Sakari Hilama

DEVIATIONS -OHJELMISTO LAATUPOIKKEAMIEN KÄSITTELYYN

Opinnäytetyö
Kajaanin ammattikorkeakoulu
Luonnontieteiden ala
Tietojenkäsittelyn koulutusohjelma
Kevät 2009

Kajaanin ammattikorkeakoulu

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Luonnontieteiden ala	Koulutusohjelma Tietojenkäsittelyn koulutusohjelma
Tekijä(t) Hilama Sakari	
Työn nimi Deviations -ohjelmisto laatupoikkeamien käsittelyyn	
Vaihtoehtoiset ammattiopinnot Ohjelmistosuunnittelu	Ohjaaja(t) Piirainen Veli-Pekka
	Toimeksiantaja Documenta Oy / Valtteri Koivunen
Aika Kevät 2009	Sivumäärä ja liitteet 42+0

Opinnäytetyö toteutettiin Documenta Oy:n toimeksiantona. Opinnäytetyön tarkoituksena oli toteuttaa Windows SharePoint Services 3.0 -alustaa hyödyntävä ohjelmistotuote nimeltä Deviations. Deviations -ohjelmiston tarkoituksena on tarjota organisaatiolle mukautuva työväline esimerkiksi palautteen tai laatupoikkeamien hallintaan ja prosessoituun käsittelyyn. Deviations -ohjelmisto toteutettiin niin, että ohjelmisto mukautuu organisaation olemassa oleviin prosesseihin. Tämä vaatimus huomioitiin ohjelmiston suunnitteluvaiheessa.

Opinnäytetyön teoriaosuudessa käydään läpi yleisesti ketteriä menetelmiä ja tarkemmin Extreme Programming ja Scrum -menetelmää. Ketterät menetelmät ovat kokoelma ohjelmistokehitysmenetelmiä ja -malleja. Näistä menetelmistä esitellään roolit, arvot ja keskeiset toiminta-ideat. Ketteriä menetelmiä hyödynnettiin opinnäytetyönä tehdyn ohjelmiston toteutuksessa. Scrum -menetelmällä hallittiin ohjelmistoprojektia ja toteutettavia ominaisuuksia. Ohjelmakoodin toteuttamisessa hyödynnettiin Extreme Programming -menetelmän käytäntöjä.

Empiriassa käsitellään ohjelmiston vaatimusmäärittelyt, ketterien menetelmien soveltaminen tässä projektissa, Windows SharePoint Services 3.0 -alusta ja toteutetun ohjelmiston keskeiset piirteet. Lyhyesti esitellään myös ohjelmistoon kohdistunut testaus sekä käyttöönotto ensimmäisen asiakkaan osalta.

Opinnäytetyö rajattiin koskemaan ensimmäistä versiota Deviations -ohjelmistosta. Ohjelmiston kehittäminen jatkuu opinnäytetyön valmistumisen jälkeen asiakkailta saadun palautteen perusteella sekä Documenta Oy:n omien näkemyksien mukaan. Deviations -ohjelmiston versio 1.0 täytti ohjelmistolle suunnitteluvaiheessa laaditut vaatimukset.

Kieli	Suomi
Asiasanat	Ketterät menetelmät, Extreme Programming, Scrum, SharePoint 2007, WSS 3.0
Säilytyspaikka	☐ Kajaanin ammattikorkeakoulun Kaktus-tietokanta ☐ Kajaanin ammattikorkeakoulun kirjasto

Kajaanin ammattikorkeakoulu
University of Applied Sciences

THESIS
ABSTRACT

School Business	Degree Programme Data Processing
Author(s) Sakari Hilama	
Title Deviations Software for Handling Quality Deviations	
Optional Professional Studies Programming	Instructor(s) Piirainen Veli-Pekka
	Commissioned by Documenta Oy / Valtteri Koivunen
Date Spring 2009	Total Number of Pages and Appendices 42+0

The thesis was commissioned by Documenta Oy. The goal of the thesis was to create a software product called Deviations. The Deviations software is a tool for organizations to handle feedback or quality deviations. The software was designed to adapt to and support the existing processes of organizations. The Deviations software uses the Windows SharePoint Services 3.0 platform.

The theoretical part of the thesis presents the Agile Software Development methods and takes a deeper look at the Extreme Programming and the Scrum methods. The roles, values and the main idea are explored from both of these methods. The Agile software development is a collection of software development models and practises. The Agile Software Development models were used in the Deviations software. The Scrum method was used to control the software project and the practises from the Extreme Programming method were used in the production of the source code.

The empirical part of the thesis addresses the requirements analysis, use of the Agile methods, Windows SharePoint Services 3.0 platform and the main features of the Deviations software. The testing and deployment of the Deviations software are briefly handled in the thesis as well.

The thesis was outlined to focus only on the Deviations software version 1.0. The future development of the software continues in Documenta Oy based on customer feedback and the opinions of employees. The version 1.0 of the Deviations software fulfilled the requirements set at the beginning of the project.

Language of Thesis	Finnish
Keywords	Agile Software Development, Extreme Programming, Scrum, SharePoint 2007
Deposited at	☐ Kaktus Database at Kajaani University of Applied Sciences ☐ Library of Kajaani University of Applied Sciences

ALKUSANAT

Haluan kiittää Documenta Oy:tä tämän opinnäytetyön aiheen tarjoamisesta ja luottamuksesta opinnäytetyön toteuttamiseen.

SYMBOLILUETTELO

ASP.NET	Active Server Pages, Microsoftin kehittämä tapa luoda dynaamisia Internet sivuja
C#	Microsoftin kehittämä ohjelmointikieli
Deviations for Domino	Documentan ohjelmistotuote IBM Domino -alustalle
Exchange	Microsoftin palvelintuote sähköpostiliikenteen hallintaan
Laatukäsikirja	Documentan ohjelmistotuote laatuasiakirjojen sähköiseen hallintaan
IBM Domino	Sovellutusalusta ja palvelinohjelmisto
Inkrementti	Lisäys kehitteillä olevan ohjelmistoon. Lisäys voi olla joko uusi ominaisuus tai ohjelmistossa olevan ominaisuuden parantaminen
Iteraatio	Ajallisesti määritelty jakso, jonka aikana kehitettävään ohjelmistoon toteutetaan ominaisuuksia
Metatieto	Tietoa tiedosta
MOSS	Microsoft Office SharePoint Server

Ohjelmiston arvo	Asiakkaan näkemä hyöty ohjelmistosta
Workflow Foundation	Microsoftin kehittämä työnkulkukirjasto
WSS 3.0	Windows SharePoint Services 3.0.

SISÄLLYS

SYMBOLILUETTELO

1 JOHDANTO 1

2 KETTERÄ OHJELMISTOKEHITYS 3
2.1 Ketterä julistus 4
2.2 Extreme Programming (XP) 7
2.2.1 Roolit 10
2.2.2 Arvot 11
2.2.3 Käytännöt 12
2.3 Scrum -menetelmä 14
2.3.1 Roolit 16
2.3.2 Tuotteen ominaisuusluettelo (Product backlog) 18
2.3.3 Sprintti (Sprint) 20
2.3.4 Päivittäinen seurantapalaveri (Daily Scrum) 22
2.3.5 Ohjelmistokehityksen riskien hallinta Scrum -menetelmässä 23
2.4 Extreme Programming ja Scrum 24

3 DEVIATIONS OHJELMISTO 26
3.1 Määrittelyt 26
3.2 Ketterien menetelmien soveltaminen 28
3.3 Tekninen toteutus 30
3.3.1 Toteutusalusta 30
3.3.2 Metatietolomake 32
3.3.3 Työnkulut 34
3.3.4 Työnkulun ja metatietolomakkeen integraatio 35
3.3.5 Sähköpostitoiminnallisuus 36
3.4 Testaus 37
3.5 Käyttöönotto 37

4 POHDINTA 39

LÄHTEET 40

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena oli toteuttaa Documenta Oy:lle kaupallinen ohjelmistotuote nimeltä Deviations. Deviations -ohjelmiston tarkoituksena on tarjota organisaatiolle mukautuva ohjelmisto palautteen tai laatupoikkeamien hallintaan. Deviations ohjelmisto tukee myös organisaatiossa mahdollisesti olevia laatuprosesseja. Projektin aikana havaittiin, että ohjelmisto pystyy tukemaan muitakin prosesseja kuten mikrotuen helpdesk -toimintaa.

Opinnäytetyön teoriataustaksi valittiin ketterät menetelmät (Agile Software Development). Ketterät menetelmät ovat kokoelma erilaisia menetelmiä, joiden tarkoituksena on edesauttaa ohjelmiston toteuttamista suunnitellun mukaan. Ketterät menetelmät pyrkivät keskittymään olennaiseen eli ohjelmiston ja ohjelmakoodin toteuttamiseen. Ketterät menetelmät ovat hyvin käytännön läheisiä ja kokemusperäisesti muodostuneita.

Opinnäytetyössä hyödynnettiin ketteristä menetelmistä kahta yleisintä menetelmää, jotka ovat Extreme Programming- ja Scrum -menetelmä. Menetelmien avulla pyrittiin ohjaamaan Deviations -ohjelmiston kehitysprosessia ja toisaalta tutustumaan syvällisemmin Documenta Oy:n puolelta ketteriin menetelmiin. Molempia menetelmiä on tarkoitus hyödyntää myöhemmin Documentan muissa ohjelmistoprojekteissa.

Ohjelmiston määrittelyssä ja suunnittelussa hyödynnettiin Documentan aiemmin toteuttamaa ohjelmistotuotetta Deviations for Domino, joka on toteutettu IBM Domino -alustalla. On kuitenkin tärkeää huomioida, että IBM Domino -alusta ja Microsoft Windows SharePoint Services 3.0 -alusta ovat hyvin erilaisia. Tämän johdosta aiemmasta ohjelmistosta pystyttiin hyödyntämään vain korkean tason ratkaisuja, joten ohjelmakoodin osalta kaikki täytyi toteuttaa kokonaan uudelleen.

Käytännön osuudessa käydään läpi opinnäytetyönä toteutetun ohjelmiston piirteet ja toteutustekniikka. Opinnäytetyön ohjelmisto toteutettiin Microsoft Windows SharePoint Services 3.0 -alustalle C# -ohjelmointikielellä. Toteutusalusta ja -ratkaisut olivat tekijälle entuudestaan tuttuja, sillä aiemmin kesällä ja syksyllä 2007 toteutettiin työharjoittelussa Laatukäsikirjaohjelmisto. Deviations- ja Laatukäsikirja -ohjelmistot perustuvat samoihin teknisiin ratkaisuihin.

Deviations -ohjelmisto saatiin ensimmäisen version tasoon joulukuussa 2008. Tällöin ohjelmisto asennettiin asiakkaalle pilottikäyttöön ja ohjelmistokehitys siirtyi tuotteen ylläpito- ja jatkokehitysvaiheeseen ennen seuraavaa merkittävän version toteutusta.

2 KETTERÄ OHJELMISTOKEHITYS

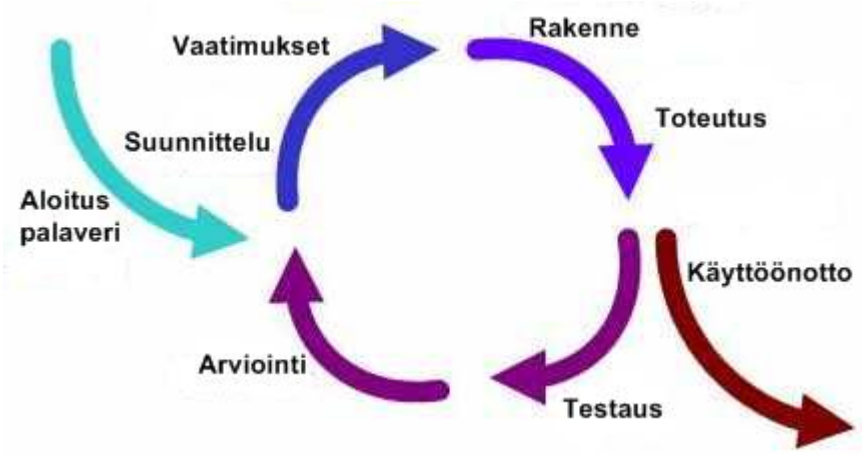
Ketterä ohjelmistokehitys (Agile Software Development) sai virallisesti alkunsa helmikuussa 2001. Ketterien menetelmien edustajat kokoontuivat Utahissa Yhdysvalloissa keskustelemaan edustamiensa menetelmien yhteisestä perustasta. Tuolloin ketteriä menetelmiä kutsuttiin kevyiksi menetelmiksi. Nimitys johtui siitä, että aiemmin hallitsevana ohjelmistokehitysmallina oli vesiputousmalli, jota pidetään raskaana ja määrämuotoisena mallina. Tapahtuman yhteydessä laadittiin ketterä julistus, jota pidetään ketterien menetelmien yhteisenä perustana. Samalla perustettiin Agile Alliance -niminen, voittoa tavoittelematon järjestö, edistämään ketteriä menetelmiä ja tarjoamaan tietoa näistä menetelmistä. (Abrahamsson, P. Salo, O. Ronkainen, J. Warsta J. 2002, 11; Cohn, M. 2005, 21.)

Ketterillä menetelmillä on muutamia piirteitä, jotka yhdistävät menetelmät toisiinsa. Selkein yhdistävä piirre liittyy ohjelmistokehitykseen, joka ketterissä menetelmissä tapahtuu inkrementaalisesti ja iteratiivisesti. Ketterät menetelmät pyrkivät paremmin huomioimaan ihmiset, jotka ovat projektissa mukana. Menetelmät ovat suoraviivaisia ja helppoja omaksua sekä soveltaa. Menetelmissä arvostetaan erityisesti toimivaa ohjelmakoodia ja osallistujien välistä yhteistyötä. (Abrahamsson ym. 2002, 15 - 17.)

Ketterien menetelmien iteratiivisuus ja inkrementaalisuus ovat erityisesti painotettuna ketterissä menetelmissä. Kehitettävänä olevaa ohjelmistoa pyritään kehittämään ennalta määrätyissä, ajallisissa jaksoissa eli iteraatioissa. Jokaisen iteraation tarkoituksena on tuottaa ohjelmistosta versio, joka parhaimmassa tapauksessa on toimiva ja myyntikelpoinen. Inkrementaalisuus liittyy läheisesti iteraatioihin. Inkrementaalisuudella tässä yhteydessä tarkoitetaan ohjelmiston täydentämistä jokaisen iteraation aikana. Ketterissä menetelmissä ei pyritä kerralla saattamaan tiettyä ominaisuutta täysin valmiiksi, vaan toteutettavan ominaisuuden tietyt piirteet toteutetaan ensin ja vasta useiden iteraatioiden jälkeen kyseinen ominaisuus on täysin valmis. (Abrahamsson ym. 2002, 14 - 16.)

Ketterien menetelmien iteratiivisuus ja inkrementaalisuus tuodaan hyvin esille Kuvio 1. Kuviossa esitetään nuolien avulla ketterien menetelmien yleinen kehityssykli siten, että ohjelmiston toteuttaminen aloitetaan aloituspalaverilla, jossa pyritään suunnittelemaan ohjelmiston toteuttamista. Tämän jälkeen siirrytään ”iteraatiosilmukkaan”, jossa edetään

suunnitellen, toteuttaen, testaten ja arvioiden. Tätä silmukkaa toteutetaan niin pitkään, kun ohjelmistossa on toteutettavia piirteitä tai projekti on tarpeellinen. Mikäli tuote saadaan syklin aikana ennalta sovittuun valmistumistasoon, voidaan silmukasta poistua. (SliQTools 2008.)



