Tomi Airaksinen

KETTERÄT MENETELMÄT

Tietojärjestelmätieteen
Kandidaatin tutkielma
22.6.2004

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Airaksinen, Tomi Tapio
Tietojärjestelmätiede, Ohjelmistotuotannon linja.
Ketterät menetelmät
Jyväskylä, Jyväskylän yliopisto, 21.5.2004.
Kandidaatin tutkielma, 30 sivua.

Ohjelmistokehityksessä esiintyy jännittyneisyyttä laadun, kustannusten ja ajan välillä. Kustannustehokkaan ja laadukkaan järjestelmän toimittaminen nykypäivän (constrained) aika teennäisille markkinoille on vaikea tehtävä. Monet perinteiset ohjelmistoprosessit ovat huippuunsa raskaita dokumentaation ja kankeiden kontrollimekanismien vuoksi, tehden vaikeaksi soveltaa niitä erilaisiin ohjelmistoprojekteihin. Uudet ohjelmistokehitysmenetelmät, ketterät menetelmät, ovat tekemässä tuloaan ohjelmistoteollisuuteen. Nämä menetelmät esiteltiin vuoden 2001 alussa ketterän allianssin manifestin muodossa. Ketterän allianssin manifesti koostuu neljästä arvosta ja kahdestatoista periaatteesta, jotka kuvaavat tarkoin kaikille ketterille menetelmille yhteisen ideologisen taustan. Ketterät menetelmät keskittyvät mieluummin ohjelmistokoodiin kuin dokumentaatioon. Niitä kutsutaan ketteriksi koska, toisin kuin perinteiset ohjelmistoprosessit, ne ovat mukautumiskykyisiä muuttuviin ympäristötekijöihin ja asiakasvaatimuksiin, eivätkä kankeita liikkeissään.

Ensiksi tutkimuksen tarkoituksena on tarkastella ketterän allianssin manifestia ja sen kautta johtaa ydinasioita ja ominaisuuksia, jotka ovat yhteisiä erilaisille ketterille menetelmille. Tätä kautta pyritään luomaan kuvaus sille mitä termillä ”ketterä” tarkoitetaan ja mikä tekee ohjelmistokehitysmetodista ketterän. Toiseksi tarkoituksena on käydä läpi olemassa olevia menetelmiä ja tarkastelemalla mitä ominaisuuksia niillä on ja millaisiin projekteihin ne ovat soveliaita. Menetelmät (Extreme Programming (vrt. XP), Scrum ja Crystal metodologiaperhe) esitellään käymällä läpi rakenne, jossa tarkastellaan menetelmän prosessia, rooleja ja vastuita, ja käytäntöjä.

AVAINSANAT: Ohjelmistotuotanto, ketterät menetelmät, ketterän allianssin manifesti, XP, Scrum, Crystal metodologiaperhe

SISÄLTÖ

1 JOHDANTO

Ohjelmistokehitys on muuttunut merkitsevästi viimeisen vuosikymmenen aikana. Liiketoimintaympäristön jatkuva muutos, kiristyvä kilpailu ja teknologian kehittyminen asettavat uusia vaatimuksia myös ohjelmistotuotannolle. Ohjelmistojen tulee olla entistä helpommin ja nopeammin muokattavissa muuttuvaan toimintaympäristöön.

Ennen verkkoliiketoimintaa ohjelmistokehitysprojektit toteuttivat pääasiassa hyvin tunnettuja liiketoimintaprosesseja. Projektit kestivät useista kuukausista jopa vuosiin. Projekti oli onnistunut, mikäli se täytti sille asetetut vaatimukset perusteellisesti. Ohjelmistoprojektien lähtökohtana oli luoda tarkka suunnitelma projektin etenemisestä ja vaatimuksista ennen käytännön toteuttamista. Tämä saavutettiin laatimalla kattava dokumentaatio, jota käytiin läpi useaan kertaan virheitä etsiessä. Laajan ja virheettömän dokumentaation luomiseen kului paljon aikaa. Vaikka tätä mittavaa dokumentaatiota vaativaa menetelmää on käytetty suhteellisen onnistuneesti ohjelmistokehitysprojekteissa, se ei ole sovelias verkkoliiketoiminnan aikakauden kaupallisiin projekteihin, joissa asiat muuttuvat nopeaan tahtiin.

Nykyään projektien tavoitteita ja toteutettavan järjestelmän toiminnallisuuksia joudutaan jatkuvasti muovaamaan vaihtuvien olosuhteiden mukaan kilpailukyvyn ylläpitämiseksi. Laajaa dokumentaatiota suosiva ohjelmistokehitys hidastaa ohjelmistotuotannon projektien etenemistä ja tilanteen ratkaisemiseksi on syntynyt tarve kehittää entistä ’kevyempiä’ lähestymistapoja ohjelmistokehitykseen. Helmikuussa 2001 ohjelmistokehityksen alan metodologistit muodostivat ”Ketterän Allianssin” ja esittivät manifestinsa ohjelmistokehitykselle (Agile Alliance 2001). Allianssin esittelemät ketterät menetelmät ovat muutosherkkiä ja sopeutuvat alati muuttuvaan toimintaympäristöön mahdollistaen projektien tehokkaan toteuttamisen. Ketterä ohjelmistokehitys tarjoaakin elinvoimaisen vaihtoehdon klassiselle ohjelmistokehitykselle.

2 KETTERÄN ALLIANSSIN MANIFESTI

Vuonna 2001 ryhmä teollisuuden ammattilaisia kokoontui luomaan arvoja ja periaatteita parantaankseen ohjelmistokehitystiimien kykyä vastata muutoksiin ja työskennellä nopeasti. He kutsuvat itseään Ketteräksi Allianssiksi (Agile Alliance). Heidän työnsä tuloksena syntyi Ketterän Allianssin manifesti (Kuva 1 Manifesto for Agile Alliance, 2001).

Paljastamme parempia tapoja kehittää ohjelmistoja tekemällä ja auttamalla muita tekemään niitä. Tämän työn kautta olemme oppineet arvostamaan

1. **Yksilöitä ja vuorovaikutusta** mieluummin kuin prosesseja ja työkaluja
2. **Toimivaa ohjelmistoa** mieluummin kuin kattavaa dokumentaatiota
3. **Yhteistyötä asiakkaan kanssa** mieluummin kuin sopimusneuvotteluja
4. **Muutoksiin vastaamista** mieluummin kuin suunnitelman seuraamista

Täten, samalla kun on olemassa arvoa oikeanpuoleisissa asioissa, me arvostamme enemmän asioita vasemmalla

Kuva 1. Ketterän Allianssin Manifesti.

2.1 Manifestin arvot

Manifestin mukaan ihmiset ovat menestyksen tärkein ainesosa. Ketterän allianssin manifestin mukaan *yksilöä ja vuorovaikutusta arvostetaan enemmän kuin prosesseja ja työkaluja*. Hyvä prosessi ei pelasta projektia epäonnistumiselta, jos tiimillä ei ole ammattitaitoisia työntekijöitä; toisaalta huono prosessi voi tehdä jopa parhaimmista työntekijöistä tehottomia. Ryhmä tehokkaita työntekijöitä saattaa epäonnistua, jos he eivät työskentele tiiminä. Ketterä toiminta painottaa ohjelmistokehittäjien keskinäistä suhdetta, yhteisöllisyyttä ja inhimillistä roolia vastakohtana laitosmaisille prosesseille ja

kehitystyökaluille. Tiimien sisäiset suhteet, työympäristön ilmapiiri ja muut tiimihengen nostatuskeinot ovat keskeisessä asemassa ketterissä menetelmissä (Martin 2002).

Ihmisten lisäksi ohjelmistotuotannossa tärkeää on itse tuotos ja tuotteen pääkomponentti, toimiva järjestelmä. Ohjelmistotiimin päämääränä on tuottaa jatkuvasti testattuja ja toimivia ohjelmiston osia. Uusia julkaisuja tuotetaan toistuvin aikavälein, joskus jopa päivittäin tai tunneittain. Kehittäjien tulee pitää ohjelman koodi yksinkertaisena, suoraviivaisena ja teknisesti edistyneenä, jotta dokumentaation tuoma taakka pienenee (Glass 2001). Vähäinen dokumentointi ei ole merkki heikosta ohjelmistosta, vaan pikemminkin päinvastoin. Kutscheran (2002) mukaan laaja dokumentointi ei välttämättä tarkoita sitä, että todelliset syyt ovat täysin ymmärretty. Manifestin mukaan *toimiva ohjelmisto onkin tärkeämpää kuin kattava dokumentaatio*.

