Ohjelmistoprosessi aloittavassa ohjelmistoyrityksessä

Mika Kuikka

24.5.2008

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Ohjelmistoprosessi määrittää, minkä vaiheiden ja tehtävien kautta projekti tuottaa ideasta valmiin ja testatun ohjelmiston. Se voi pohjautua johonkin valmiiseen prosessimalliin, joita löytyy useita. Tässä tutkielmassa esitellään muutamia yleisesti käytettyjä prosessimalleja, sekä niiden arvioinnissa ja kehittämisessä käytettäviä arviointikehyksiä ja standardeja. Kunkin prosessimallin osalta käsitellään sen vahvuuksia ja heikkouksia, ajatellen pieniä aloittavia ohjelmistoyrityksiä. Johtuen aloittavien ohjelmistoyritysten erityistilanteesta, jossa ensimmäisien asiakkaiden vakuuttamiseksi tarvittavan työn määrä on merkittävä, ja jokainen päivä on selviytymistaistelua, on myös aloittavilla ohjelmistoyrityksillä järkevää olla määritetty ja dokumentoitu ohjelmistoprosessi. Arviointikehykset, standardit ja osa valmiista prosessimalleista ovat pienille organisaatioille aivan liian raskaita, joten ketterämpien prosessimallien käyttöä kannattaa harkita. Arviointikehyksiä kannattaa kuitenkin käyttää yksittäisten olemassa olevien prosessien kehittämiseen ja parantamiseen. Tutkielman lopussa kuvataan todellisessa aloittavassa ohjelmistoyrityksessä kehitetty suunnitelmaohjautuvan ja ketterän prosessimallin yhdistelmä.

ACM-luokat (ACM Computing Classification System, 1998 version): D.2.9, K.6.3

Sisältö

1 Johdanto

Ohjelmistojen kehittäminen tapahtuu ohjelmistoprosessin kautta. Se määrittää, minkä vaiheiden ja tehtävien kautta projekti tuottaa ideasta valmiin ja testatun ohjelmiston. Yrityksen ohjelmistoprosessi voi olla mitä tahansa väljän epämuodollisen, ja standardiin pohjautuvan, tiukasti määritetyn prosessin välillä. Ohjelmistoprosessit noudattavat jotakin ohjelmistoprosessimallia. Käytettävä prosessimalli voi pohjautua johonkin valmiiseen ohjelmistoprosessimalliin, joita on olemassa valmiina useita. Boehmin ja Turnerin (2003) mukaan ne voidaan jakaa karkeasti kahteen ryhmään, suunnitelmaohjautuviin ja ketteriin prosessimalleihin. Näiden lisäksi prosessimalli voi olla sekoitus kummastakin tyypistä, jolloin päästään usein hyvään lopputulokseen.

Suunnitelmaohjautuvat prosessimallit, kuten vesiputousmalli, ovat perinteisiä prosessimalleja. Niissä ohjelmiston elinkaari jaetaan lineaarisesti peräkkäisiin vaiheisiin, jossa edellisen vaiheen valmistuminen tuottaa seuraavan vaiheen syötteet, ja prosessi siirtyy seuraavaan vaiheeseen. Ominaista suunnitelmaohjautuville prosessimalleille on kattava dokumentaatio sekä vahva etupainotteinen suunnittelu, eli järjestelmän ominaisuudet pyritään määrittämään täydellisesti ennen toteuttamista. Tämä ei useinkaan ole mahdollista, koska asiakas ei välttämättä edes tiedä kaikkia vaatimuksia projektin alussa (Morien, 2005). Suunnitelmaohjautuvien prosessimallien jäykkyyttä ja raskautta lisää muutosten vaikutusten kertautuminen. Kun vaatimuksiin tulee prosessin aikana muutoksia, joudutaan prosessissa palaamaan takaisin vaiheeseen, johon muutos kohdistuu. Vaiheiden lopussa olevien laadunvarmistusten kanssa tämä prosessi tulee yleensä hyvin kalliiksi, ja pahimmillaan venyttää projektin valmistumista.

Vastapainoksi raskaille ja jäykille suunnitelmaohjautuville prosessimalleille on kehitetty joukko ketteriä prosessimalleja, joita kutsutaan myös ketteriksi ohjelmistokehitysmenetelmiksi (agile methods). Tällaisia ketteriä prosessimalleja ovat esimerkiksi tässä tutkielmassa kuvatut Scrum ja XP. Ketterät mallit muodostuvat yleensä joukosta ohjelmistotuotannon parhaita käytäntöjä ja ovat luonteeltaan hyvin joustavia sekä muutokseen paremmin sopeutuvia (Boehm & Turner, 2003). Yleensä ketterissä malleissa ohjelmisto kehitetään lyhyissä iteraatioissa, inkrementaalisesti. Tämä tarkoittaa,

että ohjelmiston kehitys jaetaan iteraatioihin, eli lyhyisiin kehityssykleihin, joita voidaan ajatella toteutuksen aliprojekteina. Jokaisen iteraation alussa valitaan iteraatiossa toteutettavat ominaisuudet ja lopussa julkaistaan asiakkaalle versio, jossa on kaikki iteraatioon valitut toiminnallisuudet ja ominaisuudet, testattuna ja toimivana. Seuraavaan julkaisuun valitut ominaisuudet toteutetaan aina edellisen julkaisun päälle, jolloin kehitys on inkrementaalista (Koch, 2005). Ketterien mallien vahvuutena voidaan pitää parempaa sopeutuvuutta muutokseen ja nopeampia julkaisuja, mutta toisaalta heikkoutena korkean tason määrittely- ja arkkitehtuuridokumentaation puutetta. Korkean tason dokumentaation puute johtaa helposti siihen, ettei kenelläkään ole kokonaiskuvaa kehitettävästä järjestelmästä (Müller & Padberg, 2003).

