisestä tilanteesta? Mitä osaat sanoa tulevasta aikataulusta?)	1	1	1	1	1
16,2	3,4	Vaikuttaminen	Tuntuuko sinusta, että sinulla on mahdollisuus vaikuttaa projektiin? (Asteikolla 1-10 vaikutusmahdollisuudet. Onko valtaa päättää mitä tehdään tai mitä ei tehdä? Mihin asioihin tai miten haluaisit erityisesti vaikuttaa (toteutustavat, sovelluksen ominaisuudet, käyttöliittymä)? Onko kritiikkiä, ideoita tai vaihtoehtoja helppo esittää?)	1	1	1	1	1
16,5	3	Toimittaja vs. asiakas	Millainen on toimittajan ja asiakkaan välinen suhde mielestäsi ollut tässä projektissa? (Onko toimittajan ja asiakkaan välillä ollut erilainen suhde tässä projektissa verrattuna muihin projekteihin? Onko sillä ollut mielestäsi jotain vaikutusta projektiin tai omaan työhösi?)	1	1	1	1	1
16,6	3	Toimittaja vs. asiakas	Onko sopimusmalli tässä projektissa ollut mielestäsi toimiva? (Onko aiheutunut lisätyötä? Millaisen mallin kokisit itse paremmaksi?)	1	1	1	1	
		Kysymyksiä yhteensä	**40**	**39**	**39**	**38**	**26**	**27**