Riskienhallinta perinteisessä ja ketterässä ohjelmistokehityksessä

Tero Tapio

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Pirkko Nykänen
Toukokuu 2010

Haluan omistaa tämän Pro gradu-tutkielman vanhemmilleni, jotka ovat aina auttaneet ja kannustaneet minua kaikin mahdollisin tavoin. Ilman teitä, en olisi tässä.

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Tero Tapio: Riskienhallinta perinteisessä ja ketterässä ohjelmistokehityksessä
Pro gradu -tutkielma, 97 sivua
Toukokuu 2010

Tämä tutkielma käsittelee monipuolisen lähdekirjallisuuden pohjalta ketterien ohjelmistomenetelmien riskienhallintaa ja vertailee sitä, erityisesti perinteisten ohjelmistokehitysmenetelmien ja niiden riskienhallinta toimien kanssa.

Samalla kun erilaiset tekniset järjestelmät ja tietotekniikka yleensäkin ovat tulleet osaksi eri organisaatioiden jokapäiväistä liiketoiminta, on myös projektimuotoinen työskentely jatkuvasti lisääntynyt. Useimmilla teknisillä toimialoilla, se on kuitenkin jo pidemmän aikaa ollut enemmän sääntö kuin poikkeus, koska sen mukaan tuomat hyödyt ovat kiistattomat. Projektinhallinta- ja riskienhallinta tavat ovat kuitenkin vielä varsin puutteellisia, koska perinteisten ohjelmistokehitysmenetelmien avulla vain pieni osa lukuisista projekteista saadaan päätökseen vaaditussa aikataulussa, laaditussa budjetissa ja täysin tilaajan vaatimukset täyttävinä.

Perinteisissä ohjelmistokehitysmenetelmissä riskienhallinta on myös varsin hidas ja kankea prosessi. Kun taas, ketterät menetelmät pystyvät kevyen rakenteensa ansiosta huomattavasti nopeammin reagoimaan erilasiin odottamattomiin tai muuttuviin tilanteisiin. Ketterät menetelmät ovatkin koko 2000-luvun kasvattaneet suosiotaan, sillä niiden uskotaan johtavan projektien parempaan hallittavuuteen ja tätä kautta myös parempaa lopputulokseen.

Tämän tutkielman tutkimustulokset myös tukevat tätä väitettä. Ketterät ohjelmistokehitysmenetelmät tarjoavat projektille paremman hallittavuuden ja kontrollin, mikä johtaa entistä parempaan lopputulokseen. Projektin koko, laajuus ja aihepiiri kuitenkin vaikuttavat varsin merkittävästi ketteristä menetelmistä saatavien hyötyjen määrään. Tästä syystä kullekin projektille oikean ohjelmistokehitysmenetelmän valinta sekä riskienhallinta sen olennaisena osana, ovat keskeisessä osassa myös ketterien ohjelmistoprojektien onnistumisessa.

Avainsanat ja -sanonnat: Riskienhallinta, ketteräohjelmistokehitys, Scrum, XP, projektit, projektinhallinta.

Kiitokset

Haluan kiittää Professori Pirkko Nykästä erittäin joustavasta ohjauksesta. Professori Mikko Ruohosta monista mielenkiintoisista luennoista sekä inspiraatiosta, jotka johtivat ensin tietojärjestelmien maisteriohjelman valintaan ja lopulta myös tähän tutkielmaan. Haluan myös kiittää kaikkia opiskelutovereita monista hyvistä keskusteluista ja uusista näkökulmista eri asioihin.

Lisäksi haluan kiittää lukuisia ystäviä, joukkuetovereita sekä muita tuttavia, joiden kanssa pystyin hetkeksi unohtamaan opiskelun kiireet ja sain uutta energiaa jatkaa kohti uusia haasteita. Teitä on aivan liian paljon, jotta voisin mainita teidät kaikki. Kiitokset myös lukuisille joustaville yrityksille, joissa sain työskennellä opiskelun ohessa.

Erityiskiitokset äidilleni ja isälleni (1959–2004) sekä pikkuveljelleni, jotka jaksoivat aika-ajoin muistuttaa minua asioiden loppuunviemisen tärkeydestä.

Elsi, toit piristystä juuri silloin, kun sitä eniten tarvitsin. Kiitos!

Tampereen Kalevassa, toukokuussa 2010

Tero Tapio

"It must be remembered that there is nothing more difficult to plan,
more doubtful of success, nor more dangerous to manage than a new system.
For the initiator has the enmity of all who would profit by the preservation
of the old institution and merely lukewarm defenders in
those who gain by the new ones. "
— Niccolò Machiavelli / Ruhtinas (v.1513)

Sisällys

1. Johdanto

1.1. Tausta

Tämän tutkielman taustalla on opiskelun myötä herännyt kiinnostus projektimuotoiseen työskentelyyn. Opiskeluiden alussa erityisesti projektien korkea epäonnistumisprosentti herätti suurta ihmetystä. Sittemmin eri työtehtävissä lukuisat erilaiset projektien toteutustavat ja tekniikat ovat tulleet entistä tutummiksi, ja tämän myötä kiinnostus projekteja kohtaan on vain kasvanut. Pikku hiljaa myös projektien hallinto ja johtaminen ovat myös tulleet mukaan kuvaan. Tätä kautta erityisesti riskienhallinta on noussut yhdeksi suurimmista kiinnostuksen kohteista. Onnistuminen kaikessa tekemisessä on minulle itselleni kuitenkin erityisen tärkeää, niin myös työssä. Tästä huolimatta, omien projektien lopputulokset ovat kuitenkin vaihdelleet jonkin verran, vaikka kaikki projekteissa mukana olleet olisivatkin tehneet parhaansa.

Tässä ei sinänsä ole mitään erikoista. Sillä lähes poikkeuksetta, projektien luonteesta riippumatta, peruutuksia sekä täydellisiä tai osittaisia epäonnistumisia tapahtuu organisaatioissa jatkuvasti. Vaikka ohjelmistotuotanto sekä projektihallinta työkalut ja tekniikat ovat jatkuvasti kehittyneet entistä tehokkaammiksi. Monessa tapauksessa projektien ongelmat olisikin voitu välttää tai niiden aiheuttamia vaikutuksia olisi voitu konkreettisesti vähentää, jos projekteissa olisi koko niiden olemassa olon ajan tehty riskienhallintatyötä [Boehm, 1991].

Riskienhallinta onkin yksi projektien vaikeimmista ja samalla tärkeimmistä asioista [Pfleeger, 2000]. Sen onnistumisen varassa on monessa tapauksessa hyvin paljon. Riskienhallinnan vaikeus piilee ehkä osittain työn haasteellisuudessa sekä osittain siihen kuluvassa ajassa. Riskienhallinta on myös kokonaisuus, jossa mikään vaihe ei ole toista tärkeämpi [Kontio, 2001]. Tästä syystä riskienhallinnasta ei voi tehdä vain tiettyä osaa. Projektin alussa tehtävä riskien tunnistaminen on aivan yhtä olennainen osa riskienhallintaan kuin varsinainen riskien torjunta ja niihin reagoiminen.

Samalla kuitenkin riskienhallinta työtä laiminlyödään varsin usein tai sille ei anneta riittävästi painoarvoa projektinhallinnassa [Boehm, 1991]. Ehkä vielä osa projekteissa mukanaolijoista ei näe riskienhallintaa riittävän oleellisena osana projektia, koska se ei kokoajan liity suoraan heidän omiin käytännön työtehtäviinsä. Ennen kuin vasta siinä vaiheessa, kun jokin riskitapahtuma jo pakottaa tekemään konkreettisia muutoksia aikaisemmin tehtyihin projektin osiin.

Myös useimmissa ohjelmistotuotannon teoksissa [Haikala ja Märijärvi, 2004; Sommerville, 2007] riskienhallinta katsotaan vielä kuuluvan osaksi projektijohtajan tai -päällikön tehtäväkenttää. Vaikka todellisuudessa useimmiten parhaaseen tulokseen pääseminen edellyttääkin kaikkien projektissa mukana olijoiden aktiivista osallistumista riskienhallintaan tavalla tai toisella. Esimerkiksi projektin koodaajan tekemät aikaiset käytännön havainnot suunnittelun virheistä voivat olla ensiarvoisen tärkeitä projektin jatkon kannalta.

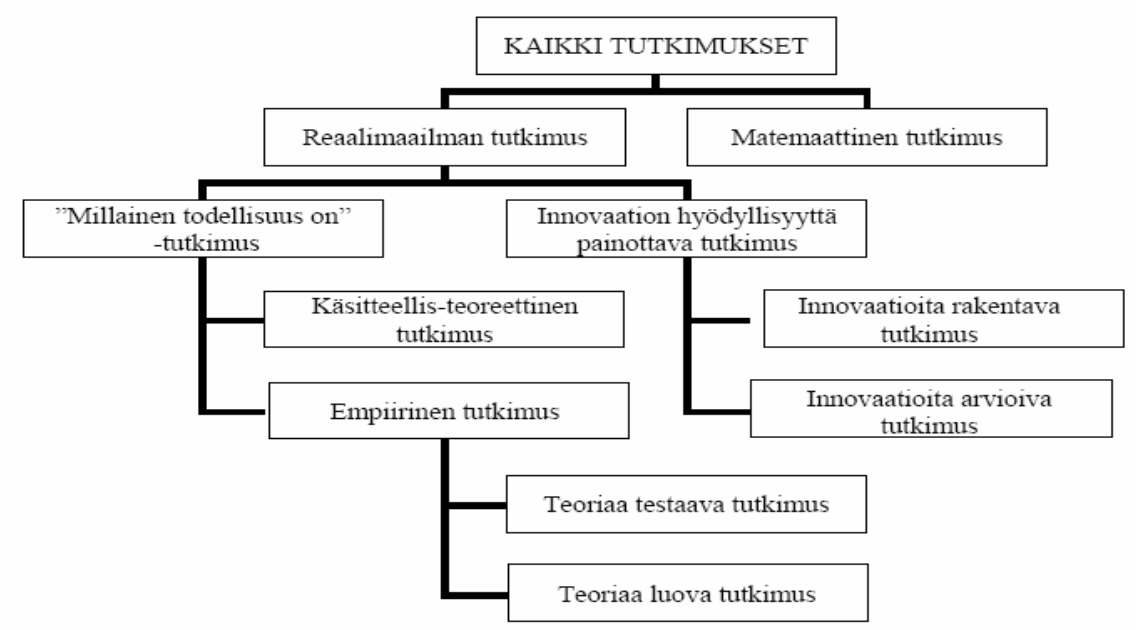
1.2. Tutkimusmenetelmä ja tavoitteet

Tässä tutkielmassa käsitellään perinteisiä ja ketteriä ohjelmistokehitysprojektien toteutustapoja sekä erityisesti niiden riskienhallintaa. Tavoitteena on kuvata erilaisten projektien riskienhallinnan perusteita ja käytännön toteutustapoja sekä löytää keinoja joiden avulla entistä suurempi osa projekteista saataisiin onnistuneesti valmiiksi.

Päädyin tähän jakoon, koska yksittäisen yrityksen sisälläkin voi olla yhtä aikaa käytössä tai käynnissä sekä perinteisiä, että ketteriä kehitysmalleja käyttäviä projekteja. Tämä johtaa siihen, että yksittäinen henkilö voi myös olla samanaikaisesti jäsenenä useassa projektissa yhtä aikaa, koska erityisesti IT-alalla kokeneet ja osaavat projektien ammattilaiset ovat erittäin kysyttyjä.

Tämän tutkimuksen teoriapohjana pyritään käyttämään kattavasti ja monipuolisesti kirjallisia lähteitä. Tutkimuksen tarkoituksena on perinteisten ohjelmistokehitysmenetelmien sekä ketterien menetelmien riskienhallinnan tutkiminen yhdistelemällä ja vertailemalla niistä lähteiden pohjalta saatavaa tietoa. Keskeisenä tutkimuksen kohteena ovat riskienhallinnan asiat, ilmiöt sekä käytännön toimet ja tavat, jotka mahdollistavat onnistuneen projektin.

Järvinen ja Järvinen [2004] esittävät kirjassa, tutkimustyön metodeista, tutkimusotteiden luokittelun. Tässä luokittelussa tutkimusmenetelmät jaetaan ensin kahteen pääluokkaan, reaalimaailmaa ja matemaattisia asioita käsitteleviin luokkiin. Tämän jälkeen reaalimaailmaa koskevat tutkimukset jaetaan taas kahteen alaluokkaan, todellisuutta ja innovaatioita tutkiviin luokkiin. Innovaatioihin keskittyvät tutkimukset jaetaan niiden toteutumista ja arviointia tutkiviin luokkiin. Todellisuuteen keskittyvät tutkimukset jaetaan käsitteellis-teoreettisiin ja empiirisiin luokkiin. Tämä on esitetty kuvassa 1.



Kuva 1: Tutkimusten luokittelu [Järvinen ja Järvinen, 2004, s.9]

Kuvan 1 luokittelun mukaan, tämä tutkielma voidaan jaotella kuuluvaksi käsitteellis-teoreettisten tutkielmien luokkaan, koska tutkimus perustuu reaalimaailman ilmiöiden kuvaukseen ja ainoastaan kirjallisiin lähteisiin. Eikä tutkimuksen yhteydessä myöskään ei tehdä empiiristä tutkimusta. Lisäksi tutkimusaineisto käsitellään vahvasti induktiivisella tutkimusotteella, jossa tavoitteena on yksittäistapausten kautta muodostaa tiettyjä yleistyksiä ja teorioita. Tutkielman varsinaisena tavoitteena on tutkia seuraavia kysymyksiä.

1. Onnistuvatko ketteriä ohjelmistokehitysmenetelmiä käyttävät projektit perinteisiä kehitysmenetelmiä käyttäviä projekteja useammin tai paremmin, miksi?

2. Onko ketterille ohjelmistokehitysmenetelmille olemassa niille tunnusomaisia riskienhallinnan toimintatapoja tai tekniikoita, joihin niiden menetys mahdollisesti perustuu?

Projekteista, perinteisistä ohjelmistokehitysmalleista ja riskienhallinnasta tietoa on saatavilla runsaasti eri aikakausilta. Sen sijaan, uudemmasta ketterästä ohjelmistokehityksestä tietoa on saatavilla vähemmän tai se ei ole ”puhdasta” tutkimustietoa. Monet ketteriä menetelmiä käsittelevistä julkaisuista tai niiden tekijöistä ovat jollain tavalla mukana ketterien menetelmien yhteenliittymässä Agile Alliancessa [2001], ja tätä kautta mainostamassa ketteriä menetelmiä. Tämä pyritään kuitenkin ottamaan huomioon tässä tutkimuksessa mahdollisimman hyvin. Lähteiden

poikkeuksellisen kriittinen tarkastelu ja niiden yhdistely tekee myös tästä tutkielmasta selkeästi muista aikaisemmista tutkimuksista poikkeavan.

1.3. Tutkielman rakenne

Tutkielma on jaettu kahdeksaan lukuun. Ensimmäisessä luvussa esitellään tutkielman aihetta yleisellä tasolla sekä kuvataan varsinaisen tutkimuksen rakennetta ja toteutustapaa.

Toisessa luvussa keskitytään projekteihin ja niiden eri ulottuvuuksiin, kuten henkilöihin ja projektinhallintaan. Lopuksi käsitellään projektien epäonnistumista.

Kolmannessa luvussa käsitellään riskien ja riskienhallinnan historiaa, perusajatuksia sekä lähtökohtia. Lopuksi tarkastellaan konkreettisia keinoja erilaisten projektien riskienhallintaan sekä tarkastellaan riskienhallinnassa viime vuosian tapahtunutta kehitystä.

Neljännessä luvussa syvennytään aluksi ohjelmistokehityksen taustaan, tarkoitukseen ja menetelmiin. Tämän jälkeen esitellään ja kuvataan yleisellä tasolla perinteisiä ohjelmistokehitysmenetelmiä. Lopuksi syvennytään tarkemmin niiden rakenteeseen ja käytännön toimintaan sekä sivutaan niissä käytettyjä riskienhallinta periaatteita.

Viidennessä luvussa käsitellään ketteriä ohjelmistokehitysmenetelmiä. Aluksi keskitytään ketterien menetelmien syntyyn sekä niiden taustaan ja yleisiin toimintaperiaatteisiin. Tämän jälkeen menetelmät jaotellaan omiksi kokonaisuuksiksi ja yksittäisten menetelmien toimintaan syvennytään tarkemmin. Lopuksi tarkastellaan menetelmissä käytettäviä riskienhallinnan periaatteita.

Kuudennessa luvussa vertaillaan perinteisten ja ketterien ohjelmistokehitysmenetelmien ominaisuuksia ja toimia rinnakkain erityisesti projektin- ja riskienhallinnan näkökulmista. Samalla tarkastellaan myös tutkielman alussa esitettyjen olettamusten täyttymistä.

Seitsemännessä luvussa lähdekirjallisuuden pohjalta saatuihin tuloksiin yhdistetään omaa pohdintaa sekä lisätään omaa tulkintaa. Samalla myös hahmotellaan joitakin yleisiä ohjeita ja tapoja, joiden avulla projektien onnistumisen olisi entistä todennäköisempää.

Kahdeksannessa luvussa tehdään yhteenveto koko tutkielman sisällöstä ja pääkohdista sekä hahmotellaan mahdollisia jatkotutkimus kohteita.

2. Projektit

2.1. Synty ja tausta

Erilaisia projekteja on varmasti ollut olemassa lähes aina. Ensimmäiset viittaukset projekteihin ja ”Project” sanaan löytyvät englanninkielestä 1300-luvulta ja latinasta paljon ennen tätä. Nykymuotonsa ja merkityksensä projektit saivat 1950- ja 60-luvun taitteessa [Ruuska, 2007; Pelin, 2008]. Suomen Standardisoimisliiton [1981] määritelmän mukaan, projektit ovat tiettyä tehtävää varten muodostettu tilapäinen kokonaisuus, jolle on etukäteen määritetty tietyt toimintarajat sekä ajalliset, taloudelliset ja laadulliset tavoitteet. Toimialasta ja tarkoituksesta riippumatta, kaikilla projekteilla on aina tiettyjä yhteisiä piirteitä. Tässä tutkielmassa keskitytään vain ohjelmistoprojekteihin.

Teknisillä aloilla projektit, tiimit sekä muut tarpeen mukaan koottavat työryhmät jonkin työtehtävän tai asian hoitamiseksi ovat nykyään erittäin tavallinen työskentelytapa. Mutta erityisesti juuri ohjelmistokehitystä pidetään erittäin projektikeskeisenä, joskus jopa riippuvaisena, toimiala. Ohjelmistotaloissa tai organisaatioiden IT-osastoilla tämä voi johtaa jopa niin sanottuun projektiputkeen, jossa yhtä valmistunutta projekti seuraa aina uusi projekti. Tämä tuo mukanaan myös aivan uusia projektinhallinnan ongelmia [Olson, 2008; Ruuska, 2007].

Useimmille projekteille on tyypillistä niiden ainutkertaisuus ja kertaluonteisuus. Muita projekteille tyypillisiä ominaisuuksia ovat laajuus ja monimutkaisuus [Sommerville, 2007]. Jokainen projekti on siis aina sen alussa jollain tavalla uniikki ja abstrakti. Tyydyttävään lopputulokseen pääseminen edellyttää useimmiten aikaisempien toimintatapojen muuttamista aina kyseisen projektin vaatimuksiin sopiviksi [Ruuska, 2007; Pelin, 2008]. Lisäksi projektit toteutetaan monessa tapauksessa ympäristössä, joka muuttuu ja sisältää samalla useita epävarmuustekijöitä. Juuri nämä asiat tekevätkin projekteista niin haastavia kuin ne tilastojen valossa ovat [Chaos report, 2009].

Tämä voi myös olla seurausta siitä, että usein projektin alussa myöskään kaikki vaatimukset toivotun lopputuloksen suhteen eivät ole vielä täysin selvillä. Koska organisaatiot pyrkivät olemaan koko ajan mahdollisimman tehokkaita, ja kaikkien organisaation resurssien sitominen yhteen paikkaan ei ole järkevää tai edes mahdollista, niinpä erilaiset projektit ovat niille keino kehittää omaa toimintaansa ja tuotteitaan jatkuvasti ilman kohtuutonta riskiä [Kontio, 2001; Ropponen, 1999].

Tällaisessa toimintaympäristössä organisaation ammattilaisten yhtyeenkokoaminen voi olla hyvän lopputuloksen edellytys. Niitä keinoja joilla projekteja pyritään johtamaan ja hallitsemaan kutsutaan projektinhallinnaksi. Riskienhallinta on perinteisesti ollut yksi projektinhallinnan osa-alueista. Sen muut osa-alueet ovat suunnittelu, aikataulutus, sekä

muut toimet [Sommerville, 2007, Pelin, 2008]. Projektien epäonnistumisen syynä ovat pääsääntöisesti joko suoraan projektinhallintaan liittyvät tekijät esimerkiksi riskienhallinnan laiminlyönnit, aikataulutuksen pettäminen ja vaatimusten kontrolloimattomuus, tai odottamattomat riskit tapahtumat [Boehm, 1989; 1991; Kontio, 2001; Lyytinen and Hirschheim, 1987; Ropponen, 1999].

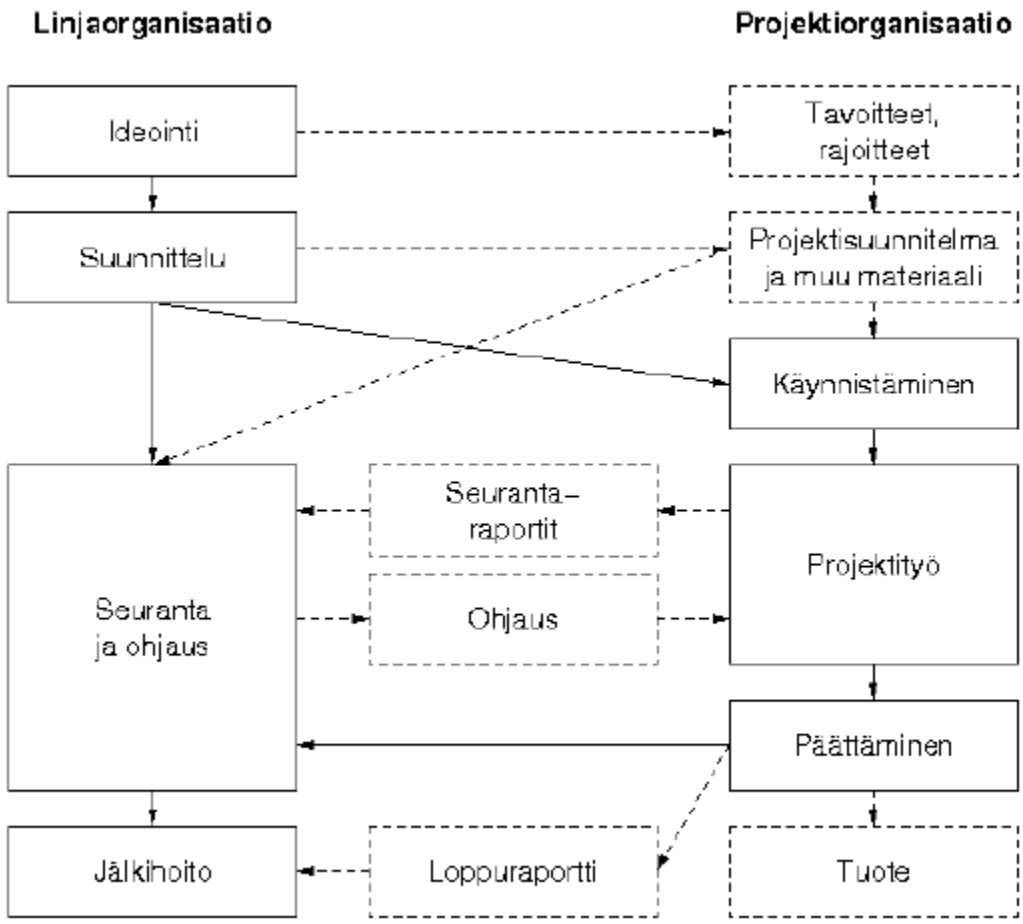
2.2. Projektit ja organisaatio

Tyypillisesti koko organisaation henkilöstö ei ole mukana jokaisessa projektissa. Organisaation johto delegoi joko koko projektitehtävän, tai jos kyseessä todella suuri projektihanke, sen osakokonaisuuden hoitamisen pienemmälle ryhmälle. Kun annettu tehtävä on suoritettu, niin projektiryhmä puretaan ja projekti päättyy [Ruuska, 2007].

On myös varsin tavallista, että organisaatiossa voi olla samaan aikaan käynnissä useita erikokoisia ja erityyppisiä projekteja. Yksi henkilö voi myös olla usean projektin jäsenenä samaan aikaan. Tämä aiheuttaa sen, että eri projektit kilpailevat jatkuvasti organisaation käytössä olevista sisäisistä ja ulkoisesta voimavaroista muiden projektien kanssa [Pelin, 2008].

Syitä tähän kehitykseen voidaan löytää monia, mutta monessa tapauksessa taustalla on resurssien kohdentaminen ja maksimaalinen tehokkuus sekä henkilöiden osaamisen hyödyntäminen. Myös organisaatioiden toimintaympäristössä tapahtuva jatkuva muutos ja myllerrys ovat varmasti myös edesauttaneet tätä kehitystä.

Suuret ja jäykät organisaatiorakenteet, linjaorganisaatiot, eivät aina pysty muuttumaan projektien vaatiman tilanteen mukaan riittävän nopeasti. Linjaorganisaatiot ovat parhaimmillaan kun ne pääset suorittamaan kestoltaan ja toiminnaltaan ennalta määrättyjä tehtäviä, joissa tapahtuu vain vähän odottamattomia muutoksia. Usein raskaan linjaorganisaation osa muuttuukin omaksi projektiorganisaatioksi, jolloin organisaation toiminta projektin osalta muuttuu vastamaan paremmin sen luonnetta ja tarpeita. Tämä muutos ja sen vaikutukset on esitetty kuvassa 2 [Markus, 2004; Pelin, 2008; Ruohonen ja Salmela, 2005].



Kuva 2: Linja- ja projektiorganisaatiot samaan aikaan yrityksessä

Kuvassa 2 on hahmoteltu projektin toiminta samaan aikaan sekä linjaorganisaation että varsinaisen projektinorganisaation alaisuudessa. Tämä muistuttaa hyvin paljon linjaorganisaatiota modernimpaa matriisiorganisaatiota, jossa jokaisella henkilöllä on aina kaksi, vertikaalinen ja horisontaalinen, esimiestä. Riippuen siitä, onko projektin toiminnan pääasiallinen valta ja vastuu projektilla itsellään vai linjaorganisaatiolla, voidaan toisistaan erottaa heikko projektiorganisaatio, jossa organisaatiolla on päätösvalta ja vahva projektiorganisaatio, jossa projektilla on päätösvalta toiminnasta [Markus, 2004; Pelin, 2008; Ruohonen ja Salmela, 2005; Ruuska, 2007].

Lisäksi kuvasta 2 voidaan nuolien avulla myös nähdä, se miten tieto, käskyt ja dokumentit liikkuvat organisaation sisällä. Tämän muutoksen mukanaan tuomat uudet toimintatavat ja konfliktit voivat rasittaa linja organisaatioiden henkilöstö monin eri tavoin [Markus, 2004; Higuera and Haimes, 1996]. Tämän välttämiseksi, useimpien pääasiallisesti projekteihin keskittyvien organisaatioiden, esimerkiksi ohjelmistotalojen ja konsulttiyritysten organisaatiorakenteet ovatkin jo valmiiksi matriisi- tai

projektiorganisaatioita, jolloin tätä organisaation rakennemuutosta projektia varten ei enää tarvita [Pelin, 2008].

Linjaorganisaatioille projektit ovatkin kuitenkin selkeästi joustavampi ja tehokkaampi tapa hoitaa joitakin tietyntyyppisiä tehtäviä. Hyvin usein näiden organisaatioiden tuotekehitys tai tutkimus toiminta on juuri tästä syystä projektimuotoista [Ruohonen ja Salmela, 2005; Ruuska, 2007; Pelin, 2008]. Projektiorganisaatioissa turha hierarkia on karsittu pois, jolloin vuorovaikutus projektin sisällä on välittömämpi. Myös vastuu ja valta suhteet ovat projekteissa huomattavasti selkeämmät kuin suurissa organisaatioissa.

2.3. Projektin tyyppi ja toteutustavat

Erilasia projekteja on käynnissä jatkuvasti. Nykypäivänä kaikki organisaatiot luottavat yhä enemmän tietotekniikkaan omassa liiketoiminnassaan, niinpä projekteilta ei voi juuri välttyä toimialasta riippumatta. Vaikka kyseessä ei olisi oman organisaation sisäinen projekti, voi silti projektin osaksi joutua myös tahtomattaan. Esimerkiksi tilanteessa jossa ulkopuolinen toteuttaja tulee organisaatioon tekemään uuden kehitteillä olevan järjestelmän vaatimustenmäärittelyä.

Projektit voidaan jaotella ja luokitella monilla eri tavoilla. Luokittelun perusteena voivat olla esimerkiksi koko, ajallinen kesto, vaikeusaste tai perusluonne. Luokittelun yhteenveto ja sen käytännön esimerkit on kuvattu seuraavassa [Haikala ja Märijärvi, 2004; Pelin, 2008; Ruuska, 2007; Sommerville, 2007].

- **Koko:**
  - Pieni; 1-5 henkilöä
  - Keskikokoinen; 5-15 osallistujaa
  - Suuri; kymmeniä osallistujia yhdestä tai useasta organisaatiosta
- **Ajallinen kesto:**
  - Lyhyt; alle 2 kuukautta
  - Normaali; 3-6 kuukautta
  - Pitkä; yli 1-5 vuotta
- **Vaikeusaste:**
  - Helppo; vanhan kertausta tai sen muuttamista
  - Normaali; osallistujille tyypillinen perus projekti
  - Vaikea; uusi tai tuntematon toteutusalue

- **Luonne:**
  - Tekninen; laitteisto, ohjelmisto, tietojärjestelmä
  - Ei-Tekninen; käyttöönotto, koulutus, hankinta, laatujärjestelmän arviointi, investointi
  - Tuotannollinen; prototyypin toteutus ja suunnittelu, toimitus
  - Tutkimus ja kehitys; esitutkimus, määrittely, kartoitus, tuotekehitys

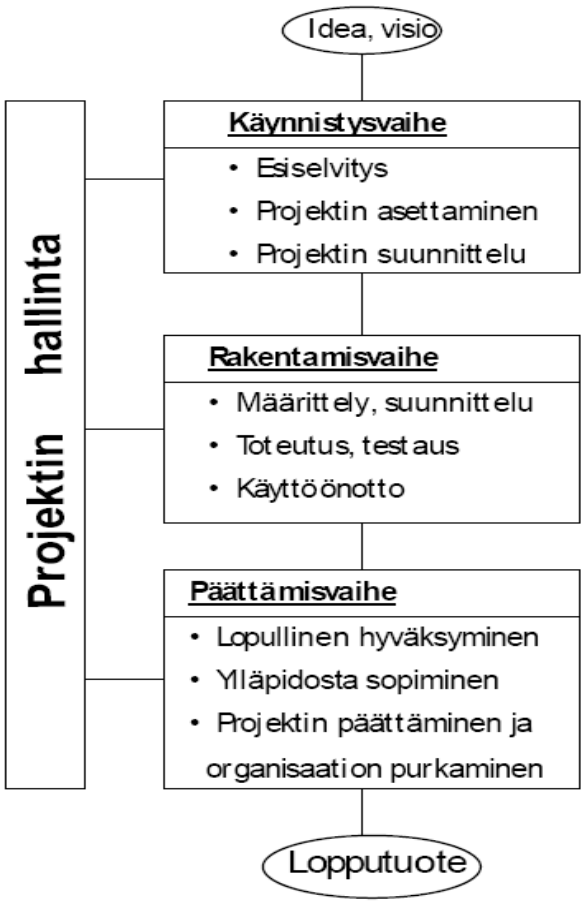
Projekti voidaan toteuttaa joko kokonaan organisaation sisällä ilman ulkoista apua tai käyttämällä organisaation ulkopuolisia asiantuntijoita. Mikäli organisaatiolla itsellään on käytössä tarpeeksi resursseja sekä riittävä osaaminen projektia koskevasta aihealueesta, niin on erittäin tärkeää, että niitä myös käytetään. Näin säilytettään ensinnäkin organisaation uskottavuus sen työntekijöiden silmissä ja toiseksi osoitetaan luottamusta ja arvostusta omia työntekijöitä kohtaan [Taylor, 2006].

Lisäksi alihankintana tai muuten pääosin ulkoisin voimin läpi viety projekti on huomattavasti riskialttiimpi kuin pääsääntöisesti yrityksen sisällä toteutettu projekti. Ulkoistettujen projektien hallintaan ja laadun kontrollointiin liittyy monia erilaisia ongelmia, joita ei ole pystytty varsinkaan perinteisiä ohjelmistokehitysmenetelmiä käyttävissä projekteissa onnistuneesti ratkaisemaan [Taylor, 2006; Williams, 1997].

Projektin toteutustapa vaikuttaa myös huomattavasti muiden projektille tarpeellisten resurssien saatavuuteen sekä projekteille tehtävien rajoitteiden eli aikataulun, henkilöstön ja rahoituksen, muodostumiseen. Tämä taas näkyy suoraan projektin tavoitteiden sekä toivotun lopputuloksen määrittelyssä. On erittäin tärkeää, että projektin tavoite on linjassa sen saamien resurssien kanssa. Tällöin projektilla on edes teoriassa mahdollisuus onnistua [Sommerville, 2007; Wallace and Keil, 2004].

2.4. Projektin vaiheet

Kaikki projektit alkavat ja päättyvät tiettynä ajankohtana. Projekti voidaankin nähdä vertikaalisena suorana tai elinkaarena, jolla on alku ja loppupiste. Projekti voidaan myös jakaa kolmeen vaiheeseen ja niiden eri tehtäviin. Ensimmäinen vaihe on käynnistysvaihe, jota seuraavat rakentamis- ja päättämisvaiheet. Kaikki projektin vaiheet ja niiden tehtävät on esitetty kuvassa 3 [Haikala ja Märijärvi, 2004; Ruohonen ja Salmela, 2005; Ruuska, 2007].