Kuvio 1. Ketterien menetelmien toimintakaavio (mukailtuna SliQTools, 2008)

Keskeisiä ketteriä menetelmiä on kahdeksan. Nämä menetelmät ovat Adaptive Software Development, Crystal Methodologies, Dynamic Systems Development, Extreme Programming, Feature Driven Development, Open Source Software Development, Rational Unified Process ja Scrum. Näistä laajinten käytössä ovat Extreme Programming ja Scrum sekä niistä muodostuvat hybridit. (Abrahamsson ym. 2002, 18; Koskela, Lasse 2008)

2.1 Ketterä julistus

Ketterä julistus laadittiin ja julkistettiin virallisesti vuonna 2001 Yhdysvalloissa. Julistuksen laativat ja allekirjoittivat 17 ketterien menetelmien edustajaa. Ketterän julistuksen neljä pääkohtaa ovat:

- Yksilöt ja yhteistyö ylitse prosessien ja työkalujen
- Toimiva ohjelmisto ylitse kattavan dokumentaation
- Yhteistyö asiakkaan kanssa neuvottelemisen sijaan

- Muutokseen vastaaminen suunnitellun seuraamisen sijaan

Seuraavissa kappaleissa on tarkoitus hieman avata näiden esiteltyjen pääkohtien sanomaa ja pyrkiä selventämään niitä. (Abrahamsson ym. 2002, 11 - 12; Cohn, M. 2005, 21; Lindberg, H. 2003, 23 - 24; The Agile Manifesto, 2001.)

Ennen ketterää julistusta ja ketteriä menetelmiä, ohjelmistokehityksestä pyrittiin tekemään mahdollisimman kaavamaista ja prosessista viimeisteltyä. Kaavojen ja prosessien tavoitteena oli sivuuttaa yksilöiden vaikutus ohjelmistokehitysprojektiin. Ohjelmistokehitysprojektin näkökulmasta tämä on merkittävä ongelma, johon ketterät menetelmät tarjoavat ratkaisua. Ketterät menetelmät muuttavat tämän asetelman siten, että keskeisessä roolissa ovat yksilöt ja yksilöiden välinen yhteistyö. Ketterät menetelmät tunnustavat yksilöiden ainutlaatuiset vahvuudet ja heikkoudet pyrkien vahvistamaan näitä ominaisuuksia ohjelmistokehitysprojektin aikana. (Abrahamsson ym. 2002, 11 - 12; Cohn, M. 2005, 21.)

Ketterät menetelmät edustavat nopeutta ja vuorovaikutusta asiakkaan kanssa. Tämän saavuttamiseen tarvitaan nopeaa ohjelmistokehitystä, joka painottaa yksinkertaista ja toimivaa ohjelmakoodia. Nopeuden saavuttamiseksi dokumentaatio on toisarvoista ketterissä menetelmissä. Tätä ajatusta noudatetaan varsinkin myöhemmin esiteltävässä Extreme Programming -ohjelmistokehitysmallissa. Jokaisella iteraatiokierroksella pyritään toimittamaan asiakkaalle tai käyttäjille testattava versio ohjelmasta. Periaatteena on, että jokaisen iteraatio kierroksen jälkeen ohjelmisto olisi myytävässä kunnossa. Käyttäjiltä saatavan palautteen perusteella tehdään muutoksia ohjelmistoon ja ohjataan tiimiä tekemään niitä asioita, jotka asiakas näkee tarpeelliseksi. (Abrahamsson ym. 2002, 11 - 12; Cohn, M. 2005, 22.)

Yhteistyötä asiakkaan ja projektiin osallistujien kanssa pidetään tärkeänä ketterissä menetelmissä. Yhteistyön avulla pyritään varmistamaan, että jokaisella osapuolella on samanlainen näkemys tavoitteista ja tarve vedota sopimuksissa laadittuihin kohtiin siten vähenee. Neuvottelun tarkoituksena ketterissä menetelmissä on saavuttaa ja ylläpitää hyvät suhteet asiakkaan ja kehittäjien välillä. Ohjelmistoliiketoiminnallisesta näkökulmasta ketterien menetelmien pääajatuksena on alusta alkaen toimittaa asiakkaalle arvoa, joka hyödyttää asiakasta ja siten vähentää riskiä, joka kohdistuu sopimuksien toimittamatta jättämispykäliin. (Abrahamsson ym. 2002, 11 - 12; Cohn, M. 2005, 22.)

Muutokseen myönteisesti suhtautumista painotetaan ketterissä menetelmissä. Muutoksen tarkoituksena on tuottaa asiakkaalle arvoa ja hyötyä, toteuttamalla niitä ominaisuuksia, jotka asiakas näkee tarpeelliseksi. Toisaalta kehittäjien näkökulmasta on mahdotonta ennalta määritellä kaikki tarvittavat ominaisuudet tai toiminnot ennen toteutusta, mikäli kyseessä on laajempi ohjelmistoprojekti. Erityisesti myöhemmin käsiteltävässä Scrum ohjelmistokehitysmallissa tämä muutokseen suhtautuminen näkyy tuotteen ominaisuusluettelon kautta. Ominaisuusluettelon avulla asiakas pystyy vaikuttamaan kehitettävänä olevan ohjelmiston ominaisuuksiin joustavasti missä tahansa kehityksen vaiheessa tietyin rajoituksin.(Abrahamsson ym. 2002, 11 - 12; Cohn, M. 2005, 22.)

Ketterän julistuksen lisäksi laadittiin myös 12 kohtaa sisältävä periaatteiden listaus, joita ketterät menetelmät noudattavat. Periaatteet ovat:

1. Tärkeimpänä tavoitteena on täyttää asiakkaan toiveet jatkuvalla ohjelmistoversioiden kehittämisellä ja julkaisemisella.

2. Toivotetaan muutokset tervetulleeksi myöhäisessäkin kehitysvaiheessa ja pyritään tuottamaan asiakkaalle kilpailuetu ohjelmiston kautta.

3. Asiakkaalle toimitetaan ohjelmistosta toimivia versioita säännöllisesti muutaman viikon tai muutamien kuukausien sykleissä.

4. Liiketoiminnan henkilöiden ja tuotekehityksen henkilöiden tulee työskennellä yhdessä päivittäin projektin ajan.

5. Projektin ympärille kerätään motivoituneet yksilöt toteuttamaan sitä. Projektin henkilöille annetaan tarvittava tuki ja luotetaan siihen, että he saavat tehtävän suoritettua.

6. Tiedon ja tietämyksen välittäminen tapahtuu tehokkaimmin kehitystiimissä kasvokkain keskustelemalla.

7. Toimiva ohjelmisto on paras onnistumisen arviointimenetelmä.

8. Ketterissä menetelmissä suositaan jatkuvaa ja vakaata kehitystä. Kaikkien projektiin osallistuvien tahojen tulisi pystyä työskentelemään tasaisessa ja säännöllisessä työtahdissa.

9. Jatkuva huomio tekniseen ylivertaisuuteen ja hyvään laatuun parantavat ketteryyttä.
10. On tärkeää korostaa yksinkertaisuutta ja tekemättömän työn määrää.
11. Paras arkkitehtuuri, vaatimukset ja suunnittelu kehittyvät itseohjautuvista tiimeistä.
12. Tasaisin väliajoin tiimi pysähtyy miettimään sitä, kuinka voisi tulla tehokkaammaksi ja muokkaa toimintaansa sen mukaisesti.

Tämän julistuksen tarkoituksena on tuoda esille muutamia perusperiaatteita, jotka ovat yhteisiä kaikille ketteriin menetelmiin lukeutuville ohjelmistokehitysmalleille. (Abrahamsson ym. 2002, 11 - 12; Huttunen, J. 2006, 15 - 16; Lindberg, H. 2003, 23 - 24; The Agile Manifesto, 2001)

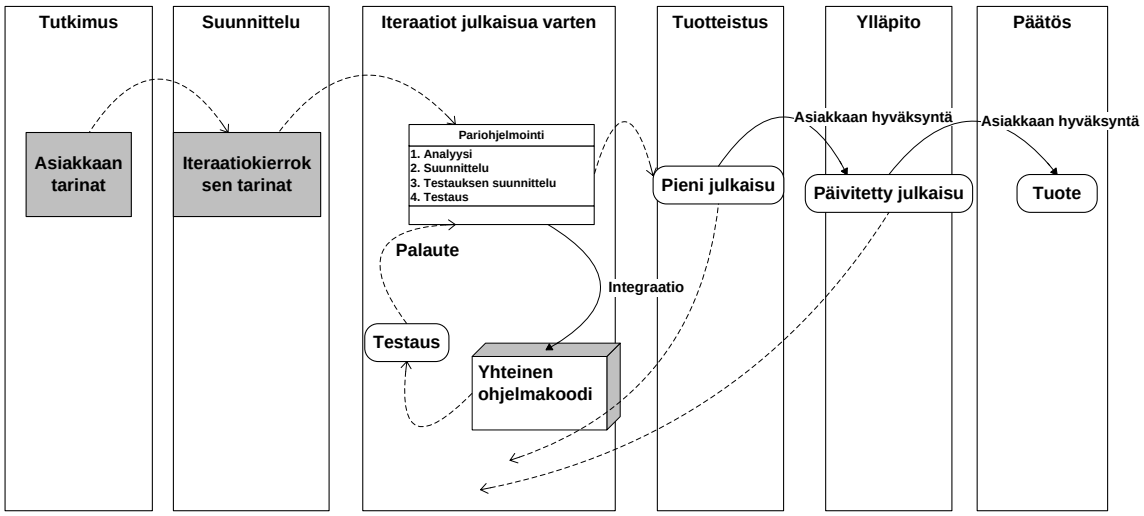
2.2 Extreme Programming (XP)

Extreme Programming (XP) on eräs laajimmin käytössä olevista ketteristä menetelmistä Scrum -menetelmän ohella. XP:n keskeisenä ajatuksena on keskittyä olennaiseen eli ohjelmakoodin tuottamiseen ja ohjelmiston toteuttamiseen. XP:tä pidetään avoimuuden ja seurattavuuden johdosta hyvin soveltuvana riskialttiisiin ohjelmistoprojekteihin. Ohjelmiston liittyvää dokumentaatiota XP:ssä pidetään vähemmän tärkeänä ja välivaiheissa sitä ei tuoteta. XP suosittelee laatimaan dokumentaatiot vasta siinä vaiheessa, kun ohjelmisto on täysin valmis ja muutoksia ei enää tule kyseiseen pääversioon. Ketterien menetelmien julistuksen yksinkertaisuuteen vedoten, XP pyrkii ohjaamaan ohjelmakoodin tuottamista siten, että ohjelmakoodi itsessään olisi riittävän yksinkertaista ymmärrettäväksi ilman kommentointia. (Abrahamsson ym. 2002, 18 - 19.)

XP:n kehittäjä, Kent Beck, luonnehtii XP:tä sosiaalisena muutoksena. Beckin mukaan XP on vanhojen tapojen unohtamista, jotka aiemmin olivat toimivia, mutta nykyään estävät tekemästä parasta mahdollista työtä. Beckin mukaan XP:n omaksuminen tarkoittaa ohjelmakoodin tuottajan osalta suojista luopumista ja asettumista suoraan kontaktiin asiakkaan kanssa. (Abrahamsson ym. 2002, 26 - 27; Beck & Andres, 2006.)

XP:n perusprosessi muodostuu kuudesta eri vaiheesta, joiden välillä liikutaan, kun tavoitteet on saavutettu tai saatu palaute vaatii palaamaan edelliseen vaiheeseen (kuvio 2.). ”Iteraatiot

julkaisua varten" -vaiheeseen voidaan palata ja tätä vaihetta toistetaan eniten XP:n perusprosessin aikana.



Kuvio 2. XP:n prosessi (mukailtuna Abrahamsson ym. 2002, 19)

Tutkimusvaiheessa asiakas laatii niistä ominaisuuksista yksinkertaiset tarinakortit, jotka halutaan kehitteillä olevassa versiossa toteuttaa. Tarinakorttien avulla asiakas kertoo yksinkertaistettuna haluamansa ominaisuudet kehittäjäryhmälle ja niiden laadinnan aikana kehittäjätiimi voi ennalta perehtyä toteutuksessa käytettäviin käytäntöihin, teknologioihin ja työvälineisiin. Tutkimusvaiheen kesto vaihtelee kahden viikon ja kahden kuukauden välillä riippuen teknologian haastavuudesta ja kehitettävänä olevan ohjelmiston ominaisuuksista. Tutkimusvaiheesta siirrytään suunnitteluvaiheeseen, kun arvioidaan, että kaikki tarvittavat pääpiirteet on laadittu tarinakorteiksi. Tämän vaiheen jälkeen asiakas voi edelleen laatia tarinakortteja, jotka otetaan mukaan kehitykseen sitä mukaa kun aiemmin valitut tarinakortit saadaan toteutetuiksi. (Abrahamsson ym. 2002, 20; Jeffries, R. Anderson, A. Hendrickson, C. 2006, 23 - 25.)

Suunnitteluvaiheessa asiakas ja kehittäjätiimi asettavat tarinakortit tärkeysjärjestykseen. Kehittäjätiimi arvioi tarinakorttien toteutuksen vaatiman ajan ja kirjaa sen ylös tarinakortille. Arvioinnin jälkeen pyritään laatimaan aikataulu ensimmäiselle pienelle julkaisulle. Yleensä suunnitteluvaiheen ja ensimmäisen pienen julkaisun välinen ajanjakso ei ylitä kahta kuukautta. Suunnitteluvaihe itsessään kestää päivästä kahteen päivään. (Abrahamsson ym. 2002, 20.)

Julkaisun iteraatiot -vaihe sisältää useita pieniä iteraatiokierroksia. Suunnitteluvaiheessa laadittu aikataulu jaetaan pienempiin osiin, joista muodostuu jokaiselle iteraatiokierrokselle aikataulu. Tavallisimmin yksittäinen iteraatiokierros kestää viikosta neljään viikkoon, jonka aikana toteutetaan asiakkaan osoittamien tarinakorttien kuvailemat ominaisuudet. Ensimmäisen iteraatiokierroksen aikana pyritään toteuttamaan ohjelmiston runko, johon seuraavissa iteraatiokierroksissa lisätään ominaisuuksia. Jokainen iteraatiokierros aloitetaan tarinakortin tarkemmalla analysoinnilla. Tämän jälkeen toteutettava ohjelmakoodi suunnitellaan ja laaditaan ohjelmakoodille testausmallit. Ohjelmakoodi toteutetaan testien laadinnan jälkeen ja ohjelmakoodin toteutusta seuraa testaus. Viimeisen iteraatiokierroksen jälkeen ohjelmisto on valmis siirtymään tuotteistusvaiheeseen. (Abrahamsson ym. 2002, 20.)

Tuotteistusvaiheessa suoritetaan erilaisia testejä koko ohjelmistolle, jota toteutetaan XP:n avulla. Tuotteistusvaiheessa ilmenevät virheet korjataan ja tarvittaessa voidaan toteuttaa vielä uusia ominaisuuksia, mikäli ne ovat ohjelmiston kannalta tarpeellisia. Tuotteistusvaiheessa ilmenevät uudet ideat suositellaan dokumentoitaviksi ja niiden toteutusta pyritään siirtämään ylläpitovaiheeseen. Tuotteistusvaiheen kesto on yhdestä kolmeen viikkoa. Tuotteistusvaiheen jälkeen asiakkaalle toimitetaan ensimmäinen julkaisu ohjelmistosta. Tuotteistusvaiheesta seuraavaan vaiheeseen siirrytään asiakkaan hyväksynnän jälkeen. (Abrahamsson ym. 2002, 20.)