Ohjelmistoprosesseja voidaan arvioida ja kehittää erilaisten arviointikehyksien avulla. Tällaisia arviointikehyksiä ovat esimerkiksi ISO/IEC 15504 (SPICE) ja CMMI, sekä yleisesti käytetty laatustandardi ISO 9001:2000 ja sen ohjelmistotuotannon sovitusohjeisto ISO 9000-3. Arviointikehyksien avulla voidaan kehittää kaikkia yrityksen ohjelmistoprosessin osa-alueita, joko kokonaisuutena tai yksittäisinä prosesseina. Yrityksen ohjelmistoprosessi osana laadunhallintajärjestelmää voidaan tarvittaessa sertifioida jollakin laatusertifikaatilla, kuten ISO 9001 -sertifikaatilla.

Arviointikehykset, standardit ja osa valmiista prosessimalleista ovat pienille organisaatioille aivan liian raskaita, joten ketterämpien prosessimallien käyttöä kannattaa harkita. Arviointikehyksiä voidaan kuitenkin käyttää yksittäisten olemassa olevien prosessien kehittämiseen ja parantamiseen, koska niiden käyttämisestä saatavat hyödyt ovat merkittäviä.

Kaikilla ohjelmistoyrityksillä on ohjelmistoprosessi, mutta aloittavan ohjelmistoyrityksen kannattaa valita käyttämänsä ohjelmistoprosessi erityisen huolellisesti. Aloittava ohjelmistoyritys elää tilanteessa, jossa ensimmäisten asiakkaiden vakuuttamiseksi tarvitaan toimiva ohjelmistoprosessi, jonka avulla yritys pystyy vähäisillä taloudellisilla ja henkilöresursseilla tuottamaan asiakkaalle toimivan ratkaisun. Aloittavan yrityksen henkilöstön, eli käytännössä yrittäjien, on usein yrityksen alkuaikoina toimittava yrittäjänä, projektiryhmänä sekä myyjinä. Tämä omien voimavarojen jakaminen use-

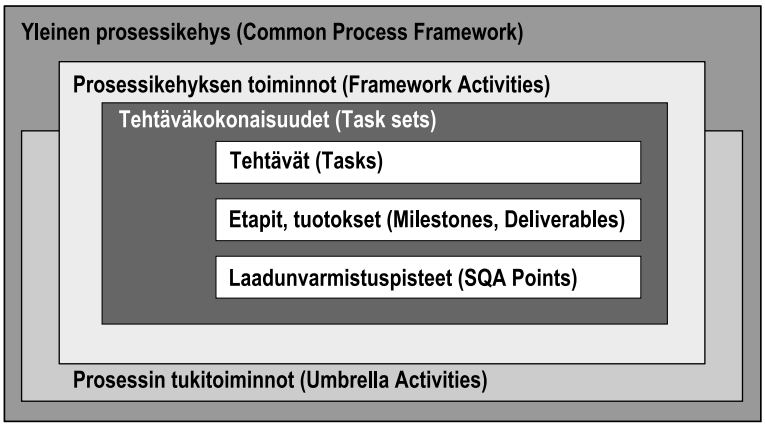
aan tehtävään on usein haastavaa. Ensimmäisen asiakkaan vakuuttaminen tilaamaan on aina vaikeinta, ja siinä toimiva ohjelmistoprosessi voi auttaa merkittävästi. Kun referenssiasiakas on olemassa, on seuraavan asiakkaan hankinta jo huomattavasti helpompaa.

Tämän tutkielman kantavana ajatuksena on selvittää, tarvitseeko aloittava ohjelmistoyritys ohjelmistoprosessin. Ja jos tarvitsee, niin minkälaisen. Tutkielman tarkoituksena on tarjota tuleville aloittaville ohjelmistoyrittäjille pohja ja työkalut ohjelmistoprosessin kehittämiseen omaan yritykseen.

Tässä tutkielmassa esitellään ensin ohjelmistoprosessin teoreettinen viitekehys luvussa 2. Tämän jälkeen, luvussa 3 esitellään esimerkkien kautta yleisimmät prosessimallityypit, eli suunnitelmaohjautuvat ja ketterät prosessimallit. Ohjelmistoprosessin ja erilaisten prosessimallien kuvaamisen jälkeen, luvussa 4 on esitelty muutamia ohjelmistoprosessin arvioinnissa ja kehittämisessä käytettäviä arviointikehyksiä sekä standardeja. Luku 5 kuvaa aloittavan ohjelmistoyrityksen ominaisuuksia, sekä sitä tilannetta, jossa aloittavan ohjelmistoyrityksen on toimittava. Luvussa 6 on kuvattu todellisessa aloittavassa ohjelmistoyrityksessä kehitetty suunnitelmaohjautuvan ja ketterän prosessimallin yhdistelmä, jota voidaan käyttää oman prosessimallin kehittämisen pohjana. Tutkielman päättää yhteenveto ja johtopäätökset luvussa 7.