Kuva 3: Projektin vaiheet ja tehtävät yleisellä tasolla [Ruuska, 2007, s.34]

Kuvasta 3 voidaan nähdä, että projektin tuotteen kehitys alkaa tarpeesta tai ideasta/visiosta jonkin asian tai tilanteen hoitamiseksi. Ajan myötä tämä idea tarkentuu ja konkretisoituu kun sen pohjalta syntyy päätös projektin aloittamisesta.

Varsinainen projekti käynnistetään esitutkimuksella (esiselvitys). Esitutkimuksen avulla on tarkoitus selvittää, onko esiin nousseen asian tai ongelma hoitamiseksi edes mahdollisuuksia. Samalla muodostetaan myös projektin yleiset toimintaehdot ja rajat. Esitutkimuksessa kerätyn tiedon pohjalta projektin tilaaja eli asiakas voi tehdä päätöksen projektin toteuttamisesta tai toteuttamatta jättämisestä. Päätös prosessi voi vaihdella organisaation mukaan, mutta sen tarkoituksena on punnita projektista saatavia etuja ja verrat niitä mahdollisiin haittoihin tai riskeihin. Samalla myös kartoitetaan alustavasti käytössä olevia resursseja (henkilöt, budjetti ja työmäärä) sekä aikataulua. Kun projektin edut katsotaan haittoja suuremmiksi, projekti toteutetaan (asetetaan) [Ruuska, 2007].

Tämän jälkeen projektin jatkuu rakentamisvaiheella ja sen määrittely sekä suunnittelutehtävillä. Aluksi projektin kaikki olennaiset yksityiskohdat määritellään riittävään tarkkuuteen asti, jonka jälkeen suunnitellaan niiden toteutus. Kun kaikkien

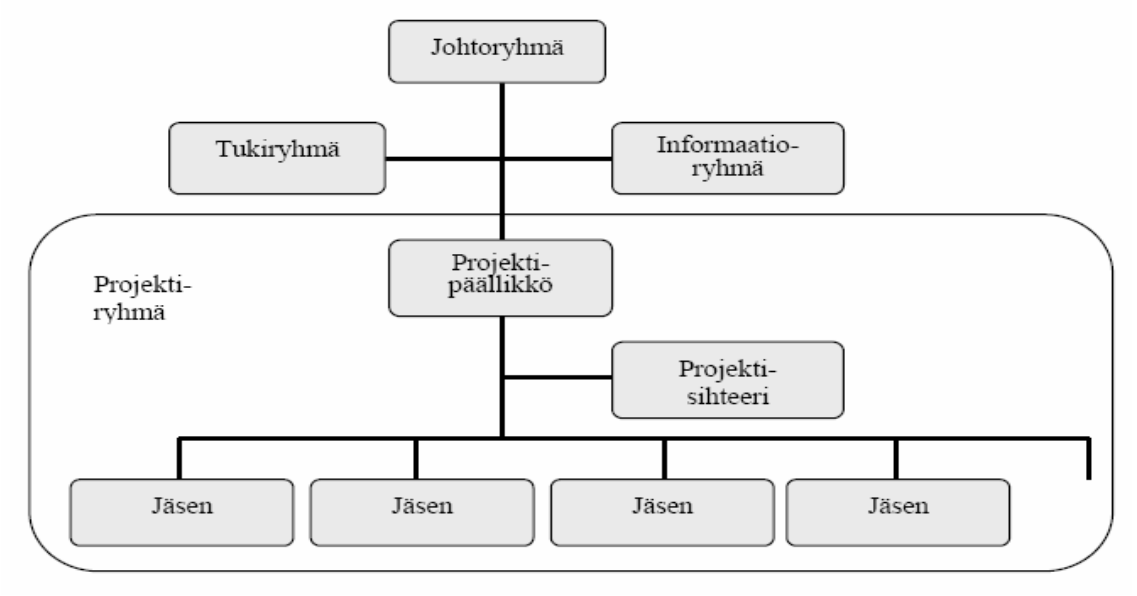
vaadittavien elementtien toteutus on saatu suunniteltua, voidaan ne toteuttaa ja niiden oikea toiminta testata. Lopuksi niistä kootaan valmis ohjelma joka otetaan testikäyttöön [Haikala ja Merijärvi, 2004; Ruuska, 2007].

Projekti viimeistellään ja lopetetaan päättämisvaiheessa. Projektin henkilöstö ei kuitenkaan voi yksipuolisesti päättää kehitettävän tuotteen olevan valmis. Ohjelmistotuotteen lopullinen hyväksyminen tapahtuu aina tilaajan kanssa yhteistyössä, samalla kun sovitaan mahdollisesta tuotteen ylläpidosta. Käytännössä projektit kuitenkin lopetetaan vasta silloin, kun toteutettu ohjelmistotuote vastaa tilaajan mielestä sille asetettuja vaatimuksia, ja he ovat siihen tyytyväisiä. Kun varsinaisten toteuttajien ja tilaajan mielipiteet tai näkemykset lopputuotteesta eroavat toisistaan merkittävästi, voidaan projektin toteutukseen vielä joutua tekemään suuriakin muutoksia. Pääsääntöisesti nämä muutokset yleensä jäävät tilaajan maksettavaksi, ellei mahdollisessa projektisopimuksessa muuta sovita. Yleensä tällaisessa tilanteessa myös ohjelmiston täydellinen käyttöönotto viivästyy [Haikala ja Merijärvi, 2004; Ruuska, 2007].

2.5. Projektin henkilöstö

Projekti on määräaikainen henkilöiden yhteenliittymä, joka on koottu hoitamaan vain tietty tehtävä. Henkilöt ovat siis yhdessä vain määräajan, minkä jälkeen he siirtyvät joko uuteen projektiin tai palaavat omiin varsinaisiin tehtäviinsä. Projektissa mukanaolevien henkilöiden määrätä voivat vaihdella projektin vaiheen ja tehtävien mukaan. Projektista riippuen tämä voi tarkoittaa sitä, että projektin alussa siinä työskentelee vain muutama henkilö, mutta suunnittelu- ja toteutusvaiheissa henkilömäärä kasvaa [Pelin, 2008].

Tämä edellyttää jokaisessa vaiheessa kaikkien projektissa mukanaolijoiden toimimista mahdollisimman saumattomasti yhdessä. Jotta tämä olisi mahdollista, on projekteissa mukana olevilla henkilöillä erilaisia rooleja sekä eri tehtäviä. Projekteissa mukanaolevat henkilöt on esitetty kuvassa 4 [Pelin, 2008; Ruuska, 2007].



Kuva 4: Projektin henkilöt [Pohjonen, 2002, s.55]

Kuvasta 4 voidaan nähdä, että projektiin kuuluu kaksi osaa, johtoryhmä ja varsinainen projektiryhmä. Yhdessä nämä muodostavat kokonaisen projektiorganisaation.

Tuki- ja informaatioryhmä ovat tyypillisesti mukana vain keskimääräistä vaativimmissa projekteissa. Yleensä tukiryhmä kootaan silloin kun, projektin oletetaan tarvitsevan jotakin erikoisosaamisesta, esimerkiksi alan asiantuntijoiden apua vaativaan asian tai toteutustehtävän hoitamiseksi. Informaationryhmä kokoaminen tulee ajankohtaiseksi silloin, kun johtoryhmän toimintaedellytykset halutaan varmistaa. Tällöin ryhmän keskeisenä tehtävä on toimia tiedonvälittäjä projektin eri yksiköiden ja johtoryhmän välillä [Pohjonen, 2002; Ruuska 2004].

Koko projektin ylin päättävä taho on sen johtoryhmä. Johtoryhmästä voidaan myös käyttää nimitystä ohjausryhmä. Johtoryhmä koostuu pääsääntöisesti sekä kehittäjän (toimittaja) että projektin tilaajan (asiakas) edustajista. Jos tilaaja ja toimittaja ovat saman yrityksen sisältä, esimerkiksi eri osastot (markkinointi ja tietotekniikka), niin tällöin molempien osastojen edustajat muodostavat johtoryhmän. Johtoryhmän koko voi vaihdella, 1-8 henkilön välillä, projektin koon ja tarpeen mukaan [Artto *et al*, 2008; Pohjonen, 2002; Ruuska, 2004].

Johtoryhmän keskeisiin tehtäviin kuuluvat erityisesti projektisuunnitelman hyväksyminen ja projektin edetessä siihen tehtävien muutosten käsittely eli hyväksyminen tai hylkääminen sekä tarvittaessa myös projektin lopettamisesta tai jatkamisesta päättäminen. Lisäksi yleinen projektin edistymisen seuranta kuuluu johtoryhmälle. Projektilla on suurempi todennäköisyys onnistua, niin kauan kun sillä on johtoryhmän

luottamus ja tuki. Jatkuva ja suora kommunikaatio projektipäällikön ja johtoryhmän välillä auttaa myös merkittävästi mahdollisten ongelmatilanteiden hoidossa [Pelin, 2008; Pohjonen, 2002; Ruuska, 2004].

Projektipäällikkö johtaa varsinaista projektiryhmää ja sen tekemistä. Projektipäällikkö on myös vastuussa johtoryhmälle projektin laadusta ja onnistumisesta. Yleensä johtoryhmä luottaa kokeneen projektipäällikön kokemukseen hyvin paljon, ja kuulee tätä ennen isoja päätöksiä. Projektipäällikön keskeisiin työtehtäviin kuuluvat projektin aikataulusta, kustannuksista ja resursseista vastaaminen (projektisuunnitelma) sekä projektin läpiviennin johtaminen (suunnitelman toteuttaminen) ja viestintä eri tahojen kanssa (raportointi). Projektipäällikkö siis johtaa projektin hallintoa, mutta joissain tapauksissa projektipäälliköllä voi myös olla käytännön toteutukseen liittyvä työtehtäviä [Pelin, 2008; Pohjonen, 2002; Ruuska, 2004].

Hyvän projektipäällikön ominaisuuksista on lähes kaikilla projekteista ja niiden johtamisesta kirjoittavilla jonkinlainen mielipide. Seuraavassa on koottu lyhyt yhteenveto, näistä hyvälle projektipäällikölle tarpeellisista ominaisuuksista [Artto *et al*, 2008; Haikala ja Märijärvi, 2004; Pelin, 2008; Pohjonen, 2002; Projektiyhdistys, 2008; Ruuska, 2004; Sommerville, 2007].

- **Avoimuus**
  - avomielisyys, hyvät ihmissuhdetaidot
- **Asenne**
  - tahto, halu, positiivisuus, sitoutuminen, vahva itsetunto
- **Innovatiivisuus**
  - luovuus, uusien ideoiden tukeminen
- **Johtamiskyky**
  - tiimit, ilmapiiri, yhteistyö, neuvottelutaito
- **Joustavuus**
  - erilaisuus, täsmällisyys, multitasking
- **Kommunikaatio**
  - yhteistyö, luottamus, viestintä, esiintymistaito
- **Riskien hallitsemine**n

- riskien hyväksikäyttö, mahdollisuudet, rohkeus, uskallus

- **Tehokkuus**
  - tuottavuus, oikea toiminta, laatu, päämäärätietoisuus

Vaikka oikeiden tai tarpeellisten ominaisuuksien suhteen mielipiteet hieman vaihtelevatkin, niin yhtä mieltä kaikki ovat kuitenkin yleensä siitä, että projektipäällikkö voi vaikuttaa omalla panoksellaan ja osaamisellaan erittäin ratkaisevasti projektin onnistumiseen.

Varsinaisen projektiryhmän muodostavat projektipäällikkö ja joskus hänen apunaan toimiva projektisihteeri sekä varsinaiset ryhmän jäsenet. Projektisihteerin tehtäviin kuluu jokapäiväisistä rutiineista ja käytännön asioista huolehtiminen. Näitä voivat olla projektin kokousten kirjaaminen ja muu dokumentointi, erilaiset ylläpitotehtävät sekä joukko juoksevia hallinnollisia tehtäviä [Pelin, 2008; Ruuska, 2004].

Projektin jäsenten tehtävät ja muu vastuunjako määräytyvät osallistujien oman kiinnostuksen tai osaamisen mukaan. Yleensä projektiin otetaan mukaan vaihteleva määrä suunnittelijoita, toteuttajia ja testaajia. Myös varsinainen jäsentenvälinen käytännön työnjako sovitaan yleensä varsin tarkasti. Tämä tarkoittaa sitä, että tietty henkilö tekee toteutuksessa jonkin tietyn osan esimerkiksi funktion, olion tai moduulin jollakin määrätyllä tavalla, jotta se sopisi muiden osien kanssa mahdollisimman saumattomasti yhteen [Pelin, 2008; Ruuska, 2004].

2.6. Projektinhallinta

Projektinhallinnalla on erittäin merkittävä vaikutus projektin onnistumisessa. Projektihallinnan voidaan katsoa tarkoittavan kaikkia niitä johtamistapoja ja toimia, joiden soveltamisen avulla pyritään varmistamaan projektin tavoitteiden ja päämäärän saavuttaminen onnistuneesti. Projektinhallinta on samalla sekä ihmisten että asioiden johtamista. Lisäksi siihen voidaan myös laskea kuuluvan erilaisia projektin onnistumista seuraavia ja mittaavia tekniikoita ja standardeja. [Artto *et al*, 2008, Ruuska, 2004]

Projektinhallintaa voidaan tarkastella useista eri näkökulmista eri toiminta- ja tietoalueiden tärkeyttä painottaen. Kirjassa Projektiliiketoiminta, Artto *et al* [2008] esittelevät oman projektinhallinta mallinsa. Tässä mallissa, projektinhallinta on jaettu kolmeen erilaiseen näkökulmaan.

Ensimmäinen näkökulma tarkastelee projektinhallintaa osaamisena ja ominaisuuksina. Tällä tarkoitetaan, projektinhallinnasta vastaavan henkilön tarvitsemia tietoja, taitoja,

asenteita sekä henkilökohtaisia ominaisuuksia. Toinen näkökulma tarkastelee projektinhallintaa ohjeistuksen ja välineiden kautta. Tällä tarkoitetaan, kaikkia niitä projektinhallinnan dokumentteja, ohjeita sekä teknisiä työvälineitä, joita käyttämällä ja soveltamalla projekteja hallitaan. Kolmas, ja samalla tunnetuin näistä näkökulmista, tarkastelee projektinhallintaa tietoalueina. Käytännössä tällä tarkoitetaan projektin jakamista strategisiin tietoalueisiin ja käytäntöihin. Pienempien tietoalueiden hallinnalla voidaan suoraan vaikuttaa koko projektin onnistumiseen [Artto *et al*, 2008].

Projektihallintaan on myös standardisoitu ja erilaisia sertifikaatteja onkin olemassa varsin runsaasti. Yksi tunnetuimmista ja laajimmin käytetty näistä, sekä Suomessa että muualla maailmassa, on Yhdysvaltalaisen Project Management Instituten PMBOK-standardi ja sen ohjeistus [Artto *et al*, 2008; PMBOK Guide, 2008]. Tässä standardissa projektinhallintaa tarkastellaan yhtä aikaa yhdeksänä prosessina ja tietoalueena, jotka vaikuttavat kiinteästi projektin menestykseen. Nämä yhdeksän osa-aluetta on esitetty kuvassa 5.

Kuva 5: Projektinhallinnan tietoalueet [PMBOK Guide, 2008, s.7]

Kuvasta 5 voidaan nähdä, PMBOK-standardin yhdeksän osa-aluetta, jotka ovat integraation hallinta, laajuuden hallinta, aikataulun hallinta, kustannusten hallinta, laadun hallinta, henkilöstön hallinta, viestinnän hallinta, riskienhallinta ja hankintojen hallinta. Onnistuneen projektin mahdollistava, projektinhallinnan kokonaisuus muodostuu näistä projekteille ominaisista toimintatavoista, menetelmistä ja työkaluista [Artto *et al*, 2008; PMBOK Guide, 2008].

Jokaisen yhdeksän projektinhallinnan osa-alueen tarkempi tietosisältö ja toiminnan kuvaus on seuraavassa.

1. Integraation hallinta eli kokonaisuuden hallinta, tarkoittaa projektin kokonaisuuden johtamista ja sen yhtenäistä koordinointia, jolloin projektin vaatimukset on mahdollista saada toteutettua. Tärkein työväline, minkä avulla projektin eri osa-alueita, tavoitteita, niiden muutoksia sekä näiden riippuvuuksia toisiinsa hallitaan, on projektisuunnitelma. Hyvä ja tehokas projektisuunnitelma on aina ajan tasalla. Tämä tarkoittaa projektin tilan jatkuvaa aktiivista seuraamista ja muutos kirjaamista. Tätä kautta projektisuunnitelmaa on mahdollista käyttää johtamisen työvälineenä. Projektin kokonaisuuden hallinta on yksi projektipäällikön keskeisimmistä työtehtävistä. Yleensä projektipäällikön vastuulla on myös projektisuunnitelman pitäminen ajan tasalla [Artto *et al*, 2008; PMBOK Guide, 2008].

2. Laajuuden hallinta, tarkoittaa kaikkia niitä käytännön työtehtäviin liittyviä asioista joiden tarkoituksena on varmistaa, että kaikki projektin vaatimukset tulevat täytettyä ilman ylimääräistä työtä. Laajuuden hallinnassa auttaa perusteellisen vaatimusmäärittely tekeminen. Jotta vaatimusmäärittelyä voidaan hyödyntää tehokkaasti, on siinä kuvattava toteutettavan tuotteen ominaisuuden ja toiminnallisuudet riittävän tarkasti, unohtamatta kuitenkaan toimintojen tehokkuutta ja lopputuotteen suorituskykyä. Lisäksi laajuuden hallintaan kuuluu olennaisena osana myös työnositus (WBS, Work Breakdown Structure) eli projektin jakaminen pienempiin ja paremmin hallittaviin osiin, jonkin ohjelmistokehitysmallin mukaisesti [Artto *et al*, 2008; PMBOK Guide, 2008].

3. Aikataulun hallinta, tarkoittaa projektille sopivan ja tehokkaan ajankäytön suunnittelua. Ajankäytön suunnittelussa voidaan käyttää erilasia malleja ja tekniikoita. Tunnetuimpia näistä ovat Gantt-kaaviot, PERT-kaaviot (Program Evaluation and Review Technique) sekä CPM-polut (Critical Path Method). Kaikkien näiden mallien perustana on työn ositus. Muita aikataulun suunnitteluun vaikuttavia tekijöitä ovat tehtävien jako, kesto, muutosten hallinta, riippuvuuksien määrittely ja aikataulun avulla

tehtävä työnohjaus sekä projekti kokonaistavoitteet.. Aikataulun hallinnan tarkoituksena on varmistaa projektin valmistuminen aikakataulussa [Artto *et al*, 2008; PMBOK Guide, 2008; Sommerville, 2007].

4. Kustannusten hallinta, tarkoittaa kaikkia niitä toimenpiteitä, joiden tarkoituksena on varmistaa projektin onnistuminen sekä samalla pysyminen suunnitellussa budjetissa. Projektien kustannusten hallinnassa käytetään yleisesti kustannusarviota, johon kuuluu projekti budjetin suunnittelu ja määrittäminen sekä sen seurantatoimet. Jotta yksittäinen projekti tai projektiliiketoiminta olisi organisaatiolle kannattavaa pitkällä aikavälillä, on projektin tuottojen ja kustannusten oltava järkevässä suhteessa toisiinsa. Tämä on mahdollista varmistaan vain projektin pysymisellä budjetissa [Artto *et al*, 2008; PMBOK Guide, 2008].

5. Laadun hallinta, tarkoittaa kaikki niitä toimia joiden avulla varmistetaan, että projekti täyttää kaikki sille asetetut tarpeet täysimääräisesti. Jotta tämä olisi mahdollista, täytyy tilaajan tarpeet ja odotukset ensin tunnistaa ja tämän jälkeen kuvata mahdollisimman tarkasti, jotta niitä voidaan toteuttaa suunnitelmallisesti. Laatu ja sen eri ulottuvuudet, kuten suunnittelu, varmistus, valvonta sekä virheettömyys, luotettavuus, oikea toiminta ja käytettävyys ovat tässä keskeisessä osassa. Laadun hallinnassa yhdistyvät asiakkaan vaatimusten täyttäminen eli projektin lopputuotteen laatu ja suunnitelmallisuus eli projektinhallinnan laatu [Artto *et al*, 2008; PMBOK Guide, 2008].

6. Henkilöstön hallinta, tarkoittaa projektin henkilöstö resurssien mahdollisimman tehokas hyväksikäyttöä koko projektin ajan. Tähän liittyvät keskeisesti projekti alussa tehtävä käytettävissä olevien henkilöiden kartoittaminen ja oikeiden henkilöiden rekrytointi projektin käyttöön. Projektin käynnistyessä ajankohtaiseksi tulee, henkilöiden tehtävien ja vastuualueiden jakaminen sekä hyvän toimintaympäristön luominen ja sujuvan yhteistoiminnan mahdollistaminen. Projektiryhmän organisoinnissa oikean ryhmädynamiikan saavuttaminen voi paikoitellen olla erittäin haasteellista, koska ideaalissa tilanteessa ryhmä jokaisella

jäsenellä on juuri heitä tyydyttävä roolia ja vastuut [Artto *et al*, 2008; PMBOK Guide, 2008].

7. Viestinnän hallinta, tarkoittaa kaiken projektin kuuluvan sisäisen (projektiryhmä) ja ulkoisen (sidosryhmät) viestinnän sekä vuorovaikutuksen varmistamista ja näiden toimintatapojen yhtenäistämistä. Projekti tuottaa käynnistämisen jälkeen runsaasti erilaista tietoa. Tieto siirtyy eri tahojen välillä sekä suullisesti että kirjallisesti. Kaiken tarpeellisen tiedon kerääminen dokumentteihin on projekti viestinnän perusta, koska kirjallinen tieto on helpommin jaettavissa sitä tarvitseville tahoille. Samalla kirjallinen viestintä toimii myös laadunhallinnan välineenä ja abstraktin toiminnan konkretisoijana. Lisäksi kirjallista tietoa on mahdollista käyttää pidemmällä aikavälillä organisaation oppimisen välineenä. [Artto *et al*, 2008; PMBOK Guide, 2008].

8. Riskienhallinta, tarkoittaa projektin riskien tunnistamista, analysointia ja niihin reagoimista. Riskienhallinnan täytyy olla luova kokonaisuus, jossa riskit otetaan huomioon mahdollisimman monipuolisesti ja jokaisen projektin erityisluonteen vaatimalla tavalla. Tämä tarkoittaa sitä, että jokaisessa projektissa riskienhallinta prosessi tarvitsee aina omanlaisensa, muista projekteista eroavan, mallin. Riskienhallinnassa ei siis voida vain olettaa aikaisempien jo kerran toimineiden ratkaisujen toimivan kaikissa muissakin projektissa. Onnistuneessa riskienhallinnassa pohjatyönä voidaan kuitenkin käyttää aikaisemmista projekteista ja muista tilastoista tehtyjä havaintoja. Näitä oleellisempia ovat kuitenkin projektiryhmän omakohtaiset havainnot ja kokemukset [Artto *et al*, 2008; PMBOK Guide, 2008].

9. Hankintojen hallinta, tarkoittaa projektin tarvitsemien tuote- ja palvelu resurssien etsintää, kartoitusta, valintaa, hankintaa, hankintojen onnistumisen seurantaa sekä näihin liittyvien sopimusten hallintaa. Projektille oikeiden työvälineiden valinta voi joko helpottaa tai vaikeuttaa projektin etenemistä ja toteutusta varsin merkittävästi. Myöskään työvälineiden vaihtaminen kesken projektin ei ole ongelmaton prosessi. Erityisesti ennen uusien tai tuntemattomien tuotteiden tai muiden työvälineiden käyttöä, on syytä harkita ulkopuolisen ammattilaisen konsultointia, jolla on

pidempi aikaista käytännön kokomusta juuri kyseisten välineiden käytöstä [Artto *et al*, 2008; PMBOK Guide, 2008].

Näitä erilaisia ja lähes kaiken kattavia projektinhallinnan malleja käytetään organisaatioissa hyvin paljon. Vaikka käytännössä nämä valmiit mallit ja niiden toimintatavat soveltuvat sellaisenaan suoraan varsin huonosti projektien ja organisaatioiden muuttuviin tarpeisiin. Valmiiden mallien räätälöinti ja joskus rajukin muokkaaminen organisaation omiin tarpeisiin, toimintakulttuuriin ja osaamiseen sopivaksi, palvelee organisaatiota huomattavasti paremmin kuin sopimattoman mallin käyttäminen. Se onko tämä organisaation oma malli sitten, kirjallinen opas, käsikirja tai joukko dokumenttipohjia yhdistettynä johonkin tietojärjestelmää, täytyy jokaisen organisaation itse ratkaista parhaaksi katsomallaan tavalla [Artto *et al*, 2008; Artto and Kähkönen, 1997; Pelin, 2008; Ruuska, 2007; Sommerville, 2007].

Tärkeintä projektinhallinnassa onkin muodostaa pitkällä aikavälillä toimivia ratkaisuja, jotka palvelevat organisaation pääasiallisen liiketoiminnan päämääriä ja tukevat sen tavoitteita. Yksittäisen projektin onnistunut läpivieminen ei yksinään enää riitä, koska projektit ovat jo nyt osa kaikkien organisaatioiden jokapäiväistä toimintaa. Joten uusia onnistumista vaaditaan yhä uudestaan - ja uudestaan.

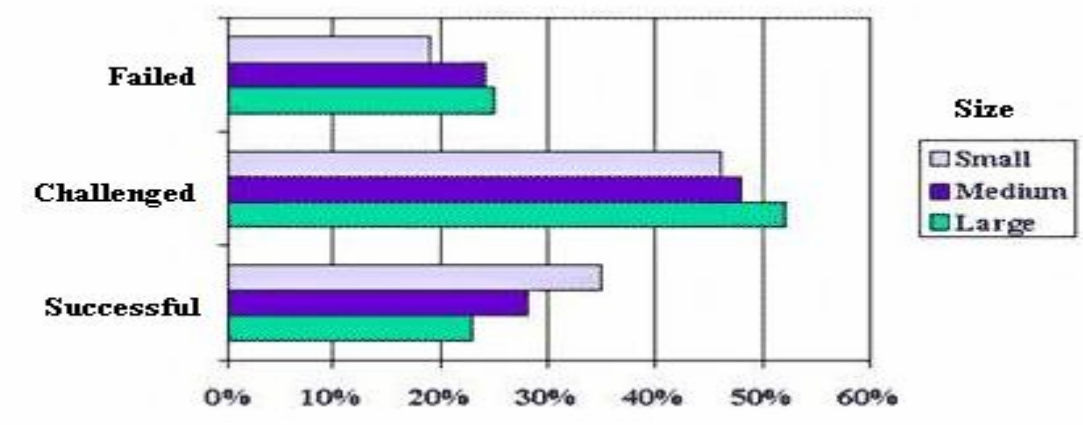
2.7. Projektin epäonnistuminen

Tehokkaalla projektihallinnalla voidaan merkittävästi vaikuttaa kaikkien projektien onnistumiseen. Kuitenkaan kaikkia projektien ongelmia ei voida juuri koskaan kokonaan estää tai hallita. Tästä syystä joiden projektien epäonnistuminen on varsin luonnollista. Projekteja kuitenkin epäonnistuu huomattavasti kohtuullisena pidettävää määrää enemmän.

Samaan aikaan kuitenkin erilaiset pienet ja keskikokoiset projektit ovat monille organisaatioille erittäin olennainen osa niiden toimintaa. Ne ovat myös juuri sitä kokoluokkaa ja tyyppiä, joita suurin osa organisaatioiden projekteista tänä päivänä juuri on. Tyypillinen pieni tai keskikokoinen projekti voi olla lähes mikä tahansa organisaation hanke, joka on ennakkoarvion perusteella vaikeusasteeltaan tai muilta ominaisuuksiltaan sopivan kokoinen. Nämä projektit ovat myös perusluonteeltaan ja ominaisuuksiltaan keskenään varsin samanlaisia, joten yleensä niitä voidaan tutkia ja tarkastella samalla tavoin [Artto *et al*, 2008; Boehm, 2000a; Chaos Report, 2009; Hoch *et al*, 2000; Keil *et al*, 1998; Olson, 2008].

Yleensä suuria koko organisaation toiminnan kattavia kehitysprojekteja, kuten ERP-hankkeita (Enterprise Resource Planning) sekä muita organisaatioiden kriittisiä

ohjelmisto- tai laitteisto hankkeita pankeissa, ydinvoimaloissa tai lentoyhtiöissä, pidetään ja tutkitaan selkeästi omana kokonaisuutena, koska näiden erityisluonne muuttuu aina tapauskohtaisesti. Kuitenkin projektien epäonnistumisen suhteen ne eivät ole niin erilasia, pieniin tai keskisuuriin projekteihin verrattuna, kuin aluksi voisi luulla. Tämä voidaan myös nähdä kuvasta 6 [Artto *et al*, 2008; Artto and Kähkönen, 1997; Boehm, 1991; 2000a; Chaos Report, 2009].



Kuva 6: Projektin koon vaikutus onnistumiseen ja epäonnistumiseen [Chaos Report, 2009]

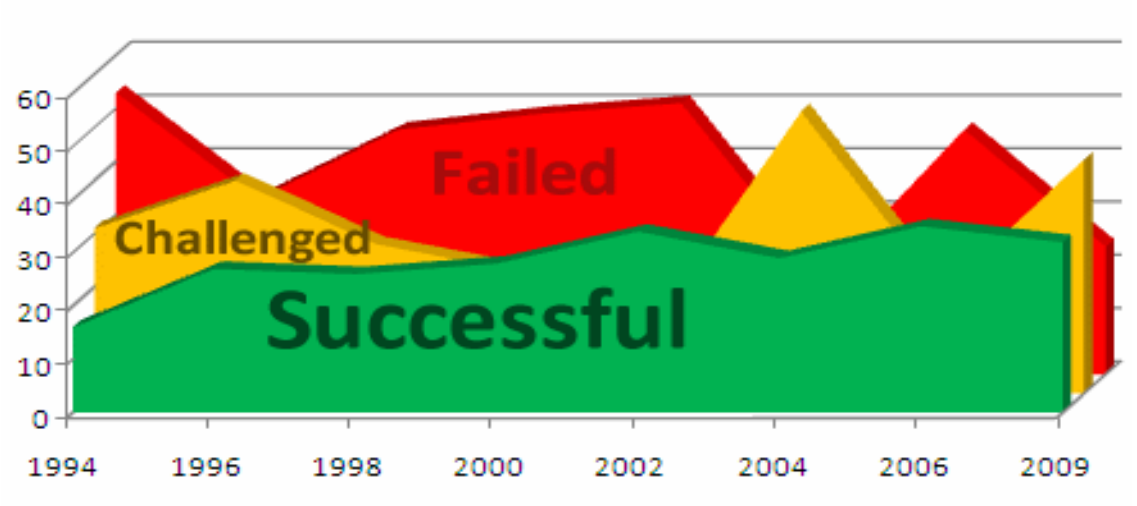
- Onnistunut (**Successful**) tarkoittaa tässä yhteydessä sitä, että projekti on valmistuttuaan täyttänyt kaikki sille alussa asetettu vaatimukset sekä pysynyt suunnitellussa aikataulussa ja budjetissa.

- Puutteellinen tai osittain epäonnistunut (**Challenged**) tarkoittaa tässä yhteydessä sitä, että projekti on valmistunut ja toimiva, mutta se on ylittänyt budjetin tai sen aikataulu on venynyt ja lisäksi sen toiminnallisuudessa voi olla puutteita.

- Epäonnistunut (**Failed**) tarkoittaa tässä yhteydessä sitä, että projektin toiminta on lopetettu kokonaan ennen sen valmistumista.

Tästä Yhdysvaltalaisen Standish Groupin vuosien 1994–2009 välillä tekemästä Chaos Report tutkimuksesta [2009] voidaan nähdä, että projektin koolla ei ole merkittävää vaikutusta siihen kuinka todennäköisesti projekti epäonnistuu. Tutkimusta voidaan pitää erittäin luetettavana sekä kattavana, koska sen tuotti ja rahoitti yksityinen organisaatio (Standish Group). Lisäksi sen tekemiseen osallistui yli 350 projektien ammattilaista. Kaiken kaikkiaan Chaos Report [2009] tutkimuksessa tutkittiin yli 23 000 projektia sekä yli 8300 erilaista sovellusta.

Kaiken kaikkiaankin tilastojen valossa projektien epäonnistumisen ja onnistumisen todennäköisyydet ovat varsin murheellista luettavaa. Chaos Report [2009] tutkimuksen sekä useista muista eri lähteistä tehdyn vertailun mukaan, vain noin 15–30 % projekteista onnistui täyttymään kaikki niille asetettu vaatimukset täysimääräisesti. Tämä tarkoittaa sitä, että lähteestä ja laskentatavasta riippuen noin 70–85 % kaikista IT-projekteista epäonnistuu joko täysin (20–50 %) tai ainakin osittain (30–60%). Tämä kehitys on esitetty kuvassa 7 [Artto *et al*, 2008; Boehm, 1989; 1991; 2000b; Chaos Report, 2009; Keil *et al*, 1998; Hoch *et al*, 2000; Kontio, 2001; Lyytinen and Hirschheim, 1987; Perminova et al, 2007; Pfleeger, 2000; Taylor, 2006; Wallace and Keil, 2004].



Kuva 7: Projektien onnistuminen sekä osittainen täydellinen epäonnistuminen vuosien 1994–2009 välillä [Chaos Report, 2009].

Ja vaikka tähän tilastoon onkin viime aikoina tullut parannusta, ja projektien onnistumisprosentti on kasvanut vuosien 1994–2009 välillä n. 10–15 prosenttia [Chaos Report, 2009], niin käytännössä tämä tarkoittaa siis sitä, että epävarmuustekijät useimmiten myös konkretisoituvat projekteja kaataviksi tapahtumiksi. Selkeitä syitä miksi näin tapahtuu tai toimivia ratkaisuja tähän poikkeuksellisen korkeaan epäonnistumisen todennäköisyyteen, ei yksikään tutkia ole vielä pystynyt selkeästi esittämään. Sen sijaan voimme todeta, että onnistumme hyvällä menestyksellä toistamaan samoja virheitä vuodesta toiseen.