Manifestin kolmas arvo on *asiakasyhteistyön korostaminen sopimusneuvottelujen sijaan*. Kehittäjien ja asiakkaiden välinen suhde ja yhteistyö on etusijalla tiukkoihin sopimuksiin nähden, vaikka hyvin asetetuttujen sopimuksien merkitys kasvaa samaa vauhtia kuin ohjelmistoprojektin kokokin. Neuvotteluprosessi itsessään voidaan nähdä apukeinona elintärkeän yhteistyösuhteen saavuttamiselle ja ylläpitämiselle (Martin 2002). Tämän arvon pääsanoma on se, että yhteistyö asiakkaan kanssa on yksi kriittisistä menestystekijöistä ohjelmistoprojekteissa, koska aktiivisen yhteistyön kautta asiakas voi auttaa tiimiä ymmärtämään kehitettävän ohjelmiston toiminnallisia vaatimuksia (Kutschera 2002).

Manifestin mukaan olennaisempaa on *vastata muutoksiin kuin noudattaa orjallisesti alkuperäistä suunnitelmaa*. Kehittämisryhmän (käsittää ohjelmiston kehittäjät ja asiakkaan edustajat) tulisi olla ajantasalla ohjelmiston toteutuksen etenemisestä Sen tulee olla asiantunteva ja kykenevä pohtimaan mahdollisia tarkennustarpeita, joita ilmenee ohjelmistoprosessin elinkaaren aikana. Kehittämisryhmän jäsenet ovat valmiita tekemään muutoksia ja olemassa olevat sopimukset on tehty sellaisella työkalulla, joka tukee ja mahdollistaa muutosten tekemisen (Martin 2002). Muutokseen vastaamiseen sisältyy myös näkökulman, jossa vaatimusten muuttamista ei oteta huomioon niiden projektin aikatauluun sopimattomuuden vuoksi. On selvää, että sellaisen järjestelmän toimittaminen asiakkaalle, joka toteuttaa ei-tärkeitä vaatimuksia, on turhaa. Ratkaisu

ongelmaan on lyhyiden julkaisusyklien käyttö yhdistettynä asiakkaan mahdollisuuteen tuoda uusia vaatimuksia järjestelmälle tai muuttaa aikaisempia (Kutschera 2002).

2.2 Manifestin periaatteet

Edellä esitellystä neljästä arvosta on johdettu 12 Manifesti-periaatetta. Periaatteiden myötä Manifestissä esitetyt arvot saavat ohjeita niiden toteuttamiseksi käytännönnössä (Agile Alliance 2001).

1. Korkein prioriteettimme on tyydyttää asiakkaan tarpeet toimittamalla varhaisessa vaiheessa ja jatkuvin aikavälein toimivia ohjelmistoversioita.

Toimivan ohjelmiston, tai ainakin osan suunnitellusta järjestelmästä, toimittaminen asiakkaalle kehitysprojektin alkuvaiheessa, sekä säännöllisesti sen aikana, tuo monia hyötyjä. Aikainen toimitus mahdollistaa palautteen antamisen kehitettävästä järjestelmästä auttaen kehitysprojektia kulkemaan seuraavissa vaiheissa oikeaan suuntaan. Toisaalta aikainen toimitus kehitettävästä ohjelmistosta vakuuttaa asiakkaan, koska se voivat arvioida rahojensa käyttöä. ”Arvokas” ohjelmisto viittaa ohjelmistoon, tai sen osiin, jotka ovat elintärkeitä asiakkaan liiketoiminnalle ja tämän vuoksi korkealla prioriteetilla mukana kehitysprojektissa (Cockburn 2001). Ketterät prosessit tuottavat julkaisuja aikaisessa vaiheessa ja usein. Prototyyppi rakennetaan projektin ensimmäisten viikkojen aikana. Pyrkimyksenä on julkaista parin viikon välein prototyyppi, jonka toiminnallisuutta on lisätty. Kun asiakkaan mielestä prototyyppi on riittävän toiminnallinen, järjestelmä voidaan laittaa tuotantoon (Martin 2002).

2. Hyväksy vaatimusten muutokset, jopa kehityksen myöhäisissä vaiheissa. Ketterät prosessit valjastavat muutoksen asiakkaan kilpailueduksi.

Kehitettävän ohjelmiston vaatimukset muuttuvat ajoittain jopa radikaalisti projektin edetessä. Edellä esitetyn periaatteen mukaan asiakkaan muuttuvat vaatimukset tulee ottaa huomioon myöhäisessäkin projektin kehitysvaiheessa. Useat ketterät menetelmät tarjoavat erilaisia tapoja reagoida muuttuviin asiakasvaatimuksiin. Iterointi ja ajettavan ohjelmiston toimittaminen aikaisessa vaiheessa ovat esimerkkejä tällaisista tavoista (Cockburn 2001). Ketterien menetelmien tarkoitus on pitää ohjelmiston rakenne

mahdollisimman joustavana, jotta vaatimusten muuttuessa vaikutus järjestelmään olisi minimaalinen (Martin 2002).

3. Julkaise toimiva ohjelmisto useasti, muutamasta viikosta pariin kuukauteen.

Periaate 3 viittaa kehityssyklien pituuteen. Käyttämällä lyhyitä syklejä saadaan useita etuja. Ensinnäkin kehityssyklit, jotka kestävät yli neljä kuukautta vähentävät toimivan prosessin korjaamismahdollisuuksia. Lyhyiden syklien avulla prosessi itsessään tulee testatuksi ja pikaisen palautteen avulla sitä voidaan korjata nopeasti. Toisaalta vaatimustenhallinta tulee tehokkaammaksi, kun tuotteen vaatimukset voidaan testata ja muuttaa nopeasti (Cockburn 2001). Tarkoituksena ei ole julkaista suuria määriä dokumentteja tai suunnitelmia, vaan tähtäimessä on julkaista asiakkaan tarpeet tyydyttävä ohjelmisto (Martin 2002).

4. Liikemiehien ja kehittäjien tulee työskennellä yhdessä päivittäin projektin ajan.

Liikemiehet työskentelevät harvoin kehitysprojekteissa tarpeeksi intensiivisesti, jolloin projektin tulos ei välttämättä vastaa heidän odotuksiaan tai tarpeitaan. Mikäli liikemiehet ja kehittäjät työskentelisivät päivittäin yhdessä projektin ajan kommunikoimalla ja jakamalla ideoita, liikemiehet olisivat tyytyväisempiä projektin tuloksiin. Sanalla ”päivittäin” ei tarkoiteta jokaista päivää vaan mieluummin projektin kohtia, joissa tärkeistä päätöksistä keskustellaan (Cockburn 2001).Ketterä projekti edellyttä jatkuvaa kommunikointia asiakkaiden ja kehittäjien välillä (Martin 2002).

5. Rakenna projektit motivoituneiden ihmisten ympärille. Anna heille heidän tarvitsemansa ympäristö ja tuki ja luota heihin saadaksesi työ tehdyksi.

Ketterät lähestymistavat painottuu enemmän ihmisläheisiin tekijöihin kuten lahjakkuuteen, taitoon ja kommunikaatioon, mieluummin kuin projekti- ja prosessitekijöihin. Yksilöllinen pätevyys on kriittinen tekijä ketterän projektin menestymiselle. Yhteistyö ja kommunikaatio parantavat yksilöllisen osaamisen hyväksikäyttöä ja kehittämistä, koska yksilöt oppivat toinen toisiltaan. He saavat aikaan parempia tuloksia mm. ongelman ratkaisussa työskennellessään ryhmässä (Cockburn 2001).

6. Suorituskykyisin ja tehokkain menetelmä tiedon kuljettamiselle kehitystiimin sisällä on kasvokkain tapahtuva viestintä.

Yhdessä työskenteleville henkilöille tulee antaa mahdollisuus sijoittua lähelle toisiaan luodakseen ympäristön, joka helpottaa vapaamuotoista kasvokkain tapahtuvaa viestintää. Tiimin läheisyys parantaa kommunikaatiota, koska tiimi voi jatkuvasti jakaa informaatiota keskenään (Cockburn 2000). Dokumentteja tuotetaan, mutta niiden tarkoituksena ei ole sisältää kaikkea projektin informaatiota. Oletuksena asioiden hoitamiselle on ihmisten välinen kommunikointi (Martin 2002).

7. Toimiva ohjelmisto on päämittari edistymiselle.

Periaate 7 viittaa toimivan ohjelmiston käyttämiseen kehitysprojektin edistymisen mittana. Myös muita edistymisen mittoja voidaan käyttää, mutta toimiva ohjelmisto saattaa kertoa enemmän kehitysprojektin edistymisen tilasta kuin esimerkiksi raportit (Cockburn 2001). Toimiva ohjelmisto sopii projektin edistymismittariksi kun lasketaan, kuinka monta prosenttia vaadituista toiminnallisuuksista on jo toteutettu (Martin 2002).

8. Ketterät prosessit tukevat kestävää kehitystä. Sponsorien, kehittäjien ja käyttäjien tulisi pystyä pitämään tasainen tahti ennalta määräämättömän ajan.

