Matti Kinnunen

KETTERIEN MENETELMIEN VERTAILU JA ANALYYSI UML/UP-VIITEKEHYKSEN AVULLA

Tietojärjestelmätieteen
pro gradu -tutkielma
11.10.2007

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Kinnunen, Matti Johannes
Ketterien menetelmien vertailu ja analyysi UML/UP-viitekehyksen avulla / Matti Kinnunen
Jyväskylä, Jyväskylän yliopisto, 2007.
114 s.
pro gradu -tutkielma

Tässä tutkimuksessa tarkastellaan ketteriä menetelmiä, niiden prosesseja ja niissä suoritettavaa mallintamista. Tavoitteena on selvittää UP-prosessimallia ja UML-kieltä viitekehyksenä käyttäen, millaisia prosessimalleja ja mallintamistekniikoita menetelmissä käytetään ja miten menetelmät eroavat näiden suhteen toisistaan. Mallintamistekniikoiden tarkastelussa pyritään selvittämään, mikä on UML:n merkitys mallintamistekniikkana suhteessa muihin käytettäviin tekniikoihin. Tarkasteltavina menetelminä ovat XP, FDD ja DSDM.

Tutkimuksessa on kuvattu kirjallisuuteen pohjautuen menetelmien arvot, periaatteet ja käytänteet, prosessit ja käytettävät mallintamistekniikat. Vertailun tuloksista ilmenee, että menetelmien vaiheet ovat yleisellä tasolla toistensa kaltaisia sisältäen kertaluonteiset kartoitus- ja määrittelyosuudet, joita seuraa iteratiivinen toteutusosuus. Käytettävien mallintamistekniikoiden määrä on pieni. Vaatimusten määrittämisessä suositaan luonnollisella kielellä tuotettuja kuvauksia. UML-kaavioista on käytetty luokkakaaviota, sekvenssikaaviota ja käyttötapauskaaviota.

AVAINSANAT: ketterä menetelmä, Agile Manifesti, Extreme Programming, Feature Driven Development, Dynamic Systems Development Method, UP, UML

SISÄLLYSLUETTELO

TAULUKOT

KUVIOT

1 JOHDANTO

Vuosituhannen vaihteessa perinteisten ohjelmistokehitysmallien ja -menetelmien vastapainoksi esitettiin uudenlaisia ajatuksia. Näiden ajatusten pohjalta muodostui Agile Manifesti (Agile Alliance 2001), jonka keskeisinä osina ovat neljä arvoa ja kaksitoista periaatetta. Se toimii pohjana niin sanotuille ketterille menetelmille (agile method). Fowlerin (2003) mukaan *ketterän menetelmän* mukaiselle toiminnalle on ominaista kevyt dokumentaatio, suuri muutosnopeus ja ihmiskeskeinen toimintatapa. Ketterien menetelmien tarkoituksena on kehittää järjestelmiä asiakaslähtöisesti hyödyntäen lyhyitä toteutussyklejä.

Ketterien menetelmien joukko on varsin kirjava, esim. Extreme Programming (XP) (Beck 1999), Scrum (Advanced Development Methods, Inc. 2005), Crystal-menetelmät (Cockburn 2005), Feature Driven Development (FDD) (Nebulon Pty. Ltd. 2004), DSDM (DSDM Consortium 2007), Adaptive Software Development (ASD) (Highsmith 2000) Agile Unified Process (AUP) (Ambler 2007b), Agile Modeling (AM) (Ambler 2007a), Essential Unified Process (EssUP) (Jacobson, Ng & Spence 2005) ja Microsoft Solutions Framework (MSF) (Microsoft Corporation 2007). Harvat menetelmistä ovat vielä saavuttaneet merkittävää asemaa ohjelmistokehityksessä. Tämä johtuu pitkälti siitä, että menetelmät ovat suhteellisen uusia eikä niiden käytöstä ole saatu riittävästi käytännön kokemusta. Vaikka menetelmät noudattavat pääosin Agile Manifestin arvoja ja periaatteita, on niistä kullakin erilaiset ominaispiirteensä ja painotuksensa.

Mallintaminen ketterissä menetelmissä on yleisesti vähäistä. Jos mallintamista tehdään, se tapahtuu useimmiten jollain epäformaalilla kuvaustavalla hyödyntäen seinätaulua tai vastaavaa apuvälinettä. Mutta myös formaalimpia kuvaustapoja käytetään, erityisesti UML-kaavioita. UML (Unified Modeling

Language) on yleisesti tunnettu mallintamiskieli, joka mahdollistaa monipuolisen kohdealueen ja järjestelmän kuvaamisen lukuisten kaavioiden avulla (Booch, Rumbaugh & Jacobson 1999). UML on yleiskäyttöinen visuaalinen mallinnuskieli, jota käytetään ohjelmistojen osien määrittämisessä, havainnollistamisessa, rakentamisessa ja dokumentoinnissa (Rumbaugh, Jacobson & Booch 1999, 3). Mutta miltä osin UML:ää hyödynnetään ketterissä menetelmissä, missä tarkoituksessa ja missä vaiheessa, siitä kirjallisuudessa ei ole kovin selkeää näkemystä. UML:ää pidetään usein raskaana ja monimutkaisena mallinnuskielenä (Dori 2002; Kobryn 2002), minkä voidaan olettaa vähentävän sen käyttökelpoisuutta ketterien menetelmien yhteydessä. Niiden tarkoituksenahan on välttää laajaa dokumentointia ja tavoitella suurta muutosnopeutta. Kuitenkin UML:llä on niin vahva asema mallinnuskielenä, ettei sen käyttöä voida helposti välttää. UML-kaavioiden joukosta voidaan kuitenkin pyrkiä etsimään ne kaaviot, jotka nähdään ketterien menetelmien kannalta hyödyllisiksi. Erickson ja Siau (2004; 2007) pyrkivät tutkimuksissaan määrittämään UML:n ydinjoukon. Tällaisen ydinjoukon löytäminen edesauttaa keskittymään olennaisiin asioihin, mikä puolestaan näkyy UML:n oppimiseen ja sillä mallintamiseen käytettävien resurssien säästönä. Kobryn (2002, 108) esittää, että tähänkin tilanteeseen voidaan soveltaa niin sanottua 80/20-sääntöä, jonka mukaan 20 prosentilla kaavioista voidaan mallintaa 80 prosenttia erilaisista ongelmista. Koska ketterät menetelmät korostavat säännöllistä vuorovaikutusta asiakkaan kanssa, tulee viestinnän ja apuvälineiden käytön olla tehokasta, mutta riittävän yksinkertaista, jotta ymmärrettävyys säilyy. Näin ollen vuorovaikutuksessa tulisi minimoida erilaisten kaavioiden käyttö keskittymällä vain olennaiseen.

Ketterien menetelmien mukaiselle toiminnalle ominaisia piirteitä ovat prosessin joustavuus, iteratiivisuus ja inkrementaalisuus (Nerur, Mahapatra & Mangalaraj 2005, 75). UP-prosessimalli (Jacobson, Booch & Rumbaugh 1999) jäsentää viitekehyksen tavoin tietojärjestelmän kehittämistehtävät vaiheiksi,

työnkuluiksi ja iteraatioiksi. UP on lähtökohdiltaan käyttötapauspainotteinen, arkkitehtuurikeskeinen sekä iteratiivinen ja inkrementaalinen (Jacobson, Booch & Rumbaugh 1999, 4). Mitkä ovat ne ensisijaiset toiminnalliset rakenteet, joiden mukaan prosessit ketterissä menetelmissä on määritelty? Miten kiinteinä prosessirakenteet on esitetty? Nämä ovat esimerkkejä kysymyksistä, joita voidaan esittää tarkasteltaessa ketteriä menetelmiä toiminnallisesta näkökulmasta. Ketterät menetelmät pyrkivät tukemaan tietojärjestelmän kehittämistä vaatimusmäärityksestä toteutukseen. Vaikka menetelmien ”keveys” voi tarkoittaa joidenkin tehtävien poisjättöä ja toisten soveltamista yleisellä tasolla, niiden mukaisessa kehittämistyössä mainitaan oletettavasti kuitenkin ydinosat tyypillisistä prosesseista.

Tämän tutkimuksen tavoitteena on vertailla ja analysoida ketteriä menetelmiä käyttäen viitekehyksenä UML-kieltä ja UP-prosessimallia. Työn tutkimusongelmana on: Missä määrin ja miltä osin ketterissä menetelmissä esiintyy UML:n ja UP:n piirteitä ja miten menetelmät eroavat näiden suhteen toisistaan? Tutkimusongelma voidaan jäsentää seuraavilla tutkimuskysymyksillä:

- Millaisia prosessimalleja ketterissä menetelmissä käytetään ja miten ne vastaavat UP-prosessia?
- Mikä on mallintamisen merkitys ketterissä menetelmissä ja millaisia mallintamistekniikoita käytetään ja miten niiden käyttöä on ohjeistettu?
- Miten ketterissä menetelmissä on ohjeistettu UML-kaavioiden käyttöä ja mikä on niiden merkitys mallintamistekniikkana?
- Miten ketterien menetelmien prosessimallit ja mallintamistekniikat eroavat toisistaan ja miten näitä eroja voidaan selittää?

Tunnettuja ketteriä menetelmiä on useita (Abrahamsson ym. 2002). Tässä tutkimuksessa rajaudutaan tarkastelemaan kolmea ketterää menetelmää: Extreme Programming (XP) (Beck 1999), Feature Driven Development (FDD) (Nebulon Pty. Ltd. 2004) ja Dynamic Systems Development Method (DSDM) (DSDM Consortium 2007). Tärkeimpänä valintakriteerinä on ollut se, että menetelmässä on määritelty prosessi ja mallintamistekniikat sillä tasolla, että niitä voidaan hyödyntää analyysissä. Myös menetelmien tunnettavuudella on merkitystä valintaan sekä sillä kuinka vakiintuneita ne ovat. Menetelmien erilaisten lähtökohtien toivotaan lisäksi tuovan mielenkiintoa tuloksiin ja niiden analysointiin. XP:n valintaa puoltaa se, että se on ehdottomasti tunnetuin ketterä menetelmä. Sen mallintamista ja prosessia koskevat analyysitulokset kiinnostavat siksi laajasti, ja sitä koskevan lähdemateriaalin löytäminen on suhteellisen helppoa. FDD on yleisimpiä ketteriä menetelmiä, ja sen tiedetään saaneen vaikutteita UML:stä. FDD ei ole lähtökohdiltaan yhtä joustava menetelmä kuin XP, joten se tuo vertailupohjaa ketteristä menetelmistä, jotka ovat lähempänä perinteisiä menetelmiä. DSDM on viitekehys, joka kattaa koko ohjelmistoprosessin elinkaaren ja poikkeaa näin ollen edellä mainituista menetelmistä.

Menetelmien tarkastelu pyritään rajaamaan koskemaan prosesseissa kuvattuja vaiheita ja tehtäviä jättäen muun muassa projektin hallinta tarkastelun ulkopuolelle. Menetelmissä määriteltyjä rooleja ja vastuita tarkastellaan lähinnä vain mallintamisen organisoinnin näkökulmasta.

Tutkimus on lähestymistavaltaan käsitteellisteoreettinen. Tarkoituksena on kartoittaa kirjallisuuden avulla, miten menetelmien prosessit on määritelty, mikä on mallintamisen merkitys ja mitä mallintamistekniikoita käytetään. Tähän tutkimukseen ei kuulu empiiristä osuutta. Pyrkimyksenä on kuitenkin löytää myös empiriaan pohjautuvia tutkimuksia, jotka tarkastelevat aihetta

enemmän käytännön näkökulmasta ja verrata näiden käsityksiä tutkimustuloksiin.

Useat ohjelmistoyritykset tulevat lähivuosien aikana soveltamaan ketteriä menetelmiä projekteissa tai ainakin harkitsevat menetelmien käyttöönottoa. Saatuja tuloksia voidaan hyödyntää muun muassa valittaessa organisaatiolle sopivinta menetelmää. Tutkimus auttaa selvittämään, kuinka hyvin tarkasteltavien menetelmien prosessit ja mallintamisen tavat sopisivat nykyiseen organisaatioympäristöön. Tuloksia voidaan hyödyntää myös mallintamiskielen, esimerkiksi UML:n, kehittämisessä. Tuloksista voidaan päätellä, millaisia ominaisuuksia mallinnuskielellä tulisi olla, jotta se tukisi entistä paremmin ketterien menetelmien tarpeita, kuten vähäistä ennakkosuunnittelua ja lyhyitä iteraatioita.

Tämä tutkielma koostuu johdannosta, neljästä sisältöluvusta ja yhteenvedosta. Toisessa luvussa tarkastellaan ketterää lähestymistapaa ja sille ominaisia piirteitä. Kolmannessa luvussa esitellään UML-mallinnuskieli ja UP-prosessimalli. Neljännessä luvussa esitellään ketteristä menetelmistä XP, FDD ja DSDM. Luvussa tarkastellaan menetelmien prosesseja ja menetelmien mukaista mallintamista. Viidennessä luvussa vertaillaan menetelmiä ja analysoidaan tuloksia. Lopuksi kootaan tutkimuksen tulokset yhteenvedossa.

2 KETTERÄ LÄHESTYMISTAPA

Tässä luvussa määritellään, mitä tarkoitetaan ketterällä lähestymistavalla ja esitellään ketterälle lähestymistavalle ominaisia piirteitä. Aluksi tarkastellaan ketterän lähestymistavan arvoja ja periaatteita Agile Manifestin avulla. Seuraavaksi pyritään luonnehtimaan ketterälle lähestymistavalle ominaisia piirteitä ja vertailemaan niitä perinteiseen lähestymistapaan. Tämän jälkeen tarkastellaan ketteriä menetelmiä ja niiden kehitystä ja kehitykseen vaikuttaneita tekijöitä. Lisäksi annetaan yleiskuva siitä, mitkä ovat tällä hetkellä keskeisimmät ja käytetyimmät menetelmät. Lopuksi tarkastellaan ketterien menetelmien käytöstä saatuja kokemuksia ja mielipiteitä niiden käytöstä sekä soveltuvuudesta.

2.1 Agile Manifesti

Vuonna 2001 joukko ketterien menetelmien avainhenkilöitä kokoontui keskustelemaan ohjelmistotuotannosta. Tarkoituksena oli luoda yhteistä pohjaa eri menetelmille ja näin edesauttaa agile-ajattelun leviämistä. Konkreettisena tuotoksena syntyi Agile Manifesti (Agile Alliance 2001), joka määrittää ketterille menetelmille tyypillisiä arvoja ja periaatteita. Seuraavana tarkastellaan näitä lyhyesti.

Agile Manifestin arvojen tarkoituksena on luoda yhtenäinen arvomaailma, joita ketterien menetelmien tulisi noudattaa. Manifesti sisältää neljä arvoa, jotka voidaan lyhyesti ilmaista seuraavasti (Agile Alliance 2001):

- ihmiset ja vuorovaikutus ennen prosesseja ja työkaluja
- toimiva ohjelmisto ennen kaikenkattavaa dokumentointia
- asiakasyhteistyö ennen sopimusneuvotteluja
- muutokseen vastaaminen ennen suunnitelmassa pysymistä.

Arvojen lisäksi Agile Manifesti rakentuu kahdelletoista periaatteelle, jotka ilmaisevat arvoja konkreettisemmin Manifestin tavoitteita:

- Tärkein tavoite on tyydyttää asiakasta toimittamalla jo aikaisessa vaiheessa toimivaa ohjelmistoa.
- Muuttuvat vaatimukset ovat tervetulleita, jopa kehityksen loppuvaiheessa. Ketterä-prosessi valjastaa muutoksen asiakkaan kilpailueduksi.
- Ohjelmistosta tulee toimittaa säännöllisesti uusi versio, mielellään lyhyin aikavälein (parista viikosta pariin kuukauteen).
- Päättäjien ja kehittäjien täytyy työskennellä yhdessä päivittäin läpi koko projektin.
- Projektit rakentuvat motivoituneiden ihmisten yhteistyönä. Heille tulee tarjota työympäristö ja heitä tulee tukea. On luotettava, että he saavat työnsä tehtyä.
- Tehokkain ja pätevin tiedon välitysmenetelmä kehittäjätiimissä on kasvokkain keskustelu.
- Toimiva ohjelmisto on edistymisen paras mittari.
- Ketterät prosessit edistävät kestävää kehitystä. Rahoittajien, kehittäjien ja käyttäjien tulisi pyrkiä työskentelemään tasaisella tahdilla.
- Jatkuva huomio tekniseen laatuun ja hyvään suunnitteluun edistää ketteryyttä.
- Yksinkertaisuus – eli ensisijaisten vaatimusten toteuttaminen jättämällä toissijaiset tehtävät tekemättä – on välttämätöntä.
- Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseohjautuvissa tiimeissä.
- Säännöllisin väliajoin tiimi tarkastelee, kuinka tulla tehokkaammaksi ja säätää sen perusteella käyttäytymistään.

Periaatteet heijastavat pehmeämpiä arvoja, kuten ihmisten välistä vuorovaikutusta, mutta niissä näkyy myös liiketoiminnan merkitys. Toimintaa

tulee tarkastella liiketoiminnan näkökulmasta keskittymällä arvon tuottamiseen.

Edellä mainitut arvot ja periaatteet eivät edusta mitään uutta näkemystä. Ne sisältävät ajatuksia, jotka ovat olleet jo vuosia useiden ohjelmistokehittäjien tiedossa. Ketterät menetelmät pyrkivät käytännössä noudattamaan näitä arvoja ja periaatteita, jolloin ne eivät jää pelkästään ajatuksen tasolle. Jokaisella ketterällä menetelmällä on kuitenkin omat piirteensä ja käytänteensä, joilla ne erottuvat toisistaan.

Vaikka ketterien menetelmien käytöstä on ollut positiivisia kokemuksia, kaikki eivät hyväksyä niiden lähestymistapaa. Rakitin (2001) kritisoi Manifestin arvoja siitä, että toissijaiseksi asetetut asiat ovat välttämättömiä, kun taas ensisijaiseksi asetetut asiat ovat tekosyitä hakkereille tuottaa koodia ilman ohjelmistotekniikan tarjoamaa kurinalaisuutta. Esimerkiksi toimivan ohjelmiston korostaminen voi aiheuttaa dokumentoinnin laiminlyömistä.

2.2 Ketterä lähestymistapa vs. perinteinen lähestymistapa

Seuraavaksi määritellään, mitä tarkoitetaan ketterällä lähestymistavalla ja vertaillaan ketterän lähestymistavan ja perinteisen lähestymistavan piirteitä keskenään. Tarkoituksena on määrittää, millaisiin tilanteisiin ketterä lähestymistapa parhaiten soveltuu. Tässä yhteydessä esitetään ideaali prosessimalli, joka tukee ketterää lähestymistapaa. Lopuksi vertaillaan sitä, kuinka ketteriä nykyiset ketterät menetelmät itse asiassa ovat.

Sanalle ketterä (agile) on tunnusomaista nopeus, keveys, liikutettavuus ja vikkelyys (Houghton Mifflin Company 2000). Qumer ja Henderson-Sellers (2007) määrittelevät ketteryyden keskeisiksi piirteiksi joustavuuden, nopeuden, kustannustehokkuuden (lean), oppimisen ja muutosvalmiuden

(responsiveness). Abrahamsson ym. (2002, 17) määrittelevät ketterän lähestymistavan keskeisiksi piirteiksi ohjelmistokehityksen inkrementaalisuuden, yhteistyön, suoraviivaisuuden ja joustavuuden (adaptive). Lindvallin ym. (2002, 201) mukaan ketterät menetelmät ovat iteratiivisia, inkrementaalisia, itseohjautuvia ja nopeasti kehittyviä.

Erityisesti Internet- ja mobiiliympäristöjen yleistymisen myötä nopeampien ja kevyempien ohjelmistotuotantoprosessien tarve on kasvanut (Abrahamsson 2002, 9). Perinteiset menetelmät on nähty riittämättömiksi näiden ympäristöjen tarpeisiin. Nandhakumar ja Avison (1999) kritisoivat perinteisiä tietojärjestelmien kehittämismenetelmiä siitä, että ne nähdään kontrollin harhakuvana tai symbolisena tekijänä. Lisäksi ne ovat liian mekanistisia päivittäiseen käyttöön.

On vaikea määritellä yksiselitteisesti, miten perinteiset ja ketterät menetelmät eroavat toisistaan (Abrahamsson ym. 2002, 8) Taulukkoon 1 on koottu perinteisen ja ketterän lähestymistavan eroja Nerurin Mahapatran ja Mangalarajin (2005, 75) sekä Boehmin ja Turnerin (2003, 33) esitysten pohjalta.

Kuten taulukosta 1 voidaan nähdä, ketterä lähestymistapa korostaa muun muassa ihmiskeskeisyyttä, yhteistyötä ja ihmisten välistä viestintää. Perinteinen lähestymistapa puolestaan alleviivaa järjestelmällisyyttä ja suunnitelmallisuutta. Näkemyserot johtuvat pitkälti eriävistä perusolettamuksista. Perinteisen näkemyksen mukaan järjestelmät ovat täysin määriteltäviä ja ennustettavia, jolloin huolellisen ja laajamittaisen suunnittelun avulla voidaan saavuttaa tavoitteet. Ketterän lähestymistavan perusolettamuksena taas on se, ettei muutoksilta voida välttyä. Näin ollen on pystyttävä reagoimaan muutoksiin nopeasti. Muutosnopeutta voidaan parantaa pienien tiimien avulla soveltamalla jatkuvaa pienimuotoista suunnittelua ja testausta.

TAULUKKO 1. Perinteisen ja ketterän lähestymistavan erot

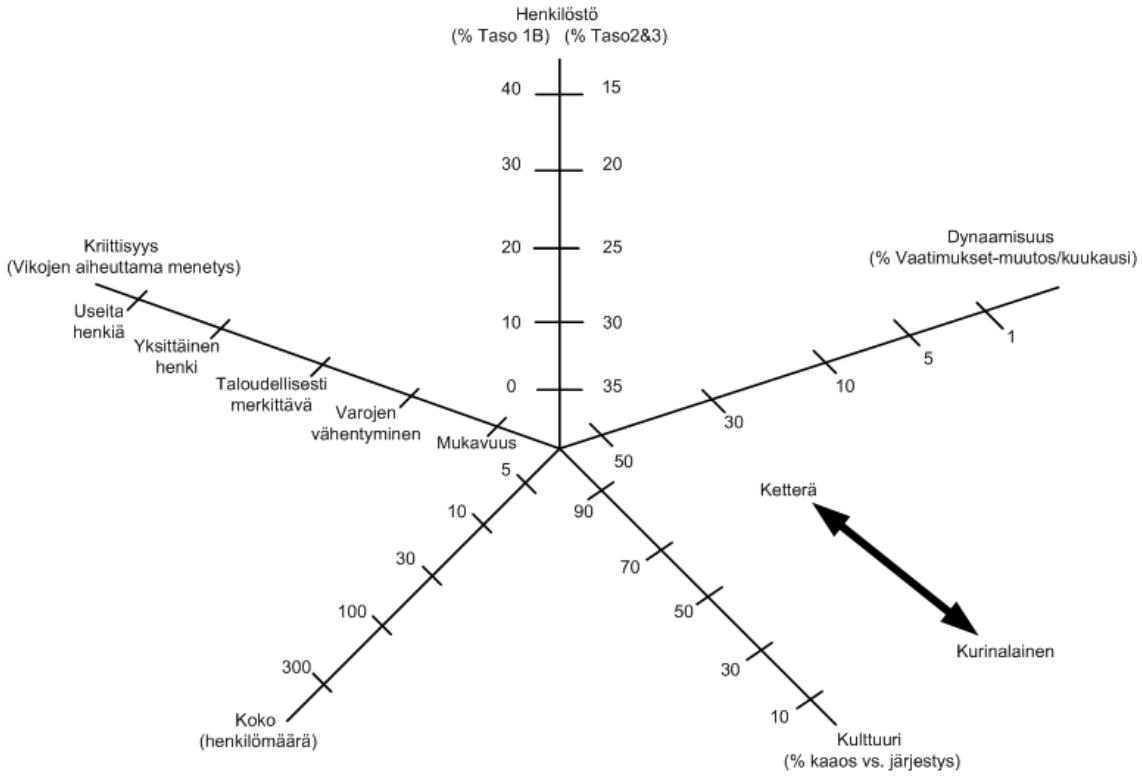
	Perinteinen	Ketterä
Organisaatiokulttuuri	Määriteltyjä käytänteitä ja menettelytapoja noudattava, byrokraattinen ja virallinen toimintamalli	Vapaamuotoinen, joustava ja sosiaaliseen vuorovaikutukseen rohkaiseva toimintamalli
Järjestelmien luonne	Järjestelmät ovat täysin määriteltäviä, vakaita, ennustettavia, ja voidaan rakentaa huolellisen ja laajan suunnittelun avulla.	Jatkuva suunnittelu, toteutus ja testaus mahdollistavat välittömän palautteen saannin, muutokseen vastaamisen ja nopean arvontuottamisen.
Viestintä	Virallinen, dokumentoitu tietämys	Epävirallinen, hiljainen ihmistenvälinen tietämys
Kehittäjät	Tiettyyn työtehtävään erikoistuneet osaajat	Monitaitoiset osaajat, vaihtuvat roolit
Asiakassuhde	Vuorovaikutus tarvittaessa asiakkaan kanssa, keskittyminen sopimusehtoihin	Sitoutuneet, jatkuvassa vuorovaikutuksessa toimivat asiakkaat
Kehitysprosessi	Elinkaarimalli (vesiputous, spiraali tai jokin variaatio); laaja suunnittelu, pitkät kehitysjaksot, refaktorointi oletetaan kalliiksi	Askeleittainen toimitus -malli; yksinkertainen suunnittelu, lyhyet kehityssyklit, refaktorointi oletetaan edulliseksi
Vaatimukset	Määrämuotoinen projekti, laatu, ennakoitavat kehitysvaatimukset	Priorisoidut epämuodolliset tarinat ja testitapaukset, jatkuva ennustamaton muutos

Edellä esitetyn tarkastelun perusteella voidaan todeta, etteivät ketterät menetelmät sovellu kaikkiin tilanteisiin. On tilanteita, joihin ketterä lähestymistapa ei sovellu, ainakaan ilman tiettyjä muunnelmia. Tästä syystä onkin tärkeää tiedostaa, millaisiin tilanteisiin ketterät menetelmät parhaiten soveltuvat ja mitä edellytyksiä niiden toimivalle käytölle on. Boehmin ja Turnerin (2003) mukaan lähestymistavan valinta pitäisi suorittaa

tilannekohtaisesti edellä mainittujen olettamusten perusteella. He nimeävät viisi kriteeriä, joiden avulla pyritään määrittelemään, tulisiko suosia ketterää vai suunnitelmalähtöistä lähestymistapaa. Nämä kriteerit ovat: henkilöstö, dynaamisuus, kulttuuri, koko ja sovelluksen kriittisyys. Henkilöstöllä tarkoitetaan kehittäjien kykyä hyödyntää muun muassa erilaisia tekniikoita ja soveltaa malleja. Erittäin ammattitaitoisten kehittäjien tarpeen määrä vaihtelee projektien mukaan. Määritellessään ammattitaitoisten kehittäjien tarvetta Boehm ja Turner (2003) viittaavat Cockburnin (2001) esittämiin tasoluokituksiin. Luokituksen mukaan henkilöt voidaan jakaa viiteen tasoon seuraavasti: -1, 1B, 1A, 2, ja 3. Luokitus kertoo sen, kuinka hyvin kyseinen henkilö kykenee omaksumaan ja soveltamaan käytettävää menetelmää. Esimerkiksi -1-tason henkilö ei kykene järjestelmällisesti noudattamaan mitään menetelmää, kun taas toiseen ääripäähän luokiteltavat tasojen 2 ja 3 henkilöt eivät pelkästään pysty noudattamaan tiettyä menetelmää, vaan myös soveltamaan sitä uusissa tilanteissa. Dynaamisuudella tarkoitetaan sitä, kuinka usein vaatimukset muuttuvat projektin aikana. Kulttuuri kuvaa sitä, suositaanko organisaatiossa järjestystä vai onko toimintaympäristö lähempänä kaaosta. Koko-kriteerillä tarkoitetaan projektin henkilöstömäärää. Viimeisenä tekijä on kriittisyys, joka määrittelee sen, minkä asteisia seurauksia ohjelmiston virhetoiminnasta voi aiheutua. Kuviossa 1 on esitetty asteikko edellä mainittujen kriteereiden pohjalta, jonka avulla voidaan määritellä, onko käytetty lähestymistapa ketterä vai ei.

Mitä lähempänä organisaation tunnusmerkit ovat kuvion keskustaa, sitä todennäköisemmin ketterän lähestymistavan mukainen toiminta olisi sovellettavissa. Kriteereistä koko, dynaamisuus ja kriittisyys ovat tekijöitä, jotka vaikuttavat usein lähestymistavan valintaan. Pieni kehitysryhmä, nopeasti muuttuvat vaatimukset ja ei-kriittisen järjestelmän kehittäminen ovat piirteitä, joita voidaan pitää selkeästi ketterän lähestymistavan valintaa suosivina. Sen sijaan henkilöstö-kriteerin merkitys ei ole niin selkeä. Boehmin ja Turnerin

(2003) mukaan ketterää lähestymistapaa noudattavassa projektissa tulisi olla noin kolmannes korkeatasoisia kehittäjiä (tasot 2 ja 3), kun puolestaan perinteisessä, suunnitelmahakuisessa lähestymistavassa heidän osuutensa tulee alussa olla suurempi, mutta myöhemmässä vaiheessa tarve on merkittävästi pienempi. Joka tapauksessa ketterän lähestymistavan uskotaan toimivan selkeästi sitä paremmin, mitä vähemmän alhaisemman tason kehittäjiä työskentelee projektissa. Organisaatiossa vallitsevalla kulttuurilla on vaikutusta ketterän lähestymistavan käyttöönottoon. Jos organisaatiossa on käytössä runsaasti erilaisia sääntöjä ja standardeja, voi vapaamuotoisemman lähestymistavan käyttöönotto aiheuttaa erilaisia ristiriitoja.



KUVIO 1. Lähestymistavan valintaan vaikuttavat ulottuvuudet (Boehm & Turner 2003, 34)

Visconti ja Cook (2004) ovat muodostaneet ideaalin prosessimallin ketterille menetelmille, joka tukee Manifestin periaatteita. Taulukossa 2 on esitetty

kyseinen malli, joka sisältää neljä vaihetta: tunnistus-, tarkkailu-, mittaus- ja palautevaihe ja koostuu kahdesta ulottuvuudesta: ydinprosessista ja asiakastyytyväisyydestä. Kunkin vaiheen kohdalla on lueteltu ne Manifestin periaatteet, jotka nähdään keskeisiksi ydinprosessin ja asiakastyytyväisyyden kannalta. Tunnistusvaiheen tavoitteena on määrittää prosessin keskeisimmät käytänteet laadukkaan ja asiakasvaatimukset tyydyttävän tuotteen toimittamiseksi. Tarkkailuvaiheessa tarkkaillaan prosessin etenemistä, laatua ja asiakastyytyväisyyttä. Mittausvaiheessa määritetään kerättävä tieto, kerätään tietoa ja analysoidaan kerättyä tietoa. Palautevaiheessa luodaan, arvioidaan, priorisoidaan ja yhdistetään suosituksia prosessin kehittämiseksi.

Viscontin ja Cookin (2004) mukaan prosessimallin avulla voidaan määrittää, kuinka ketterä kukin menetelmä on. Heidän mukaan sen avulla voidaan lisäksi arvioida ketterien menetelmien vahvuuksia ja heikkouksia, jonka myötä voitaisiin kehittää menetelmiä lisäämällä niihin uusia käytänteitä.