Ylläpitovaiheen tärkeimpänä tavoitteena on pitää asiakkaalle asennettu ohjelmisto toimivana ja samalla ohjata uusia iteraatiokierroksia. Ylläpitovaiheessa voidaan edelleen toteuttaa ohjelmiston kehittämistä, mutta yleensä kehitysvauhti on hitaampi kuin aiemmissa vaiheissa. Toteutettavien parannusten johdosta asiakkaan osallistumista tarvitaan edelleen ylläpitovaiheessa. Myös kehitystiimin muutokset ovat mahdollisia tässä vaiheessa. Ylläpitovaiheen kestosta XP ei anna suosituksia. Kuten tuotteistusvaiheessa, ylläpitovaiheesta siirrytään seuraavaan vaiheeseen asiakkaan hyväksynnän kautta. (Abrahamsson ym. 2002, 20 - 21.)

Päätösvaihe koittaa kun asiakkaalla ei ole uusia tarinakortteja ohjelmistoa varten ja ohjelmisto siten täyttää asiakkaan odotukset. Päätösvaiheessa ohjelmistoon ei enää kohdisteta muutoksia ja ohjelmakoodi ei enää muutu. Tässä vaiheessa suoritetaan ohjelmiston dokumentointi ja ohjelmakoodin tarvittava kommentointi. Päätösvaihe toteutuu myös silloin, kun ohjelmisto ei täytä asiakkaan vaatimuksia tai projektin kustannukset nousevat liian korkeiksi. (Abrahamsson ym. 2002, 20 - 21.)

2.2.1 Roolit

XP pyrkii ohjaamaan rooli ajattelua siten, että jokainen tiimin jäsen pyrkii tuomaan oman panoksensa tiimin työskentelyyn ja tekemään parhaansa. Tässä kappaleessa esitetyt roolit avustavat erityisesti XP:n käytäntöön ottamisessa, sekä edustavat selkeimmin erottuvia rooleja. Nämä roolit ovat: (Beck & Andres 2006, 17 - 18.)

- Arkkitehti (Architect)
- Asiakas (Customer)
- Konsultti (Consultant)
- Ohjelmoija (Programmer)
- Projektipäällikkö (Project Manager)
- Seuraaja (Tracker)
- Tekninen kirjoittaja (Technical Writer)
- Testaaja (Tester)
- Tuotepäällikkö (Product Manager)
- Valmentaja (Coach)
- Vuorovaikutuksen suunnittelija (Interaction Designer)

Myöhemmin käsiteltävään Scrum -menetelmään nähden XP:ssä roolijako on huomattavasti laajempi ja väljempi. Toisaalta tässä luetellut roolit pyrkivät vastaamaan jokaisesta organisaatiosta ennestään löytyviä rooleja. (Abrahamsson ym. 2002, 21 - 22; Beck & Andres 2006, 74 - 81.)

2.2.2 Arvot

XP tarjoaa viisi perusarvoa, joita XP:n ohjelmistokehitysprosessissa arvostetaan ylitse muiden. Arvot noudattavat pitkälti ketterien menetelmien yleisiä periaatteita. Nämä viisi perusarvoa ovat kommunikointi, yksinkertaisuus, palaute, rohkeus ja arvostus.(Beck & Andres 2006, 17 - 18.)

Kommunikointi on keskeisellä sijalla kaikissa ketterissä menetelmissä ja erityisesti XP:ssä. Ohjelmistokehityksessä tulee jatkuvasti vastaan ongelmia ja lähes aina joku on törmännyt vastaavaan ongelmaan ja ratkaissut sen. Kommunikaation avulla voidaan joko suoraan välttää ongelmien muodostumista tai ratkaista niitä nopeammin. Kommunikointi myös edesauttaa tiimihengen muodostumista ja tehokasta yhteistyötä. (Beck & Andres 2006, 18.)

Yksinkertaisuus on XP:n haastavin perusarvo. Yksinkertaisuuden tavoittelussa tulee osata nähdä se piste, johon asti yksinkertaisuutta kannattaa tavoitella. Liian yksinkertaistettu malli ei välttämättä tavoita vaadittuja ominaisuuksia tai rajoittaa tulevia ominaisuuksia. Tämän johdosta yksinkertaisuuden omaksuminen voi olla hankalaa. Toisaalta myös näkökulma, josta yksinkertaisuutta ja yksinkertaisia ratkaisuja tarkastellaan, muuttuu kokoajan. Kommunikointi tukee yksinkertaisuutta ja sen tavoittamista poistamalla tarpeettomia vaatimuksia, joita voi muodostua, kun toteutetaan suunnittelua yhden henkilön toimesta. (Beck & Andres 2006, 18.)

Palautteen merkitys XP -menetelmässä on suuri. Ohjelmistolle asetetut vaatimukset muuttuvat ja ketterien menetelmien yleisien periaatteiden mukaan muutokseen tulee suhtautua vastaanottavasti ja mukautuvasti. Palaute ohjaa tätä mukautumista ja kertoo tekijöille, mihin suuntaan ohjelmistoa tulisi kehittää. Palautetta pyritään XP -menetelmässä keräämään kaikkialta: asiakkaalta, kehittäjiltä ja kehitettävältä ohjelmistolta itseltään testauksien muodossa. XP pyrkii saamaan palautteen mahdollisimman nopeasti käytettäväksi ohjelmiston kehitysprosessissa. (Beck & Andres 2006, 19 - 20.)

Ohjelmistokehittäjiltä tarvitaan toisinaan *rohkeutta* toteuttaa tarvittavat asiat, ilman että ne ovat etukäteen tuttuja tai on varmuutta onnistumisesta. Toisaalta rohkeutta tarvitaan kehittäjien puolelta myös siihen, että uskalletaan myöntää virheet ja korjata ne ajoissa. XP vaatii rohkeutta myös asiakkaalta luottaa ohjelmistokehittäjiin ja heidän kykyynsä toimittaa ohjelmisto. (Beck & Andres 2006, 20.)

Viimeisenä pääperiaatteena XP kannustaa tiimin sisäiseen *kunnioitukseen*. Tiimin jäsenten tulisi kunnioittaa toisiaan ja kantaa vastuuta toisistaan ohjelmistoprojektin lisäksi. XP kohtelee jokaista prosessiin osallistuvaa henkilöä yksilönä ja jokaisen yksilön arvo ohjelmistoprojektille on yhtä suuri. (Beck & Andres 2006, 20.)

2.2.3 Käytännöt

Tämän kappaleen tarkoituksena on käydä läpi muutamia XP:n käytäntöjä. XP suosittelee hyödyntämään niitä, muttei ehdottomasti vaadi noudattamaan jokaista käytäntöä. XP tarjoaa tässä esiteltäviä käytäntöjä turvallisina välineinä XP:n käyttöönottoa varten. (Beck & Andres 2006, 37.)

XP suosittelee toteuttamaan ohjelmoinnin lähes poikkeuksetta *pariohjelmointina*. Pariohjelmointina tapahtuvassa ohjelmakoodin tekemisessä ohjelmointiparilla on käytössään yksi työasema, jolla tehdään ohjelmakoodia. Tämän käytännön merkittävimpiin etuihin listataan ohjelmakoodin selkeys, tietyissä tilanteissa nopeampi virheiden ratkaisu ja selvemmät ideat ohjelmakoodin toteutuksessa. Eduiksi luetaan myös se, että parit auttavat toisiaan paremmin keskittymään tehtävään. XP suosittelee vaihtamaan ohjelmointipareja muutamien tuntien välein riippuen toteutettavasta ohjelmistosta.(Beck & Andres 2006, 42 - 43; Jeffries ym. 2006, 88 - 91.)

Pariohjelmointia täydentää *yhteisen ohjelmakoodin omistajuuden* käytäntö. Yhteisellä ohjelmakoodin omistajuudella tarkoitetaan sitä, että koko tekijätiimi on yhteisesti vastuussa tuotetusta ohjelmakoodista. Periaatteessa kuka tahansa tiimin jäsenistä voi tehdä muutoksia mihin tahansa ohjelmakoodin osaan. Tämä kuitenkin edellyttää vahvaa yhteisvastuun tunnetta tekeillä olevasta ohjelmistosta toimiakseen hyvin. (Beck & Andres 2006, 66; Jeffries ym. 2006, 72 - 73.)

XP:n käytännöissä tarjotaan kehityksen jakamista kierroksiin (ts. iteraatioihin tai sykleihin). Pienin kierros XP:ssä on *viikkokierros*. Viikkokierros kestää 7 päivää ja yleensä alkaa kehitystiimin yhteisellä palaverilla. Palaverissa käydään läpi siihen mennessä toteutetut piirteet ja arvioidaan julkaisuaikataulua (joko pienen julkaisun tai pääversion julkaisuaikataulua). Tämän jälkeen asiakas valitsee seuraavalle viikolle ”tarinat”, jotka asiakas näkee olevan toteutettavissa kyseisen viikon aikana. Kehitystiimi muokkaa annetut tarinat

yksittäisiksi tehtäviksi. Kehitystiimin parit valitsevat tehtävät ja arvioivat näiden toteutusajan. Tehtävien valinnan ja arvioinnin jälkeen viikkokierros aloitetaan. (Beck & Andres 2006, 46 - 47.)

Neljänneskierros on XP:n käytännöissä yleensä rinnastettu yrityksien kvartaali- eli vuosineljännes ajatteluun. Neljänneskierroksen tarkoituksena on heijastaa työtilannetta pidemmällä aikavälillä. Neljänneskierroksen aloituspalaverissa keskitytään ratkaisemaan projektien aikana ilmenneitä ongelmia ja erityisesti niitä ongelmia, jotka ovat kehitystiimin ulkopuolelta johtuvia. Neljänneskierroksen palaverissa käydään läpi myös organisaation tilanne ja se, miten toteutettavat projektit soveltuvat yrityksen kokonaiskuvaan. (Beck & Andres 2006, 47 - 48.)

Yleensä toteutettava ohjelmisto jaetaan pienempiin kokonaisuuksiin. Tällöin XP suosittelee ”*jatkuvan integraation*” -käytäntöä. Jatkuvalla integraatiolla tarkoitetaan XP:ssä tuotetun ohjelmakoodi osan yhdistämistä tekeillä olevaan ohjelmistoon mahdollisimman aikaisessa vaiheessa. Ohjelmakoodin yhdistäminen tuo esille piilevät virheet, jotka muuten eivät olisi havaittavissa tai jotka eivät ilmene kuin vasta yhdistämisen jälkeen. Jatkuva integraatio kuluttaa aikaa paljon, eräissä tapauksissa jopa enemmän kuin itse ohjelmakoodin tekeminen, mutta vastineena saavutetaan paremmin toimiva ohjelmisto ja laadukkaampi lopputulos. (Beck & Andres 2006, 49 - 50; Jeffries ym. 2006, 78 - 79.)

Ketterien menetelmien mukaan myös XP suosittelee noudattamaan *inkrementaalista suunnittelua*. Toteutettavana olevan ohjelmiston suunnittelua ja rakennetta tulisi muokata jokaisessa iteraatiossa vastaamaan iteraation aikana ilmeneviä tarpeita. Tämä käytäntö ohjaa ohjelmiston suunnittelua ja rakennetta vastaamaan ohjelmiston vaatimuksia ja minimoi huonosta suunnittelusta johtuvia riskejä. (Beck & Andres. 2006, 51 - 53.)

Refaktorointi (Refactoring) eli uudelleen tekeminen on pariohjelmoinnin ohella toinen suoraan ohjelmointiin vaikuttava käytäntö XP:ssä. Refaktoroinnilla tarkoitetaan XP:ssä ohjelmakoodin uudelleenkirjoittamista siten, että arkkitehtuurillista toteutusta muutetaan, mutta toiminnallinen ajatus pysyy ennallaan. Refaktoroinnin tarkoituksena on tukea inkrementaalista suunnittelua ja tukea muutosten tekemistä ohjelmistoon. Refaktorointi vaatii onnistuakseen luotettavat testit ja kunnollisen testauksen. Mikäli refaktoroinnilla tehdään sellainen muutos, joka vaikuttaa ohjelmiston toimintaan, muutos pitää pystyä

jäljittämään ja peruuttamaan. Näiden vaatimusten johdosta refaktorointi edellyttää muiden käytäntöjen ajan tasalla olemisen. (Jeffries ym. 2006, 76 - 77.)

XP ohjaa laatimaan testit ennen ohjelmakoodin toteutusta. Testien tulee alussa päätyä epäonnistumiseen, sillä ohjelmakoodia ei ole toteutettu. *Testaus ennen ohjelmointia* -käytännön tarkoituksena on vähentää epäolennaisen ohjelmakoodin laatimista ja näyttää ohjelmoijalle selvä päämäärä: testi toimivaksi tai uuden testin laadinta. Käytäntö nopeuttaa merkittävästi ohjelmavirheiden löytämistä ja siten edesauttaa toimivan ohjelmiston toteuttamista. (Beck & Andres 2006, 50 - 51.)

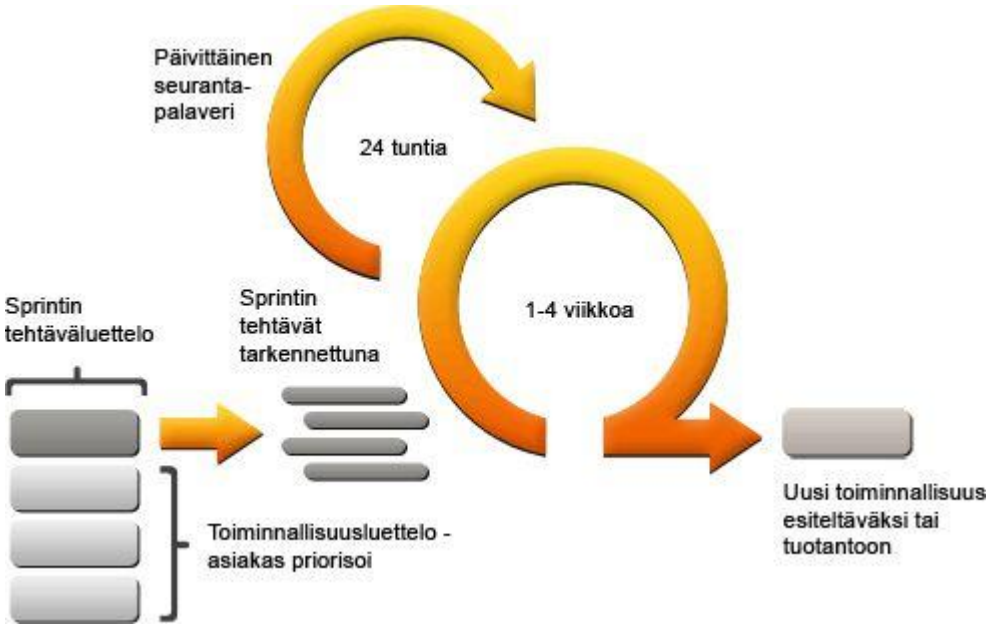
Työajan pituudeksi XP suosittelee 40 tuntia viikossa. XP:n kehityksen aikana tämä tuntimäärä on osoittautunut parhaimmaksi. Mikäli tehdään enemmän työtunteja viikossa, kasvaa riski, että tehdään liian paljon ohjelmakoodia, joka voitaisiin toteuttaa nopeammin ja kenties yksinkertaisemmin. Työajan pituudella on merkitystä myös ohjelmakoodin virheiden määrään, testauksen laatuun ja yleiseen ohjelmakoodin laatuun. (Jeffries ym. 2006, 81.)