2 Ohjelmistoprosessi

Ohjelmistotuotannossa ohjelmistojen kehittäminen tapahtuu tietyllä tavalla, tiettyjä käytäntöjä, menetelmiä ja lainalaisuuksia noudattaen. Nämä käytännöt, menetelmät ja lainalaisuudet sisältyvät ohjelmistoprosessiin. *Ohjelmistoprosessi* (software process) on erilaisista projektissa suoritettavista toiminnoista muodostuva kokonaisuus, jonka onnistuneen suorittamisen tuloksena syntyy valmis ohjelmistotuote (tSoft, 2008a).



Kuva 1: Ohjelmistoprosessin rakenne (Pressman, 2000).

Pressman (2000) kuvaa ohjelmistoprosessin muodostuvan pienestä joukosta kaikkiin ohjelmistoprojekteihin soveltuvia, yleisiä ja osittain tietyssä järjestyksessä suoritettavia *prosessikehyksen toimintoja*. Nämä prosessikehyksen yleisen tason toiminnot sovitetaan erilaisiin projekteihin *tehtäväkokonaisuuksien* avulla. Tehtäväkokonaisuuksia voi olla useita ja kukin niistä muodostuu spesifisistä ohjelmistotuotannon elinkaaren *tehtävistä*, projektin *etapeista* ja *tuotoksista*, sekä *laadunvarmistuspisteistä*. Tällaisia tehtäväkokonaisuuksia ovat esimerkiksi vaatimusmäärittely ja integraatiotestaus, jotka itsessään sisältävät joukon alemman tason tehtäviä ja laadunvarmistuspisteitä.

Näiden tietyssä vaiheessa prosessia suoritettavien tehtäväkokonaisuuksien lisäksi koko ohjelmistoprosessin ajan suoritetaan tiettyjä itsenäisiä *ohjelmistoprosessin tukitoimintoja*, kuten version- ja muutostenhallinta sekä mittaaminen. Tukitoimintojen tarkoitus on tukea varsinaisen ohjelmistoprosessin läpivientiä sekä mahdollistaa halutun lopputuloksen saavuttaminen ja prosessin parantaminen.

Ohjelmistoprosessin sisällöstä on olemassa kaksi päästandardia, IEEE 1974-1991 ja ISO/IEC 12207. Kumpikin näistä standardeista määrittää joukon olennaisia toimintoja, jotka tulee suorittaa, jotta voidaan tuottaa kokonainen ohjelmistotuote. Kumpikaan standardi ei ota kantaa näiden toimintojen keskinäiseen järjestykseen, vaan kuvaa ainoastaan vaatimukset toiminnoille (Acuña & Ferré, 2001).

3 Ohjelmistoprosessimallit

Koska ohjelmistoprosessi muodostuu erillisistä ja osittain tietyssä järjestyksessä olevista käytännöistä, tarvitaan malli, jonka mukaan vaiheet järjestetään ja tehtävät suoritetaan. Tällaisen mallin tarjoaa *ohjelmistoprosessimalli* (software process model), jota kutsutaan myös *elinkaarimalliksi* (life-cycle model). Ohjelmistoprosessimalli kuvaa ohjelmiston kehittämisen vaiheet, ja ohjelmistoprosessi määrittää mitä tehtäviä niissä suoritetaan (Robillard *& al.*, 2002).

Ohjelmiston *elinkaarella* (life cycle) tarkoitetaan Haikalan ja Märijärven (2003) mukaan aikaa ohjelmiston kehittämisen aloittamisesta sen poistamiseen käytöstä. Elinkaarimalli kuvaa ohjelmiston toteuttamisprosessissa suoritettavat vaiheet ja tehtävät sekä niiden keskinäisen suorittamisjärjestyksen. Lisäksi elinkaarimalli kuvaa vaiheiden syötteet, tuotokset, laadunvarmistuspisteet sekä vaiheisiin liittyvät etapit. *Vaiheet* ovat ylemmän tason kokonaisuuksia, jotka suoritetaan tietyssä järjestyksessä. Tällaisia vaiheita ovat esimerkiksi määrittely- ja toteutusvaihe. Vaiheet sisältävät joukon ylemmän tason toimintoja, kuten vaatimusmäärittely, sekä alemman tason tuote- tai projektikohtaisia tehtäviä, kuten esimerkiksi vaatimusmäärittelyn toteutuksen ohessa tehtävä asiakashaastattelu.

Käytettävä ohjelmistoprosessi valitaan Robillardin *& al.* (2002) mukaan tilanteesta riippuen, yleensä sovellusalueen, projektissa olevien ihmisten ja organisaatioiden ominaisuuksien, tuotteen elinkaaren tai jonkun käytettävän ohjelmistoprosessistandardin perusteella. Esimerkiksi liiketoimintasovelluksissa elinkaaren suunnittelu- ja testausvaiheita korostetaan verrattuna muunlaisten sovellusten kehittämiseen. Toisaalta pienille yrityksille ei sovi samanlainen ohjelmistoprosessi kuin suurille yrityksille, joissa ohjelmisto-organisaation koko voi olla merkittävästi suurempi. Tämän sovitettavuusvaatimuksen vuoksi ohjelmistoprosessin onkin tarpeen mukaan oltava sovitettavissa ja konfiguroitavissa.

Ohjelmistoprosessin konfiguroitavuus tarkoittaa käytännössä sitä, että esimerkiksi hyvin kriittiseen ympäristöön tuotettavien sovellusten kehittämisprosessissa voidaan

käyttää tarkempia suunnittelu- ja testauskäytäntöjä. Näin projektissa käytettävä ohjelmistoprosessi on sovitettu tarpeita vastaavaksi ja noudattaa edelleen prosessimallia.