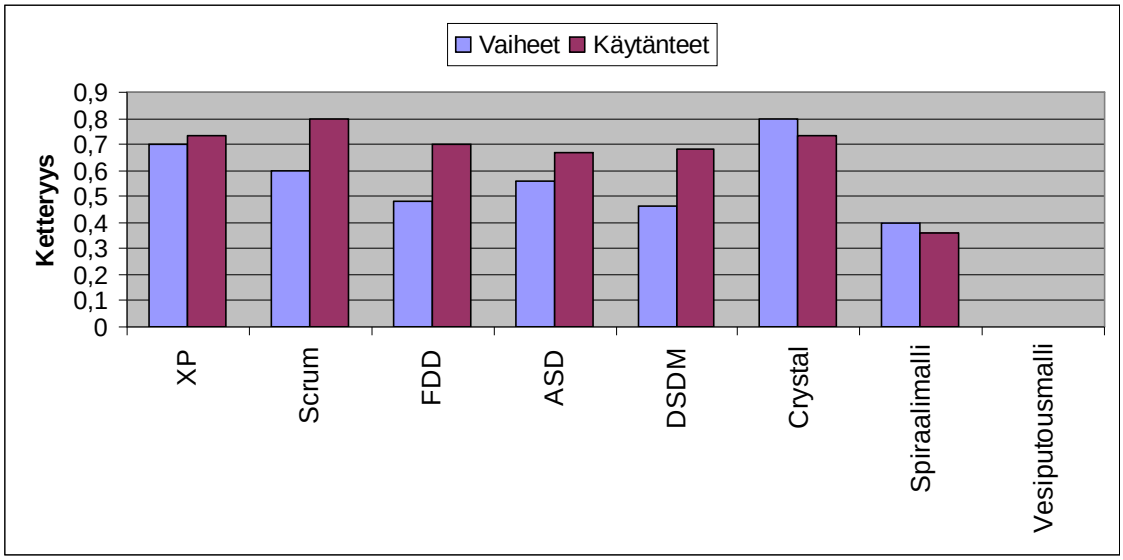
Qumer ja Henderson-Sellers (2007) ovat myös tarkastelleet ketteriä menetelmiä ja pyrkineet määrittelemään, mitkä ovat ketteriä menetelmiä ja mitkä eivät. He ovat tarkastelleet kuutta ketterää menetelmää 4-DAT-viitekehyksen avulla. Viitekehys on nelijakoinen. Ensimmäisessä kohdassa tarkastellaan menetelmän laajuutta (method scope), josta saatuja tuloksia voidaan käyttää menetelmien vertailuun yleisellä tasolla. Toinen kohta tarkastelee sitä, missä määrin menetelmän vaiheet ja käytänteet tukevat heidän määrittelemiä viittä ketteryyden piirrettä: joustavuutta, nopeudetta, kustannustehokkuutta, oppimista ja muutosvalmiutta. Kolmannessa kohdassa tarkastellaan sitä, kuinka menetelmän käytänteet tukevat ketterän lähestymistavan mukaisia arvoja. Viimeisessä kohdassa tarkastellaan sitä, miten hyvin menetelmän käytänteet tukevat ohjelmistokehityksen prosesseja. Näistä neljästä kohdasta ainoastaan kohta 2 on kvantitatiivinen, muiden ollessa kvalitatiivisia.

TAULUKKO 2. Ideaali prosessimalli ketterille menetelmille (Visconti & Cook 2004, 435)

Vaihe	Ulottuvuus	
	Ydinprosessi	Asiakastyytyväisyys
Tunnistusvaihe	• Toimita säännöllisesti toimivaa ohjelmistoa • Jatkuva vuorovaikutus asiakkaiden ja kehittäjien välillä • Maksimoi tekemättä jätetty työ • Kasvokkain kommunikointi kehitystiimissä • Jatkuva kehitystahti • Itseorganisoituvat tiimit	• Tyydytä asiakastarpeet • Toivota muutokset tervetulleiksi • Jatkuva huomio tekniseen erinomaisuuteen ja hyvään suunnitteluun • Sopivan ympäristön ja tuen tarjoaminen
Tarkkailuvaihe	• Varhainen ja jatkuva toimivan ohjelmiston toimittaminen • Liiketoiminnasta vastaavien ja kehittäjien päivittäinen yhteistyö • Luota siihen, että kehittäjät saavat työn tehtyä • Sopivan ympäristön ja tuen tarjoaminen	• Varhainen ja jatkuva toimivan ohjelmiston toimittaminen • Liiketoiminnasta vastaavien ja kehittäjien päivittäinen yhteistyö
Mittausvaihe	• Toimiva ohjelmisto on edistyksen tärkein mittari	• Toimiva ohjelmisto on edistyksen tärkein mittari
Palautevaihe	• Säännöllisin väliajoin tiimi tarkastelee, kuinka tulla tehokkaammaksi ja säätää sen perusteella käyttäytymistään	• Tyydytä asiakastarpeet • Säännöllisin väliajoin tiimi tarkastelee, kuinka tulla tehokkaammaksi ja säätää sen perusteella käyttäytymistään

Seuraavaksi keskitytään tarkastelemaan Qumerin ja Henderson-Sellersin (2007) tutkimuksessa kvantitatiivisesti esitettyjä tuloksia. Tutkimuksessa tarkasteltavat ketterät menetelmät olivat XP, Scrum, FDD, ASD, DSDM ja Crystal-

menetelmät. Lisäksi vertailun vuoksi oli valittu kaksi perinteistä lähestymistapaa: vesiputousmalli ja spiraalimalli. Menetelmien vaiheet ja käytänteet oli pisteytetty sen mukaan, tukivatko ne ketteryyden piirteitä vai eivät. Kuviossa 2 on esitetty saadut tulokset. Mitä suuremman arvon menetelmä on saanut, sitä ketterämpi se on.



KUVIO 2. Menetelmien ketteryyden taso (Qumer & Henderson-Sellers 2007, 14)

Tuloksista voidaan havaita, että ketterissä menetelmissä on keskenään selkeitä eroja ketteryyden suhteen. Kuitenkin erityisesti käytänteitä vertailtaessa alhaisimmatkin arvot saanut ketterä menetelmä voidaan luokitella selkeästi ketterämmäksi kuin vertailussa ollut spiraalimalli. Vesiputousmalli on jäänyt kokonaan ilman pisteitä. Tuloksia voidaan pitää suuntaa-antavina, joskin jotkut niiden muodostamiseksi käytetyt perusolettamukset ovat tulkinnanvaraisia. Esimerkiksi prosessien vaiheita tarkasteltaessa on käytetty toisista lähteistä poikkeavaa vaiheiden lukumäärää XP:ssä (Abrahamsson ym. 2002, 19) ja DSDM:ssä (Stapleton 1997, 3). Lisäksi XP:tä tarkasteltaessa on käytetty alkuperäisiä (Beck 1999) eikä päivitetty (Beck & Andres 2004) käytänteitä.

Nämä tekijät luonnollisesti vaikuttavat jonkin verran tuloksiin ja näin ollen myös menetelmien keskinäiseen järjestykseen.

2.3 Yleiskatsaus ketteriin menetelmiin

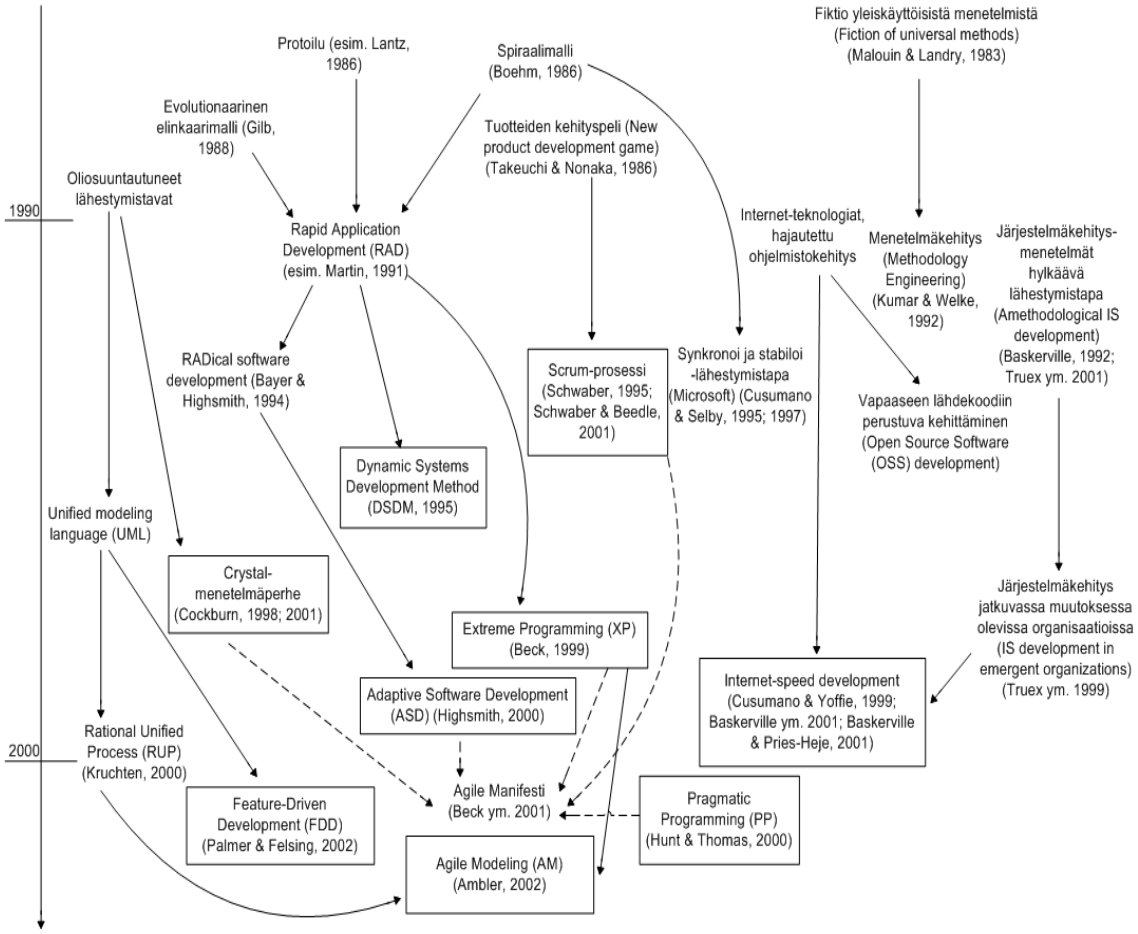
Tunnetuimpana esimerkkinä ketterästä menetelmästä on Extreme Programming (XP) (Beck 1999). Muita ketteriä menetelmiä ovat muun muassa Scrum (Advanced Development Methods, Inc. 2005), Crystal-menetelmät (Cockburn 2005), Feature Driven Development (FDD) (Nebulon Pty. Ltd. 2004), DSDM (DSDM Consortium 2007) ja Adaptive Software Development (ASD) (Highsmith 2000). Seuraavaksi esitellään ketterien menetelmien historiaa ja kehittymistä, joiden avulla pyritään selvittämään keskeisimpiä menetelmien välisiä eroja. Lopuksi tarkastellaan sitä, mitkä ovat tällä hetkellä käytetyimmät menetelmät.

2.3.1 Tausta ja evoluutio

Vaikka ketterät menetelmät nousivat pinnalle vasta vuosituhannen vaihteessa, voidaan niiden syntyyn vaikuttaneita suuntauksia tunnistaa jo 1980-luvun puolivälistä. Ohjelmistotuotannon kehittyessä syntyneet uudet ajatukset ja mallit, kuten spiraalimalli, prototyyppiajattelu ja oliopohjainen lähestymistapa, loivat pohjaa ketterien menetelmien synnylle. Kuvio 3 valaisee tarkemmin ketterien menetelmien kehitystä.

Kuviosta voidaan nähdä, kuinka erilaiset ohjelmistotuotannon elinkaarimallit, kehittämismenetelmät ja oliolähestymistapa ovat tarjonneet lähtökohdan ketterien menetelmien synnylle. Esimerkiksi oliolähestymistapa ja UML ovat vahvasti FDD:n (Coad ym. 1999) ja AM:n (Ambler 2007a) kehityksen taustalla. Vastapainona suunnitelmalähtöiselle suuntaukselle ”ametodinen” lähestymistapa on muokannut ISD:n (Internet-Speed Development) (Baskerville

ym. 2001) kehitystä. Eräät ohjelmistotuotantoprosessit ovat vaikuttaneet useampiin myöhemmin kehittyneisiin menetelmiin. Esimerkkinä tällaisesta prosessista on Rapid Application Development eli RAD (Blue Ink 2006). Se korostaa muun muassa prototyyppien kehittämistä ja iteratiivista lähestymistapaa hyödyntäen voimakkaasti case-työkaluja. Sen voidaan nähdä olevan DSDM:n (DSDM Consortium 2007), XP:n (Beck 1999) ja ASD (Highsmith 2000) kehityksen taustalla. Kuviossa on esitetty suurin osa tunnetuimmista ketteristä menetelmistä, mutta menetelmien kirjo on kasvanut ja tulee jatkossa kasvamaan.



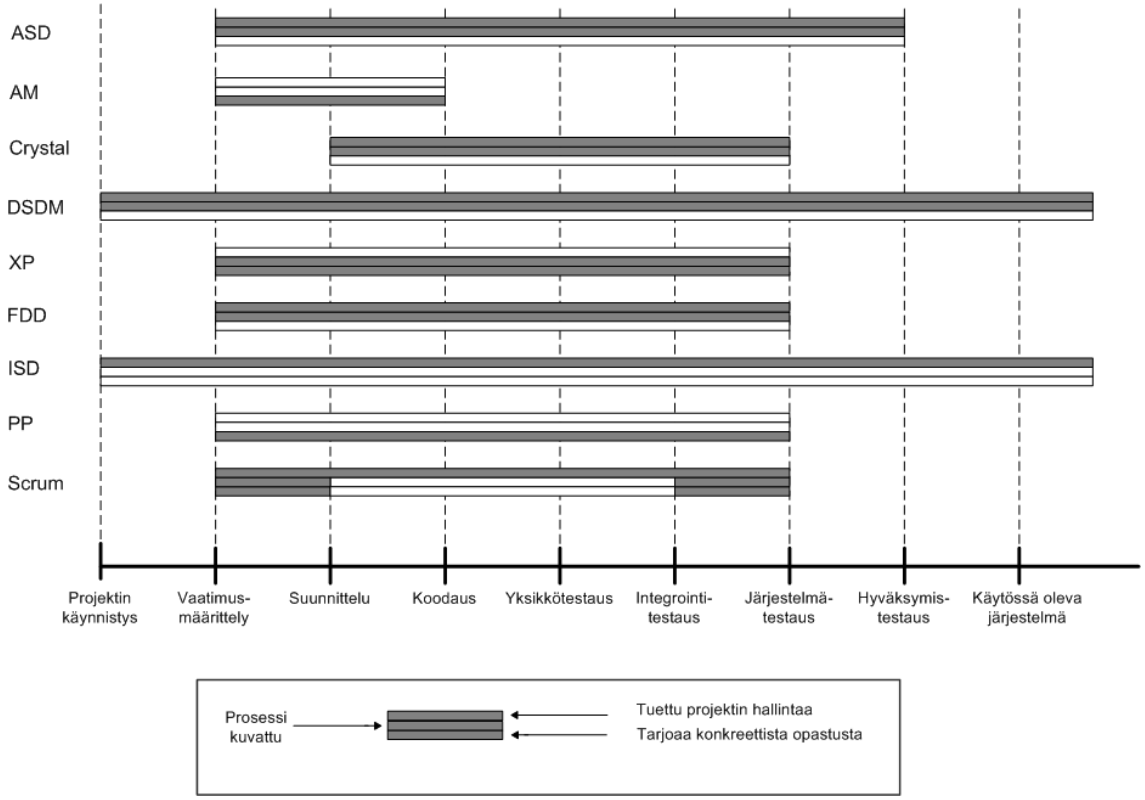
KUVIO 3. Ketterien menetelmien kehityslinjat (Abrahamsson ym. 2003, 246)

Jokaisella ketterällä menetelmällä on omat vaikutteensa, mikä selittää osaltaan myös niiden välisiä eroavuuksia. Kuviosta voidaan havaita, etteivät kaikki ketterät menetelmät ole vaikuttaneet suoraan Agile Manifestin syntyyn. Kuitenkin niitä ja uusia ketteriä menetelmiä tullaan vertailemaan toisiin menetelmiin Manifestin arvojen ja periaatteiden perusteella.

Erilaisten taustojensa vuoksi kukin ketterä menetelmä on keskittynyt painottamaan tiettyjä periaatteita ja käytänteitä. Tämä on johtanut siihen, että joku menetelmä keskittyy paremmin esimerkiksi projektinhallintaan, kun puolestaan toinen korostaa enemmän ohjelmiston toteutukseen liittyviä käytänteitä. Abrahamsson ym. (2003) esittävät kuviossa 4 näkemyksensä ketterien menetelmien erilaisuudesta. He vertailevat ketterien menetelmien tarjoamaa tukea prosessin, projektinhallinnan ja konkreettisen ohjeistuksen osalta suhteessa ohjelmistotuotannon elinkaaren vaiheisiin. Vaiheet on esitetty kuviossa siten, että kukin vaihe alkaa sen nimen osoittamasta virstanpylväästä päättyen sen oikeanpuoleiseen virstanpylvääseen. Ylin palkki kuvaa, kuinka hyvin menetelmä tukee projektinhallintaa. Keskimmäinen palkki kertoo sen, mitä ohjelmistotuotannon elinkaaren vaiheita prosessi kattaa. Alin palkki kuvaa sen, millä tarkkuudella menetelmä tarjoaa konkreettista ohjeistusta. Palkin harmaa väri osoittaa, että menetelmä kattaa tarkasteltavan kriteerin kyseisessä elinkaaren vaiheessa, kun valkoinen väri puolestaan osoittaa tuen puuttumisen.

Kuten kuviosta voidaan havaita, täysin kattavaa ketterää menetelmää tarkastelun kriteereillä ei ole, vaan jokainen menetelmä on keskittynyt tarjoamaan tietynlaista tukea tiettyihin ohjelmistokehityksen elinkaaren vaiheisiin. Jotkut menetelmät, kuten DSDM ja ISD pyrkivät kattamaan ohjelmistotuotannon elinkaaren kokonaan. Molemmat menetelmät kattavat projektinhallinnan ja DSDM tarjoaa kuvatun prosessin, muttei kumpikaan tarjoa konkreettista opastusta. Toiset menetelmät puolestaan keskittyvät enemmän konkreettisen opastuksen tarjoamiseen, kuten AM, XP ja PP. AM ja

PP jättävät kuitenkin muut osa-alueet huomioimatta, minkä lisäksi AM kattaa ohjelmistotuotannon vaiheista ainoastaan vaatimusmäärittelyn ja suunnittelun.



KUVIO 4. Ketterien menetelmien projektinhallinnan, prosessin ja konkreettisen ohjeistuksen kattavuuden vertailu (Abrahamsson ym. 2003, 248)

2.3.2 Käytetyimmät ketterät menetelmät

Tutkimusten (Ambler 2006; Barnett 2006; Behrens 2006; VersionOne, Inc. 2006) avulla on pyritty selvittämään, mitkä ovat käytetyimmät ketterät menetelmät. Tulokset ovat keskenään hyvinkin ristiriitaisia johtuen erilaisista kysymyksen asetteluista, vastausten tulkinnoista sekä mahdollisista vastaajaryhmien maantieteellisistä eroista. Behrensin (2006) ja VersionOnen (2006) tutkimuksissa todettiin Scrum selkeästi yleisimmäksi menetelmäksi XP:n ollessa toiseksi yleisin. Behrensin (2006) mukaan lähes 60 prosenttiyksikköä vastaajaorganisaatioista käytti pääsääntöisenä menetelmänä Scrumia, kun taas

vastaava lukema VersionOnen (2006) tutkimuksessa oli 40 prosenttiyksikköä. Amblerin (2006) mukaan XP:tä sovellettiin yli puolessa ketteriä menetelmiä soveltavista organisaatioista, kun taas FDD:n ja Scrumin osuus jäi alle kolmannekseen. Amblerin (2006) tutkimuksessa vastaajilla oli mahdollisuus valita useampia käytössä olevia menetelmiä, minkä vuoksi ne eivät ole suoraan verrattavissa muiden tutkimusten tuloksiin. Barnettin (2006) tutkimuksessa XP, Scrum ja FDD olivat erittäin tasavertaisia keskenään. Shine Technologiesin (2003) tutkimuksessa lähes 60 prosenttia vastaajista käytti XP:tä. Kun verrataan uudempia tuloksia Shine Technologiesin (2003) tekemään tutkimukseen, voidaan havaita, ettei XP enää olekaan ainoa merkittävä menetelmä. Scrumin suosion kasvua voidaan selittää esimerkiksi sillä, että sitä on onnistuneesti hyödynnetty muiden menetelmien kanssa, eritoten XP:n (Control Chaos 2007; Vriens 2003). Edellä esitetyn kuvion 4 (Abrahamsson ym. 2003, 248) perusteella voidaan myös havaita, että XP ja Scrum täydentävät hyvin toistensa puutteita menetelmien kattavuudessa. Taulukkoon 3 on tiivistetty edellä esitettyjen tutkimusten tulokset. Taulukossa 3 on esitetty XP:n, Scrumin ja FDD:n lisäksi sarake, joka koostuu muista menetelmistä, joiden yksittäinen käyttöaste on parhaillaankin alle 10 prosenttiyksikköä, kuten DSDM ja Crystal-menetelmät.

TAULUKKO 3. Ketterien menetelmien käyttöaste

	XP	Scrum	FDD	Muut
Behrens (2006)	*23 %	*58 %	*5 %	*14 %
VersionOne (2006)	23 %	40 %	ei määritelty	37 %
Ambler (2006)	**57 %	**28 %	**30 %	**42 %
Barnett (2006)	28 %	20 %	26 %	26 %
Shine Technologies (2003)	59 %	*9 %	*9 %	*22 %

* Arvo on johdettu kuviosta ilman tarkkaa numeraalista arvoa.

** Mahdollisuus valita useita vaihtoehtoja.

2.4 Kokemuksia ketterien menetelmien käytöstä

Ketterät menetelmät ovat viime vuosina herättäneet suurta kiinnostusta. Seuraavaksi esitellään alan asiantuntijoiden paneelikeskusteluun pohjautuvia kokemuksia ja mielipiteitä ketterien menetelmien käytöstä (Lindvall ym. 2002). Keskustelun tarkoituksena oli vertailla menetelmien käytöstä saatuja kokemuksia liittyen olemassa oleviin perusolettamuksiin ja käytänteisiin. Lopuksi on esitetty muita kokemuksia ketterien menetelmien käytöstä.

Projektin koko on yksi tärkeimmistä kriteereistä, jonka avulla voidaan määritellä, voidaanko ketterää lähestymistapaa soveltaa. Lähestymistapaa on laajalti sovellettu korkeintaan kaksitoistahenkisissä tiimeissä, mutta mitä isommaksi tiimin koko kasvaa, sitä vähemmän on kokemusta kyseisen lähestymistavan käytöstä. Useiden asiantuntijoiden mukaan tiimi voi olla ketterä sen koosta huolimatta. Cockburn väittää kuitenkin koon olevan olennainen asia. Tiimin koon kasvaessa kasvoittain viestintä häviää. (Lindvall ym. 2002)

Ketteriä menetelmiä on kritisoitu siitä, etteivät ne sovellu kriittisten järjestelmien kehittämiseen. Kuitenkin testilähtöisen kehittämisen (Test-Driven Development) on myös todettu kasvattavan luotettavuutta. Ketteristä menetelmistä Crystal-menetelmät (Cockburn 2005) ovat ainoita, jotka on suunniteltu skaalautumaan erikokoisille ja kriittisyysluokitukseltaan erilaisille projekteille. Crystal-menetelmät muodostavat menetelmäperheen, jossa menetelmät erotetaan toisistaan värin perusteella: kirkas (clear), keltainen (yellow), oranssi (orange) ja punainen (red). Vaaleammilla väreillä nimetyt prosessit ovat luonteeltaan kevyempiä ja soveltuvat pieniin ja vähemmän kriittisiin projekteihin. Tummemmilla väreillä nimetyt prosessit ovat puolestaan raskaampia ja soveltuvat isoihin ja kriittisiin projekteihin. Käytettävä menetelmä valitaan siis sen perusteella, minkä kokoisessa

projektissa sitä sovelletaan ja kuinka kriittinen projekti on. Edellä mainituista menetelmistä kirkas ja oranssi ovat tunnetuimmat.

Kysymys siitä, edellyttääkö ketterän lähestymistavan soveltaminen pätevää henkilöstöä, jakaa mielipiteitä. Jonkinasteisen kompromissin mukaan päteviä ja kokeneita kehittäjiä tulisi olla 25–33% projektin henkilömäärästä. On myös havaittu, että kokemus järjestelmien kehittämisestä on tärkeämpää kuin aikaisempi kokemus ketterien menetelmien käytöstä. Tiettyjen käytänteiden, kuten pariohjelmoinnin, on todettu vähentävän kehittäjien kokemuksen tarvetta. Ketterän menetelmän käyttöönotto vaatii perinteisiä menetelmiä vähemmän muodollista koulutusta. Pariohjelmoinnin käytön on todettu myös vähentävän koulutuksen tarvetta. (Lindvall ym. 2002)

Kokemuksen perusteella tärkeimpiä menestystekijöitä ovat kulttuuri, ihmiset ja viestintä. Organisaatiokulttuurin tulee olla suotuisa ketterän lähestymistavan mukaiselle toiminnalle. Ihmisten tulee olla päteviä ja heidän päätöksiinsä tulee luottaa. Nopea tiimin jäsenten välinen viestintä on välttämätöntä. Viestintää voidaan tukea sijoittamalla henkilöitä samaan työtilaan ja pariohjelmoinnin käytöllä. (Lindvall ym. 2002)

Dokumentoinnin määrä ketterien menetelmissä jakaa mielipiteitä. Eräiden mielipiteiden mukaan se on välttämätöntä elintärkeän tiedon säilyttämiseksi. Toiset pitävät dokumentaation tuottamista tarpeettomana. Joka tapauksessa tavoitteena on pyrkiä tehokkaaseen viestintään dokumentaatiotarpeen välttämiseksi. (Lindvall ym. 2002)

Pariohjelmoinnin lisäksi toinen laajamittaisesti hyödynnettävä käytänne on ohjelmistokoodin uudelleenrakentaminen (refactoring), jota hyödynnetään erityisesti XP:n yhteydessä. Kaikki eivät kuitenkaan miellä sitä hyvänä asiana. Laajamittainen koodin uudelleenmuokkaus voi vaikuttaa sidosryhmiin. Sen

sijaan jatkuva pienimuotoinen ja paikallinen muokkaus nähdään tarpeellisena. (Lindvall ym. 2002)

Boehm ja Turner (2003) esittävät, että valittu lähestymistapa voi olla sekoitus ketterän ja kurinalaisen lähestymistavan piirteitä, esimerkiksi laajoja ja kriittisiä tuotteita kehitettäessä. Vastaavanlaista lähestymistapaa on sovellettu muun muassa Motorolalla (Drobka ym. 2004). Motorola sovelsi XP:tä neljässä liiketoiminnan kannalta kriittisessä projektissaan. He lähtivät liikkeelle siitä perusolettamuksesta, ettei XP sovellu sellaisenaan joka tilanteeseen. Näin ollen vaikka he toimivat pääosin XP:n periaatteiden mukaan, nähtiin suunnittelun ja dokumentaation osuus edelleen merkittävänä tekijänä luotettavuuden takaamiseksi. Lisäksi sovellettiin yhtiön omaa teknisen katselmoinnin prosessia. Kokonaisuudessaan ketterän lähestymistavan ja laadunvarmistusta tukevien toimenpiteiden yhdistämistä pidettiin toimivana ratkaisuna. XP:tä onnistuneesti soveltaneita suuryrityksiä on muitakin, kuten ABB, Daimler-Chrysler ja Nokia (Lindvall ym. 2004). Kaikissa projekteissa saavutettiin parannuksia yhden tai useamman mittarin mukaan: asiakastyytyvyys, laatu, tuottavuus ja kustannukset. Lisäksi havaittiin muitakin positiivisia vaikutuksia, kuten tiimien moraalin nousua. Kaikki organisaatiot oppivat, että XP:n räätälöinti on välttämätöntä. Suuret organisaatiot noudattavat usein joitain määriteltyjä ohjelmistoprosesseja, mikä saattaa aiheuttaa kaksinkertaisen työmäärän, kun otetaan käyttöön uusia käytänteitä. Tämän välttämiseksi ketterää lähestymistapaa noudattavan tiimin ja sen ympäristön rajapinnat tulee määritellä riittävällä tarkkuudella.

Vaikka ketterien menetelmien käytöstä on julkaistu pääasiassa vain positiivisia kokemuksia, löytyy myös kriittisempiä näkemyksiä. Kuten aiemmin mainittiin, Rakitin (2001) on arvostellut Agile Manifestin arvoja. Suurin osa kritiikistä kohdistuu kuitenkin menetelmien periaatteisiin ja käytänteisiin. Koska XP on tunnetuin ketteristä menetelmistä, sen käytöstä löytyy myös negatiivisia

kokemuksia (Stephens 2005). Kritiikin perusteella kaikkia XP:n käytänteitä ei käytännössä noudateta, koska niiden toimimiseen ei uskota, ja jotkin käytänteet, kuten pariohjelmointi, koetaan epämiellyttävänä.

2.5 Yhteenveto

Tässä luvussa tarkasteltiin ketterää lähestymistapaa. Aluksi esiteltiin Agile Manifesti ja sen mukaiset arvot ja periaatteet. Arvoissa ja periaatteissa heijastuvat ketterän lähestymistavan mukainen ajattelu, jonka mukaan tavoitteena on toimittaa ohjelmistoa kustannustehokkaasti ja ihmislähtöisesti. Seuraavaksi vertailtiin ketterän ja perinteisen lähestymistavan eroja. Ketterä lähestymistapa tarjoaa vaihtoehtoisen lähestymistavan perinteisten ohjelmistokehitysmenetelmien rinnalle. Ketterä lähestymistapa soveltuu lähtökohtaisesti perinteistä lähestymistapaa paremmin henkilömäärältään pienempiin, jatkuvasti muuttuvaan ympäristöön ja vaatimuksiin sekä vähemmän kriittisten järjestelmien rakentamiseen. Ketterälle lähestymistavalle ominaisten piirteiden pohjalta on pyritty mittamaan menetelmien ketteryyttä ja määrittämään ideaaliprosessia.

Ketterien menetelmien kehittymiseen vaikuttaneita suuntauksia voidaan havaita jo 1980-luvulta, kuten protoilu ja spiraalimalli. Suuntaukset ilmenevät menetelmissä eri tavoin ja ovat saattaneet vaikuttaa keskeisesti menetelmän kehittymiseen. Menetelmät painottavat eri tavoin ohjelmistokehityksen eri vaiheita, projektinhallintaa ja konkreettista opastusta käytänteiden avulla. Tällä hetkellä käytetyimmät menetelmät ovat XP ja Scrum.

Kokemusten perusteella keskeisimmät menestystekijät ketterien menetelmien käytössä ovat organisaatiokulttuuri, ihmiset ja viestintä. Monet lähtökohdat ja käytänteet, kuten tiimin koko, dokumentointi ja pariohjelmointi jakavat

mielipiteitä. Ketterät menetelmät ovat joustavia ja niitä voidaan täydentää tarvittaessa, esimerkiksi laadunvarmistusta edistävillä toimenpiteillä.

3 UML JA UP

Tässä luvussa esitellään UML-mallinnuskieli (Booch, Rumbaugh & Jacobson 1999) ja UP-prosessimalli (Jacobson, Booch & Rumbaugh 1999). UML:ää tarkastellaan pääasiassa kaavioiden ja niiden käyttötarkoitusten näkökulmasta. UP:ssa keskitytään taustan, työnkulkujen ja vaiheiden tarkastelemiseen.

3.1 UML

Seuraavaksi esitellään UML-mallinnuskieli. Ensiksi esitellään lyhyesti UML:n käyttötarkoitus ja sen historia. Lisäksi määritellään tutkielman osalta keskeisiä mallintamiseen liittyviä termejä. Sen jälkeen esitellään UML-kaaviot ja niiden käyttötarkoitukset.