2.3 Scrum -menetelmä

Scrum -menetelmä kuuluu pääluvussa esiteltyihin ketteriin menetelmiin ja on toinen laajimmin käytössä olevista ketteristä menetelmistä. Scrum -menetelmä pyrkii lähestymään ohjelmistokehitysprojektia käytännönläheisesti (empiirisesti), kuitenkin pyrkien hallitsemaan projektia prosessin omaisesti. Käytännönläheisen ja kokemusperäisen lähestymistavan avulla pyritään saavuttamaan joustavuutta, mukautuvuutta ja tuottavuutta ohjelmistokehitykseen. Scrum ohjelmistokehitysmallia on luonnehdittu enemmän projektitason ohjelmistokehitysmalliksi. (Schwaber & Beedle 2002, 106 - 107.)

Scrum -menetelmän keskeisiä painoarvoja ovat tekijät ja teknologiat. Scrum pyrkii ohjaamaan ajattelua siten, että keskitytään tekemään sitä mitä voidaan ja mihin pystytään. Scrum -menetelmässä pyritään kannustamaan tekijöitä löytämään parhaimmat ratkaisut ja tavoitteet pyritään muodostamaan siten, että tekijöillä on vapaat kädet keksiä ratkaisu. Toisaalta kaikki ohjelmistokehittäjät eivät välttämättä pysty Scrum -menetelmän mukaiseen ohjelmistokehitykseen, jossa panostetaan tekijöiden omaan luovuuteen. (Schwaber 2003, 33, 57 - 60.)

Scrum -menetelmä sisältää monia piirteitä, jotka heijastavat sen kuuluvuutta ketteriin menetelmiin. Scrum -menetelmän ydinajatuksena on sprintti, ennalta määritelty ajanjakso, jonka aikana pyritään toteuttamaan ohjelmiston täydennys eli inkrementti (kuvio 3.). Scrum -menetelmän sprintti noudattaa pitkälti aiemmin esiteltyjä ketterien menetelmien iteratiivisia ja inkrementaalisia piirteitä. (Abrahamsson ym. 2002, 27.)



Kuvio 3. Scrum -menetelmän sprintin rakenne (Reaktor Innovations, 2006)

Kuviossa 3. esitetään Scrum -menetelmän toimintamalli. Sprintti on kuviossa keskimmäisenä kuvattu 1 - 4 viikon pituinen jakso. Jokaiselle sprintille muodostetaan sprintin tehtäväluettelo tuotteen ominaisuusluettelosta. Sprintissä toteutettavat ominaisuudet valitaan sprintin tehtäväluetteloon tiimin toimesta ja tiimi huomioi ominaisuuksille annetut prioriteetit valintoja tehdessään. Toteutettavat ominaisuudet muotoillaan tehtäviksi tiimin pitämässä palaverissa, jonka jälkeen aloitetaan sprintti. Sprintin tarkoituksena on toteuttaa tehtäväluetteloon valitut kohteet. (Reaktor Innovations, 2006; Schwaber & Beedle 2002, 29 - 30.)

Sprintti jakaantuu yksittäisiin 24 tunnin jaksoihin. Jokaisen 24 tunnin aikana tiimin jäsenet pyrkivät toteuttamaan hyvin pieniä osia kehitteillä olevan ohjelmiston vaatimasta ohjelmakoodin tuottamisesta. Jokaisen 24 tunnin jakson jälkeen pidetään päivittäinen seurantapalaveri, jossa käsitellään edellisen 24 tunnin aikaansaannokset. (Abrahamsson ym. 2002, 29 - 30.)

Sprintti päättyy ennalta sovittuna ajankohtana, jonka jälkeen pidetään päätöspalaveri. Päätöspalaverissa käydään läpi sprintin aikana toteutettu ohjelmisto ja käydään keskustelua sprintistä. Tämän jälkeen aloitetaan uusi sprintti ja rakennetta toistetaan niin pitkään kuin toteuttamista riittää. (Schwaber & Beedle 2002, 54 - 55.)

Johtamisen kannalta Scrum -menetelmä vaatii erilaista asennoitumista perinteisiin ohjelmistokehitysmalleihin nähden. Scrum -menetelmässä johto nähdään enemmän urheiluvalmentaja -tyyppisenä henkilönä, jonka tulisi kannustaa tiimiä ja tarjota tiimille mahdollisuus toteuttaa niitä asioita, jotka tiimi katsoo tarpeelliseksi tavoitteen saavuttamisen kannalta. Scrum -menetelmässä painotetaan myös tiimin vapautta päättää asioista ja vastuuta toteuttaa suunniteltu ohjelmisto, joka ilmenee mm. sprintin muodossa. (Schwaber & Beedle 2002, 68 - 69; Schwaber 2007, 4 - 7.)

Yhteenvetona Scrum -menetelmä koostuu tuotteen ominaisuusluettelosta, sprintin tehtäväluettelosta, sprintistä ja yksittäisestä 24 tunnin jaksosta. Sprintin aikana ohjelmistoon toteutetaan ennalta määritellyt täydennykset. Useiden sprinttien kautta muodostuu kehitettävänä oleva ohjelmisto.

2.3.1 Roolit

Scrum -menetelmän sisäinen roolirakenne on hyvin yksinkertainen. Scrum -menetelmässä esiintyvät roolit ovat tekijöistä muodostuva *tiimi*, *tuotteen omistaja* ja *Scrum -mestari*. Tulkinta tavasta riippuen Scrum -menetelmän rooleihin voidaan lisäksi mainita asiakas, käyttäjä sekä johto (management). Jälkimmäisenä mainittujen roolia ei tarkemmin esitellä, sillä heidän merkitys korostuu päärooleien esittelyn kautta. Ohjelmistoprojektin hallinnolliset tehtävät on jaettu kolmen pääroolin kesken seuraavasti: tuotteen omistaja on vastuussa projektista, tiimi vastaa käytännön toteutuksesta ja Scrum -mestari vastaa Scrum -prosessista. Seuraavissa kappaleissa käydään tarkemmin läpi roolien sisältöä ja sitä, millaisista tavanomaisista henkilöistä Scrum -menetelmän roolit muodostuvat. (Abrahamsson ym. 2002, 30; Schwaber & Beedle 2002, 7 - 8; Schwaber 2007, 106.)

Tiimi (Team)

Tiimi sisältää ne henkilöt, jotka käytännössä toteuttavat ohjelmiston tekemisen. Tiimin rakenteeseen Scrum -menetelmä tarjoaa muutamia suosituksia. Tehokkaan Scrum tiimin

koko tulee olla enimmillään 8 henkilöä ja tiimin olisi hyvä rakentua eri alojen osaajista. Ohjelmistokehittäjät, testaajat ja tekniset kirjoittajat tulisivat olla samassa tiimissä. Tämän avulla vaikutetaan merkittävästi siihen, että jokaisen sprintin aikana tehty tuotteen täydennys on toimiva ja dokumentoitu riittävällä tarkkuudella. Tavoitteena on, että toteutettuun täydennykseen ei ole tarvetta palata seuraavien täydennysten aikana elleivät ohjelmiston vaatimukset muutu. (Ketterät Käytännöt 2008; Schwaber 2007, 22 - 23; Schwaber & Beedle 2002, 36 - 37.)

Scrum -menetelmän periaatteiden mukaisesti tiimin tulee olla itsenäinen yksikkö, joka muotoilee tuotteen ominaisuusluettelosta tuotteen täydennyksen jokaisen sprintin aikana. Tiimillä tulee olla tarpeeksi valtaa organisaation sisällä toteuttaa haluamiaan asioita, jotka edistävät ohjelmistotuotteen toteuttamista. Luonnollisesti tiimi on myös vastuussa sprintin ja ohjelmistoprojektin käytännön osuuden (esim. ohjelmakoodi) onnistumisesta. Ainoastaan tuotteen omistaja saa antaa tiimille tehtäviä. Organisaatiolle itselleen tulisi olla myös tämän käytännön selvillä, että tiimiin ei vaikuteta suoraan vaan tuotteen omistajan kautta. (Schwaber & Beedle 2002, 34, 106.)

Tuotteen omistaja (Product Owner)

Tuotteen omistaja ylläpitää kehitettävän ohjelmiston tuotteen ominaisuuslistaa (Product backlog, tarkemmin kappaleessa 2.3.2. tuotteen ominaisuusluettelo). Tuotteen omistaja on yleensä virallisesti vastuussa projektista, johon Scrum -menetelmää käytetään. Tuotteen omistaja on organisaatiossa yleensä tuotepäällikkö tai sisäisessä kehitystyössä projektipäällikkö. Scrum -menetelmän periaatteiden mukaisesti tuotteen omistaja on aina oltava yksittäinen henkilö. Luonnollisesti organisaatiossa voi olla ohjausryhmä, joka vaikuttaa kehitettävänä olevaan ohjelmistoon, mutta viime kädessä tuotteen omistaja tekee päätökset ja kirjaa asiat tuotteen ominaisuusluetteloon. (Schwaber & Beedle 2002, 34 - 35.)

Asiakasprojekteissa tuotteen omistaja toimii läheisesti yhteistyössä asiakkaan kanssa. Asiakas vaikuttaa tilaamansa ohjelmiston kehitykseen tuotteen ominaisuusluettelon kautta, jota ylläpidetään tuotteen omistajan kanssa yhteistyössä. Asiakas tuo haluamansa ominaisuudet kohdiksi tuotteen ominaisuusluetteloon, josta tuotteen omistaja priorisoi kehitettävät kohteet tiimille seuraavaa sprinttiä varten. Asiakkaan suora kontakti kehitystiimin kanssa Scrum -menetelmässä on kielletty. Tällä säännöllä vähennetään kehitystiimin häiriöitä kehitystyössä ja annetaan ryhmälle työrauha. Extreme Programming -menetelmään nähden tämä on suurin

poikkeavuus, sillä Extreme Programming -menetelmässä asiakas on suoraan vuorovaikutuksessa kehitystiimin kanssa. (Schwaber & Beedle 2002, 6, 34 - 35.)

Scrum -menetelmässä tuotteen omistajan tulee toimia mahdollisimman näkyvästi. Kaikki tehdyt päätökset tulisi olla projektiin kuuluvien henkilöiden saatavilla. Sama pätee myös tuotteen ominaisuusluetteloon, jota tuotteen omistaja ylläpitää. Tuotteen omistajan vastuulla on pitää tuotteen ominaisuusluettelo saatavilla organisaation sisäisille henkilöille sekä asiakkaalle. Tällöin sekä asiakas että organisaation henkilöstö pystyvät näkemään mitä tiimi on parhaillaan tekemässä kyseiseen projektiin. (Schwaber & Beedle 2002, 34 - 35.)

Scrum -mestari (Scrum master)

Scrum -mestari on henkilö, joka vastaa Scrum prosessista. Scrum -mestari muistuttaa perinteistä projektipäällikköä, mutta tämä rinnastus ei ole täysin osuva Scrumin kohdalla. Scrum -mestarin tehtävänä on toimia tiimin ”valmentajana” perinteisen käskevän esimiesaseman sijaan. Scrum -mestarin tehtäviin kuuluu kannustaa tiimiä toimimaan itsenäisesti, etsimään luovia ratkaisuja ja tarvittaessa ratkaista tiimin kohtaamia esteitä, jotka estävät tiimin kehitystyön toteuttamista. Scrum -mestari toimii myös tuotteen omistajan kanssa yhteistyössä tuotteen ominaisuusluetteloa laadittaessa ja priorisoinnissa sekä varmistaa, että tuotteen omistaja ohjaa tiimin toteuttamaa tuotekehitystä. Scrum -mestarin tehtäviin kuuluu myös varmistaa, että tiimi saa tarvittavan työrauhan organisaation sisällä. Myös Scrum -menetelmän sääntöjen ja menetelmien noudattamisen seuraaminen kuuluu Scrum -mestarin tehtäviin.(Schwaber & Beedle 2002, 31 - 32; Schwaber 2003, 25 - 26, 36.)

Tärkeimpiin Scrum -mestarin tehtäviin kuuluu päivittäisen seurantapalaverin johtaminen. Scrum -mestarilta vaaditaan johtamistaitoja sekä uskallusta puuttua asioihin, jotka vaikuttavat suorasti tai epäsuorasti Scrum -menetelmän prosessiin.

2.3.2 Tuotteen ominaisuusluettelo (Product backlog)

Tuotteen ominaisuusluettelo (Product backlog) sisältää kaikki kehitettävänä olevat ohjelmiston piirteet ja ominaisuudet mahdollisimman tarkasti kuvailtuna. Ominaisuusluettelo toimii myös asiakkaan vaikutuskanavana toteutettavalle ohjelmistolle, mikäli kyseessä on asiakasprojekti. Tuotteen ominaisuusluetteloa ylläpitää tuotteen omistaja ja tarvittaessa Scrum -mestari osallistuu ominaisuusluettelon kohteiden tarkentamiseen ja priorisointiin.

Mikäli kyseessä on projekti johon osallistuu asiakas (organisaation ulkoinen), niin tällöin asiakas osallistuu tuotteen ominaisuusluettelon kohteiden priorisointiin ja ylläpitoon. Ominaisuusluettelon kohteiden priorisoinnilla varmistetaan, että kehitystiimi tekee juuri niitä asioita mitkä ovat tärkeitä. (Schwaber & Beedle 2002, 32.)

Scrum -menetelmän periaatteiden mukaisesti kehitystiimi valitsee ominaisuusluettelosta ne kohteet työnalle, jotka nähdään mahdollisiksi toteuttaa sen hetkisillä tiedoilla seuraavan sprintin aikana. Kun ominaisuusluettelon kohteet on valittu, niitä ei saa muuttaa kesken sprintin. Tämä ehto varmistaa kehitystiimin työrauhan valituiden kohteiden osalta ja pureutuu perinteiseen ongelmaan, jossa ominaisuudet muuttuvat kesken toteutuksen ja aiheuttavat projektin viivästymistä. (Schwaber & Beedle 2002, 32 - 33; Schwaber 2004, 10 - 11.)

Tuotteen ominaisuusluettelo voidaan rakentaa useiden erilaisten tietolähteiden perusteella. Tavanomaisin perustamistapa on purkaa vaatimusmäärittely erilaisiksi kohdiksi ominaisuusluetteloon. Tuotteen ominaisuusluettelon laadintaan voivat osallistua useat eri tahot, joista yleisin osallistuva taho on asiakas, joka esittää haluamansa ominaisuudet ja vaatimukset ohjelmistolle. Yrityksen myynnillisiltä henkilöiltä saadaan tietoa siitä, miten ohjelmisto olisi kilpailukykyinen muihin verrattuna tai millaisia ominaisuuksia asiakkaat odottavat. Tuotekehityksen henkilöt täydentävät tuotteen ominaisuusluetteloa teknologian osalta ja myöhemmässä vaiheessa tekninen tuki lisää ominaisuusluetteloon käytössä ilmenevät virheet, jotka on korjattava ohjelmistoon.(Schwaber & Beedle 2002, 33 - 35.)