Standish Groupin tutkimuksen [Chaos Report, 2009] mukaan, yleisimmät projektien onnistumiseen, epäonnistumiseen tai puutteelliseen lopputulokseen johtaneet syyt olivat:

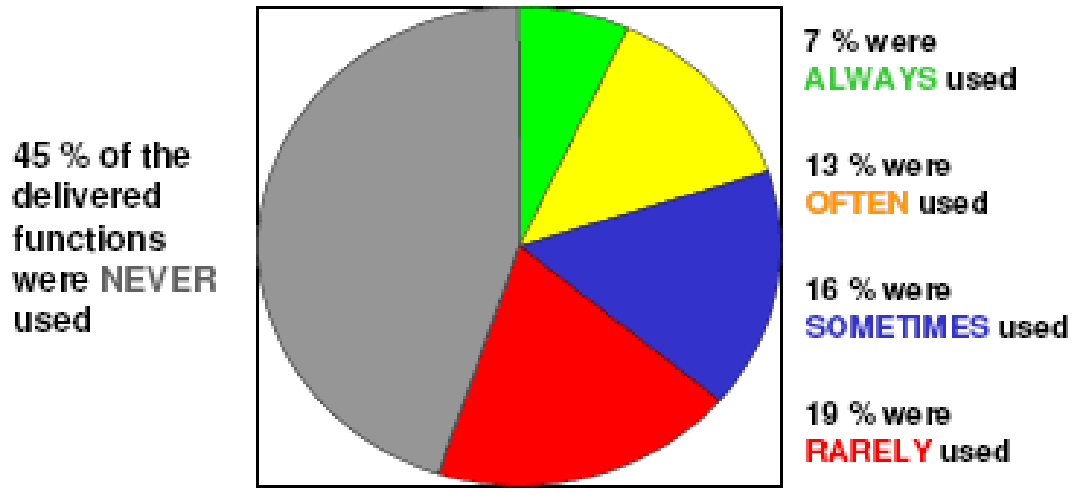
- **Onnistunut**
    1. Käyttäjien osallistuminen (15.9 %)

2. Johtoportaan tuki (13.9 %)
3. Selvästi kirjatut vaatimukset (13.0 %)

- **Puutteellinen**
  1. Käyttäjien palautteen puuttuminen (12.8 %)
  2. Epätäydelliset vaatimukset ja määritykset (12.3 %)
  3. Muuttuvat vaatimukset ja määritykset (11.8 %)

- **Epäonnistunut**
  1. Epätäydelliset vaatimukset (13.1 %)
  2. Käyttäjien osallistumisen puute (12.4 %)
  3. Resurssien puute (10.6 %)

Lisäksi Chaos Report [2009] arvostelee myös varsin voimakkaasti monia projekteissa tehtäviä asioita. Erityisesti ohjelmistoihin tehtävien ominaisuuksien todellinen tarve sekä kaikkien toteutettujen toimintojen käyttömäärät ensimmäisen vuoden aikana ohjelman käyttöönotosta ovat kyseenalaiset. Tätä jakoa on esitetty kuvassa 8.



Kuva 8: Ohjelmiston ominaisuuksien ja toimintojen käyttö sekä tarpeellisuus [Chaos 2009].

Kuvasta 8 voidaan nähdä, että ohjelmistoprojektien lopputuotteet sisältävät erittäin paljon (45 %) turhia ominaisuuksia joita ei koskaan käytetä. Kaiken kaikkiaankin vain 36 % kaikesta toiminnallisuudesta on säännöllisesti käytössä ja 19 % erittäin harvoin. Voidaan siis kärjistäen todeta, että 55 % tehdystä työstä oli ainakin tällä hetkellä tarpeellista ja 45 % turhaa resurssien hukkaamista.

Kaikki tämä yhdessä tekeekin kaikista teknisistä projekteista monella tapaa poikkeuksellisia suhteessa muihin organisaatioiden kehityshankkeisiin. Esitetyt todennäköisyydet ja muut tilastot eivät ainakaan lisää organisaatioiden mielenkiintoa tai halukkuutta teknisiä investointeja sekä muita hankkeita kohtaan – päinvastoin. Tämän epäröinnin pitkän aikavälin seurauksia esimerkiksi tuotekehitykselle tai innovaatioiden tuotteistukselle voi vain arvailla.

Kuitenkaan kaikki sairaudet parantavaa ihmelääkettä projektien onnistumisen takaamiseksi ei siis ainakaan vielä ole näköpiirissä. Ehkä vastaus näihin ongelmiin on kuitenkin olemassa. Viimeaikaisten projektien- ja riskienhallinnan tutkimusten aiheita ja suuntauksia tarkasteltaessa, vastaus voi hyvinkin löytyä esimerkiksi tutkimalla juuri epäonnistuneita projekteja tapauskohtaisesti ja oppimalla asioita näiden kautta.

3. Riskit ja riskienhallinta

3.1. Riskin käsite ja tausta

Alkujaan riski sana tulee joko kreikankielisestä karia tarkoittavasta sanasta ”rhi zikon” tai latinankielisestä karin kiertämistä tarkoittavasta sanasta ”risicare”. Riskit liittyvät myös kiinteästi vakuutustoimintaan ja vakuutustoiminta kehittyi alkujaan juuri kuljetusten ja merenkulun ympärille. Nykyisessä varsin modernissa ja teknisessä yhteiskunnassa riskit ovat kuitenkin muuttaneet muotoaan ympäröivän maailman kehityksen mukana [Kuusela ja Ollikainen, 2005].

Kun arkikielessä puhutaan riskistä, tarkoitetaan yleensä jonkin asian, tapahtuman, tilan tai tuloksen epävarmuutta sekä siitä mahdollisesti aiheutuvaa haittaa ja mahdollista vahinkoa. Riski sanaa käytetään myös kuvaamaan sitä vaaraa ja epätietoisuutta, joka liittyy jonkin odottamattoman asian mahdollisuuteen. Riskeihin liittyy olennaisesti tappion mahdollisuus ja siihen kuuluva menettämisen uhka [Kuusela ja Ollikainen, 2005].

Olennainen riskeihin liittyvä tekijä on myös epävarmuus. Kukaan ei voi varmuudella tietää tai ennustaa tulevia tapahtumia, vaikka tuntisimme näiden tapahtumien todennäköisyyksiä. Todennäköisyyksiin liittyy kuitenkin aina virhearvioinnin mahdollisuus. Ihmisillä on myös taipumus olla tietyissä tapauksissa liian luottavaisia omiin riskiarvioihin. Riski ja sen todennäköisyys ovatkin kuitenkin aina vain arvion tehneen henkilön tai ryhmän subjektiivinen näkemys lopputuloksesta. Tästä johtuen riskiarviot voivat vaihdella paljon eri tahojen välillä [Artto and Kähkönen, 1997; Kuusela ja Ollikainen, 2005].

Riskejä voidaan luokitella monella eri tavalla. Tavallisesti riskit jaetaan ensiksi vahinkoriskeihin ja liikeriskeihin. Vahinkoriskin toteutuminen aiheuttaa pelkästään haittaa. Liikeriskeihin eli spekulatiivisiin riskeihin sisältyy myös voiton tai hyödyn mahdollisuus. Toiseksi riskit voidaan luokitella myös niiden aiheuttajan ja riskille altistuvien henkilöiden lukumäärän mukaan. Partikulaariset riskit ovat yksittäisen henkilön toiselle aiheuttamia riskejä. Fundamentaaliset riskit koskettavat laajoja ihmisjoukkoja eivätkä ne ole yksittäisten henkilöiden aiheuttamia [Artto and Kähkönen, 1997; Kuusela ja Ollikainen, 2005].

Osa ihmisistä sietää riskejä paremmin ja he haluavat ottaa riskejä. Kun taas toiset, pelaavat niin sanotusti varman päälle. Jotkut myös ali- tai yliarvioivat riskejä. Syitä tähän käytökseen on monia. Se ympäristö jossa elämme, ja jonka toiminnan tunnemme, on luultavasti yksi keskeisimmistä. Mutta myös riskin arvioijan tiedot ja tietoisuustaso

mahdollisista riskeistä vaikuttaa asiaan [Kuusela ja Ollikainen, 2005; Pfleeger, 2000; Williams, 1997].

Tämä johtaakin siihen, että riskienhallintaan liittyy aina olennaisena osana asiantuntevat henkilöt, jotka osaavat kartoittaa riskit aiempien kokemustensa ja tietopohjansa kautta. Kaiken tämän pohjalta voidaan tehdä päätelmä, että riskit voivat olla samaan aikaan sekä mahdollisuus että uhka. Tietyissä tilanteissa täytyy olla rohkeutta luottaa onnistumiseen, vaikka lopputuloksesta ei olisikaan varmuutta. Joskus juuri tämä hyppy tuntemattomaan, voi jopa olla hankkeen tai projektin menestymisen edellytys – tai vaihtoehtoisesti sen tuho.

3.2. Riskienhallinta eri yhteyksissä

Kaikkeen tekemiseen sekä toimintaa kohdistuu ja liittyy aina joitakin riskejä. Riskejä toteutuu jatkuvasti eri yhteyksissä, mutta niitä myös tunnistetaan ja hallitaan päivittäin eri tavoin. Voidaankin sanoa, että elämme jatkuvaa epätietoisuuden ja epävarmuuden aikaa, tiedostimme sitä sitten itse tai emme. Jotta tämä ei olisi meille liian raskasta, täytyy riskejä pystyä jollain tavalla hallitsemaan. Riskienhallinnan idea syntyi juuri näiden ajatusten pohjalta käyttämällä apuna logiikkaa ja systematiikkaa [Artto and Kähkönen, 1997; Riskienhallinta kirjanen, 2000].

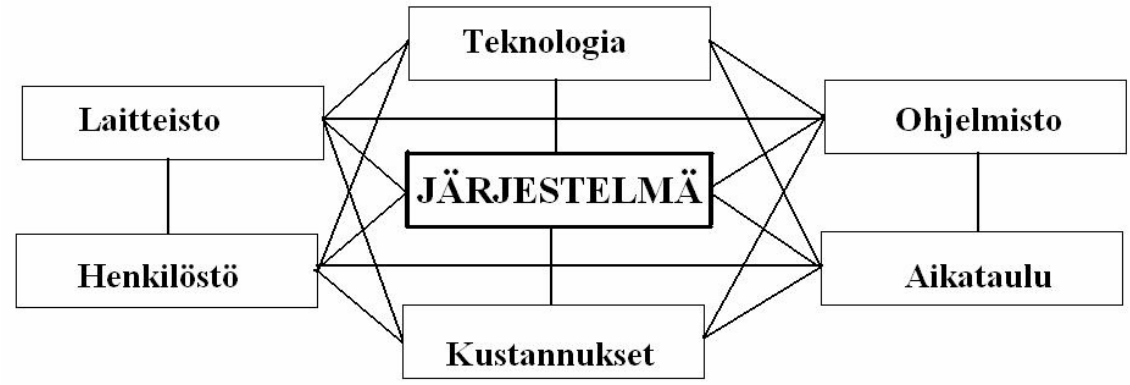
Riskejä ja riskienhallintaa määriteltäessä tulee ensin ottaa huomioon määrittelyn asiayhteys. Esimerkiksi riskianalyysi tarkoittaa eri hieman asiaa ohjelmistoprojekteille, pankeille ja yrityksille. Tästä syystä, sen yleisluontoinen määrittely onkin vaikeaa. Pankkien ja vakuutusyhtiöiden käyttämät menetelmät ovat erittäin pikkutarkkoja koska riskit ovat niiden pääliiketoiminnan ja tuloksen tekemisen elinehtoja [Artto and Kähkönen, 1997; Suominen, 2000].

Erilaisten organisaatioiden riskienhallinta on taas luonteeltaan yleisempää, mutta se on yhä kuitenkin suunnitelmallinen ja jatkuva prosessiksi, joka tähtää erilaisten vaiheiden kautta organisaation liiketoimintaan ja käytössä olevien resursseihin vaikuttavien riskien tunnistamiseen, arvioimiseen ja hallitsemiseen. Pääsääntöisesti riskienhallinta kuuluukin joko organisaation johdon tehtäväkuvaan tai oman hallinto-osaston vastuualueeseen [Artto and Kähkönen, 1997; Kuusela ja Ollikainen, 2005].

Tämän pohjalta voidaankin todeta, että eri tahot ja mukana olevat osapuolet tarkastelevat sekä riskejä että niiden hallintaa aina omasta näkökulmastaan. Toisille taloudelliset tai aikatauluun liittyvät seikat ovat tärkeämpi kuin laatu ja oikeanlainen toiminnallisuus. Tämä vaikuttaa moniin asioihin, mutta erityisesti käytettävissä oleviin resursseihin, riskienhallinnan suuntaan ja tehtävien priorisoitiin.

3.3. Lähtökohdat ja toimintamallit

Osa projektien riskeistä on tyypiltään yleisluonteisia riskejä, mutta toiset ovat juuri kyseiselle projektille tyypillisiä. Suuret budjetin ylitykset, vuosien myöhästymiset aikatauluissa, puutteet toiminnallisuudessa ja laadussa sekä viimekädessä koko hankkeen peruutukset ovat niitä asioita, jotka monesti liitetään erilasiin projekteihin [Lyytinen and Hirschheim, 1987]. Käytännössä kuitenkin projektien riskit voivat olla lähtöisin mistä tahansa projektissa mukana olevasta perusedellytyksestä. Nämä on esitetty kuvassa 9 [Higuera and Haimes, 1996].

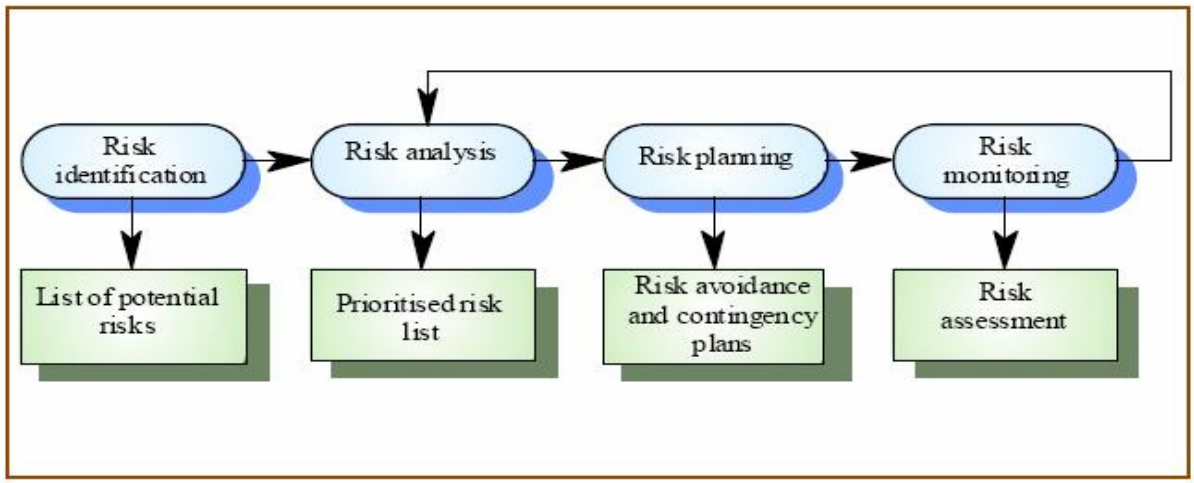


Kuva 9: Projektin riskien lähtökohdat [Higuera and Haimes, 1996, s.5]

Vaikka projektien riskien lähtökohta olisi mikä tahansa kuvassa 9 mainituista asioista, niin käytännössä ne voivat vaikuttaa kolmeen asiaan. Riski voi olla ensinnäkin projektikohtainen, jolloin se vaikuttaa joko projektin aikatauluun tai sen käytössä oleviin resursseihin. Toiseksi se voi olla tuotekohtainen, jolloin se vaikuttaa kehitettävän tuotteen laatuun. Kolmanneksi se voi olla organisaatiokohtainen, jolloin se vaikuttaa joko toimittaja- tai tilaajaorganisaatioon [Sommerville, 2007]. Aikojen kuluessa ja projektienhallinnan kehittyessä useat tutkijat ovat huomanneet, että tietyt riskit esiintyvät kuitenkin toisia useammin projektin luonteesta tai sen tyyppistä riippumatta [Artto et al, 2008; Boehm, 1989; 1991; Higuera and Haimes, 1996; Hoch *et al*, 2000; Pfleeger, 2000].

Projektinhallinnan, ja riskienhallinnan sen osana, ensisijaisena tehtävänä on pyrkiä takaamaan projektin valmistuminen sekä onnistuminen kaikissa tilanteissa. Tämä vaatii toimia, joiden avulla pystytään sekä ennakoimaan, että tarvittaessa myös reagoimaan riskeihin niiden toteutuessa. Pitää myös muistaa, että riskien määrä kasvaa aina sitä mukaan, mitä vaativampi toteutettava projekti on. Tietyissä tilanteissa kasvu voi myös joskus olla eksponentiaalista [Charette, 1996; Ropponen, 1999; Sommerville, 2007;].

Projektinhallinnassa ja projekteissa yleensä riskienhallintaprosessi voidaan jakaa hieman lähteistä riippuen monella tavalla. Tässä tutkielmassa käytän jakoa ennakoivaan ja reagoivaan riskienhallintaan. Riskienhallintaprosessi on esitetty kuvassa 10.



Kuva 10: Riskienhallintaprosessi [Sommerville, 2007 s.106]

Ennakoivaan riskienhallintaan voidaan käsittää kuuluvan riskienhallintaprosessin kaksi ensimmäistä vaihetta eli riskien tunnistaminen ja riskien analysointi. Reagoivaan riskienhallintaan taas voidaan käsittää kuuluvan riskienhallintaprosessin kaksi jälkimmäistä vaihetta eli riskien suunnittelu ja riskien valvonta tai seuranta. Kun tämän jälkeen riskienhallintaprosessia puretaan vielä pienempiin kokonaisuuksiin, niin voidaan muodostaa käytännön toimia projektien riskienhallintaan [Boehm, 1989; 1991 Higuera and Haimes, 1996; Hoch et al, 2000; PMBOK Guide, 2008; Pressman, 2010; Ropponen, 1999; Sommerville, 2007].

3.4. Ennakoiva riskienhallinta

Ennakoivassa riskienhallinnassa on nimensä mukaisesti kyse erilaisista keinoista ja toimintatavoista, joiden avulla pyritään ennakoimaan ja varautumaan projektien riskeihin ennen kuin mitään negatiivista on vielä tapahtunut.

3.4.1. Riskien tunnistaminen

Riskien tunnistaminen on kaiken riskienhallinnan lähtökohta, joka luo samalla pohjan muulle riskienhallinta työlle. Yleensä suurin riski onkin se, että ei tiedetä mitä riskit kulloinkin ovat. Tästä syystä, tähän vaiheeseen on syytä panostaa todella paljon, mutta samalla pitää myös muistaa se, että kaikkeen ei voi milloinkaan etukäteen varautua. Projekteissa sattuu usein odottamattomia asioita, jolloin korostuu ryhmän osaaminen sekä projektipäällikön rooli. Yleisesti tunnustettu fakta onkin, että menestyvät

projektipäälliköt ovat yleensä myös hyviä riskienhallinnassa [Boehm, 1989; 1991; Keil *et al*, 1998; PMBOK Guide, 2008].

Koska jokainen projekti on aina jollakin tavalla uniikki ja yksilöllinen monessa suhteessa, niin mitään kaikkiin projekteihin sopivia yleispäteviä, Top10 tai vastaavia yleisiä, riski- ja tarkistuslistoja ei voida pelkästään käyttää riskien tunnistamiseen [Boehm, 1991]. Näitä voidaan käyttää ainoastaan referensseinä tai oman toiminnan apuna, mutta kokonaan niiden varaan ei riskien tunnistamista voida kuitenkaan laskea. Jokaiseen projektiin täytyykin koota aina uudet ja kyseiseen tilanteeseen sekä toimintaympäristöön sopivat omat riskilistat [Keil et al, 1998; PMBOK Guide, 2008; Ropponen, 1999; Sommerville, 2007]. Kuvassa 11 on esitetty joitakin ohjelmistoprojekteille tyypillisiä riskejä sekä niiden kuvaukset ja varsinaiset vaikutuskohteet.

Risk	Affects	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organisational management with different priorities.
Hardware unavailability	Project	Hardware that is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool under-performance	Product	CASE tools which support the project do not perform as anticipated
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

Kuva 11: Ohjelmistoprojekteille tyypilliset riskit [Sommerville, 2007, s.105]

Varsinainen käytännön riskien tunnistaminen tapahtuu yleensä monen asian ja tavan summana. Tässä voi olla mukana haastatteluita, kaavioita, aikaisempia kokemuksia, mukanaolijoiden omia tapoja sekä monia muita asioita. Pääasia on kuitenkin lopputulos, eli tuotteeseen, projektiin ja organisaatioon liittyvien riskien tunnistaminen. Eikä niinkään se, miten ne muodostetaan - kunhan saadut tulokset ovat oikeat ja riittävän monipuoliset [Boehm, 1989; 1991; Haikala ja Märijärvi, 2004]. Yleensä aluksi pitää pyrkiä löytämään kaikki mahdolliset riskit, jotka voivat vaikuttaa projektin onnistumiseen. Käytännössä

hyvin usein kuitenkin erittäin epätodennäköiset ja merkityksettömät riskit voidaan jättää vähemmälle huomiolle, jotta riskienhallinnasta ei tulisi ylimitoitettua prosessia, mikä taas voisi johtaa projektin käytössä olevien niukkien resurssien kuluttamiseen tarpeettomasti [Keil *et al*, 1998; Pressman, 2010].

3.4.2. Riskien analysointi

Riskien tarkemmassa analysointi vaiheessa keskitytään tarkastelemaan lähemmin tunnistusvaiheessa koottuja riskejä. Kun riskit on ensin tunnistettu, niin tämän jälkeen ne voidaan jaotella kategorioittain esimerkiksi teknologisiin, henkilö- ja hallinnollisiin riskeihin. Jako voidaan myös tehdä aina kunkin projektin vaatimalla tavalla. Pääsääntönä voidaan pitää toiminnan mitoittamista muuhun projektin toimintaan. Suurissa projekteissa voidaan käyttää myös kattavampaa jaottelua esimerkiksi 6 ylätason riski kategoriaan ja näiden alakategorioihin [Kontio, 2001; Sommerville, 2007].

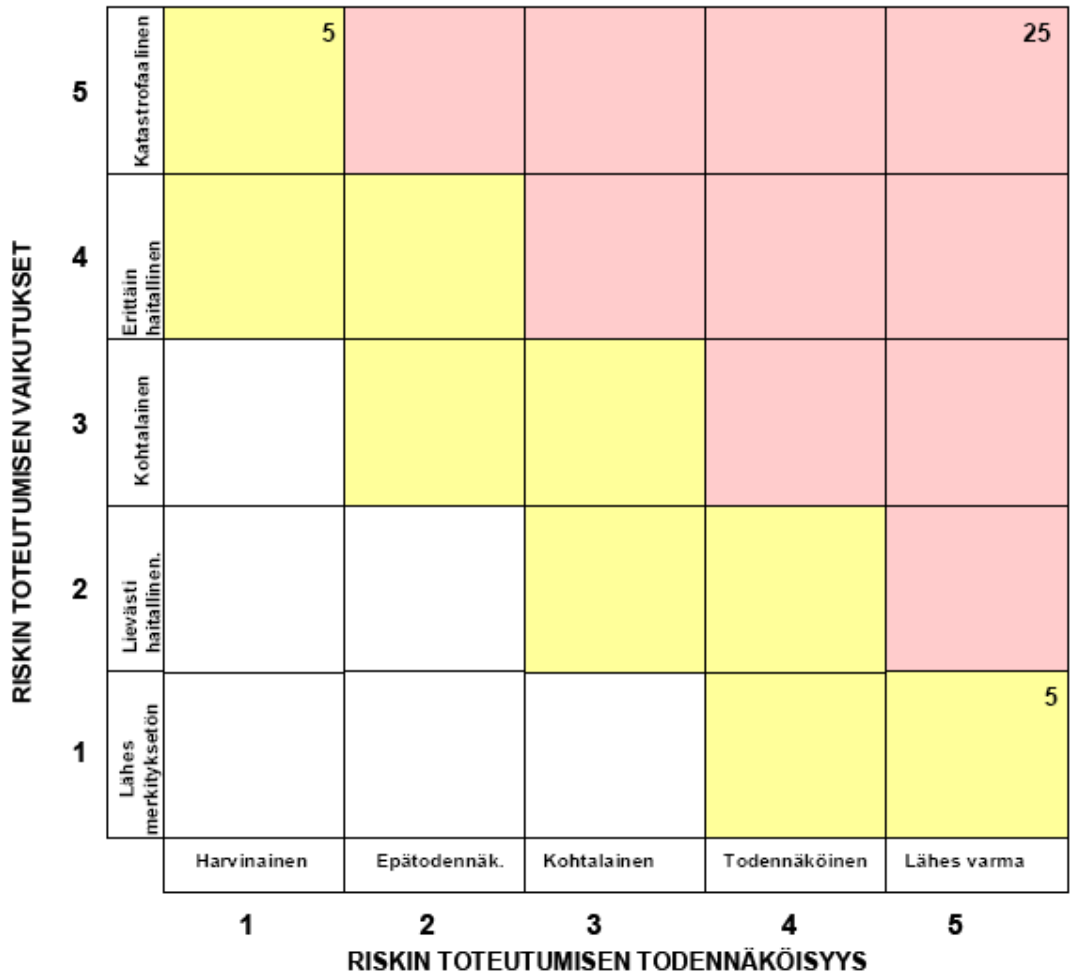
Jaottelun yhteydessä riskit myös nimetään ja niille kirjoitetaan kuvaus, jonka muoto, tyyppi ja laajuus riippuvat käyttävästä ohjelmistokehitysmenetelmästä. Yleensä tähän sisällytetään pelkän riskin kuvauksen lisäksi myös riskin seuraukset projektille sekä keinot joilla riskin toteutuminen voidaan mahdollisesti havaita ennakolta. Myöhemmin näihin lisätään myös riskin toteutumisen estämiseen tarkoitettu vastatoimet sekä toimintaohjeet mahdollisimman nopeaan riskin toteutumisesta toipumiseen [Kontio, 2001; Ropponen, 1999; Sommerville, 2007].

Analysoinnin lopuksi riskien ominaisuudet saatetaan mittavaan muotoon ilmaisemalla niitä kvantitatiivisilla arvoilla. Tämä tapahtuu laskemalla kaikkien luokiteltujen riskien suuruus tai merkitys kyseiselle projektille. Tässä käytetään jotakin valittua mitta-asteikkoa, jonka avulla riskeille saadaan niitä kuvaavat numeraaliset arvot. Riskien ominaisuuksien kuvaukseen voidaan myös ilmaista kvalitatiivisesti. Pääsääntöisesti yleinen riskien suuruus numeraalisesti lasketaan kaavalla [Boehm, 1989; Kontio, 2001]:

Riskin todennäköisyys * Riskin vakavuus/vaikutus = Riskin suuruus/merkitys.

Jossa todennäköisyys on määritetty arvioimalla riskin toteutumisen todennäköisyys asteikolla 1–5 siten, että numero 1 tarkoittaa harvinaista riskiä ja numero 5 lähes varmasti toteutuvaa riskiä. Tämän jälkeen myös riskin vakavuus/vaikutus määritetään samalla tavalla asteikolla 1–5, jossa numero 1 tarkoittaa lähes merkityksetöntä vaikutusta ja numero 5 katastrofaalista vaikutusta projektille. Todennäköisyydelle ja vakavuudella voidaan myös antaa muita arvoja sen mukaan miten ne päätetään määritellä tai muodostaan. Näitä tapoja en tässä tutkielmassa käsittele [Boehm, 1989; 1991; Kontio, 2001; PMBOK Guide, 2008; Ropponen, 1999; Sommerville, 2007].

Lopuksi luvut kerrotaan keskenään, jolloin lukujen summa tarkoittaa riskin suuruutta tai merkitystä projektille. Mitä suurempi yhteisluku, niin sitä suurempi vaikutus riskillä on projektin onnistumisen ja toiminnan kannalta. Tätä voidaan myös havainnollistaa projektissa mukanaolijoille kuvan 12 mukaisella tavalla, jonka avulla varmistetaan että kaikki mukanaolijat ymmärtävät riskien luonteen. [Boehm, 1989; 1991; Kontio, 2001; PMBOK Guide, 2008; Ropponen, 1999; Sommerville, 2007].



Kuva 12: Riskien suuruuden havainnollistaminen

Kuvassa 12 valkoiselle alueelle sijoittuvat riskit ovat merkitykseltään vähäisiä. Vastaavasti keltaiselle alueelle sijoittuvat riskit ovat jo selkeästi merkityksellisiä ja ne pitää ottaa mukaan aktiiviseen riskienhallintaan. Punaiselle sijoittuvat riskit ovat erittäin vakavia, joten niiden hallinnassa täytyy olla erityisen aktiivinen.

Toinen teoreettisempi, mutta kuitenkin hyvin paljon riskien analysoinnissa käytetty tekniikka on riskin vähentymiseen tarvittavan voiman tai panostuksen suuruuden

laskeminen. Tämän avulla mitataan riskien vähentämiseen käytettävien tekniikoiden toiminnan tehokkuutta. Tämä voidaan laskea kaavalla [Boehm, 1989, Kontio, 2001]:

Riskin vaikutuksen arvo ennen – Riskien vaikutuksen arvo jälkeen	
Riskien vähentämisen kustannukset	**= Riskin vähentämisen voima**

Riskien vaikutuksen arvo ennen ja jälkeen tarkoittaa yksittäisen riskitapahtuman odotettua arvoa ennen ja jälkeen sen toteutumista. Riskien vähentämisen kustannukset taas tarkoittavat kaikkia niitä kustannuksia, jotka syntyvät välittömästi ja välillisesti kaikista niistä toimista joiden tarkoituksena on vähentää yksittäisen riskin toteutumista [Boehm, 1989; Kontio, 2001].

Riskien vähentämiseen tarvittavien panostusten suuruus tulisi aina olla mahdollisimman pieni, koska tällöin riskienhallinta toiminta on kaikkien tehokkainta. Tällöin myös suuremmalle osalle tunnetuista riskeistä voidaan suunnitella vastatoimet ilman että riskienhallinnan kokonaiskustannukset yksittäistä projektia kohden nousevat kohtuuttoman suuriksi. Ainakin teoriassa, projektin kaikkien riskien toteutuminen voidaan ennaltaehkäistä tai estää, mutta tämän vaatimat kustannukset sekä muut resurssit ovat niin suuret, että käytännössä tämä ei kuitenkaan ole mahdollista projektin käytössä olevien rajallisten resurssien vuoksi.

3.5. Reagoiva riskienhallinta

Reagoivassa riskienhallinnassa on nimensä mukaisesti kyse erilaisista keinoista ja toimintatavoista, joiden avulla pyritään seuraamaan koko projektin ajan riskien kehitystä sekä reagoimaan niihin riskeihin, jotka ovat lähellä toteutua tai ovat jo toteutuneet.

3.5.1. Riskien suunnittelu

Riskien suunnittelussa jo aikaisemmin löydetyille ja jaotelluille riskeille mietitään tai suunnitellaan keinoja sekä käytännön toimintatapoja, joiden avulle niiden toteutuminen pyritään joko kokonaan välttämään, vaikutukset minimoimaan tai toteutumisen jälkeisistä palautumista sekä projektin toiminnan normalisoitumista nopeuttamaan [Boehm, 1989; 1991; Sommerville, 2007]. Riskien välttämisen tarkoituksena on siis pyrkiä pienentämään riskin toteutumisen todennäköisyyttä. Todennäköisyyden laskeminen esimerkiksi 4:sta 3:en pienentää merkittävästi riskin toteutumista, vaikka sen vaikutus pysyisikin ennallaan. Riskien vaikutusten minimoinnilla taas pyritään pienentämään riskien vaikutusta projektiin. Tämä on kuitenkin paljon työläämpää kuin

todennäköisyyden pienentäminen ja vaatii huomattavasti enemmän panostusta sekä resursseja [Keil *et al*, 1998; Kontio, 2001; Pelin, 2008; Ropponen, 1999].

Mikäli kuitenkin molempia, sekä todennäköisyyttä että vaikutusta, pystytään jollakin keinoilla pienentämään, niin riskin toteutuminen on huomattavasti epätodennäköisempää kuin ennen näitä toimia. Lisäksi myös riskin suuruus tai merkitys projektille pienenee, joten riskien konkretisoituessa sen vaikutukset eivät ole niin suuret kuin ennen riski suunnittelun aloittamista. Kun riskien suunnitteluvaihe on hoidettu hyvin, niin tämän jälkeen voidaan ryhtyä miettimään ja listaamaan vastatoimia, joiden avulla riskeihin pystytään reagoimaan mahdollisimman pian niiden toteutumisen jälkeen.

Jos kaikista ponnisteluista huolimatta jokin riski kuitenkin toteutuu, niin valmiiden toimintatapojen miettiminen näihin tilanteisiin nopeuttaa merkittävästi tapahtumasta toipumista, sekä pitää samalla projektin jatkon ja etenemisen kannalta oikeassa suunnassa. Ja vaikka kaikkiin tapahtumiin ei voidakaan suunnitella vastatoimia, niin silti ennakolta ainakin kaikkien tuhoisimmiksi tai kohtalaisen todennäköisiksi arvioituihin riskeihin täytyy olla etukäteen varauduttu. Joissakin tapauksissa merkitykseltään suurimpiin riskeihin tehtyjä suunnitelmia ja vastatoimia voidaan tarvittaessa ainakin soveltaen käyttää myös pienempiin riskeihin [Boehm, 1989; Keil *et al*, 1998; Pfleeger, 2000].