Periaatteella 8 on kaksi puolta: sosiaalinen vastuu ja projektin tehokkuus. Työntekijöitä ei tulisi pitää ylitöissä liian pitkää aikaa. Väsynyt henkilökunta ei ole ketterä heidän tulostason madaltuessa stressin kasvamisen myötä. Projektit pitää organisoida siten, että jokaisella työntekijällä on kohtuullinen työmäärä. Tämä takaa työntekijöiden sitoutumisen työhönsä johtaen myös parempaan työn tulokseen (Cockburn 2001).

9. Jatkuva huomio tekniseen laadukkuuteen ja hyvään suunnitteluun parantaa ketteryyttä.

Hyvä projektisuunnittelu edistää projektin ketteryyttä. Suunnittelijoiden tulisi tuottaa hyviä ja toimivia suunnitelmia heti projektin alussa. Suunnitelmien jatkuva päivittäminen ja kehittäminen on mahdollista ja toivottavaa myös projektin edetessä (Cockburn 2001). Korkea laatu on avain nopeaan etenemiseen projektissa (Martin 2002).

3 KETTERÄT MENETELMÄT

Ketterän Allianssin piiriin lukeutuu jo useita eri ohjelmistotuotantomenetelmiä. Seuraavassa keskitymme tarkastelemaan kolmea ketterän allianssin periaatteita noudattavaa menetelmää: Extreme Programming, Scrum, Cystal metodologiaperhe. Menetelmät esitellään käymällä läpi niiden rakenne sisältäen prosessit, roolit ja vastuut sekä käytäntöjä. Prosessi viittaa tuotteen elinkaaren vaiheisiin, joita läpikäymällä ohjelmistoja tuotetaan. Roolit ja vastuut ovat työntekijöiden kohdentamista rooleihin, joiden avulla ohjelmistojen tuottaminen tiimissä hoidetaan toimivasti. Käytännöt ovat konkreettisia aktiviteetteja ja työkaluja, jotka menetelmä määrittelee käytettäväksi prosessissa.

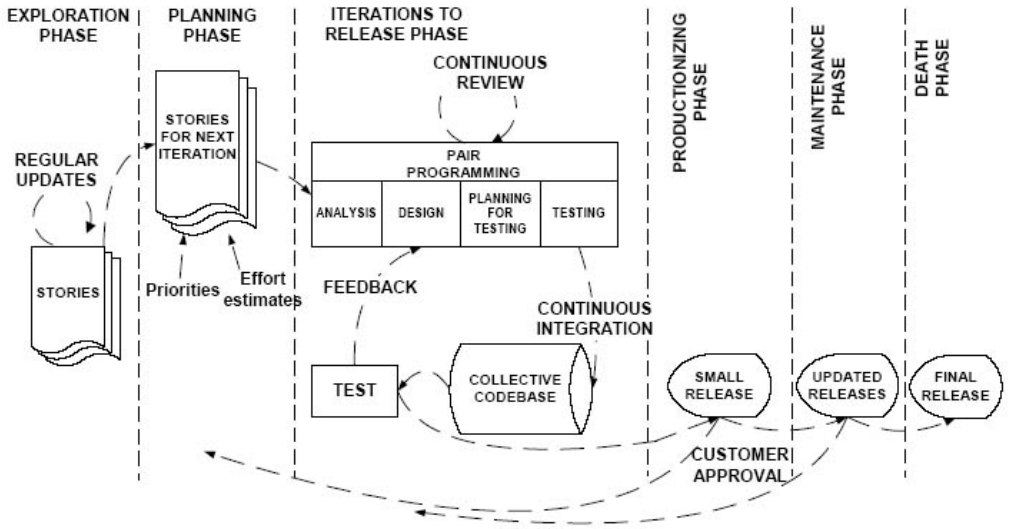
3.1 XP - Extreme Programming

Extreme Programming on ohjelmistokehityksen tieteenala joka perustuu neljään arvoon: yksinkertaisuus, kommunikaatio, palaute ja rohkeus. Se toimii tuomalla koko tiimin yhteen yksinkertaisten käytäntöjen avulla, antamalla tarpeeksi palautetta jotta tiimi voi nähdä missä he ovat menossa ja mukauttaa käytäntönsä heidän ainutlaatuiseen tilanteeseen (Beck, 1999).

Extreme Programming menetelmän (lyh. XP) kehittämisen ideaan on johtanut perinteisten ohjelmistokehitysmenetelmien pitkien kehityssyklien tuomat ongelmat. XP:n kehittäminen alkoi ajatuksesta, että ohjelmistoprojektit saadaan suoritettua loppuun käytännöillä, jotka on todettu tehokkaiksi ohjelmistojen kehitysprosesseissa aikaisempien vuosikymmenien aikana. XP menetelmään on siis kerätty toimivia eri käytäntöjä, joita on hieman muokattu siten että ne toimivat toistensa kanssa ennennäkemättömällä tavalla ja näin muodostaen uuden ohjelmistokehitysmenetelmän (Highsmith 2000).

3.1.1 Prosessi

XP:n elinkaari muodostuu viidestä eri vaiheesta: tutkiminen, suunnittelu, iteraatiot tuotteen julkaisemiseen, tuotteistaminen, ylläpito ja lopetus (Kuva 2). Seuraavana esitellään nämä vaiheet Beck'in (Beck 1999) mukaan:



Kuva 2. XP:n prosessi (Extreme Programming, 2003)

Tutkimisvaiheessa (Exploration phase) (Kuva 2) asiakkaat kirjoittavat ylös erillisille korteille asioita, joita he haluavat sisällyttää ensimmäiseen julkaisuun. Jokainen kortti kuvaa lisättävää ominaisuutta toteuttettavaan ohjelmaan. Projektin työntekijät tutustuvat käytettäviin työkaluihin, teknologiaan ja käytäntöihin. Tutkimisvaiheen kesto vaihtelee muutamasta viikosta muutamaan kuukauteen, riippuen siitä miten tuttu käytetty teknologia on ohjelmoijille.

Suunnitteluvaiheessa (Planning phase) (Kuva 2) ohjelmaan lisättävät ominaisuudet priorisoidaan ja päätetään ensimmäisen julkaisun sisältö. Ominaisuuksien vaativuus arvioidaan ja niiden pohjalta luodaan aikataulu ensimmäistä julkaisua varten. Ensimmäisen julkaisun tuottamiseen kuluva aika ei yleensä ylitä kahta kuukautta. Suunnitteluvaihe kestää yleensä noin pari päivää.

Iteraatiot tuotteen julkaisemiseen (Iterations to release phase) (Kuva 2)**.** Tässä vaiheessa suunnitteluvaiheessa haluttujen lisättävien ominaisuuksien perusteella luotu aikataulu hajotetaan useisiin iteraatioihin, joista jokaisen toteutukseen kuluu aikaa yhdestä neljään viikkoa. Ensimmäinen iteraatio luo koko järjestelmän arkkitehtuurin. Tämä saavutetaan valitsemalla ne asiakkaan määrittelemät ominaisuudet, jotka muodostavat käytettävän järjestelmän koko arkkitehtuurin. Asiakas valitsee kaikki ominaisuudet, jotka sisältyvät jokaiseen yksittäiseen iteraatioon. Iteraation lopussa ajetaan asiakkaan määrittelemät funktionaaliset testit. Viimeisen iteraation jälkeen järjestelmä on valmis tuotteistettavaksi.

Tuotteistamisvaiheessa (Productionizing phase) (Kuva 2) tehdään lisää testauksia ja tarkkaillaan järjestelmän suorituskykyä ennen kuin järjestelmä voidaan antaa asiakkaalle. Tässä vaiheessa voi vielä ilmetä muutoksia ja niistä täytyy tehdä päätös lisätäänkö ne nykyiseen julkaisuun. Tämän vaiheen aikana iteraatioita saatetaan joutua nopeuttamaan esimerkiksi kolmesta viikosta yhteen viikkoon. Ideoita ja ehdotuksia, joita ei tämän takia ehditä käsittelemään, dokumentoidaan, ja toteutetaan esimerkiksi myöhemmin ylläpitovaiheen aikana.

Ylläpitovaihe (Maintenance phase) (Kuva 2) vaatii usein ponnisteluja asiakkaan tukitehtäviin. Tämä siksi, että ensimmäinen julkaisu on tuotteistettu asiakkaan käyttöön ja XP projektin täytyy samanaikaisesti tuottaa uusia iteraatioita ja pitää järjestelmän tuotanto liikkeellä. Vaikka järjestelmän kehittämisvauhti voi hidastua sen ollessa tuotannossa, ylläpito vaihe saattaa silti vaatia uusien työntekijöiden sisällyttämistä tiimiin ja tiimirakenteen muuttamista.