3.1.1 Tausta

The Unified Modeling Language (UML) on yleiskäyttöinen visuaalinen mallinnuskieli, jota käytetään tietojärjestelmän osien määrittämisessä, havainnollistamisessa, rakentamisessa ja dokumentoinnissa. Sen avulla voidaan taltioida päätökset ja muodostaa käsitys rakennettavasta järjestelmästä. (Rumbaugh, Jacobson & Booch 1999, 3) UML:ää voidaan käyttää myös liiketoiminnan ja tietojärjestelmien ulkopuolisten osien mallintamiseen (Object Management Group, Inc. 2007). UML on saavuttanut selkeän ykkösaseman vertailtaessa erilaisia ohjelmistokehityksessä käytettäviä mallintamiskieliä. Vuonna 1997 julkaistiin virallisesti UML 1.0 ja vuoteen 2003 mennessä on julkaistu versio 1.5 (Vinciguerra 2004). Nykyään uusin virallinen versio on UML 2.1.1 (Object Management Group, Inc. 2007). Vaikka UML 2.x onkin tuonut uudistuksia, uuden standardin laajamittainen käyttöönotto vie oman aikansa. Tämä näkyy muun muassa nykyisissä UML:n käyttöä käsittelevissä tutkimuksissa, joissa on käytetty UML 1.x:n mukaista standardia. Muutokset ydinosan muodostaviin kaavioihin ovat olleet myös erittäin vähäisiä.

Seuraavaksi määritellään tämän tutkielman tärkeimmät mallintamiseen liittyvät termit, joita ovat UML-kaavio, mallintamistekniikka ja malli. *UML-kaavio* on graafinen esitys, joka koostuu joukosta elementtejä, jotka rakentuvat solmuista (things) ja kaarista (arcs) (Booch ym. 1999, 24). Kaavioissa esiintyviä solmuja ovat esimerkiksi luokat, käyttötapaukset, tilat ja viestit. Kaaret koostuvat solmujen välisistä suhteista, esimerkiksi riippuvuus- ja yleistyssuhteista. (Booch ym. 1999, 18–24). Tässä tutkielmassa käytetään UML-kaavioista ja vastaavista graafisista tai tekstuaalisista kuvauksista termiä *mallintamistekniikka. Malli* on abstrahoitu kuvaus järjestelmästä, joka määrittää mallinnettavan järjestelmän tietystä näkökulmasta tietyllä abstraktiotasolla (Jacobson ym. 1999, 22). UP:ssa on määritelty erilaisia malleja, esimerkiksi analyysi- ja suunnittelumallit. Yksittäisen mallin muodostamisessa on käytetty eri UML-kaavioita, ja malli toimii eräänlaisena säiliönä tuotetuille kaavioille (Jacobson ym. 1999, 23). Tässä tutkielmassa malli-termillä viitataan UP:ssa määriteltyihin malleihin.

UML:n määritys ei ohjaa käyttämään mitään tiettyä menetelmää, koska se on itsessään vain mallinnuskieli. Se on kuitenkin suunniteltu tukemaan iteratiivista ja oliosuuntautunutta lähestymistapaa hyödyntäviä menetelmiä. Tämä on usein syynä siihen, että UML:n käyttö mielletään olevan sidoksissa tietyn menetelmän käyttöön. Fowlerin ja Kendallin (2004, 2) mukaan UML:n käytön mielletään usein edellyttävän RUP:n (Rational Unified Process) (Kruchten 2000) käyttöä.

3.2 UML-kaaviot

Tämä kohta pohjautuu pääasiassa Boochin, Rumbaughin ja Jacobsonin (1999) esittämiin UML-kaavioihin ja niiden käyttötarkoituksiin täydennettynä UML 2.1.1:n (Object Management Group, Inc. 2007) mukaisilla määrityksillä. Booch, Rumbaugh ja Jacobson esittävät, kuinka UML-kaavioita voidaan soveltaa UP-prosessimallin yhteydessä. UP:ta tarkastellaan lähemmin vasta seuraavassa

kohdassa. Jotta kaavioiden käyttötilanteet tulevat selväksi, annetaan seuraavaksi lyhyt selvitys prosessimallista. UML-kaavioiden yksityiskohtainen käsittely on jätetty tämän tutkielman ulkopuolelle.

UML tarjoaa notaation ja käsitteet mallintamiselle ohjelmistokehityksen elinkaaren eri vaiheisiin. Elinkaari jaetaan UP:ssa neljään vaiheeseen: aloitus- (interception), tarkentamis-, (elaboration), rakentamis- (construction) ja luovutusvaiheeseen (transition) (Booch, Rumbaugh & Jacobson 1999, 33). Jokaiseen elinkaaren osaan liittyy erilaisia työnkulkuja (workflows). Ne voidaan jakaa viiteen pääryhmään, jotka ovat: vaatimusten vaatimusmäärittely (requirements), analyysi (analysis), suunnittelu (design), toteutus (implementation) ja testaus (test). (Jacobson, Booch & Rumbaugh 1999, 11) Työtehtävät ovat usein samantyyppisiä prosessista riippumatta, joten UML:ää voidaan soveltaa eri ohjelmistotuotantoprosesseissa.

UML sisältää kolmetoista eri kaaviota. Nämä kaaviot ovat: luokkakaavio (class diagram), oliokaavio (object diagram), käyttötapauskaavio (use case diagram), sekvenssikaavio (sequence diagram), viestintäkaavio (communication diagram), tilakaavio (state machine diagram), toimintokaavio (activity diagram), komponenttikaavio (component diagram), sijoituskaavio (deployment diagram), koostekaavio (composite structure diagram), pakettikaavio (package diagram), kokoava vuorovaikutuskaavio (interaction overview diagram) ja ajoituskaavio (timing diagram). UML 2.x:n myötä kaavioiden lukumäärä kasvoi neljällä. Uusia kaavioita ovat paketti-, kooste-, ajoitus- ja kokoavat vuorovaikutuskaaviot. Lisäksi viestintäkaavio tunnettiin ennen yhteistyökaaviona (collaboration diagram), muttei kaaviossa ja sen käyttötarkoituksissa ole tapahtunut merkittäviä muutoksia. UML-kaaviot voidaan jakaa kahteen eri ryhmään sen mukaan, ilmentävätkö ne järjestelmän staattista vai dynaamista näkymää. Staattinen näkymä korostaa järjestelmän rakennetta, kun puolestaan dynaaminen näkymä painottaa järjestelmän

käyttäytymistä (Booch ym. 1999, 461; 466). Staattisia kaavioita ovat luokka-, olio-, komponentti-, paketti-, kooste- ja sijoituskaaviot. Dynaamisia kaavioita ovat puolestaan käyttötapaus-, sekvenssi-, viestintä-, tila-, toiminto-, ajoitus ja kokoavat vuorovaikutuskaaviot. (Object Management Group, Inc. 2007, 680)

Luokkakaavio esittää luokkia, rajapintoja, yhteistöitä (collaboration) ja niiden välisiä suhteita. Se on yleisin oliolähestymistavan mukaisista kaavioista. Luokkakaaviot ilmentävät pääasiassa järjestelmän staattisen suunnittelun näkökulmaa. (Booch ym. 1999, 25) Luokkakaaviota käytetään yleensä staattisessa mallintamisessa kolmeen tarkoitukseen (Booch ym. 1999, 108). Ensinnäkin järjestelmän sanaston mallintamisessa rajataan, mitkä käsitteet kuuluvat järjestelmän sisälle ja mitkä sen ulkopuolelle. Tässä yhteydessä käytetään usein kohdealuemallia (domain model), jonka tarkoituksena on osoittaa järjestelmän olennaiset kohteet (Jacobson ym. 1999, 119). Toiseksi luokkakaavion avulla voidaan mallintaa yksinkertaisia yhteistöitä. Sen avulla voidaan visualisoida ja määrittää luokat ja niiden väliset suhteet. Kolmanneksi luokkakaaviota voidaan käyttää tietokantakaavion loogiseen mallintamiseen. (Booch ym. 1999, 108) Luokkakaaviot voivat sisältää myös aktiivisia luokkia. Tämä tarkoittaa, että luokan ilmentymät ovat aktiivisia, joten ne omistavat prosessin tai säikeen ja voivat hallita toimintaa (Booch ym. 1999, 457). Vaatimusmäärittelyn, analyysin ja suunnittelun lisäksi luokkakaaviolla on keskeinen asema myös järjestelmän testauksessa (McQuillan & Power 2005; Zhen 2004).

Oliokaavio esittää olioita ja niiden välisiä suhteita. Se esittää staattisia tilannekuvia luokkakaavioissa esiintyvistä ilmentymistä. Samoin kuin luokkakaaviot, oliokaaviot esittävät staattisen suunnittelun näkökulman kuvaten kuitenkin todellisia tilanteita tai prototyyppejä. (Booch ym. 1999, 25)

Käyttötapauskaavio kuvaa joukon käyttötapauksia, aktoreita sekä niiden välisiä suhteita. Käyttötapauskaaviot ovat erityisen tärkeitä järjestelmän käyttäytymisen organisoinnissa ja mallintamisessa. (Booch ym. 1999, 25) Vaikka käyttötapauksia käytetään yleisesti vaatimusten määrittämiseen, UP:ssa ne ohjaavat myös suunnittelua, toteutusta ja testausta (Jacobson ym. 1999, 5).

Tilakaavio esittää tilakoneen, erilaiset tilat, siirtymät, tapahtumat ja toiminnot. Tilakaavio ilmentää järjestelmän dynaamista näkymää. Se on erittäin tärkeä rajapintojen ja luokkien käyttäytymistä mallinnettaessa. Tilakaavio korostaa olioiden tapahtumapohjaista käyttäytymistä, mikä on erityisen hyödyllistä reagoivien järjestelmien mallintamisessa. (Booch ym. 1999, 25) Tilakaaviot ovat hyödyllisiä suunnitteluvaiheessa kuvattaessa olioiden tilojen siirtymiä (Jacobson ym. 1999, 261) kuin myös testauksessa (McQuillan & Power 2005; Zhen 2004).

Toimintokaavio on tilakaavion erikoistapaus, joka osoittaa toimintojen välisen vuon. Toimintokaavio ilmentää järjestelmän dynaamisen näkymän. Se on erityisen tärkeä järjestelmän toimintojen mallintamisessa ja olioiden välisten kontrollivirtojen esittämisessä. (Booch ym. 1999, 25) Toimintokaaviota voidaan käyttää yksityiskohtaisen logiikan mallintamisessa, mutta sen merkitys liiketoimintaprosessien mallintamisessa on jatkuvasti kasvanut (Ambler 2007a; Russell ym. 2006). Toimintokaaviota käytetään myös järjestelmän testauksessa (McQuillan & Power 2005; Zhen 2004).

Komponenttikaavio osoittaa komponentit ja niiden väliset riippuvuudet. Komponenttikaaviot ilmaisevat järjestelmän staattisen toteutuksen näkymän. Ne liittyvät luokkakaavioihin, koska komponentti yleensä kattaa yhden tai useamman luokan, rajapinnan tai yhteistyön. (Rumbaugh, Jacobson & Booch 1999, 135).

Sijoituskaavio näyttää ajonaikaisesti toimivien solmujen rakenteen ja niissä olevat komponentit. Sijoituskaaviot ilmaisevat arkkitehtuurin staattisen näkymän. Ne sisältävät usein yhden tai useamman komponentin. (Booch ym. 1999, 26) Sijoituskaavio toimii tärkeänä syötteenä suunnittelulle ja toteutukselle (Jacobson ym. 1999, 227).

Koostekaavio esittää esimerkiksi luokkien tai komponenttien sisäisen rakenteen ja niiden yhteydet muihin järjestelmän osiin (Object Management Group, Inc. 2007, 193; Ambler 2007a).

Pakettikaavio kuvaa järjestelmän pakkausten, tyypillisesti alijärjestelmien, väliset riippuvuussuhteet (Koskimies ym. 2004, 25). Paketteja on käytetty aiemmissa UML-standardeissa järjestelmän kokonaisuuksien ryhmittelyyn, mutta vasta UML 2.x:n myötä pakettikaaviot ovat mukana standardikaavioiden joukossa. Paketteja hyödynnetään analyysissä ja suunnittelussa (Jacobson ym. 1999, 190; 225).

Sekvenssi-, *viestintä-*, *ajoitus-* ja *kokoavat vuorovaikutuskaaviot* ovat vuorovaikutuskaavioita. Niistä ilmenevät vuorovaikutukset, oliot ja niiden suhteet sekä niiden väliset viestit. Vuorovaikutuskaaviot ilmaisevat järjestelmän dynaamisen näkymän. Yhteistyökaaviot (UML 2.x viestintäkaaviot) korostavat niiden olioiden rakenteista järjestystä, jotka lähettävät ja vastaanottavat viestejä. (Booch ym. 1999, 25) Jacobson ym. (1999, 186) suosivat analyysivaiheessa mieluummin niiden kuin sekvenssikaavioiden käyttöä. Tämä johtuu siitä, että yhteistyökaavioiden (UML 2.x viestintäkaaviot) avulla voidaan keskittyä paremmin vaatimusten ja olioiden vastuiden yksinkertaiseen kuvaamiseen. Sekvenssikaaviot puolestaan painottavat viestien aikajärjestystä. Jacobsonin ym. (1999, 222) mukaan sekvenssikaaviot sopivatkin paremmin suunnitteluvaiheeseen kuvaamaan yksityiskohtaisemmin vuorovaikutusten

järjestystä. Lisäksi niillä on keskeinen merkitys testauksessa (McQuillan & Power 2005; Zhen 2004).

Kokoava vuorovaikutuskaavio on toimintokaavion muunnelma, joka antaa yleiskatsauksen vuorovaikutuksista ja niiden kontrollivirroista (Object Management Group, Inc 2007, 514). Kaaviossa esitettävät toimintosolmut ovat kokonaisia, esimerkiksi sekvenssikaavioilla kuvattuja vuorovaikutuksia (Koskimies ym. 2004, 29).

Ajoituskaaviota käytetään silloin, kun tarkoituksena on kuvata vuorovaikutuksia aikasidonnaisesti. Kaavio osoittaa, kuinka olion tila vaihtuu aikajanan tietyissä pisteissä. (Object Management Group, Inc. 2007, 517) Ajoituskaaviota käytetään usein sulautettujen järjestelmien suunnittelussa, mutta niitä voidaan käyttää myös liiketoimipohjaisten järjestelmien kehittämisessä (Ambler 2007a).

Taulukossa 4 on esitetty kolmetoista UML-kaaviota, niiden käyttötarkoitukset ja työnkulut, joiden yhteydessä niitä pääasiallisesti käytetään.

TAULUKKO 4. UML-kaaviot

Kaavio	Käyttötarkoitus	Työnkulut
Luokkakaavio	Järjestelmän sanaston rajaaminen, luokkien ja niiden suhteiden määritys, loogisen tietokantakaavion esittäminen	Vaatimusmäärittely, analyysi, suunnittelu ja testaus
Oliokaavio	Olioiden vastuiden ja yhteistyön esittäminen	Suunnittelu
Käyttötapauskaavio	Käyttötapausten, toimijoiden ja niiden välisten suhteiden esittäminen	Vaatimusmäärittely, analyysi, suunnittelu, toteutus ja testaus

(jatkuu)

TAULUKKO 4. UML-kaaviot. (jatkuu)

Kaavio	Käyttötarkoitus	Työnkulut
Sekvenssikaavio	Olioiden yhteistyön ja viestien aikajärjestyksen esittäminen	Suunnittelu ja testaus
Viestintäkaavio	Viestejä lähettävien ja vastaanottavien olioiden rakenteellisen järjestyksen esittäminen	Analyysi ja suunnittelu
Tilakaavio	Rajapintojen ja luokkien käyttäytymisen sekä olioiden tapahtumapohjaisen käyttäytymisen esittäminen	Suunnittelu ja testaus
Toimintokaavio	Liiketoimintaprosessien mallintaminen, toimintojen mallintaminen ja olioiden välisten kontrollivirtojen esittäminen	Vaatimusmäärittely, suunnittelu, toteutus ja testaus
Komponenttikaavio	Komponenttien ja niiden välisten riippuvuuksien esittäminen	Suunnittelu, toteutus ja testaus
Sijoituskaavio	Ajonaikaisesti toimivien solmujen rakenteen ja niissä sijaitsevien komponenttien esittäminen	Suunnittelu ja toteutus
Pakettikaavio	Järjestelmän osien ryhmittely paketteihin ja niiden välisten riippuvuuksien esittäminen	Analyysi ja suunnittelu
Koostekaavio	Järjestelmän osien rakenteiden ja yhteyksien esittäminen	Suunnittelu
Kokoava vuorovaikutuskaavio	Vuorovaikutusten ja kontrollivirtojen yleiskuvan esittäminen	Suunnittelu
Ajoituskaavio	Olioiden tilojen kuvaaminen aikasidonnaisesti	Suunnittelu

Kuten taulukosta voidaan havaita, jokaisella UML-kaaviolla on oma sisältönsä ja käyttötarkoituksensa. Silti joitakin kaavioita, kuten luokkakaaviota ja käyttötapauskaaviota, käytetään useammassa yhteydessä kuin monia muita kaavioita.

UML on saavuttanut standardinomaisen aseman ohjelmistojen mallintamisessa ja suunnittelussa (Eshuis & Wieringa 2004; Russell ym. 2006). Siitä huolimatta UML:ää on kritisoitu voimakkaasti (Berkenkötter 2003; Henderson-Sellers & Gonzalez-Perez 2006). Berkenkötterin (2003, 3) mukaan UML 1.4:ää on kritisoitu pääasiassa sen spesifikaatiosta, metamallista, käytettävyydestä, mahdollisista ristiriitaisista näkymistä ja kaavioista, mallien rakenteesta ja riittämättömästä virheidenhallinnan tuesta. Vaikka UML 2.x on tuonut mukanaan parannuksia, on siinä vielä suuria puutteita. Spesifikaatio on edelleen vapaamuotoisesti laadittu ja on näin ollen riittämätön. UML on ylikuormitettu erilaisilla kaavioilla ja elementeillä, joista saatu hyöty on kyseenalaista. (Berkenkötter 2003, 13)

3.3 UP-prosessimalli

Unified Process eli UP tarjoaa viitekehyksen ohjelmistotuotantoprosessille. Tässä kohdassa UP:ta tarkastellaan Jacobsonin, Boochin ja Rumbaughin (1999) mukaisesti. UP ohjaa toimintaa vastaamalla seuraaviin kysymyksiin: Kuka tekee mitä, milloin ja miten, jotta voidaan saavuttaa tavoiteltu tulos? UP:ssa nostetaan esille neljä tärkeää kohtaa, joiden välinen tasapaino on syytä huomioida prosessia sovellettaessa: teknologia, työkalut, ihmiset ja organisatoriset toimintamallit. (Jacobson, Booch & Rumbaugh 1999, xviii–xix) Aluksi esitellään UP:n taustaa ja rakennetta. Tämän jälkeen tarkastellaan työnkulkuja ja vaiheita.

3.3.1 Tausta ja rakenne

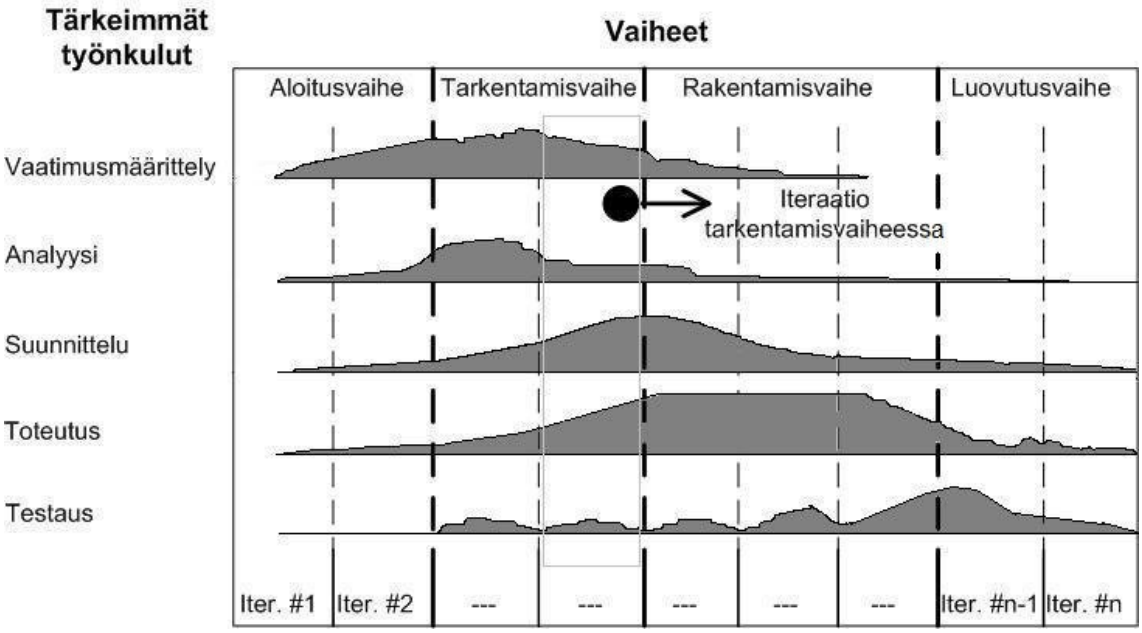
UP:n voidaan katsoa olevan kymmenien vuosien kehittämisen ja käytännön työn lopputuotos. Prosessimallin juuret juontavat vuoteen 1967, jolloin Ericssonin käyttämä järjestelmien kehitystapa loi pohjan UP:lle. Ericsson mallinsi järjestelmän jakaen sen toisiinsa liittyviin lohkoihin (blocks), jotka

nykyään tunnetaan paremmin osajärjestelminä ja komponentteina. Lohkojen tunnistamisessa ja luokittelussa käytettiin erilaisia kaavioita, jotka muistuttavat yksinkertaistettuja UML-kaavioita, kuten käyttötapaus-, luokka-, sekvenssi-, yhteistyö- ja toimintokaavioita. Ericssonilla työskennellyttä Ivar Jacobsonia voidaan pitää tämän komponenttipohjaisen kehityksen alkuunpanijana. Vuonna 1987 Jacobson lähti Ericssonilta ja perusti Objectory AB:n. Vuosina 1988–1995 kehitettiin useita versioita Objectory-prosessituotteesta, viimeisenä versiona Objectory 3.8. Käyttötapaukset nousivat keskeiseen osaan ohjaten kehitystä. Prosessi koostui peräkkäisistä työnkuluista: vaatimusmäärittely, analyysi, suunnittelu, toteutus ja testaus. Erilaiset mallit yhdistivät näitä työnkulkuja mahdollistaen jäljitettävyyden läpi prosessin. Vuonna 1995 Objectory AB siirtyi Rational Software Corporationin omistukseen ja prosessin nimeksi tuli Rational Objectory Process (ROP). Rationalin mukaantulo toi mukanaan arkkitehtuurikeskeisyyden sekä inkrementaalisen ja iteratiivisen lähestymistavan. Booch, Rumbaugh ja Jacobson toivat UML:n ROP:ssa käytettäväksi mallinnuskieleksi. Prosessin tuotenimi muuttui Rational Unified Processiksi (RUP) yrityksen yhdistyttyä muiden ohjelmistoyritysten kanssa. RUP:n taustalla on tässä kohdassa esiteltävä UP-viitekehys, mutta siihen on lisätty työnkulkuja ja ominaisuuksia kaupallisten tarpeiden tyydyttämiseksi. (Jacobson ym. 1999, xx–xxvi)

UP rakentuu kolmesta perusolettamuksesta. Se on lähtökohdiltaan käyttötapauspainotteinen, arkkitehtuurikeskeinen sekä iteratiivinen ja inkrementaalinen (Jacobson, Booch & Rumbaugh 1999, 4). Ensimmäinen tarkoittaa sitä, että käyttötapaukset ohjaavat kehitystyötä projektin alusta loppuun. Niitä ei käytetä pelkästään vaatimusten keräämiseen, vaan ne toimivat pohjana suunnittelulle, toteutukselle ja testaukselle. Arkkitehtuuri rakentuu järjestelmän tärkeimpien käyttötapausten pohjalta luoden vankan arkkitehtuurisen perustan. Iteratiivisen lähestymistavan avulla isompienkin järjestelmien kehitys voidaan jakaa pienemmiksi osaprojekteiksi.

Inkrementaalisuus puolestaan ohjaa askeleittain eteneviin lisäyksiin, jolloin kehittäjien ei tarvitse palata takaisin useamman inkrementin päähän tehdäkseen muutoksia (Jacobson ym. 1999, 385). Näin ollen iteratiivisen ja inkrementaalisen lähestymistavan avulla voidaan varautua entistä paremmin riskeihin ja muutoksiin.

UP pitää sisällään viisi erilaista työnkulkua (workflow): vaatimusmäärittely, analyysi, suunnittelu, toteutus ja testaus. Lisäksi UP jakautuu neljään vaiheeseen: aloitus- (interception), tarkentamis-, (elaboration), rakentamis- (construction) ja luovutusvaiheeseen (transition). Jokaiseen vaiheeseen voi sisältyä yhdestä useampaan iteraatiota. Iteraation sisällä suoritetaan edellä mainitut työnkulut. Niiden painotus kuitenkin vaihtelee vaiheesta riippuen. Esimerkiksi aloitusvaiheessa vaatimusmäärittelyn osuus on huomattavasti suurempi kuin toteutuksen osuus. Kuviossa 5 on annettu esimerkki siitä, miltä UP:ta hyödyntävän projektin kulku voisi näyttää. Kuviossa esitetty harmaa alue kuvaa työnkulkujen painoarvoa projektin eri vaiheissa.



KUVIO 5 UP:n työnkulut ja vaiheet (Jacobson ym. 1999, 11)

Kuviosta voidaan nähdä, kuinka esimerkiksi analyysi painottuu aikaisempiin vaiheisiin, kun taas testauksen pääpaino on loppuvaiheessa. Silti testaaminen aloitetaan jo tarkentamisvaiheessa. Projektin aikana voi esiintyä myös lyhyitä ajanjaksoja, jolloin tietyn työnkulun merkitys väliaikaisesti kasvaa. Esimerkkinä tästä on luovutusvaihe, jossa testauksen avulla löydetyt virheet nostavat tilapäisesti toteutustehtävien osuutta. Seuraavaksi esitellään tarkemmin ensin työnkulkujen ja sitten vaiheiden sisältöä.

3.3.2 UP:n työnkulut

UP:n viisi keskeisintä työnkulkua ovat: vaatimusmäärittely, analyysi, suunnittelu, toteutus ja testaus. Seuraavaksi esitellään kunkin työnkulun tehtävät, tuotokset ja toimijat. Toimijat esitetään rooleina.

Vaatimusmäärittelyn tarkoituksena on määrittää järjestelmään kohdistuvat toiminnalliset ja ei-toiminnalliset vaatimukset. Tämä edellyttää potentiaalisten vaatimusten listaamista ja liiketoiminnan ymmärtämistä. Piirrelistaa (feature list) käytetään potentiaalisten vaatimusten listaamisessa. Listaa ei kuitenkaan säilytetä varsinaisena työnkulun tuotoksena, vaan se toimii apuvälineenä vaatimusten tunnistamisessa ja arvioinnissa (Jacobson ym. 1999, 114). Liiketoimintaa mallinnetaan liiketoimintamallin (business model) tai kohdealuemallin (domain model) avulla (Jacobson ym. 1999, 115). Kohdealuemallia kuvattaessa käytetään lähinnä luokkakaaviota, ja liiketoimintamallia kuvattaessa joko käyttötapauskaavioita tai oliokaavioita (Jacobson ym. 1999, 119; 122). Toiminnalliset vaatimukset kuvataan yksittäisten käyttötapausten sekä käyttötapauskaavion avulla, ja ei-toiminnalliset vaatimukset joko täydentävät toiminnallisten vaatimusten muodostamia käyttötapauksia tai niistä muodostetaan erillisiä käyttötapauksia.

Vaatimusmäärittely etenee seuraavalla tavalla. Järjestelmäanalyytikko tunnistaa ja kuvaa järjestelmään liittyvät aktorit ja käyttötapaukset (Jacobson ym. 1999, 144). Lisäksi hän muodostaa lähinnä liiketoimintamallin tai kohdealuemallin pohjalta sanaston (glossary) järjestelmään liittyvistä käsitteistä (Jacobson ym. 1999, 140). Tämän jälkeen arkkitehti priorisoi käyttötapaukset ja muodostaa tärkeimpien tapausten pohjalta arkkitehtuurikuvauksen. Yksittäisten käyttötapausten määrittämisestä ja tarkentamisesta ovat vastuussa yksittäiset käyttötapausten määrittäjät (use-case specifier). (Jacobson ym. 1999, 153–154) Järjestelmäanalyytikko muodostaa käyttötapausten pohjalta käyttötapausmallin, joka kuvaa järjestelmän toiminnalliset ja ei-toiminnalliset vaatimukset. Malli sisältää järjestelmän yleiskuvauksen, joukon kaavioita ja tarkemman kuvauksen jokaisesta käyttötapauksesta. (Jacobson ym. 1999, 171) Käyttöliittymäsuunnittelija hahmottelee ja määrittää prototyypit järjestelmän tärkeimmistä käyttöliittymistä (Jacobson ym. 1999, 161). Taulukkoon 5 on koottu vaatimusmäärittelyn tehtävät, sen aikana syntyneet tuotokset ja tuotoksista päävastuussa olevat roolit. Usein tuotoksesta on vastuussa useampi samassa roolissa toimiva henkilö, vaikkei sitä ole tässä eikä vastaavissa tulevissa taulukoissa korostettu.

Vaatimusmäärittelyn tärkeimpänä tuotoksena on käyttötapausmalli, josta järjestelmäanalyytikko on vastuussa. Sen tarkoituksena on kuvata järjestelmän toiminnalliset ja ei-toiminnalliset vaatimukset. Taulukossa on esitetty erillisenä tehtävänä käyttötapausten määrittäminen, vaikka määrityksen tuotoksena syntyneet käyttötapaukset ovat osa käyttötapausmallia. Tällä on tarkoitus korostaa käyttötapausmäärittäjien roolia, joka on vastuussa yksittäisistä käyttötapauksista. Käyttötapausmalli toimii lähtökohtana seuraavalle työnkululle, analyysille. Tämä on ominaista UP:lle, jossa työnkulut on sidoksissa toisiinsa edeltävän työnkulun tuotoksen toimiessa seuraavan työnkulun lähtökohtana. Näin ollen syntyneet mallit riippuvat toisistaan, mikä puolestaan parantaa ratkaisujen jäljitettävyyttä. Taulukossa on tuotu esille

myös UP:lle ominainen arkkitehtuurikeskeisyys. Työnkulkujen aikana järjestelmän arkkitehtuurikuvausta päivitetään kuvaamalla mallien tärkeimmät osat.

TAULUKKO 5. UP:n vaatimusmäärittelyn tehtävät, tuotokset ja vastuut

Tehtävä	Tuotos	Vastuurooli
Järjestelmän kontekstin ymmärtäminen	Liiketoimintamalli tai kohdealuemalli	Arkkitehti
Käyttötapausten määrittäminen	Yksityiskohtaisesti määritellyt käyttötapaukset	Käyttötapausten määrittäjä
Toiminnallisten ja ei-toiminnallisten vaatimusten määrittäminen	Käyttötapausmalli	Järjestelmäanalyytikko
Käyttöliittymien määrittäminen	Käyttöliittymäluonnokset ja prototyypit	Käyttöliittymäsuunnittelija
Arkkitehtuurin kuvaaminen	Käyttötapausmalliin pohjautuva arkkitehtuurikuvaus	Arkkitehti

Analyysin aikana vaatimuksia hiotaan ja jäsennetään, jotta voitaisiin ymmärtää niitä paremmin ja näin ollen muodostaa rakenne koko järjestelmälle (Jacobson ym. 1999, 173). Kyseisen työnkulun tuloksena syntyy analyysimalli, jota käytetään vaatimusten analysointiin. Analyysimalli koostuu useista erillisistä tehtävistä: pakettien ja palvelupakettien analyysista, luokkien analyysista, käyttötapausten analyysista ja analyysimallin arkkitehtuurinäkymästä. (Jacobson ym. 1999, 213) Työnkulun alussa arkkitehti suorittaa järjestelmän arkkitehtuurianalyysin. Tämän tarkoituksena on hahmotella analyysimallia ja arkkitehtuuria tunnistamalla analyysipaketteja, olennaisimpia analyysiluokkia ja yleisiä erityisvaatimuksia (Jacobson ym. 1999, 197). Käyttötapausinsinöörin (use-case engineer) tekemän analyysin tarkoituksena on tunnistaa käyttötapaukseen liittyvät luokat, jakaa käyttötapauksen käyttäytyminen

vuorovaikutukseen osallistuvien olioiden kesken ja tunnistaa käyttötapauksen erityisvaatimukset (Jacobson ym. 1999, 204–205). Olioiden välisten vastuiden tunnistamisessa Jacobson ym. (1999, 186) suosivat yksinkertaisempia yhteistyökaavioita tarkempien sekvenssikaavioiden sijaan. Komponentti-insinööri (component engineer) tunnistaa tarkasteltavan luokan vastuut, attribuutit ja suhteet sekä luokkaan liittyvät erityisvaatimukset (Jacobson ym. 1999, 207). Analyysiluokkien tarkoituksena on keskittyä toiminnallisten vaatimusten käsittelyyn jättäen ei-toiminnallisten vaatimusten tarkastelu suunnitteluun ja toteutukseen (Jacobson ym. 1999, 181). Komponentti-insinööri muodostaa analyysipaketteja tuotoksien organisoimiseksi hallittaviin kokonaisuuksiin. Analyysipaketti voi koostua analyysiluokista, käyttötapausten analyyseistä ja muista analyysipaketeista. (Jacobson ym. 1999, 190) Kuten analyysiluokat, myös paketit rakentuvat toiminnallisista vaatimuksista. Analyysipakettien lisäksi komponentti-insinööri voi muodostaa myös palvelupaketteja, jotka tarjoavat joukon erilaisia palveluita asiakkaille, jotta liiketoiminnan kannalta tarpeelliset käyttötapaukset voitaisiin suorittaa. (Jacobson ym. 1999, 191) Taulukossa 6 on esitetty analyysin tehtävät, tuotokset ja vastuut. Analyysin aikana syntyvät muut tuotokset ovat osa analyysimallia. Analyysimalli toimii lähtökohtana seuraavalle työnkululle, joka on suunnittelu.