Boehmin ja Turnerin (2003) mukaan prosessimallit voidaan käytännössä jakaa karkeasti kahteen lähestymistapaan - suunnitelmaohjautuviin (plan-driven methods) ja ketteriin (agile methods) prosessimalleihin.

Suunnitelmaohjautuvat prosessimallit perustuvat alun perin teollisuuden ja avaruusteknologian kehittämisen puolelta ohjelmistotuotantoon tuotuun ”vaatimukset-suunnittelu-toteutus” -malliin, jossa kaikki ohjelmistoprosessin elinkaaren aikana suoritettavat asiat on määritelty tarkasti prosesseiksi, joita mitataan ja parannetaan jatkuvasti (Boehm & Turner, 2003). Yleispiirteenä näille malleille on etupainoinen projektin laajuuden, kustannusten ja aikataulun tarkka määrittäminen sekä erittäin laaja dokumentaatio (Salo, 2006).

Vastapainoksi raskaille suunnitelmaohjautuville prosessimalleille kehitettiin joukko kevyempiä kehitysmenetelmiä, jotka ovat yleensä kokoelmia ohjelmistotuotannon parhaista käytännöistä. Näitä joustavia ja muutoksiin paremmin reagoivia *ketteriä prosessimalleja* alettiin vuonna 2001 pidetyssä kevyiden menetelmien kehittäjien kokoontumisessa kutsua yhteisellä nimellä ketteriksi kehitysmenetelmiksi, ja niiden perusarvot on kirjattuna ”Ketterän ohjelmistokehityksen manifestiin” (Agile Manifesto, 2008).

Ketterien kehitysmenetelmien perusajatuksena on tarjota menetelmiä, jotka pystyvät vastaamaan ohjelmistoprojektien aikaisiin muutoksiin paremmin. Manifestin mukaan ketterät kehitysmenetelmät suosivat yksilöitä ja kommunikointia prosessien ja työkalujen sijaan, toimivaa ohjelmistoa kattavan dokumentaation sijaan, asiakasyhteistyötä sopimusneuvotteluiden sijaan ja muutoksiin reagoimista suunnitelman noudattamisen sijaan (Koch, 2005). Yleensä ketterät kehitysmenetelmät ovatkin kevyitä, hyvin joustavia, *iteratiivisia* ja *inkrementaalisia* ohjelmistokehitysmenetelmiä (Boehm & Turner, 2003). Esimerkkinä ketteristä kehitysmenetelmistä, tässä tutkielmassa on kuvattu yleisimmin käytetyt Scrum ja XP (Extreme Programming). Scrum on kuvattu kohdassa 3.2, ja XP kohdassa 3.3.

3.1 Yleinen suunnitelmaohjautuva lähestymistapa

Haikalan ja Märijärven (2003) mukaan suunnitelmaohjautuvissa prosessimalleissa on yleensä erotettavissa viisi vaihetta, jotka suoritetaan samassa järjestyksessä: määrittely, suunnittelu, ohjelmointi, testaus sekä käyttöönotto ja ylläpito. Vaiheiden suoritusjärjestyksen vuoksi Barry Boehm (1998) onkin Salon (2006) mukaan kutsunut suunnitelmaohjautuvia prosessimalleja myös dokumentti- ja koodiohjautuviksi sekä perinteisiksi prosessimalleiksi. Yleensä kukin vaihe suoritetaan vain kerran, mutta tarvittaessa aiempaan vaiheeseen voidaan palata takaisin. Tällainen palaaminen on esimerkiksi tarpeen, jos toteutuksen aikana havaitaan suunnitteluvirhe, joka täytyy korjata.

Suunnitelmaohjautuvissa prosessimalleissa prosessin tukitoimintoja, kuten projektin- ja muutostenhallintaa, suoritetaan koko ajan varsinaisten kehitystoimintojen rinnalla.

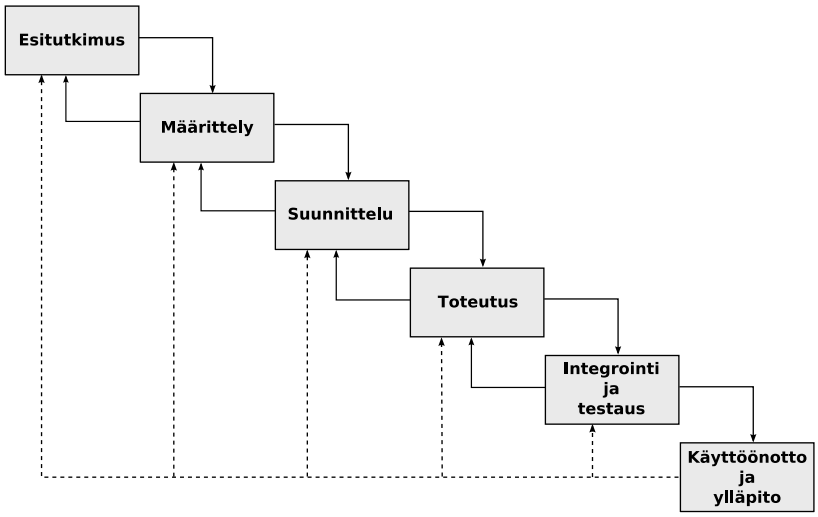
Ehkä tunnetuin ja käytetyin suunnitelmaohjautuvista prosessimalleista on Roycen jo vuonna 1970 kehittämä vesiputousmalli (Royce, 1987). Pohjimmiltaan lähes kaikki suunnitelmaohjautuvat prosessimallit perustuvat vesiputousmallin vaiheistukseen. Suunnitelmaohjautuvia prosessimalleja ovat vesiputousmallin lisäksi esimerkiksi prototyyppi- ja spiraalimalli. Kumpikin malleista sisältää jossakin muodossa vesiputousmallin vaihejaon.