Lopetusvaihe (Death phase) (Kuva 2) on lähellä kun asiakkaalla ei ole enää ominaisuuksia, joita täytyisi toteuttaa järjestelmässä. Tämä vaatii sen, että järjestelmä tyydyttää asiakkaan tarpeet ja takaa riittävän suorituskyvyn ja luotettavuuden. Tässä vaiheessa XP prosessia, kun järjestelmään ei tule muutoksia arkkitehtuuriin, ulkoasuun tai koodiin, tarpeellinen dokumentaatio kirjoitetaan**.**

3.1.2 Roolit ja vastuut

XP menetelmää käyttävän organisaation jäsenillä on erilaisia rooleja eri tehtäviin ja tarkoituksiin menetelmän prosessin ajalle ja sen käytäntöihin. Seuraavassa esitellään nämä roolit Beck'in mukaan (Beck 1999):

Ohjelmoijat kirjoittavat testit ja pitävät ohjelmakoodin niin yksinkertaisena ja tarkoin määrättynä kuin mahdollista. Ensimmäinen ongelma tehdä XP menestykselliseksi on saada kommunikointi ja koordinointi toisten ohjelmoijien ja tiimin jäsenten kanssa toimimaan.

Asiakas kirjoittaa ylös funktionaaliset testit ja järjestelmään haluamansa ominaisuudet, ja päättää milloin jokainen yksittäinen vaatimus on tyydytetty. Asiakas asettaa vaatimuksille toteutusprioriteetit.

Testaajat auttavat asiakkaita kirjoittamaan funktionaaliset testit. He ajavat funktionaaliset testit säännöllisesti, julkaisevat testi tulokset ja ylläpitävät testaustyökaluja.

Jäljittäjä antaa palautetta XP menetelmässä. Hän jäljittää tiimin tekemiä aikatauluja sekä työmääräarviointeja ja antaa palautetta siitä miten tarkkoja he ovat estimoinneissaan, jotta tiimi voi parantaa tulevaisuudessa tulevia estimointeja. Hän myös jäljittää jokaisen iteraation edistymistä sekä määrittää onko päämäärä tavoitettavissa annetuilla resursseilla ja aikaehdoilla tai onko tarpeen tehdä joitain muutoksia itse prosessiin.

Valmentaja on henkilö, joka on vastuussa prosessista kokonaisuudessaan. XP:n hyvä ymmärtäminen on tärkeää tässä roolissa ja se tekee valmentajalle mahdolliseksi ohjata muita tiimin jäseniä seuraamaan prosessia.

Konsultti on ulkopuolinen jäsen, joka omaa erityistä teknistä tietoa. Konsultti ohjaa tiimiä ratkaisemaan heidän erityisongelmansa.

Manageri tekee päätökset asioista. Jotta hän pystyy tekemään päätöksiä, hänen täytyy kommunikoida projektitiimin kanssa määrittääkseen projektin sen hetkisen tilan, ja erottaa kaikki vaikeudet tai vajavaisuudet prosessissa.

3.1.3 Menetelmän käytännöt

XP pyrkii mahdollistamaan menestyksekkään ohjelmistokehityksen pienissä tai keskisuurissa tiimeissä epäselvistä tai jatkuvasti muuttuvista vaatimuksista huolimatta. Lyhyet iteraatiot sisältäen nopean palautteen ja pienet julkaisut, asiakkaan osallistuminen, kommunikointi ja koordinointi, jatkuva integrointi ja testaus, kollektiivinen koodin omistus, rajoitettu dokumentointi ja pariohjelmointi kuuluvat XP:n pääpiirteisiin. Seuraavassa esitellään XP:n käytännöt:

Projektin suunnittelu. Läheistä interaktiota asiakkaan ja ohjelmoijien välillä. Ohjelmoijat arvioivat työtarpeen joka tarvitaan asiakkaan kertomuksien toteuttamiseen. Tämän jälkeen asiakas päättää julkaisujen ajoituksen ja laajuuden (Highsmith 2000).

Pienet/lyhyet julkaisut. Jokaisen julkaisun tulisi olla niin pieni kuin mahdollista, sisältäen vain arvokkaimmat vaatimukset (Beck 1999). Yksinkertainen järjestelmä toteutetaan nopeasti – ainakin kerran joka toinen tai kolmas kuukausi. Täten uusia versioita julkaistaan jopa päivittäin, mutta ainakin kuukausittain (Highsmith 2000).

Metaforat. Järjestelmä määritellään asiakkaan ja ohjelmoijien määrittämällä metaforalla (tai metaforilla). Tämä asiakkaan ja ohjelmoijien yhteinen määritelmä ohjaa kaikkea kehittämistä kuvaamalla kuinka järjestelmä toimii. Metaforat siis auttavat tekemään teknologian enemmän ymmärrettäväksi ihmisen termein, erityisesti asiakkaille (Highsmith 2000).

Yksinkertainen suunnittelu. Painopiste on yksinkertaisimman ratkaisun suunnittelussa, joka voidaan tietyllä hetkelle toteuttaa. Tarpeeton kompleksisuus ja ylimääräinen koodi poistetaan välittömästi kaikista tuotoksista. Yksinkertaisella suunnittelulla on kaksi osaa. Ensinnäkin, suunnittele toiminnallisuudet, jotka on määritelty, älä potentiaalisia tulevaisuuden toiminnallisuuksia. Toiseksi, luo paras suunnitelma, joka voi tuottaa vaaditun toiminnallisuuden (Highsmith 2000).

Testaus. Ohjelmiston kehittäminen on testien ohjaamaa. XP käyttää kahden tyyppistä testausta: yksikkö- ja funktionaalinen testaus. Yksikkötestit toteutetaan ennen koodin kirjoittamista ja niitä ajetaan jatkuvasti. Asiakkaat kirjoittavat järjestelmälle ajettavat funktionaaliset testit (Sharma 2004).

Refaktorointi. Refaktorointi on jatkuvaa ohjelmiston uudelleen suunnittelua, jotta voidaan parantaa sen reaktiokykyä vaatimusten muutoksiin. Järjestelmä refaktoroidaan poistamalla duplikaatiot, kommunikaatiota parantamalla, yksinkertaistamalla ja lisäämällä joustavuutta (Highsmith 2000).

Pariohjelmointi. Kaksi ihmistä kirjoittaa koodia yhdellä tietokoneella. Pariohjelmointi on dialogi kahden ihmisen välillä, jotka yrittävät ohjelmoida yhtä aikaa ja ymmärtää kuinka ohjelmoida paremmin (Beck 1999).

Kollektiivinen omistusoikeus. XP määrittelee kollektiivisen omistusoikeuden menetelmäksi, joss kuka tahansa voi muuttaa mitä tahansa osaa koodista milloin vain haluaa (Highsmith 2000).

Jatkuva integrointi. Uusi koodikokonaisuus liitetään kehitettävään versioon heti kun se on valmis, vaikka järjestelmää integroidaan ja rakennetaan monta kertaa päivässä. Kaikki testit ajetaan ja niiden täytyy mennä läpi, jotta koodiin tehdyt muutokset voidaan hyväksyä (Sharma 2004).

40 tunnin työviikko. Maksimi työmäärä viikolle on 40 tuntia. Kahta ylityöviikkoa peräkkäin ei hyväksytä. Jos näin kuitenkin käy niin sitä ajatellaan ongelmana joka täytyy ratkaista. XP suosittelee, että ihmisten ei tulisi tehdä ylitöitä jatkuvasti. Kun tekee useasti ylitöitä niin tuottavuus laskee ja ei nauti enää työskentelystä. Kehittäjien tulisi tuntea itsensä raikkaaksi tullessaan aamulla töihin ja olla tyytyväinen päivän saavutuksiin lähtiessä kotiin (Sharma 2004).

Paikalla oleva asiakas. XP:n mukaan asiakas on tärkeä ja kehitystiimiin kuuluva osa. XP suosittelee, että asiakas olisi kehitystiimin saatavilla jatkuvasti kirjoittaakseen käyttäjätarinoita (vrt. käyttötapaukset) ja hyväksymistestejä, osallistuakseen julkaisujen ja iterointien suunnitteluun (Sharma 2004).

Ohjelmointistandardit. XP projekteissa ohjelmointistandardit ei käsketä käyttää. Kehitystiimi vapaaehtoisesti omaksuu standardit. Ohjelmakoodin kirjoittamiseen liittyvät säännöt on olemassa ja ohjelmoijat noudattavat niitä. Kommunikointia koodin kautta tulisi painottaa (Sharma 2004).