3.5.2. Riskien seuranta

Riskien tiedostamista ja niiden seurantaa pidetään merkittävimpänä riskeihin reagoivana toimintana. Seuranta voi tapahtua monella tavalla projektin luonteesta ja käyttävästä ohjelmistokehitysmenetelmästä riippuen. Tärkeintä on, että seuranta on säännöllistä ja kestää koko projektin ajan. Riskin toteutuessa, on myös empimättä ryhdyttävä aikaisempien suunnitelmien mukaisiin vastatoimenpiteisiin [Boehm, 1989; 1991; Pressman, 2010; Ropponen, 1999; Sommerville, 2007].

Käytännössä tätä riskien seurantaa voidaan yksikertaisimmillaan suorittaa esimerkiksi kahden listan avulla. Jossa ensin toiseen listaan kirjataan ja arvioidaan projektin 15–30 vakavinta riskiä tärkeysjärjestyksessä säännöllisin väliajoin. Ja samaan aikaan, toisessa listassa seurataan ja pidetään kirjaa siitä, kuinka monta kertaa kukin riski on ensimmäisellä listalla esiintynyt sekä millä sijalla se on edellisillä kerroilla ollut. Näitä kahta listaa seuraamalla ja vertailemalla voidaan havaita pitkään listalla pysyvät tai lähellä toteumista olevat tunnetut riskit. Tämä helpottaa huomattavasti kokonaisvaltaista riskienhallintaa projekteissa niiden koosta riippumatta. Kuvassa 13 on esitetty joitakin tunnetuimpien riskien toteutumiseen viittaavia merkkejä [Boehm, 1989; 1991: Pressman, 2010; Ropponen, 1999; Sommerville, 2007].

Risk type	Potential indicators
Technology	Late delivery of hardware or support software, many reported technology problems
People	Poor staff morale, poor relationships amongst team member, job availability
Organisational	Organisational gossip, lack of action by senior management
Tools	Reluctance by team members to use tools, complaints about CASE tools, demands for higher-powered workstations
Requirements	Many requirements change requests, customer complaints
Estimation	Failure to meet agreed schedule, failure to clear reported defects

Kuva 13: Riskien toteutumiseen viittaavia merkkejä [Sommerville, 2007, s.111]

Tämän lisäksi on järkevää luottaa projektissa mukanaolijoiden aikaisempiin kokemuksiin sekä heidän kykyynsä lukea erilaisia signaaleja projektin aikana. Kokeneet projektinhallinnan ammattilaiset sekä muut ryhmän jäsenet pystyvät näkemään erilaisia merkkejä ja seuraamaan tapahtumaketjuja, jotka voivat viitata jonkin tuntemattoman riskin toteutumiseen. Tästä voidaan käyttää nimitystä substantiivinen ja normatiivinen asiantuntemus. Lisäksi erilaiset matemaattiset sekä historialliset mallit ja simulaatiot ovat tuoneet hyviä tuloksia [Boehm; 1991; Higuera and Haimes, 1996; Pressman, 2010; Ropponen, 1999].

Kun projektin alussa kootuista riskeistä muodostettuja ja analysoituja riskilistoja sekä projektissa mukanaolijoiden omista kokemuksista muodostettuja riskilistoja verrataan ensin keskenään ja lopuksi yhdistettään, niin tällöin voidaan puhua tehokkaasta ja riittävästä riskien seurannasta. Jonka viimeisenä keinona toimii etukäteen mietittyjen vastatoimien kokonaisuus, minkä vuoksi suurin osa riskeistä ei missään tapauksessa pääse aiheuttamaan projektin etenemisen kannalta suuria viivytyksiä. Lisäksi mikäli projektipäälliköillä on riittävästi tietoa, taitoa sekä tarvittavaa käytännön kokemusta nähdä projektin kehitystä pidemmällä aikajänteellä, niin kokonaisuutena projektin toiminta on vakaalla pohjalla. Tällöin myöskään uusia ja ennestään tuntemattomia riskejä ei pääse enää syntymään ja vaikuttamaan projektin toimintaympäristöön.

3.6. Riskienhallinnan kehitys

Aikojen kuluessa riskit ja riskienhallinta ovat muuttaneet muotoaan. Muinaisessa Kreikassa suurimmat riskit tulivat merenkulussa, jossa karien välttäminen ja laivojen lastin perille pääsy varmistaminen oli erityisen tärkeää. Riskit ovat muuttuneet ja

muuttuvat yhä edelleen, kokoajan monimutkaisemmiksi, samalla myös niiden määrä on kasvanut. Lisäksi myös niiden hallintatavat ovat muuttuneet.

Nykyisessä kompleksisessa maailmassa, jossa tietokoneet ja erilaiset järjestelmät vastaavat monista kriittisistä asioista, riskit ja riskienhallinta ovat nousseet täysin uudelle tasolle. Kun esimerkiksi sähkönjakelu keskeytyy muutamaksi päiväksi tai vain tunniksi, niin tuloksena on luultavasti laaja paniikki, jolloin ilmassa on suurkatastrofin ainekset, jonka kaikkia vaikutuksia on lähes mahdotonta ennakoida. Myös pienten ja keskisuurten sekä samalla vähemmän kriittisten projektin suuri määrä eri organisaatioissa tuo mukaan uusia haasteita riskienhallinnalle.

Siinä miten projektien riskienhallinta nähdään tänä päivänä, ja miten se nähtiin aikaisemmin, on tapahtunut selkeä muutos [Boehm, 1991; Charette, 1996; Pfleeger, 2000]. Vielä 1990-luvun alussa riskienhallinta oli joukko erillisiä toimia ja tapahtumia, ilman selkeää linkkiä vaiheesta toiseen. Mutta 2000-luvulle tultaessa ja riskienhallinnan monimutkaisuuden kasvaessa sekä teknistyessä alettiin nähdä suurten kokonaisuuksien hallinnan merkitys koko hankkeen kannalta. Tämän kehityksen jatkumona voidaan myös nähdä lainsäädännön kehittyminen ja erilaisten uusien laatujärjestelmien syntyminen [Charette, 2001c; Chin, 2004; Perminova *et al*, 2007; Pressman, 2010; Ropponen, 1999].

Kokonaisvaltaisesta riskienhallinnasta onkin tullut kokoajan entistä merkittävämpi keino uusien projektien onnistumisen varmistamisessa. Myös kokonaan uusien projektinhallinta- ja ohjelmistokehitysmenetelmien käyttö kaiken tyyppisissä projekteissa, ovat antaneet lupaavia tuloksia. Toimivan riskienhallinnan lisäksi, niiden menestyksen perusta on ollut keskittyminen olennaiseen sekä kyky yhdistää projektinhallintaan tehokkuutta ja luovuutta. [Charette, 2001d; Chin, 2004; Perminova *et al*, 2007]. Seuraavat vuodet näyttävätkin sen onko tämä kehityksen suunta oikea ja mihin suuntaa riskienhallinta on kehittymässä.

4. Ohjelmistokehitys ja perinteiset menetelmät

4.1. Ohjelmistokehityksen tausta

Ohjelmistokehitys (Software engineering) termi esiteltiin ensimmäisen kerran maailmalle jo vuonna 1968 ja siitä lähtien se on ollut kiinteä osa insinööritieteitä [Haikala ja Märijärvi, 2004; Welke, 1981]. Ohjelmistokehityksestä näkee myös käytettävän synonyyminä termiä ohjelmistotuotanto. Tässä tutkielmassa käytetään jatkossa termiä ohjelmistokehitys, koska tutkielmassa keskitytään ohjelmistoprojektien riskienhallinnan tutkimiseen erilasissa ohjelmistokehitysmenetelmissä.

Ohjelmistokehitys voidaan määritellä joukoksi tiettyjä toimenpiteitä, tehtäviä ja tekniikoita, jotka johtavat erilaisiin ohjelmistotuotteisiin. Näiden tehtävien voidaan aina katsoa alkavan vaatimustenmäärittelystä ja päättyvän erilaisten tehtävien kautta, käytettävästi kehitysmallista riippuen, valmiin ohjelmistotuotteen luovutukseen ja tarvittaessa sen ylläpitoon [Sommerville, 2007; Welke, 1981].

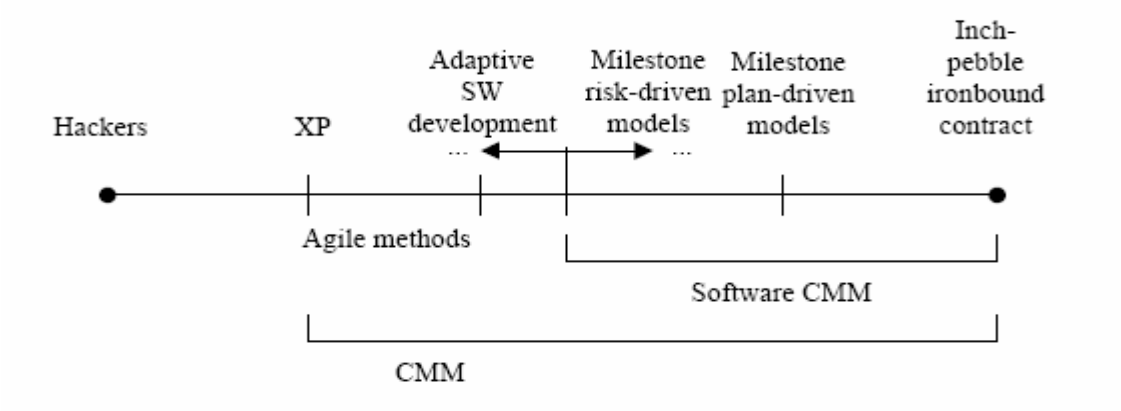
4.2. Menetelmien luokittelu

Erilaiset ohjelmistokehityksen tavat, teoriat ja menetelmät sekä niiden kannattajat voidaan jakaa käytännössä kahteen luokkaa, toiminnan painopisteiden ja niiden tuottaman dokumentaation mukaan. [Haikala ja Märijärvi, 2004] Ensimmäistä näistä kahdesta luokasta voidaan kutsua suunnitelmaohjautuviksi menetelmiksi, vaihejako malleiksi tai perinteisiksi ohjelmistokehitysmenetelmiksi (Plan-driven Models). Näissä kaikki käytännöntoiminta, kuten koodaus, testaus, aikataulu ja muu tekeminen suunnitellaan sekä dokumentoidaan mahdollisimman tarkasti vaihevaiheelta aina pienintä yksityiskohtaa myöten [Sommerville, 2007]. Kaikkia vaiheesta toiseen eteneviä ohjelmistokehitysmalleja kutsutaan tässä tutkielmassa perinteisiksi ohjelmistokehitysmenetelmiksi.

Kaikki muut ohjelmistokehityksen menetelmät voidaan laskea kuuluvan toiseen luokkaan. Näitä kutsutaan tässä tutkielmassa ketteriksi ohjelmistokehitysmenetelmiksi (Agile Software Development Methods) [Abrahamsson *et al*, 2002; AgileAlliance, 2001; Boehm, 2002]. Ketterissä menetelmissä ohjelmistokehitys perustuu tarkan suunnittelun ja dokumentoinnin sijasta ihmisten välisillä keskusteluilla tapahtuvaan tiedon keräämiseen ja sitä kautta korkeaan osaamisen hyödyntämiseen.

Jos erilaisia ohjelmistokehitysmenetelmiä sijoitettaisiin suoralle siten, että menetelmien ääripäät olisivat suoran vastakkaisissa päissä, niin toisessa ääripäässä olisivat

hakkereiden käyttämät menetelmät ja toisessa pilkuntarkat sopimukset. Tätä jakoa on hahmoteltu kuvassa 14 [Boehm, 2002].



Kuva 14: Ohjelmistokehitysmenetelmien jaottelu [Boehm, 2002, s.65]

Ketterät menetelmät sijoittuvat hieman hakkereista katsottuna kohti keskikohtaa ja perinteiset menetelmät taas kirjallisista sopimuksista katsottuna hieman kohti keskikohtaa. Keskikohdan molemmin puolin, ketterien ja perinteisten menetelmien väliin, voidaan sijoittaa vielä monia muita ohjelmistokehityksen menetelmiä ja tekniikoita. Kuten esimerkiksi mukautuva ohjelmistokehitys, inkrementaaliset ja iteratiiviset menetelmät sekä formaalit mallit [Boehm, 2002; Haikala ja Märijärvi, 2004]. Ketterät ja perinteiset ohjelmistokehitysmenetelmät ovat siis lähes toistensa vastakohtia.

Kaikki ohjelmistokehityksen menetelmät ovat syntyneet tarpeesta. Tämän seurauksena on joko muokattu aikaisemmin käytettyä menetelmää tarpeeseen paremmin sopivaksi tai kehitetty kokonaan täysin uusi menetelmä. Kulloinkin käytettävän ohjelmistokehitysmenetelmän valinta tapahtuu ideaalissa tilanteessa täysin kehitteillä olevan sovelluksen ehdoilla. Sovelluksen toteuttajan tulee tällöin valita käytettävä menetelmä sovelluksen laajuuden, käytettävissä olevien resurssien ja toivotun lopputuloksen mukaan [Ruuska, 2007; Sommerville, 2007].

Käytännössä näin ei kuitenkaan juuri tapahdu, vaan ohjelmistokehitysmenetelmän valintaan vaikuttavat yleensä enemmän toteuttajan oma osaaminen ja vahvuudet. Toteuttajan aikaisemminkin menestyksekkäästi käyttämän ja hyväksi havaitseman toimintamallin käyttämistä voidaan pitää perusteltuna silloin, kun se ei tarkoita suurien kompromissien tekemistä projektin toteutuksen kustannuksella.

4.3. Perinteiset ohjelmistokehitysmenetelmät

Perinteiset ohjelmistokehitysmenetelmät ovat kaikkein vanhimpia ohjelmistokehityksen menetelmiä. Kun tietokoneet alkoivat yleistyä 1900-luvun jälkipuoliskolla ja erilaisia tietokoneohjelmia ryhdyttiin käyttämään asioiden hoitamiseen, niin ohjelmistokehitysmenetelmien tavoitteeksi tuli systemaattisesti hallita ohjelmistojen kehitysprosessia. Erityisen tärkeäksi tehtäväksi nousi samalla myös ohjelmistojen laadun parantaminen ja tätä kautta ohjelmointivirheiden määrää vähentäminen. Ohjelmistokehitysmenetelmien perusajatukset ja tehtävät eivät ole juuri muuttuneet vuosien kuluessa [Haikala ja Märijärvi, 2004; Sommerville, 2007; Welke, 1981].

Perinteisessä ohjelmistokehityksessä projektit nähdään yleensä eräänlaisena elinkaarena, joka on selkeästi vaiheesta toiseen etenevänä aikaan sidottu looginen prosessi. Prosessi alkaa pääsääntöisesti aina tilaajan, joko yrityksen ulkoinen asiakas tai yrityksen sisäinen yksikkö, tarpeiden määrittelystä ja päättyy valmiin tuotteen ylläpitoon. Tunnetuimpia ja samalla eniten käytettyjä näistä perinteisistä ohjelmistokehitysmalleista ovat vesiputousmalli, prototyyppimalli ja spiraalimalli [Haikala ja Märijärvi, 2004; Sommerville, 2007].

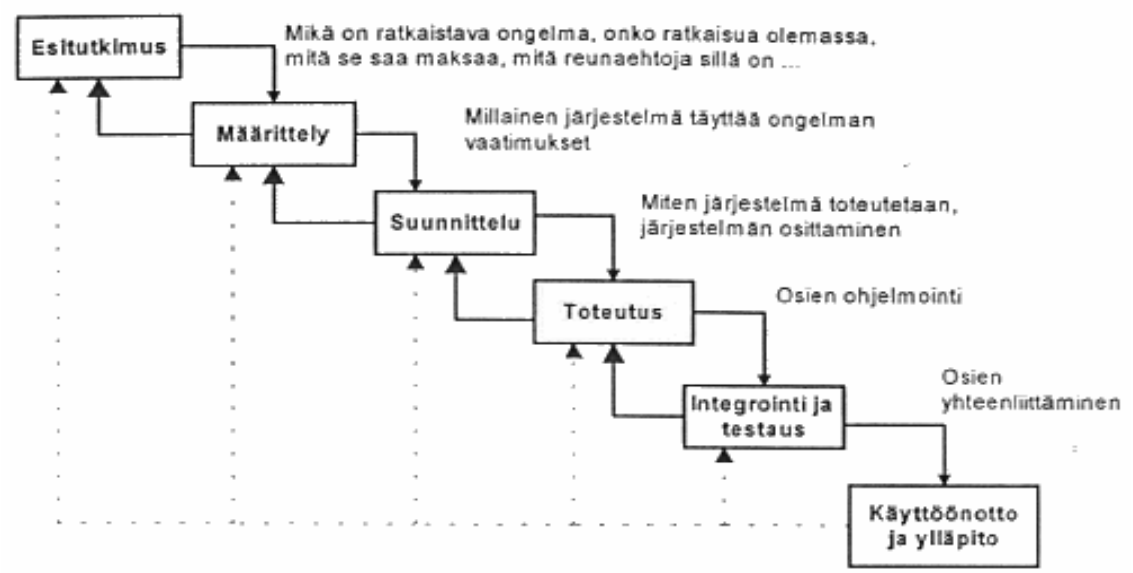
Kaikissa näissä ohjelmistokehitysmalleissa toiminta nähdään, joukkona tehtäviä ja vaiheita, jotka seuraavat aina toinen toistaan. Peräkkäisten työvaiheiden tuloksena syntyy aina jokin valmis projektin osa. Edellinen vaihe toimii samalla aina myös seuraavan työvaiheen osan pohjana. Tämä tarkoittaa sitä, että ennen aikaisemman vaiheen valmistumista projektin jatkaminen ei ole mahdollista. Kaikkien perinteisten ohjelmistokehitysmallien toimintavaiheet ovat tarpeiden määrittely, suunnittelu, toteutus, testaus, käyttöönotto ja ylläpito. Toiminnan perusajatukset pysyvät yleensä samoina, vaikka näiden esiintymistiheys ja toteutustapa saattavatkin vaihdella eri mallien välillä [Haikala ja Märijärvi, 2004; Ruuska, 2007; Sommerville, 2007].

Kaikkiin perinteisten ohjelmistokehitysmenetelmien työvaiheisiin liittyy olennaisena osana myös erittäin paljon dokumentaatiota. Dokumentoinnin tarkoituksena on varmistaa vaiheen olevan valmistunut halutunlaisena sekä kirjata muistiin tehdyt asiat. Lisäksi perinteissä malleissa on tyypillistä, että asiakkaan osallistuminen tuotteen kehitykseen tapahtuu pääasiassa vain määrittely- ja suunnitteluvaiheissa [Haikala ja Märijärvi, 2004; Ruuska, 2007; Sommerville, 2007].

4.3.1. Vesiputousmalli

Vesiputousmalli on perinteinen ohjelmistokehitysmalli, jonka Winston W. Royce kehitti vuonna 1970. Vielä tänä päivänäkin, malli on erittäin paljon käytetty. Malli on myös

toiminut pohjana lukuisille uudemmille kehitysmalleille. Vesiputousmallin toiminta on esitetty kuvassa 15 [Haikala ja Märijärvi, 2004].



Kuva 15: Vesiputousmalli [Haikala ja Märijärvi, 2004, s.36]

Nimensä mukaisesti, vesiputousmallissa ohjelmistokehitys tapahtuu lineaarisesti vesiputouksen tavoin. Mallissa kehitys on jaettu osiin, jolloin kaikki vaiheet seuraavat aina toista ollen samalla riippuvaisia edellisestä. Tästä syystä joskus voidaan myös joutua palaamaan takaisin edelliseen vaiheeseen esimerkiksi seuraavassa vaiheessa huomattujen ongelmien vuoksi. Todellisuudessa tämä takaisin palaaminen on kuitenkin erittäin harvinaista ja vaikeaa. Vesiputousmallista on myös olemassa hieman toisistaan poikkeavia sovelluksia, mutta pääsääntöisesti mallin vaiheet ovat esitutkimus, määrittely, suunnittelu, toteutus, integrointi ja testaus sekä käyttöönotto ja ylläpito. Olennaisena osana kaikkiin näihin vaiheisiin kuuluu myös dokumentaatio [Haikala ja Märijärvi, 2004; Sommerville, 2007].

Vesiputousmallin varsinainen toiminta alkaa aina esitutkimuksella. Esitutkimuksen tarkoituksena on löytää toteutettavalle ohjelmistolle yleiset toimintaperiaatteet ja vaatimukset. Yleensä näiden lähtökohtana ovat tilaajan esiintuomat ongelmat tai tarpeet, joiden pohjalta varsinainen totuttaja voi alkaa kartoittaa erilaisia ratkaisuja näihin ongelmiin. Esitutkimusvaihe onkin selkeästi yhteydessä määrittelyvaiheeseen ja siihen voi myös liittyä alustavan projektisuunnitelman teko [Haikala ja Märijärvi, 2004].

Koko määrittelyvaiheen ajan asiakkaan tarpeita ja vaatimuksia analysoidaan syvällisemmällä tasolla. Tämän seurauksena voidaan muodostaan toteutettavan ohjelmiston toiminnalliset, tekniset ja muut vaatimukset sekä näiden kaikkien mahdolliset

rajoitteet. Nämä kaikki kirjataan muistiin vaatimusmäärittely dokumenttiin. Samalla myös tarkennetaan aikaisemmin aloitettua alustavaa projektisuunnitelmaa [Haikala ja Märijärvi, 2004].

Suunnitteluvaiheessa keskitytään miettimään ohjelmiston erilasia teknisiä toteutustapoja, joita voivat olla yleisen ohjelmistoarkkitehtuurinsuunnittelu tai joskus jopa yksittäisten moduuliensuunnittelu. Ohjelmiston tyypistä ja toteuttajille annattavien vapauksien määrästä riippuen, suunnitteluvaiheen lopputuloksena on yksi tai useampi yksityiskohtainen tekninen dokumentti [Haikala ja Märijärvi, 2004].

Varsinaisen toteutusvaiheen ainoana tehtävänä on ohjelmoida määrittely- ja suunnitteluvaiheissa dokumentoitu ohjelmisto sekä dokumentoida tehty ohjelma. Toteutetun ohjelmiston pitää sisältää kaikki dokumentoidut ominaisuudet ja olla toimiva kokonaisuus ennen kuin voidaan siirtyä seuraavaan vaiheeseen [Haikala ja Märijärvi, 2004].

Testausvaiheen tarkoituksena on varmistaa ohjelman oikea toiminta kaikissa tilanteissa. Ohjelmiston testaus voidaan hoitaa useilla eri tavoilla. Vesiputousmallissa kaikki dynaamiset testaustavat, kuten automaattinen ja manuaalinen testaus, ovat kaikkien käytetyimpiä, mutta asiakkaan osallistumista vaativia staattisia menetelmiä, kuten katselmointeja, ei juurikaan käytetä. Testausvaiheessa löydetyt virheet dokumentoidaan testausraporttiin sekä myös korjataan ennen siirtymistä seuraavan vaiheeseen [Haikala ja Märijärvi, 2004].

Käyttöönottovaiheessa asiakas ottaa käyttöönsä toteutetun ohjelmiston. Mikäli asiakkaan käyttöönoton jälkeen havaitaan ongelmia tai puutteita ohjelman toiminnassa, niin ylläpitovaiheessa nämä pyritään ratkaisemaan ylläpitosopimusten mukaisesti [Haikala ja Märijärvi, 2004; Sommerville, 2007].

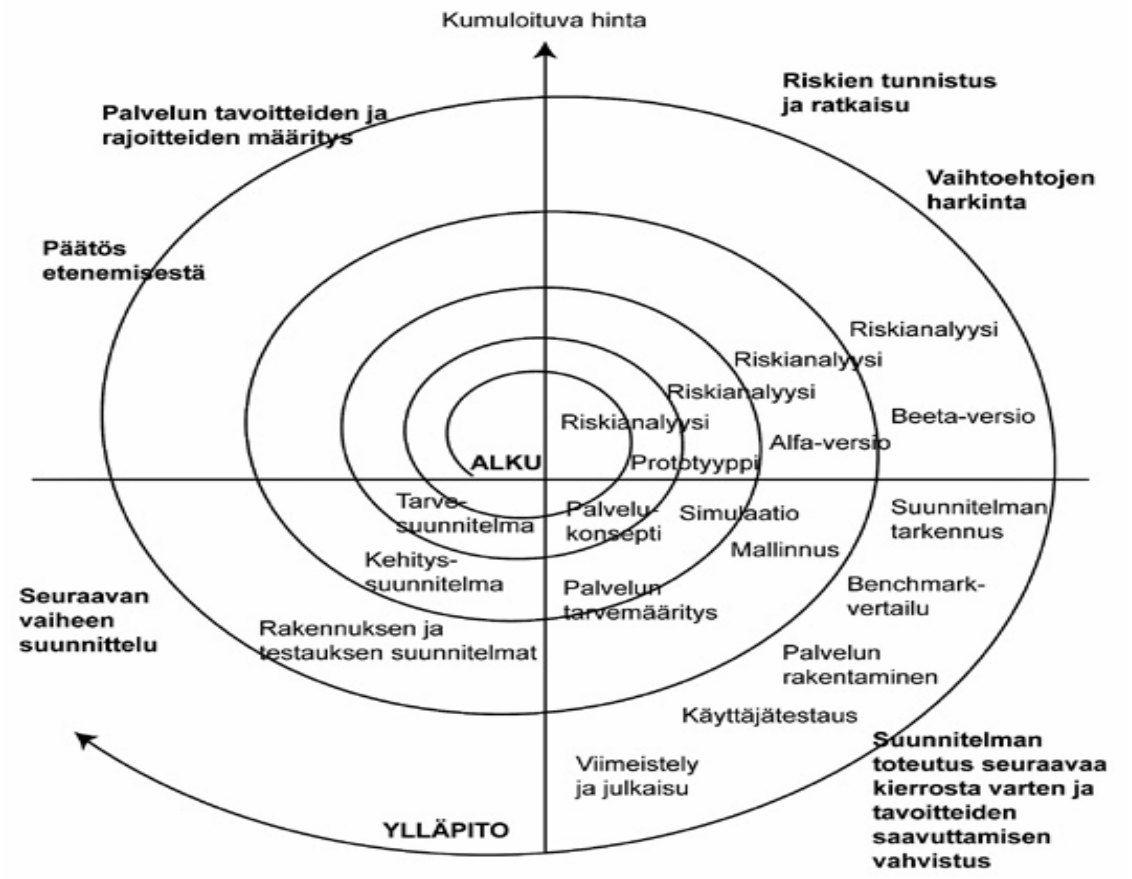
Vesiputousmallissa on sekä hyviä että huonoja puolia. Hyvinä puolina voidaan pitää perusteellista suunnittelutyötä ja johdonmukaisuutta. Samalla kuitenkin varsinaiseen toteutukseen käytetään vain vähän aikaan. Lisäksi asiakkaan osallistuminen kehitykseen on varsin minimaalista, jolloin haluttuun lopputulokseen pääseminen on sitä vaikeampaa mitä haasteellisempi ohjelmiston toteutusalue tai aihepiiri on kyseessä.

Lisäksi vesiputousmallia on kritisoitu koko sen olemassa olon ajan liiallisesta keskittymisestä dokumentaatioon muun toiminnan kustannuksella [Boehm, 2002; Charette, 2001d; DeMarco and Lister, 1987]. Myös kehitysmallin isänä pidetty Winston W. Royce, kritisoi malliaan julkisesti useaan otteeseen, muun muassa toteamalla sen

soveltuvan hyvin vain tiettyihin tilanteisiin. Käytännössä tämä tarkoittaa vain sellaisia tilanteita, joissa toteutettavan ohjelmiston kaikki vaatimukset ovat alusta asti selvillä ja niiden ei oleta muuttuvan. Nykyisen maailman ja organisaatioiden toimintaympäristön monimutkaisuuden tuntien voi tällaisen tilanteen löytäminen olla erittäin vaikeaa – ellei jopa mahdotonta.

4.3.2. Spiraalimalli

Spiraalimalli on Barry Boehmin vuonna 1988 kehittämä perinteinen ohjelmistokehitysmalli. Mallin suunnittelun pohjana on käytetty monia aikaisempia kehitysmalleja, kuten vesiputous- ja evoluutiomalleja sekä code-fix -menetelmää. Malli ei ollut ensimmäinen, jossa käytettiin iteraatioihin eli kehitysvaiheiden tai -syklien toistamiseen perustuvaa kehitystapaa. Mutta Boehmin [1988] perustelu iteraatioiden käytölle tekivät siitä ainutlaatuisen aikaisempiin malleihin verrattuna. Merkittävä ero muihin malleihin nähden on myös riskien käyttäminen toiminnan ohjauksessa ja dokumentoinnin tehostaminen vesiputousmalliin verrattuna. Spiraalimalli on esitetty kuvassa 16 [Pressman, 2010].



Kuva 16: Spiraalimalli [Boehm, 1988, s.64]

Spiraalimallissa ohjelmistokehitys tapahtuu siis eripituisten iteraatioiden kautta. Mallin toteuttaminen aloitetaan kuvassa 16 keskeltä ja päätetään ulkoreunaan, jolloin sen vaiheet ovat suunnittelu, riskinanalyysi, kehitys ja testaus sekä arviointi. Vertikaalisella akselilla on kuvattu kehityksen kanssa samaan aikaan tapahtuvaa kustannusten nousua jokaisen kierroksen aikana [Boehm, 1988; Pohjonen, 2002; Ruuska, 2007].

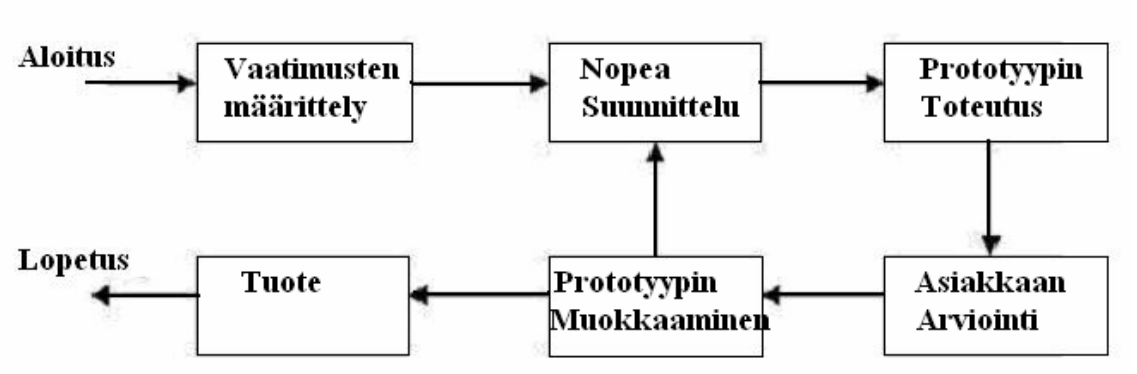
Tyypillinen iteraation pituus voi olla 2–24 kuukautta. Jokaiseen uuteen kierrokseen kuuluu useita eri vaiheita ja ne toistuvat aina jokaisessa kierroksessa. Iteraatioiden tarkoituksena on myös vähentää riskejä jokaisen kierroksen aikana sekä aktivoida asiakas osallistumaan projektiin entistä tiiviimmin. Kehityskierrosten määrää ei ole rajoitettu, joten niitä toistetaan järjestelmällisesti niin kauan kunnes saavutetaan haluttu lopputulos [Boehm, 1988; Sommerville, 2007; Pohjonen, 2002; Ruuska, 2007].

Spiraalimallissa yhdistyvät samaan aikaan lineaarinen ohjelmistokehitys ja protyyppimalli. Spiraalimallia on pitkään ja menestyksekkäästi käytetty erityisesti peliteollisuudessa sekä suurissa tai laajoissa ohjelmistoprojekteissa, koska se on erittäin riskitietoinen kehitysmalli ja sen avulla saadaan toimiva prototyyppi jo projekti alkuvaiheessa. Samalla spiraalimallia on kuitenkin myös kritisoitu sen projektien venymisen ja keston vuoksi sekä tästä johtuvien suurien kokonaiskustannusten vuoksi [Pohjonen, 2002; Pressman, 2010; Ruuska, 2007; Sommerville, 2007].

4.3.3. Prototyyppimalli

Prototyyppimallissa ohjelmistokehitys tapahtuu eriasteisten prototyyppien kautta. Prototyyppien avulla on tarkoitus ymmärtää paremmin asiakkaan vaatimuksia, ja sitä kautta toteuttaa parempia ohjelmistotuotteita. Usein prototyyppimallin synnyn taustana pidetään erityisesti vesiputousmallissa ilmenneitä vaatimusmäärittelyn ongelmia [Haikala ja Märijärvi, 2004; Pressman, 2010].

Toisin kuin vesiputousmallissa, prototyyppimallissa toteutettavan ohjelmiston vaatimuksia ei löydä lukkoon ennen toteutuksen aloittamista. Sen sijaan, ensimmäisen prototyypin toteutus aloitetaan alustavien vaatimusten pohjalta, niin nopeasti ja edullisesti kuin se on mahdollista. Tällöin toteuttajat ja asiakas voivat yhteistyössä sekä resursseja hukkaamatta tarkentaa ohjelmiston vaatimuksia jonkin konkreettisen pohjalta. Tämä edellyttää kuitenkin asiakkaan aktiivista osallistumista koko projektin ajan, jotta lopputulos vastaisi todellisia tarpeita. Prototyyppimallin toiminta ja eteneminen on esitetty kuvassa 17 [Haikala ja Märijärvi, 2004; Pressman, 2010; Sommerville, 2007].