3.1.1 Vesiputous-malli esimerkkinä suunnitelmaohjautuvuudesta

Vesiputousmalli on perinteisin ja ehkä yksi käytetyimmistä suunnitelmaohjautuvista prosessimalleista. Se noudattaa aiemmin kuvattua ”vaatimukset-suunnittelu-toteutus” -mallia. Kuvassa 2 on esitetty esimerkki vesiputousmallin mukaisesta vaihejaosta.

Tässä esimerkissä kuvattu vesiputousmalli sisältää kuusi päävaihetta, jotka tuottavat ideasta valmiin käyttöönotetun ja ylläpidetyn ohjelmistotuotteen. Nämä kuusi päävaihetta ovat esitutkimus (feasibility study), määrittely (requirements specification), suunnittelu (design), toteutus (implementation), integrointi ja testaus (integration and testing) sekä käyttöönotto ja ylläpito (installation and maintenance). Näiden vaiheiden

rinnalla suoritetaan prosessin tukitoimintoja, kuten projektin-, muutosten- ja kokoonpanonhallinta.



Kuva 2: Vesiputousmalli (Haikala & Märijärvi, 2003).

Kehitysprosessi alkaa määrittelyvaiheesta, josta se etenee eri vaiheiden kautta käyttöönottoon ja ylläpitoon. Tarvittaessa ennen määrittelyä voidaan tehdä esitutkimus, jos ongelman ratkaisumahdollisuuksista halutaan varmistua ennen kehitysprojektin aloittamista. Kunkin elinkaaren vaiheen (lukuun ottamatta ylläpitovaihetta) päätteeksi suoritetaan laadunvarmistus, joka on yleensä vaihetuotosten virallinen katselmointi. Laadunvarmistuksen jälkeen siirrytään seuraavaan vaiheeseen, ja edellisen vaiheen tuotokset toimivat seuraavan vaiheen syötteinä.

Esitutkimusvaihe

Esitutkimusvaiheen tarkoituksena on selvittää järjestelmän kehittämisen syyt tai mahdolliset esteet järjestelmän kehittämiselle. Esitutkimusvaihetta kutsutaan usein myös *tarvekartoitukseksi*, ja siinä selvitetään asiakkaan yleisen tason vaatimukset järjestelmälle (Haikala & Märijärvi, 2003). Lisäksi esitutkimusvaiheessa arvioidaan järjestelmän teknistä ja liiketaloudellista toteutettavuutta käytettävissä olevien teknologioiden sekä arvioitujen työmäärien ja kustannusten perusteella. Olennaisena osana esitutkimusta on myös riskienhallinta, eli mahdolliset riskit pyritään jo hyvissä ajoin tunnistamaan ja niiden hallintakeinot löytämään (McConnell, 1998).

Määrittelyvaihe

Määrittelyvaiheessa kerätään asiakkaalta vaatimukset kehitettävälle järjestelmälle, jos sitä ei jo esitutkimusvaiheessa aiemmin ole tehty. Määrittelyvaiheessa näistä asiakkaan yleisistä vaatimuksista johdetaan ohjelmistovaatimuksia, jotka ovat kehitettävän järjestelmän ominaisuuksia. Tässä vaiheessa määritellään kaikki ohjelmiston toiminnot ja toteutukselle asetettavat ei-toiminnalliset vaatimukset sekä rajoitukset. Periaatteena on, että määrittelyvaihe tuottaa *toiminnallisen määrittelyn*, jossa on kuvattu kaikki kehitettävälle ohjelmistolle asetetut vaatimukset. Nämä vaatimukset täytettyään ohjelmisto on valmis ja täyttää asiakkaan esittämät asiakasvaatimukset (Haikala & Märijärvi, 2003). Lisäksi määrittelyvaiheessa tuotetaan *testaussuunnitelmaan* testitapauksia, joilla ominaisuuksien toiminta testataan testausvaiheessa.

Suunnitteluvaihe

Suunnitteluvaiheessa suunnitellaan sellaisen järjestelmän arkkitehtuuri ja tekninen toteutus, joka sisältää kaikki toiminnallisen määrittelyn toiminnot. Tällä tavoin kehitetty järjestelmä täyttää alkuperäiset asiakasvaatimukset. Suunnitteluvaihe voidaan jakaa kahteen tai useampaan alivaiheeseen. Yleisimmin se jaetaan arkkitehtuurisuunnitteluun (architectural design) ja moduulisuunnitteluun (detailed design). *Arkkitehtuurisuunnittelussa* suunnitellaan järjestelmän tekninen yleisarkkitehtuuri ja moduulijako. *Moduulisuunnittelussa* suunnitellaan yksittäiset moduulit ja niiden tekninen toteutustapa. Suunnitteluvaihe tuottaa *teknisen määrittelyn*, joka kuvaa, minkälaisella teknisellä ratkaisulla toiminnallisen määrittelyn mukainen järjestelmä voidaan toteuttaa (Haikala & Märijärvi, 2003). Lisäksi suunnitteluvaiheessa tuotetaan teknisiä testitapauksia testaussuunnitelmaan.