3.2 SCRUM

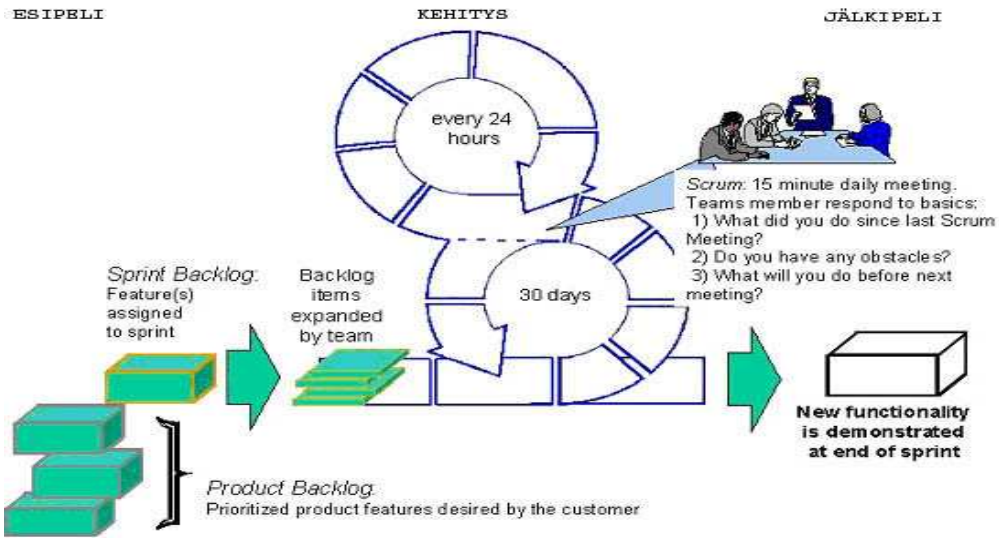
Scrum menetelmä on mukautuva, nopea, itse-organisoituva tuotekehitysprosessi, joka on lähtöisin Japanista. Scrum lähestymistapa on kehitetty järjestelmien kehitysprosessien hallintaan. Se on empiirinen lähestymistapa, joka soveltaa teollisia prosessin kontrollointiteorioita järjestelmien kehittämiseen. Tällöin tuloksena syntyy lähestymistapa, joka ottaa uudelleen käyttöön ajatukset joustavuudesta, sopeutumiskyvystä ja tuotantokyvykkyydestä. Se ei määrittele mitään erityisiä ohjelmistokehitystekniikoita toteutusvaiheeseen. Scrum keskittyy siihen, miten tiimin jäsenten tulisi toimia tuottaakseen järjestelmään joustavuutta jatkuvasti muuttuvassa ympäristössä (SCRUM 2004).

Scrumin perusajatus on se, että järjestelmien kehitys aiheuttaa useita ympäristöllisiä ja teknisiä muuttujia (esim. vaatimukset, aikarajoitukset, resurssit, ja teknologia), jotka todennäköisesti muuttuvat prosessin aikana. Tämä tekee kehittämisprosessista arvaamattoman ja monimutkaisen, ja vaatii järjestemän kehittämisprosessilta joustavuutta pystyäkseen vastaamaan muutoksiin. Kehittämisprosessin tuloksena syntynyt järjestelmä on käyttökelpoinen kun se toimitetaan asiakkaalle (SCRUM 2004).

Scrum auttaa parantamaan olemassa olevia suunnittelukäytäntöjä (esim. testauskäytäntöjä) organisaatiossa, koska se mahdollistaa ajoittaisia hallinnollisia toimintoja, joiden avulla voidaan jatkuvasti identifioida vajavaisuuksia tai haittoja kehittämisprosessissa (SCRUM 2004).

3.2.1 Prosessi

Scrum-prosessi sisältää kolme vaihetta: esipeli, kehitys ja jälkipeli (Kuva 3). Seuraavana esitellään Scrum’in vaiheet:



Kuva 3. Scrum-prosessi (SCRUM 2004)

Esipelivaihe (Kuva 3) sisältää kaksi alavaihetta: Suunnittelu ja Arkkitehtuuri/Korkean tason suunnittelu (Bach 1995).

Suunnittelu sisältää kehitettävän järjestelmän määritelmän. Luodaan tuotteen toteuttamattomien vaatimusten listaus (Kuva 3 Product Backlog), joka sisältää kaikki vaatimukset, jotka on sillä hetkellä tiedossa. Vaatimukset priorisoidaan ja niiden toteuttamiseen vaadittavat työmäärät arvioidaan. Toteuttamattomien vaatimusten listausta päivitetään jatkuvasti uusilla tai yksityiskohtaisemmilla kirjauksilla, sekä tarkemmilla arvioinneilla ja uusilla prioriteettiarvoilla. Suunnittelu sisältää myös projektiryhmän määrittelyn, työkalujen ja muiden resurssien määrittelyn, riskien arvioinnin ja ongelmien kontrolloinnin, koulutustarpeen määrittelyn ja johdon hyväksymisen saannin projektille. Jokaisessa iteraatiossa päivitetty toteuttamattomien vaatimusten listaus käydään läpi Scrum tiimin voimin ja tällä tavoin tiimi saa velvoitteen suorittaa seuraavan iteraation (Bach 1995).

Arkkitehtuurivaiheessa järjestelmän korkean tason suunnitelma, joka sisältää järjestelmän arkkitehtuurin, suunnitellaan sen hetkisen toteuttamattomien vaatimusten listauksen kirjauksien mukaan. Toteuttamattomien vaatimusten listauksen kirjaukset ja niiden mukana mahdollisesti tulevat ongelmat identifioidaan siltä varalta, että sen hetkiseen järjestelmään joudutaan tekemään muutoksia. Tämän jälkeen pidetään

suunnittelun katselmustilaisuus, jossa käydään läpi toteutusehdotukset ja tehdään päätökset toteutuksesta, ja lisäksi alustavat suunnitelmat julkaisujen sisällöistä valmistellaan (Bach 1995).

Kehitysvaihe (Kuva 3) on Scrum lähestymistavan ketterä osuus. Tätä vaihetta pidetään ”mustana laatikkona”, jossa arvaamattomia tapahtumia odotetaan. Erilaiset Scrum-menetelmässä identifioidut ympäristölliset ja tekniset muuttujat (esim. aikaraja, laatu, vaatimukset, resurssit, toteutusteknologiat, työkalut ja jopa kehitysmenetelmät), jotka saattavat muuttua prosessin aikana, huomioidaan ja kontrolloidaan Scrum:in käytäntöjen mukaan kehitysvaiheen ”pikajuoksujen” eli iteraatioiden (Sprints) aikana. Näitä asioita ei oteta huomioon vain ohjelmistoprojektin alussa vaan Scrum pyrkii kontrolloimaan niitä jatkuvasti ja näin olemaan joustava menetelmä, joka mukautuu muutoksiin (Bach 1995).

Tuotettava järjestelmä kehitetään pikajuoksuissa. Pikajuoksut ovat iteratiivisia kierroksia (Kuva 3), joissa järjestelmän toiminnallisuus kehitetään. Jokainen pikajuoksu sisältää perinteisen ohjelmistokehityksen vaiheet: vaatimusmäärittelyn, analyysin, suunnittelun, tuotteen kehitysvaiheen, testauksen ja tuotteen julkaisun. Järjestelmän arkkitehtuuri kehittyy juuri pikajuoksu-kierrosten aikana. Yhden pikajuoksun kesto on keskimäärin noin yhdestä viikosta kuukauteen (Bach 1995).

Jälkipelivaiheessa (Kuva 3) tuote on valmis julkaistavaksi. Tähän vaiheeseen tullaan kun esimerkiksi ympäristölliset muuttujat, kuten vaatimukset, ovat toteutettu. Eli enää ei löytää kirjauksia tai ongelmia toteuttamattomien vaatimusten listauksesta, eikä myöskään keksiä uusia sellaisia. Valmistelut julkaisua varten, kuten järjestelmän integrointi, testaus ja dokumentointi, tehdään juuri tässä vaiheessa (Bach 1995).

3.2.2 Roolit ja vastuut

Scrum-menetelmästä voidaan identifioida projektiryhmän jäsenille kuusi eri roolia. Seuraavassa esitellään nämä roolit (SCRUM 2004):

Scrum-mestari on vastuussa siitä, että projekti viedään läpi Scrum-menetelmän käytäntöjen, arvojen ja sääntöjen mukaan, ja että projekti etenee suunnitelmien mukaan.

Scrum-mestari toimii projektin aikana yhdessä projektiryhmän, asiakkaan ja johdon kanssa. Hän on myös pitää huolta siitä, että prosessin vajaavaisuudet poistetaan, jotta projektiryhmä voisi saada mahdollisimman hyviä tuloksia aikaan.

Tuotteen omistaja on vastuussa projektista, johtamisesta, sekä toteuttamattomien vaatimusten listauksen sisällön kontrolloinnista. Scrum-mestari, asiakas ja johto valitsevat tähän tehtävään kuuluvan henkilön. Hän arvioi tuotteen toteuttamattomien vaatimusten listaukseen (katso 3.2.3 Menetelmän käytännöt) tehtyjen kirjausten kehittämiseen tarvittavia työmääriä ja pitää huolta, että kaikki kirjaukset tullaan kehittämään ajallaan.