Scrum -menetelmän käyntiin saattamiseksi tarvitaan ainoastaan ensimmäiseen sprinttiin tarvittavat kohdat, jonka jälkeen kehitystiimi voi aloittaa työskentelyn. Tällä aikaa muut tahot (mm. tuotteen omistaja, asiakas) voivat täydentää tuotteen ominaisuusluetteloa haluamillaan ominaisuuksilla tai toiminnoilla. Tuotteen ominaisuusluettelo on hyvin dynaaminen ja muuttuu koko tuotteen kehitysprojektin ajan. Tämä on eräs merkittävistä Scrum -menetelmän vahvuuksista verrattuna perinteisiin ohjelmistokehitysmalleihin (mm. vesiputousmalli), joissa pyritään aluksi määrittelemään mahdollisimman tarkasti toteuttavat ominaisuudet ja alun jälkeen muutoksia ei oteta vastaan tai niihin reagoidaan negatiivisesti. (McConnell 2002, 136 - 139; Schwaber & Beedle 2002, 33 - 35.)

Tuotteen ominaisuusluettelo sisältää kehitettävänä olevan ohjelmistotuotteen:

- ominaisuudet (features)

- toiminnallisuudet (functionality)
- teknologiat (technology)
- parannukset (enchancements)
- virhekorjaukset (bug fixes), jotka vaikuttavat myös seuraaviin versioihin
- ongelmat (issue), jotka estävät tiettyjen asioiden toteuttamisen tai koko ohjelmiston toteuttamisen.

Kokoamalla kaikki ohjelmiston piirteet yhteen luetteloon saadaan kehityksestä helpommin hallittavaa. Tuotteen ominaisuusluettelon tarjoaa myös ohjelmistokehityksen jälkeiseen toimintaan hyvän pohjan, sillä tuotteen ominaisuusluetteloon kerätään käytössä ilmenneet virheet ja niiden korjaukset. Tällöin tukihenkilöiden työskentely tehostuu ja ohjelmiston kokonaiskuva on helpommin hahmotettavissa. (Schwaber & Beedle 2002, 33 - 35.)

2.3.3 Sprintti (Sprint)

Sprintti on Scrum -menetelmässä ajanjakso, jolloin kehitystiimi toteuttaa tuotteen täydennyksen. Scrum -menetelmässä sprintin pituudeksi suositellaan 30 päivää. Kokeneemmilla Scrum tiimeillä tämä ajanjakso voi vaihdella. Sprintin aikana kehitystiimi ja Scrum -mestari ovat vastuussa sprintin onnistumisesta. Kehitystiimin tulisi saada työskennellä keskeytyksettä valitsemissaan tehtävissä, jotka tähtäävät tuotteen täydennykseen sprintin aikana. (Schwaber & Beedle 2002, 47 - 48.)

Sprintti aloitetaan palaverilla, jossa kehitystiimi käy asiakkaan, tuotteen omistajan ja muiden mahdollisten tahojen kanssa läpi tuotteen ominaisuusluettelon kohteet, jotka olisi hyvä toteuttaa seuraavassa sprintissä. Tämän jälkeen kehitystiimi pitää sisäisen palaverin tuotteen omistajan ja Scrum -mestarin kanssa siitä, mitkä kohteet valitaan tuotteen ominaisuusluettelosta. Samassa palaverissa myös aletaan muotoilla sprintin tehtäväluetteloa. Tässä palaverissa huomioidaan kehitystiimin aiemmat sprintit ja niistä saadut kokemukset. Käytettävissä olevien tietojen avulla pyritään arvioimaan sprintin tehtävät ja se, pystytäänkö valitut ominaisuudet saavuttamaan sprintin aikana.(Schwaber & Beedle 2002, 47 - 48.)

Sprintille on tehtäväluettelon lisäksi hyvä asettaa tavoite. Sprintin tavoitteen tulee olla mahdollisimman selkeä ja ymmärrettävissä oleva. Tavoitteen avulla tiimi saa liikkumavaraa sprintin aikana. Tavoite auttaa kehitystiimiä motivoitumaan sprintin aikana ja antaa tiimille mahdollisuudet toteuttaa erikoisia ratkaisuja, kunhan tavoite täyttyy.(Schwaber & Beedle 2002, 47 - 48.)

Kun sprintin tehtäväluetteloon on saatu tarvittavat kohteet ja sprintillä on tavoite, voidaan aloittaa sprintti. Sprintin aikana tiimi toteuttaa sprintin tehtäväluetteloon kerätyt yksittäiset tehtävät muodostaen näistä tuotteen täydennyksen. Sprintin tarkoituksena on viedä loppuun asti jokainen tuotteen täydennys, siten, että toteutettuihin kohtiin ei tarvitse palata ohjelmiston myöhemmissä kehitysvaiheissa. (Schwaber & Beedle 2002, 47 - 48.)

On kuitenkin hyvä huomioida, että kehitettävänä olevan ohjelmiston arkkitehtuuria ei rakenneta yhden sprintin aikana. Scrum -menetelmässä arkkitehtuuri ja ohjelmiston malli muodostuvat useiden sprinttien kautta inkrementaalisesti rakentuen. (Schwaber & Beedle 2002, 9.)

Sprintin päättymisen ajankohta on aina selvillä. Sprintin päätöksessä pidetään palaveri, jossa kehitystiimi esittelee sprintin aikaansaannokset. Päätöspalaverit ovat yleensä julkisia tilaisuuksia, johon saa periaatteessa kuka tahansa osallistua ja on erittäin suositeltavaa, että asiakas osallistuu tällaiseen palaveriin. Aikaansaannosten tarkastelun jälkeen päätöspalaverissa arvioidaan sprintin onnistumista kehitystiimin osalta ja aikaansaannosten osalta. Tällöin asiakas voi esittää näkemyksensä tuotteen täydennyksen onnistuneisuudesta. Mikäli aikaansaannokset eivät täytä osapuolien odotuksia, niistä voidaan lisätä uusia kohtia tuotteen ominaisuusluetteloon odottamaan seuraavaa sprinttiä. (Schwaber & Beedle 54 - 56.)

Kuitenkin on mahdollista, että sprintti päättyy ennen aikojaan. Mikäli kehitystiimi ei pysty saavuttamaan valitsemiaan kohteita sprintin aikana eikä valittuja ominaisuuksia voida keventää siten, että ne olisivat toteutettavissa, voidaan sprintti keskeyttää. Sprintin keskeyttäminen on Scrum -menetelmässä rinnastettu hyvin vakavaksi toimenpiteeksi, jota ei tulisi tehdä kevyin perustein. Sprintin keskeyttämisestä päättää tiimi, Scrum -mestari ja tuotteen omistaja yhdessä. Tarvittaessa Scrum -mestari voi päättää sprintin, mikäli sprintin aikana ilmenee tarpeeksi syitä. Syinä voivat olla mm. kehitystiimin sisäiset ongelmat tai organisaation ongelmat, jotka häiritsevät sprinttiä. (Schwaber & Beedle 2002, 53 - 54.)

Sprintin aloituspalaverin jälkeen tiimi käy sisäisesti läpi kohdat, jotka otetaan tuotteen ominaisuusluettelosta seuraavaan sprinttiin mukaan. Kohteiden valinnan jälkeen tiimi alkaa työstää sprintille tehtäväluetteloa (Sprint backlog). Sprintin tehtäväluettelo sisältää sprintin aikana toteutettavat pienet tehtävät, joiden tarkoituksena on sprintin jälkeen muodostaa tuotteen täydennys. (Schwaber & Beedle 2002, 49 - 50.)

Scrum -menetelmässä kehitystiimillä on haastava tehtävä muotoilla tuotteen ominaisuusluettelolla olevat kohteet tehtäviksi. Tehtävän olisi hyvä olla kestoltaan 4 - 16 tunnin mittainen, jolloin se mahtuu päivittäisen seurantapalaverin sykliin. Tehtävä voi olla esimerkiksi teknologian tutkimista tai ohjelman osan tekemistä. Sprintin tehtäväluettelon ei tarvitse olla täydellinen ennen sprintin aloittamista. Sprintin tehtäväluetteloa voidaan täydentää tarvittaessa. Tehtäväluettelon kohteita voidaan jopa poistaa, mikäli siihen on painavat syyt. Tällöin päätös tehdään yhdessä tuotteen omistajan ja Scrum mestarin kanssa, sillä yleensä poistettava tehtävä siirtyy seuraavaan sprinttiin. (Schwaber & Beedle 2002, 49 - 50.)

2.3.4 Päivittäinen seurantapalaveri (Daily Scrum)

Päivittäinen seurantapalaveri on Scrum -menetelmän tarjoama mahdollisuus seurata kehitystiimin etenemistä sprintin aikana ja kuuluu Scrum -mestarin vastuulle järjestää. Päivittäinen seurantapalaveri on Scrum -menetelmässä määritelty hyvinkin tarkkaan, sillä se on eräs merkittävimpiä Scrum -menetelmän onnistumisen työkaluja. Päivittäisen seurantapalaverin tulisi kestää maksimissaan 15 minuuttia tiimin koosta riippumatta. Tiimin jokaisen henkilön tulisi osallistua tähän palaveriin joko paikan päällä tai etäyhteyksien kautta. Päivittäisen seurantapalaverin tulisi tapahtua jokaisena päivänä sprintin aikana ennalta sovitussa paikassa, sovittuna ajankohtana.(Beedle & Schwaber 2002, 40 - 42.)

Päivittäistä seurantapalaveria johtaa Scrum -mestari. Palaverissa käydään läpi jokaisen tiimin jäsenen osalta seuraavat kysymykset:

- ”Mitä olet tehnyt edellisen seurantapalaverin jälkeen?”
- ”Mitä aiot tehdä seuraavaan seurantapalaveriin mennessä?”
- ”Mikä vaikeutti sinun työskentelyä?”

Viimeisin kysymys esitetään, mikäli edistystä ei ole tapahtunut edellisen palaverin jälkeen. Tämän palaverin avulla Scrum -mestari pystyy seuraamaan kehitystiimin etenemistä ja huomaa nopeasti, mikäli jokin haittaa kehitystiimin työskentelyä tai aiheuttaa töiden pysähtymisen. Toisaalta tämä myös motivoi tiimin jäseniä tekemään töitä, sillä seuraavassa palaverissa pitää esittää muille mitä on saanut aikaan. Palaverin avulla tiimin muut jäsenet tietävät, mitä on työn alla ja kuinka siinä edistytään. Korkeampana tavoitteena seurantapalaverilla on parantaa kehitystiimin sisäistä kommunikointia ja ohjata tiimiä toimimaan paremmin yhteistyössä. (Beedle & Schwaber 2002, 40 - 46.)

Mikäli seurantapalaverissa ilmenee jokin asia, mitä pitää käsitellä tarkemmin, tulee sille järjestää erikseen oma palaveri. Scrum -menetelmässä pyritään pitämään päivittäinen seurantapalaveri mahdollisimman hallittuna ja kompaktina. On myös mahdollista, että päivittäiseen seurantapalaveriin osallistuu myös muita henkilöitä kuin tiimin jäsenet. Mikäli asiakas tai esimiehet haluavat tietää missä mennään projektissa, on hänellä mahdollisuus osallistua päivittäiseen seurantapalaveriin kuuntelijoina. Ulkopuoliset osallistujat eivät saa osallistua varsinaisesti päivittäiseen seurantapalaveriin tai suoraan keskustella tiimin jäsenten kanssa projektista. Mikäli ulkopuolisella osallistujalla on jotain huomautettavaa palaverin perusteella, tulee hänen keskustella siitä Scrum -mestarin tai tuotteen omistajan kanssa palaverin jälkeen. (Beedle & Schwaber 2002, 40 - 46.)

2.3.5 Ohjelmistokehityksen riskien hallinta Scrum -menetelmässä

Riskien hallinta Scrum -menetelmässä lähtee siitä perusoletuksesta, että toteutetaan tuotteen kehittämistä, ei valmistamista. Tällöin oletetaan, että mukana on huomattava osuus tutkimusta ja luovuutta, jotka asettavat vaatimuksia tiimien itse organisoitumiseen, oppimiseen ja lomittain tai päällekkäin tapahtuvaan kehittämiseen. Edellä mainittujen tekijöiden ennustettavuus ja suunniteltavuus on usein haasteellista tai mahdotonta. Tähän ongelmaan Scrum -menetelmä tarjoaa useita mahdollisia ratkaisuja, joiden avulla voidaan arvioida, seurata, suunnitella ja hallita ohjelmistokehitystä. Näitä ratkaisuja ovat esimerkiksi päivittäinen seurantapalaveri ja sprintti. (Schwaber & Beedle 2002, 109 - 110.)

Asiakkaan tyytymättömyys ohjelmistoon on yksi erittäin merkittävä riski. Tätä voidaan hallita tarjoamalla asiakkaalle mahdollisuus seurata ohjelmiston kehittämistä jatkuvasti säännöllisin määräajoin. Scrum vaatii, että asiakas osallistuu ainakin jokaisen sprintin päättävään

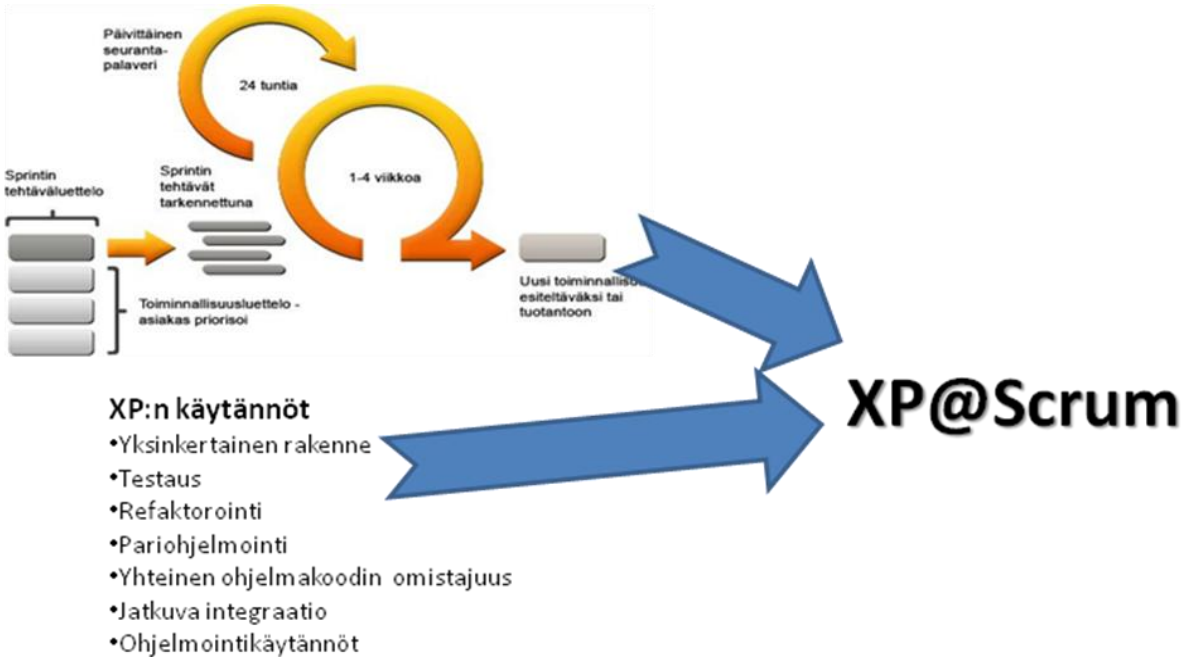
palaveriin. Tällöin on yleensä nähtävillä toimiva osa ohjelmasta jolloin asiakas on paremmin ajan tasalla. Mikäli kehitettävänä olevasta ohjelmistosta huomataan puutteita tai löydetään uusia kehitettäviä ideoita, näitä voidaan ottaa mukaan seuraavassa sprintissä. (Schwaber & Beedle 2002, 109.)