Toteutusvaihe

Toteutusvaiheessa suoritetaan varsinainen ohjelmointityö. Siinä järjestelmä toteutetaan aiemmin valmistuneiden teknisten ja toiminnallisten määritysten pohjalta. Tämän vaiheen päättyessä järjestelmästä on olemassa täydellinen, kaikki asiakasvaatimukset täyttävä ja kaikki määritetyt ominaisuudet sisältävä versio (Haikala & Märijärvi, 2003).

Integrointi- ja testausvaihe

Integrointi- ja testausvaiheessa järjestelmä integroidaan ja testataan kokonaisuutena. Testausvaihe jaetaan yleensä moduuli-, integrointi- ja järjestelmätestaukseen. *Moduulitestauksella* varmistetaan, että yksittäiset moduulit täyttävät vaatimuksensa ja toimivat virheettömästi. *Integrointitestauksessa* testataan kokonaisuudeksi integroitujen moduulien yhteistoimintaa. Kun kaikki moduulit on integroitu yhdeksi kokonaisuudeksi ja integraatiotestaus on suoritettu, suoritetaan *järjestelmätestaus*, jossa järjestelmän kaikki ominaisuudet varmistetaan ja testataan (Haikala & Märijärvi, 2003). Kaikki testaus suoritetaan testaussuunnitelman testitapauksien mukaisesti, ja testien tulokset kirjataan testausraportteihin.

Käyttöönotto- ja ylläpitovaihe

Valmis ja testattu järjestelmä toimitetaan asiakkaalle ja asennetaan käyttöympäristöönsä. Asennuksen ja käyttöönoton jälkeen aloitetaan *ylläpitovaihe*, jossa järjestelmätoimittaja tarjoaa asiakkaalle ylläpitopalvelua. Ylläpito sisältää yleensä tukipalvelut, ohjelmistovirheiden korjauksia, ohjelmiston päivitysversioita ja joskus myös uusien ominaisuuksien kehittämistä (Haikala & Märijärvi, 2003). Ylläpitovaihe kestää ohjelmiston elinkaaren loppuun saakka, eli se päättyy, kun kukaan ei enää kehitä tai käytä järjestelmää. Käytännössä ylläpitovaihe päättyy usein toimittajan ilmoitukseen ylläpidon lopettamisesta ja uuden sovelluksen tarjoamiseen lopetetun tilalle.

3.1.2 Hyödyt ja haasteet

Suunnitelmaohjautuville prosessimalleille on yhteistä, että elinkaaren alussa suoritetaan tarkka määrittely- ja suunnitteluvaihe, joka tuottaa kaikki järjestelmältä vaaditut ominaisuudet sisältävät ja yksityiskohtaisen tarkat määrittelydokumentit toteutusta varten (Haikala & Märijärvi, 2003). Yksityiskohtaisella toiminnallisten ja teknisten määritysten tekemisellä ennen toteutuksen aloittamista pyritään lukitsemaan järjestelmään kehitettävät ominaisuudet, mikä mahdollistaa projektin toteutusvaiheen tarkan aikatauluttamisen ja kustannusten sekä työmäärien paikkansapitävyyden.

Valitettavasti juuri tämä etupainoinen suunnittelu ja projektin laajuuden, kustannusten sekä aikataulun tarkka määrittäminen on tehnyt suunnitelmaohjautuvista prosessimalleista jäykkiä ja muutoksille herkkiä.

Morien (2005) on tutkinut perinteisten prosessimallien mukanaan tuomien haasteiden syitä. Ongelmat aiheutuvat hänen mukaansa aikataulujen ja kustannusten etukäteen arvioinnin vaikeudesta, puutteellisesti määritetyistä vaatimuksista sekä prosessin aikana vaatimuksiin kohdistuvista muutoksista. Puutteelliset vaatimukset johtuvat osaltaan siitä, ettei asiakas yksinkertaisesti pysty toimittamaan täydellisiä, yksityiskohtaisia ja virheettömiä vaatimuksia heti projektin alussa. Toisaalta vaatimuksiin kohdistuu prosessin aikana monia muutoksia, joihin reagointi vaatii palaamista takaisin prosessin aiempiin vaiheisiin.

Jos jossakin vaiheessa havaitaan virhe, voidaan prosessissa palata taaksepäin siihen vaiheeseen, jossa virhe on aiheutunut. Käytännössä tämä tarkoittaa sitä, että jos esimerkiksi toteutuksen yhteydessä havaitaan puutteellinen määritys, joudutaan prosessissa palaamaan sen osalta määritysvaiheeseen ja määrittämään kyseinen kohta uudelleen. Tämän jälkeen seurataan prosessia eteenpäin suunnitteluvaiheen läpi takaisin toteutukseen ja päivitetään muutoksen osalta myös suunnitteludokumentaatio sekä mahdollisesti kaikki jo toteutetut ominaisuudet, joihin muutos vaikuttaa. Mitä myöhemmin virhe havaitaan, sitä kalliimmaksi virheen korjaaminen tulee.

Huolimatta näistä puutteista ja haasteista, vesiputousmalli on laajasti käytetty prosessimalli. Selkeytensä vuoksi sitä käytetään tietotekniikan opetuksessa edelleen.