Scrum-tiimi on projektiryhmä, jolla on oikeudet päättää tarvittavista toimista ja organisoida itseään saavuttaakseen määritetyt tavoitteet jokaisessa pikajuoksussa. Scrum-tiimi joutuu tekemään työmääräarviointeja, luomaan pikajuoksun toteuttamattomien vaatimusten listauksen (katso 3.2.3 Menetelmän käytännöt), katselmoimaan tuotteen toteuttamattomien vaatimusten listausta, sekä tuo esille asioita, joista on haittaa prosessin kululle.

Asiakas osallistuu tehtäviin, jotka liittyvät tuotteen toteuttamattomien vaatimusten listaukseen kirjattuihin vaatimuksiin (katso 3.2.3 Menetelmän käytännöt).

Johto vastaa lopullisesta päätöksenteosta, sekä projektiin liittyvistä perussäännöistä, standardeista ja sopimuksista, joita projektissa seurataan. Johto osallistuu myös projektin tavoitteiden ja vaatimusten asettamiseen.

3.2.3 Menetelmän käytännöt

Scrum ei vaadi mitään erityisiä ohjelmistokehitysmetodeja joita pitää käyttää. Sen sijaan se vaatii tiettyjä johtamiskäytäntöjä ja –työkaluja menetelmän eri vaiheisiin, jotta voidaan välttää arvaamattomuuden ja kompleksisuuden aiheuttamaa kaaosta (SCRUM 2004).

Seuraavassa esitellään Scrum-menetelmän käytännöt Bach’in mukaan (Bach 1995):

Tuotteen toteuttamattomien vaatimusten listaus (Product Backlog). Määrittää kaiken mitä tarvitaan lopulliseen tuotteeseen perustuen sen hetkiseen tietämykseen. Se sisältää priorisoidun ja jatkuvasti päivitetyn listan teknisistä ja ja liiketoiminnallisista vaatimuksista, jotka tulevat rakennettavaan järjestelmään. Toteuttamattomat vaatimuskirjaukset voivat sisältää esimerkiksi rakennettavan järjestelmän piirteitä, toiminnallisuuksia, heikkouksia ja parannusehdotuksia. Tämä käytäntö sisältää tehtävät tuotteen toteuttamattomien vaatimusten listauksen luomiseen ja sen kontrollointiin prosessin aikana lisäämällä, poistamalla, päivittämällä ja priorisoimalla toteuttamattomien vaatimusten listauksen kirjauksia.

Työmäärän arviointi. Tuotteen omistaja ja Scrum-tiimi yhdessä ovat vastuussa työmäärän arvionnista. Se on iteratiivinen prosessi, jossa toteuttamattomien vaatimusten kirjausten arviointeja tarkastellaan tarkemmalla tasolla, kun toteuttamattomien vaatimusten kirjauksista on olemassa enemmän tietoa. Tällä tavoin voidaan arvioida työmäärää mahdollisimman tarkasti.

Pikajuoksu. Pikajuoksu (Sprint) on menettelytapa jolla voidaan mukautua muuttuviin ympäristötekijöihin (vaatimukset, aika, resurssit jne). Scrum-tiimi organisoi itsensä tuottaakseen uuden ajettavan tuoteinkrementin sprintissä, joka kestää noin kuukauden. Pikajuoksun työkaluja ovat pikajuoksun suunnittelutapaamiset, pikajuoksun toteuttamattomien vaatimusten listaus ja päivittäiset Scrum-tapaamiset.

Pikajuoksun suunnittelutapaaminen. Kaksivaiheinen tapaaminen, jonka Scrum-mestari organisoi. Osallistuvia osapuolia ovat johto, käyttäjät, asiakkaat, tuotteen omistaja ja Scrum-tiimi.

Pikajuoksun toteuttamattomat vaatimukset (Sprint Backlog). Pikajuoksun toteuttamattonien vaatimusten listaus on jokaisen pikajuoksun aloituksen pohja. Se on lista tuotteen toteuttamattomien vaatimusten listauksesta valittuja kirjauksia, jotka toteutetaan seuraavassa pikajuoksussa. Kirjaukset valitaan prioriteettien mukaan ja tavoitteet asetetaan seuraavalle pikajuoksulle pikajuoksun suunnittelutapaamisessa Scrum-tiimin, Scrum-mestarin ja tuotteen omistajan yhteistyön voimin. Kun kaikki pikajuoksun toteuttamattomat vaatimuskirjaukset ovat toteutettu, uusi iteraatio järjestelmästä toimitetaan.

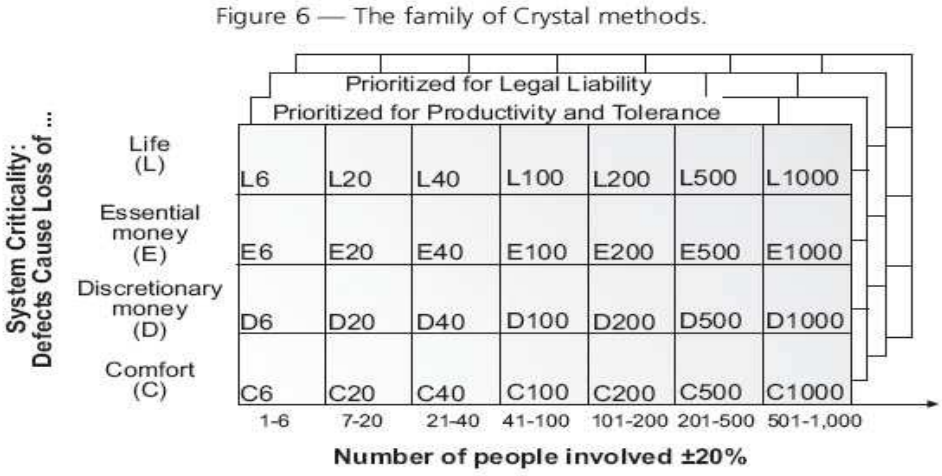
Päivittäinen Scrum-tapaaminen. Järjestetään, jotta voidaan jatkuvasti seurata Scrum-tiimin edistymistä. Ne myös toimivat suunnittelutapaamisina eli voidaan tarkastella mitä on tehty edellisen tapaamisen jälkeen ja mitä pitää tehdä ennen seuraavaa tapaamista. Tapaamisen aikana keskustellaan löytyykö kehitysprosessissa vajeita tai haittoja, löytyessään ne identifioidaan ja poistetaan jotta voidaan parantaa prosessia. Scrum-mestari johtaa Scrum-tapaamiset.

Pikajuoksun katselmointitapaaminen. Pikajuoksun päätyttyä, Scrum-tiimi ja Scrum-mestari esittävät pikajuoksun tulokset asiakkaille, johdolle, käyttäjille ja tuotteen omistajalle epävirallisessa tapaamisessa. Tapaamisen osanottajat vahvistavat tuoteinkrementin ja tekevät päätökset tulevista tehtävistä. Katselmointitapaaminen saattaa tuoda esille uusia toteuttamattomat vaatimukset kirjauksia.

3.3 Crystal metodologiaperhe

Crystal'in metodologiaperhe (Crystal family of methodologies) sisältää useita erilaisia metodologioita sopivimman metodologian valitsemiseen jokaiselle yksilölliselle projektille. Crystal-lähestymistapa sisältää myös periaatteita joilla voidaan räätälöidä metodologiat sopimaan erilaisten projektien vaihteleviin olosuhteisiin (Cockburn 2001).

Crystal-lähestymistavassa valitaan sopivan metodologian väri taulukosta perustuen projektin kokoon ja sen kriittisyyteen (Kuva 4).



Kuva 4. Crystal metodologioiden jaottelu (Highsmith 2000)

Jokainen Crystal'in metodologioista omaa värin joka kuvaa metodologian "raskautta". Ensinnäkin suuremmat projektit saattavat tarvita enemmän koordinointia ja raskaampia metodologioita, toisaalta mitä kriittisempää järjestelmää kehitetään sitä enemmän tarkkuutta vaaditaan. Kuvassa olevat kirjain symbolit viittaavat järjestelmän häiriön mahdollisesti aiheuttamaa liiketappiota tai vahinkoa. Nämä edellä mainitut kriittisyystasot ovat C (mukavuuden menetys), D (harkinnanvaraisen rahan menetys), E (elintärkeän rahan menetys) ja L (elämän menetys). Eli tasolla C järjestelmän kaatuminen aiheuttaa mukavuuden menettämisen ja L-tasolla se saattaa johtaa jopa ihmishenkien menettämiseen. Luvut kriittisyystason jälkeen ilmaisevat sen kuinka monta työntekijää kehitysprojektiin osallistuu (Cockburn 2001).