Ohjelmiston tärkeiden ominaisuuksien ja toimintojen toteuttamatta jääminen on eräs ja suhteellisen yleinen riskitekijä ohjelmistoalalla. Tähän ongelmaan Scrum -menetelmä tarjoaa varsinaisena ratkaisuna aiemmin käsiteltyä sprinttiä. Sprintin avulla pystytään seuraamaan ominaisuuksien kehittymistä ohjelmistossa ja varmistaa, että kaikki tärkeät piirteet ovat mukana ohjelmistossa priorisoimalla toteutettavia piirteitä sprinttien aikana. Näin ollen ainoastaan vähemmän tärkeät ominaisuudet voivat jäädä ohjelmiston ulkopuolelle, mikäli kaikki osapuolet osallistuvat aktiivisesti sprinttien palavereihin. (Schwaber & Beedle 2002, 109.)

Scrum -menetelmän päivittäisellä seurantapalaverilla voidaan vähentää riskiä, joka kohdistuu *huonoon ennakointiin ja suunnitteluun*. Suunnittelusta johtuviin riskeihin ketterät menetelmät tarjoavat osissa tapahtuvaa kehitystä, joka sallii toteutettuun ominaisuuteen palaamisen ja sen täydentämisen. (Schwaber & Beedle 2002, 109.)

2.4 Extreme Programming ja Scrum

Extreme Programming ja Scrum -menetelmä ovat laajimmin käytössä olevia ketteriä menetelmiä. Nämä kaksi menetelmää poikkeavat toisistaan siten, että Extreme Programming keskittyy enemmän ohjelmiston tekemiseen ja Scrum keskittyy enemmän ohjelmistoprojektin hallintaan. Useissa eri yhteyksissä näitä kahta ohjelmistokehitysmallia on pyritty yhdistämään ja ottamaan käyttöön kummankin ohjelmistokehitysmallin parhaat ominaisuudet. Kuvio 4. esittelee näiden kahden ohjelmistokehitysmallin vahvuuksia. Scrum -menetelmän vahvuutena pidetään projektinhallintamalleja ja XP:n vahvuutena ovat käytännöt, joita suositellaan käytettäväksi ohjelmakoodin toteuttamisessa. (Control Chaos. 2008; Koskela, 2008.)



Kuvio 4. Scum ja XP (mukailtuna Control Chaos ja Reaktor Innovations 2008)

Tavallisimmin näitä kahta ohjelmistokehitysmallia yhdistetään siten, Scrum ohjelmistokehitysmallista otetaan projektin hallintaan soveltuvat osat (mm. tuotteen ominaisuusluettelo ja sprintti) ja XP ohjelmistokehitysmallista käytännöt. XP:sta otettujen käytäntöjen tarkoituksena on vaikuttaa ohjelmoijien käytännön toteutukseen ja vahvistaa tätä osa-aluetta. (Control Chaos, 2008; Koskela, 2008)

3 DEVIATIONS OHJELMISTO

Tämän opinnäytetyön tarkoituksena oli toteuttaa ohjelmistoprojekti nimeltä Deviations. Ohjelmiston toteutuksessa sovellettiin ketteriä menetelmiä ja hyödynnettiin Documenta Oy:n Deviations for Domino -ohjelmistoa. Deviations for Domino -ohjelmisto on vastaava laatupoikkeamien ja palautteiden hallintaohjelmisto IBM Domino -alustalla. Tämän opinnäytetyön kannalta kyseistä ohjelmistoa voitiin hyödyntää ainoastaan korkealla tasolla, sillä toteutusalustat poikkeavat merkittävästi toisistaan.

Opinnäytetyön teknisessä toteutuksessa käytettiin Microsoftin Windows SharePoint Services 3.0 (WSS 3.0) -alustaa sekä Microsoftin kehittämää C# ohjelmointikieltä. Ohjelmointi WSS 3.0 -alustalle muistuttaa pitkälti perinteistä ASP.NET ohjelmointia. Deviations -ohjelmiston rungon muodostavat web-osat (Web Parts) ja työnkulut (Workflow).

Deviations -ohjelmistossa käytetään termejä, jotka on luettavuuden kannalta hyvä selventää. Ohjelmistossa yksittäistä poikkeamaa (esimerkiksi asiakasreklamaatio tai palaute) nimitetään *tapaukseksi*. Tapaukseen kohdistuva toimia kutsutaan *toimenpiteiksi*, jotka voivat olla esimerkiksi selvityspyyntöjä tai korjaavia toimenpiteitä.

3.1 Määrittelyt

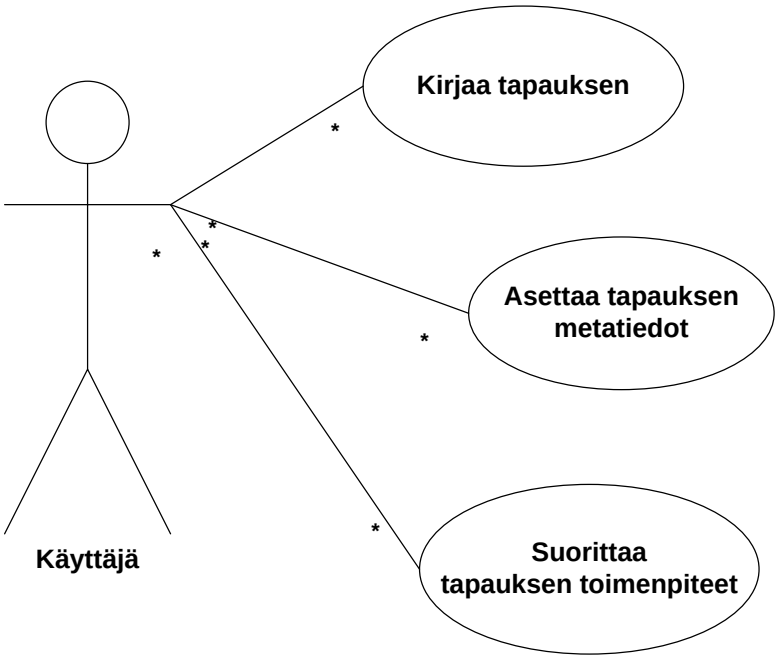
Deviations -ohjelmiston toteutus alkoi toukokuussa 2008. Määrittelyssä oli kaksi tasoa: käytännön tekninen osuus ja korkeamman tason käyttötapauksien määrittely. Ohjelmiston teknisessä toteutuksessa merkittävänä haasteena oli ketterien menetelmien periaatteiden mukaisesti tekemisen minimointi siten, että toteutetaan oikeanlaista ohjelmakoodia, jota voidaan käyttää uudelleen mahdollisimman hyvin. Tämä tavoite palveli myös aikataulutusta ja ohjelmiston saantia myynnin käyttöön mahdollisimman nopeasti.

Vaatimusmäärittelyssä nousi esille muutamia toteutukseen merkittävästi vaikuttavia asioita. Eniten toteutukseen vaikutti se perusnäkemys, että asiakkaalla on jo ennalta olemassa prosessit laatupoikkeamien hallitaan, kuten myös palautteiden hallintaan. Tämän johdosta ohjelmiston tulisi olla joustava ja mukautua mahdollisimman hyvin asiakkaiden olemassa oleviin prosesseihin. Vaatimusmäärittelyssä huomioitiin myös se, että ohjelmistoa ei sidota

käytettäväksi minkään tietyn laatustandardin mukaan, vaan ohjelmiston tulee soveltua käytettäväksi laatustandardeista riippumatta.

Tekniset vaatimukset Deviations -ohjelmistolle asetettiin seuraavasti: toteutuksen tulee toimia Windows SharePoint Services (WSS) 3.0 -alustalla, ohjelmiston tulisi olla helppokäyttöinen, tulisi olla Internet selaimella käytettävissä, selainriippumaton ja ohjelmiston tulisi tukea useita yhtäaikaisia käyttäjiä.

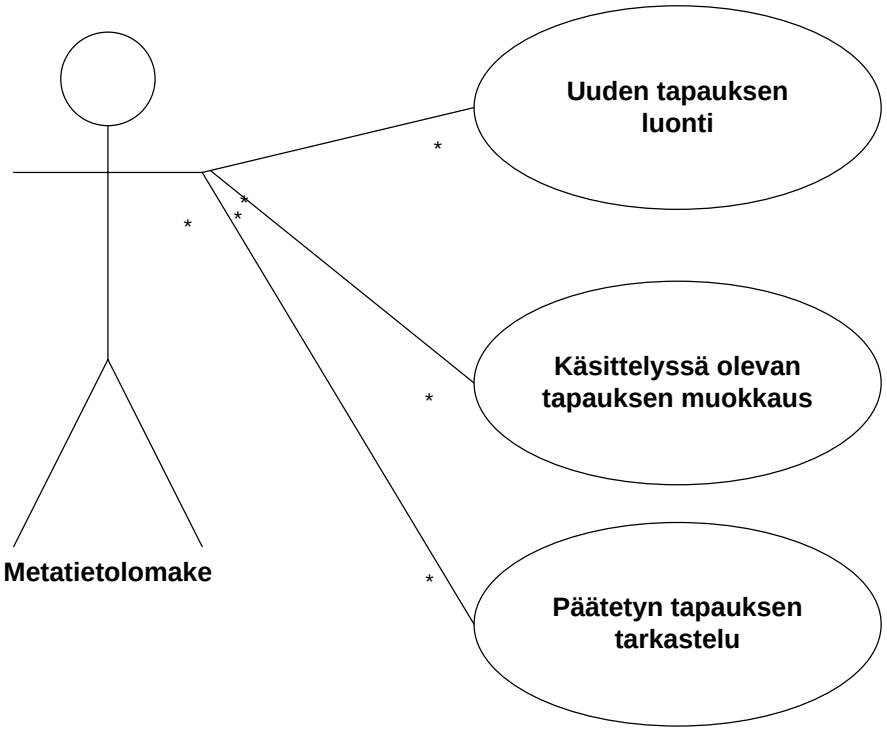
Deviations -ohjelmiston käyttötapaukset laadittiin perustuen vaatimusmäärittelyyn sekä olemassa olevaan Deviations for Domino tuotteeseen. Käyttäjälle selkeimmät käyttötapaukset ovat tapauksen kirjaus, metatietojen asettaminen ja tapauksen toimenpiteiden suorittaminen (kuvio 5). Tapauksen toimenpiteiden suorittaminen voi organisaatiosta riippuen tapahtua toteutetun ohjelmiston ulkopuolella.



Kuvio 5. Käyttäjän käyttötapaukset

Metatietolomakkeen lomakkeen kannalta nähtävät käyttötapaukset (kuvio 6.) tuli huomioida ohjelmakoodin käsittelyssä, sillä ne poikkesivat merkittävästi käyttäjän kannalta nähdyistä käyttötapauksista. *Uuden tapauksen luonti* -käyttötapauksessa luodaan täysin uusi tapaus, jota käytännössä katsoen ei ole olemassa ennen kuin kyseinen tapaus tallennetaan ensimmäisen

kerran. *Käsittelyssä olevan tapauksen muokkaus* -tilanteessa muokataan olemassa olevaa tapausta ja tarvittaessa päivitetään olemassa olevan tapauksen tietoja. *Päätetyn tapauksen tarkastelu* käyttötapauksessa tarkastellaan jo päätettyä tapausta. Tällöin merkittävässä roolissa on tietojen tarkastelu, jonka tulisi olla mahdollisimman helppoa käyttäjälle. Nämä kohdat muodostivat tarpeen tehdä metatietokortin näkökulmasta oma käyttötapauskaavio.



Kuvio 6. Metatietolomakkeen käyttötapaukset

3.2 Ketterien menetelmien soveltaminen

Ketteriä menetelmiä pyrittiin soveltamaan Deviations laatupoikkeamaohjelmiston toteutuksessa. Käytännössä soveltaminen tapahtui suunnittelun, kehittämisen ja testauksen muodostamassa kierroksessa. Tällainen kierros kesti yleensä viikon ja päättyi palaveriin, jossa käytiin läpi kyseisen kierroksen aikaansaannokset.

Deviations -ohjelmiston toteutus aloitettiin muuntamalla muutamia vaatimusmäärittelyn kohtia kokonaisuuksiksi, joita tarkennettiin yksittäisiksi tehtäviksi toteutuksen edetessä. Tehtävä-ajattelussa pyrittiin hyödyntämään Scrum -menetelmää, sillä tuotteella ei ollut

toteutusvaiheessa varsinaista yrityksen ulkopuolista asiakasta. Tällöin projektin ohjaaminen Extreme Programming -periaatteiden ja käytäntöjen mukaan olisi ollut haastavaa.

Varsinaisessa ohjelmakoodin tuottamisessa hyödynnettiin muutamia Extreme Programming -menetelmän käytäntöjä. Hyödynnettyjä käytäntöjä olivat ohjelmakoodin refaktorointi, inkrementaalinen suunnittelu, työajan pituus ja myöhemmissä vaiheissa jatkuvaa integraatiota, kun kokonaisuudet olivat yhdistettävässä kunnossa.

Refaktorointia eli ohjelmakoodin uudelleen kirjoitusta suoritettiin erityisesti ohjelmiston tehokkuutta silmällä pitäen. Deviations -ohjelmiston eräänä käyttötapauksena on puhelimitse ilmoitettavan poikkeaman kirjaus. Tällöin erityisesti ohjelmiston suorituskyvyn ja toimivuuden tulee olla kunnossa. Refaktoroinnissa pyrittiin huomioimaan myös ohjelmiston usean käyttäjän yhtäaikainen käyttö ja toimivuus, jota parannettiin esimerkiksi ohjelman asetuksien lukemisen optimoinnilla ja Internet selaimen välimuistia hyödyntämällä.

Inkrementaalisen suunnittelun käytäntöä toteutettiin koko ohjelmistokehityksen ajan. Tämä onnistui erityisesti sen seikan johdosta, että ohjelmistoa käytännössä toteutti aktiivisesti ainoastaan yksi henkilö. Tällöin ohjelmiston rakenteeseen ja suunnitteluun oli helppoa ja nopeaa tehdä muutoksia. Tosin näin helppo muutosalttius toi myös vähemmän onnistuneita muutoksia, joita jouduttiin korjaamaan ja palauttamaan ohjelmakoodia muutoksia edeltävään tilaan.

Jatkuvan integraation käytäntöä Deviations -ohjelmistossa noudatettiin sitä mukaa, kun kehitettävänä olleita kokonaisuuksia saatiin valmiiksi. Jokaisen valmistuneen kokonaisuuden jälkeen suoritettiin integraatio muihin osioihin ja ilmenneet yhteensopivuusongelmat ratkaistiin ennen kuin aloitettiin seuraavien kokonaisuuksien toteutus.

Muita XP:n käytäntöjä ei tämän ohjelmistokehitysprojektin aikana päästy ottamaan käyttöön. Yhden komponentin osalta ohjelmistoa toteutti toinen ohjelmistokehittäjä ja varsinaisessa asennusvaiheessa oli muita henkilöitä mukana. Nämä seikat huomioon ottaen esimerkiksi pariohjelmointi käytäntö jätettiin tietoisesti käyttöönottamatta.