Suunnittelun tarkoituksena on rakentaa järjestelmää tyydyttämään sen kaikkia vaatimuksia, mukaan lukien ei-toiminnalliset vaatimukset ja muut rajoitukset (Jacobson ym. 1999, 215). Työnkulun päätuotoksena syntyy suunnittelumalli, joka toimii suunnitelmana toteutukselle. Suunnittelumalli koostuu seuraavista elementeistä: paketeista ja palvelupaketeista, luokista, käyttötapauksista ja suunnittelumallin arkkitehtuurinäkymästä. Suunnittelumallin lisäksi syntyy myös sijoitusmalli, joka määrittää järjestelmän fyysiset solmut ja komponentit. (Jacobson ym. 1999, 265)

TAULUKKO 6. UP:n analyysin tehtävät, tuotokset ja vastuut

Tehtävä	Tuotos	Vastuurooli
Vaatimusten analysointi	Analyysimalli	Arkkitehti
Arkkitehtuurin kuvaaminen	Analyysimalliin pohjautuva arkkitehtuurikuvaus	Arkkitehti
Käyttötapauksiin liittyvien olioiden, yhteistöiden ja erityisvaatimusten kuvaaminen	Käyttötapausten analyysi	Käyttötapausinsinööri
Luokkien vastuiden, attribuuttien, suhteiden ja erityisvaatimusten kuvaaminen	Analyysiluokat	Komponentti-insinööri
Järjestelmän jakaminen hallittaviin kokonaisuuksiin	Analyysi- ja palvelupaketit	Komponentti-insinööri

Arkkitehti suunnittelee järjestelmän arkkitehtuurin. Tämä sisältää osajärjestelmien ja niiden rajapintojen tunnistamisen, mikä edesauttaa osajärjestelmiin jakamista eri järjestelmätasoilla ja näin ollen mahdollistaa osajärjestelmien hajauttamisen. Käyttötapausinsinööri tunnistaa käyttötapaukseen osallistuvat luokat, vuorovaikutuksessa olevat oliot, määrittää suunnitteluluokkien operaatiot ja osajärjestelmät ja niiden rajapinnat sekä määrittää käyttötapausten toteutukseen liittyvät vaatimukset (Jacobson ym. 1999, 249). Luokkien ja osajärjestelmien kuvaamisessa käytetään apuna luokkakaavioita (Jacobson ym. 1999, 250; 254). Kun halutaan kuvata, kuinka olioiden välinen vuorovaikutus toimii, käytetään sekvenssikaavioita (Jacobson ym. 1999, 251). Komponentti-insinööri luo suunnitteluluokat, jotka toteuttavat käyttötapausten toiminnalliset ja ei-toiminnalliset vaatimukset. Hänen tehtävänään on suunnitella luokan operaatiot ja metodit, attribuutit, suhteet, tilat, riippuvuudet geneerisiin suunnittelumekanismeihin, toteutukseen liittyvät vaatimukset ja rajapinnat. (Jacobson ym. 1999, 255) Luokan eri tilojen kuvaamisessa käytetään tilakaavioita (Jacobson ym. 1999, 261). Geneeristen

suunnittelumekanismien avulla pyritään käsittelemään vaatimuksia, jotka liittyvät pysyvyyteen, läpinäkyvään olioiden jakamiseen, tietoturvaan, virheiden tunnistukseen ja niistä toipumiseen sekä tapahtumien hallintaan (Jacobson ym. 1999, 246). Suunnitteluluokkien yhteydessä tämä voi käytännössä tarkoittaa esimerkiksi abstraktien luokkien ja yleistyssuhteiden lisääntymistä. Komponentti-insinööri on vastuussa myös osajärjestelmien suunnittelusta. Tarkoituksena on varmistaa, että osajärjestelmä on mahdollisimman riippumaton ja tarjoaa oikeat rajapinnat, sekä suorittaa rajapintojen määrittämät operaatiot (Jacobson ym. 1999, 263). Vaikka arkkitehti on määritellyt osajärjestelmien rajapinnat, voidaan niitä joutua muokkaamaan suunnittelumallin kehittymisen ja käyttötapausten suunnittelun myötä (Jacobson ym. 1999, 263–264). Taulukossa 7 on esitetty suunnittelun tehtävät, tuotokset ja vastuut. Lukuun ottamatta sijoitusmallia muut tuotokset kuuluvat vaiheen päätuotokseen, suunnittelumalliin, joka toimii lähtökohtana toteutukselle.

Neljäntenä työnkulkuna on toteutus. Toteutuksen päätavoitteena on tuottaa varsinaista koodia suunnittelun perusteella, mutta työnkulun tuotoksena syntyvä toteutusmalli sisältää muutakin kuin pelkkää ohjelmakoodia. Työnkulun aikana toteutetaan osajärjestelmät ja niiden riippuvuudet, rajapinnat ja sisällöt. Lisäksi toteutetaan komponentit ja tiedostot sekä suoritetaan komponenttien yksikkötestaus. Arkkitehti luo toteutusmallista arkkitehtuurikuvauksen, joka sisältää arkkitehtuurisesti tärkeimmät elementit. Toteutusmalliin sisältyvien tuotoksien lisäksi arkkitehti päivittää sijoitusmallia sitä mukaa, kun suoritettavia komponentteja lisätään solmuihin. (Jacobson ym. 1999, 293) Järjestelmäintegroija tekee integrointisuunnitelman, joka kuvaa iteraation sisältämät versiot (build) ja niiden vaatimukset. Lisäksi hän integroi jokaisen version ennen integrointitestausta. (Jacobson ym. 1999, 283) Komponentti-insinööri toteuttaa osajärjestelmät ja luokat sekä suorittaa yksikkötestauksen (Jacobson ym. 1999, 280). Yksikkötestaus sisältää

kahdenlaista testausta. Määritystestauksen (specification testing, ”black-box testing”) tarkoituksena on varmistaa, että komponentti suorittaa haluttuja toimintoja, puuttumatta komponentin sisäiseen toteutukseen. Rakennetestauksen (structure testing, ”white-box testing”) tarkoituksena on puolestaan varmentaa komponentin sisäinen toiminta. (Jacobson ym. 1999, 289–291) Taulukkoon 8 on tiivistetty toteutuksen tehtävät, tuotokset ja vastuut.

TAULUKKO 7. UP:n suunnittelun tehtävät, tuotokset ja vastuut

Tehtävä	Tuotos	Vastuurooli
Toteutuksen suunnittelu	Suunnittelumalli	Arkkitehti
Järjestelmän hajauttamiseen liittyvä suunnittelu	Sijoitusmalli	Arkkitehti
Arkkitehtuurin kuvaaminen	Suunnittelumalliin pohjautuva arkkitehtuurikuvaus	Arkkitehti
Käyttötapauksiin osallistuvien olioiden, luokkien, operaatioiden, osajärjestelmien ja rajapintojen sekä käyttötapausten toteutukseen liittyvien vaatimusten määrittäminen	Käyttötapausten suunnittelu	Käyttötapausinsinööri
Luokan operaatioiden, metodien, attribuuttien, suhteiden, tilojen, geneeristen suunnittelumekanismien ja toteutukseen liittyvien vaatimuksien suunnittelu sekä rajapintojen tuottaminen	Suunnitteluluokat	Komponentti-insinööri
Osajärjestelmien riippumattomuuden ja rajapintojen oikeellisuuden varmistaminen	Osajärjestelmien suunnittelu	Komponentti-insinööri

Viimeisen työnkulun eli testauksen tarkoituksena on verifioida toteutuksen tulokset, aina välivaiheista lopullisiin versioihin (Jacobson ym. 1999, 295). Työnkulun päätuotoksena syntyy testimalli, joka pitää sisällään testitapaukset, testikäytännöt ja testikomponentit. Testimallin lisäksi testauksen aikana tehdään testaussuunnitelma, suoritetaan toteutettujen testien arvioinnit ja

kirjataan puutteet (defect), jotka voidaan ohjata syötteeksi muille työnkuluille, kuten suunnitteluun ja toteutukseen. (Jacobson ym. 1999, 313)

TAULUKKO 8. UP:n toteutuksen tehtävät, tuotokset ja vastuut

Tehtävä	Tuotos	Vastuurooli
Järjestelmän toteutus	Toteutusmalli	Arkkitehti
Arkkitehtuurin kuvaaminen	Toteutusmalliin pohjautuva arkkitehtuurikuvaus	Arkkitehti
Integroinnin määrittäminen ja hallinta	Integrointisuunnitelma	Järjestelmäintegroija
Järjestelmän osien toteutus	Osajärjestelmien ja komponenttien toteutus ja testaus	Komponentti-insinööri
Komponenttien sijoittaminen solmuihin	Sijoitusmallin päivitys	Arkkitehti

Testaukseen osallistuu neljässä eri roolissa toimivia testaajia. Testi-insinööri tekee testaussuunnitelman (test planning), jossa kuvataan testausstrategia, resurssiarviot ja resurssien aikataulutus. Suunnitelman syötteenä ovat edellisissä työnkuluissa syntyneet mallit, kuten analyysi- ja suunnittelumalli, sekä lisävaatimukset ja arkkitehtuurikuvaus. (Jacobson ym. 1999, 305) Testaussuunnitelman jälkeen testi-insinööri suunnittelee yksittäiset testit (design test). Testien suunnittelun tarkoituksena on tunnistaa ja kuvata versioiden testitapaukset sekä tunnistaa ja organisoida testikäytänteet, jotka määrittävät, kuinka testitapaukset tulisi suorittaa. (Jacobson ym. 1999, 306) Testitapaukset liittyvät erilaisiin testeihin, kuten integrointi-, järjestelmä-, regressiotesteihin. Komponentti-insinööri on vastuussa testien toteuttamisesta. Tarkoituksena on pyrkiä automatisoimaan testikäytänteet mahdollisimman pitkälti luomalla testikomponentteja. (Jacobson ym. 1999, 309) Integrointitestaaja suorittaa iteraatioiden aikana integrointitestejä ja järjestelmätestaaja järjestelmään kohdistuvia testejä puutteiden havaitsemiseksi

(Jacobson ym. 1999, 310–311). Testien jälkeen testi-insinööri arvioi niiden onnistumista. Hän vertaa testeistä saatuja tuloksia testaussuunnitelmaan, minkä perusteella voidaan tarkastella testien kattavuutta ja järjestelmän luotettavuutta. Arvioinnin tuloksia voidaan hyödyntää seuraavia iteraatioita suunniteltaessa. (Jacobson ym. 1999, 312) Taulukkoon 9 on koottu testauksen tehtävät, tuotokset ja vastuut.

TAULUKKO 9. UP:n testauksen tehtävät, tuotokset ja vastuut

Tehtävä	Tuotos	Vastuurooli
Järjestelmän testauksen kuvaaminen	Testimalli	Testi-insinööri
Testien suunnittelu	Testitapaukset ja -käytänteet	Testi-insinööri
Testauksen automatisointi	Testikomponenttien toteutus	Komponentti-insinööri
Testauksen suunnittelu	Testaussuunnitelma	Testi-insinööri
Puutteiden löytäminen	Havaitut ja raportoidut puutteet	Integrointi-, ja järjestelmätestaaja
Järjestelmän kypsyyden ja luotettavuuden arviointi	Testauksen arviointi	Testi-insinööri

3.3.3 UP:n vaiheet

Seuraavaksi esitellään lyhyesti UP:n neljä vaihetta: aloitus-, tarkentamis-, rakentamis- ja luovutusvaihe. Tässä yhteydessä tuodaan esille jokaisen vaiheen keskeisimmät työnkulut. Jokaisen vaiheen sisällä suoritetaan yhdestä useampaan iteraatiota, joista jokainen sisältää edellä esitellyt työnkulut. Se, kuinka kauan yksi iteraatio kestää ja montako iteraatiota yhdessä vaiheessa on, riippuu muun muassa projektin koosta ja monimutkaisuudesta. Jacobson ym. (1999, 327) sanovat yksittäisen iteraation keston vaihtelevan keskimäärin viikosta kolmeen kuukauteen. Jokaisen vaiheen alussa suunnitellaan tarvittavat

iteraatiot ja vaiheen lopussa tehdään seuraavan vaiheen alustava suunnitelma. Vaikka iteraatiot voivat olla vaiheiden sisällä hyvinkin lyhyitä ja toisiinsa sidottuja, saattaa tarkentamis- ja rakentamisvaiheen välinen aikaväli kasvaa suunniteltua suuremmaksi (Jacobson ym. 1999, 382). Jacobsonin ym. (1999, 382) mukaan tämän seurauksena projektisuunnitelmaa joudutaan usein muuttamaan mahdollisten henkilöstö-, resurssi- ja aikataulumuutosten vuoksi.

Ensimmäisen vaiheen, aloitusvaiheen, päätarkoituksena on projektin käynnistys. Jotta kyettäisiin varmistamaan, että projekti voidaan käynnistää, tulee muun muassa tehdä järjestelmän rajaus, määrittää kustannus- ja aikataulurajoitteet sekä arvioida kriittisiä riskejä (Jacobson ym. 1999, 342). Aloitusvaiheessa tehdään projektisuunnitelma, joka sisältää yleisen suunnitelman projektin jokaiselle vaiheelle. Koska tässä vaiheessa projektia ei voida olla vielä varmoja, kannattaako järjestelmää lähteä rakentamaan, heijastuu se myös aloitusvaiheen toimintoihin ja tuotoksiin. Alussa järjestelmän tavoitteet ja vaatimukset voivat olla hyvinkin karkeajakoisia. Näin ollen on selvää, että vaiheen toiminnot liittyvät ensimmäisten työnkulkujen tuotoksiin. Vaikka toteutus ja testaus jäävät muita työnkulkuja vähemmälle huomiolle, voidaan aloitusvaiheessa tarvittaessa toteuttaa prototyyppi lähinnä uudentyyppisen järjestelmän konseptin demonstroimiseksi ja siten riskien vähentämiseksi (Jacobson ym. 1999, 346). Vaatimusmäärittely on vaiheen keskeisin työnkulku. Vaatimusten ja käyttötapausten tunnistamisen avulla rajataan järjestelmä ja luodaan alustava järjestelmäarkkitehtuuri (Jacobson ym. 1999, 347). Tarkempaa käyttötapausten analyysia tai niihin pohjautuvaa suunnittelua ei yleensä suoriteta vielä tässä vaiheessa.

Tarkentamisvaiheen tavoitteena on kerätä lähes kaikki vaatimukset ja muotoilla ne käyttötapauksiksi, luoda pysyvä arkkitehtuuri, jatkaa kriittisten riskien tarkkailua ja täydentää projektisuunnitelmaa (Jacobson ym. 1999, 359–360). Tässä vaiheessa järjestelmää pyritään tarkastelemaan mahdollisimman laajasta

näkökulmasta tavoitteiden saavuttamiseksi. Vaiheen tärkeimmät työnkulut ovat vaatimusmäärittely, analyysi ja suunnittelu. Järjestelmän vaatimuksista muodostetaan vaiheen aikana liiketoimintamalli, käyttötapauskaavio ja -kuvaukset sekä tunnistetaan luokkia ja paketteja. Vaihe kattaa myös edellisten tuotosten sekä arkkitehtuurin suunnittelun. Arkkitehtuurisesti tärkeimpien luokkien ja osajärjestelmien toteuttaminen ja testaus aloitetaan tässä vaiheessa (Jacobson ym. 1999, 375–376).

Rakentamisvaiheen tavoitteena on tuottaa ensimmäinen julkaisukunnossa oleva versio järjestelmästä eli niin sanottu beta-julkaisu (Jacobson ym. 1999, 381). Vaatimusmäärittely ja analyysi ovat tässä vaiheessa pienemmässä roolissa ja suunnittelun osuus on vähentynyt edellisvaiheeseen verrattuna, kun taas toteutuksen ja testauksen merkitys on kasvanut huomattavasti. Rakentamisvaiheessa tunnistetaan ja kuvataan loputkin vaatimukset. Järjestelmän arkkitehtuurin kannalta tärkeimpien osien analyysi ja suunnittelu on tehty jo edellisessä vaiheessa, mutta niiden yksityiskohtainen suunnittelu ja etenkin vähemmän tärkeiden osien suunnittelu on vielä kesken.

Vaiheista viimeisenä on luovutusvaihe, jonka lopullisena tavoitteena on asiakkaan vaatimukset täyttävän järjestelmän toimittaminen sekä asiakasympäristössä toimivaan järjestelmään kohdistuvien ongelmien, kuten vikojen korjaaminen (Jacobson ym. 1999, 395).

3.4 Yhteenveto

Tässä luvussa esiteltiin UML-mallinnuskieli ja UP-prosessimalli. UML on yleiskäyttöinen visuaalinen mallinnuskieli, jota käytetään tietojärjestelmän osien määrittämisessä, havainnollistamisessa, rakentamisessa ja dokumentoinnissa. UML 2.x sisältää kolmetoista eri kaaviota, ja ne voidaan jakaa kahteen eri ryhmään sen mukaan, ilmentävätkö ne järjestelmän staattista

vai dynaamista näkymää. Staattisia kaavioita ovat luokka-, olio-, komponentti-, paketti-, kooste- ja sijoituskaaviot. Dynaamisia kaavioita ovat puolestaan käyttötapaus-, sekvenssi-, viestintä-, tila-, toiminto-, ajoitus ja kokoavat vuorovaikutuskaaviot.

UP on ohjelmistotuotannon viitekehys, joka rakentuu kolmesta perusolettamuksesta. Se on lähtökohdiltaan käyttötapauspainotteinen, arkkitehtuurikeskeinen sekä iteratiivinen ja inkrementaalinen. UP sisältää viisi erilaista työnkulkua: vaatimusmäärittelyn, analyysin, suunnittelun, toteutuksen ja testauksen. Kunkin työnkulun osalta on määritelty sen tuotokset ja tuotosten vastuuroolit. Lisäksi UP jakautuu neljään vaiheeseen: aloitus-, tarkentamis-, rakentamis- ja luovutusvaiheeseen. Kuhunkin vaiheeseen sisältyy yksi tai useampi iteraatio, joissa suoritetaan edellä mainittuja työnkulkuja.

Vaikka UML ja UP ovatkin toisistaan riippumattomia, on UML-kaavioiden käyttöä pyritty selittämään UP:n työnkulkujen avulla. Eri kaaviot soveltuvat käytettäviksi eri työnkulkujen yhteydessä. Eräitä kaavioita, kuten käyttötapaus- ja luokkakaavioita, on käytetty useammissa työnkuluissa kuin muita kaavioita.

4 KETTERÄT MENETELMÄT

Tämän luvun tarkoituksena on tarkastella ketteriä menetelmiä ja niiden prosesseja ja niissä suoritettavaa mallintamista. Aluksi valitaan tarkasteltavat menetelmät, jotka täyttävät tutkimuksen kannalta keskeisimmät valintakriteerit. Valittujen menetelmien osalta kuvataan niille ominaisia näkemyksiä ja käytänteitä sekä prosessin vaiheet ja tuotokset. Sen jälkeen tarkastellaan mallintamista ja sen yhteyttä prosessin vaiheisiin ja tuotoksiin. Tämän tarkoituksena on selvittää sitä, mitä UML-kaavioita tai muita mallintamistekniikoita menetelmässä käytetään.

4.1 Menetelmien valinta

Seuraavaksi esitellään valintakriteerit tarkasteltavien menetelmien osalta, listataan potentiaaliset menetelmät ja valitaan tämän tutkimuksen kannalta sopivimmat menetelmät. Menetelmien tunnettavuudella ja yleisyydellä on merkitystä menetelmien valintaan. Mitä yleisempi menetelmä, sitä merkittävämpänä ja hyödyllisempänä saatuja tuloksia voidaan pitää. Lisäksi menetelmän yleisyys tarkoittaa yleensä sitä, että lähdemateriaalia on riittävästi saatavissa. Lähdemateriaalin laajuus lisää luonnollisesti tutkimuksen luotettavuutta. Yleisimmät menetelmät ovat Extreme Programming (XP) (Beck 1999) ja Scrum (Advanced Development Methods, Inc. 2005). Muita tunnettuja ketteriä menetelmiä ovat muun muassa, Feature Driven Development (FDD) (Nebulon Pty. Ltd. 2004), Crystal-menetelmät (Cockburn 2005), DSDM (DSDM Consortium 2007) ja Adaptive Software Development (ASD) (Highsmith 2000), Agile Unified Process (AUP) (Ambler 2007b) ja Microsoft Solutions Framework (MSF) (Microsoft Corporation 2007). Tutkimuksen kannalta on myös oleellista, että menetelmässä on määritelty prosessi ja mallintamistekniikat sillä tasolla, että niiden analysoinnissa voidaan hyödyntää UP:ta ja UML:ää. Menetelmiä voidaan luokitella muun muassa sen mukaan, mitkä eri lähteet ovat

vaikuttaneet niiden kehitykseen. Valintakriteerinä voidaan käyttää esimerkiksi sitä, että valittavat menetelmät edustavat mahdollisimman erilaisia tai samanlaisia lähtökohtia.

Tähän tutkimukseen valittavien menetelmien tärkeimpinä valintakriteereinä ovat olleet menetelmän yleisyys ja tunnettavuus sekä menetelmän tarkastelulla saavutettava kontribuutio tuloksia arvioitaessa. Kun tarkastellaan yllä listattujen menetelmiä toisessa luvussa esitettyjen tutkimusten pohjalta, voidaan todeta, että XP, Scrum ja FDD täyttävät valintakriteerit yleisyyden ja tunnettavuuden suhteen. Näistä kolmesta Scrum on ainoa menetelmä, joka ei määrittele eikä ota kantaa lainkaan käytettäviin mallintamistekniikoihin (Abrahamsson ym. 2002, 27). Näin ollen saatujen tulosten oletettu kontribuutio jäisi tässä tapauksessa olemattomaksi. XP ja FDD täyttävät molemmat valintakriteerit. Muita menetelmiä on vaikea asettaa tarkkaan järjestykseen sen mukaan, kuinka yleisiä ne ovat. DSDM:n ja Crystal-menetelmien tulokset oli mainittu erikseen useammassa tutkimuksessa kuin AUP:n ja MSF:n. Muiden menetelmien, esimerkiksi ASD:n merkitys, näytti olevan häviävän pieni, koska niiden osuuksia ei raportoitu erikseen yhdessäkään tutkimuksessa. Crystal-menetelmät eivät määrittele käytettäviä mallintamistekniikoita, joten ne jätetään pois tästä tutkimuksesta. DSDM ja MSF ovat molemmat parhaiden käytäntöjen pohjalta kehittyneitä viitekehyksiä. DSDM:ssä on määritelty mahdolliset käytettävät mallintamistekniikat, kun taas MSF:ssä niitä ei ole määritelty. AUP on yksinkertaistettu versio RUP:sta (Ambler 2007b). Siinä esiintyy vahvasti UP:n ja UML:n piirteitä, ja se olisi näin ollen potentiaalinen vaihtoehto, kun haluttaisiin tarkastella sitä, missä määrin prosessin ja mallintamisen luonne eroaa ketterässä ohjelmistokehitysympäristössä. Tarkasteltavien menetelmien määrä on resurssien vuoksi rajattu kolmeen. XP:n ja FDD:n lisäksi kolmanneksi menetelmän valinta suoritetaan DSDM:n ja AUP:n välillä. DSDM on lähtökohdiltaan perinteisempi ketterä menetelmä kuin AUP ja se on ollut vuosikausia laajemmassa käytössä. Tutkimukseen halutaan

valita lähtökohdiltaan perinteisempi menetelmä kuin ääripään edustaja, joten kolmanneksi menetelmäksi valitaan DSDM.

4.2 Extreme Programming (XP)

Extreme Programming (XP) on tunnetuin ketterä menetelmä. XP on kehittynyt ratkaisuksi perinteisten menetelmien pitkien kehityskaarien aiheuttamille ongelmille. Menetelmän tärkeimpänä kehittäjänä ja puolestapuhujana voidaan pitää Kent Beckiä, vaikkakin osallisina ovat olleet myös Ward Cunningham ja Ron Jeffries. (Astels, Miller & Novak 2002, xxi).

Kuten kaikilla ketterillä menetelmillä, myös XP:llä on sille ominaisia päämääriä ja periaatteita, jotka ohjaavat toimintaa. Seuraavaksi esiteltävät arvot, periaatteet ja käytänteet pohjautuvat Waken (2005) tiivistelmään Beckin ja Andresin (2004) teoksesta sekä Marchesin (2006) vertailuun aiempien ja nykyisten arvojen, periaatteiden sekä käytänteiden välillä. Arvoissa, periaatteissa ja käytänteissä on tapahtunut selkeitä muutoksia Beckin viisi vuotta aiemmin julkaisemiin listauksiin. XP:n viisi arvoa ovat: viestintä, yksinkertaisuus, palaute, rohkeus ja kunnioitus. Näistä arvoista viimeisin eli kunnioitus on tullut myöhemmin mukaan. Neljätoista periaatetta ovat: inhimillisyys, talous, molemminpuolinen etu, lähtökohtainen samankaltaisuus (self-similarity), kehitys (improvement), moninaisuus, reflektointi, tasainen työvirtaus (flow), mahdollisuus, ylimäärä, epäonnistumisen hyväksyminen (failure), laatu, eteneminen pienin askelein ja vastaanotettu vastuu. Useimmat arvoista käyvät tarpeeksi selvästi ilmi asianomaisista termeistä. Seuraavassa kerrotaan lyhyesti muista. Lähtökohtainen samankaltaisuus tarkoittaa sitä, että pyritään soveltamaan ennestään toimivia ratkaisuja muissa vastaavanlaisissa tilanteissa. Käytännössä tämä voi tarkoittaa esimerkiksi suunnittelumallien (design pattern) soveltamista. Moninaisuudella viitataan siihen potentiaalin, mikä syntyy, kun erityyppiset henkilöt työskentelevät yhdessä yhteisen

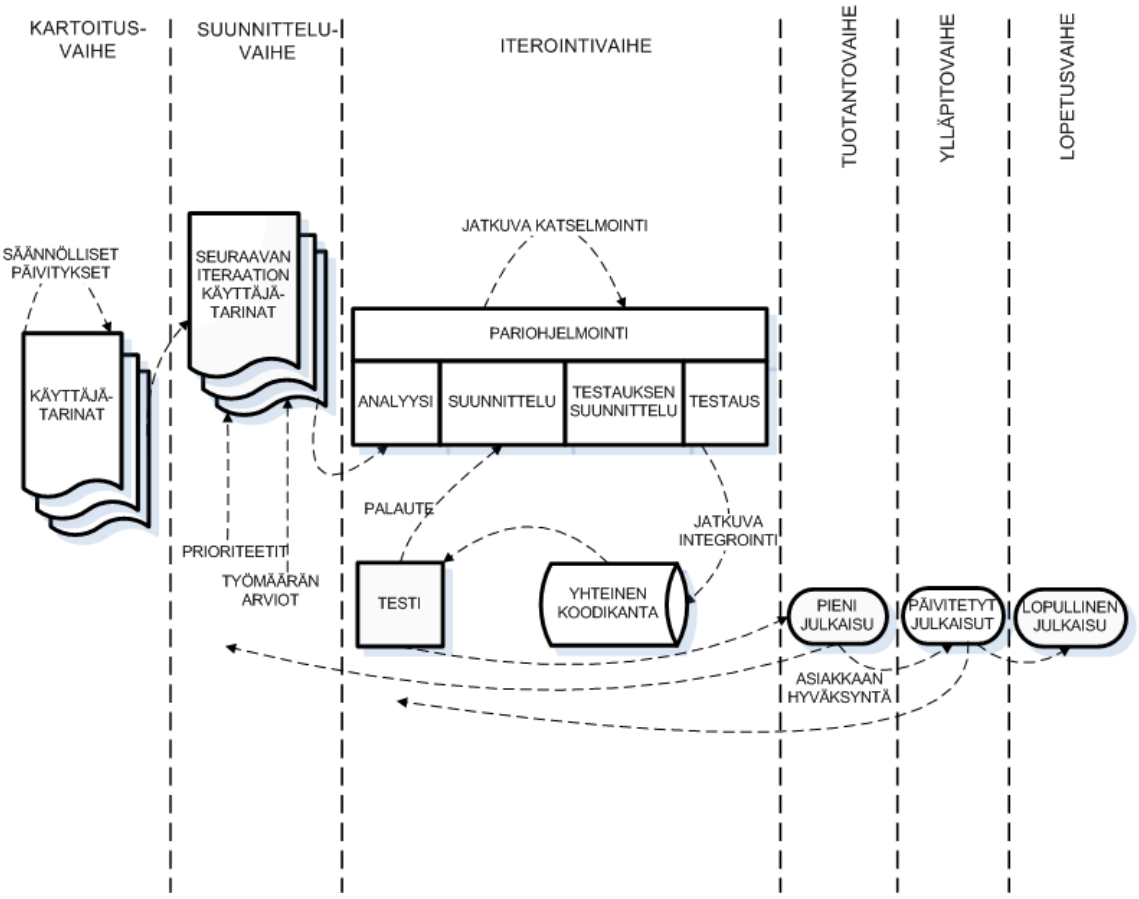
ongelman ratkaisemiseksi. Henkilöiden erilaisuus mahdollistaa useampien ratkaisuvaihtoehtojen esiintulon. Ylimäärällä tarkoitetaan sitä, että pyritään löytämään erilaisia, osittain päällekkäisiä ratkaisuja, erityisesti vaikeita ja tärkeitä ongelmia ratkaistaessa.

Arvojen ja periaatteiden pohjalta on rakentunut kolmetoista pääkäytännettä: istu yhdessä, yhtenäinen tiimi, informatiivinen työtila, innostava työ, pariohjelmointi, käyttäjätarinat, viikoittainen kiertokulku, neljännesvuosittainen kiertokulku, väljyys, kymmenen minuutin päivitysvälit, jatkuva integrointi, testaa ensin -ohjelmointi ja inkrementaalinen suunnittelu. Kiertokuluilla tarkoitetaan sitä, että jatkuva kehitystyö etenee lyhyissä, viikon mittaisissa sykleissä. Sen lisäksi toimintaa suunnitellaan pitemmissä, kuukausien mittaisissa, ajanjaksoissa. Väljyydellä tarkoitetaan sitä, että suunnitelmaan on otettu mukaan toissijaisia tehtäviä, jotka voidaan tarvittaessa jättää pois. Pääkäytänteiden lisäksi voidaan tarvittaessa soveltaa erilaisia lisäkäytänteitä. Pääkäytänteitä voidaan ottaa käyttöön yksi kerrallaan, mutta lisäkäytänteet vaativat tuekseen toisia käytänteitä. Käytänteitä voidaan siis soveltaa tilanteeseen parhaiten katsotulla tavalla. (Wake 2005)

Kuviossa 6 on kuvattu XP:n prosessi, joka koostuu viidestä vaiheesta: kartoitus (exploration), suunnittelu (planning), iterointi (iterations to release), tuotanto (productionizing) sekä ylläpito ja lopetus (maintenance and death). (Abrahamsson ym. 2002, 19) Prosessi on luonteeltaan inkrementaalinen ja iteratiivinen. Pienet, jatkuvat julkaisut ajavat kehitystä eteenpäin.