Kaikilla Crystal'in metodologioilla on tiettyjä sääntöjä, piirteitä ja arvoja jotka ovat samoja: projektit käyttävät aina inkrementaalisia kehittämissyklejä (kesto maksimissaan 4 kk), painopiste on työntekijöiden kommunikoinnissa ja yhteistyössä, sekä Crystal'in metodologiat eivät rajoita kehityskäytäntöjen tai työkalujen käyttöönottoa. Täten se mahdollistaa vaikka XP:n tai Scrumin käytäntöjen käyttöönoton. Nykyään on olemassa kolme kontruoitua Crystal'in päämetodologiaa: Crystal-kirkas (vrt. Crystal Clear), Crystal-oranssi (vrt. Cystal Orange) ja Crystal-oranssi web (vrt. Crystal Orange Web) (Cockburn 2001).

3.3.1 Prosessi

Kaikki Crystal'in metodologiat antavat suuntaviivat toimintaperiaatteiden standardeille, työn tuloksille, projektikohtaisille tehtäville, työkaluille ja rooleille, joita voidaan seurata kehitysprosessin aikana.

Crystal-kirkas on suunniteltu käytettäväksi pienissä projekteissa (esim. D6-luokan projektit), jotka koostuvat enintään kuudesta työntekijästä. Metodologiaa voidaan käyttää myös hieman isompiin projekteihin (esim. E8-luokan projektit) jos kommunikaatiota ja testausta laajennetaan. Crystal-kirkas metodologiaa käyttävän projektin tulisi sen kommunikointirakenteen rajoitusten takia sijoittua yhteen jaettuun työskentelytilaan, jotta projektin sisäinen kommunikointi helpottuisi (Cockburn 2001).

Crystal-oranssi on suunniteltu käytettäväksi keskikokoisissa projekteissa (D40-luokan projektit), jotka koostuvat 10-40 työntekijästä. Myös tätä metodologiaa voidaan käyttää hieman suuremmissa projekteissa (esim. E50) jos sen verifiointi-testauksen prosesseihin tehdään lisäyksiä (Cockburn 2001).

Seuraavassa käsitellään Crystal-kirkkaan ja Crystal-oranssin eroja ja yhtäläisyyksiä Crystal-metodologioiden eri elementtien suhteen.

Toimintaperiaate standardit. Toimintaperiaate standardit ovat käytäntöjä joita täytyy soveltaa kehittämisprosessin aikana. Kummallakin metodologialla (kirkas ja oranssi) on seuraavia toimintaperiaate standardeja (Cockburn 2001):

- Inkrementaalista tuotteen toimittamista säännöllisin aikavälein
- Edistymisen seuraamista ”kilometripylväiden” avulla, jotka perustuvat ohjelmistojen toimituksiin, sekä tehdään mieluummin isoja päätöksiä kuin kirjoitettuja dokumentteja
- Käyttäjän suora mukanaolo toiminnassa
- Toiminnallisuuden automatisoitu regressiotestaus
- Käyttäjä tarkastelee tuotetta kaksi kertaa jokaista julkaisua kohden
- Tuotteen tai metodologian säätäminen jokaisen inkrementin alussa sekä sen puolivälissä

Erona näillä metodologioilla on toimintaperiaatteiden standardeissa sen suhteen, että Crystal-oranssissa inkrementtien toimittaminen asiakkaalle voi tapahtua maksimissaan neljän kuukauden välein, kuten taas Crystal-kirkkaassa maksimissaan kolmen kuukauden välein. Crystal-metodologioiden toimintaperiaate standardit ovat pakollisia, mutta ne voidaan kuitenkin vaihtaa toiseen vastaavaan menetelmään kuten XP tai Scrum (Cockburn 2001).

Työn tulokset. Crystal-kirkkaan ja Crystal-oranssin vaatimukset työn tuloksille vaihtelevat hieman. Molempien metodologioiden vastaavanlaiset työn tulokset kuitenkin ovat: julkaisu sekvenssit, komponenttimallit, käyttöohje, testitapaukset ja siirtymiskoodit (migration code) (Cockburn 2001).

Projektikohtaiset tehtävät. Projektikohtaiset tehtävät (Local matters) ovat Crystal'in menettelytapoja joita tulee soveltaa, mutta niiden toteuttaminen jätetään projektille itselleen. Crystal-kirkkaan ja Crystal-oranssin projektikohtaiset tehtävät eivät poikkea paljoakaan toisistaan. Molemmat metodologiat kuitenkin ehdottavat, että työn tuloksille, ohjelmoinnille, testaamiselle ja rajapintojen standardeille tulisi asettaa mallit (esim. kaaviomallit, ohjelmointityyli jne.), joita projekti ryhmä automaattisesti noudattaa ja ylläpitää (Cockburn 2001).

Työkalut. Crystal-kirkas vaatii käytettäviksi työkaluiksi kääntäjän, version- ja konfiguraationhallintatyökalut, ja videotykin esityksiä ja havainnollistamista varten. Crystal-oranssin vaatii minimissään käytettäväksi työkaluja, joita voidaan käyttää versiointiin, ohjelmointiin, testaamiseen, kommunikointiin, projektin tilan seuraamiseen, piirtämiseen ja suorituskyvyn mittaamiseen (Cockburn 2001).

Standardit. Crystal-oranssi suosittelee, että valitaan notaatiostandardit, suunnittelukonventiot, muotoilustandardit ja laatustandardit, joita käytetään ja noudatetaan koko projektin ajan (Cockburn 2001).

3.3.2 Roolit ja vastuut

Crystal-kirkkaan päärooleja ovat: sponsori, vanhempi suunnittelija-ohjelmoija, suunnittelija-ohjelmoija ja käyttäjä. Nämä roolit sisältävät myös alarooleja. Tällaisia alarooleja joita voidaan jollekin tietylle roolille antaa ovat koordinaattori, liiketoimintaekspertti ja vaatimusten kerääjä (Cockburn 2001).

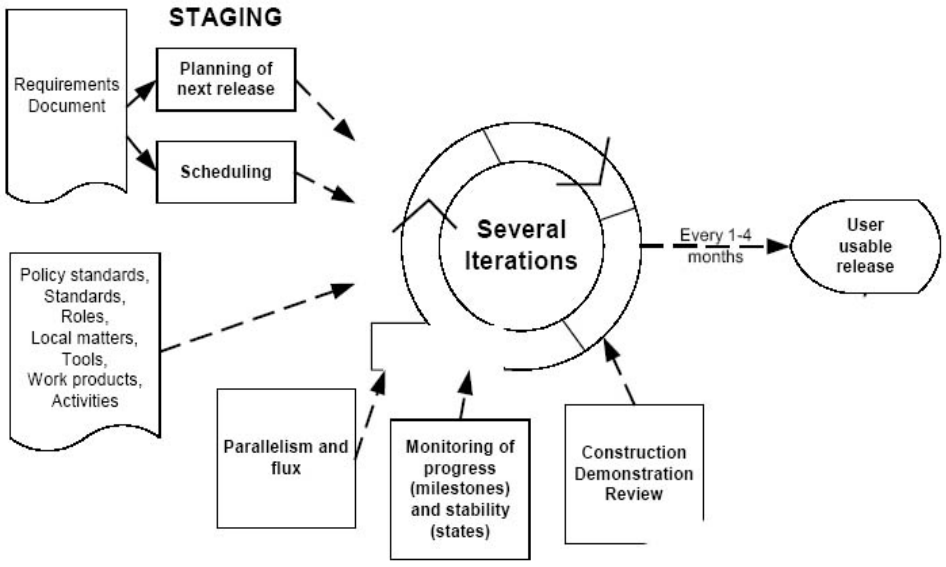
Crystal-kirkkaan roolien lisäksi Crystal-oranssi sisältää suuren joukon päärooleja joita tarvitaan projektissa. Roolit on ryhmitelty tiimeihin kuten järjestelmän suunnittelu-, projektinhallinta-, arkkitehtuuri-, teknologia-, toiminto-, infrastruktuuri- ja ulkoinen

testaus tiimeihin. Crystal-oranssi esittelee myös uusia rooleja kuten käyttöliittymä suunnittelija, tietokantasuunnittelija, arkkitehti ja testaaja (Cockburn 2001).

Perusero Crystal-kirkkaan ja Crystal-oranssin välillä on se, että aiemmassa työskentelee vain yksi tiimi koko projektissa ja jälkimmäisessä on useampia tiimejä, jotka muodostavat erilaisia toimintakokonaisuuksia.

3.3.3 Menetelmän käytännöt

Seuraavassa esitellään käytäntöjä, jotka kuuluvat Crystal-kirkas ja Crystal-oranssi metodologioihin.



Kuva 5. Crystal-kirkkaan ja Crystal-oranssin käytännöt (Cockburn 2001)

Vaiheistaminen. Vaiheistaminen (Staging) (Kuva 5) sisältää järjestelmän seuraavan inkrementin suunnittelun. Se pitäisi aikatauluttaa siten, että voidaan tuottaa toimiva julkaisu joka kolmas tai neljäs kuukausi. Kehitystiimi valitsee toteutettavat vaatimukset seuraavaan inkrementtiin. Valinta tehdään kuitenkin siten, että pysytään toivotussa aikataulussa (Cockburn 2001).