3.2 Scrum

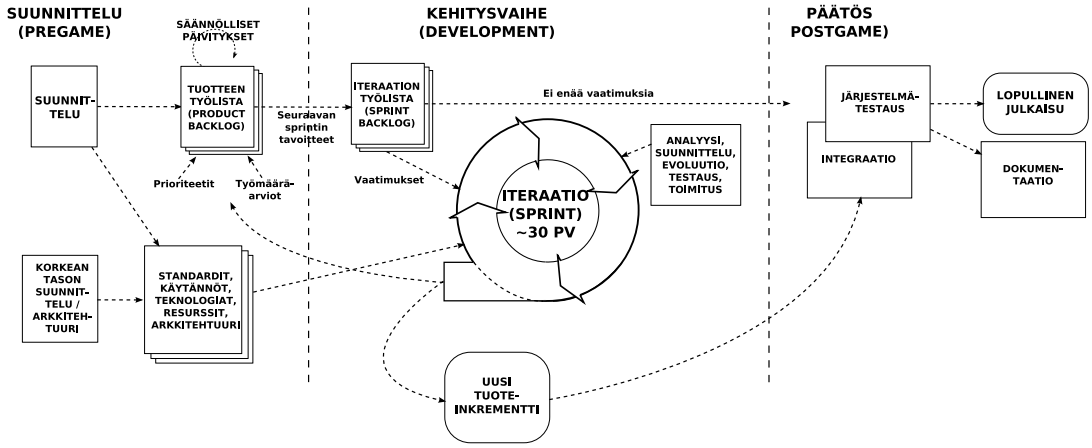
Vastakohtana edellisessä kohdassa kuvatun vesiputousmallin jäykkyydelle voidaan pitää ketteriä kehitysmalleja. *Scrum* on XP:n ohella tunnetuin ja yleisimmin käytetty ketterä, iteratiivinen ja inkrementaalinen prosessimalli, joka on suunnattu pienille, alle kymmenen hengen projektiryhmille (Rising & Janoff, 2000). Sen kehittivät Ken Schwaber ja Jeff Sutherland 1990-luvulla. Scrum pohjautuu Schwaberin ja Beedlen (2002) mukaan teollisuuden prosessiohjauksen menetelmien soveltamiseen ohjelmistotuotannossa, josta johtuen Scrumin prosessin perusominaisuuksiin kuuluvat joustavuus, sopeutuvuus ja tuottavuus. Nämä perusominaisuudet näkyvät Scrum-mallin eri menetelmissä ja vaiheistuksissa.

Scrumin pääpaino on projektinhallinnassa ja projektien joustavassa läpiviennissä jatkuvasti muuttuvassa ympäristössä. Abrahamssonin *& al.* (2002) mukaan Schwaber (1995) esittää Scrum-mallin kantavaksi ajatukseksi sen, että johtuen järjestelmien kehittämiseen liittyvien ympäristö- ja tekniikkavaatimuksien prosessinaikaisista muutoksista, prosessin etenemisen ennustaminen on hyvin vaikeata ja monimutkaista. Tämän vuoksi tarvitaan hyvin joustava ohjelmistokehitysprosessi, jotta muutoksiin pystytään reagoimaan.

Abrahamssonin *& al.* (2002) mukaan Scrum ei tarjoa tarkkoja toteutusvaiheen kehityskäytäntöjä, mutta enemmänkin auttaa parantamaan olemassa olevia käytäntöjä tehokkaan projektin- ja riskienhallinnan avulla. Scrum onkin Boehmin ja Turnerin (2003) mukaan hyvä tapa sovittaa ketterät kehitysmenetelmät perinteisiä suunnitelmaohjautuvia prosessimalleja käyttäviin organisaatioihin.

3.2.1 Prosessin kuvaus

Scrumin prosessi muodostuu kolmesta päävaiheesta. Ne ovat suunnitteluvaihe (pregame phase), kehitysvaihe (development phase) ja päätösvaihe (post-game phase). Suunnitteluvaihe jakautuu vielä edelleen kahteen alivaiheeseen, suunnitteluun (planning) ja arkkitehtuurisuunnitteluun (architecture / high level design).



Kuva 3: Scrum-prosessin elinkaarimalli (Abrahamsson *& al.*, 2002).

Kehitysprosessi alkaa suunnitteluvaiheella, josta se etenee *kehitysiteraatioiden* (sprint) kautta päätösvaiheeseen (Abrahamsson *& al.*, 2002).

Suunnitteluvaihe

Scrum-projekti lähtee Schwaberin (2004) mukaan kehitettävän järjestelmän visiosta. Tämän vision pohjalta luodaan suunnitteluvaiheessa kehitettävälle järjestelmälle asetettavat toiminnalliset ja muut vaatimukset tuotteen työlistaan (product backlog), jota priorisoidaan liiketoiminnallisen merkityksen mukaisesti. Tähän listaan Abrahamssonin *& al.* (2002) mukaan kootaan kaikki, mitä lopullisessa tuotteessa nykytietämyksen perusteella tarvitaan.

Tuotteen työlista kuvaa kaikki työtehtävät, jotka projektissa tulee suorittaa, sekä niiden arvioidut toteutustyömäärät. Tehtävät voivat olla esimerkiksi järjestelmän ominaisuuksia, bugikorjauksia, puutteita tai muutospyyntöjen kautta saatuja parannusehdotuksia.

Projektin edetessä tuotteen työlistaa voidaan muokata. Tuotteen työlistan tehtäviä voidaan priorisoida uudelleen, päivittää ja tarkentaa, sitä mukaa kun tarkempaa tietoa asioista saadaan. Myös tehtävien työmääräarvioita tarkennetaan projektin edetessä. Tarvittaessa tehtäviä voidaan poistaa tai uusia lisätä. Työlistan lisäksi suunnitteluvaiheessa määritetään mm. käytettävät resurssit, teknologiat ja työkalut (Abrahamsson *& al.*, 2002).