Abrahamsson ym. (2002, 19–21) tiivistävät Beckin (1999) esittämän XP:n prosessin seuraavanlaisesti. Kartoitusvaiheessa kerätään asiakkailta käyttäjätarinoiden (user stories) avulla vaatimuksia, joita he haluavat ensimmäiseen ohjelmiston julkaisuun sisällytettävän. *Käyttäjätarinat* ovat asiakkaiden laatimia kuvauksia järjestelmän toiminnasta. Käyttäjätarinoiden

tarkoitus ei ole tuottaa yksityiskohtaista kuvausta, vaan tarkoituksena on antaa riittävät perustiedot kehittäjille vaatimusten toteutusajan arvioimiseksi. Yksityiskohdista sovitaan myöhemmin käyttäjien kanssa. (Wells 2004) Kartoitusvaiheessa työkalut, käytettävä teknologia sekä käytänteet kartoitetaan ja sovitaan. Vaihe kestää muutamasta viikosta pariin kuukauteen.



KUVIO 6. XP:n prosessin elinkaari (Abrahamsson ym. 2002, 19)

Suunnitteluvaiheen tavoitteena on muodostaa julkaisusuunnitelma. Vaatimukset asetetaan tärkeysjärjestykseen ja sovitaan ensimmäisen julkaisun sisällöstä. Jokaisen käyttäjätarinan osalta ohjelmoijat arvioivat toteutukseen kuluvan ajan jakamalla käyttäjätarinan tehtäviin ja kirjaamalla ne tehtäväkorteille. Tämän perusteella muodostetaan asiakkaan kanssa aikataulu.

Ensimmäisen julkaisun aikataulu on yleensä enintään kaksi kuukautta. Itse suunnitteluvaihe kestää muutaman päivän.

Iterointivaiheessa suoritetaan useita iteraatioita, joista kukin kestää viikosta neljään viikkoon. Jokaisen iteraation alussa muodostetaan iterointisuunnitelma. Mittaamalla edellisessä iteraatiossa valmistuneita käyttäjätarinoita ja vertaamalla niitä käyttäjätarinoiden työmääräarvioihin, saadaan määriteltyä niin sanottu projektin nopeus (project velocity). Projektin nopeutta käytetään apuna alkavan iteraation aikataulutuksessa. Asiakas valitsee alkavaan iteraatioon projektin nopeutta vastaavan määrän käyttäjätarinoita. Käyttäjätarinat valitaan julkaisusuunnitelmassa olevien toteuttamattomien ja hyväksymistesteissä läpäisemättömien käyttäjätarinoiden joukosta. Kehittäjät jakavat tehtäväkortit keskenään. Asiakas luo hyväksymistestit käyttäjätarinoiden perusteella toiminnallisuuden varmentamiseksi. (Wells 2004) Iteraatioiden aikana kehitystyö suoritetaan pareittain XP:n käytänteiden mukaisesti. Parit suorittavat koodauksen kanssa rinnakkain Kuvion 5 mukaisesti järjestelmän analyysiä, suunnittelua, testauksen suunnittelua ja testausta. Ensimmäisellä iteraatiokerralla luodaan koko järjestelmän kattava arkkitehtuuri. Vaiheeseen valitaan koko järjestelmää tukevat käyttäjätarinat. Viimeisen iteraation jälkeen järjestelmä on valmis siirtymään tuotantovaiheeseen.

Tuotantovaiheen iteraatiot ovat sisällöltään vastaavanlaisia edellisen vaiheen iteraatioiden kanssa. Testauksen merkitys korostuu, kun halutaan varmistua etenkin järjestelmän suorituskyvyn toimivuudesta ennen julkaisemista. Tässä vaiheessa voidaan tehdä vielä uusia muutoksia. Vaiheen aikana iteraatiot saattavat lyhentyä ja kestää viikosta kolmeen viikkoon.

Ylläpitovaiheessa tuetaan asiakkaalle toimitettua järjestelmää ja samalla suoritetaan uusia iteraatioita. Kehitys hidastuu yleensä tässä vaiheessa.

Lopetusvaiheessa ei toteuteta enää mitään, vaan tehdään välttämätön dokumentointi. Arkkitehtuuriin, suunnitteluun ja koodiin ei tehdä muutoksia. Joskus voidaan tehdä myös projektin lopettamispäätös ennenaikaisesti, esimerkiksi kustannusten noustessa.

Kuvion 6 tarkoituksena on kuvata vaiheiden ajallista päällekkäisyyttä, mikä on luonnollista XP:lle. Kun tuotantovaiheessa ilmenee korjattavaa tai muita muutosvaatimuksia, palataan varhaisempiin vaiheisiin. Vastaavasti menetellään myös ylläpitovaiheessa, kun aloitetaan uusi iteraatio. Jokainen uusi julkaisu tulisi aloittaa kartoitusvaiheella (Beck 1999, 93). Iterointivaihetta voidaan pitää prosessin keskeisimpänä vaiheena, jonka aikana varsinainen kehitystyö tapahtuu. Iterointivaiheessa esiintyy rinnakkain ohjelmistokehitykselle tyypillisiä työnkulkuja, kuten analyysi, suunnittelu, toteutus ja testaus.

Taulukkoon 10 on koottu XP:n vaiheet, niiden tavoitteet ja tuotokset. Kuten nähdään, XP:n vaiheiden tuotokset ovat jokseenkin epäformaaleja. Tämä johtuu siitä, ettei vaiheille ole määritelty selkeitä tavoitteita eikä tuotoksia. Käyttäjätarinat ovat keskeisin prosessia ohjaava tuotos, jonka merkitys näkyy useamman vaiheen aikana.

Seuraavaksi tarkastellaan tarkemmin XP:ssä suoritettavaa mallintamista ja sen yhteyttä prosessin vaiheisiin, työnkulkuihin ja niiden aikana syntyviin tuotoksiin.

XP:n tärkein tuotos ohjelmakoodin lisäksi ovat käyttäjätarinat. Niiden on tarkoitus kuvata järjestelmän toiminnallisia vaatimuksia. Ne ovat joissain määrin rinnastettavissa käyttötapauksiin, jotka toimivat vaihtoehtona käyttäjätarinoille. Jeffries sanoo kuitenkin, ettei XP suosittele niiden käyttöä, mutta toki niitä voi halutessaan käyttää (Astels ym. 2002, 43). Astels ym. (2002,

45) näkevät käyttötapaukset ongelmallisiksi XP:n yhteydessä aikataulutuksen ja asiakkaan roolin vuoksi. Jotta käyttötapaukset voitaisiin aikatauluttaa oikein, tulisi niiden eri vaiheet, alku- ja jälkitilat määrittää tarkasti. Tämä voi olla hyvinkin vaikeaa. Kehittäjät kirjoittavat usein käyttötapaukset. Näin ollen asiakkaiden tietämys kohdealueesta ei välttämättä välity käyttötapauksiin. Haittapuolena käyttäjätarinoissa on se, ettei niitä luotaessa huomioida ei-toiminnallisia vaatimuksia, toisin kuin käyttötapauksissa.

TAULUKKO 10. XP:n prosessin vaiheet, tavoitteet ja tuotokset

Vaihe	Tavoite	Tuotokset
Kartoitusvaihe	Vaatimusten kerääminen, työkalujen, teknologian sekä käytänteiden kartoitus ja sopiminen	Alustavat käyttäjätarinat
Suunnitteluvaihe	Vaatimusten priorisointi, ensimmäisen julkaisun sisällön sopiminen ja aikataulun muodostus	Julkaisusuunnitelma, priorisoidut käyttäjätarinat, tehtäväkortit
Iterointivaihe	Järjestelmäarkkitehtuurin luominen, iteraatioiden sisällön suunnittelu, järjestelmän toteutus ja testaus	Iterointisuunnitelma, järjestelmän arkkitehtuuria tukevien käyttäjätarinoiden toteutus, hyväksymistestit
Tuotantovaihe	Järjestelmän toteutus ja testaus, suorituskyvyn tarkistaminen, muutosten tekeminen	Tuotantovalmis järjestelmä
Ylläpito- ja lopetusvaihe	Dokumentointi, toimitetun järjestelmän tukeminen	Dokumentaatio, järjestelmäpäivitykset

Käyttäjätarinat laaditaan tavalliselle kortistokortilla luonnollisella kielellä. Käyttämällä yksinkertaista kuvaustekniikkaa mahdollistetaan se, että käyttäjillä on alhainen oppimiskynnys kuvauksen tekemiseen ja he voivat näin ollen ottaa vastuun käyttäjätarinoiden laatimisesta. Vaatimusten ei tarvitse olla täydellisiä,

koska käyttäjätarinoiden oletetaan toimivan vain kommunikaation apuvälineenä asiakkaan ja ohjelmoijan välillä.

Käyttäjätarinat ovat keskeisessä roolissa kartoitusvaiheessa, kun tehdään vaatimusmäärittelyä. Suunnitteluvaiheessa näitä vaatimuksia analysoidaan jakamalla tarinat tehtäviin. Iterointivaiheessa käyttäjätarinat toimivat ohjelmoijien ja asiakkaiden välisen kommunikaation tukena. Asiakkaat laativat hyväksymistestit käyttäjätarinoiden pohjalta. Kun käyttäjätarina on toteutettu ja läpäissyt hyväksymistestit, voidaan se hävittää. On siis selvää, ettei niistä alun perinkään haluta tehdä kattavia ja laajoja. Käyttäjätarinoissa on havaittavissa samoja piirteitä kuin UP:ssa käytettävissä käyttötapauksissa. Molemmat ovat tärkeä osa prosessia vaatimusmäärittelystä testaukseen.

Ennen käyttäjätarinoiden luomista voidaan järjestelmän haluttua toiminnallisuutta käsitteellistää metaforien avulla. Metafora tarjoaa asiakkaille, jotka eivät kykene käsitteellistämään järjestelmää, mahdollisuuden kommunikoida vapaamuotoisesti kehittäjien kanssa. Beckin (1999, 78) mukaan metafora pitää sisällään osan järjestelmän arkkitehtuurista. Suunnitteluvaiheen alussa arkkitehtuuri täydentyy, kun valitaan tärkeimmät käyttäjätarinat. Metafora on kuitenkin poistunut XP:n uusimpien käytänteiden joukosta. Syynä tähän lienee se, että se on koettu vaikeaksi määritellä, ymmärtää ja soveltaa käytännössä (Marchesi 2006; Rumpe & Schröder 2002).

XP:ssä ei ilmene muita tuotoksia käyttäjätarinoiden lisäksi, joissa käytettäisiin jotain mallintamistekniikkaa. Iterointivaiheessa hyödynnetään kuitenkin joitain mallintamistekniikoita, jotka esitellään seuraavaksi. Tuotetut mallit eivät ole kuitenkaan pysyviä ja eivät näin ollen liity suoranaisesti mihinkään tuotokseen. Tyypillistä iterointivaiheessa tapahtuvalle suunnittelulle on se, että se on pienimuotoista, jatkuvaa, inkrementaalista ja tiiviisti ohjelmoinnin kanssa rinnakkain etenevää. Etukäteissuunnittelua pyritään välttämään.

CRC-kortit esiteltiin alun perin (Beck & Cunningham 1989) tekniikkana, jonka avulla opetettiin oliosuuntautuneisuuden käsitettä. Ne ovat kuitenkin vakiinnuttaneet asemansa myös mallintamistekniikkana. *CRC-kortti* on kortistokortti, jolle kirjataan luokkien vastuut ja yhteistyöt (Cunningham & Cunningham, Inc. 2005). CRC-kortteja käytetään XP:ssä usein iterointivaiheen aikaisessa suunnittelussa.

UML:ää ei lainkaan ohjeisteta käytettäväksi XP:n kanssa (Beck 1999). Beck, Jeffries ja monet muut eivät suosittele UML:llä mallintamista. Kuitenkin niiden käyttöä suosittelevat Fowler ja Kendall (2004, 18, 47, 68) ja Astels ym. (2002, 142). XP:ssä käytetyimmät UML-kaaviot ovat luokka- ja sekvenssikaavio. Astelsin ym. (2002, 144) mielestä luokka- ja sekvenssikaaviot ovat hyödyllisiä, kun käytössä on sopiva työkalu, jonka avulla kaaviot voidaan kääntää vaivattomasti koodiksi. Fowler ja Kendall (2004, 18) näkevät luokka- ja sekvenssikaaviot tarpeellisiksi jo luotaessa yleismallia kohdealueesta. Luokkien välisiä vuorovaikutuksia tarkastellessa Fowler ja Kendall (2004, 66) suosivat kuitenkin käytännöllisyyden vuoksi ennemmin CRC-kortteja kuin sekvenssikaavioita.

Empiiriset tutkimukset osoittavat, että 35 prosentissa XP-projekteista käytettiin UML:ää (Rumpe & Schröder 2002). Tämä tarkoittaa sitä, että jonkinasteista kiinnostusta UML:n ja XP:n yhdistämiseen ilmenee. Toisaalta 53 prosenttia vastaajista ei halua hyödyntää lainkaan UML:ää XP-projekteissa (Rumpe & Schröder 2002).

Taulukkoon 11 on tiivistetty tärkeimmät XP:ssä käytettävät mallintamistekniikat. Tässä yhteydessä termillä *mallintamistekniikka* tarkoitetaan UML-kaaviota tai vastaavaa kuvaustekniikkaa. Taulukossa kuvataan mallien käyttötarkoitus ja XP:n prosessin vaihe, jolloin niitä käytetään. Koska UML:n käyttö mallintamistekniikkana XP:ssä jakaa mielipiteitä, on UML-kaaviot

merkitty taulukkoon tähdellä erottamiseksi yleisesti XP:ssä hyväksytyistä mallintamistekniikoista.

TAULUKKO 11. XP:n tärkeimmät mallintamistekniikat

Mallintamistekniikka	Käyttötarkoitus	Vaihe
Käyttäjätarinat (Beck 1999)	Toiminnallisten vaatimusten kuvaaminen	Kartoitus ja suunnittelu
CRC-kortit (Beck 1999)	Olioiden vastuiden ja yhteistyön tarkastelu	Iterointi
* Luokkakaavio (Fowler & Kendall 2004)	Kohdealueen mallintaminen, ohjelmiston rajapintojen määrittäminen, toteutuksen yksityiskohdat	Kartoitus ja iterointi
* Sekvenssikaavio (Fowler & Kendall 2004)	Olioiden yhteistyön esittäminen	Iterointi

* UML:ssä määritelty mallintamistekniikka

4.3 Feature Driven Development (FDD)

Feature Driven Development (FDD) -menetelmä esiteltiin ensimmäistä kertaa Peter Coadin, Eric Lefebvren ja Jeff De Lucan kirjassa Java Modeling in Color with UML vuonna 1999. Sitä oli kuitenkin sovellettu jo aikaisemmin henkilömääriltään suurissakin projekteissa. (CMP Media LLC 2005) Tässä suhteessa FDD poikkeaa lähtökohtaisesti useista muista ketterissä menetelmistä hyvän skaalautuvuutensa ansiosta tuoden näin erilaisen näkökulman. Tämä kohta perustuu pääosin Coadin ym. (1999) sekä Palmerin ja Felsingin (2002) näkemyksiin.

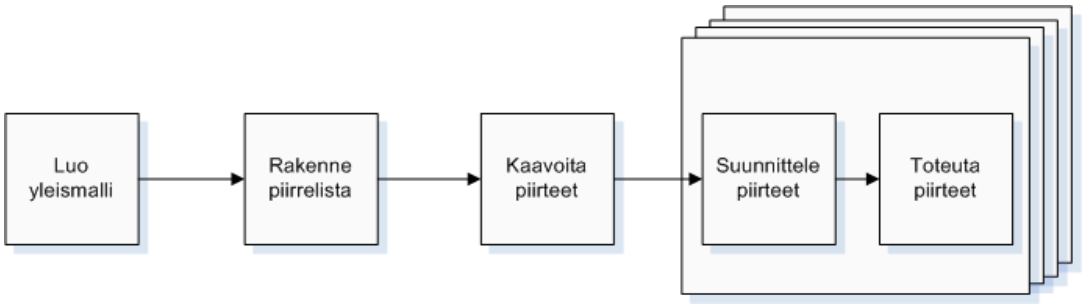
Lähtökohtainen ongelma nykyaikaiselle ohjelmistotuotantoprosessille on entistä nopeammin muuttuvan liiketoiminnan asettamat vaatimukset. FDD

pyrkii ratkaisemaan tämän ongelman piirteiden (features) avulla. (Coad ym. 1999, 183–184) Vaatimusmäärittelyssä on tarkoitus listata sellaiset toiminnalliset vaatimukset, jotka tuottavat arvoa asiakkaalle tai käyttäjälle. Näitä vaatimuksia kutsutaan arvoa tuottaviksi toiminnoiksi (client-valued functions) tai piirteiksi. (Palmer & Felsing 2002, 39) *Piirre* on pieni, asiakkaalle arvoa tuottava toiminto, joka ilmaistaan muodossa <toiminto> <kohde> <tavoite> (<action> <result> <object>). Esimerkkinä piirteestä on Laske summa myynnistä (Calculate the total of a sale). Piirteiden avulla voidaan tunnistaa toteutettavat luokat ja operaatiot. Piirteet ovat tarpeeksi pieniä, jos ne voidaan toteuttaa kahden viikon sisällä. Näin asiakkaat pystyvät toteamaan säännöllisesti mitattavan edistyksen. (Palmer & Felsing 2002, 41–42)

FDD:ssä on kahdeksan sille ominaista käytännettä: kohdealueen mallintaminen, piirteiden kehittäminen, henkilökohtainen luokkaomistajuus, piirretiimit, tarkastukset, säännöllisesti päivitettävät versiot (regular builds), konfiguroinnin hallinta ja raportointi/tulosten näkyvyys (Palmer & Felsing 2002, 36). Seuraavaksi tarkennetaan niitä käytänteitä, jotka eivät ole niin itsestään selviä. FDD:ssä on käytössä henkilökohtainen luokkaomistajuus toisin kuin esimerkiksi XP:ssä. Luokkaomistajuuden avulla määritetään, kenen vastuulla luokan sisältö on. Piirretiimi koostuu tiimistä vastuussa olevasta pääohjelmoijasta sekä niistä ohjelmoijista, joiden vastuunalaisia luokkia tarvitaan kyseisen piirteen toteuttamiseksi. Ohjelmoijat voivat työskennellä useammassa piirretiimissä samanaikaisesti. Yhden piirretiimin koko on yleensä kolmesta kuuteen henkilöä (Palmer & Felsing 2002, 48). Tarkastuksilla tarkoitetaan arviointitilaisuuksia, joissa käydään läpi suunnittelussa tai toteutuksessa saavutettuja ratkaisuja.

FDD jakautuu kahteen vaiheeseen: käynnistys- ja rakennusvaiheeseen. Vaiheilla ei itsessään ole suurempaa merkitystä. Sen sijaan menetelmässä on käytössä viisi prosessia, jotka käytännössä ohjaavat toimintaa. Prosessit ovat: luo

yleismalli, rakenna piirrelista, kaavoita piirteet (plan by feature), suunnittele piirteet (design by feature) ja toteuta piirteet (build by feature) (Coad ym. 1999, 190). Kolme ensimmäistä prosessia ovat ainutkertaisia ja muodostavat yhdessä projektin käynnistysvaiheen. Käynnistysvaiheen jälkeen siirrytään iteratiiviseen rakennusvaiheeseen, joka koostuu kahdesta jälkimmäisestä prosessista. (Nebulon Pty. Ltd. 2004) Yksi iteraatio kestää muutamasta päivästä maksimissaan kahteen viikkoon (Abrahamsson ym. 2002, 49). Prosessien tehtävät on jaettu joko pakollisina tai vaihtoehtoisina suoritettaviksi projektin eri vastuuhenkilöille. Kuviossa 7 on kuvattu Abrahamsson ym. (2002, 48) mukaan Palmerin ja Felsingin (2002) esityksen pohjalta FDD:n prosessit.



KUVIO 7. FDD:n prosessit (Abrahamsson ym. 2002, 48)

Seuraavaksi esitellään tarkemmin edellä mainitut viisi prosessia Coadin ym. (1999) mukaan. Esitystä täydennetään Palmerin ja Felsingin (2002) näkemyksillä.

Ensimmäisen prosessin, luo yleismalli, tavoitteena on muodostaa korkean tason kuvaus kohdealueesta. Aluksi muodostetaan mallinnustiimi, joka koostuu kehittäjistä ja kohdealueen asiantuntijoista. Mallinnustiimi työskentelee pääarkkitehdin johdolla. Kohdealueen asiantuntija voi olla rooliltaan käyttäjä, asiakas, sponsori, liiketoiminta-analyytikko tai näiden yhdistelmä. Hänellä on tietämys järjestelmän vaatimuksista. Asiantuntijat siirtävät tietämystään kehittäjille ja varmistavat näin, että kehittäjät toimittavat vaatimukset täyttävän

järjestelmän. Asiantuntijan kautta saadun tietämyksen lisäksi lähtökohtana toimivat olemassa olevat dokumentit. Mallinnustiimi jaetaan korkeintaan kolmeen pienryhmään (Nebulon Pty. Ltd. 2004). Ryhmät luovat oman yleismallin kohdealueesta, jonka jälkeen ne esitellään kaikille. Vaihtoehdoista valitaan pohjaksi yksi malli, jota täydennetään toisilla malleilla. Mallinnuksen yhteydessä kirjataan ylös vaihtoehtoiset mallinnusvaihtoehdot, joiden käyttöä harkittiin. (Coad ym. 1999, 191)

Yleismallin luomisen jälkeen rakennetaan piirrelista. Tässäkin prosessissa kehittäjät ja kohdealueen asiantuntijat työskentelevät yhdessä vielä joissain määrin. Edellisen prosessin yhteydessä oli hahmoteltu toteutettavia piirteitä, mutta tässä vaiheessa niitä tarkastellaan lähemmin. (Coad ym. 1999, 192) Kohdealue jaetaan aihealueisiin (subject area) ja aihealueiden liiketoiminnan toimintoihin (business activity), jotka puolestaan voidaan jakaa askelmiksi (step) eli piirteiksi. Aihealueesta käytetään myös nimitystä pääpiirrejoukko (major feature set) ja liiketoiminnan toiminnoista nimitystä piirrejoukko (feature set) (Nebulon Pty. Ltd. 2007). Alkuperäisen mallin mukaan piirteet ja piirrejoukot priorisoitiin. Nykyään tätä ei enää tehdä, koska se nähdään liiallisena resurssien tuhlauksena. Sen sijaan seuraavassa prosessissa laadittavan piirteiden ja piirrejoukkojen toteutusjärjestyksen mielletään riittäväksi priorisoinnin tueksi sellaisenaan. (Nebulon Pty. Ltd. 2004) Jos tunnistetut piirteet ovat liian laajoja tai monimutkaisia, voivat kehittäjät jakaa ne pienemmiksi osiksi pääarkkitehdin johdolla. (Coad ym. 1999, 192)

Kolmannen prosessin tarkoituksena on kaavoittaa piirteiden suunnittelua ja toteutusta ja laatia tämän perusteella kehityssuunnitelma pohjaksi rakennusvaiheelle. Projektipäällikkö, kehitysjohtaja ja pääohjelmoijat laativat piirteiden toteutusjärjestyksen ja toteutusaikataulun niiden keskinäisten riippuvuuksien, kehitystiimien kuormitusten ja piirteiden monimutkaisuuden perusteella (Nebulon Pty. Ltd. 2004). Piirrejoukkojen vastuut jaetaan

pääohjelmoijien kesken. Piirteiden toteuttamiseksi tarvittavien luokkien vastuut jaetaan ohjelmoijille. (Coad ym. 1999, 193)

Neljännessä prosessissa suunnitellaan tarkemmin piirteiden toteutusta. Piirteiden yksityiskohtaista suunnittelua ja toteutusta varten muodostetaan piirretiimejä. Jokaista piirrettä varten luodaan suunnittelupaketti (design package), joka sisältää kuvauksen piirteestä, mahdollisen tukimateriaalin sekä suunnittelumallit (Nebulon Pty. Ltd. 2004). Suunnittelumallit käydään lopussa läpi katselmoinneissa. Tarvittaessa pääohjelmoija voi kutsua ulkopuolisia jäseniä osallistumaan tulosten arviointiin. Tarkastuksessa käsitellyt kohdat kirjataan ylös luokkakohtaisesti ohjelmoijia varten. (Coad ym. 1999, 194; 196)

Viimeisen prosessin tarkoituksena on toteuttaa ja testata piirteet. Luokkaomistajat toteuttavat piirrettä tukevat metodit ja lisäävät testimetodit. Piirretiimi tarkastaa ohjelmakoodin. Tarkastuksen tulokset kirjataan jälleen luokkakohtaisesti ylös ohjelmoijia varten. Luokkaomistajat testaavat koodinsa ja varmistavat, että koodi täyttää piirteen asettamat vaatimukset. Pääohjelmoija vastaa piirteen integroinnista ja sen kokonaisvaltaisesta testaamisesta. Kun on varmistettu, että piirteen luokat toimivat yhdessä, on piirre valmis liitettäväksi seuraavaan julkaisuun. (Coad ym. 1999, 195)

Rakennusvaiheessa on käytössä kuusi erillistä virstanpylvästä, jotka kuvaavat, missä vaiheessa kunkin piirteen kehitys on (Williams 2004, 16). Nämä vaiheet ovat kohdealueen läpikäynti, suunnittelu, suunnittelun tarkastus, koodaus, koodin tarkastus ja piirteen julkaisu. Piirteen kehityksen edetessä pääohjelmoija päivittää piirteen tilan piirrelistaan.

Taulukkoon 12 on koottu FDD:n vaiheet, tavoitteet ja tuotokset. FDD:n prosessi jakautuu siis selkeästi kahteen vaiheeseen. Käynnistysvaiheen aikana kartoitetaan kohdealuetta muodostamalla yleismalli ja kerätään vaatimukset

piirrelistan avulla. Sen päätteeksi muodostetaan kehityssuunnitelma rakennusvaiheelle. Rakennusvaihe jaetaan iteratiivisiin suunnittelu- ja toteutusprosessiin.

TAULUKKO 12. FDD:n prosessin vaiheet, tavoitteet ja tuotokset

Vaihe	Tavoite	Tuotokset
Käynnistysvaihe: Luo yleismalli	Korkean tason kuvauksen muodostaminen kohdealueesta	Kohdealueen yleismalli
Käynnistysvaihe: Rakenna piirrelista	Piirteiden tunnistaminen ja listaaminen	Piirrelista
Käynnistysvaihe: Kaavoita piirteet	Piirteiden toteutusjärjestysten ja aikataulujen laatiminen, toteutusvastuiden jakaminen	Kehityssuunnitelma
Rakennusvaihe: Suunnittele piirteet	Piirteiden suunnittelu	Suunnittelupaketit
Rakennusvaihe: Toteuta piirteet	Piirteiden toteutus ja testaus	Julkaisuvalmiit piirteet

Seuraavaksi tarkastellaan tarkemmin FDD:ssä suoritettavaa mallintamista prosessin vaiheiden, työnkulkujen ja niiden aikana syntyvien tuotosten avulla.

Ensimmäisen prosessin aikana luodaan yleismalli kohdealueesta. Kohdealueen yleismalli on korkean tason kuvaus, jonka kuvaamisessa käytetään UML:n luokkakaavioita. Mallin tarkoituksena on luoda edellytykset tarkemmalle vaatimusmäärittelylle rajaamalla ja tunnistamalla kohdealueen olennaiset kohteet ja niiden väliset yhteydet. Prosessin aikana jokainen mallinnustiimin pienryhmä muodostaa luokkakaavion kohdealueesta keskittyen aluksi luokkiin ja yhteyksiin, myöhemmin metodeihin ja lopulta attribuutteihin (Coad ym. 1999, 191). Kohdealueen mallintamisessa suositellaan käytettäväksi

värimallinnusta (”modeling in color”). Tämän tekniikan avulla voidaan rakentaa nopeasti joustava ja laajennettava oliomalli. Värimallinnus ei kuitenkaan ole pakollinen ominaisuus FDD:ssä, joskin sen käyttö on suositeltavaa. (Palmer & Felsing 2002, 36–38) Luokkakaavioiden lisäksi luodaan tarvittaessa yksi tai useampi vapaamuotoinen ja korkean tason sekvenssikaavio käyttäytymisen mallintamiseksi. Pienryhmissä kehitetyistä malleista valitaan sopivin ja täydennetään muiden pienryhmien luomilla malleilla. Yleismalli ja vapaamuotoinen sekvenssikaavio säilytetään ja niitä päivitetään jatkossa tarvittaessa. Malleja tarkennetaan tarvittaessa erilaisin selittein. (Coad ym. 1999, 191)

Piirrelista muodostetaan FDD:n toisen prosessin aikana. Se on kategorisoitu aihealueiden ja niissä tapahtuvien liiketoimintojen mukaan. Jokaisen liiketoiminnan askelman toteuttamiseksi on tunnistettu ja kirjattu ylös vastaava piirre. Vaikka piirrelista on yksinkertaisesti ilmaistuna vain kokoelma piirteitä, sisältää se järjestelmään kohdistuvat toiminnalliset vaatimukset. Ideaalitilanteessa piirteestä voidaan suoraan muodostaa sekvenssikaavio. Piirteen toiminnosta muodostuu metodin nimi, tavoitteesta paluuarvo ja kohteesta luokka, johon metodi sijoitetaan (Miller 2003).

Neljännessä prosessissa muodostetaan suunnittelupaketteja, jotka sisältävät luokka- ja sekvenssikaavioita. Prosessin aikana piirretiimi rakentaa virallisen ja yksityiskohtaisen sekvenssikaavion piirteestä. Suunnitteluvaihtoehdot, päätökset, olettamukset ja huomautukset kirjataan ylös. Pääohjelmoija lisää sekvenssikaavion ja sitä vastaavat luokkakaavion päivitykset projektimalliin. Tämän jälkeen jokainen luokkaomistaja päivittää luokkansa ja metodien esittelyt sekvenssikaavioon. Tämä tarkoittaa parametrityyppien, palautustyyppien, poikkeuksien ja viestien lähetysten määrittämistä. (Coad ym. 1999, 194)

Taulukossa 13 esitetään tärkeimmät FDD:ssä käytettävät mallintamistekniikat. Taulukossa kuvataan tekniikoiden käyttötarkoitus ja vaiheet eli FDD:n prosessit, joissa niitä käytetään.

TAULUKKO 13. FDD:n tärkeimmät mallintamistekniikat

Mallintamistekniikka	Käyttötarkoitus	Prosessi
Luokkakaavio	Yleismalli sovellusalasta, yksityiskohtainen suunnittelu	Luo yleismalli, suunnittele piirteet
Sekvenssikaavio	Käyttäytymisen kuvaaminen, yksityiskohtainen suunnittelu	Luo yleismalli, suunnittele piirteet
Piirrelista (ei kuulu UML:ään)	Toiminnallisten vaatimusten listaaminen	Rakenna piirrelista

4.4 Dynamic Systems Development Method (DSDM)

DSDM (Dynamic Systems Development Method) on saanut alkunsa vuonna 1994 Iso-Britanniassa perustetun yhteisön, DSDM Consortiumin, muodostuessa. Tämä liiketaloudellista voittoa tavoittelemattoman yhteisö koostuu erilaisista informaatioteknologian ammattilaisista. DSDM on koko ohjelmistoelinkaaren kattava viitekehys, joka pohjautuu lähtökohtaisesti RAD:iin (Blue Ink 2006) sekä yhteisön jäsenten kautta esille tuomiin parhaisiin käytänteisiin. (Stapleton 1997, xiii-xv) DSDM:n pääajatuksena on, että projektien epäonnistumiset johtuvat pääasiassa ihmisiin liittyvistä ongelmista eivätkä niinkään teknologiasta. DSDM pyrkii tarjoamaan viitekehyksen, joka pyrkii estämään nämä epäonnistumiset. Seuraava esitys pohjautuu pääasiassa Stapletonin (1997) teokseen ja DSDM-manuaaliin (DSDM Consortium 2007).