Tarkastus ja katselmointi. Jokainen inkrementti sisältää useita iteraatioita. Jokainen iteraatio taas sisältää seuraavat aktiviteetit: inkrementin päämäärien rakentaminen, esittäminen ja katselmointi (Kuva 5) (Cockburn 2001).

Monitorointi. Projektin edistymistä monitoroidaan tiimin tuotoksien suhteen kehittämisprosessin aikana (Kuva 5). Projektin monitorointia vaaditaan Crystal-kirkas ja Crystal-oranssi metodologioissa (Cockburn 2001).

Metodologian säätämistekniikka. Metodologian säätämistekniikka on yksi Crystal-kirkkaan ja Crystal-oranssin perustekniikoista. Siinä käytetään hyväksi projektihaastatteluita ja aivoriihiä, jotta saadaan tuotettua erityinen Crystal-metodologia jokaista yksilöllistä projektia varten (Cockburn 2001).

Käyttäjän tuotetarkastelut. Crystal-kirkkaassa suositellaan, että käyttäjä tarkastelee kehitettävää tuotetta kaksi kertaa yhden inkrementin aikana. Crystal-oranssissa käyttäjän tuotetarkastelut pitäisi järjestää kolme kertaa yhden inkrementin aikana (Cockburn 2001).

Reflektointitapaamiset. Crystal-kirkas ja Crystal-oranssi sisältävät säännön, jonka mukaan kehitystiimin tulisi pitää reflektointitapaamisia jokaisen inkrementin ennen ja jälkeen. Tarvittaessa niitä voidaan järjestää edellisten lisäksi myös inkrementtien puoliväleissä (Cockburn 2001).

4 YHTEENVETO

Ketterät ohjelmistokehitys metodit tulivat esiin virallisesti ketterän manifestin muodossa vuonna 2001. Yrityksenä oli tuoda tietoa paradigmojen (vrt. ajattelumallien) muutoksista ohjelmistotuotannon alalla. Ketterät menetelmät väittävät asettavan enemmän painoa ihmisiin, kommunikointiin, toimivaan ohjelmistoon ja muutoksiin.

Tällä tutkimuksella pyrittiin kahteen asiaan. Ensinnäkin tutkimuksen tarkoituksena oli koota olemassa olevasta kirjallisuudesta kattava kuvaus sille mitä termillä ”ketterä” tarkoitetaan ja mikä tekee ohjelmistokehitysmetodista ketterän. Menetelmä on ketterä, kun ohjelmistokehitys on:

1. Inkrementaalista (lyhyet kehityssyklit ja pienet julkaisut)
2. Yhteistyöhaluista (asiakkaat ja kehittäjät työskentelevät yhdessä voimakkaasti kommunikoiden)
3. Mukautuvaa (mahdollista tehdä viime hetken muutoksia)
4. Suoraviivaista (menetelmä on helppo oppia ja mahdollistaa modifioinnin erityistarpeisiin)

Toiseksi tarkoituksena oli käydä läpi olemassa olevia menetelmiä ja tarkastelemalla mitä ominaisuuksia eri ketterillä menetelmillä on ja millaisiin projekteihin ne ovat soveliaita.

XP-menetelmän pääarvoja ovat pienet kehitystiimit, asiakaslähtöinen kehittäminen ja päivittäiset ajettavien järjestelmä-versioiden rakentaminen. Refaktorointi on yksi XP:n tärkeistä piirteistä. Jatkuvalla kehitettävän järjestelmän uudelleensuunnittelulla pyritään parantamaan sen suorituskykyä ja kykyä muuntautua vaatimusten muuttuessa.

Beck’in (1999) mukaan XP-menetelmä ei ole sovelias kaikkiin tapauksiin eikä sen kaikkia rajoitteita ole vielä identifioitu. XP on tarkoitettu pienille tai keskisuurille tiimeille (3-20 työntekijää). Sen vahvuusaluetta on projektit joissa muutoksia tulee runsaasti projektin edetessä. Menetelmässä on myös vahva painotus asiakkaan näkökulmaan ja kaikki tehdään asiakkaan ehdoilla. Korkea laatu (eräs

ohjelmistotuotannon nykyongelmista) ei periaatteessa ole projektissa muuttuva parametri, vaan siihen pyritään aina. Myös ympäristö, jossa menetelmää käytetään on tärkeä. Täten projektin työntekijöiden tulisi sijaita samassa tilassa, koska se mahdollistaa hyvän projektin jäsenten välisen kommunikoinnin ja koordinoinnin.

Extreme Programming -prosessi olettaa, että projektin vaatimukset kehitetään projektin edetessä, mistä merkkinä on vahvasti iteratiivinen menetelmä sekä aikataulujen muotoutuminen projektin edetessä. Lähtökohtaisesti oletetaankin, että joko aikataulut tai toiminnallisuus joutuvat joustamaan. Jos halutaan pyrkiä tiettyyn toiminnallisuuteen tietyssä aikataulussa, voi XP siis olla väärä valinta.

Scrum-menetelmän pääarvoja ovat 30:n päivän julkaisusyklit ja pienet, itsenäiset, itseorganisoituvat kehitystiimit. Eräs Scrum'in erikoisominaisuuksista on se, että sen avulla voidaan pakottaa paradigma (vrt. ajattelumalli) muuttumaan määritellystä ja toistettavasta uudeksi Scrum'in tuotteenkehittämisnäkökulmaksi. Scrum-menetelmä kuvaa tarkoin kuinka kehitystiimi suoriutuu 30:n päivän julkaisusykien tuomista vaatimuksista, mutta integrointi- ja hyväksymistestejä ei ole esitetty tarkasti.

Scrum-menetelmää voidaan käyttää projekteihin, joissa on yksi tai useampi pieniä tiimejä, joiden työntekijä määrä on vähemmän kuin 10 ihmistä. Bach'in (2002) mukaan tiimien tulisi koostua 5-9 työntekijästä ja jos työntekijöitä on enemmän tarjolla niin tulisi muodostaa useampia tiimejä.

Crystal-metodologiaperhe on metodologiaperhe, jonka jokaisella yksittäisellä metodologialla on samat perustana olevat pääarvot ja periaatteet. Eri metodologioiden välillä tekniikat, roolit, työkalut ja standardit voivat vaihdella. Metodologiat tarjoavat menetelmänsuunnitteluperiaatteita, jotka mahdollistavat soveliaimman menetelmän valitsemisen, perustuen projektin kokoon ja kriittisyyteen.

Crystal'in metodologiaperhe ei tarjoa tällä hetkellä metodologiaa, joka kattaisi kriittisyystason L projekteja (järjestelmävirheen sattuessa mahdollisuus menettää ihmishenkiä). Cockburn'in (2001) mukaan metodologiaperheen käytön rajoitteena on myös se, että sillä voidaan ohjailla vain samaan tilaan sijoittuneita tiimejä. Crystal-kirkas metodologiassa on melko rajoitettu kommunikaatiorakenne ja täten se on

sovellettavissa yksittäiseen tiimiin joka sijaitsee samassa työskentelytilassa. Crystal-oranssi vaatii myös että sen tiimien tulisi samassa tilassa. Menetelmä kattaa maksimissaan kriittisyystason D projektit ja on sovellettavissa projekteihin joissa työskenetelee enintään 40 työntekijää.

Ideat, joita ketterät menetelmät esittelevät eivät ole uusia, eivätkä menetelmien luojat niin väitäkään. Ne eivät yritäkään ratkaista kaikkia olemassa olevia ongelmia. On kuitenkin olemassa uskomuksia siitä, että ketterät ohjelmistokehitysmetodit tarjoavat ennennäkemättömän tavan lähestyä ohjelmistokehityksen ongelmia.

LÄHTEET

Agile Alliance: Manifesto for Agile Software Development [online], 2001. Saatavilla www-muodossa <http://www.agilealliance.org>.

Bach, James 1995, SCRUM Software Development Process, American Programmer.

Beck, Kent 1999, Extreme Programming Explained, Addison Wesley Professional.

Cockburn, Alistair 2001, Agile Software Development, Agile Software Development Series.

Extreme Programming: A Gentle Introduction [online], 2003. Saatavilla www-muodossa <http://www.extremeprogramming.org>.

Glass, Robert L. 2001, Agile versus traditional: Make love not war!, Cutter IT Journal.

Highsmith, Jim 2000, Extreme programming, Cutter Consortium.

Kutschera, Peter 2001, Applying Agile Methods in Changing Environments.

Martin, Robert C. 2002, Agile Software Development: Principles, Patterns, and Practices, Prentice Hall.

SCRUM [online], 2004. Saatavilla www-muodossa <http://www.controlchaos.com>.

Sharma, Pradyumn 2004, An Introduction to Extreme Programming, Development Advisor.