Kuva 17: Prototyyppimalli [Pressman, 2010, s.84]

Prototyyppimallin alussa projektin toteuttaja kokoaa yhdessä asiakkaan kanssa ohjelmiston alustavat ja pääpiirteiset toiminnalliset vaatimukset sekä muut rajoitteet. Näiden pohjalta toteutetaan nopealla aikataululla ensimmäinen prototyyppi. Vielä tässä vaiheessa yleisellä tasolla olevat vaatimukset ovat riittäviä, koska monessa tapauksessa lopulliset vaatimukset eivät ole tässä vaiheessa selvillä. Usein myöskään asiakkaan oma osaaminen ei riitä yksityiskohtaisten vaatimusten määrittelyyn itsenäisesti, ja tästä syystä ne voivat olla useista suurista virheistä johtuen käyttökelvottomia. Jokaisen prototyypin toteutukseen liittyy aina myös suunnittelu- ja testausvaiheet sekä testauksessa ilmenneiden virheiden korjaus. Prototyyppimalliin kuuluu myös yleinen toiminnan seuranta ja raportointi, joka on kuitenkin muodoltaan muita perinteisiä ohjelmistokehitysmalleja merkittävästi vapaamuotoisempaa [Haikala ja Märijärvi, 2004; Pressman, 2010; Sommerville, 2007].

Tämän jälkeen ensimmäinen toimiva prototyyppi esitellään asiakkaalle ja sitä testataan sekä yhdessä että erikseen toteuttajan kanssa. Projektin kehitystyön jatkaminen on mahdollista siinä vaiheessa kun, uusia vaatimuksia on tullut esiin siinä määrin, että niistä voidaan muodostaa toisen prototyypin vaatimukset. Tämä onkin yksi prototyyppimallin tärkeimmistä vaiheista, josta riippuu lähes kokonaan projektin onnistuminen. Parhaassa tapauksessa asiakkaan tekemä prototyypin arviointi on kattava ja monipuolinen, jolloin se vastaan heidän todellisia tarpeitaan entistä paremmin ja uuden prototyypin jatkokehitys helpottuu merkittävästi. Samalla projekti myös etenee riittävästi käytössä oleviin resursseihin nähden [Pressman, 2010; Sommerville, 2007].

Toteutuksen päätökseen saamiseen on olemassa kaksi vaihtoehto. Uusia prototyyppejä voidaan suunnitella, toteuttaa, testata ja arvioida niin kauan, kunnes kaikki osapuolet ovat tyytyväisiä viimeiseen prototyyppiin. Aikaisempaa prototyyppiä käytetään siis aina uuden prototyypin pohjana, jolloin sen toiminnallisuutta ja ominaisuuksia lisätään tasaiseen tahtiin vaihe vaiheelta. Toinen tapa on hylätä ensimmäinen prototyyppi arvioinnin jälkeen ja koota sen pohjalta vain vaatimukset lopulliselle ohjelmistolle, jonka

toteuttaminen aloitetaan uudestaan alusta [Pressman, 2010; Ruuska, 2007; Sommerville, 2007].

Molemmissa viimeistely tavoissa on sekä hyvät että huonot puolensa. Usein valinta kuitenkin kohdistuu ensimmäiseen tapaan, koska siinä resursseja ei koeta heitettävän samalla tavalla hukkaan kuin jälkimmäisessä vaihtoehdossa. Toisaalta taas tämän nopeasti kootun ensimmäisen prototyypin päälle rakentamisessakin on myös omat riskinsä. Näitä voivat olla esimerkiksi valmiin ohjelmisto jatkokehityksen rajoitukset, laajennettavuuden tai yhteensopivuuden puute, rakenteen monimutkaisuus sekä suuret ylläpitokustannukset. Tästä syystä toteuttajan onkin erittäin tärkeä informoida asiakasta hyvissä ajoin näistä prototyyppimallin käyttöön liittyvistä mahdollisista ongelmakohdista ja rajoitteista, kehityksen aikana sekä sen jälkeen [Pressman, 2010; Ruuska, 2007 Sommerville, 2007].

4.4. Riskienhallinta perinteisissä menetelmissä

Perinteisessä ohjelmistokehityksessä projektinhallintaa pidetään pääsääntöisesti olennaisena osana projektia, jossa riskienhallinta on yksi sen osista. Käytännössä kuitenkin vain hyvin pieneen osaan projekteista kuuluu minkäänlaista aktiivista riskienhallintaa, jonka menetelmiä ja toimintatapoja kuvattiin yksityiskohtaisesti luvussa 3.

Janne Ropposen [1999] ja Mika Kontion [2001] väitöskirjoissa esittämät luvut riskienhallinta menetelmien käytön aktiivisuudesta erilaissa ohjelmistoprojekteissa vaihtelevat 20–25 % välillä. Tätä voidaan pitää erittäin huolestuttava. Molemmat tutkimukset on julkaistu jo 2000-luvun taitteessa, josta voidaan päätellä niiden tulosten koskevan pääosin juuri perinteisiä ohjelmistokehityksen menetelmiä, koska ketterien menetelmien syntyminen ja aktiivinen käyttöönottaminen tapahtui vasta tämän jälkeen.

Useissa perinteisissä ohjelmistokehityksen menetelmissä onkin hyvin tavallista, että mahdollisiin ongelmiin eri juuri varauduta etukäteen, jolloin esiin tulevat ongelmat korjataan sitä mukaa kun niitä ilmenee. Vaikka pitkällä aikavälillä tämä ei ole millään tavalla järkevää tai tehokasta. Useat tutkijat ovatkin osoittaneet tutkimuksissaan, että on huomattavasti halvempaa välttää riskejä, kuin korjata niiden aiheuttamia vahinkoja jälkikäteen. Jopa erittäin pienet, vain 2–8 % panostukset, projektin kokonaisbudjetista vaikuttavat merkittävästi riskien vähentymiseen [Boehm, 1988, 1989, Charette, 1996, DeMarco and Lister, 1987; Keil et al, 1998; Kontio, 2001, Lyytinen and Hirschheim, 1987; Ropponen, 1999].

Kuitenkin ensimmäinen henkilö joka konkreettisesti teki asialle jotakin, on nykyisin riskienhallinnan pioneerina tunnettu Professori Barry Boehm. Hän muutti perinteistä vesiputousmallia riskitietoisemmaksi julkaistessaan uuden spiraalimallina tunnetun ohjelmistokehitysmenetelmän, johon hän oli sisällyttänyt kiinteästi riskienhallintaa riskianalyysin muodossa. Mutta ennen kaikkea, Boehm [1988; 1989] julkaisi myös konkreettiset ohjeet riskienhallintaan spiraalimallissa.

Kaikesta tästä huolimatta esimerkiksi kattavia ja yksityiskohtaisia projektien riskienhallinnan oppaita perinteisiä ohjelmistokehityksen menetelmiä varten on erittäin vaikea löytää. Lisäksi näiden menetelmien muussa projektihallinta kirjallisuudessa riskienhallinta käsitellään vain pienellä pintaraapaisulla tai enimmillään muutaman kappaleen pituisena kokonaisuutena.

5. Ketterät ohjelmistokehitysmenetelmät

5.1. Tausta

Kun ohjelmistokehittäjät 1990-luvun lopulla huomasivat, että perinteiset ohjelmistokehitysmallit eivät enää toimineet samalla tavalla kuin aikaisemmin, niin pieni joukko heistä ryhtyi yhdistelemään uusia teorioita ja ideoita vanhoihin perinteisiin menetelmiin. Tämä toi mukanaan paljon uusia kevyitä toimintatapoja perinteiseen ohjelmistokehitykseen. Lopulta näiden muutosten seurauksena syntyi suuri määrä uusia ohjelmistokehitysmenetelmiä, joista uusimpia kutsutaan nykyään ketteriksi ohjelmistokehitysmenetelmiksi (Agile Methods) [Abrahamsson *et al*, 2002; Beck and Andres, 2006; Boehm, 2002; Charette, 2001ab; Highsmith, 2000; Schwaber, 2000].

Useimmissa näistä uusissa ohjelmistokehitysmenetelmissä pidettiin alusta asti tärkeänä kasvokkain tapahtuvaa kommunikaatiota eri tahojen välillä, tiimien muodostamista ja uusia tehokkaampia ohjelmointitekniikoita. Näiden toimintatapojen avulla organisaatioissa uskottiin pystyttävän paremmin vastaamaan tulevaisuuden mukaan tuomiin uusiin haasteisiin sekä tarjoamaan ratkaisuja erilaisiin tilanteisiin ja nopeasti muuttuviin vaatimuksiin [Cockburn, 2007; Glass, 2001; Highsmith, 2000; Schwaber, 2000].

Ohjelmistokehitys onkin yksi niistä tietotekniikan aloista, joka on jatkuvasti erilaisten muutosvoimien ja -paineiden kohteina. Erilasia ohjelmistotuotteita tarvitsevat asiakkaat asettavat ohjelmistokehittäjille sekä ohjelmistoille suuria odotuksia ja tavoitteita. Erityisesti kustannukset, aikataulu ja toiminnallisuus ovat niitä asioita, jotka tuottavat jatkuvasti päänvaivaa ohjelmistokehitys liiketoimintaa harjoittaville organisaatioille. Tästä syystä sekä asiakkaiden että organisaatioiden kiinnostus ja selkeä tarve joustavampia malleja kohtaan on jatkuvasti kasvanut.

Vuonna 2001 Yhdysvalloissa joukko ohjelmistoalan moniosaajia kokoontui yhteen pohtimaan näiden uusien ohjelmistokehitysmenetelmien todellisia mahdollisuuksia ja tehokkaita hyödyntämistapoja. Kokoontumisen yhteydessä pidetyssä workshopissa pyrittiin muun muassa löytämään yhtenäisyyksiä näiden uusien menetelmien välillä. Kokoontumisen tuloksena esiteltiin ja otettiin ensikertaa käyttöön ”ketterä” (Agile) termi, kuvaamaan näitä uusia ohjelmistokehityksen menetelmiä [Abrahamsson *et al*, 2002; AgileAlliance, 2001; Beck and Andres, 2006; Cockburn, 2007; Highsmith, 2000; Scum, 2009]. Kaikkein merkittävin asia joka tässä tapahtumassa julkaistiin, oli kuitenkin ketterän ohjelmistokehityksen manifesti (Manifesto for Agile Software Development) [AgileManifesto, 2001]. Tämä julkaisu määrittelee yleiset arvot ja periaatteet kaikelle ketterälle ohjelmistokehitykselle.

5.2. Perusarvot ja yleiset periaatteet

Ketterä ohjelmistokehitys pyrkii aina kussakin tilanteessa parhaiten tarkoitukseen sopivan ohjelmiston kehittämiseen mahdollisimman nopeasti, kevyesti ja kaikin puolin tehokkaasti. Samalla kuitenkin kaikki toiminta tapahtuu suhteellisen vapaamuotoisesti. Mutta tästä huolimatta, toiminta ei silti missään nimessä ole sattumanvaraista, eikä se muistuta täydellistä kaaosta. Hallittu keskitie, täydellisen vapauden ja tiukan suunnitelmallisen tekemisen välillä, onkin paljon todenmukaisempi kuvaus ketterästä ohjelmistokehityksestä. Viime aikoina ketterät menetelmät ovatkin saaneet yhä enemmän käyttäjiä.

Ketterän ohjelmistokehityksen manifesti [AgileManifesto, 2001], koostuu 4 perusarvosta, joihin usein lisätään ketterien menetelmien käyttöä edistävän Agile Alliance [2001] -nimisen järjestön 12 periaatetta. Yhdessä nämä muodostavat ketterän ohjelmistokehityksen perusarvot ja yleiset periaatteet. Käyttämällä näitä perusarvoja ja periaatteita yleisenä organisaation projektitoiminnan ohjenuorana, myös soveltamalla sekä etsimällä omanlaisentavan toimia, on mahdollista saada ketterän ohjelmistokehityksen mukanaan tuomat hyödyt omiin ohjelmistohankkeisiin. Ketterän ohjelmistokehityksen perusarvot ja periaatteet ovat seuraavat [AgileManifesto, 2001; AgileAlliance, 2001].

PERUSARVOT:

”*Me etsimme parempia keinoja ohjelmistojen kehittämiseen tekemällä sitä itse ja auttamalla siinä muita. Tässä työssämme olemme päätyneet arvostamaan.*”
[AgileManifesto, 2001]

1. **Yksilöitä ja vuorovaikutusta** enemmän kuin prosesseja ja työkaluja
2. **Toimivaa sovellusta** enemmän kuin kokonaisvaltaista dokumentaatiota
3. **Asiakasyhteistyötä** enemmän kuin sopimusneuvotteluita
4. **Muutokseen reagoimista** enemmän kuin suunnitelman noudattamista

”Vaikka oikeallakin puolella olevilla tekijöillä on arvoa, niin me arvostamme vasemmalla olevia asioita enemmän.”
[AgileManifesto, 2001]

PERIAATTEET:

1. Korkein prioriteettimme on asiakastyytyväisyyden saavuttaminen arvokkaan ohjelmiston nopealla ja jatkuvalla toimittamisella.

2. Ota vaatimusmuutoksia vastaan vielä kehityksen loppuvaiheissa. Ketterä prosessi valjastaa muutoksen asiakkaan kilpailueduksi.

3. Toimita toimiva ohjelmisto tihein aikavälein, kahdesta viikosta muutamaan kuukauteen suosien lyhyttä aikaskaalaa.

4. Liiketoiminta-asiantuntijoiden ja kehittäjien on työskenneltävä yhdessä päivittäin koko projektin ajan.

5. Rakenna projektit motivoituneiden yksilöiden ympärille. Anne heille heidän tarvitsemansa ympäristö ja tuki, sekä luota heidän saavan työnsä tehdyksi.

6. Tehokkain ja eniten tuloksia tuottava tapa siirtää tietoa kehitystiimille sekä sen sisällä on tavata kasvokkain.

7. Toimiva ohjelmisto on edistyksen päämittari.

8. Ketterät prosessit edistävät kestävää kehitystä. Sponsorien, kehittäjien sekä käyttäjien pitäisi pystyä säilyttämään tasainen tahti rajattomasti.

9. Jatkuvan huomion kiinnittäminen tekniseen erinomaisuuteen sekä hyvään suunnitteluun parantaa ketteryyttä.

10. Yksinkertaisuus – tekemättömän työn maksimoinnin taito – on elintärkeää.

11. Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat kumpuavat itseorganisoituvista tiimeistä.

12. Tasaisin väliajoin tiimi reflektoi, miten voisi tulla tehokkaammaksi ja sitten virittää ja säätää käytöstään sen mukaisesti.
[AgileManifesto, 2001]

Yllä olevia arvoja ja periaatteita käytetään ainakin soveltuvin osin useissa erilaisissa ketterissä ohjelmistokehitysmenetelmissä ja malleissa. Näille kaikille projektien toteutusmenetelmille on yhteistä myös inkrementaalinen ja/tai iteratiivinen ohjelmistokehitys. Iteratiiviselle ohjelmistokehitykselle on tyypillistä, että suunnittelua ja

toteutusta tehdään pienimmissä osissa ja prosessia toistetaan niin kauan, kunnes saavutetaan haluttu lopputulos. Toimintaan liittyy myös tilaajan tai hänen edustajansa tiivis mukanaolo [AgileAlliance, 2001; Cockburn, 2007; Highsmith, 2000]. Tällä tavoin toteutettuna ohjelmisto kehittyy inkrementaalisesti eli koko ajan kasvaen kohti lopullista muotoaan, tilaajan valvovien silmien alla sekä korjaavien tai ohjaavien kommenttien saattelemana.

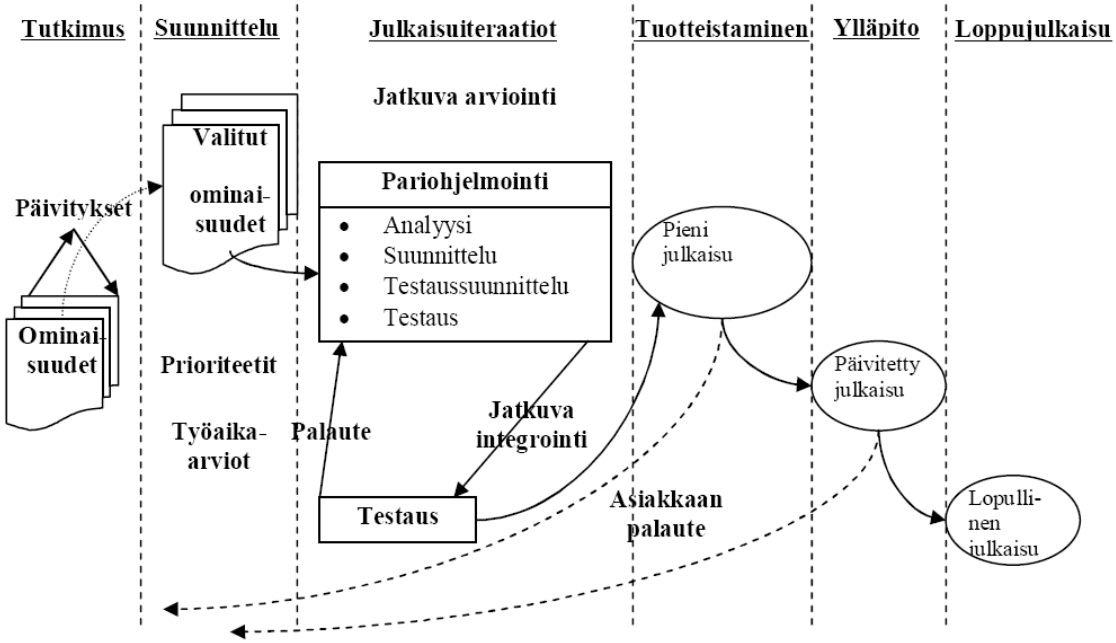
Kaikkien ketterien ohjelmistokehitysmenetelmien tärkein tavoite on kuitenkin toimivan ohjelmiston toteuttaminen. Menetelmissä ohjelmiston prototyypin toiminnallisuuksien määrää käytetäänkin yleisesti tärkeimpänä ohjelmistoprojektin etenemisen mittarina. Eroja mallien välille tulee ainoastaan ohjelmiston osien toteutuksen tärkeydessä, painotuksessa ja soveltamisessa käytännössä. Osa ketteristä menetelmistä on myös varsin yleisluonteisia, jopa täysin abstrakteja. Kun taas toiset ovat, hieman ketterän ohjelmistokehityksen idean vastaisesti, hyvinkin tarkkoja varsinaisen ohjelmointityön ohjeita [Boehm, 2002; Cockburn, 2007].

Kaikkien näiden menetelmien kattava esittely ja vertailu ei tässä yhteydessä ole mahdollista tai edes järkevää. Niinpä tyydyn tässä tutkielmassa käsittelemään vain joitakin tunnetuimpia ja eniten käytettyjä näistä ketteristä menetelmistä. Ja nämäkin vain esittely luontoisesti ja pääpiirteittäin niiltä osin, kun se tämän tutkielman aiheen kannalta on olennaista.

5.3. Extreme Programming (XP)

Hyvin usein kaikkien ketterien ohjelmistokehitysmenetelmien alkuna pidetään, vuonna 1999, yhdysvaltalaisen Kent Beckin ja hänen kollegoidensa kanssa esittelemää Extreme Programming (XP) menetelmää [Beck and Andres, 2006]. Tosiasia on kuitenkin, että Beck [2006] kuitenkin vain kokosi yhteen joukon aikaisempia ideoita ja malleja muodostaakseen niistä oman teoriansa [Abrahamsson *et al*, 2002]. Tästäkin huolimatta XP-menetelmää pidetään yleisesti kaikkein tunnetuimpana ja uraauurtavimpana kaikista ketteristä ohjelmistokehityksen menetelmistä. Sillä onkin selkeitä yhteyksiä ja vaikutuksia muihin teorioihin ja menetelmiin [Extreme Programming, 2002].

Yleisesti XP-menetelmää voidaan kuvata sosiaalisen muutoksen menetelmäksi, koska se vaatii onnistuakseen täydellistä luopumista vanhoista toimintamalleista ja tavoista. XP-menetelmässä toiminta etenee 6 vaiheen kautta, jotka ovat tutkimus, suunnittelu, julkaisuiteraatiot, tuotteistaminen, ylläpito ja loppujulkaisu. Yleinen XP-menetelmän toiminnan eteneminen ja työvaiheet on esitetty kuvassa 18.



Kuva 18: Extreme Programming menetelmän eteneminen ja työvaiheet [Beck and Andres, 2006; Abrahamsson *et al,* 2002, s.19]

Yleisesti Extreme Programming menetelmää käyttävässä projektissa toiminta on keskittynyt vahvasti varsinaisen ohjelmointityön ympärille. Ihanteellisessa tilanteessa XP-menetelmä etenee aina suoraan kaikkien välivaiheiden kautta tutkimuksesta projektin loppujulkaisuun eli päätökseen asti. Kuitenkin niin, että tarvittaessa kaikista vaiheista voidaan aina palata myös taaksepäin ja takasin edellisiin vaiheisiin. [Abrahamsson *et al,* 2002; Beck and Andres, 2006]

XP-menetelmä alkaa tiivisti yhdessä tilaajan kanssa tehtävästä tutkimusvaiheesta, joka voi kestää projektin koosta ja muista ominaisuuksista riippuen yhdestä viikosta muutamaan kuukauteen asti. Tutkimusvaiheen alussa projektin tilaajan ainoana tehtävänä on kirjoittaa lyhyesti ja omin-sanoin, niin sanotuille tarinakorteille, jokaisen uudelta ohjelmistolta/järjestelmältä vaadittavan ominaisuuden kuvaus. Tämän jälkeen projektin toteuttajat kartoittavat erilasia tekniikoita ja työvälineitä, joiden avulla paperilla olevat asiat pystytään mallintamaan ohjelmistoon mahdollisimman tehokkaasti. Vaiheen lopuksi hahmotellaan ensimmäisen prototyypin alustava suunnitelma [Abrahamsson *et al,* 2002; Beck and Andres, 2006].

Tämän jälkeen menetelmä etenee korkeintaan viikon kestävään suunnitteluvaiheeseen, jonka tarkoituksena on antaa tilaajan tarinakorttien ominaisuuksille prioriteetit. Toteuttajien tehtävä on arvioida kunkin kortin ominaisuuden toteuttamiseen vaadittavat resurssit, ja kirjata ne tarinakortteihin. Kun kaikki kortit on käyty läpi, kortit järjestetään tai ryhmitellään toteutusaikataulun mukaisesti vapaasti nimettäviin kategorioihin.

Kategorioita voivat olla esimerkiksi valmiit, kesken ja ei-aloitettu. Tarinakorttien järjestystä voidaan kuitenkin jatkuvasti muuttaa ja niiden määrää voidaan myös tarvittaessa lisätä [Abrahamsson *et al,* 2002; Beck and Andres, 2006].

Kun projektin aikataulu on sovittu, niin tämän jälkeen voidaan aloittaa julkaisuiteraatiovaihe. Tyypillinen XP projekti sisältää useita iteraatio kierroksia. Iteraation sisältöön kuuluu kerrallaan vain yhden tarinakortin ominaisuuden toteuttaminen ja testaus, aikataulun mukaisessa järjestyksessä. Yleensä työt aloitetaan ohjelmiston perusarkkitehtuurin rakentamisesta ja siitä edetään aina kohti pienempiä yksityiskohtia. Kukin iteraatio siis rakentaa sekä laajentaa ohjelmistoa palapalalta eteenpäin. Viimeisen tarinakortin ominaisuuksien toteuttamisen ja testauksen jälkeen, ohjelmisto on valmis tuotteistusvaiheen kokonaisvaltaisen testauksen tekemiseen. Myös tuotteistusvaihe saattaa sisältää useita testikierroksia ja testitapoja, aina ohjelmiston suorituskyvystä virheiden käsittelyyn sekä niistä palautumiseen asti [Abrahamsson *et al,* 2002; Beck and Andres, 2006].

Ylläpitovaiheessa ohjelmisto otetaan rajoitetusti käyttöön tilaajan kanssa. Tämän vaiheen tarkoituksena on että, uutta ohjelmistoa käyttävät henkilöt pääsevät ennen lopullista projektin päättämistä sanomaan oman mielipiteensä ohjelmistosta. Tarvittaessa ohjelman varsinaiset käyttäjät voivat myös määritellä ohjelmistolle vielä uusia tarinakortteja, jotka ylläpitäjät totuttavat muiden tukitoimintojen osana. Kun uusia ominaisuuksia ei enää löydetä, voidaan siirtyä loppujulkaisu vaiheeseen. Tässä vaiheessa ohjelmalle kirjoitetaan vapaamuotoinen dokumentaatio. Lisäksi se todetaan sekä tilaajan että toteuttajan puolelta valmiiksi, ja projektin alussa sille asetettu vaatimukset täyttäväksi [Abrahamsson *et al,* 2002; Beck and Andres, 2006].

XP-menetelmän keskeisin ominaisuus on yhteisöllisyys. Menetelmän 12 muuta periaatetta ja toimintaa ohjaavaa käsitettä ovat:

1. Pienet/nopeat julkaisut periaate tarkoittaa, että toteutettavasta ohjelmistosta julkaistaan mahdollisimman nopeassa tahdissa uusia versioita [Abrahamsson *et al,* 2002; Beck and Andres, 2006; Extreme Programming, 2002].

2. Yhteisomistus periaate tarkoittaa, että koko ohjelmiston lähdekoodi on kaikkien toteuttajien ja muiden tiimiläisten muutettavissa ilman mitään rajoituksia. Tämä mahdollistaa huomattavan nopean ohjelmistonkehittämisen, koska kriittisiäkään muutoksia ei tarvitse hyväksyttää projektin johtajalla. Samalla myös kaikki tuntevat koko

ohjelmiston rakenteen ja henkilöiden puuttuminen projektin käytöstä ei hidasta tai estä toteutuksen jatkamista.[Abrahamsson *et al,* 2002; Beck and Andres, 2006; Extreme Programming, 2002].

3. Asiakkaan läsnäolo periaate tarkoittaa, että projektin asiakaan eli tilaajan edustajan on oltava osana projektia koko sen elinkaaren ajan. XP-menetelmissä tilaajan edustajan on myös oltava sellainen henkilö, jolla on valta sekä tietotaito tehdä projekti koskevia päätöksiä itsenäisesti ja nopeassa tahdissa. Tällöin hänelle on aidosti mahdollisuus vaikuttaa projektin lopputulokseen [Abrahamsson *et al,* 2002; Beck and Andres, 2006; Extreme Programming, 2002].

4. Yhtenäiset ohjelmointikäytännöt periaate tarkoittaa, että kaikki toteuttajat käyttävät aina tiettyjä standardoituja ohjelmointitapoja. Tällöin koodin luettavuus paranee ja rakenne pysyy yksinkertaisena. Samalla myös koodin integrointi onnistuu kivuttomammin, rakenteen parantaminen helpottuu ja koodin tehokkuus paranee [Abrahamsson *et al,* 2002; Beck and Andres, 2006; Extreme Programming, 2002].

5. Pariohjelmointi periaate tarkoittaa, että kaksi ohjelmoijaa yhdessä työskentelemällä tuottaa huomattavasti parempaa ohjelmakoodia, kuin kaksi ohjelmoijaa erikseen. Toinen koodaajista istuu vuorollaan varsinaisen koodaajaan paikalla ja toinen varmistaa vieressä, että kaikki tulee tehdyksi mahdollisimman tehokkaalla tavalla, tehtävien vaihtoon asti. Tehtävien vaihto voi tapahtua vapaasti parin sopimalla tavalla. Tästä syystä, XP-menetelmässä koko toteutus tehdään aina pariohjelmointina [Abrahamsson *et al,* 2002; Beck and Andres, 2006; Extreme Programming, 2002].

6. Testaus ensin periaate tarkoittaa, että jokainen iteraatio alkaa aina toteutettavan osan testauksen suunnittelulla, jonka jälkeen voidaan aloittaa vasta varsinainen ohjelmakoodin tekeminen. Tarkoituksena on testata aina kaikki ne osat mitkä voivat joskus mennä rikki. Testien suunnittelemisella ensin halutaan varmistua, että kaikki toiminnallisuudet on toteutettu ja näiden osat testattu. Samalla myös vältytään pitkiltä ja raskaita manuaalisilta kokonaisuuksien testauksilta lähes kokonaan [Abrahamsson *et al,* 2002; Beck and Andres, 2006; Extreme Programming, 2002].

7. Rakenteen parantaminen periaate tarkoittaa, että ohjelmiston lähdekoodin rakenteeseen tehdään mahdollisuuksien mukaan toimintaa parantavia muutoksia ilman, että ohjelman ulkoinen toiminta käyttäjille ei muutu. Tämä voi tarkoittaa esimerkiksi jonkin ohjelmankohdan, mm. funktioiden tai olioiden, uudelleen kirjoittamista ja johtaa selkeämpään ohjelmistoarkkitehtuuriin [Abrahamsson *et al,* 2002; Beck and Andres, 2006; Extreme Programming, 2002].

8. Yksinkertainen rakenne periaate tarkoittaa, että toteutettavan ohjelmiston perusarkkitehtuuriksi on aina valittava kaikkien kevyin ja parhaiten sopivin vaihtoehto juuri tällä hetkellä. Tulevaisuuden mahdollisia kehitys tai laajennustarpeita ei pidä miettiä valintaa tehdessä. Arkkitehtuuriin rakenne päivittyy kuitenkin automaattisesti, sitten kun sille on tarvetta [Abrahamsson *et al,* 2002; Beck and Andres, 2006; Extreme Programming, 2002].

9. Jatkuva integraatio periaate tarkoittaa, että kaikkien toteuttajien tulee liittää tekemänsä koodinosat täydellisen lähdekoodin osaksi mahdollisimman usein. Tämän tulisi tapahtua heti kun toteuttaja on varmistunut koodinsa toimivuudesta. Tavoitteena on välttää tilanteita, joissa kaksi tai useampi toteuttaja tekee tai muuttaa samaan aikaan samaan ohjelmiston osaa muiden kanssa [Abrahamsson *et al,* 2002; Beck and Andres, 2006; Extreme Programming, 2002].

10. 40-tuntinen työviikko ja tasainen työtahti periaate tarkoittaa, että ylitöiden tekemistä useaa peräkkäisenä päivänä tai viikkona ei sallita. Samalla myös pitkiä kokouksia tai palavereja pyritään välttämään, koska niiden tuottavuus luovuutta vaativissa tehtävissä on yleensä heikko. Tämän tarkoituksena on varmistaa tiimiläisille myös täyteläinen ja merkityksellinen elämä organisaation ulkopuolella. [Abrahamsson *et al,* 2002; Beck and Andres, 2006, Extreme Programming, 2002].

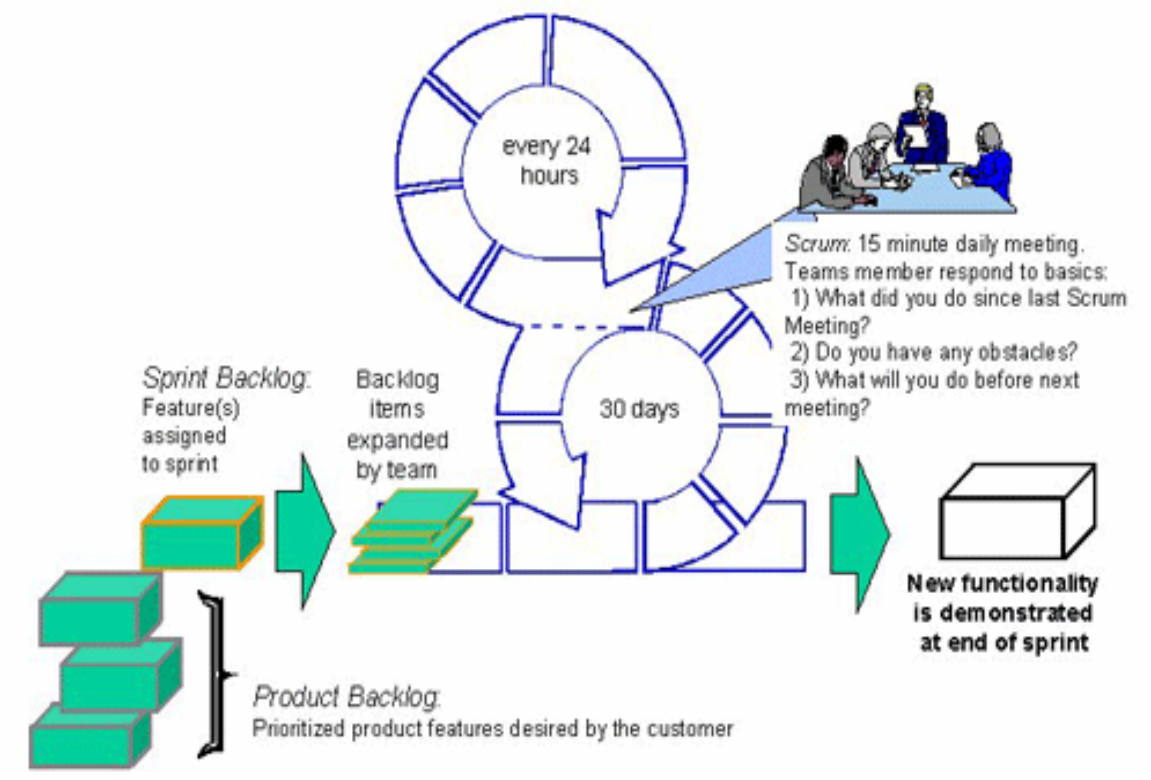
11. Vertauskuvat periaate tarkoittaa, että vaikeiden, haastavien tai abstraktien asioiden täydelliseen kuvaamiseen ei käytetä turhaan aikaan. Tämän sijaan kuvataan, millainen se tai sitä millainen tämä jokin on. [Abrahamsson *et al,* 2002; Beck and Andres, 2006; Extreme Programming, 2002].

12. Suunnittelu periaate tarkoittaa, että projektin etenemistä on etukäteen suunniteltava, mutta samalla pitää varautua siihen, että nämä suunnitelmat muuttuvat jatkuvasti. Tällöin ei ole järkevää suunnitella kaikkea pikkutarkasti. Yleisen tason suunnitelmia voidaan tarvittaessa helposti tarkentaa projektin edetessä, jos näille tulee tarvetta. Avoimet suunnitelmat myös mahdollistavat toteuttajille luovemmat ratkaisumahdollisuudet eri tilanteisiin [Abrahamsson *et al*, 2002; Beck and Andres, 2006; Extreme Programming, 2002].