DSDM pohjautuu yhdeksään periaatteeseen (Stapleton 1997, xvi):

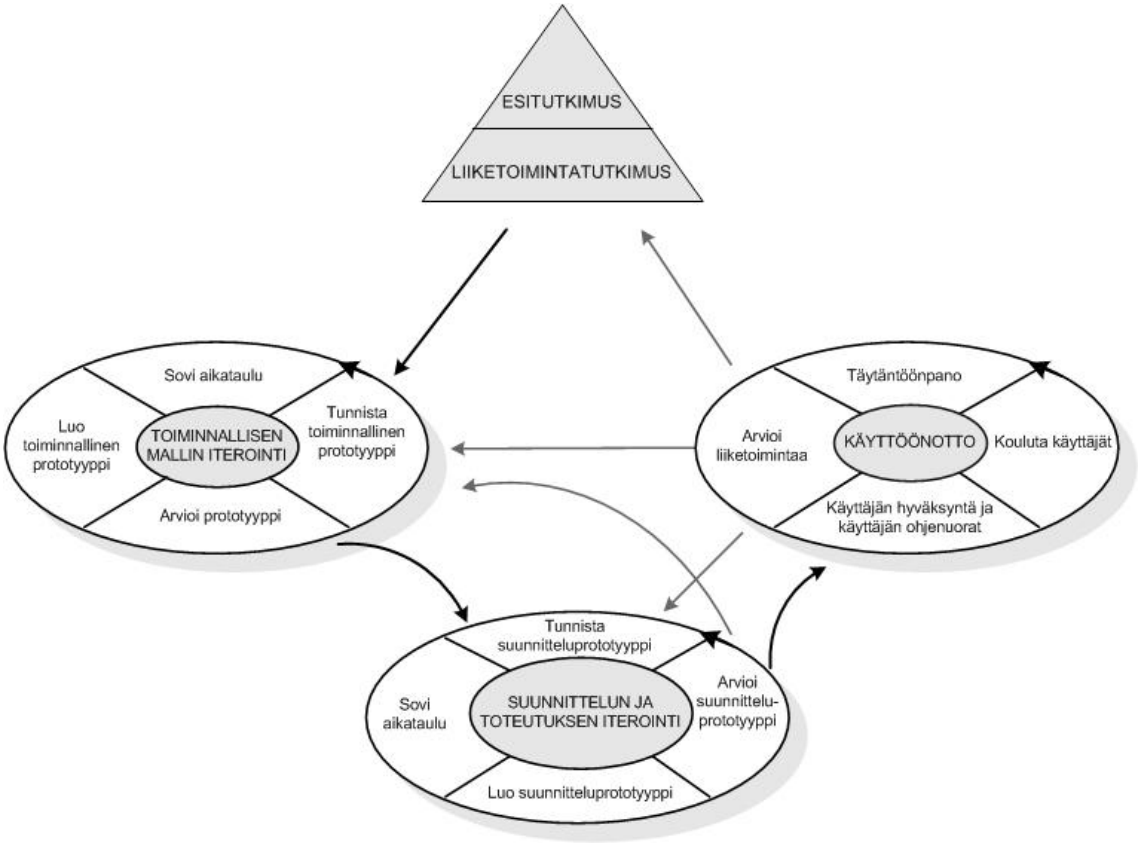
- Käyttäjän aktiivinen osallistuminen on välttämätöntä.

- DSDM-tiimeille täytyy antaa päätösvaltaa.
- Tavoitteena on jatkuva tuotteiden toimitus.
- Soveltuvuus liiketoimintatarkoitukseen on välttämätön kriteeri tuotosten hyväksymiselle.
- Iteratiivinen ja inkrementaalinen kehitys on välttämätöntä virheettömän liiketoimintaratkaisun saavuttamiseksi.
- Kaikki kehityksen aikana tehdyt muutokset ovat peruttavissa.
- Vaatimukset kiinnitetään lähtökohtaisesti korkealla tasolla.
- Testausta suoritetaan läpi elinkaaren.
- Yhteistyö ja yhteistoiminta kaikkien sidosryhmien välillä on välttämätöntä.

Kuten periaatteista voidaan havaita, ne ovat pitkälti rinnastettavissa myös muihin ketteriin menetelmiin. Vaikka ketterä lähestymistapa yleisesti hyväksyy vaatimusten muutokset ohjelmistokehityksen myöhäisessäkin vaiheessa, DSDM pohjaa toimintansa lähtökohtaisten vaatimusten kiinnittämiseen. Tämä on kuitenkin perusteltavissa sillä, että nämä vaatimukset ovat elintärkeitä asiakkaan liiketoiminnalle. Näin ollen kyseisten vaatimusten muuttaminen kesken kehitystä olisi käytännössä mahdotonta.

DSDM:n prosessi jakautuu viiteen vaiheeseen: esitutkimus, liiketoimintatutkimus (business study), toiminnallisen mallin iterointi, suunnittelun ja toteutuksen iterointi, käyttöönotto. Näistä vaiheista kaksi ensimmäistä ovat peräkkäisiä ja kertaluonteisia, kun taas loput ovat iteratiivisia ja inkrementaalisia. (Stapleton 1997, 4) Eräissä lähteissä, kuten DSDM-manuaalissa (DSDM Consortium 2007), nimetään edellä mainittujen vaiheiden lisäksi kaksi muuta vaihetta, esiprojekti- ja jälkiprojektivaiheet. Ne ovat pääosin hallinnollisia vaiheita eikä niitä tämän tutkielman yhteydessä lähemmin tarkastella. Kuviossa 8 on esitetty DSDM-projektin elinkaari. Tummemmat nuolet kuvaavat siirtymistä prosessin vaiheesta toiseen ja vaaleammat nuolet osoittavat mahdolliset paluusiirtymät aikaisempiin vaiheisiin.

Esitetty elinkaari vaiheineen on tarkoitettu sovellettavaksi projektikohtaisesti, ja se on näin ollen viitekehys (DSDM Consortium 2007). Seuraavaksi käydään läpi vaiheiden sisällöt Stapletonin (1997) esitykseen pohjautuen täydentäen DSDM-manuaali versio 4.2:n (DSDM Consortium 2007) ohjeistuksella etenkin vaiheiden tuotoksien osalta.



KUVIO 8. DSDM:n elinkaari (Stapleton 1997, 3)

Esitutkimuksen tavoitteena on ennen kaikkea päättää, onko DSDM sopiva käytettäväksi kyseisessä projektissa. Päätökseen vaikuttaa muun muassa projektin luonne sekä organisaatio ja ihmiset. Tämän lisäksi käsitellään myös perinteisempiä kysymyksiä, kuten teknisiä ratkaisuja ja mahdollisia riskejä. Vaiheen aikana syntyy kolme tuotosta: esitutkimusraportti, alustava kehityssuunnitelma ja riskilogi (DSDM Consortium 2007). Tarvittaessa voidaan

muodostaa pikainen prototyyppi. Esitutkimuksen tulisi kestää korkeintaan pari viikkoa. (Stapleton 1997, 5)

Liiketoimintatutkimuksen aikana analysoidaan liiketoiminnan ja teknologian näkökulmasta tärkeimmät asiat. Vaiheen aikana järjestetään työpajoja, joiden tarkoituksena on kartoittaa liiketoimintaprosessit ja käyttäjäryhmät. Työpajojen tuloksena syntyy liiketoimintamääritys (business area definition). Liiketoimintamäärityksen pohjalta muodostetaan lista priorisoiduista vaatimuksista. Näiden lisäksi vaiheen tuotoksia ovat järjestelmäarkkitehtuurimääritys, alustava prototyyppisuunnitelma ja päivitetty riskilogi. Arkkitehtuurimääritys on aluksi vain luonnos, joka kehittyy projektin aikana. Prototyyppisuunnitelman tulisi esittää seuraavien vaiheiden protoilustrategia sekä konfiguroinnin hallintasuunnitelma. (Stapleton 1997, 6)

Toiminnallisen mallin iterointi on ensimmäinen iteratiivinen ja inkrementaalinen vaihe. Sen pääasiallisena tarkoituksena on toteuttaa järjestelmän tärkeimmät toiminnalliset vaatimukset. Iteraatioiden aikana luodaan rinnakkain analyysimalleja ja rakennetaan ohjelmistokomponentteja. Ohjelmistokomponentit kehitetään protoilemalla ja niiden tulisi toteuttaa järjestelmän päätoiminnot sekä joitain ei-toiminnallisia vaatimuksia, erityisesti käytettävyyteen liittyen. Prototyyppejä ei välttämättä heitetä pois, vaan ne voivat toimia osana lopullista järjestelmää. Näin ollen jatkuva testaus on tärkeää jo tässä vaiheessa. Vaiheen päätuotoksena syntyy toiminnallinen malli, joka sisältää toiminnalliset prototyypit sekä analyysimallit. Toiminnallisen mallin lisäksi vaiheen aikana muodostetaan kolme uutta tuotosta. Ensimmäinen niistä on toiminnallisten prototyyppien arviointidokumentit, jotka ovat käyttäjiltä kerättyjä kommentteja protyypeistä. Ne ohjaavat jatkossa tapahtuvaa kehitystyötä paljastaen nykyiseen protomalliin tehtäviä muutostarpeita. Toinen on lista ei-toiminnallisista vaatimuksista. Se sisältää vaatimukset, jotka ovat paljastuneet liiketoimintatutkimuksen ja toiminnallisen mallin iteroinnin

yhteydessä. Osa näistä vaatimuksista on käsitelty tässä vaiheessa, mutta suurin osa niistä jää käsiteltäväksi seuraavaan vaiheeseen. Viimeisenä tuotoksena laaditaan aikataulusuunnitelma, jossa määritetään iteraation sisältö kehityssuunnitelmaa yksityiskohtaisemmin (DSDM Consortium 2007). Lisäksi päivitetään riskianalyysiä ja priorisoitua vaatimuslistaa (DSDM Consortium 2007). Tässä vaiheessa riskeihin on vielä helpompi puuttua ennen kuin järjestelmää aletaan toimittaa oikeaan käyttöympäristöönsä. (Stapleton 1997, 7–8)

Suunnittelun ja toteutuksen iterointi on se vaihe, jossa varsinainen järjestelmän toteutus pääsiassa suoritetaan. Vaiheen alkuvaiheessa muodostettavia väliaikaisia tuotoksia ovat suunnitteluprototyypit ja niihin liittyvät arviointidokumentit. Suunnitteluprototyyppejä kehitetään evolutionaarisesti toiminnallisten prototyyppien tavoin. Vaiheen päätuotos on sovitut minimivaatimukset täyttävä, testattu järjestelmä. (Stapleton 1997, 8–9) Muita tuotoksia ovat testitilastot ja edellisen vaiheen tapaan laadittu aikataulusuunnitelma (DSDM Consortium 2007).

Käyttöönotto on viimeinen vaihe. Sen aikana järjestelmä siirretään kehitysympäristöstä käyttöympäristöön. Vaihe sisältää käyttäjien kouluttamisen ja järjestelmän toimituksen käyttäjille. Vaihe voi olla iteratiivinen, jos käyttäjäryhmät ovat hajautuneita. Toimitetun järjestelmän lisäksi vaiheen tuotoksia ovat käyttäjämanuaali ja projektin arviointiraportti. Manuaalin tekeminen voidaan aloittaa jo aikaisemmissa vaiheissa, mutta luovutetaan käyttöönoton yhteydessä. Arviointiraportti kokoaa projektissa saavutetut tulokset. Lisäksi siitä ilmenee kehityksen aikana havaitut vaatimukset ja luovutettavan järjestelmän tilan suhteessa näihin vaatimuksiin.

DSDM määrittää neljä mahdollista jatkokehityssuuntaa. Ensimmäisenä vaihtoehtona on ideaalitilanne, jossa kaikki vaatimukset on saavutettu eikä

jatkotoimenpiteitä tarvita. Toisaalta joskus voi ilmetä tilanne, jolloin kehityksen aikana havaittu liiketoiminnallisesti tärkeä toiminnallisuus joudutaan sivuuttamaan aikatavoitteen saavuttamiseksi. Tämä tarkoittaa sitä, että joudutaan palaamaan takaisin liiketoimintatutkimukseen ja etenemään uudestaan alusta asti. Kolmantena vaihtoehtona on, ettei kaikkia vähemmän tärkeitä vaatimuksia ehditty toteuttaa aikarajaan mennessä. Tämä johtaa paluuseen toiminnallisen mallin iterointiin ja jatkamaan prosessia siitä eteenpäin. Viimeisessä vaihtoehdossa joitakin teknisiä kysymyksiä on jouduttu sivuuttamaan suunnittelun ja toteutuksen yhteydessä aikarajan vuoksi. Ne liittyvät järjestelmän ei-toiminnallisiin ominaisuuksiin, esimerkiksi suorituskykyyn. Jos niiden korjaaminen nähdään oleellisiksi, voidaan palata takaisin järjestelmän suunnitteluun ja toteutukseen. (Stapleton 1997, 9–10)

Taulukkoon 14 on koottu DSDM:n vaiheet, tavoitteet ja tuotokset. Tuotoksien määrä on suuri verrattaessa muihin ketteriin menetelmiin. Osa tuotoksista on kuitenkin luotu jo aiemmassa vaiheessa ja sitä on vain päivitetty myöhemmin vaiheen aikana.

Seuraavaksi tarkastellaan lähemmin DSDM:n tuotoksia mallinnuksen näkökulmasta. Lähteenä ovat DSDM-manuaali (DSDM Consortium 2007) sekä tuotoksien dokumenttipohjat ja ohjeistukset (Surgeworks 2007). Mallintaminen nähdään tärkeänä osana järjestelmäkehitystä eri osapuolten välisen kommunikaation edistämiseksi. Mallit liitetään usein osaksi jotain vaiheen tuotosta. DSDM ei pyri rajaamaan käytettävien mallintamistekniikoiden määrää, vaan esittää laajan joukon vaihtoehtoisia tekniikoita käytettäväksi prosessin eri vaiheisiin. Tässä yhteydessä on tarkoituksena esitellä eri vaiheissa käytettävät yleisimmät ja oleellisimmat mallintamistekniikat.

Esitutkimuksen aikana muodostetaan esitutkimusraportti, jossa määritetään projektinhallintaan liittyviä tyypillisiä osia, kuten tavoitteet, aikataulut ja riskit.

Määrityksen tueksi muodostetaan erilaisia malleja, joiden päätarkoituksena on muodostaa kuvaus liikeyrityksestä ja sen avainkohteista sekä rajata kohdealuetta (DSDM Consortium 2007). Kontekstikaavion tarkoituksena on rajata kohdealuetta, ja se on yksi käytetyimmistä esitutkimusraporttiin liitettävistä kaavioista (Surgeworks 2007).

TAULUKKO 14. DSDM:n prosessin vaiheet, tavoitteet ja tuotokset

Vaihe	Tavoite	Tuotokset
Esitutkimus	DSDM:n soveltuvuuden varmentaminen kyseiseen projektiin, perinteisten esitutkimuskysymysten selvittäminen, riskien tunnistaminen	Esitutkimusraportti, alustava kehityssuunnitelma ja riskilogi
Liiketoimintatutkimus	Liiketoimintaprosessien ja käyttäjäluokkien kartoitus, teknisten rajoitteiden tunnistaminen	Liiketoimintamääritys, priorisoitu vaatimuslista, järjestelmäarkkitehtuurimääritys, alustava prototyyppisuunnitelma ja päivitetty riskilogi
Toiminnallisen mallin iterointi	Tärkeimpien toiminnallisten vaatimusten toteuttaminen	Toiminnallinen malli, toiminnallisten prototyyppien arviointidokumentit, lista ei-toiminnallisista vaatimuksista, aikataulusuunnitelma, päivitetty riskilogi ja päivitetty priorisoitu vaatimuslista
Suunnittelun ja toteutuksen iterointi	Järjestelmän toteutus ja testaus	Sovitut vaatimukset täyttävä testattu järjestelmä, testitilastot ja aikataulusuunnitelma
Käyttöönotto	Järjestelmän toimitus asiakkaalle, käyttäjien koulutus	Toimitettu järjestelmä, käyttäjämanuaali, projektin arviointidokumentti ja koulutetut käyttäjät

Liiketoimintatutkimuksen aikana pyritään muodostamaan korkean tason kuvaus järjestelmästä ja sen prosesseista. Lisäksi tunnistetaan ja kirjataan ylös

järjestelmän vaatimukset. Kuvaukset ja vaatimukset kirjataan osaksi liiketoimintamääritystä. Käytettäviä tekniikoita ovat muun muassa korkeantason ER-kaaviot ja tietovirtakaaviot, käyttötapauskaaviot sekä liiketoimintamallit. Vaatimusmäärittelyn osalta tässä vaiheessa keskitytään erityisesti toiminnallisiin vaatimuksiin. Määrittelyn tuloksena syntyy priorisoitu vaatimuslista, jonka muodostamisessa käytetään niin sanottuja MoSCoW-sääntöjä (MoSCoW Rules) (Highsmith & Cockburn 2001), joiden avulla vaatimukset luokitellaan. (DSDM Consortium 2007)

Järjestelmäarkkitehtuurimäärityksen tarkoituksena on kuvata järjestelmän toiminnallinen ja tekninen arkkitehtuuri. Arkkitehtuurin kuvaamisessa käytetään erilaisia teknisiä arkkitehtuurimalleja, jotka määrittävät laitteiston, tietovarastot, komponentit ja niiden väliset yhteydet. Ohjelmistoarkkitehtuurista on tarkoitus muodostaa alustava kuvaus, esimerkiksi tärkeimpien olioiden tai komponenttien avulla kuvaten prosesseja, tietoa ja niiden vuorovaikutuksia. Koska järjestelmäarkkitehtuurimääritys on erittäin tekninen tuotos, on siitä vastuussa tekninen koordinaattori apunaan muut kehittäjät.

Toiminnallista mallia muodostettaessa pyritään mallintamaan aiempaa tarkemmin järjestelmän vaatimuksia. Tietovirtakaaviot ja käyttötapauskaaviot ovat toisilleen vaihtoehtoisia kuvaustapoja demonstroimaan liiketoimintaprosessien osia (Surgeworks 2007). Lisäksi muodostetaan looginen tietomalli. Kehitysstandardien luomiseksi muodostetaan erilaisia näyttö-, menu ja navigointimalleja.

Suunnittelun ja toteutuksen iteroinnin tuotoksena syntyy testattu järjestelmä. Vaikkei tuotokseen liitetä mitään malleja, voidaan käyttää malleja apuna luotaessa suunnitteluprototyyppiä. Niiden avulla pyritään kuvaamaan

käytettävää teknologiaa ja järjestelmän fyysistä rakennetta. Esimerkkeinä mahdollisista malleista ovat fyysiset tieto- ja prosessimallit.

Käyttöönottovaiheessa toimitetun järjestelmän lisäksi muodostettavien tuotosten on tarkoitus tukea järjestelmän dokumentointia. Käyttäjäedustaja on vastuussa käyttäjämanuaalin luomisesta. Käyttäjien ja heidän tehtäväkuvaustensa lisäksi saattaa ilmetä tarvetta tekniselle dokumentaatiolle, kuten fyysisten komponenttien rakenteen kuvaaminen. Kehittäjät tuottavat tarvittaessa nämä tekniset kuvaukset.

Taulukkoon 15 on koottu yleisimmät ja käytetyimmät mallintamistekniikat, joita DSDM:ssä käytetään. Näiden lisäksi on lukuisia vaihtoehtoisia tekniikoita, joiden käytöstä ei kuitenkaan löytynyt riittävästi näyttöä. Mallintamistekniikoista näkyy se, että DSDM on kehitetty ennen kuin oliosuuntautunut järjestelmäkehitys yleistyi. Vielä tänäkin päivänä useat suositeltaviksi käytettävistä malleista pohjautuvat rakenteisten tietojärjestelmien kehittämiseen.

TAULUKKO 15. DSDM:n tärkeimmät mallintamistekniikat

Mallintamistekniikka	Käyttötarkoitus	Vaihe
Kontekstikaavio (ei kuulu UML:ään)	Kohdealueen ymmärtäminen ja rajaaminen	Esitutkimus
ER-kaavio (ei kuulu UML:ään)	Käsitteellisen tietomallin luominen	Liiketoimintatutkimus
Tietovirtakaavio (ei kuulu UML:ään)	Liiketoimintaprosessien esittäminen	Liiketoimintatutkimus, toiminnallisen mallin iterointi
Käyttötapauskaavio	Liiketoimintaprosessien esittäminen	Liiketoimintatutkimus, toiminnallisen mallin iterointi
Priorisoitu vaatimuslista (ei kuulu UML:ään)	Vaatimusten määrittäminen ja priorisointi	Liiketoimintatutkimus

4.5 Yhteenveto

Tässä luvussa esiteltiin kolme ketterää menetelmää: XP, FDD ja DSDM. Kunkin menetelmän osalta kuvattiin sille ominaisia näkemyksiä ja käytänteitä sekä prosessin vaiheet ja tuotokset. Sen jälkeen tarkasteltiin mallintamista ja sen yhteyttä prosessin vaiheisiin ja tuotoksiin. Tarkoituksena oli selvittää, mitä UML-kaavioita tai muita mallintamistekniikoita menetelmässä käytetään ja missä yhteydessä.

Agile Manifestin arvot ja periaatteet heijastuvat tarkasteltavissa menetelmissä, kuten asiakaslähtöisyys, viestinnän tärkeys ja toimiva ohjelmisto. XP:ssä on erikseen määritelty sille ominaiset arvot ja periaatteet. DSDM:ssä on myös määritelty omat periaatteet. XP:ssä ja FDD:ssä on lisäksi määritelty käytänteet, jotka ohjaavat konkreettisemmin prosessin kulkua.

Tarkastelussa ilmeni, että menetelmien prosessit olivat keskenään samankaltaisia. Prosessien ensimmäiset vaiheet ovat kertaluonteisia ja keskittyvät vaatimusten määrittämiseen ja projektin aikataulutukseen. Myöhemmät toteutusvaiheet ovat iteratiivisia. Vaiheissa syntyvien tuotosten määrä on XP:ssä ja FDD:ssä pieni, mutta DSDM:ssä se on jo selkeästi suurempi. Erilaisten mallintamistekniikoiden määrä on niin ikään pieni. XP:ssä toiminnalliset vaatimukset kuvataan käyttäjätarinoiden avulla. FDD:ssä vaatimukset ilmaistaan piirrelistana, ja DSDM:ssä käytetään priorisoitua vaatimuslistaa. XP:ssä suositellaan eräiden lähteiden mukaan käytettäväksi CRC-korttien lisäksi myös luokka- ja sekvenssikaaviota. FDD:ssä vastaavilla kaavioilla on tärkeä merkitys niin yleismallia luotaessa kuin yksityiskohtaisessa suunnittelussa. DSDM:ssä voidaan käyttää käyttötapauskaavioita liiketoimintaprosessien esittämiseksi. Vaihtoehtoisesti voidaan käyttää tietovirtakaaviota muiden rakenteisen lähestymistavan mukaisten tekniikoiden lisäksi.

5 VERTAILU JA ANALYYSI

Tässä luvussa vertaillaan ja analysoidaan edellä esiteltyjen ketterien menetelmien prosesseja ja mallintamistekniikoita. Ensin tarkastellaan menetelmiä niiden prosessien näkökulmasta UP:n (Jacobson, Booch & Rumbaugh 1999) avulla. Prosessien tarkastelussa kiinnitetään huomiota vaiheisiin, iteraatioihin ja työnkulkuihin. Tarkastelussa pyritään selvittämään se, millaisia prosessimalleja ketterissä menetelmissä käytetään ja miten ne vastaavat UP-prosessia. Lisäksi pyritään selvittämään menetelmien prosessien välisiä eroja. Tämän jälkeen vertaillaan menetelmissä tapahtuvaa mallintamista ja mallintamistekniikoita sekä sitä, millainen rooli UML:llä (Booch, Rumbaugh & Jacobson 1999) on mallintamisessa. Havaittuja yhtäläisyyksiä ja eroja pyritään selittämään menetelmien arvojen, periaatteiden ja käytänteiden avulla. Kolmanneksi ketteriä menetelmiä tarkastellaan kirjallisuuden pohjalta. Menetelmien sisältämiä UML-mallintamistekniikkoja verrataan tutkimustuloksiin, jotka koskevat yleisesti UML-mallintamistekniikoiden käyttöä käytännössä. Tällä halutaan selvittää, missä määrin tekniikoiden käyttö ketterissä menetelmissä vastaa yleistä käyttöjakaumaa. Valitut ketterät menetelmät edustavat ”puhdasoppista” lähestymistapaa. Kirjallisuudessa on löydettävissä lähestymistapoja, jotka on johdettu Agile Manifestin pohjalta mutta joissa on päädytty vähän erityyppisiin ratkaisuihin. Luvun viimeisessä kohdassa kuvataan näistä kahta, AM:ää (Agile Modeling) (Ambler 2007a) ja EssUP:ssa (Essential Unified Process) (Jacobson, Ng & Spence 2005) käytettävää Essential UML:ää (Jacobson, Ng & Spence 2005) ja vertaillaan niitä tässä työssä tarkasteltuihin menetelmiin erityisesti mallintamisen osalta.

5.1 Prosessit

Tässä kohdassa jäsennetään ketterien menetelmien prosesseja UP:n avulla. Ensin tarkastellaan vaiheiden sisältöä ja tavoitteita verraten niitä UP:hen ja

toisiinsa. Tämän jälkeen tarkastellaan vaiheiden iteratiivisuutta ja pituuksia. Lopuksi tarkastellaan, missä määrin UP:n työnkulut esiintyvät menetelmien eri vaiheissa.

UP:ssa aloitus- ja tarkentamisvaiheessa tarkastellaan rakennettavaa järjestelmää ja sen vaatimuksia koko elinkaaren tasolla. Tarkoituksena on vähentää riskejä ja mahdollistaa projektin aikataulujen ja kustannusarvioiden laatiminen. XP:ssä kartoitus- ja suunnitteluvaiheella on pitkälti sama tarkoitus, kuten myös FDD:ssä käynnistysvaiheella. DSDM:n kahden ensimmäisen vaiheen aikana valmistaudutaan vastaavasti iteratiivisiin protoiluvaiheisiin. DSDM:ssä arvioidaan esitutkimuksen aikana itse menetelmän soveltuvuutta kyseiseen ympäristöön ja projektiin, mikä tekee siitä erilaisen muihin menetelmiin verrattuna. UP:ssa ja FDD:ssä toteutus ja testaus painottuvat pääasiassa yhteen vaiheeseen. XP:ssä ja DSDM:ssä iteratiivisia toteutusvaiheita on useampia. UP:n luovutusvaihe ja DSDM:n käyttöönottovaihe ovat luonteeltaan samanlaisia. Molemmissa kiinnitetään dokumentoinnin lisäksi huomiota myös käyttäjien koulutukseen. XP:ssä ei korosteta järjestelmän luovutukseen liittyviä tehtäviä, mutta menetelmässä on määritelty luovutusta seuraava ylläpitovaihe. FDD:n prosessikuvaus päättyy siihen, kun piirteet on saatettu julkaisuvalmiiksi. Järjestelmän luovutusta ei ole kuvattu lainkaan.

UP:n kaikki vaiheet ovat luonteeltaan iteratiivisia, kun taas tarkasteltavissa ketterissä menetelmissä vain osa vaiheista on kuvattu iteratiivisiksi. Ketterissä menetelmissä ensimmäiset vaiheet ovat kertaluonteisia ja yleensä lyhytkestoisia suhteessa toteutusvaiheisiin. Tämä selittynee sillä, että ketterissä menetelmissä pyritään toimittamaan asiakkaalle toimivaa ohjelmistoa mahdollisimman nopeasti. Menetelmien välillä on kuitenkin eroja, sillä FDD:ssä pyritään määrittämään vaatimukset mahdollisimman kattavasti ennen toteutusta, kun taas XP:ssä ollaan avoimempia kehityksen aikana ilmeneville uusille vaatimuksille. FDD poikkeaakin tässä suhteessa merkittävästi muista ja sen

lähestymistapa on hyvin epätyypillinen ketterille menetelmille. Arkkitehtuurilla ei ole ketterissä menetelmissä yhtä suurta painoarvoa kuin UP:ssa, mikä vähentää etukäteissuunnittelun määrää ja aikaa. UP:ssa iteraatioiden kestot vaihtelevat järjestelmän koon mukaan tyypillisesti viikosta muutamaan kuukauteen. Ketterissä menetelmissä iteraatioiden pituudet pyritään pitämään mahdollisimman lyhyinä, keskimäärin muutaman viikon mittaisena. FDD:ssä mainitaan iteraation minimipituuden olevan ainoastaan muutaman päivän rakennusvaiheessa. DSDM:n käyttöönottovaihe voi olla iteratiivinen. Iteratiivisuuden tarve kasvaa, jos järjestelmä on monimutkainen ja laaja tai käyttäjäryhmät ovat hajautuneita.

Taulukoissa 16, 17, 18 ja 19 on eritelty yhteenvedon omaisesti UP:n, XP:n, FDD:n ja DSDM:n vaiheet, vaiheiden päätavoitteet, kestot ja tiedot vaiheiden iteratiivisuudesta.

Seuraavaksi tarkastellaan, missä ketterien menetelmien vaiheessa ja millä laajuudella UP:ssa esitetyt työnkulut esiintyvät. Tämän havainnollistamiseksi on seuraavassa esitetty jokaisen tarkasteltavan ketterän menetelmän osalta kuvio, joka on jaettu vaakasuunnassa prosessin vaiheisiin ja katkoviivoin mahdollisiin iteraatioihin. Pystysuunnassa näkyvät UP:n työnkulut. Prosessien läpi leikkaava alue on esimerkki siitä, millä painotuksella UP:n mukaiset työnkulut voisivat esiintyä prosessin eri vaiheessa. Esitetyt työnkulkujen painotukset ovat arvioita ja voivat vaihdella huomattavastikin projektista riippuen. Arviot perustuvat edellä esitettyihin vaiheiden kuvauksiin, niiden tavoitteisiin ja tuotoksiin. Piirrosteknisistä syistä vaiheiden kestot eivät vastaa niitä vastaavien sarakkeiden leveyksiä.