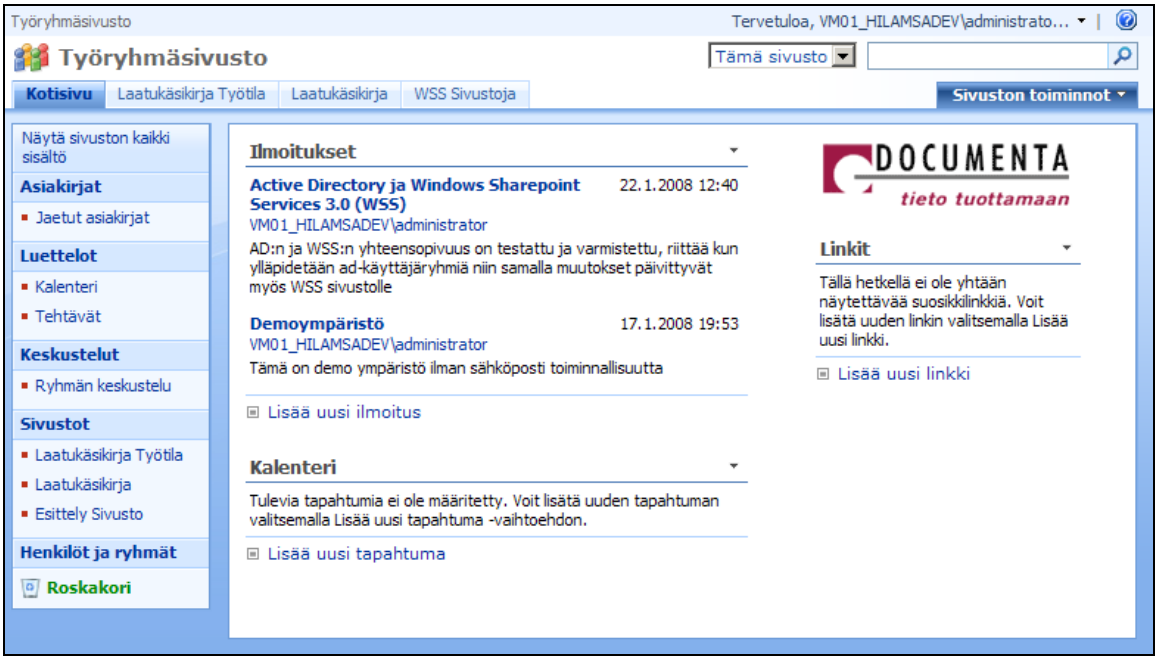
3.3 Tekninen toteutus

Käytännön toteutus aloitettiin elokuussa 2008. Kuitenkin aiemmin toteutettu Laatukäsikirja -tuote vaati jatkokehittämistä, jonka johdosta Deviations -ohjelmistoa pystyttiin toteuttamaan vaihtelevissa sykleissä. Säännöllinen ohjelmistokehitys Deviations -ohjelmistolla alkoi lokakuussa 2008.

Toteutuksessa hyödynnettiin käyttöliittymän osalta Web Parts -tekniikkaa ja työnkulkujen osalta Windows Workflow Foundation -kirjastoa. Yhteisinä piirteinä kaikille toteutuksen osille voidaan nimetä ohjelmointikieli, -työnkalut ja -ympäristö. Ohjelmointikielenä toteutuksessa oli C#, työkaluina Microsoft Visual Studio 2005 ja Visual Studio 2008 -ohjelmistot ja käyttöjärjestelmänä Windows Server 2003 käyttöjärjestelmä virtualisoituna. Virtualisoinnin avulla opinnäytetyön toteutusvaihetta pystyttiin nopeuttamaan merkittävästi ja ohjelmointityökaluja pystyttiin hyödyntämään parhaiten.

3.3.1 Toteutusalusta

Deviations -ohjelmiston toteuttamisessa käytettiin Microsoftin kehittämää Windows SharePoint Services 3.0 (WSS 3.0) -alustaa. WSS 3.0 on ilmainen, Internet -selaimessa toimiva, palvelinkäyttöjärjestelmään (Microsoft Windows Server 2003 tai Windows Server 2008) asennettava laajennusosa. WSS 3.0 toimii myös runkona Microsoft Office Server System (MOSS) -alustalle. MOSS pohjautuu samoihin teknisiin ratkaisuihin kuten WSS 3.0, mutta on itsenäinen tuote omalla lisensointimallillaan. WSS 3.0 sivuston perusnäkymä on esitetty kuviossa 7.



Kuvio 7. Kuvankaappaus Microsoft WSS 3.0 sivustosta (Windows SharePoint Services 3.0, 2008.)

Microsoft on kehittänyt SharePoint alustaa noin 10 vuotta ja kolmas versio, jota tässä opinnäytetyössä hyödynnettiin, julkaistiin vuoden 2006 lopulla. SharePoint -alustan tarkoituksena on tarjota organisaatiolle (yritykset, julkishallinto) mm. keskitetty sivustorunko Intra- ja Extranet käyttöön, tietosäilö sekä yhteistyötä helpottava työskentelyalusta. Microsoftin näkemyksenä on pyrkiä SharePointin avulla siirtämään osa työasemalla toteutettavista tehtävistä keskitettyyn verkkoympäristöön.(Sterling 2007, 3.)

Teknisiltä ratkaisuiltaan WSS 3.0 perustuu Microsoftin kehittämään ASP.NET 2.0 tekniikkaan ja hyödyntää Microsoft SQL Server tietokantaa. WSS 3.0 on pyritty viemään mahdollisimman helpoksi asentaa ja hallita. WSS 3.0 yhdistää seuraavien palvelinkomponenttien hallinnan: Active Directory (AD), Internet Information Server (IIS) ja SQL Server 2005. WSS 3.0 pystyy hyödyntämään myös Microsoft Exchange -sähköpostipalvelinta, mikäli sellainen on organisaatiossa käytettävissä. Exchange integraation avulla voidaan lisätä saapuvan sähköpostin tuki WSS 3.0 sivustolla olevaan luetteloon. (Sterling 2007, 7, 11; Roini 2007, 21, 23, 197 - 198.)

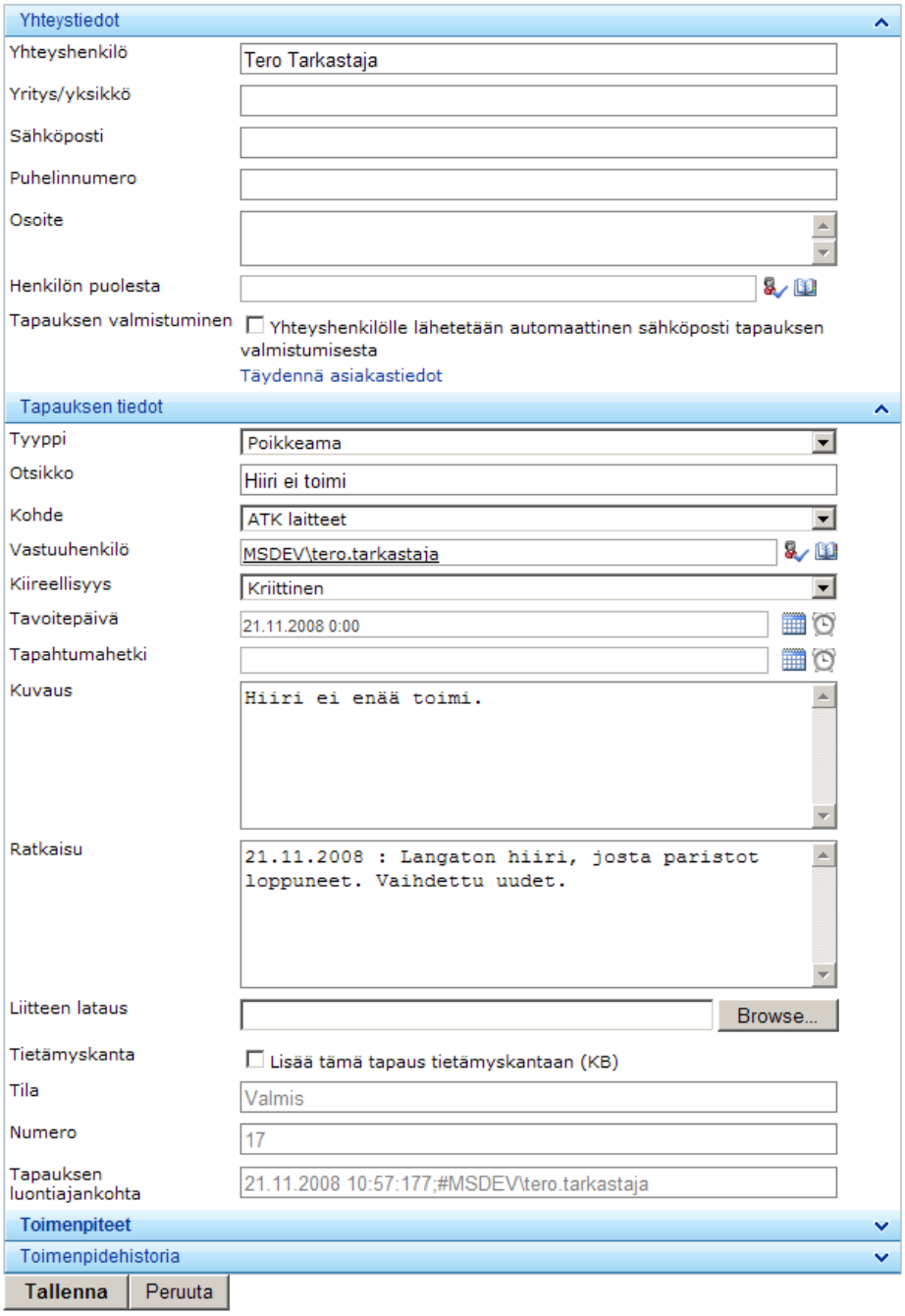
Toimintalogiikaltaan WSS 3.0 noudattaa pitkälti hierarkkista lähestymistä: sivustot ovat toisiinsa nähden hierarkkisessa suhteessa ja sisältötyypit periytyvät pääsisältötyypeistä. Periytyminen alkaa sivustokokoelma tasolta päättyen yksittäisiin objekteihin (luettelon

kohdat, yksittäiset sivut yms.). Hierarkkisuuden avulla saavutetaan alustan helppokäyttöisyyttä ja vähennetään merkittävästi päällekkäisten tehtävien määrää. Esimerkiksi oma sisältötyyppi voidaan periyttää pääsisältötyypistä ja tämän kautta pääsisältötyypin sarakkeet ovat sisältötyypin käytettävissä. (Sterling 2007, 12, 327.)

WSS 3.0 soveltuu seuraavien piirteiden johdosta hyvin Deviations -ohjelmiston toteutusalustaksi: integraatio Microsoft Office tuotteiden kanssa, lisensointimalli, yhteistyö valmiudet käyttäjien välillä ja dokumentoidut rajapinnat, jotka helpottivat ohjelmistokehitystä WSS 3.0 -alustalle. Tekniseltä kannalta WSS 3.0 toteuttaa selainriippumattomuuden vaatimuksen ja soveltuu ilman suuria muutoksia isoille organisaatioille. WSS 3.0:n perustuvat ratkaisut eivät vaadi työasemiin asennettavia laajennuksia, mikä on merkittävä etu isoissa organisaatioissa.

3.3.2 Metatietolomake

Ensimmäisenä toteutuskokonaisuutena oli metatiedon talletuskenttien ja syöttölomakkeiden laadinta. SharePoint 2007 -alusta tukee hyvin vapaavalintaisten metatietokenttien luomista ympäristöön, mutta tiedon syöttölomakkeiden osalta oli jo etukäteen tiedossa tarve tehdä oma toteutus (kuvio 8.). Omien syöttölomakkeiden etuna on lisätä toiminnallisuutta, joka helpottaa ohjelmiston käyttämistä sekä mahdollisuus hallita metatietokenttiä, mikä SharePointin vakiototeutuksella ei ole mahdollista. Hallinnalla tässä yhteydessä tarkoitetaan kenttien muokkausmahdollisuuden rajoittamista, sillä tietyissä tilanteissa osaan kentistä ei tulisi käyttäjän pystyä tekemään muokkauksia.



Kuvio 8. Metatietolomake (Deviations Laatupoikkeamaohjelmisto, 2009)

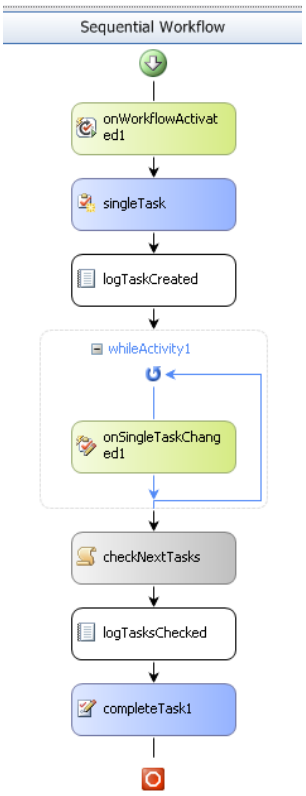
Metatietolomakkeen käytännön toteutuksessa hyödynnettiin Web Parts tekniikkaa. Web Part on itsenäinen komponentti ASP.NET -tekniikalla toteutetulla Internet sivulla. Web Partin avulla voidaan Internet -sivulla toteuttaa toiminnallisuutta, joka on eristetty sivusta ja hyödyntää omaa ohjelmakooditiedostoa. Tätä Web Partin toiminnallisuutta hyödynnettiin Deviations -ohjelmistossa metatietolomakkeen ja muutamien muiden pienempien komponenttien osalta.

Metatietolomake toteutettiin alusta alkaen täyttämään kaksi päätarkoitusta: tiedon syöttäminen ja tiedon tarkastelu. SharePoint -alusta tukee laajasti käyttöoikeuksien määrittelyä, joten metatietolomakkeelle oli oleellista toteuttaa luonteva metatietojen tarkastelu. Tälle lomakkeelle ohjataan käyttäjät, joilla ei ole muokkausoikeuksia Deviations -ohjelmiston tapauksiin.

3.3.3 Työnkulut

Opinnäytetyön toteutuksessa merkittävänä osana olivat työnkulut. Vaatimusmäärittelyssä listattiin useita erilaisia työnkulkuja, joita varten nähtiin järkeväksi rakentaa yksi työnkulku. Toteutuksessa hyödynnettiin Microsoftin Windows Workflow Foundation -kirjastoa.

Työnkulkurunko toteutettiin laatimalla kuvion 9. mukainen työnkulku. Työnkulku saa käynnistyessään parametrina ohjaustiedon, jonka perusteella työnkulku osoittaa tehtävät suorittajille. Tämän jälkeen työnkulku jää odottamaan osallistujien vuorovaikutusta työnkulkuun ja suorittaa toiminnot loppuun osallistujien valintojen perusteella.