Extreme Programming menetelmässä näiden toimintamallien ja käytäntöjen kaikkein tärkein tehtävä on vähentää erilaisia muutoksesta aiheutuvaa epävarmuutta sekä erityisesti taloudellisia kuluja. Vaikka ohjelman vaatimukset yleensä muuttuvatkin projektin edetessä, niin kulujen tulisi tästä huolimatta pysyä aina samana. XP-menetelmässä tämän katsotaan onnistuvan poikkeuksellisen hyvin. Tästä johtuen, XP-menetelmää pidetään erityisen joustavana ja muutoksen hyväksyvänä ohjelmistoprojektin toteutusmenetelmänä.

5.4. Scrum

Scrum on toinen erittäin tunnettu ja nykyään paljon käytetty ketterä ohjelmistokehitysmenetelmä. Menetelmän nimi on lainattu rugby-pelin aloitusryhmityksestä, jossa pallo pelataan takaisin kentälle. Scrum-menetelmä perustuu ajatukselle, jonka mukaan projekteihin aina tiettyjä toimenpiteitä joihin kuuluu tietty määrä epävarmuutta tai muutosta. Scrum-menetelmä pyrkii ratkaisemaan tämän mukanaan tuomat ongelmat tarjoamalla projektille samaan aikaan sekä toiminnan viitekehyksen että tukiverkoston, mutta vain yleisellä tasolla. Toiminnan tavoitteena on yhdessä tilaajan kanssa keskittyä vain olennaiseen ja välttää kaikkea resurssien tuhlausta. Tähän tavoitteeseen pyritään itsenäisillä ja mukautumiskykyisillä 4-10 henkilön tiimeillä. Scrum-menetelmän eteneminen ja työvaiheet on esitetty kuvassa 19 [Schwaber, 2000; 2004; Scrum, 2009; Sutherland, 2001].



Kuva 19: Scrum menetelmän eteneminen ja työvaiheet [Scrum, 2009]

Scrum-menetelmässä ohjelmistokehitys tapahtuu kolmen vaiheen kautta. Ensimmäiseen vaiheeseen kuluu arkkitehtuurin ja kaikkien muiden ominaisuuksien sekä toiminnan toteutuksen alustava suunnittelu yleisellä tasolla. Suunnittelun lopuksi nämä kootaan yhteen, niin sanotuksi työlistaksi (Product Backlog), joka vastaa hyvin pitkälle perinteisestä ohjelmistokehityksestä tuttua vaatimusmäärittelyä tai karkeaa projektisuunnitelmaa. Työlistaa myös päivitetään kokoajan projektin edetessä, aina silloin kun uutta tietoa on saatavilla esimerkiksi aikataulusta, riskeistä ja työvälineistä. Tämä jälkeen menetelmä etenee toiseen eli toteutusvaiheeseen [Pressman, 2010; Schwaber, 2000; 2004].

Toteutusvaiheessa toiminta jatkuu pyrähdyksiksi kutsutuissa iteraatioissa, joissa kulloinkin toteutettava kokonaisuus (Sprint Backlog) on jaettu toiminnan tärkeyden mukaan erikokoisiksi osiksi eli inkrementeiksi. Pääsääntöisesti yhden pyrähdyksen kesto on 30 päivää ja siihen kuulu vaatimusten tarkennus, toteutuksen suunnittelu, toteutus, testaus ja valmiin osan julkaisu sekä mahdollisesti toiminnan esittely tilaajalle. Tätä jatketaan niin kauan, kunnes kaikki työlistan asiat saadaan käsitellyksi. Projektiin kuuluu tavallisesti 3–8 pyrähdystä [Pressman, 2010; Schwaber, 2004].

Projektin toiminnasta ja etenemisestä vastuussa olevat tiimit kokoontuvat myös päivittäin projektipäällikön johtamiin lyhyisiin 15 minuutin tapaamisiin, joissa pohditaan ja

kartoitetaan yhteistyössä muiden projektiin kuuluvien tiimien kanssa toimintakokonaisuuden etenemistä. Jokainen toteuttaja joutuu myös vuorollaan vastaamaan kolmeen kysymykseen. Tämän tarkoituksena on varmistaa, että jokainen toteuttajista on tehnyt vain sovittuja asioita ja saanut ne valmiiksi. Ilmoittaa mahdollista ongelmista tai toiminnan jatkamisen esteistä muille sekä kertoa oman toiminnan jatkosta tai niistä inkrementtiin kuuluvista töistä, jotka on tarkoitus saada valmiiksi seuraavan tapaamiseen mennessä [Pressman, 2010; Schwaber, 2004].

Kolmanteen ja samalla viimeiseen jälkivaiheeseen kuuluu kaikkien osien integrointi toisiinsa sekä kokonaisuuden testaus. Jos tässä ei ilmene ongelmia ja uusille pyrähdyksille ei ole ilmennyt tarvetta projektin toteutuksen aikana, niin projekti viimeistellään toteutuksen dokumentoinnilla. Jos kuitenkin tarvetta uusille ominaisuuksille on ilmennyt tai jonkin pyrähdyksen kaikkia vaadittuja asioita ei ole saatu toteutettua täydellisesti jonkin inkrementin kohdalla, niin ne viimeistellään tässä vaiheessa. Vasta tämän jälkeen, projektin valmis lopputuote toimitetaan tilaajalle [Pressman, 2010; Schwaber, 2000; 2004].

Scrum-menetelmää käyttävissä projekteissa kaikki toiminta on valjastettu tukemaan pyrähdyksen onnistumista ja sille asetettujen työtavoitteiden täyttymistä. Myös tiimien itseohjautuvuus ja yksilön toiminnan vapaus tukee tätä. Tästä syystä scrum-menetelmän oletettiin soveltuvan hyvin vain lokaaleihin sekä pieniin tai keskisuuriin projekteihin [Sutherland, 2001]. Nykyään scrum-menetelmää kuitenkin käytetään myös suurissa ja globaaleissa projekteissa. Scrum-menetelmää käyttävä projekti vaatii kuitenkin onnistuakseen sitä käyttävältä organisaatiolta sen täydellistä käyttöönottoa kaikkine siihen kuuluvine osineen ja työvaiheineen. Mikäli joitain osia jätetään pois tai toimintaa yritetään selkeästi esimerkiksi hallinnoida, johtaa tai kontrolloida, niin tällöin scrum-menetelmä ei voi toimia vapaasti ja sen hyödyt yleensä menetetään [Pressman, 2010; Schwaber, 2000; 2004; Scrum, 2009].

5.5. Muut menetelmät lyhyesti

Suurimpien ja tunnetuimpien ketterien menetelmien lisäksi, on myös olemassa suuri joukko muita pienempiä ja samalla vähemmän tunnettuja ketteriä menetelmiä, malleja sekä erilaisten teorioiden tai menetelmien sovelluksia. Osa näistä on lokeroitu kuuluvaksi ketterien ohjelmistokehitysmenetelmien alle, lähes pelkästään niiden myöhäisen synty- ja esittelemisajankohdan vuoksi.

Olen tässä tutkimuksessa valinnut mukaan vertailuun näistä menetelmistä, vain omasta mielestäni sekä tutkielman näkökulman kannalta, oleellisimpia menetelmiä. Kaikki seuraavien kappaleiden menetelmät pohjautuvat tai saavat ideansa suoraan ketterästä

ohjelmistokehityksestä, ja lisäksi niiden katsotaan yleisesti todella kuuluvan ketterien menetelmien joukkoon. Samalla niistä on myös edes jossain määrin saatavilla erilaista kirjallista aineistoa.

5.5.1. Dynamic Systems Development Model (DSDM)

Dynamic Systems Development Model (DSDM), menetelmä on vuonna 1994 kehitetty organisaatio ja liiketoiminta keskeinen ketterä ohjelmistokehitysmenetelmä. Menetelmää on jatkokehitetty ja tarkennu jatkuvasti ja sen nykyinen versio on 4.2. DSDM-menetelmän toimintaa on ehkä XP- ja Scrum-menetelmien jälkeen parhaiten dokumentoitu kaikista ketteristä menetelmistä.

DSDM-menetelmän tavoitteena on vastata liiketoiminnan haasteista ja tarpeista nouseviin vaatimuksiin mahdollisimman nopeasti. Tähän pyritään kiinnittämällä muuttamattomiksi osa projektin toimintaympäristöstä, yleensä ne ovat vaatimukset, perinteisten ohjelmistokehityksen menetelmien tapaan. Samalla ollaan kuitenkin tarvittaessa valmiita joustamaan muista osista, pääsääntöisesti aikataulusta ja budjettista. Menetelmä toimiikin lähinnä projektin toimintaa ohjaavan tekijänä sen viitekehyksenä. DSDM-menetelmän vaiheet on esitetty kuvassa 20 [DSDM, 2002; Stapleton, 1997; 2003].

Kuva 20: DSDM-menetelmän vaiheet [Stapleton, 1997 s.3; 2003, s.4, s.171]

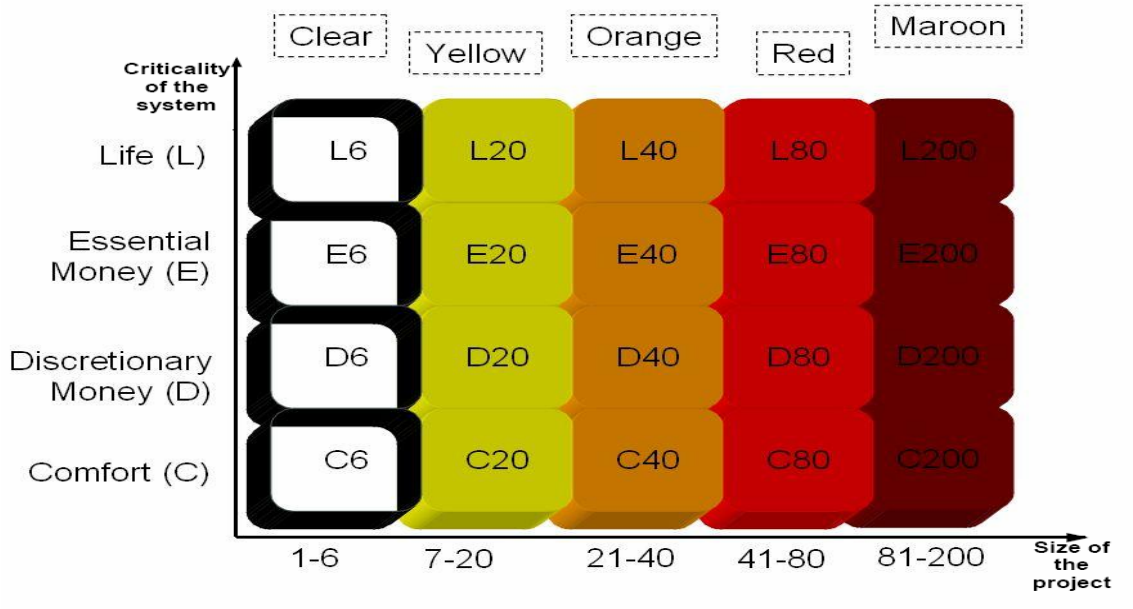
Kuvasta 20 voidaan nähdä, että DSDM-menetelmässä toiminta etenee viiden vaiheen kautta. Näistä kaksi ensimmäistä ovat toteutettavuuden määrittely ja liiketoiminnan tutkimus, jotka tehdään vain kerran yhden projektin aikana. Näiden vaiheiden lopputuloksena kootaan esitutkimus ja projektisuunnitelma dokumentit sekä aina arkkitehtuuritasolle asti ulottuva vaatimusmäärittely ja siihen liittyvät prototyyppisuunnitelmat. Tätä seuraavaa kolmea vaihetta tehdään niin kauan, kunnes projekti täyttää sille alussa asetettu vaatimukset [DSDM, 2002; Stapleton, 2003].

Nämä ovat toimintojen määrittely, jossa tarkennetaan aikaisemmin muodostettuja vaatimuksia sekä rakennetaan näiden pohjalta prototyypit sekä näihin liittyvät riskianalyysit ja katselmointiraportit. Suunnittelu- ja toteutusvaiheessa toteutetaan varsinainen ohjelmisto edellisen vaiheen prototyyppien pohjalta, jonka jälkeen se esitellään tilaajalle mahdollisten muutosehdotusten tai muiden kommenttien saamiseksi. Käyttöönotto vaiheessa valmis ohjelmisto luovutetaan tilaajan käyttöön ja tarvittaessa sitä käyttäville henkilöille järjestetään koulutusta sen käyttämiseen. Samalla myös viimeistään kaikki aikaisempiin projektin vaiheisiin kuuluvat dokumentit sekä kootaan täydellinen loppuraportti [DSDM, 2002; Stapleton, 2003]

DSDM-menetelmä korostaa ryhmätyön merkitystä projektin onnistumisessa, antamatta kuitenkaan ohjeita siihen miten varsinainen työ tulisi jakaa tai suorittaa. Myös merkittävästi suurempi dokumentaation määrä ja toiminnan varsin yksityiskohtainen suunnittelu projektin alussa, useimpiin muihin ketteriin menetelmiin nähden, erottaakin sen selkeästi muista menetelmistä.

5.5.2. Crystal Method

Crystal Method on oikeastaan erilaisten menetelmien kokoelma, joka esiteltiin ensimmäisen kerran jo vuonna 1992 Alistair Cockburnin toimesta. Menetelmistä voidaan aina valita käytettäväksi kulloiseenkin projektiin parhaiten sopiva kokonaisuus sen tarpeista riippuen. Käytännössä valintaan vaikuttavat ensiksi projektiryhmän koko ja projektin kriittisyys sekä näiden eri ulottuvuudet. Kaikille Crystal Method-menetelmille on tyypillistä ihmiskeskeisyys ja toiminnan lyhyt inkrementaalinen kehityssykli. Crystal Method-menetelmät ja niiden ulottuvuudet on esitetty kuvassa 21 [Cockburn, 2007; Crystal, 2009].



Kuva 21: Crystal Methods-menetelmät [Cockburn, 2007, s.338]

Crystal Method-menetelmissä projektin kokoa arvioidaan siinä mukana olevien henkilöiden lukumäärän perusteella. Henkilömäärä määrää käytettävän menetelmän värin. Henkilöiden lukumäärä voi vaihdella 1–200 välillä siten, että 1-6 henkilöä (Kirkas), 7–20 henkilöä (keltainen), 21–40 henkilöä (oranssi), 41–80 henkilöä (punainen) ja 81–200 henkilöä (viininpunainen) [Cockburn, 2007].

Crystal Method-menetelmissä projektin kriittisyyttä arvioidaan 4 ulottuvuuden avulla, jotka ovat mukavuus (C), kohtuullinen raha (D), kriittinen raha (E) ja elämä (L). Kriittisyys määräytyy virhetilanteen seurausten perusteella. Mukavuus tarkoittaa tilannetta, jossa ohjelmistovirheestä johtuen työt joudutaan tekemän käsin, jolloin saman toiminnan suorittamiseen kuluu enemmän aikaan. Kohtuullinen raha tarkoittaa tilannetta, jossa ohjelmistovirheestä aiheutuvat rahalliset vahingot voidaan myöhemmin saada takaisin. Kriittinen raha tarkoittaa tilannetta, jossa ohjelmistovirheen rahalliset seuraukset ovat jollekin mukana olevalle taholle erittäin merkittäviä. Elämä tarkoittaa tilannetta, jossa ohjelmistovirheen seurauksena syntyvät tilanteet saattavat vaarantaa tai johtaa ihmishenkien menetykseen [Cockburn, 2007].

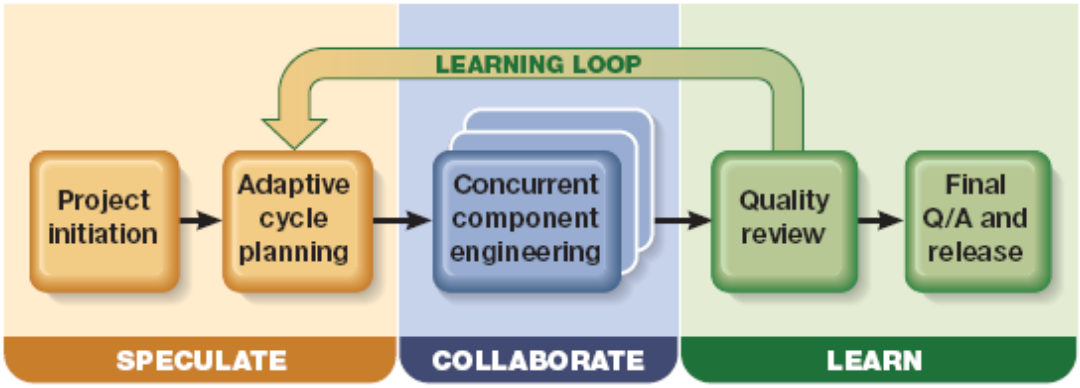
Käytettävän Crystal Method-kehitysmenetelmän valinta tapahtuu näiden kahden ulottuvuuden yhdistelmällä. Tällöin esimerkiksi E80 tarkoittaa projektia, jossa ohjelmistoa on kehittämässä 41–80 henkilöä ja sen virheellinen toiminta aiheuttaa merkittäviä rahallisia menetyksiä. Crystal Method-menetelmien kokoelma ei ole vielä kaikilta osiltaan täysin valmis. Toimintaohjeet eri tilanteille on julkaistu vasta kirkkaille ja oransseille projekteille.

Julkaistuissa Crystal Method-menetelmissä toiminta on iteratiivista ja se on jaettu viiteen vaiheeseen. Vaiheet ovat järjestäminen, palautteen keruu, iteraatiokierros, katselmointi ja inkrementin viimeistely. Järjestämisvaiheessa valitaan toteutettavaan iteraatioon kuuluvat toiminnot sekä arvioidaan niiden toteuttamiseen kuluva aika. Jokaista toteutettavaa inkrementtiä kohden tehdään joko yksi tai useampia iteraatiokierroksia. Toiminnan etenemistä ja ohjelmiston valmistumista mitataan sekä merkkipaalujen että myös ohjelmisto vakauden avulla. Kun riittävä taso on saavutettu, toteutus katselmoidaan yhdessä tilaaja kanssa palautteen saamiseksi. Mikäli tilaaja on tyytyväinen näkemäänsä ja muutakaan korjattavaa ei ole, niin inkrementti voidaan viimeistellä ja siirtyä seuraavaan inkrementin järjestämiseen. Tätä jatketaan kunnes kaikki ohjelmiston toiminnot on toteutettu [Cockburn, 2007; Crystal, 2009].

Crystal Method-menetelmissä keskitytään pääasiassa projektiryhmän henkilöiden rooleihin ja ryhmien välisen kommunikaation määrittelemiseen. Menetelmä ei tarjoa juuri mitään konkreettisia ohjeita siitä miten varsinainen ohjelmistokehitys tulisi projektissa suorittaa, jotta päästäisiin parhaaseen mahdolliseen lopputulokseen. Lähes kaikki asiat projektin toimintatapojen valinnasta alkaen aina käytännön asioiden hoitamiseen asti, jäävätkin projektissa mukanaolijoiden itsensä ratkaistaviksi [Cockburn, 2007; Crystal, 2009].

5.5.3. Adaptive Software Development (ASD)

Adaptive Software Development (ASD) menetelmä on Jim Highsmithin [2000] 2000-luvun alussa esittelemä ketterä ohjelmistokehitysmenetelmä, joka pohjautuu sekä vesiputousmalliin että Radical Software Development (RAD) menetelmään. Alussa ASD-menetelmän lähtökohtana oli korjata ja välttää edellisiin menetelmiin liittyviä ongelmia. ASD-menetelmän taustalla on ajatus, jonka mukaan ohjelmistoprojektien toimintaympäristö on aina ennalta arvaamaton ja jatkuvasti muuttuva. Tästä syystä menetelmässä toiminta tapahtuu aina inkrementaalisesti ja iteratiivisesti, johon liittyy myös jatkuva prototyyppien käyttö. Lisäksi menetelmässä luotetaan hyvin paljon henkilöiden toiminnan itseohjautuvuuteen ja yhteistyöhön. Toiminnan hyödyt tulevatkin parhaiten esiin monimutkaisissa ohjelmisto tai tietojärjestelmä hankkeissa. ASD-menetelmän toiminnan päävaiheet ovat spekulointi, yhteistyö ja oppiminen. Menetelmän eteneminen näiden vaiheiden kautta on esitetty kuvassa 22 [Highsmith, 2000; Pressman, 2010].



Kuva 22: ASD-menetelmän vaiheet ja toiminnan eteneminen [Highsmith, 2000, s.26]

ASD-menetelmän spekulointi vaiheeseen kuluu projektin aloitus sekä siihen liittyvä mission muodostaminen. Missio pitää sisällään projektin yleisen tavoitteen lisäksi myös projektin vision sekä lähes kaiken muun projektin toteuttamiseen tarvittavan tiedon keräämisen. Kaiken alustavan suunnittelun pohjana käytetään projektin tilaajalta saatavaa tietoja aikataulusta, tavoitteista ja vaatimuksista. Spekulointivaiheen lopuksi projektille määrätään siinä julkaistavien prototyyppien lukumäärä ja niiden julkaisupäivät sekä projektin päättymispäivä. Tämän jälkeen nämä ovat projektissa kiinteitä ja niitä ei voi enää tämän jälkeen muuttaa. Jos jokin päivämäärä kuitenkin uhkaa ylittyä, on toteuttajien karsittava kyseisen toteutuksen osan ominaisuuksia, jotta alkuperäinen päivämäärä saavutetaan ajallaan [Highsmith, 2000; Pressman, 2010].

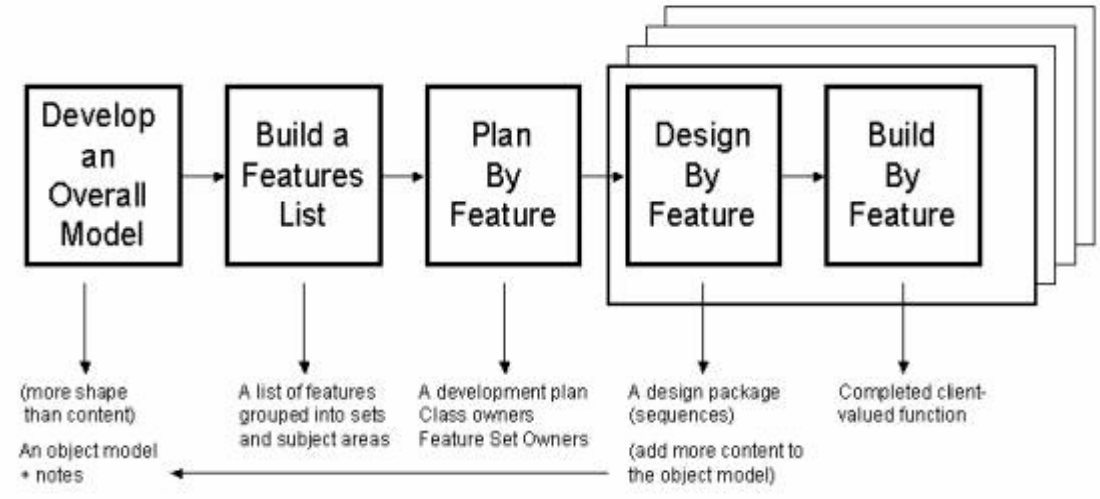
Yhteistyövaiheessa tehdään projektin varsinainen toteutustyö. Iteraation yleiset vaatimukset ja rajoitteet annetaan kehitystiimien itsensä ratkaistavaksi ilman tarkkoja ohjeita. Projektin toteuttajista muodostetuilla tiimeillä onkin siis hyvin paljon vapauksia sekä toteutuksen tekniikoiden että toimintatapojen suhteen. Tämä tarkoittaa myös, että toiminnan itseohjautuvuuteen luotetaan hyvin paljon. Kehitysryhmien sisäinen ja ryhmien välinen kommunikaatio onkin tärkeässä osassa projektin kokonaisuuden onnistumisen kannalta. Kaiken yhteistyön onnistumista voidaan pitää yhtenä ASD-menetelmien kriittisenä menestystekijänä [Highsmith, 2000; Pressman, 2010].

Oppimisvaiheeseen siirrytään aina siinä vaiheessa kun yksi iteraatio saadaan valmiiksi. Vaiheen tavoitteena on tarkastella ja arvioida toteutuksen onnistumista eri näkökulmista. Nämä ovat tilaajannäkökulma, tekninen näkökulma, kehitysryhmän toiminnan näkökulma sekä projektin sisäinen näkökulma. Tämän tarkoituksena on varmistaa aina kulloisenkin iteraation kaikkien laadullisten tavoitteiden täyttyminen. Tämän jälkeen siirrytään aina uudestaan uuden iteraation toteutuksen alkuun, kunnes kaikki iteraatiot on saatu valmiiksi tai projektin alussa määrättynä päättymispäivänä on saavutettu. Projekti päättyy kuitenkin joka tapauksessa tuotteen luovutukseen tilaajalle sovittuna

päättymispäivänä, vaikka sen kaikkia ominaisuuksia ei olisikaan saatu valmiiksi täydellisinä [Highsmith, 2000; Pressman, 2010].

5.5.4. Feature Driven Development (FDD)

Feature Driven Development (FDD), menetelmä esiteltiin ensimmäisen kerran vuonna 1999. Menetelmää oli kuitenkin jo ennen tätä ehditty soveltaa useissa erikokoisissa projekteissa, joissa se yleensä kuitenkin liitettiin osaksi jotakin toista ohjelmistokehitysmenetelmää. Tästä syystä FDD-menetelmää onkin alusta asti pidetty erittäin monipuolisena ja hyvin erilaisten projektin muuttuviin tarpeisiin mukautuvana menetelmänä. FDD-menetelmä ei ole samalla tavoin kokonaisvaltainen, ja tarjoa täydellistä projektinhallintamallia, useiden muiden perinteisten tai ketterien ohjelmistokehitysmenetelmien tapaan. Sen sijaan, FDD-menetelmässä toiminta on keskittynyt hyvin pitkälti projektin tekniseen toteutuksen ja testauksen ympärille. FDD-menetelmä koostuu viidestä vaiheesta, jotka on esitetty kuvassa 23 [Coad *et al*, 1999; Palmer, 2002].



Kuva 23. FDD-menetelmän vaiheet ja toiminnan eteneminen [Palmer, 2002; Coad *et al*, 1999, s.199]

FDD-menetelmässä toiminta lähtee liikkeelle yleisen toimintamallin kehittämisellä, johon voidaan käyttää lähes mitä tahansa ohjelmiston mallinnustekniikkaa, kuten luokka- ja sekvenssikaavioilla. Tämän yleisluontoisen suunnitelman pohjalta kootaan myös toisen vaiheen tuotteen toiminnallisten ominaisuuksien lista, jossa projektin vaatimukset ryhmitellään yhdessä projektin tilaajan kanssa pienemmiksi ja tarkemmiksi osajoukoiksi. Tämä tarkoituksena on varmistaa, että kehitettävä ohjelmisto täyttää kaikki tilaajan vaatimukset oikealla tavalla. Kun tämä on saatu valmiiksi, niin voidaan ryhtyä ohjelmiston kehityksen suunnitteluun, jossa ohjelmistovaatimukset annetaan projektin toteuttajille. Toteuttajien tehtävänä on tämän jälkeen kehittää ja ylläpitää aikaisempaa

projektimallia sekä suunnitella varsinaisen ohjelmiston toteutuksen eri osien aikataulut [Coad *et al*, 1999; Palmer, 2002].

FDD-menetelmän kaksi viimeistä vaihetta toteutetaan iteratiivisesti. Toimintojen suunnittelu- ja toteutusvaiheissa valitaan ensin kussakin iteraatiossa toteutettavat toiminnallisuudet perinteisiä suunnittelu metodeita käyttäen. Tämän jälkeen valitaan niiden toteuttamisesta vastuussa olevat henkilöt tai ryhmät, jotka aloittavat osien toteutuksen. Toteutusvaiheeseen liittyy aina myös testaus ja katselmointi sekä valmistuneen inkrementin integrointi muihin projektin osiin. Samaan aikaan voi myös olla käynnissä useiden projektin osien toteutus. Vaiheita toistetaan niin kauan, kunnes kaikki projektin toiminnallisten ominaisuuksien listan asiat on saatu toteutettua, ja haluttu lopputulos on saavutettu. Tämän jälkeen projekti viimeistellään ja päätetään [Coad *et al*, 1999; Palmer, 2002].

5.6. Riskienhallinta ketterissä menetelmissä

Ketteriä menetelmiä käyttävissä projekteissa riskienhallinta perustuu siihen, että erilaisiin muutoksiin pystytään reagoimaan entistä nopeammin, koska ne havaintaan erittäin aikaisessa vaiheessa. Lisäksi turhan hierarkian puuttuminen alentaa myös merkittävästi aikaa joka tarvitaan päätösten tai suorien vastatoimien tekemiseen. Myös ketterille menetelmille tyypillinen inkrementaalinen ja/tai iteratiivinen kehitysmalli tarjoaa erinomaiset mahdollisuudet esimerkiksi riskianalyysille aina jokaisen uuden inkrementin/iteraation alussa. Mutta ennen kaikkea, menetelmät tarjoavat tilaisuuden oppia tästä prosessista jotakin aina seuraavan osan alkuun mennessä, sekä tekemään riskienhallintaan tarvittavat korjaukset ja muutokset, jotta ei toistettaisi samoja virheitä, jotka mahdollisesti tehtiin aikaisemmin [Chin, 2004; Cockburn, 2007].

Ketterän riskienhallinnan tavoitteena, on ymmärtää projekti entistä paremmin dynaamisena prosessina, jossa mukana olijat etsivät jatkuvasti vaihtoehtoisia toimintatapoja ja -menetelmiä siihen, miten tarkoituksenmukainen lopputulos olisi mahdollista saavuttaa entistä paremmin ja tehokkaammin. Tämä koskee myös käytännön riskienhallintaa. Koska ketterät menetelmät mahdollistavat nopean reagoimisen erilasiin asioihin ja tapahtumiin, niin tästä syystä ketterässä riskienhallinnassa ei tarvitse aina tehdä vain tavanomaisia ratkaisuja. Erilaisten kokeellisten tai muulla tavoin vaihtoehtoisten riskienhallintatapojen käyttäminen on myös mahdollista ilman merkittävää vahinkoa. Jopa suoranaisten virheiden tekeminen ei ole katastrofaalista, koska niiden mahdollisiin vaikutuksiin ollaan käytännössä kokoajan valmiina. Voidaankin sanoa, että ketterät menetelmät ja niiden riskienhallinta menetelmät mahdollista tai jopa kannustavat oppimista sekä henkilö että organisaatio tasolla [Boehm, 2002, Chin, 2004; Nelson et al, 2008].

Ketterien menetelmien luonteeseen sekä muuhun yleiseen rakenteeseen onkin siis sisällytetty suoraan hyvin paljon riskienhallintaa ja sen eri elementtejä. Samalla riskienhallinnasta on myös tehty kiinteä osa jokapäiväistä työtä, jossa se on aina kaikkien osatekijöidensä summa [Chin, 2004]. Riskienhallinta ei synny pelkästään dynaamisesta riskien tunnistamisesta, analysoinnista ja niiden jatkuvasta hallitsemisesta sekä muista toimista, muun projektitoiminnan tai päätöksen teon tueksi. Prosessi on esitelty tarkemmin luvussa 3. Sen sijaan, ketterässä riskienhallinnassa on kysymys tämän toiminnan täydellisestä sovittamisesta osaksi valittua kehitysmallia sekä siinä kehitettävää sovellusta tai tuotetta [Cockburn, 2007; Schwaber, 2004;].

Hyvin usein ketterissä menetelmissä riskienhallintaa ei siis ajatella pelkästään vain muutoksen ja sen ikävien vaikutusten valossa, vaan se nähdään myös toiminnan ohjaajana sekä joskus myös tarpeellisena muutosvoimana oikeaan lopputulokseen pääsemiseksi. Täydellisesti toimiakseen tämä ajattelutapa kuitenkin edellyttää, että projektien riskienhallinta integroidaan entistä enemmän osaksi myös muita organisaation johtamis- ja päätöksentekojärjestelmiä [Boehm, 2002; Sutherland, 2001; Nelson et al, 2008].

5.7. Ketterien menetelmien yhteenveto

Ketterät menetelmät ovat vielä varsin uusi ilmiö kaikessa projektitoiminnassa ja ohjelmistotuotannossa, kun niitä verrataan koko toimialan historiaan. Menetelmien todellista potentiaalia ja toiminnan tehokkuutta onkin hyvin vaikea arvioida objektiivisesti. Kuitenkin ketterien menetelmien tekniikoita käytettään tai sovelletaan jatkuvasti erilaissa projektiympäristöissä ympäri maailmaa. Taulukossa 1 on koottu yhteen ja vertailtu aikaisemmissa kappaleissa esiteltyjä ketteriä menetelmiä ja niiden keskeisiä ominaisuuksia.

	XP	Scrum	DSDM	Crystal	ASD	FDD
Suunnitteluvaiheen laajuus ja tarkoitus	kevyt, mahdollisimman lyhyt, osin määritelty	lyhytkestoinen, vain tärkeimmät osat	jaksottainen, tarkoin määritelty, kattava ja pitkäkestoinen	heikosti määritelty	jaksottainen, tarkasti määritelty, kattava ja pitkäkestoinen	jaksottainen tarkasti määritelty kattava ja pitkäkestoinen
Iteratiivisuus ja inkrementaalisuus	iteratiivinen toteutusvaihe	iteratiivinen ja inkrementaalinen toteutusvaihe	iteratiivinen toteutusvaihe	inkrementaalinen ja iteratiivinen toteutusvaihe	iteratiivinen toteutusvaihe	inkrementaaliset ja iteratiiviset toimintojen suunnittelu- ja toteutusvaiheet
Iteraation kesto	2–4 viikkoa	30 päivää	ei määritelty	1-4 kuukautta	4–10 viikkoa	2 viikkoa
Tilaajan osallistuminen	päivittäin aktiivisesti tiimin jäsenenä	suunnittelu, iteraatioiden rakenne, lopputulokset	suunnittelu, iteraatioiden tuotosten katselmointi	ei määritelty	suunnitteluun ja katselmointeihin	suunnitteluun
Tilaajan osaaminen	korkea	perustiedot	perustiedot	ei määritelty	perustiedot	perustiedot
Prosessin rakenne	vaiheet määritelty yleisellä tasolla, prototyyppien rakenne ja työskentelytavat määritelty yksityiskohtaisesti	vaiheet tarkasti määritelty, työskentelytavat vapaat	vaiheet ja prototyyppien rakenne yksityiskohtaisesti määritelty. työskentelytavat vapaat	ei täysin määritelty	vaiheet määritelty yleisellä tasolla, vapaat toteuttavat	vaiheet ja prototyyppien rakenne yksityiskohtaisesti määritelty. työskentelytavat vapaat
Vaiheiden ja prototyyppien lukumäärä	6 vaihetta ja 6 prototyyppiä	3 vaihetta ja 4 prototyyppiä	5 vaihetta ja 7 prototyyppiä	5 vaihetta mutta prototyyppien lukumäärää ei määritelty	5 vaihetta ja 5 prototyyppiä	5 vaihetta ja 5 prototyyppiä
Henkilöstön määrä	pääsääntöisesti alle 10 henkilöä.	ei rajoituksia, mukautuu hyvin erikokoisiin projekteihin	ei rajoituksia, yleisempi suurissa projekteissa	1–200 henkilö	ei rajoituksia, mukautuu hyvin monimutkaisiin projekteihin	ei rajoituksia. suuret projektit yleisiä.