TAULUKKO 16. UP:n vaiheet ja niiden ominaispiirteet

	UP			
Vaihe	Aloitusvaihe	Tarkentamis-vaihe	Rakennusvaihe	Luovutusvaihe
Päätavoite	Liiketoiminnan ja tärkeimpien vaatimusten kartoitus	Alustavan arkkitehtuurin ja vaatimusten määrittäminen	Järjestelmän toteutus ja testaus	Järjestelmän siirtäminen käyttöympäristöön, tukimateriaalin tuottaminen
Vaiheen kesto	muutama päivä – useita kuukausia; 1 vko – 3 kk / iteraatio	1 vko – 3 kk / iteraatio	1 vko – 3 kk / iteraatio	1 vko – 3 kk / iteraatio
Iteratiivisuus	Kyllä	Kyllä	Kyllä	Kyllä

TAULUKKO 17. XP:n vaiheet ja niiden ominaispiirteet

	XP				
Vaihe	Kartoitus	Suunnittelu	Iterointi	Tuotanto	Ylläpito ja lopetus
Päätavoite	Vaatimusten kerääminen	Projektin aikataulut-taminen	Järjestelmän pääsääntöinen toteutus ja testaus	Järjestelmän tuotantoval-miuteen saattaminen	Järjestelmän tukeminen
Vaiheen kesto	2 vko – 2 kk	muutama päivä	1 vko – 4 vko / iteraatio	1 vko – 3 vko / iteraatio	Mahdollisesti useita vuosia
Iteratiivisuus	Ei	Ei	Kyllä	Kyllä	Kyllä

Kuviossa 9 on esitetty XP:n vaiheissa esiintyvät UP:n mukaiset työnkulut. Kartoitus- ja suunnitteluvaiheissa esiintyvät piikit suunnittelussa ja toteutuksessa kuvaavat XP:lle ominaisia arkkitehtuuripiikkejä (spikes). Piikkien

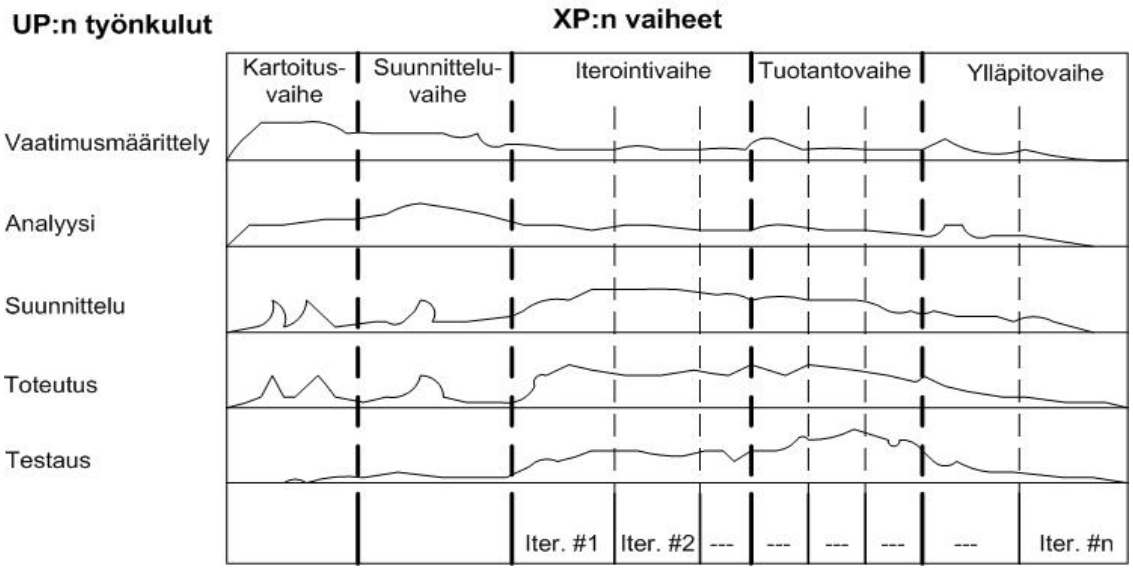
tavoitteena on löytää vastaus teknisiin tai suunnitteluongelmiin luotettavampien käyttäjätarinoiden arvioiden tuottamiseksi (Wells 2004).

TAULUKKO 18. FDD:n vaiheet ja niiden ominaispiirteet

	FDD	
Vaihe	Käynnistys	Rakennus
Päätavoite	Kohdealueen kartoitus, vaatimusten määrittäminen ja projektin aikataulutus	Järjestelmän yksityiskohtainen suunnittelu, toteutus ja testaaminen
Vaiheen kesto	–	muutama päivä – 2vko / iteraatio
Iteratiivisuus	Ei	Kyllä

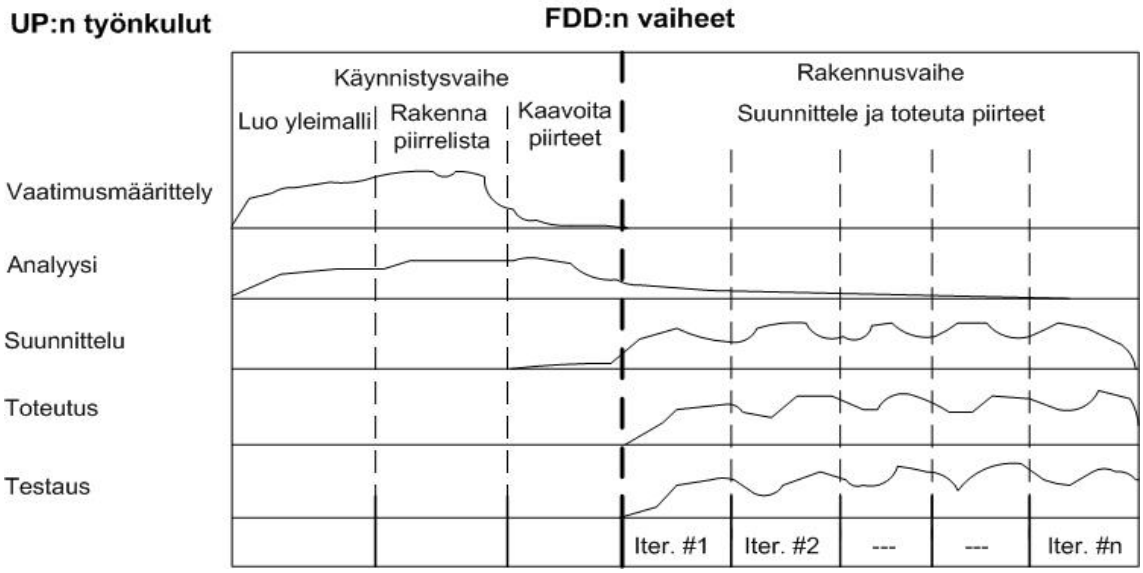
TAULUKKO 19. DSDM:n vaiheet ja niiden ominaispiirteet

	DSDM				
Vaihe	Esitutkimus	Liiketoimintatutkimus	Toiminnallisen mallin iterointi	Suunnittelun ja toteutuksen iterointi	Käyttöönotto
Päätavoite	Projektin kriittisten lähtökohtien tarkastelu	Liiketoiminnan kartoitus, arkkitehtuurin ja vaatimusten määrittäminen	Tärkeimpien toiminnallisten vaatimusten toteutus ja testaus	Järjestelmän toteutus ja testaus	Järjestelmän siirtäminen käyttöympäristöön ja käyttäjien koulutus
Vaiheen kesto	Korkeintaan muutama viikko	–	–	–	–
Iteratiivisuus	Ei	Ei	Kyllä	Kyllä	Mahdollisesti



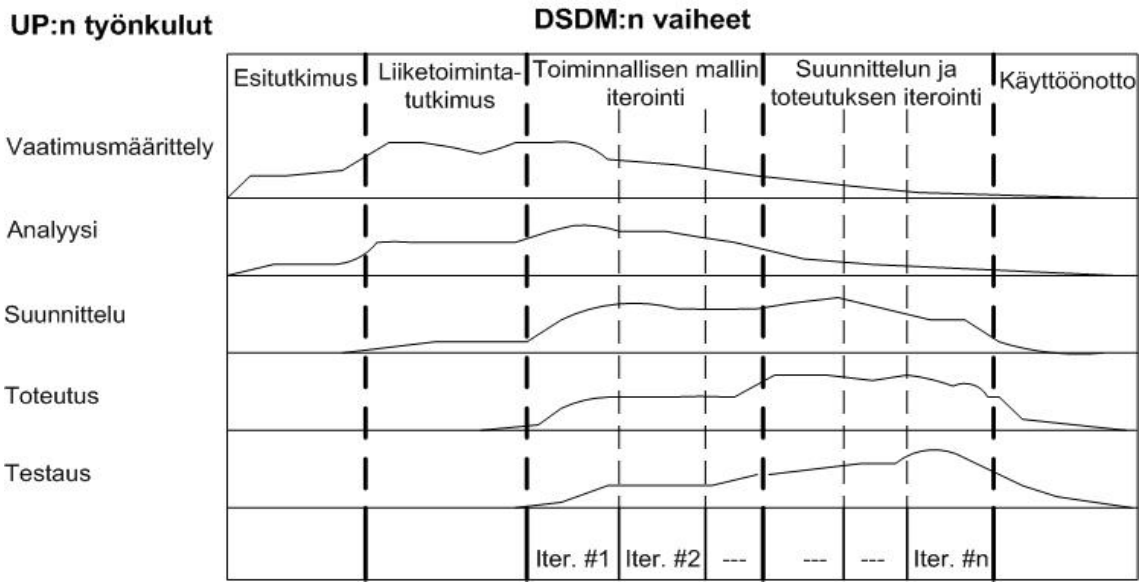
KUVIO 9 UP:n työnkulkujen esiintyminen XP:n vaiheissa

Kuviossa 10 on esitetty vastaavasti FDD:n vaiheet yhdessä UP:n työnkulkujen kanssa. Kuten kuviosta nähdään, FDD jakaantuu kahteen selkeästi erilaiseen vaiheeseen. Käynnistysvaiheessa suoritetaan vaatimusmäärittely ja analyysi. Rakennusvaiheessa puolestaan suunnitellaan, toteutetaan ja testataan. Tällaista lähestymistapaa voidaan pitää erittäin poikkeava ketterille menetelmille.



KUVIO 10 UP:n työnkulkujen esiintyminen FDD:n vaiheissa

Kuviossa 11 on esitetty DSDM:n vaiheissa esiintyvät UP:n mukaiset työnkulut. Työnkulut esiintyvät vaiheissa melko perinteisellä tavalla. Ensimmäisissä vaiheissa vaatimusmäärittelyn ja analyysin merkitys on suuri, ja jatkossa muiden työnkulkujen merkitys kasvaa. Kuvion 11 mukaisessa projektissa ei ole muodostettu esitutkimusvaiheessa tarvittaessa muodostettavaa pikaista prototyyppiä. Prototyypin luominen kasvattaisi suunnittelun ja toteutuksen painotusta jo esitutkimuksen aikana. Lisäksi käyttöönottovaihe voisi olla iteratiivinen, toisin kuin tässä esitetyssä tapauksessa.



KUVIO 11 UP:n työnkulkujen esiintyminen DSDM:n vaiheissa

Agile Manifestin mukaan asiakkaan tulisi voida muuttaa vaatimuksia projektin myöhemmissä vaiheissakin (Agile Alliance 2001). Tarkasteltavista menetelmistä XP on ainoa, jossa tämä periaate aidosti näyttää toteutuvan. Vaatimusmäärittely painottuu XP:ssä kartoitusvaiheeseen, jossa suurin osa käyttäjätarinoista laaditaan. Tässä vaiheessa ei kuitenkaan pyritä löytämään kaikkia vaatimuksia. Iterointi- ja tuotantovaiheen aikana ilmenee korjausvaatimuksia ja uusia vaatimuksia tulee esiin jatkuvan testauksen ja palautteen myötä. FDD:ssä vaatimusmäärittely painottuu käynnistysvaiheeseen. Toisin kuin XP:ssä,

FDD:ssä pyritään määrittämään vaatimukset mahdollisimman kattavasti ennen toteutukseen siirtymistä. DSDM:ssä vaatimusmäärittely on tärkeimmässä osassa liiketoimintatutkimuksen aikana, mutta jakaantuu myös muihin vaiheisiin. Esitutkimuksen aikana ilmenevät jo keskeisimmät vaatimukset, ja toiminnallisen mallin iteroinnissa toiminnalliset vaatimukset päivittyvät ja ei-toiminnallisia vaatimuksia kartoitetaan. DSDM on näistä menetelmistä ainoa, jossa keskitytään erityisesti ei-toiminnallisiin vaatimusten määrittämiseen.

Analyysi painottuu XP:ssä kartoitus- ja etenkin suunnitteluvaiheiseen, jolloin käyttäjätarinat jaetaan ohjelmointitehtäviksi. Iterointivaiheessa analyysia suoritetaan tarpeen mukaan, esimerkiksi päätettäessä, mitä luokkia tarvitaan käyttäjätarinoiden toteuttamisessa. FDD:ssä piirteiden listaamisen jälkeen muodostetaan toisiinsa liittyvistä piirteistä piirrejoukkoja. Kunkin piirteen osalta tunnistetaan Kaavoita piirteet -prosessissa toteutuksessa tarvittavat luokat. DSDM:ssä arkkitehtuuria aletaan muodostaa liiketoimintatutkimuksen aikana, mutta varsinainen analyysi painottuu toiminnallisen mallin iterointiin, jolloin luodaan analyysimalleja.

Suunnittelu XP:ssä tapahtuu rinnakkain ohjelmoinnin kanssa iteratiivisissa toteutusvaiheissa. FDD:ssä suunnittelu painottuu Rakennusvaiheen piirteiden suunnittelu -prosessiin, jolloin muodostetaan suunnittelupaketit. DSDM:ssä suunnittelu painottuu Toiminnallisen mallin iterointi - ja Suunnittelun ja toteutuksen iterointi -vaiheisiin. Edellä mainitussa vaiheessa pyritään kuvaamaan pääasiassa järjestelmän loogista rakennetta, kun taas jälkimmäisen vaiheen suunnittelussa keskitytään järjestelmän fyysisen rakenteen kuvaamiseen.

Toteutus painottuu XP:ssä pääasiassa iterointi- ja tuotantovaiheisiin sekä joissain määrin ylläpitovaiheeseen. FDD:ssä toteutus keskittyy voimakkaasti rakennusvaiheen Piirteiden toteutus -prosessiin. DSDM:ssä järjestelmän

toteutus tapahtuu toiminnallisen mallin iteroinnissa ja etenkin suunnittelun ja toteutuksen iteroinnissa.

Testaus on poikkeuksellisessa asemassa XP:ssä verrattuna muihin menetelmiin. XP:ssä on nimittäin käytänne, jonka mukaan yksikkötestit kirjoitetaan ennen koodausta. Lisäksi käytänteet, kuten kymmenen minuutin päivitysvälit ja jatkuva integrointi korostavat säännöllisen integrointitestauksen merkitystä. Näin ollen testaus on vahvasti sidoksissa toteutukseen. Tuotantovaiheessa keskitytään lisäksi suorituskyvyn testaamiseen. FDD:ssä testaus suoritetaan piirteiden toteutuksen jälkeen. Yksikkötestauksen lisäksi piirretiimeissä tehdään virallisia koodin tarkistuksia. Lopuksi suoritetaan integrointitestaus. DSDM:ssä testausta ei eroteta ohjelmoinnista, vaan menetelmän periaatteiden mukaan testausta suoritetaan läpi elinkaaren. Testaus painottuu toteutuksen tapaan suunnittelun ja toteutuksen iterointi -vaiheeseen, jolloin testauksen yhteydessä laaditaan myös testitilastot.

5.2 Mallintamistekniikat

Tässä kohdassa vertaillaan eri mallintamistekniikoiden käyttöä ketterissä menetelmissä. Tarkoituksena on selvittää, mikä on UML:n merkitys mallintamistekniikkana ja miten sitä käytetään. Mallintamisessa havaittavia eroja ja yhteneväisyyksiä pyritään selittämään muun muassa menetelmien arvojen, periaatteiden ja käytänteiden avulla. Tuloksia verrataan UML-kaavioiden osalta siihen, miten niitä käytännöstä käytetään. Lopuksi tuodaan esille lähestymistapoja, jotka yhdistävät UML:n ja ketterän lähestymistavan.

Taulukkoon 20 on listattu tarkasteltavissa menetelmissä mainittuja mallintamistekniikoita. Listan alkuosa koostuu UML-kaavioista, jonka jälkeen on esitetty menetelmissä käytettävät UML:n ulkopuoliset mallintamistekniikat. Mallintamistekniikan kohdalle on merkitty iso rasti, jos kyseisellä

mallintamistekniikalla on merkittävä osa tarkasteltavassa menetelmässä. Pienempi rasti puolestaan ilmaisee, että mallintamistekniikkaa käytetään, muttei se kuitenkaan ole keskeisimpiä menetelmässä käytettäviä mallintamistekniikoita. Sulkeilla ympäröidyt merkinnät tarkoittavat sitä, että niiden käytöstä mallintamistekniikkana on vahvasti eriäviä näkemyksiä kyseisen menetelmän keskeisissä lähteissä.

TAULUKKO 20. Ketterien menetelmien mallintamistekniikat

	XP	FDD	DSDM
Luokkakaavio	(x)	X	
Käyttötapauskaavio			x
Sekvenssikaavio	(x)	X	
Käyttäjätarinat	X		
CRC-kortit	x		
Piirrelista		X	
Kontekstikaavio			x
ER-kaavio			x
Tietovirtakaavio			x
Priorisoitu vaatimuslista			X

Kuten taulukosta voidaan havaita, käytettävien mallintamistekniikoiden määrä ei ole järin suuri. Syitä tähän voidaan hakea menetelmille yhteisen Agile Manifestin (Agile Alliance 2001) arvoista ja periaatteista. Manifestin arvoissa nostetaan toimiva ohjelmisto tärkeämmäksi kuin kattava dokumentointi. Tämän voidaan katsoa selittävän ainakin sitä, ettei malleja luoda ensisijaisesti dokumentoinnin vaan kommunikoinnin tueksi. Vähäinen mallintamistekniikoiden määrä ei kuitenkaan tarkoita laadun ja suunnittelun

laiminlyöntiä. Manifestin erään periaatteen mukaan laatuun ja suunnitteluun tulisi kiinnittää jatkuvasti huomiota. Erään toisen periaatteen mukaan ohjelmistosta tulisi toimittaa säännöllisin ja lyhyin aikavälein uusi versio. Jotta voitaisiin painottaa sekä laatua että nopeutta, on tehtävä kompromisseja. Manifestin periaatteiden mukaan on pyrittävä yksinkertaisuuteen. On osattava priorisoida vaatimukset ensisijaisiin ja toissijaisiin. Voidaan siis olettaa, että mallintamistekniikat, jotka mahdollistavat riittävän pienien kokonaisuuksien hallinnan ja ovat vaivattomasti päivitettävissä, tukisivat parhaiten ketterän lähestymistavan mukaista järjestelmäkehitystä. Jokaisella menetelmällä on kuitenkin tarkemmin määritellyt periaatteet tai käytänteet, jotka mahdollisesti ohjaavat tiettyjen mallintamistekniikoiden käyttöön niiden ollessa suotuisampia kuin toiset tekniikat. Periaatteet on määritelty ylemmällä tasolla kuin käytänteet, eivätkä niistä käy yhtä selvästi ilmi ne syyt, jotka ohjaavat mallintamistekniikoiden valintaa ja käyttöä. XP:ssä on määritelty arvot, periaatteet ja käytänteet. FDD:ssä vain käytänteet ja DSDM:ssä periaatteet.

XP:n arvoissa ja periaatteissa heijastuvat samat lähtökohdat kuin Manifestin vastaavista. Laadun, viestinnän ja säännöllisesti etenevän kehityksen merkitys on havaittavissa. Käytänteistä ilmenee niin ikään jatkuvan kehityksen ja inkrementaalisuuden merkitys. Käyttäjätarinat on ainoa nimetty käytänne, joka selkeästi ohjaa mallintamista ja määrittää tekniikan, jolla vaatimuksia muodostetaan ja hallitaan. Asiakkaan laatimat käyttäjätarinat ovat epäformaaleja eikä niiden laatimiselle ole tarkkoja sääntöjä. Inkrementaalisen suunnittelun periaatteen voidaan katsoa ohjaavan sitä, minkälaiset suunnittelutekniikat sopivat XP:n yhteydessä käytettäväksi. Tarkoitus on suunnitella koodauksen kanssa rinnakkain ja välttää laajamittaista etukäteissuunnittelua. Tämä selittää suunnittelutekniikkana yksinkertaisten CRC-korttien soveltuvuuden XP:n yhteydessä. XP:ssä ollaan erimielisiä UML-kaavioiden tarpeellisuudesta. Niissä lähteissä, joissa UML-kaavioiden käyttöä suositellaan, käytetään luokkakaavioita kohdealueen mallintamiseen ja

iterointivaiheissa yksityiskohtaisempaan suunnitteluun yhdessä sekvenssikaavioiden kanssa.

FDD:ssä on kaksi käytännettä, jotka ohjaavat mallintamista. Nämä käytänteet ovat kohdealueen mallintaminen ja piirteiden kehittäminen. Kohdealueen mallintamisen tarkoituksena on muodostaa yhtenäinen näkemys kohdeympäristöstä. Mallintamisessa käytetään pääasiassa luokkakaavioita tukenaan korkean tason sekvenssikaaviot käyttäytymisen mallintamiseksi. Syynä siihen, miksi käytetään juuri luokkakaaviota, esimerkiksi ER-kaavioiden sijaan, on kaksi. Luokkakaaviot mahdollistavat periyttämisen, yleistys- ja erilaistamissuhteet luokkien välillä sekä operaatiot, jotka määrittävät, miten oliot käyttäytyvät (Palmer & Felsing 2002, 36). FDD:ssä nähdään siis oliosuuntautuneisuus niin merkittävänä tekijänä, että se ohjaa myös käytettävien mallintamistekniikoiden valintaa. Piirteiden kehittäminen tapahtuu kohdealueen mallintamisen jälkeen. Kaikki piirteet määritellään samassa muodossa toiminnon, kohteen ja tavoitteen avulla. Pienet, asiakkaalle arvoa tuottavat piirteet ohjaavat kehitystä käyttäjätarinoiden tai käyttötapausten tavoin.

DSDM:ssä käytetään vaatimusten hallinnassa priorisoitua vaatimuslistaa. DSDM:n periaatteiden mukaan vaatimukset halutaan alustavasti kiinnittää korkealla tasolla, mikä poikkeaa muista ketterien menetelmien tavasta käsitellä vaatimuksia. Toisaalta asiakkaalla on kuitenkin mahdollisuus tehdä muutoksia ja kaikkien muutosten pitää olla kehityksen aikana peruttavissa. Tämä edellyttäisi selkeää jäljitettävyyttä prosessien vaiheiden, tuotosten ja mallien välillä, jota ei ole kuitenkaan havaittavissa. DSDM:ssä käytettävät mallintamistekniikat eivät ole tiukasti sidottuja, vaan ne voivat vaihdella projektin mukaan. DSDM:n periaatteiden mukaan mallintamistekniikoiden tulee soveltua liiketoimintatarkoituksiin ja käyttäjän osallistuminen on välttämätöntä. Näin ollen pyritään valitsemaan tilanteeseen sopivat

mallintamistekniikat, joita sekä käyttäjät että kehittäjät ymmärtävät. DSDM:ssä käytetään kontekstikaaviota, ER-kaaviota ja tietovirtakaaviota, jotka ovat rakenteisessa lähestymistavassa käytettäviä mallintamistekniikoita. Yhtenä selityksenä niiden merkittävään asemaan suhteessa oliolähestymistapaan pohjautuviin tekniikoihin voidaan pitää DSDM:n kehittymisajankohtaa. DSDM on kehitetty 1990-luvun puolessa välissä, jolloin rakenteisella lähestymistavalla oli vielä merkittävämpi asema kuin nykyään.

Menetelmissä on käytetty vain kolmea UML-kaaviota, luokkakaaviota, sekvenssikaaviota ja käyttötapauskaaviota. DSDM:ssä käyttötapauskaavio on ainoa UML-kaavio, jonka käyttöä erityisesti ohjeistetaan. Sen tarkoituksena on liiketoimintaprosessien esittäminen. XP:ssä ja FDD:ssä luokkakaavioita käytetään kohdealueen mallintamiseen sekä iterointivaiheissa yksityiskohtaisempaan suunnitteluun yhdessä sekvenssikaavioiden kanssa. On kuitenkin huomattava, että XP:ssä UML-kaavioiden tarpeellisuudesta ollaan erimielisiä. FDD:ssä luokka- ja sekvenssikaaviolla on erittäin tärkeä merkitys, ja voidaankin todeta, että se on tarkasteltavista menetelmistä ainoa, jossa UML-kaaviot ovat keskeisessä asemassa mallintamistekniikkana.

UML:n tarkoitus ketterissä menetelmissä on havainnollistaa kohdealuetta sekä tukea toteutuksen yksityiskohtaisempaa suunnittelua. Vaatimusmäärittelyssä UML:ää ei nähdä hyödyllisenä, vaan luotetaan luonnollisella kielellä laadittuihin vaatimuskuvauksiin tai listoihin. UML:ää ei myöskään nähdä hyödyllisenä dokumentoinnissa, mikä on linjassa sen kanssa, että dokumentoinnin osuutta ei korosteta ketterissä menetelmissä.

5.3 Vertailu empiirisiin tutkimuksiin

Saatuja tuloksia voidaan verrata siihen, millaista UML-kaavioiden käyttö on käytännössä. Dobing ja Parsons (2005) ovat pyrkineet selvittämään, mitä UML-

kaavioita käytetään ja miksi. He rajasivat tarkastelun seitsemään kaavioon, joita käytetään järjestelmän toiminnallisuuden kuvaamisessa rakenteen ja käyttäytymisen näkökulmasta. Tarkasteltavia tekniikoita olivat luokka-, käyttötapaus-, sekvenssi-, toiminto-, tila- ja yhteistyökaaviot (UML 1.x) sekä käyttötapauskuvaukset (use case narrative). Tuloksista ilmeni, että luokkakaavio oli käytetyin ja yhteistyökaavio oli vähiten käytetyin tarkasteltavista kaavioista. Luokkakaaviot nähtiin hyödyllisiksi kehittäjien välisessä viestinnässä. Yhteistyökaaviot olivat vähiten hyödyllisiä, sillä ne eivät tuoneet juurikaan lisäarvoa suhteessa sekvenssikaavioihin. Käyttötapauskuvauksia pidettiin tärkeimpänä muodostettaessa vaatimuksia asiakkaan kanssa. Lisäksi niillä on tärkeä merkitys tuotosten arvioinnissa ja hyväksymisessä. Dobing ja Parsons (2005) sanovat, että käyttötapauskuvausten avulla voidaan kommunikoida asiakkaan kanssa tehokkaammin kuin varsinaisten UML-kaavioiden avulla. Yksinkertainen ja helposti omaksuttava tapa viestiä näyttää toimivan asiakkaiden ja kehittäjien välisessä viestinnässä, etenkin vaatimusmäärittelyssä. Tämä tukee sitä, että ketterissä menetelmissä suositaan yksinkertaisuutta ja luonnollista kieltä vaatimuksia kuvattaessa. XP:ssä käyttötapauskuvaukset ovatkin tästä syystä vaihtoehto käyttäjätarinoille, vaikkei niiden käyttöä suositellakaan.

Erickson ja Siau (2004) ovat määrittäneet empiiristen tutkimusten avulla, kuinka tärkeäksi kukin UML-kaavio käytännössä koetaan. Kyselyn vastausten perusteella Erickson ja Siau (2004) esittävät UML:n tärkeimpien kaavioiden, niin sanotun ydinjoukon, muodostuvan neljästä kaaviosta: luokka-, käyttötapaus-, sekvenssi- ja tilakaaviosta. Myöhemmin ilmestyneessä vastaavanlaisessa tutkimuksessa Erickson ja Siau (2007) määrittivät ydinjoukkoa eri kohdealueiden mukaan. Tarkasteltavia kohdealueita olivat reaaliaikaiset järjestelmät, web-pohjaiset järjestelmät ja yritysjärjestelmät (enterprise). Kohdealueriippumattomissa tuloksissa päädyttiin vastaaviin tuloksiin aiemman tutkimuksen kanssa. Luokka-, käyttötapaus-, sekvenssi- ja

tilakaavio muodostivat ydinjoukon myös reaaliaikaisissa ja web-pohjaisissa järjestelmissä. Yritysjärjestelmien kohdalla ydinjoukon koostumus kuitenkin muuttui. Toimintokaavio oli syrjäyttänyt tilakaavion. Tämän perusteella Erickson ja Siau (2007) määrittävät ydinjoukon koostuvat enää kolmesta UML-kaaviosta: luokka-, käyttötapaus- ja sekvenssikaaviosta. Tutkimuksessa tuli myös ilmi, että luokkakaaviota pidettiin selkeästi tärkeimpänä kaaviona. Voidaan siis todeta, että ketterissä menetelmissä on käytetty vain niitä UML-kaavioita, jotka muodostavat Ericksonin ja Siaun (2007) määrittelemän UML:n ydinjoukon.

5.4 Vertailu muihin lähestymistapoihin

Seuraavaksi esitellään Agile Modeling (AM) (Ambler 2007a) ja Essential Unified Processissa (EssUP) (Jacobson, Ng & Spence 2005) käytettävä Essential UML. Nämä ovat lähestymistapoja, jotka hyödyntävät voimakkaasti mallintamista, erityisesti UML:n avulla, ketterässä ohjelmistokehityksessä.

AM (Agile Modeling) (Ambler 2007a) on käytänteisiin perustuva menetelmä ohjelmistojen mallintamiseen ja dokumentointiin. AM ei ole varsinainen ohjelmistoprosessi, eikä se kata muun muassa projektinhallintaa tai järjestelmän sijoitusta (Ambler 2004, 129). AM:ää sovelletaankin yleensä AUP:n (Agile Unified Process) (Ambler 2007b) kanssa, mutta sitä voidaan hyödyntää myös muissa ohjelmistoprosesseissa, kuten XP:ssä ja RUP:ssa (Ambler 2007a). AM on kokoelma arvoja, periaatteita ja käytänteitä ohjelmistojen mallintamiseksi, joita voidaan soveltaa ohjelmistojen kehitysprojekteissa tehokkaalla ja kevyellä tavalla (Ambler 2007a).

AM pohjautuu viiteen arvoon, joista neljä ensimmäistä on lainattu XP:stä. Nämä arvot ovat kommunikaatio, yksinkertaisuus, palaute, rohkeus ja inhimillisyys (Ambler 2004, 108). Arvojen lisäksi on yksitoista perusperiaatetta,

joista useat vastaavat XP:n periaatteita. Periaatteet ovat: mallinna tarkoituksenmukaisesti, maksimoi sidosryhmien sijoitus, omaksu muutos, inkrementaalinen muutos, lukuisat mallit, etene kevyesti, toimiva ohjelmisto on päätavoite, seuraavan ponnistuksen mahdollistaminen on seuraava tavoite, laatutyö, nopea palaute, omaksu yksinkertaisuus (Ambler 2004, 110–111). Lukuisilla malleilla viitataan siihen, että tarjolla tulee olla runsaasti vaihtoehtoisia malleja, jotka mahdollistavat eri näkökulmien tarkastelun. Mallien joukosta tulee valita tilanteeseen sopivat mallit ja käyttää vain niitä. Seuraavan ponnistuksen mahdollistamisella tarkoitetaan sitä, että kun toimitetaan ohjelmistojulkaisu, huolehditaan riittävästä dokumentoinnista ja tukimateriaalista, jotka helpottavat jatkokehitystä. (Ambler 2007a)

Periaatteiden pohjalta on muodostettu kolmetoista pääkäytännettä, joita ovat: aktivoi sidosryhmien osallistuminen, käytä yksinkertaisimpia työkaluja, mallinna yhdessä, todista koodilla, sovella oikeaa välinettä, luo useita rinnakkaismalleja, iteroi toisen tuotoksen avulla, mallinna vähän kerrallaan, esitä mallit julkisesti ja säilytä tieto yhdessä lähteessä (Ambler 2007a). Toisen tuotoksen avulla iteroinnilla tarkoitetaan sitä, että jos kehitystyössä ilmenee ongelmia, esimerkiksi työstäessä CRC-korttia, sekvenssikaaviota tai itse koodia, on syytä vaihtaa toiseen tuotokseen. Tämän avulla varmistutaan siitä, että kehitystyö etenee koko ajan. Lisäksi toisen tuotoksen työstäminen voi auttaa ratkaisemaan edellisen tuotoksen ongelmakohtia. (Ambler 2007a)

AM:n periaatteista ja käytänteistä ilmenee, että mallintaminen on keskeisessä asemassa. AM luo pohjan Agile Model Driven Developmentille (AMDD), joka on ketterä versio Model Driven Developmentista (MDD). AMDD eroaa perinteisestä MDD:stä siinä, että tarkoituksena ei ole luoda täydellisiä ja kattavia malleja ennen koodausta, vaan luoda malleja, jotka ovat riittävän hyviä. (Ambler 2004, 118)

Ambler (2007) luokittelee UML-kaaviot asteikolla tärkeä, keskinkertainen ja alhainen sen mukaan, kuinka tärkeänä hän pitää kutakin kaaviota. Ambler (2007) nostaa tärkeimmiksi kaavioiksi luokka-, sekvenssi- ja toimintokaaviot. Käyttötapauskaavio ja tilakaavio saavat keskinkertaisen luokituksen. Huomion arvoista on se, että kaikki UML 2:n myötä tulleet uudet kaaviot saavat alimman luokituksen.