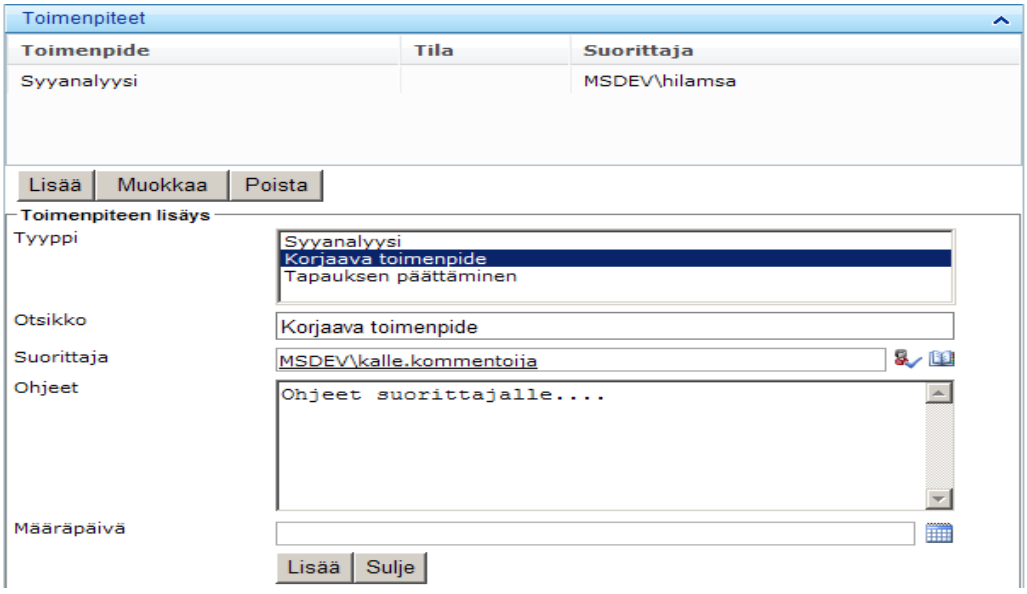
Kuvio 9. Deviations -ohjelmiston työnkulku (Microsoft Visual Studio 2008, 2009)

Yksinkertaisen rakenteen johdosta toteutuksen aikana ilmeni tarve pystyä ohjaamaan työnkulkua. Tarkennettuna tarve oli pystyä muokkaamaan työnkulun tehtäviä, jotka annetaan metatietojen laadinnan yhteydessä. Tämä tarve täytettiin luomalla Deviations -ohjelmiston yksittäiselle tapaukselle oma työnkulkupuskuri. Puskuriin voidaan lisätä ja poistaa tehtäviä sekä muokata jo osoitettuja tehtäviä. Tällöin esimerkiksi työnkulun tehtävän suorittaja voi määrätä seuraavat tehtävät suorittaessaan hänelle osoitettua tehtäväänsä.

3.3.4 Työnkulun ja metatietolomakkeen integraatio

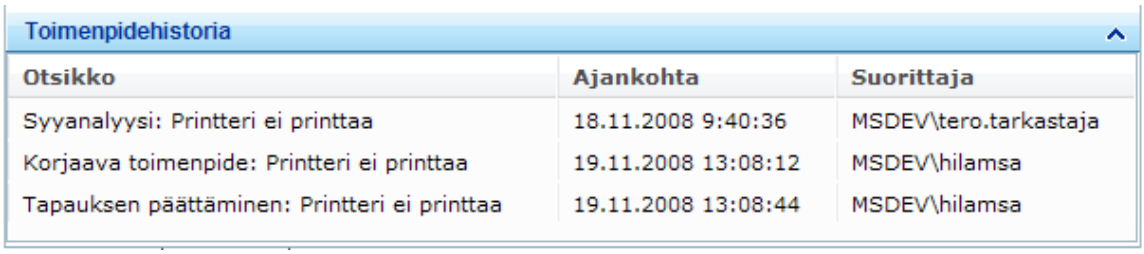
Kolmantena merkittävänä kokonaisuutena Deviations -ohjelmiston toteutuksessa oli aiemmin esiteltyjen kohtien yhdistäminen yhdeksi kokonaisuudeksi XP:n jatkuvan integraation käytännön mukaisesti. Käytännössä tämä yhdistäminen tarkoitti työnkulun ohjausta metatietolomakkeen avulla. Tämän toiminnallisuuden avulla ohjelmiston käytettävyys parani merkittävästi ja vastaa paremmin asiakkaiden todellisia käyttötapauksia ja -tilanteita.

Työnkulun ohjaus tapahtuu XML -muotoisen ohjaustiedon avulla. Työnkulku käynnistetään tallentamalla metatietolomake. Tallennushetkellä metatietolomakkeella tarkastetaan toimenpiteiden tilanne ja toimenpiteet muunnetaan XML -muotoiseksi tiedoksi. Kuviossa 10. esitetään muutamia toimenpiteitä Deviations -ohjelmistossa.



Kuvio 10. Toimenpide (Deviations -ohjelmisto, 2009)

Kuten kuviosta 10. on nähtävillä, yksittäinen toimenpide sisältää kaikki tarvittavat tiedot työnkulun käynnistämiseen ja loppuunsaattamiseen. Tarvittavat tiedot ovat toimenpiteen tyyppi, toimenpiteen otsikko, suorittaja, suoritusohjeet ja määräpäivä. Suoritettu työnkulku tallennetaan tapauksen toimenpidehistoriaan, jota esitellään kuviossa 11.


Toimenpidehistoria

Otsikko	Ajankohta	Suorittaja
Syyanalyysi: Printteri ei printtaa	18.11.2008 9:40:36	MSDEV\tero.tarkastaja
Korjaava toimenpide: Printteri ei printtaa	19.11.2008 13:08:12	MSDEV\hilamsa
Tapauksen päättäminen: Printteri ei printtaa	19.11.2008 13:08:44	MSDEV\hilamsa

Kuvio 11. Toimenpidehistoria (Deviations -ohjelmisto, 2009)

Muutokset käynnissä oleviin työnkulkuihin on mahdollista tehdä metatietolomakkeen kautta. Muutettavan toimenpiteen tietoja muokataan toimenpiteet kohdan kautta ja tietojen tallennushetkellä ohjelmisto lopettaa aiemman työnkulun ja käynnistää uuden työnkulun muuttuneilla tiedoilla.

3.3.5 Sähköpostitoiminnallisuus

Sähköpostitoiminnallisuus oli viimeinen merkittävä kokonaisuus Deviations -ohjelmiston toteutuksessa. Palautteen hallinnan kannalta sähköpostitoiminnallisuus on lähes pakollinen piirre. Sähköpostitoiminnallisuudella tarkoitetaan yksinkertaistettuna tilannetta, jossa asiakas lähettää organisaation sähköpostiosoitteeseen (esimerkiksi tuki@yritys.fi) sähköpostia. Sähköpostiosoitteeseen lähetetty viesti siirtyy automaattisesti palautteenhallintajärjestelmään käsiteltäväksi. Tähän toiminnallisuuteen liittyy myös oleellisesti automaattisen kuittausviestin lähettäminen sähköpostin lähettäjälle.

Nämä toiminnallisuudet toteutettiin Deviations -ohjelmistoon hyödyntämällä WSS 3.0:n sisäänrakennettuja toiminnallisuuksia. WSS 3.0:n sähköpostitoiminnallisuudet sisältävät mahdollisuuden lähettää ja vastaanottaa sähköpostiviestejä. Sähköpostin vastaanottaminen WSS 3.0 -ympäristössä tapahtuu joko Microsoft Exchange sähköpostipalvelinohjelmiston tai paikallisen IIS SMTP -palvelun kautta. Opinnäytetyön versiossa toiminnallisuus toteutettiin hyödyntämällä Microsoft Exchange -vaihtoehtoa.

Käytännössä sähköpostitoiminnallisuus toteutettiin työnkulun avulla. Deviations -ohjelmistoon lisättiin luettelo, joka toimii sähköpostin ensimmäisenä pysäkkinä. SharePoint 2007 -alusta konfiguroitiin ohjaamaan saapuvat sähköpostit edellä mainittuun luetteloon. Luettelosta työnkulku muotoilee sähköpostin tapaukseksi ja siirtää sen varsinaiseen tapausluetteloon. Tämä rakenne helpottaa muutosten tekemisen sähköpostin käsittelyyn ja tarvittaessa kyseiseen luetteloon voidaan kohdistaa esimerkiksi roskapostisuodatuksia, ennen kuin luettelon kohteesta muodostetaan tapaus.

3.4 Testaus

Ohjelmiston testaus suoritettiin ketterien menetelmien periaatteiden mukaisesti. Käytännössä testaus toteutettiin kahdella tasolla. Ensimmäisenä testauksena oli ohjelmakoodin tekijän toteuttama testaus kehitysympäristössä. Tämän jälkeen testattavat ominaisuudet asennettiin erilliselle testauspalvelimelle, jossa testauksen suorittivat Documenta Oy:n myynti- ja projektihenkilöstö. Testauksesta saadut palautteet ohjattiin Documenta Oy:n muutoshallinnan kautta ohjelmistokehitykseen. Muutoshallinnan avulla pyrittiin dokumentoimaan ohjelmistossa ilmenneet virhetilanteet ja hyödyntämään tätä dokumentaatiota ohjelmiston teknisessä tuessa.

Testitapaukset koostuivat useista erilaisista tilanteista. Tavanomaisimpina testitilanteina olivat tapauksen luonti ja muutokset. Kehittyneemmät testitapaukset sisälsivät tapauksen vastuuhenkilöiden muutoksia, käynnissä olevan työnkulun muutoksia ja tapauksen ennenaikaisia päättämisiä.

3.5 Käyttöönotto

Deviations -ohjelmistolle saatiin ensimmäinen asiakas joulukuussa 2008. Asiakkaan kanssa sovittiin ohjelmiston pilottikäyttöjaksosta, jonka aikana ohjelmiston toimivuus ja soveltuvuus asiakkaalle pyrittiin hiomaan kuntoon.

Asiakas otti ensimmäisessä vaiheessa Deviations -ohjelmiston käyttöön organisaationsa mikrotukiosastolla. Ohjelmiston tavoitteena oli ohjata organisaation tukitapausten käsittelyä määrämuotoisemmaksi toiminnaksi ja nopeuttaa tukitapausten käsittelyä. Asiakkaalta saatiin

palautetta koskien ohjelmistoa ja palautteeseen reagoitiin toteuttamalla ohjelmistoon muutokset. Pääasiassa ohjelmistoon tehdyt muutokset olivat ulkoasuun vaikuttavia.

Merkittävänä haasteena käyttöönoton osalta oli asiakkaan SharePoint -ympäristö, joka poikkesi oletetusta SharePoint -asennuksesta. Tämän poikkeavuuden johdosta ohjelmiston asennus kesti suunniteltua pidempään, mutta esteeksi tämä ei muodostunut.

4 POHDINTA

Opinnäytetyön teoriataustaksi valikoituivat ketterät menetelmät juuri siksi, koska halusin perehtyä tekemisen lisäksi myös siihen, kuinka ohjelmistokehitysprosessia voidaan ohjata ja hallita. Tämän teoriataustan valintaan osaksi vaikutti myös aiemmin työharjoittelussa toteuttamani Laatukäsikirja ohjelmisto. Kyseistä ohjelmistoa toteuttaessa huomasin, kuinka tärkeää on ohjelmistoprojektin ja -prosessin hallinta, johon merkittävästi liittyy käytännön toteutuksen ohjaaminen. Myös Documentan puolelta osoitettiin kiinnostuneisuutta ketteriin menetelmiin.

Deviations -ohjelmiston toteuttaminen aloitettiin keväällä 2008 ohjelmiston määrittelypalavereiden kautta. Varsinainen toteutus käynnistyi syyskuussa 2008. Tosin tässä on hyvä huomioida, että aiemmin toteuttamaani Laatukäsikirja -ohjelmistoa jouduttiin ylläpitämään samanaikaisesti Deviations -ohjelmiston kehityksen kanssa.

Käytännön toteutuksessa muutamissa kohdissa toteutusalusta saneli pitkälti toteutustavan, sillä tietyt asiat toimivat eri tavalla SharePoint -alustalla kuin tavallisessa ASP.NET -alustassa. Esimerkkinä tästä voidaan pitää ASP.NET AJAX -toiminnallisuutta, jota ei saatu toimimaan juuri niin kuin tässä versiossa suunniteltiin. Käytännön toteutuksessa törmäsin myös siihen tavalliseen korkean tason ongelmaan, jossa ei tiedetä vastaako toteutustapa asiakkaan odotuksia tai ovatko toteutetut toiminnot asiakkaan tapoja toimia. Projektin aikana kohdattuihin erityisiin haasteisiin tiedon löytäminen osoittautui ongelmalliseksi, vaikka kolmas versio SharePoint -alustasta on ollut yleisesti saatavilla vuoden 2006 lopusta lähtien.

Nähdäkseni tässä opinnäytetyönä toteutetussa ohjelmistoprojektissa toteutuivat hyvin ketterien menetelmien periaatteet ja käytännöt valittujen käytäntöjen osalta. Projektin johdolliselta puolelta ainoat selvät vaatimukset olivat muotoiltuja tavoitteiksi ja käytännön toteutus oli vastuullani ketterien menetelmien ja erityisesti Scrum -menetelmän mukaisesti. Käytännön toteutus tapahtui hyvin itseohjautuvasti, kuten myös ongelmien ratkaisu.

Ohjelmiston versio 1.0 valmistui joulukuussa 2008. On kuitenkin syytä huomioida, että ohjelmistotuotteen kehittäminen on jatkuva prosessi, jonka välietappeina ovat versiot. Versioita parannellaan saadun palautteen perusteella ja tätä kautta muodostuu ohjelmistotuote Documenta Oy:llä.

LÄHTEET

Agile Alliance. 2002. http://www.agilemanifesto.org (Luettu 12.8.2008).

Beck, K. Andres, C. 2006. Extreme Programming Explained. Addison-Wesley.

Cohn, Mike. 2005. Agile Estimating and Planning. New Jersey, USA: Prentice Hall.

Abrahamsson, P. Salo, O. Ronkainen, J. Warsta J. 2002. Agile software development methods. VTT. Web dokumentti. http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf (Luettu 13.8.2008).

Control Chaos. 2008. Web sivusto. http://www.controlchaos.com/about/xp.php (Luettu 13.10.2008).

Huttunen, Janne. 2006. Ketterän ohjelmistokehitysmenetelmän määrittely, vertailu ja käyttäjätutkimus. Diplomityö. http://lib.tkk.fi/Dipl/2007/urn007665.pdf (Luettu 14.7.2008).

Jeffries, R. Anderson, A. Hendrickson, C. 2006. Extreme Programming Installed. Addison-Wesley.

Lindberg, Harri. 2003. Extreme Programming. Pro gradu. http://www.cs.uta.fi/research/thesis/masters/Lindberg_Harri.pdf(Luettu 14.7.2008).

Ketterät Käytännöt. 2008. http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/Scrum/ (Luettu 24.6.2008).

Koskela, Lasse. Scrum: Ketterien menetelmien markkinajohtaja. Web dokumentti. http://ttlry-fi-bin.directo.fi/@Bin/4b657f56a9018dc3ff7784e863e9a2b2/1214319459/application/pdf/11062393/04_ScrumMarketLeaderOfAgileMethods_handout_LasseKoskela.pdf (Luettu 24.6.2008).

McConnell, S. 2002. Ohjelmistotuotannon hallinta. Helsinki: Edita.

Reaktor Innovations. Web sivusto. http://www.ri.fi (Luettu 11.7.2008).

Roine, J. 2007. Office 2007 System. Helsinki: Readme.fi.

Schwaber, K. & Beedle, M. 2002. Agile Software Development with Scrum. New Jersey, USA: Prentice Hall.

Schwaber, K. 2007. The Enterprise and Scrum. Washington, USA: Microsoft Press.

Schwaber, K. 2003. Agile Project Management with Scrum. Washington, USA: Microsoft Press.

SliQTools. 2008. Web sivusto. http://www.software-development-resource.com (Luettu 7.9.2008).

Sterling, D. 2007. Microsoft Office SharePoint Server 2007: The Complete Reference. USA: McGraw-Hill Companies.