Taulukko 1: Ketterien menetelmien ominaisuuksien vertailu

6. Ketterän ja perinteisen ohjelmistokehityksen vertailu

6.1. Merkittävimmät eroavuudet

Ketterä ohjelmistokehitys eroaa merkittävästi perinteisistä ohjelmistokehitysmenetelmistä monessakin suhteessa. Kuvassa 24 projektien ja riskienhallinnan pioneeri Barry Boehm [2002] esittää oman varsin tunnetun näkemyksensä menetelmien eroista.

	Agile methods	**Plan-driven methods**
Developers	Agile, knowledgeable, collocated, and collaborative	Plan-oriented; adequate skills; access to external knowledge
Customers	Dedicated, knowledgeable, collocated, collaborative, representative, and empowered	Access to knowledgeable, collaborative, representative, and empowered customers
Requirements	Largely emergent; rapid change	Knowable early; largely stable
Architecture	Designed for current requirements	Designed for current and foreseeable requirements
Refactoring	Inexpensive	Expensive
Size	Smaller teams and products	Larger teams and products
Primary objective	Rapid value	High assurance

Kuva 24: Ohjelmistokehitysmenetelmien eroaavuudet [Boehm, 2002, s.66]

Kuvasta 24 voidaan nähdä, että perinteiset ja ketterät ohjelmistokehityksen menetelmät ovat lähes kaikkien tarkastelussa mukana olevien ulottuvuuksien suhteen lähes toistensa vastakohtia. Ja artikkelin julkaisuvuoden, 2002, jälkeen tämä ero on vain selkiytynyt, samalla kun ketterien menetelmien käyttö on lisääntynyt kiihtyvällä vauhdilla. Merkittävimmät muutokset kuvan 12 asioihin, ovat tulleet koko (size) ja arkkitehtuuri (architecture) kohtiin. Nykyisin myös suuria ja laajoja järjestelmiä toteutetaan ketterillä menetelmillä, jolloin hyödynnetään niiden koko potentiaali entistä paremmin. Ja vaikka ohjelmistokehittäminen ydinvoimalaan tai pankkiin onkin täysin eriasia kuin matkapuhelimeen, esimerkiksi jatkokehityksen ja ylläpidettävyyden näkökulmasta, niin

projektin onnistuminen on aina tärkeää [Charette, 1996; Glass, 2001; Lindvall *et al*, 2004; Simons, 2002].

Myös useat muut tutkijat ovat tutkineet ketterien ja perinteisten menetelmien eroavuuksia useista eri näkökulmista. Kuvassa 25 on esitetty yhteenveto yhdestä näistä uudemmista tutkimuksista.

	Traditionaalinen	Ketterä
Keskeiset oletukset	Järjestelmät ovat kokonaan määriteltävissä ja ne voidaan rakentaa huolellisella sekä mittavalla suunnittelulla	Korkealaatuiset ja mukautuvat ohjelmistot voidaan kehittää pienissä tiimeissä käyttämällä toimintaperiaatteina jatkuvaa prosessin kehitystä ja nopean palautteen ja muutoksen päälle rakentuvaa testausta.
Kontrolli	Prosessikeskeinen	Henkilöstökeskeinen
Johtamistyyli	Käskytys ja kontrollointi	Johtajuus ja yhteistyö
Tietämyksenhallinta	Ulkoinen	Sisäinen
Roolijaottelu	Erikoistumista suosiva yksilökeskeisyys	Roolinvaihtoa suosivat itsejärjestäytyvät tiimit
Kommunikaatio	Formaali	Informatiivinen
Asiakkaan rooli	Tärkeä	Kriittinen
Projektisykli	Tehtävien tai aktiviteettien ohjaama	Tuotteen ominaisuuksien ohjaama
Kehitysmalli	Elinkaarimallin mukainen (vesiputous, spiraali tai joku variaatio)	Ketterien metodien mukainen iteratiivinen evoluutio-toimitusmalli
Suositeltava organisaatiorakenne	Mekaaninen, byrokraattisen formaali	Orgaaninen, joustavuutta, sitoutumista ja sosiaalista yhteistyötä suosiva rakenne
Teknologia	Ei rajoituksia	Suosii oliopohjaista teknologiaa

Kuva 25: Menetelmien eroavuudet toisista näkökulmista [Nehur *et al*, 2005, s.75]

Kuvasta 25 voidaan nähdä hieman Boehmin [2002] näkökulmaa laajempi ominaisuuksien vertailu, jonka avulla pystytään vielä entistä paremmin näkemään ketterien ja perinteisten menetelmien eroavuudet sekä syyt siihen miksi uuden menetelmän käyttöönotossa ilmenee ongelmia [Boehm and Turner, 2005, Nehur *et al*, 2005]. Samalla myös voidaan konkreettisesti nähdä se, mistä menetelmien kannattajien ja vastustajien selkeä toisen menetelmän pitäminen toista parempana johtuu. Toiminnan painopistealueet, kehitystavat ja suhtautuminen asioihin on täysin erilaista, vaikka kuitenkin molempia menetelmiä voidaan käyttää samoissa projekteissa [Glass, 2001].

Juuri ketterien menetelmien kehityksen inkrementaalisuuden vuoksi myös niiden käyttäminen perinteisten menetelmien tavoin on mahdollista. Sillä inkrementaalisuus mahdollistaa myös laajojen järjestelmien kehittämisen samalla tavoin kuin perinteissä menetelmissä, ellei jopa paremmin, koska asioita, aikatauluja tai tehtäviä voidaan tarkentaa myös jokaisen inkrementin tai iteraation aikana. Käytännössä aina sitä mukaa, kun tarpeita tälle ilmenee, eikä ainoastaan vain vaiheen jo päätyttyä, kuten perinteissä menetelmissä [Lindvall *et al*, 2004].

Lyhyet kehityssyklit sekä säännöllinen ja tiheä julkaisurytmi johtaa väistämättä myös testivetoiseen ohjelmistokehitykseen, joka taas parantaa työn laatua automaattisesti. Lisäksi jatkuva virheiden korjaaminen ja ohjelma kehittäminen esimerkiksi tilaajalta saadun palautteen tai tilanteiden/tarpeiden mukaan on erittäin tehokasta ja myös järkevää [AgileAlliance, 2001; Cockburn, 2007; Cohen *et al*, 2004].

6.2. Projektin- ja riskihallinnan eroavuudet

Perinteisten ja ketterien ohjelmistokehitysmenetelmien merkittävimmät projektin- sekä riskienhallinnan eroavuudet löytyvät molempien menetelmien peruslähtökohdista ja toiminnan painopistealueista sekä niissä tehtävistä alku oletuksista. Tämä johtaa väistämättä myös siihen, että luonnollisesti myös käytännön projektinhallinta keinot, tavat, tekniikat ja välineet ovat todella erilaisia. Ja vaikka kattavaa empiiristä tutkimusta ketterien menetelmien riskienhallinnan käytöstä ei ole vielä kattavasti saatavilla, niin näiden menetelmien perusta on kuitenkin jonkin verran perinteisiä menetelmiä riskitietoisempi [Boehm, 1991; 2002; Charette, 2001a; 2001b; Payne *et al*, 2005].

Ketterille menetelmille tyypilliset kurinalaiset matalan tason laatu- ja toimintakäytännöt ovat osa niiden sisäistä, ja paikoitellen myös varsin näkymätöntä, projektinhallintaa. Lähes kaikki ketterät menetelmät delegoivat päätösvaltaa organisaatiossa ja/tai projektissa myös alaspäin [AgileAlliance, 2001; Boehm, 2002]. Tämä onkin ehkä ketterien menetelmien suurimpia eroja suhteessa perinteisiin menetelmiin. Kaikki henkilöt projektissa ottavat vastuuta työstään ja seuraavat omien tehtävien etenemisen lisäksi myös koko projektin etenemistä sekä havainnoivat jatkossa töiden lopettamiseen vaadittavaa yksilö- ja kokonaispanosta. Kun taas perinteisissä menetelmissä, tämän jää lähes kokonaan projektipäällikön tai johtoryhmän tehtäväksi [Chin, 2004; Ocampo, 2007; Nelson et al, 2008; Pressman, 2010; Ribeiro *et al*, 2009].

Ketterissä menetelmissä myös projektin sisällä eroavista näkemyksistä toiminnan tai tehtävien suhteen voidaan avoimesti ja kokoajan keskustella sekä tehdä niihin tarvittavia resurssien kohdennuksia tai muita korjauksia, jos tarvetta näille todella on. Perinteisiä menetelmiä käyttävissä projekteissa tämä ei korkeamman käsky ja valta hierarkian

vuoksi ole yleensä mahdollista tai sen toimeenpaneminen kestää kohtuuttoman kauan [Charette, 2001c; 2001d; Hoch *et al*, 2000; Nelson et al, 2008].

Lisäksi ketterät menetelmät avaavat uusia kommunikaatio reittejä ja mahdollisuuksia. Samalla ne kuitenkin myös asettavat projektia johtavat ja ennen kaikkea siitä vastuussa olevat henkilöt tilanteeseen, jossa heidän omat todelliset projektipäällikön kykynsä, erityisesti juuri henkilöjohtamisen osa-alueella todella punnitaan. Lisäksi projektipäällikön oma henkilökohtainen sosiaalinen älykkyys ja kommunikaatiotaidot joutuvat tässä tilanteessa kovaan testiin [Cohen *et al*, 2004; Markus, 2004].

Ja vaikka perinteisten menetelmien kannattajat usein kritisoivatkin ketteriä menetelmiä dokumentoinnin puutteesta, kaivaten lisää konkreettisi ohjeita esimerkiksi juuri projektinhallintaan, niin todellisuudessa ketterien menetelmien yksikertaisemmat rakenteet ovat usein merkittävistä perinteisiä malleja helpommin omaksuttavissa myös ilman moni satasivuisia riskienhallinta oppaita tai muita dokumentteja [AgileAlliance, 2001; Jacobson, 2002].

Ketterillä menetelmillä onkin jo nykyisellään varsin merkittävä paikka ohjelmistokehityksen maailmamassa. Tarve ja tilaus tulevat vain kasvamaan entisestään alati muuttuvassa toimintaympäristössä. Ketterät menetelmät syntyivät kuitenkin alun perin korjaamaan juuri perinteisten menetelmien puutteita [Boehm, 2002]. Tästä johtuen kaiken toiminnan ja tekemisen taustalla on perusteltu tarve pitää projekti mahdollisimman kevyenä lopputuotteen koosta ja hankkeen vaikeudesta riippumatta.

Samaan aikaan kuitenkin ketterät menetelmät ovat kuitenkin erittäin muutos- ja riskitietoisia hankkeita, joissa riskit on yleensä minimoitu. Samalla myös projektin sisäisen ja ulkoisen toimintaympäristön tuntemus on merkittävästi parempi kuin perinteisillä menetelmillä toteutetuissa projekteissa [Cockburn, 2007]. Tämä näkyy konkreettisesti kouriin tuntuvana hallittavuutena kaikille projektin jäsenille. Ketterissä menetelmissä myös resurssit on kohdennettu juuri sinne, missä ne eniten tilaajalle merkitsevät ja viimekädessä myös näkyvät eli lopputulokseen [Chin, 2004].

6.3. Menetelmien vahvuudet ja heikkoudet

Perinteisillä ja ketterillä menetelmillä on lähes alusta asti ollut omat kannattajansa, Myös menetelmien toiminnan vahvuudet ja heikkoudet ovat olleet kaikkien tiedossa jo pidemmän aikaa. Taulukoissa 2 ja 3 on tehty yhteenveto näistä ominaisuuksista.

6.3.1. Perinteisten menetelmien vahvuudet ja heikkoudet

	VAHVUUDET	HEIKKOUDET
Vesiputous	• Formaali prosessi, joka etenee aina ennakkosuunnitelma mukaan • Prosessi selkeä ja yksinkertainen • Toimii kaikissa tutuissa ja stabiileissa toimintaympäristöissä projektin koosta riippumatta	• Tuntemattomissa tai uusissa kehityshankkeissa ongelmia • Kallis, hidas, joustamaton • Näkee ohjelmistotyön kokonaisuutena, jossa toteutetaan suuri työkokonaisuus kerralla • Sisältää riskialttiita osia ja työvaiheita • Määrittelyvaihe lukitsee toteutuksen • Muutosten huomioonottaminen vaikeaa • Ei tue nykyaikaista ohjelmistokehitystä, jossa inkrementaalisesti ja iteratiivisesti kasvava rakenne (esimerkiksi stabiilit ja kiireelliset osat ensin)
Spiraali	• Virheiden poistaminen ja korkea laatu etusijalla • Riskienhallinta olennaisena ja pakollisena osana • Huomioi jatkokehityksen ja uudelleen käytön • Käyttökelpoinen erityisesti suurissa ja haastavissa hankkeissa	• Vaatii soveltamista ja organisaation toiminnan muuttamista malliin sopivaksi • Julkaisupäivien määrittäminen etukäteen vaikeaa • Kallis, jos projektin on vaikea
Prototyyppi	• Määrittely ja toteutus toimivat rinta rinnan mikä pienentää toiminnan riskiä • Toimii hyvin myös uusilla toteutusalueilla	• Toimintaprosessi ja sen eteneminen varsin abstraktia

Taulukko 2: Perinteisten menetelmien vahvuudet ja heikkoudet

Taulukosta 2 voidaan nähdä, että perinteiset ohjelmistokehityksen menetelmät ovat parhaimmillaan varsin kapealla osalla projektien ja ohjelmistokehityksen maailmaa. Viimeisen 10 vuoden aikana menetelmät eivät enää ole myöskään merkittävästi kehittyneet tai muuttuneet, koska tutkimus niitä kohtaan on selkeästi vähentynyt.

6.3.2. Ketterien menetelmien vahvuudet ja heikkoudet

	VAHVUUDET	HEIKKOUDET
XP	• Vahvat tekniset toimintatavat • Tilaaja kontrolloi ominaisuuksia ja toteuttaja kontrolloi niiden toteutusta • Jatkuva vuorovaikutus ja palaute toiminnasta • Erittäin laajalti käytössä maailman laajuisesti	• Vaatii tilaajan aktiivista osallistumista jatkuvasti • Dokumentointi pääasiassa suullista tai osana ohjelmiston koodia • Arkkitehtuurin ja suunnittelun ongelmien ratkaisu haastavaa erityisesti uusilla käyttäjillä
SCRUM	• Ei syrjäytä kaikkia vanhoja toimintatapoja • Vapaasti muodostuvat tiimit ja toiminnan seuranta ja palautejärjestelmät • Tilaajan osallistuminen ja toiminnanohjaus • Toteutuksen ja muun toiminnan tärkeysjärjestys määräytyy liiketoiminnan tarpeiden mukaan • Selkeästi määritelty prosessi	• Ei varsinaista näkyvää projektinhallintaa, konkreettiset toimintatavat puuttuvat • Ei määrittele tarkkaan teknisiä toimintatapoja eri tilanteille • Organisaation muun toiminnan priorisointi menetelmän vaatimusten mukaan voi olla vaikeaa

DSDM	• Testauskeskeinen menetelmä, jossa vähintään yksi testaaja on kokoajan projektin käytössä • Suunniteltu alusta asti vastaamaan liiketoiminnan vaatimuksiin, joten tehdyille investoinneille odotetaan selkeästi vastinetta • Selkeä lähestymistapa, jossa määritellään kaikkien vaatimusten tärkeys kullekin iteraatiolle • Rajoittaa tilaajan ylisuuria odotuksia projektin alusta asti, jolloin joidenkin vaatimusten puuttuminen lopputuotteesta ei ole yllätys	• Raskaansarjan menetelmät, joka ei tästä syystä sovellu kaikille • Vaatii jatkuvaa varsinaisen käyttäjän osallistumista • Raskas dokumentaatio • Menetelmä lähdemateriaali on maksullista ja ongelmatilanteessa vastauksen löytäminen ilman näitä voi olla hankalaa
Crystal	• Crystal-menetelmän eri vaihtoehtojen avulla pystytään tehokkaasti arvioimaan projektin koko ja kriittisyys • Ainoa menetelmä joka ottaa oikeasti huomioon ihmisten elämän kannalta todella kriittiset projektit • Projektin koon kasvaessa, voidaan käyttää risti ja päällekkäisiä tiimejä varmistamaan eri osien yhteensopivuus ja toteutuksen yhtenäisyys • Ihmisnäkökulmaa korostetaan kokoajan projektin tukitoiminnoissa ja rakenteessa • Aktiivinen testaus olennainen osa	• Olettaa kaikkien tiimiläisten olevan aina fyysisesti läsnä, joten toimii huonosti hajautetuissa tai virtuaalisissa tiimeissä • Muutokset projektin rakenteeseen ovat erittäin yleisiä kun halutaan seurata valittua Crystal-menetelmää • Crystal menetelmästä toiseen siirtyminen ei ole mahdollista kesken projektin • Ei ole myöskään menetelmästä toiseen yhteensopiva
ASD	• Erittäin mukautuva erilasiin organisaatio kulttuureihin ja tehtäviin	• Toimintakonseptina tai kulttuurina, mutta ei kattava kehitysmenetelmä

	• Voidaan helposti ottaa käyttöön • Helpottaa yksittäisten ihmisten kiinnittäminen osaksi suurempaa kokonaisuutta	• Hierarkkiset tai jäykät organisaatiorakenteet eivät aina sovi vapaaseen ja luovaan kehitystekniikkaan
FDD	• Mahdollistaa useat rinnakkain sekä ristiin toimivat tiimit. • Kaikki projektin näkökulmat ovat johdettavissa tuotteen ominaisuuksiin • Suunnittele ja toteuta osa kerrallaan -tekniikka on helppo ymmärtää ja käyttää • Mukautuu erityisen hyvin kaikenlaisiin isoihin projekteihin ja usean henkilön ryhmiin	• Tukee yksilöllistä koodausta ja omistusta, yhteisomistajuuden sijasta • Iteraatioiden sisältö on erittäin avoin ja jättää paljon enemmän tulkinnan varaa kuin muissa ketterissä menetelmissä • Muutos selkeästä ja toimintamallikeskeisestä menetelmästä vapaampaan voi olla liian suuri

Taulukko 3: Ketterien menetelmien vahvuudet ja heikkoudet

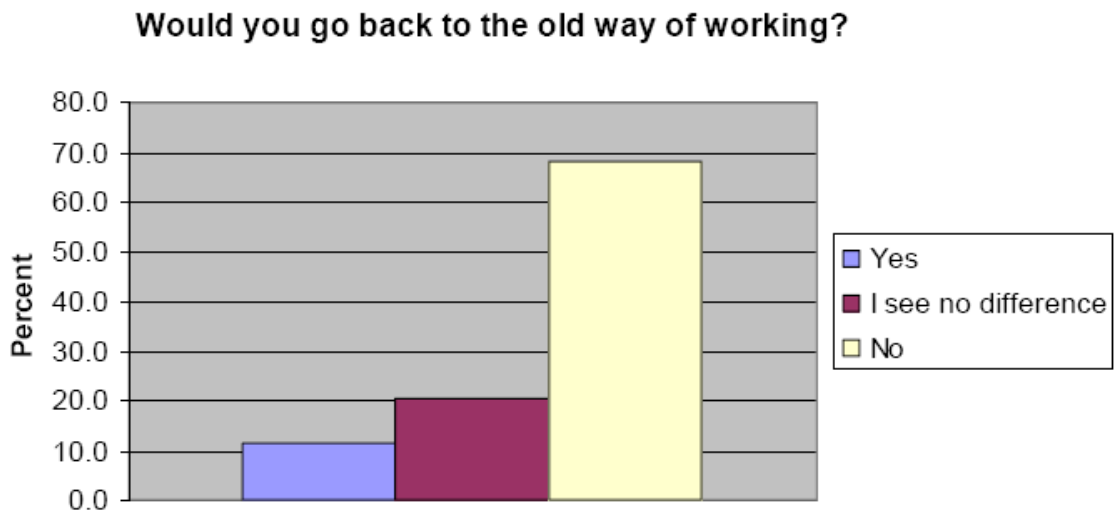
Taulukosta 3 voidaan nähdä, että ketterät menetelmät tarjoavat erittäin moninaisen ja monipuolisen joukon erilaisia toimintatapoja kaiken kokoisten ja aihe alueelta vaihtelevien projektien toteuttamiseen. Lisäksi aktiivinen tutkimustyö menetelmiä kohtaan tulee vain lisäämään niiden käyttö- ja soveltamismahdollisuuksia tulevaisuudessa.

6.4. Käyttäjien kokemuksia ketteristä menetelmistä

Niin kuin aikaisemmissa kappaleista voimme nähdä, kaikki ketterät menetelmät tuovat mukanaan hieman erilaisen tavan tehdä asioista ja toimia projekteissa. Vaadittavan muutoksen suuruus riippuu nyt käyttöön otettavasta uudesta menetelmästä sekä organisaation aikaisemmin käytössä olleista toimintatavoista. Mutta ennen kaikkea onnistuneen muutosprosessin läpiviennin ytimessä ovat sitä menetelmän käyttäjät eli kaikki projekteissa mukanaolevat henkilöt.

Viime aikoina ketteriä menetelmiä on ryhdytty käyttämään yhä kasvavassa määrin. Tämä kehitystrendi jatkuu yhä, mutta nyt mukaan on tullut pienten ohjelmistotalojen ja organisaatioiden lisäksi myös suurempia organisaatioita, jotka ovat pääsääntöisesti käyttäneet omissa projekteissaan perinteisiä ohjelmistokehitysmenetelmiä. Suomessa

esimerkiksi F-Secure, TeliaSonera, VTT ja Nokia sekä monet muutkin organisaatiot myös ulkomailla, ovat varsin menestyksekkäästi onnistuneet tekemään ketterien menetelmien vaatimat muutokset omien projektiensa toimintamalleihin. Kuvassa 26 on esitetty Nokian henkilöstölleen tekemän kyselyn tulos ketterien menetelmien käyttöönoton jälkeen [AgileAlliance, 2001; F-Secure, 2007; Nokia, 2006; Ribeiro *et al*, 2009; Sutherland, 2001; VTT, 2010].



Kuva 26: Ketterien menetelmien käyttökokemukset [Nokia, 2006, s.7]

Kuvasta 26 nähdään selkeästi se, miten tyytyväisiä joissakin organisaatioissa ollaan onnistuneen ketterien menetelmien käyttöönoton jälkeen. Pitää kuitenkin muistaa, että myös täysin päinvastaisia tuloksia julkaistaan lähes yhtä paljon. Syitä tähän on varmasti yhtä monta kuin on organisaatioitakin. Joskus muutos voi myös olla erittäin vaikea ja aikaa vievä, jolloin sen onnistuminen havaitaan vasta paljon myöhemmin. Vastaavasti se voi myös olla helppo, jolloin sen edut ovat heti käsin kosketeltavat [Boehm and Turner, 2005; Coram and Bohner, 2005; Nehur *et al*, 2005].

Juuri ketterien menetelmien mukanaan tuomat selkeät toiminnalliset edut ja erilaiset taloudelliset tekijät kuten säästöt sekä parempi tuottavuus sijoitetulle pääomalle perinteisiin menetelmien verrattuna ovat syynä siihen, miksi näihin vaativiin ja varsin riskialttiisiinkin muutosprosesseihin ollaan ylipäätänsä edes valmiita ryhtymään [Cockburn, 2007; Ocampo, 2007]

Selkeä mittari, jossa ketterissä menetelmissä käyttävien työskentelytapojen tehokkuus näkyy parhaiten, on toteutettavan ohjelmisto monimutkaisuus ja koko. Näitä voidaan mitata aliohjelmien, metodien ja funktioiden määrällä sekä koodirivien lukumäärän avulla. Perinteisten ja ketterien menetelmien toiminnan tehokkuuden muutokset

ohjelmistokoodissa siirryttäessä perinteisistä menetelmistä ketteriin menetelmiin on esitetty kuvassa 27 [Ocampo, 2007; Sanjay, 2006].

	Before Adopting Agile	After Adopting Agile	% Change
Total Code Size	45,773	15,048	-67%
Average Methods Per Class	630	10.95	+73%
Average Lines Per Method	11.36	5.86	-48%
Average Cyclometric Complexity	3.44	1.56	-54%

Kuva 27: Menetelmien ohjelmistokoodin toiminnan tehokkuuden vertailu [Sanjay, 2006, s.8]

Kuvasta 27 nähdään selkeästi se, millä tavoin ketterien menetelmien ohjelmointikäytännöt, kuten pariohjelmointi ja jatkuva testaus, vaikuttavat varsin merkittävästi projektin toimintaa jo lähdeoodi tasolta asti toimintaa tehostavasti. Pelkästään yksittäisten koodienrivien määrän väheneminen 67 % on erittäin merkittävä muutos. Mutta kun siihen vielä yhdistetään funktioiden ja metodien määrään lisääminen, joiden avulla ohjelmiston toiminta voidaan hahmottaa selkeämmin, niin muutosta voidaan pitää erittäin radikaalina. Samalla kun toteutettava ohjelmisto kevenee, niin vähenee myös mahdollisten virheiden ja ongelmien määrä. Pääsääntöisesti tämän seurauksena myös ohjelman vakauteen ja toiminnan vasteaikoihin tulee myös selkeitä positiivisia parannuksia [Sanjay, 2006].

6.5. Käytettävän menetelmän valinta

Kullekin projektille aina oikean kehitysmallin valinta on yksi kriittisimmistä projektin onnistumiseen vaikuttavista tekijöistä. Tässä jatkuvassa keskustelussa on useita erilaisia näkökulmia ja koulukuntia. Vielä muutama vuosi sitten vallalla olleen käsityksen mukaan, ketterät menetelmät soveltuvat hyvin vain tietyn tyyppisille projekteille. Tyypillisestä näitä olivat pienet tai keskisuuret uutta kehittävät hankkeet. Ja vastaavasti perienteisten mallien vahvuudet löytyivät suurista ja tutuista toimintaympäristöistä. Enää tämä jako ei kuitenkaan ole näin mustavalkoinen.

Viime aikoina myös erittäin vaikeita ja haastavia hankkeita on ryhdytty toteuttamaan ketteriä menetelmiä käyttämällä, ja vielä hyvällä menestyksellä. Tämä on aiheuttanut

varsin paljon keskustelua sekä synnyttänyt uutta tutkimusta, jonka avulla vanhoja ennakkoluuloja on pystytty todistamaan perusteettomiksi tai ainakin liioitelluiksi. Voidaankin olettaa, että ainakin osa ketterien menetelmien kritiikistä tai mahdollisista ongelmista juuri kriittisissä hankkeissa, ei ole niin perusteltua kuin voisi luulla [Boehm and Turner, 2003; Cockburn, 2007; Cohen *et al*, 2004; Payne *et al*, 2005; Ribeiro *et al*, 2009].

Tosin on myös totta, että ketterät menetelmät vaativat projektissa mukanaolijoilta paljon enemmän, muun muassa sitoutumista, aikaa ja osaamista, kuin perinteiset menetelmät, koska kaikkien panosta tarvitaan kokoajan. Lisäksi pitää myös muista, että myöskään kaikkiin projekteihin eivät ketterät tai perinteiset menetelmät yksinkertaisesti vain sovi. Ketterissä menetelmissä ongelmia voi edelleen tulla juuri suurten järjestelmien toteuttamisessa, aiemmin tehtyjen osien uudelleen käytössä sekä kriittisissä tai muuten yleiseen turvallisuuteen liittyvissä hankkeissa. Ja vastaavasti perinteisten menetelmien kohdalla suurimmaksi kompastuskiveksi nousee yhä edelleen dokumentointi ja toiminnan raskauden mukaan tuomat ongelmat. Yksi näkökulma projektin soveltuvuudesta joko ketterille tai perinteisille menetelmille on esitetty taulukossa 4 [Chin, 2004; Cockburn, 2007; Lindvall *et al*, 2004; Payne *et al*, 2005; Simons, 2002].

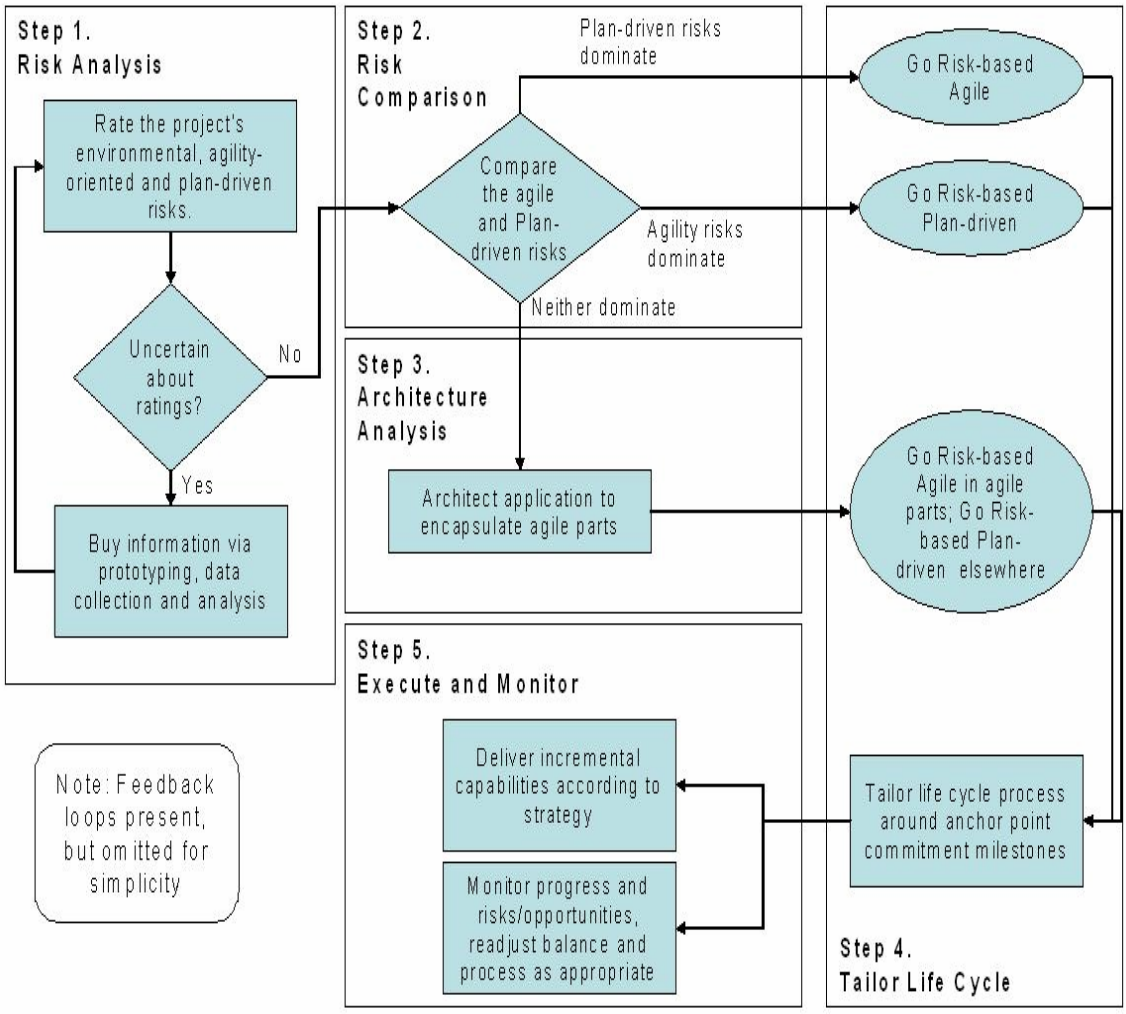
	PÄÄOSIN ULKOISTETTU PROJEKTI	**ORGANISAATION SISÄINEN PROJEKTI, JOSSA MUKANA USEITA OSASTOJA**	**ORGANISAATION SISÄINEN PROJEKTI**
Aikaisemman toiminnan kehitysprojekti	Perinteinen	Perinteinen	Perinteinen
Uuden tuotteen tai prosessin kehitysprojekti	Perinteinen/ Ketterä	Perinteinen/ Ketterä	Ketterä
Muu tekninen kehitysprojekti	Perinteinen/ Ketterä	Ketterä	Ketterä

Taulukko 4: Projektin soveltuvuus eri kehitysmenetelmille [Chin, 2004]

Taulukosta 4 voidaan vielä nähdä vanhan ajattelun vaikutuksia yhdistettynä uudempiin suuntauksiin ja tutkimuksiin, koska sekä perinteisille että ketterille menetelmille on löydetty varsin selkeät vahvuudet, mutta rajaa näiden välillä on hieman yritetty häivyttää.

Chin [2004] kuitenkin näkee ketterien menetelmien potentiaalin ja tunnustaa niiden monipuoliset käyttömahdollisuudet esimerkiksi ulkoistetuissa projekteissa, jota moni muu tutkija ei vielä tänäkään päivänä ole täysin hyväksynyt [Chin, 2004; Cohen *et al*, 2004].