Essential Unified Process (EssUP) (Jacobson, Ng & Spence 2005) on UP:hen pohjautuva, parhaita käytäntöjä hyödyntävä ketterä lähestymistapa. EssUP:ssa mallintamisen tarkoituksena on edistää viestintää järjestelmän vaatimuksista, rakenteesta ja käyttäytymisestä. Tavoitteena on käyttää oikeita malleja oikeissa tilanteissa tarkoituksenmukaisella ja käytännönläheisellä tavalla välttäen ylimääräistä mallintamista ja dokumentointia. Mallintamisessa käytetään UML:ää, muttei kuitenkaan niin laajamittaisesti kuin UP:ssa. Essential UML (Jacobson, Ng & Spence 2005) ohjaa EssUP:ssa suoritettavaa mallintamista. Se edustaa vastaavanlaista näkemystä kuin AM. Essential UML keskittyy Pareto-säännön mukaisesti olennaisimpaan UML:n osajoukkoon, jolla voitaisiin suorittaa suurin osa mallintamistehtävistä. Jacobson, Ng ja Spence (2005, 24) listaavat Essential UML:ssä käytettävät UML-kaaviot ja niiden käyttötarkoitukset. Listauksesta ei kuitenkaan ilmene, mitkä UML-kaaviot ovat keskeisimpiä. Oliokaavio ja koostekaavio puuttuvat kokonaan listasta. Listasta ei myöskään ilmene yksiselitteisesti UML:n 2:n myötä tulleiden uusien vuorovaikutuskaavioiden merkitystä, koska vuorovaikutuskaavioita ei ole eritelty toisistaan. Ketteryyttä ei siis tavoitella pitäytymällä harvoissa ja valikoiduissa UML-kaavioissa, vaan soveltamalla oikeita kaavioita oikeissa tilanteissa.

5.5 Yhteenveto vertailusta ja analyysistä

Tässä luvussa vertailtiin ja analysoitiin XP:n (Beck 1999), FDD:n (Nebulon Pty. Ltd. 2004) ja DSDM:n (DSDM Consortium 2007) prosesseja ja mallintamistekniikoita. Menetelmien prosesseja jäsennettiin UP-viitekehyksen avulla tarkastellen vaiheita, iteraatioita ja työnkulkuja. Vertailussa havaittiin, että prosessien ensimmäiset kertaluonteiset vaiheet keskittyvät pääasiassa vaatimusmäärittelyyn ja analyysiin, joita seuraavat iteratiiviset toteutusvaiheet. FDD:ssä tämä kaksivaiheisuus näkyi poikkeuksellisen voimakkaasti siinä, että vaatimukset pyrittiin määrittämään mahdollisimman kattavasti ennen toteutukseen siirtymistä, mikä on poikkeuksellista ketterälle lähestymistavalle.

Tarkastelussa havaittiin edelleen, että menetelmissä käytettävien mallintamistekniikoiden määrä on vähäinen. Menetelmissä käytetään vaatimusten määrittämisessä luonnollisella kielellä tuotettuja kuvauksia: XP:ssä käyttäjätarinoita, FDD:ssä piirrelistoja ja DSDM:ssä priorisoituja vaatimuslistoja. Tätä voidaan selittää sillä, että malleja tuotetaan kommunikoinnin eikä dokumentoinnin tueksi. Epäformaalit tekniikat ovat yksinkertaisia ja helposti päivitettäviä. Menetelmissä on käytetty kolmea UML-kaaviota, luokkakaaviota, sekvenssikaaviota ja käyttötapauskaaviota. DSDM:ssä käyttötapauskaaviota voidaan käyttää liiketoimintaprosessien esittämiseen. XP:ssä ja FDD:ssä luokkakaavioita käytetään kohdealueen mallintamiseen sekä iterointivaiheissa yksityiskohtaisempaan suunnitteluun yhdessä sekvenssikaavioiden kanssa. XP:ssä UML-kaavioiden tarpeellisuudesta ollaan erimielisiä, kun taas FDD:ssä ne ovat keskeisessä asemassa.

Empiiriset tutkimukset (Erickson & Siau 2004; 2007) tukevat sitä, että ketterissä menetelmissä suositaan niitä UML-kaavioita, jotka koetaan yleisestikin tärkeimmiksi kaavioiksi. On kuitenkin lähestymistapoja, kuten AM (Ambler 2007a) ja Essential UML (Jacobson, Ng & Spence 2005), jotka pyrkivät

hyödyntämään UML:ää tätäkin laajemmin, mutta kuitenkin tarkoituksenmukaisesti ja ketterän lähestymistavan mukaisesti.

6 YHTEENVETO JA JOHTOPÄÄTÖKSET

Tämän tutkielman tarkoituksena oli verrata ja analysoida ketteriä menetelmiä. Tarkasteltavina menetelminä olivat XP (Beck 1999), FDD (Nebulon Pty. Ltd. 2004) ja DSDM (DSDM Consortium 2007), joita verrattiin toisiinsa prosessien ja mallintamisen osalta. Tarkoituksena oli vastata seuraaviin tutkimuskysymyksiin:

- Millaisia prosessimalleja ketterissä menetelmissä käytetään ja miten ne vastaavat UP-prosessia?
- Mikä on mallintamisen merkitys ketterissä menetelmissä ja millaisia mallintamistekniikoita käytetään ja miten niiden käyttöä on ohjeistettu?
- Miten ketterissä menetelmissä on ohjeistettu UML-kaavioiden käyttöä ja mikä on niiden merkitys mallintamistekniikkana?
- Miten ketterien menetelmien prosessimallit ja mallintamistekniikat eroavat toisistaan ja miten näitä eroja voidaan selittää?

Tarkasteltavat menetelmät kuvattiin jokaisen menetelmän osalta seuraavasti. Ensin tuotiin esille menetelmille ominaisia arvoja, periaatteita ja käytänteitä. Seuraavaksi esiteltiin prosessien vaiheet, tavoitteet ja tuotokset, jonka jälkeen kuvattiin menetelmissä käytettävät mallintamistekniikat ja niiden käyttötarkoitukset. Tätä seuranneen vertailun ja analyysin perustana käytettiin UP-prosessimallia (Jacobson, Booch & Rumbaugh 1999) ja UML-kieltä (Booch, Rumbaugh & Jacobson 1999). Prosessien vaiheita, niiden sisältöjä ja iteratiivisuutta verrattiin toisiinsa ja UP:hen. Lisäksi verrattiin ja analysoitiin sitä, miten ja missä vaiheissa menetelmiä UP:n työnkulut esiintyvät. Menetelmissä käytettäviä mallintamistekniikoita vertailtiin toisiinsa. Erityisesti kiinnitettiin huomiota UML:n käyttöön ja merkitykseen mallintamistekniikkana. Menetelmien sisältämiä UML-mallintamistekniikkoja verrattiin tutkimustuloksiin, jotka koskevat yleisesti UML-mallintamistekniikoiden käyttöä käytännössä. Lisäksi kuvattiin kaksi voimakkaasti mallintamista hyödyntävää ketterää lähestymistapaa, AM

(Ambler 2007a) ja Essential UML (Jacobson, Ng & Spence 2005) ja vertailtiin niitä tässä työssä tarkasteltuihin menetelmiin erityisesti mallintamisen osalta.

Vertailun ja analyysin pohjalta saatiin seuraavanlaisia tuloksia. Tarkasteltujen ketterien menetelmien vaiheet ovat yleisellä tasolla toistensa kaltaisia sisältäen kertaluonteiset kartoitus- ja määrittelyosuudet, joita seuraa iteratiivinen toteutusosuus. FDD:n kohdalla tämä kaksivaiheisuus näkyi poikkeuksellisen voimakkaasti siinä, että vaatimukset pyrittiin määrittämään mahdollisimman kattavasti ennen toteutukseen siirtymistä, mikä on poikkeuksellista ketterälle lähestymistavalle.

Vaiheiden aikana muodostettavien tuotosten määrä on suhteellisen vähäinen paitsi DSDM:ssä. Osittain tästä syystä myös erilaisten mallintamistekniikoiden määrä on vähäinen. Vaikka mallintaminen nähdään hyödyllisenä jokaisessa menetelmässä, lähtökohdat poikkeavat toisistaan. XP:ssä pyritään välttämään laajamittaista määrittelyä ja etukäteissuunnittelua, hyväksymään uudet vaatimukset ja suunnittelemaan toteutuksen kanssa rinnakkain. FDD:ssä puolestaan pyritään mahdollisimman tarkkaan vaatimusten määrittämiseen projektin alkuvaiheessa ja muodostamaan järjestelmän suunnittelu ennen toteutusta.

Menetelmissä käytetään vaatimusten määrittämisessä luonnollisella kielellä tuotettuja kuvauksia: XP:ssä käyttäjätarinoita, FDD:ssä piirrelistoja ja DSDM:ssä priorisoituja vaatimuslistoja. DSDM:ssä hyödynnetään erilaisia mallintamistekniikoita, etenkin rakenteisen lähestymistavan mukaisia kontekstikaavioita, ER-kaavioita ja tietovirtakaavioita. Mallintamistekniikoiden käyttöä ei ole kovinkaan tarkkaan määritelty. XP:ssä suunnittelussa käytetään apuna usein CRC-kortteja. FDD:ssä on ohjeistettu muita yksityiskohtaisemmin mallintamistekniikoiden käyttö. Menetelmissä on käytetty kolmea UML-kaaviota, luokkakaaviota, sekvenssikaaviota ja käyttötapauskaaviota.

DSDM:ssä käyttötapauskaaviota voidaan käyttää liiketoimintaprosessien esittämiseen. XP:ssä ja FDD:ssä luokkakaavioita käytetään kohdealueen mallintamiseen sekä iterointivaiheissa yksityiskohtaisempaan suunnitteluun yhdessä sekvenssikaavioiden kanssa. XP:ssä UML-kaavioiden tarpeellisuudesta ollaan erimielisiä, kun taas FDD:ssä ne ovat keskeisessä asemassa.

Empiiriset tutkimukset (Erickson & Siau 2004; 2007) tukevat sitä, että ketterissä menetelmissä suositaan niitä UML-kaavioita, jotka koetaan yleisestikin tärkeimmiksi kaavioiksi. AM (Ambler 2007a) ja Essential UML (Jacobson, Ng & Spence 2005) pyrkivät hyödyntämään UML:ää tätäkin laajemmin, mutta kuitenkin tarkoituksenmukaisesti ja ketterän lähestymistavan mukaisesti.

Tämän tutkimuksen tarkoituksena on ollut jäsentää ketterien menetelmien prosesseja ja mallintamistekniikoita. Useimmat tutkimukset keskittyvät antamaan yleiskuvan menetelmistä ja niiden ominaispiirteistä (esim., Abrahamsson ym. 2002; Cohen ym. 2003). Menetelmien prosesseja on usein pyritty jäsentämään menetelmien omien käytänteiden avulla (esim., Qumer & Henderson-Sellers 2007; Visconti & Cook 2004), eikä niinkään yhteisen viitekehyksen, kuten UP:n, avulla. Vain jotkut tutkimukset, kuten Stojanovic ym. 2003, vertailevat mallintamista eri menetelmissä, kuitenkin melko pintapuolisesti.

Ketterien menetelmien analyysia ja vertailua voidaan laajentaa ja syventää useammalla tavalla. Tässä tutkielmassa on tarkasteltu vain kolmea menetelmää. Tutkimusta rajoittavana tekijänä oli myös se, että pääsy DSDM:ää käsittelevään aineistoon oli osittain rajoitettu. Vaikka kyseiset menetelmät ovat tunnetuimpia ja edustavat hyvin ketterää lähestymistapaa, niin tarkemman kuvan saamiseksi laajan menetelmäjoukon prosessi- ja mallintamisominaisuuksia, on tarpeen laajentaa tarkastelu koskemaan muitakin menetelmiä (esim., Scrum (Advanced Development Methods, Inc. 2005), Crystal-menetelmät (Cockburn 2005),

Adaptive Software Development (ASD) (Highsmith 2000) Agile Unified Process (AUP) (Ambler 2007b), Essential Unified Process (EssUP) (Jacobson, Ng & Spence 2005) ja Microsoft Solutions Framework (MSF) (Microsoft Corporation 2007). Menetelmiä analyysia ja vertailua voitaisiin myös tarkentaa käyttämällä metamallintamisen periaatteita (vrt. Leppänen 2005). Ketterien menetelmien käytöstä on olemassa lukuisia tutkimuksia (esim., Drobka ym. 2004; Lindvall ym. 2004; Sfetsos ym. 2006). Kiintoisaa olisi selvittää näiden tutkimusten perusteella, missä määrin kyseisiä menetelmiä sovelletaan käytännössä menetelmäkuvausten mukaisina erityisesti prosessien ja mallintamistekniikoiden osalta.

Tutkimuksessa muodostettua kuvaa menetelmien prosesseista ja mallintamistekniikoista voidaan käyttää asianomaisten menetelmien jatkokehittelyn pohjana sekä suunniteltaessa empiirisiä tutkimuksia. Käytännön työssä tuloksia voidaan suoraan hyödyntää tehtäessä menetelmävalintaa ja -sovitusta organisaatiota tai yksittäistä projektia varten. Tutkielmaa voisi käyttää myös perusoppimateriaalina ketteriä menetelmiä käsittelevällä kurssilla.

LÄHDELUETTELO

Abrahamsson P., Salo O., Ronkainen J. & Warsta J. 2002. Agile software development methods: review and analysis. Espoo: VTT.

Abrahamsson P., Warsta J., Siponen M. & Ronkainen J. 2003. New Directions on Agile Methods: A Comparative Analysis. Teoksessa A. Jacobs & F. Titsworth (toim.) International Conference on Software Engineering Proceedings of the 25th International Conference on Software Engineering Portland, Oregon, USA, May 3–10. Washington: IEEE Computer Society, 244–254.

Advanced Development Methods, Inc. 2005. Scrum [online]. Advanced Development Methods, Inc [viitattu 31.5.2005]. Saatavilla www-osoitteessa <http://www.controlchaos.com/>.

Agile Alliance 2001. Agile Manifesto [online]. Agile Alliance [viitattu 27.1.2005]. Saatavilla www-osoitteessa <http://agilemanifesto.org/>.

Ambler S. 2004. The Object Primer: Agile Model-Driven Development with UML 2.0. Cambridge : Cambridge University Press New York.

Ambler S. 2006. Agile Adoption Rate Survey [online]. Ambler [viitattu 29.7.2007]. Saatavilla www-osoitteessa <http://www.ambysoft.com/surveys/agileMarch2006.html#Discussion>.

Ambler S. 2007a. Agile Modeling (AM) Home Page: Effective Practices for Modeling and Documentation [online]. Ambler [viitattu 6.7.2007]. Saatavilla www-osoitteessa <http://www.agilemodeling.com/>.

Ambler S. 2007b. The Agile Unified Process (AUP) [online]. Ambler [viitattu 3.9.2007]. Saatavilla www-osoitteessa <http://www.ambysoft.com/unifiedprocess/agileUP.html>.

Astels D., Miller G. & Noval M. 2002. A practical guide to eXtreme programming. Upper Saddle River (NJ): Prentice-Hall.

Barnett L. 2006. And The Agile Survey Says… [online]. Agile Journal [viitattu 29.7.2007]. Saatavilla www-osoitteessa <

http://www.agilejournal.com/articles/from-the-editor/and-the-agile-survey-says%85/>.

Baskerville R., Travis J. & Truex D. P. 1992. Systems without method: The impact of new technologies on information systems development projects. Teoksessa K. E. Kendall, K. Lyytinen & J. I. DeGross (toim.) in the impact of computer supported technologies in information systems development. Amsterdam: North-Holland Publishing Co., 241–269.

Baskerville R., Levine L., Pries-Heje J., Ramesh B. & Slaughter S. 2001. How Internet companies negotiate quality. Computer 34(5), 51–57.

Baskerville R. & Pries-Heje J. 2001. Racing the E-bomb: How the Internet is redefining information systems development methodology. Teoksessa B. Fitzgerald, N. Russo & J. DeGross (toim.) in Realigning research and practice in IS development. New York: Kluwer, 49–68.

Bayer S. & Highsmith J. 1994. RADical Software Development. American Programmer 7(6), 35–42.

Beck K. 1999. Extreme Programming Explained: Embrace Change. Reading, Massachusetts: Addison-Wesley Professional.

Beck K. & Andres C. 2004. Extreme Programming Explained: Embrace Change. Addison-Wesley Professional.

Beck K. & Cunningham W. 1989. A laboratory for teaching object oriented thinking. Teoksessa N. Meyrowitz (toim.) Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications, New Orleans, Louisiana, USA, October 1–6. New York: ACM Press, 1–6.

Behrens P. 2006. Agile Project Management (APM) Tooling Survey Results [online]. Trail Ridge Consulting [viitattu 27.7.2007]. Saatavilla www-osoitteessa
<http://www.trailridgeconsulting.com/files/2006AgileToolingSurveyResults.pdf>.

Berkenkötter K. 2003. Using UML 2.0 in Real-Time Development A Critical Review. SVERTS workshop in UML 2003, San Francisco, October 20.

Saatavilla www-osoitteessa <http://www-verimag.imag.fr/EVENTS/2003/SVERTS/PAPERS-WEB/04-Berkenkoetter-UMLRT-critic.pdf>.

Blue Ink. 2006. Rapid Application Development [online]. Blue Ink [viitattu 27.1.2006]. Saatavilla www-osoitteessa <http://www.blueink.biz/RapidApplicationDevelopment.aspx>.

Boehm B. 1986. A spriral model of software development and enhancement. ACM SIGSOFT Software Engineering Notes 11(4), 14–24.

Boehm B. & Turner R. 2003. Observations on Balancing Discipline and Agility. Teoksessa Proceedings of the Agile Development Conference (ADC ’03), Salt Lake City, Utah, June 25–28. Washington, DC: IEEE Computer Society, 32–39.

Booch G., Rumbaugh J. & Jacobson I. 1999. The unified modeling language user guide. Reading, Massachusetts: Addison-Wesley Longman, Inc.

CMP Media LLC. 2005. Agile Modeling and Feature-Driven Development: Not just another agile methodology, Feature-Driven Development [online]. CMP Media LLC [viitattu 22.5.2005]. Saatavilla www-osoitteessa <http://www.sdmagazine.com/documents/s=6977/sdmam0502/samam0502.htm>.

Coad P., LeFebvre E. & De Luca J. 1999. Java Modeling In Color With UML: Enterprise Components and Process. Upper Saddle River, New Jersey: Prentice Hall, Inc.

Cockburn A. 1998. Surviving Object-Oriented Projects: A Manager’s Guide. Reading, Massachusetts: Addison Wesley.

Cockburn A. 2001. Agile Software Development. Boston, Massachusetts: Addison Wesley.

Cockburn A. 2005. Crystal Methodologies [online]. Cockburn [viitattu 31.5.2005]. Saatavilla www-osoitteessa <http://alistair.cockburn.us/crystal/crystal.html>.

Cohen D., Lindvall M. & Costa P. 2003. Agile Software Development, DACS State-of-the-Art/Practice Report. Fraunhofer Center for Experimental Software Engineering Maryland and The University of Maryland.

Control Chaos. 2007. Scrum [online]. Control Chaos [viitattu 13.8.2007]. Saatavilla www-osoitteessa <http://www.controlchaos.com/>.

Cunningham & Cunningham, Inc. 2005. Crc Card [online]. Cunningham & Cunningham [viitattu 22.5.2005]. Saatavilla www-osoitteessa <http://c2.com/cgi/wiki?CrcCard>.

Cusumano M. & Selby R. 1995. Microsoft Secrets: How the World's Most Powerful Software-Company Creates Technology, Shapes Markets and Manages People. New York, NY. Free Press/Simon & Schuster.

Cusumano M. & Selby R. 1997. How Microsoft builds software. Communications of the ACM, 40(6), 53–61.

Cusumano M. & Yoffie D. 1999. Software development on Internet time. IEEE Computer, 32(10), 60–69.

Dobing B. & Parsons J. 2005. Current Practices in the Use of UML. Teoksessa J. Akoka ym. (toim.) Perspectives in Conceptual Modeling, ER 2005 Workshops AOIS, BP-UML, CoMoGIS, eCOMO, and QoIS Klagenfurt, Austria, October 24–28. Berlin: Springer-Verlag, 2–11.

Dori D. 2002. Why Significant UML Change Is Unlikely. Communications of the ACM 45(11), 82–85.

Drobka J., Noftz D. & Raghu R. 2004. Piloting XP on Four Mission-Critical Projects. IEEE Software 21(6), 70–75.

DSDM Consortium. 1995. Dynamic Systems Development Method (DSDM) Version 2. Surrey: UK, Tesseract Publishing.

DSDM Consortium. 2007. DSDM [online]. DSDM Consortium Ltd [viitattu 8.3.2007]. Saatavilla www-osoitteessa <http://www.dsdm.org/>.

Erickson J. & Siau K. 2004. Theoretical and Practical Complexity of Unified Modeling Language: Delphi Study and Metrics Analyses. Teoksessa R. Agarwal, L. Kirsch & J. I. DeGross (toim.) Proceedings of the International

Conference on Information Systems Washington, DC, USA, December 12–15. Atlanta: Association for Information Systems, 183–194.

Erickson J. & Siau K. 2007. Can UML Be Simplified? Practioner Use of UML in Separate Domains. Proceedings of the 12th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'07) Trondheim, Norway, June 11–12. Saatavilla www-osoitteessa <http://www.emmsad.org/2007/00.pdf>.

Eshuis R. & Wieringa R. 2004. Tool Support for Verifying UML Activity Diagrams. IEEE Transactions on Software Engineering 30(7), 437–447.

Fowler M. 2003. The new Methodology [online]. Fowler [viitattu 31.1.2005]. Saatavilla www-osoitteessa <http://www.martinfowler.com/articles/newMethodology.html>.

Fowler M. & Kendall S. 2004. UML. Jyväskylä: Docendo.

Gilb T. 1988. Principles of Software Engineering Management. Wokingham: Addison-Wesley Longman, Inc.

Henderson-Sellers B. & Gonzalez-Perez C. 2006. Uses and Abuses of the Stereotype Mechanism in UML 1.x and 2.0. Teoksessa O. Nierstrasz, J. Whittle, D. Harel, G. Reggio (toim.) Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1–6, Heidelberg: Springer-Verlag, 16–26.

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

Highsmith J. & Cockburn A. 2001. Agile software development: the business of innovation. Computer 34(9), 120–122.

Houghton Mifflin Company. 2000. The American Heritage Dictionary of the English Language, Fourth Edition [online]. Houghton Mifflin Company [viitattu 22.5.2005]. Saatavilla www-osoitteessa <http://www.bartleby.com/61/22/A0142200.html>.

Hunt A. & Thomas D. 2000. The Pragmatic Programmer: from journeyman to master. Boston, Massachusetts: Addison Wesley.

Jacobson I., Booch G. & Rumbaugh J. 1999. The Unified Software Development Process. Reading, Massachusetts: Addison-Wesley Longman, Inc.

Jacobson I., Ng P.W. & Spence I. 2005. Next Generation Process with Essential Modeling [online]. Ivar Jacobson International [viitattu 4.10.2007]. Saatavilla www-osoitteessa <http://www.cs.tut.fi/tapahtumat/olio2006/jacobson.pdf>.

Kobryn C. 2002. Will UML 2.0 Be Agile or Awkward? Communications of the ACM 45(1), 107–110.

Koskimies K., Koskinen J., Maunumaa M., Peltonen J., Selonen P., Siikarla M. & Systä T. 2004. UML työvälineenä ja tutkimuskohteena. Tietojenkäsittelytiede, 21, 19–51.

Kruchten P. 2000. The Rational Unified Process: An Introduction, Second Edition. Reading, Massachusetts: Addison-Wesley Longman, Inc.

Kumar K. & Welke R. 1992. Methodology Engineering: A Proposition For Situation-Specific Methodology Construction. Teoksessa W. W. Cotterman & J. A. Senn (toim.) Challenges and Strategies for Research in Systems Development, Chichester, UK: Wiley & Sons, 257–269

Lantz K. E. 1986. The Prototyping Methodology. Upper Saddle River, New Jersey: Prentice Hall, Inc.

Leppänen M. 2005. An Ontological Framework and a Methodical Skeleton for Method Engineering, Dissertation Thesis, Jyväskylä Studies in Computing 52, University of Jyväskylä.

Lindvall M., Basili V., Boehm B., Costa P., Dangle K., Shull F., Tesoriero R., Williams L. & Zelkowitz M. 2002. Empirical Findings in Agile Methods. Teoksessa Proceedings of Extreme Programming and Agile Methods - XP/Agile Universe 2002, Second XP Universe and First Agile Universe Conference Chicago, IL, USA, August 4–7. London: Springer-Verlag, 197–207.

Lindvall M., Muthig D., Dagnino A., Wallin C., Stupperich M., Kiefer D., May J. & Kähkönen T. 2004. Agile Software Development in Large Organizations. Computer 37(12), 26–34.

Malouin J. L. & Landry M. 1983. The miracle of universal methods in systems design. Journal of Applied Systems Analysis 10, 47–62.

Marchesi M. 2006. The New XP [online]. Agile Experience [viitattu 22.10.2006]. Saatavilla www-osoitteessa <http://www.agilexp.org/downloads/TheNewXP.pdf>.

Martin J. 1991. Rapid Application Development. New York: Macmillan Publishing Company.

McQuillan J.A. & Power J.F. 2005 A Survey of UML-Based Coverage Criteria for Software Testing [online]. Department of Computer Science, NUI Maynooth, Co. Kildare, Ireland [viitattu 4.10.2007]. Saatavilla www-osoitteessa <http://www.minds.nuim.ie/~jmcq/nuim-cs-tr-2005-08.pdf>.

Microsoft Corporation. 2007. Microsoft Solutions Framework [online]. Microsoft Corporation [viitattu 3.9.2007]. Saatavilla www-osoitteessa <http://www.microsoft.com/technet/solutionaccelerators/msf/default.mspx>.

Miller G. 2003. Want a Better Software Development Process? Complement It. IT Professional 5(5), 49–51.

Nandhakumar J. & Avison D.E. 1999. The fiction of methodological development: a field study of information systems development. Information Technology & People 12(2), 176–191.

Nebulon Pty. Ltd. 2004. Agile Software Development using Feature Driven Development (FDD) [online]. Nebulon Pty. Ltd. [viitattu 5.9.2006]. Saatavilla www-osoitteessa <http://www.nebulon.com/fdd/>.

Nebulon Pty. Ltd. 2007. Feature Driven Development [online]. Nebulon Pty. Ltd. [viitattu 6.2.2007]. Saatavilla www-osoitteessa <http://www.featuredrivendevelopment.com/>.

Nerur S., Mahapatra R. & Mangalaraj G. 2005. Challenges of Migrating to Agile Methodologies. Communications of the ACM 48(5), 73–78.

Object Management Group, Inc. 2007. UML Superstructure, v2.1.1 [online]. Object Management Group, Inc [viitattu 7.7.2007] Saatavilla www-osoitteessa < http://www.omg.org/docs/formal/07-02-03.pdf>.

Palmer S. & Felsing J. 2002. Practical Guide to Feature-Driven Development. Upper Saddle River, New Jersey: Prentice Hall, Inc.

Qumer A. & Henderson-Sellers B. 2007. An evaluation of the degree of agility in six agile methods and its applicability for method engineering. Information and Software Technology [viitattu 9.10.2007], painossa (hyväksytty 4.2.2007). Saatavilla www-osoitteessa <http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V0B-4N1JRNN-3&_user=1234512&_coverDate=02%2F11%2F2007&_alid=629477363&_rdoc=3&_fmt=full&_orig=search&_cdi=5642&_sort=d&_docanchor=&view=c&_ct=95&_acct=C000052082&_version=1&_urlVersion=0&_userid=1234512&md5=23b8e6cebcb96378922102297529c914>.

Rakitin S. 2001. Manifesto Elicits Cynicism. Computer 34(12), 4.

Rumbaugh J., Jacobson I. & Booch G. 1999. The unified modeling language reference manual. Reading, Massachusetts: Addison-Wesley Longman, Inc.

Rumpe B. & Schröder A. 2002. Quantitative Survey on Extreme Programming Projects. Teoksessa M. Marchesi & G. Succi (toim.) Proceedings of the Third International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2002), Alghero, Italy, May 26–30. University of Cagliari, 95–100.

Russell N., van der Aalst W.M.P., ter Hofstede A.H.M & Wohed P. 2006. On the suitability of UML 2.0 activity diagrams for business process modelling. Teoksessa M. Stumptner, S. Hartmann & Y. Kiyoki (toim.) Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling (APCCM2006),

Hobart, Australia, January 16–19. Darlington: Australian Computer Society, Inc. 95–104.

Schwaber K. 1995. Scrum Development Process. Teoksessa J. Sutherland, D. Patel, C. Casanave, G. Hollowell & J. Miller (toim.) The Proceedings of the OOPSLA ’95 Workshop on Business Object Design and Implementation, Austin, Texas, USA, October 15–19. London: Springer-Verlag, 117–134.

Schwaber K. & Beedle M. 2001. Agile Software Development with Scrum. Upper Saddle River, New Jersey: Prentice Hall.

Sfetsos P., Angelis L. & Stamelos I. 2006. Investigating the extreme programming system – An empirical study. Empirical Software Engineering 11(2), 269–301.

Shine Technologies. 2003. Agile Methodologies Survey Results [online]. Shine Technologies [viitattu 29.7.2007]. Saatavilla www-osoitteessa <http://www.agilealliance.org/system/article/file/1121/file.pdf>.

Stapleton J. 1997. Dynamic Systems Development Method. Harlow: Addison-Wesley.

Stephens M. 2005. Extreme Programming [online]. Stephens [viitattu 12.12.2005]. Saatavilla www-osoitteessa <http://www.softwarereality.com/ExtremeProgramming.jsp>.

Stojanovic Z., Dahanayke A. & Sol H. 2003. Modeling and Architectural Design in Agile Development Methodologies. Teoksessa J. Krogstie, T. Halpin & K. Siau (toim.) Eighth CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD’03) Velden, Austria, June 16-20. 180–189.

Surgeworks. 2007. EnterpriseXP (DSDM) Templates [online]. Surgeworks [viitattu 8.3.2007]. Saatavilla www-osoitteessa <http://www.surgeworks.com/enterprisexp-templates>.

Takeuchi H. & Nonaka I. 1986. The new new product development game. Harvard Business Review 64(1), 137–146.

Truex D.P., Baskerville R. & Klein H. 1999. Growing systems in emergent organizations. Communications of the ACM, 42(8), 117–123.

Truex D.P., Baskerville R. & Travis J. 2001. Amethodological systems development: The deferred meaning of systems development methods. Accounting, Management and Information technologies, 10(1), 53–79.

VersionOne, Inc. 2006. Survey: "The State of Agile Development" [online]. VersionOne, Inc [viitattu 29.7.2007]. Saatavilla www-osoitteessa <http://www.versionone.net/pdf/StateofAgileDevelopmentSurvey.pdf>.

Vinciguerra R. 2004. UML History [online]. Vinciguerra [viitattu 22.5.2005]. Saatavilla www-osoitteessa <http://www.vinci.org/uml/history.html>.

Visconti M. & Cook. C. 2004. An Ideal Process Model for Agile Methods. Teoksessa F. Bomarius & H. Iida (toim.) Proceedings of the 5th International Conference on Product Focused Software Process Improvement (PROFES 2004), Kansai Science City, Japan, April 5–8. Berlin: Springer-Verlag, 431–441.

Vriens C. 2003. Certifying for CMM Level 2 and ISO9001 with XP@Scrum. Teoksessa Proceedings of the Agile Development Conference (ADC '03), Salt Lake City, Utah, June 25–28. Washington, DC: IEEE Computer Society, 120–124.

Wake W. 2005. Overview of XP Explained, Second Edition [online]. Wake [viitattu 16.5.2005]. Saatavilla www-osoitteessa <http://www.xp123.com/xplor/xp0502/index.shtml>.

Wells D. 2004. Extreme Programming: A gentle introduction [online]. Wells [viitattu 27.8.2005]. Saatavilla www-osoitteessa <http://www.extremeprogramming.org/rules/userstories.html>.

Williams L. 2004. A Survey of Agile Development Methodologies [online]. Williams [viitattu 20.4.2007]. Saatavilla www-osoitteessa <http://agile.csc.ncsu.edu/SEMaterials/AgileMethods.pdf >.

Zhen R.D. 2004. Model-Driven Testing with UML 2.0. Teoksessa D.H. Akehurst (toim.) Second European Workshop on Model Driven Architecture (MDA)

with an emphasis on Methodologies and Transformations, Canterbury, England September 7–8. Canterbury: University of Kent, 179–187.