Chin [2004] taulukossa 4 esittämä näkemys käytettävän kehitysmallinvalinnasta on vielä kuitenkin varsin suppea ja kapea-alainen, koska se ei ota huomioon riittävästi todellisia valintaan vaikuttavia asioita ja niiden eri ulottuvuuksia. Hieman laajemmin parhaiten kullekin projektille sopivan kehitysmenetelmän valintaa ja siihen johtavien seikkoja tarkastelevat Boehm ja Turner [2003] omassa artikkelissaan. Kehitysmallin valintaan johtava päätösketju ja sen vaiheet on esitetty kuvassa 28. Kuvasta on jätetty pois joitakin elementtejä selkeämmän esitystavan vuoksi.



Kuva 28: Menetelmän valinta ja siihen päätyminen [Boehm and Turner, 2003, s.60]

Kuvasta 28 voidaan nähdä, että Boehm ja Turner [2003] ovat pyrkineet luomaan loogisen riskeihin pohjautuvan päätösketjun, jonka avulla on mahdollista valita oikea

kehitysmalli kullekin projektille sen luonteesta riippumatta. Mallin avulla on tarkoitus saavuttaa parempia tuloksia sekä varmistaa projektin onnistuminen. Toiminta lähtee liikkeelle riskianalyysistä ja sen muut vaiheet ovat riskien vertailu, arkkitehtuurin analyysi, kehityksen etenemisen suunnittelu sekä toteutus ja valvonta. Mallin tarkoitus ei ole antaa absoluuttisia vastauksia, vaan toimi enemmänkin ohjenuorana, sillä viime kädessä projektin henkilöstön itse on tehtävä päätös projektin totutustavasta ja siinä käytettävistä välineistä sekä toimintamalleista.

Projektin kehitysmallin valinta on erittäin monimutkainen ongelma. Tämä korostuu erityisesti silloin, kun organisaation oma osaaminen riittää vain yhden, sille tutun, kehitystavan totuttamiseen, vaikka jokin toinen tapa voisi olla kyseiselle projektille paremmin sopiva. Tällöin onkin erittäin tärkeää huomata oman toiminnan rajoitteet ja ne riskit, jotka ollaan valmiita ottamaan joko tiedostamatta tai tiedostaen, samalla kun päätetään olla hankkimatta ulkopuolista osaamista ja käyttää vanhaa tuttua kehitysmallia kaikesta huolimatta. Toisaalta oman tavan erinomainen hallinta useissa erilaisissa toimintaympäristöissä ja näiden monimutkaisissa hankkeissa on tuonut mukanaan tietyn rutiinin, jonka avulla hyvä kehitystiimi pystyy voittamaan suuren joukon erilaisia vastoinkäymisiä ja ongelmia.

7. Pohdinta ja johtopäätökset

7.1. Menetelmien vaikutus yleisimpiin projektien riskeihin

Ketteristä menetelmistä puhuttaessa on varsin tavallista, että niiden käyttöönottoon liittyy kritiikkiä ja erilasia ongelmia raportoidaan vaihtelevia määriä. Erityisesti vanhoihin ja samalla perinteisiin ohjelmistokehitysmenetelmiin tottuneilla henkilöillä ja organisaatioilla on ollut vaikeuksia muuttaa toimintatapojaan ketterien menetelmien vaatimalla tavalla. Tästä huolimatta ketterien menetelmien omaksumista pidetään yhä yleisesti keinona, jonka avulla organisaatiot voivat kasvattaa projektin tuottavuutta ja parannetaan toteuttavien ohjelmistojen laatua sekä nyt ja myös tulevaisuudessa.

Ennen kaikkea projektien riskienhallintaan ketterät menetelmät vaikuttavat erittäin merkittävästi. Ketterien menetelmien mukanaan tuoma vaikutus, apu tai muu korjaustoimikeino Chaos Report [2009] tutkimuksessa esiin tulleisiin yleisimpiin projektien riskeihin sekä epäonnistumiseen tai keskeyttämiseen johtaneisiin syihin on esitetty taulukossa 5.

RISKI/ONGELMAN KUVAUS	KETTERIEN MENETELMIEN KORJAUSTOIMI
PROJEKTIN SUUNNITTELUN JA ALOITUKSEN ONGELMAT	
Resurssien puute	Vaikka ketterät menetelmät eivät suoraan ratkaise ajan, rahan ja henkilöiden saatavuuden ongelmia, niin ne auttavat perustemaan niiden todellisen tarpeen. Lisäksi ne tuottavat jatkuvasti jotakin konkreettista (inkrementti/valmis osa) ja pystyvät näin havainnollistamaan toiminnan etenemisen ja sen mitä juuri nyt tehdään sekä mitä vielä puuttuu.
Työmääräarvioin pettäminen	Työmääräarvioin tekemisen osallistuvat kaikki projektin jäsenet, jolloin yhteinen tavoite ja näkemys siihen pääsemiseksi muodostuvat paremmin. Myös iteratiivinen ja/tai inkrementaalinen kehitysprosessi helpottavat arvion jakamista pienempiin ja paremmin hallittavaan osiin. Samalla myös pienten muutosten tekeminen on mahdollista.

TEKNISET JA VAATIMUKSIIN LIITTYVÄT ONGELMAT	
Epätäydelliset vaatimukset ja määritykset	Ketterissä menetelmissä vaatimusten oletetaan olevan epätäydelliset, koska niitä tullaan joka tapauksessa täydentämään projektin edetessä.
Muuttuvat vaatimukset ja määritykset	Ketteriin menetelmiin kuuluu vaatimusten muuttaminen ja täydentäminen projektin edetessä. Tätä ei nähdä huonona asiana vaan tarpeellisena, minkä vuoksi lopputulos on vastaa entistä paremmin tilaajan todellisia tarpeita.
Lopputulos ei vastaa tilaajan teknisiä tai laadullisia tarpeita	kevyen ohjelmistorakenteen, pariohjelmoinnin ja jatkuvan testauksen sekä tähän kuuluvan eri osien yhteen sovittamisen ansioista suurin osa teknisistä ja laadullista ongelmista pystytään välttämään tai ainakin tarvittaessa korjaaman varsin nopeasti.
PROJEKTIN HALLINNAN JA TYÖTAPOJEN ONGELMAT	
Käyttäjien osallistumisen puute	Suurin osa ketteristä menetelmistä vaatii asiakkaan aktiivista ja säännöllistä osallistumista lähes koko projektin ajan. Jo kehitysmenetelmää valittaessa tilaaja on sitoutettava tähän tapaan toimia, jotta projektin lopputulos olisi paras mahdollinen.
Käyttäjien palautteen puuttuminen	Aktiivisesta osallistumisesta johtuen tilaaja voi kokoajan helposti antaa virallista ja epävirallista palautetta tai muita ohjeita (suullinen ja kirjallinen) kaikille projektissa mukanaolijoille. Kommunikaatio toimii myös päinvastaisen suuntaan.
Johtajuuden ja vastuun puute	Ketterissä menetelmissä vastuuta delegoidaan vahvasti kaikille mukana olijoille, jolloin jokainen on vastuussa itsensä lisäksi myös muista. Projektipäällikön rooliksi jää projektin esteiden poistaminen ja toiminnan etenemisen varmistaminen sekä toimivan työskentely ilmapiirin luominen.

Taulukko 5: Ketterien menetelmien keinot yleisimpien riskienhallintaan

Taulukosta 5 voidaan nähdä, että ketterät menetelmät pystyvät varsi hyvin vastaamaan erilaisiin projektien ongelma- ja riskitilanteisiin. Tämän johdosta projektit ovatkin melko vakaita, vaikka niiden sisäisessä tai ulkoisessa toimintaympäristössä tapahtuisi jatkuvasti erilasia muutoksia.

Vastaavasti perinteisten menetelmien vaikutus, apu tai muu korjauskeino projektien yleisimpiin riskeihin, epäonnistumiseen tai keskeyttämiseen johtaneisiin ongelmakohtiin on esitetty taulukossa 6.

	RISKI/ONGELMAN KUVAUS	**PERINTEISTEN MENETELMIEN KORJAUSTOIMI**
AIKATAULU	Projekti myöhästyy	GANTT -kaavion tai muiden työmäärän seuranta- tai arviointityökalujen käyttö ja ylläpito.
KUSTANNUS	Budjetti ylittyy	Budjetin tarkka seuranta tehtävä tehtävältä koko projektin ajan.
TOIMINTA	Tulos ei vastaa tilaajan tarpeita	Jokaisessa vaiheessa käyttäjien arviot sekä parannus- ja muutosehdotukset.
LAATU	Lopputuote ei vastaa määrittelyä ja sisältää virheitä tai hitautta	Jokaisen osion täydellinen viimeistely ja perusteellinen testaus ennen luovutusta.
ULKOISESTI	Jokin osan toimitus puutuu tai viipyy	Tarkistuspisteet kaikille osille ja työvaiheille.
POLIITTISET	Projektia arvostellaan vaikka se olisikin täyttänyt vaatimukset ja lopputulos olisi muutoinkin hyvä	Kaikkien osapuolten sitouttaminen ja muutosvastarinnan murtaminen.

Taulukko 6: Perinteisten menetelmien korjaustoimet projektin eri ongelmakohtiin

Taulukosta 6 voidaan nähdä, että myös perinteisistä menetelmät löytyvät keinot riski- ja ongelmatilanteiden ratkaisemiseen. Mutta samoin kuin varsinaiset ohjelmistokehitys menetelmätikin, niin niiden tarjoamat ratkaisutkin ovat varsin hitaita ja kalliita, koska ne

eivät ole luontainen ja kiinteä osa kaikkia perinteisiä menetelmiä. Samaan tapaan kuin ne ketterissä menetelmissä ovat.

Vaikka kaikki ketterät menetelmät ovat erittäin riskitietoisia, niin ilman osaavia henkilöitä ei mikään projekti kuitenkaan onnistu. Projektit tarvitsevat oikeiden henkilöiden mukaan tuoman tiedon, taidon ja jatkuvan työpanoksen käyttämään niitä työkaluja, joita ketterät menetelmät tarjoat.

7.2. Ketterä vs perinteinen ohjelmistokehitys

Ensisilmäyksellä monet ketterät menetelmät tuntuvat varsi kaoottisilta ja vailla selkeää hallintaa tai kontrollia. Käytännössä kuitenkin ketterien kehitysmenetelmien yleiset toimintatavat sekä sen piilevä rakenteellisuus toimivat tavallaan automaattisena ja näkymättömänä projektinhallintana kaiken muun toiminnan rinnalla. Ketteriä menetelmiä voidaankin siis pitää jo lähtökohtaisesti, ja vastoin yleisiä oletuksia, varsin hallittuna sekä riskitietoisina menetelmiä. Tämä piilevä kontrolli onkin erittäin merkittävä syy siihen, miksi ketterät menetelmät ylipäätänsä toimivat varsin tehokkaasti. Ja samalla myös paine ja tarve niiden käyttämiseen on suurimmillaan juuri erityisen riskialttiissa ja muutosherkissä toimintaympäristöissä.

Eri tahojen ja osapuolten välinen jatkuva reaaliaikainen kommunikointi ja tähän kiinteästi liittyvä yhdessä tekeminen johtaa väistämättä siihen, että projektin työtehtävät sekä toiminnan eteneminen/tilanne ovat aina kaikkien osapuolten, myös asiakkaan tiedossa. Vaikka ketterissä menetelmissä korostetaankin tiimiä, yhdessä toimimista sekä kommunikaatiota, on taustalla kuitenkin voimakkaat ja osaavat yksilöt. Tiimiläisten pitää osata toimia sekä yksin, että olla valmiita ottamaan valtaa ja kantamaan vastuuta myös muista. Kokoajan pitää lisäksi myös olla luovia ja innovatiivisia, jotta projekti etenee mahdollisimman tehokkaasti. Kriteerit projektiin valittavilla henkilöille ovat siis melko korkeat. Mutta jos niistä pidetään kiinni, niin projektinhallinta ja kokonaisuus seuraavat mukana, myös ilman mittavaa ja aika vievää asioiden dokumentointia ja jatkuvaa suunnittelua.

Ketterän ohjelmistokehityksen merkittävät eroavuudet perinteisiin ohjelmistokehitysmenetelmiin nähden tuovat mukaan erilasia ongelmia. Näiden todellinen vakavuus ja vaikutukset ovat kuitenkin lähes täysin riippuvaisia niiden raportoijista. Samaa aikaa ketterien menetelmien käyttö kuitenkin myös yleistyy jatkuvasti. Kaikesta huolimatta, edut perinteisiin menetelmiin nähden ovat kuitenkin kiistattomat ja yleisesti tunnustetut. Kysymys on enää oikeastaan siitä, kuinka paljon näitä etuja todellisuudessa saavutetaan. Lisäksi uusia ohjelmistokehitys ja projektinhallinta menetelmiä syntyy

kokoajan, joten on vain ajankysymys milloin jokin täysin uusi menetelmä tai aikaisempien menetelmien uudenlainen yhdistäminen syrjäyttää jälleen vahan tavan tehdä asioita

Nopealla vilkaisulla katsottuna ketterät menetelmät vaikuttavat lähes ylivertaiselta vaihtoehdolta perinteisille menetelmille, vaikka todellisuudessa näin ei kuitenkaan ole, sillä mikään menestysautomaatti nekään eivät kuitenkaan ole. Tästä syystä ketterät menetelmät eivät tuskin koskaan tule täysin korvaamaan perinteisiä menetelmiä. Lisäksi ohjelmistotuotanto ja projektit tarvitsevat jatkuvasti hyvin erilaisia toteutusvaihtoehtoja ja malleja, joten käytettävien vaihtoehtojen rajoittamista kapealle sektorille ei voi pitää perusteltuna. Suurimpana ohjelmistoalan muutosvoimana ja kehityksen suunnan näyttäjä toimivat kuitenkin ohjelmistojen tilaajat, jotka asettavat toimittajille jatkuvasti kasvavia vaatimuksia toteuttaa parempia ja laadukkaampia ohjelmistoja entistä, nopeammin ja entistä halvemmalla.

7.3. Jatkotutkimus

Ketteriä menetelmiä tutkitaan juuri nyt hyvin paljon useista eri näkökulmista. Omasta mielestäni kaksi mielenkiintoisinta jatkotutkimus aluetta ovat ketterien menetelmien empiirinen vertailu erilaissa projektiympäristöissä eli organisaatioin sisäisissä ja ulkoisissa, erityisesti globaaleissa projekteissa. Organisaatioiden jatkuvasti ulkoistaessa toimintaansa ympäri maapallo, olisi erittäin tarpeellista tutkia sitä, miten ketterät menetelmät toimivat virtuaalisessa kehitysympäristössä ja mitä ongelmia tai haasteita tämä tuo mukanaan.

Toinen aina ajankohtainen jatkotutkimusalue voisi olla ketterien menetelmien onnistumisen (success) syihin paneutuva kattava tapaustutkimus. Tämän tyyppinen tutkimus olisi varmasti varsin tervetullut, jos sen kautta olisi mahdollista muodostaa joitakin yleisiä menetystekijöitä edes pienelle joukolle projekteja. Olemme kuitenkin tähän asti menestyksekkäästi onnistuneet vain toistamaan samoja virheitä vuosi toisensa jälkeen, niin kuin Chaos Report [2009] tutkimus on osoittanut.

8. Yhteenveto

Olen tässä tutkielmassa pyrkinyt tarkastelemaan projekteja, perinteisiä ja ketteriä ohjelmistokehitysmenetelmiä sekä näiden riskienhallintaan mahdollisimman monipuolisesti useista eri näkökulmista.

Tutkielman aluksi käsittelin projekteja yleisellä tasolla, jonka jälkeen siirryin tutkimaan yksityiskohtaisemmin perinteisen ja ketterän ohjelmistokehityksen menetelmiä ja niiden toimintamalleja. Tämän jälkeen ryhdyin tarkentamaan omaa tutkimussuuntaani kohti ennakoivan ja reagoivan riskienhallinnan perusteita sekä juuri niitä riskejä jotka ovat läsnä tämän päivän projekteissa. Lopuksi keskityin tekemään vertailevaan tutkimusta ketterien ja perinteisten ohjelmistokehitysmenetelmien riskienhallinnan eroavuuksia ja niistä riskienhallinnan syistä, jotka vaikuttavat projektien onnistumiseen. Samalla käsittelin myös käytettävän kehitysmallin valintaa projektin riskienhallinnan ja sitä kautta myös onnistumisen näkökulmasta.

Tutkielman varsinaisena tavoitteena oli tutkia sitä, onko ketterille ohjelmistokehitysmenetelmille olemassa niille tunnusomaisia riskienhallinnan toimintatapoja tai tekniikoita, joihin niiden menetys mahdollisesti perustuu? Ja lisäksi, onnistuvatko ketteriä ohjelmistokehitysmenetelmiä käyttävät projektit perinteisiä kehitysmenetelmiä käyttäviä projekteja useammin tai paremmin? Halusin tutkia erityisesti juuri näitä asioita, koska yhä aivan liian suuri prosentti projekteista ei saavuta toivottua lopputulosta niille asetetussa budjetissa ja aikataulussa.

Tutkimus tavoitteet saavutettiin varsin hyvin. Tutkimuksen tuloksena voidaan sanoa, että riskienhallinta on koko projektin läpi kestävä prosessi, jonka onnistumisesta riippuu usein koko projektin lopputulos. Tästä syystä se asettaa erityisen suuria vaatimuksia siitä vastuussa oleville henkilöille. Voidaan myös todeta, että ennakoivan ja reagoivan riskienhallinnan eri osat kulkevat aina käsi kädessä ja paras lopputulos saavutetaan kokonaisvaltaisella riskienhallinnalla. Kuitenkaan mitään kaikkiin projekteihin ja tilanteisiin sopivaa riskienhallinnan oikotietä ei ole olemassa. Kokonaisvaltainen riskienhallinta vaatii jatkuvasti entistä enemmän työtä, mutta se on vain hyväksyttävä. Ainoa neuvo tai suositus, jonka voin antaa missä tahansa projektissa mukana oleville henkilöille, on suhtautua riskienhallintaa sen vaatimalla vakavuudella.

Lisäksi huomattiin, että käyttöön ottoon liittyvästä kritiikistä huolimatta ketterät menetelmät onnistuvat riskienhallinnassa perinteisiä menetelmiä paremmin, koska niiden toiminnan rakenteeseen on sisällytetty hyvin paljon riskienhallinnan eri elementtejä. Samalle ne myös pystyvät vastaamaa projektien vaatimuksia sekä muita asioita koskeviin odottamattomiin tilanteisiin tai muutoksiin perinteisiä menetelmiä nopeammin ja

paremmin. Lisäksi ketteriä menetelmiä käyttävissä projekteissa tilaajat saavat enemmän vastinetta kaikelle sijoittamalle pääomalle, koska niiden tarvitsemat resurssit ovat selkeästi pienemmät ja tarpeet perustellummat. Voidaan siis sanoa, että ketterät menetelmät mahdollistavat sekä taloudellisesti kannattavamman, teknisesti laadukkaamman että yleiseltä lopputulokseltaan paremman ohjelmisto toteuttamisen.

Tutkimuksen lopputulos ja tulosten merkitys voidaan nähdä kahdesta näkökulmasta. Ensinnäkin se voidaan nähdä aikaisemman tiedon vahvistajana projekteista ja riskienhallinnasta. Tämän lisäksi, se voidaan ennen kaikkea nähdä selkeänä tuen osoituksena ketterien menetelmien perusarvoille ja periaatteille sekä kannustimena näitä noudattavien menetelmien käytön lisäämiselle.

Viiteluettelo

[Abrahamsson *et al*, 2002] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen and Juhani Warsta, Agile software development methods, Review and analysis, VTT Publications, Espoo, 2002, http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf (viitattu 13.11.2010).

[AgileAlliance, 2001] Agile Alliance, 2001, http://www.agilealliance.org (Viitattu 2.1.2010).

[AgileManifesto, 2001] Agile Manifesto, 2001, http://agilemanifesto.org/ (Viitattu 2.1.2010)

[Artto *et al*, 2008] Karlos Artto, Miia Martinsuo ja Jaakko Kujala, *Projektiliiketoiminta*, 2. painos, WSOY, Helsinki, 2008.

[Artto and Kähkönen, 1997] Karlos Artto and Kalle Kähkönen, *Managing Risks in Projects*, Routhledge, London, United Kingdom, 1997.

[Beck and Andres, 2006] Kent Beck and Cynthia Andres, Extreme Programming Explained: Embrace Change, 2nd Edition, Addison-Wesley, Boston, United States, 2006.

[Boehm, 1988] Barry W. Boehm, A spiral model of software development and enhancement, *IEEE Computer*, Vol.**21,** No.5, May, 1988, Pp. 61-72.

[Boehm, 1989] Barry W. Boehm, Software risk management, Keynote Paper, ESEC '89, Volume **387**, 1989.

[Boehm, 1991] Barry W. Boehm, Software Risk Management: Principles and Practices, *IEEE Software*, Vol.**8,** No.1, 1991, Pp.32–41.

[Boehm, 2000a] Barry W. Boehm, Project Termination doesn't equal project Failure. Computer, Vol.**33**, No.9, September 2000, Pp.94–96.

[Boehm, 2000b] Barry Boehm, Spiral Development: Experience, Principles, and Refinements. Special Report CMU/SEI-00-SR-08, ESC-SR-00-08, June, 2000.

[Boehm, 2002] Barry Boehm, Get ready For the Agile Methods, With Care, *Computer*, Vol.**35**, 1/2002, Pp.64–69.

[Boehm and Turner 2003] Barry Boehm and Richard Turner, Using Risk to Balance Agile and Plan- Driven Methods, *IEEE Computer*, Vol.**36**, Issue 6, June 2003, Pp.57–66.

[Boehm and Turner, 2005] Barry Boehm and Richard Turner, Management Challenges to Implementing Agile Processes in Traditional Development Organizations, Special Issue, *IEEE Software*, Vol.**22**, No.5, 2005, Pp.30-39.

[Chaos Report, 2009] CHAOS Report (Updated 2009) and database, Standish Group International, 2009, http://www1.standishgroup.com/newsroom/chaos_2009.php (viitattu 27.11.2009).

[Charette, 1996] Robert N. Charette, Large-Scale Project Management Is Risk Management, IEEE Software, Vol. **13**, No 4, July 1996, Pp.110-117.

[Charette, 2001a] Robert N. Charette, Fair fight? Agile versus heavy methodologies, *Cutter Consortium e-Project Management Advisory Service*, Vol.**13**, 2001.

[Charette, 2001b] Robert N. Charette, The fight is on? Agile versus heavy methodologies, *Cutter Consortium e-Project Management Advisory Service*, Vol.**14**, 2001.

[Charette, 2001c] Robert N. Charette, Fists are flying? Agile versus heavy methodologies, *Cutter Consortium e-Project Management Advisory Service*, Vol.**17**, 2001.

[Charette, 2001d] Robert N. Charette, R. The decision is in: Agile versus heavy methodologies, *Cutter Consortium e-Project Management Advisory Service*, Vol.**19**, 2001.

[Chin, 2004] Gary Chin, *Agile project management: How to succeed in the face of changing project requirements*, AMACOM, New York, United States, 2004.

[Coad *et al*, 1999] Peter Coad, Eric Lefebvre and Jeff De Luca, Java Modeling in Color With UML: Enterprise Components and Process. Prentice Hall International, USA, 1999.

[Cockburn, 2007] Alistair Cockburn, *Agile Software Development - The cooperative game* (2nd edition.), Addison-Wesley, Boston, United States, 2007.

[Cohen *et al*, 2004] David Cohen, Mikael Lindvall, And Patricia Costa, An Introduction to Agile Methods, Advances In Computers, VOL.**62**, 2004.

[Coram and Bohner, 2005] Michael Coram and Shawn Bohner, The impact of agile methods on software project management, ECBS '05, 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, *IEEE Computer Society*, 2005, Pp.363 - 370.

[Crystal, 2009] Crystal Family, http://crystalmethodologies.org/ (Viitattu 21.12.2009).

[DeMarco and Lister, 1987] Tom DeMarco and Timothy Lister, *Peopleware: Productive Projects and Teams*, Dorset House, New York, 1987.

[DSDM, 2002] DSDM Consortium - Enabling Business Agility, http://www.dsdm.org/ (Viitattu 13.1.2010).

[F-Secure, 2007] F-Secure, F-Secures Transition to Agile, 2007, http://www.scrumalliance.org/resources/286 (viitattu 12.4.2010).

[Glass, 2001] Robert L. Glass, Agile Versus Traditional: Make Love, Not War! *Cutter IT Journal*, Vol.**14,** No.12, 2001 Pp.12–18.

[Extreme Programming, 2002] Extreme Programming: A Gentle Introduction, 2002, http://www.extremeprogramming.org/ (Viitattu 12.12.2009).

[Haikala ja Märijärvi, 2004] Ilkka Haikala ja Jukka Märijärvi, *Ohjelmistotuotanto* (10. uudistettu painos), Talentum Media Oy, 2004.

[Highsmith, 2000] Jim Highsmith, Retiring Lifecycle Dinosaurs, *Software Testing & Quality Engineering*, July/August, 2000, Pp.22–28.

[Higuera and Haimes, 1996] Ronald P. Higuera and Yacov Y. Haimes, Software Risk Management, Technical Report (Final rept), Carnegie-Mellon University Pittsburgh Pa Software Engineering Inst, June 1996.

[Hoch *et al*, 2000] Deltiv J. Hoch, Cyriac R. Roeding, Gert Pukert, Sandro K. Lindner & Ralph Müller, *Secrets of Software Success: Management Insights from 100 Software Firms around the World*, Harvard Business School Press, Boston, USA, 2000.

[Jacobson, 2002] Ivar Jacobson, A Resounding "Yes" to Agile Processes - But Also to More, *Cutter IT Journal*, Vol.**15**, No.1, 2002, Pp.18–24.

[Järvinen ja Järvinen, 2004] Pertti Järvinen ja Annikki Järvinen, *Tutkimustyön metodeista*, opinpajan kirja, Tampere, 2004.

[Keil *et al*, 1998] Mark Keil, Paul E. Cule, Kalle Lyytinen and Roy C. Schmidt, A framework for identifying software project risks, ACM Press, *Communications of the ACM*, Volume **41** Issue 11, November 1998.

[Kontio, 2001] Jyrki Kontio, Software Engineering Risk Management - A Method, Improvement Framework, and Empirical Evaluation, Väitöskirja, Helsingin Yliopisto, http://lib.tkk.fi/Diss/2001/isbn951225655X/isbn951225655X.pdf (Viitattu 24.11.2009).

[Kuusela ja Ollikainen, 2005] Hannu Kuusela & Reijo Ollikainen (toim.), *Riskit ja riskienhallinta*, Tampere University Press, Tampere, 2005.

[Lindvall *et al*, 2004] Mikael Lindvall, Dirk Muthig, Aldo Dagnino, Christina Wallin, Michael Stupperich, David Kiefer, John May and Tuomo Kähkönen 2004. Agile software development in large organizations. *IEEE Computer*, Vol. **37**, 12/2004, Pp.26–34.

[Lyytinen and Hirschheim, 1987] Kalle Lyytinen, and Rudy A. Hirschheim, Information systems failures: a survey and classification of the empirical literature, Oxford Surveys in Information Technology, Volume **4**, 1987, Pp.257–309.

[Markus, 2004] Lynne M. Markus, Technochange management: Using IT to drive organizational change, *Journal of Information Technology*, Volume **19**, Number 1, March 2004, Pp.4–20.

[Nehur *et al*, 2005] Sridhar Nerur, Radhaknata Mahapatra, George Mangalaraj, Challenges of migrating to agile methodologies, *Communications of the ACM* , Vol.**48**, No.5, May 2005, Pp.73-78.

[Nelson et al, 2008] Christopher R. Nelson, Gil Taran, Lucia de Lascurain Hinojosa, Explicit Risk Management in Agile Processes, *Agile Processes in Software Engineering and Extreme Programming*, Vol.**9**, 2008 Pp.190-201.

[Nokia, 2006] Nokia Networks and Agile Development, August 2006, Http://www.odd-e.com/material/2006/nokia_agile.pdf (Viitattu 2.2.2010).

[Ocampo, 2007] Joe Ocampo, Agile vs. traditional development cost models, Http://www.lostechies.com/blogs/joe_ocampo/archive/2007/09/20/Agile-vs-traditional-development-cost-models-maybe.aspx (Viitattu 1.3.2010).

[Olson, 2008] Rolf Olsson, Risk management in a multi-project environment: An approach to manage portfolio risks, International, *Journal of Quality & Reliability Management*, Volume **25**, Issue 1, 2008, Pp.60–71.

[Palmer, 2002] Stephen Palmer, Feature Driven Development and Extreme Programming, *The Coad Letter: Modeling and Design Edition*, Issue **70**, Feb 2002.

[Payne *et al*, 2005] Bob Payne, Fred Sencindiver, Susan Woodcock, Sanjiv Augustine, Agile project management: steering from the edges, *Communications of the ACM*, Vol. **48**, Issue 12, Dec 2005, Pp.85–89.

[Pelin, 2008] Risto Pelin, *Projektinhallinnan käsikirja,* 5 uudistettu painos. Gummerus, Jyväskylä, 2008.

[Perminova *et al*, 2007] Olga Perminova, Magnus Gustafsson and Kim Wikström, Defining uncertainty in projects – a new perspective, ACM Press, *International Journal of Project Management*, In Press, Corrected Proof, 2007.

[Pfleeger, 2000] Shari Lawrence Pfleeger, Risky business: what we have yet to learn about risk management, *The Journal of Systems and Software*, Volume **53**, Issue 3, 2000, pp.265–273.

[PMBOK Guide, 2008] PMBOK Guide: A Guide to the Project Management Body of Knowledge, 4th Edition, Project Management Institute, United States, 2008.

[Pohjonen, 2002] Risto Pohjonen, *Tietojärjetelmien kehittäminen*, Docendo Finland Oy, Sanoma WSOY, Jyväskylä, 2002.

[Pressman, 2010] Roger S. Pressman, *Software Engineering: A Practitioner's approach*, Seventh Edition, McGraw-Hill Higher Education, 2010.

[Projektiyhdistys, 2008] Projektiyhdistys Ry, Toimituskomitea Esa Koskelainen, Kalle Kähkönen, Jukka Lahtinen, Pekka Mäkelä, Juhani Silvasti ja Jouko Vaskimo, *Projektin johdon pätevyys*, Project Management Association Finland, 2008.

[Ribeiro *et al*, 2009] Luis Ribeiro, Cristine Gusmao, Wilmar Feijo, Victor Bezerra, A case study for the implementation of an agile risk management process in multiple projects environments, International Conference on 2-6 Aug, 2009, Management of Engineering & Technology, Portland, PICMET 2009, Pp. 1396-1404.

[Riskienhallinta kirjanen, 2000] Riskienhallinnan perusteet (Pk-yrityksille ja työntekijöille), kirjanen ja kalvot, Valtion teknillinen tutkimuskeskus,

http://www.pk-rh.fi/tyovalineet/tyovalineiden-tulostettavat-versiot/ (viitattu 20.12.2009).

[Ropponen, 1999] Janne Ropponen, Software Risk Management - Foundations, Principles and Empirical Findings, Väitöskirja, Jyväskylän yliopisto, 1999.

[Ruohonen ja Salmela, 2005] Mikko Ruohonen ja Hannu Salmela, *Yrityksen tietohallinto*, 3 painos, Edita Publishing Oy, Helsinki, 2005.

[Ruuska, 2007] Kai Ruuska, *Pidä projekti hallinnassa. Suunnittelu, menetelmät, vuorovaikutus*, 6. painos, Talentum Media, 2007.

[Sanjay, 2006] Addicam.V.Sanjay, Overview of Agile Management & Development Methods, The PROJECT PERFECT White Paper Collection, http://projectperfect.com.au/downloads/Info/info_agile_programming.pdf (viitattu 1.3.2010).

[Schwaber, 2000] Ken Schwaber, Against a Sea of Troubles: Scrum Software Development, *Cutter IT Journal*, Vol.**13**, No.11, 2000, Pp.34–39.

[Schwaber, 2004] Ken Schwaber, *Agile Project Management with Scrum*, Microsoft Press, Redmond, 2004.

[Scrum, 2009] Scrum - It's About Common sense, Http://www.controlchaos.com (Viitattu 2.12.2009).

[Simons, 2002] Matt Simons, Big and agile? *Cutter IT Journal*, Vol.**15**, No.1, 2002, Pp. 34–39.

[Sommerville, 2007] Ian Sommerville, *Software Engineering* (8th edition), Addison-Wesley, Essex, 2007.

[Stapleton, 1997] Jennifer Stapleton, *Dynamic Systems Development Method*, Addison-Wesley, 1997.

[Stapleton, 2003] Jennifer Stapleton (toim.), DSDM Business Focused Development (Second edition), Pearson Education, United States, 2003.

[Suominen, 2000] Arto Suominen, *Riskienhallinta*, WSOY, Vantaa 2000.

[Suomen SL, 1981] Suomen Standardisoimisliitto. Projektitoiminta, verkkosanasto, Suomen Standardisoimisliitto, 1981.

[Sutherland, 2001] Jeff Sutherland, Agile can scale: Inventing and Reinventing SCRUM in Five Companies, *Cutter IT Journal*, Vol.**14**, No.12, 2001 Pp.5–11.

[Taylor, 2006] Hazel Taylor, Critical risks in outsourced IT projects: the intractable and the unforeseen, ACM Press, *Communications of the ACM*, Volume **49** Issue 11, November 2006.

[VTT, 2010] VTT – Valtion Teknillinen Tutkimuskeskus, Mobile-D, 2010, http://agile.vtt.fi/ (viitattu 12.4.2010).

[Wallace and Keil, 2004] Linda Wallace and Mark Keil, Software project risks and their effect on outcomes, ACM Press, *Communications of the ACM*, Volume **47** Issue 4, April, 2004.

[Welke, 1981] Richard J. Welke, IS/DSS: DBMS support for information systems development. ISRAM WP-8105_1.0, McMaster University, Hamilton, Canada, 1981.

[Williams, 1997] Terry M. Williams, Empowerment vs risk management, *International Journal of Project Management*, Vol.**15**, No.4, 1997, pp.219–222.