Suunnitteluvaiheessa suoritettavassa arkkitehtuurisuunnittelussa suunnitellaan järjestelmälle karkean tason arkkitehtuuri tuotteen työlistan perusteella. Tämä arkkitehtuuri tarkastetaan ennen toteuttamisen aloittamista *arkkitehtuurin tarkastuspalaverissa* (design review meeting). Lisäksi tuotteen työlistan ominaisuudet kiinnitetään alustavasti *julkaisuihin* (Abrahamsson *& al.*, 2002).

Kehitysvaihe

Järjestelmän kehittäminen tehdään *kehitysvaiheessa* inkrementaalisesti 1-4 viikon mittaisissa iteraatioissa eli *sprinteissä*. Kehitysvaiheen alussa pidetään *sprintin suunnittelupalaveri* (sprint planning meeting), jossa valitaan tuotteen työlistalta ne tehtävät, jotka alkavassa sprintissä suoritetaan. Suoritettavien tehtävien valinta tehdään prioriteetin mukaisesti, jotta liiketoiminnallinen hyöty ja tuottavuus saadaan maksimoitua. Sprintissä toteutettavaksi valituista ominaisuuksista muodostetaan *sprintin työlista* (sprint backlog), jota ei muuteta sprintin aikana. Ensimmäisten sprinttien aikana toteutetaan järjestelmän perusarkkitehtuuri, jotta varsinaisten ominaisuuksien kehittäminen seuraavissa sprinteissä olisi mahdollista (Schwaber, 2004).

Kukin sprintti sisältää perinteisen ohjelmistokehityksen työvaiheet - vaatimukset, analysointi, suunnittelu, toteutus ja toimitus. Näiden vaiheiden kautta tuotetaan ja integroidaan seuraavaan julkaisuversioon uusia ominaisuuksia (Abrahamsson *& al.*, 2002). Schwaberin (2004) mukaan sprintin aikana pidetään projektiryhmän kesken päivittäin n. 15 minuutin mittainen *seurantapalaveri* (daily scrum). Tässä palaverissa tarkastellaan työn etenemistä ja jäljellä olevan työn määrää. Jokainen kehittäjä vastaa seuraaviin kysymyksiin: 1) Mitä olet tehnyt projektissa edellisen palaverin jälkeen? 2) Mitä aiot tehdä projektissa tämän ja seuraavan palaverin välissä? ja 3) Onko sinulla ongelmia, jotka estävät sinua pääsemästä tässä sprintissä asettamiisi tavoitteisiin? (Schwaber, 2004).

Näiden palavereiden avulla Risingin ja Janoffin (2000) mukaan saadaan keskitettyä resurssit olennaisiin asioihin, priorisoitua tuotteen työlistan ominaisuuksia uudelleen sekä havaittua esteet ajoissa ja reagoitua niihin. Lisäksi projektin etenemistä voidaan seurata tarkemmalla tasolla, mikä auttaa projektin riskien minimoinnissa.

Sprintin valmistuttua pidetään yhteinen *sprintin tarkastelutilaisuus* (sprint review meeting), jossa esitellään sprintin toimivat tuotokset johdolle, asiakkaille ja käyttäjille (Abrahamsson *& al.*, 2002). Tässä tilaisuudessa päätetään seuraavista toimista ja tarvittaessa voidaan tuoda uusia tehtäviä tuotteen työlistaan, poistaa tehtäviä työlistalta tai priorisoida niitä uudelleen (Rising & Janoff, 2000).

Yleisen tarkastelutilaisuuden lisäksi projektiryhmä pitää Schwaberin (2004) mukaan oman *sisäisen sprintin tarkastelutilaisuuden* (sprint retrospective). Tässä tilaisuudessa tarkastellaan mennyttä sprinttiä ja käydään läpi mm. hyvin menneet sekä parantamista tarvitsevat asiat. Parannustoimenpiteet lisätään mahdollisuuksien mukaan tuotteen työlistaan, jotta ne voidaan ottaa huomioon seuraavassa sprintissä. Tämä tarkastelutilaisuus onkin enemmän empiirinen tapahtuma, jonka tavoitteena on tehdä työskentelystä tarkoituksenmukaisempaa ja tehokkaampaa.

Projektin etenemistä seurataan Schwaberin (2004) mukaan tuotteen ja sprintin työlistoissa toteuttamatta olevien tehtävien määrällä. Kokonaisprojektin etenemisen seuranta tapahtuu vertaamalla tuotteen työlistassa vielä suorittamatta olevien tehtävien määrää kaikkien tuotteen työlistassa olevien työtehtävien määrään. Tätä kokonaisprojektin etenemisen seurantaa kuvataan usein työmäärän vähenemistä kuvaavalla graafilla (*product burndown*), jossa kuvataan tuotteen työlistassa kunkin seurantapalaverin kohdalla jäljellä olevien tehtävien määrää.

Vastaavasti sprintin etenemistä seurataan vertaamalla sprintin työlistassa vielä suorittamatta olevien tehtävien määrää kaikkien sprintin työlistassa olevien työtehtävien määrään. Tätä sprintin etenemisen seurantaa kuvataan samalla tavalla graafisesti (*sprint burndown*).

Päätösvaihe

Kun tuotteen työlistalla ei enää ole uusia tehtäviä toteutettavaksi, siirrytään projektin *päätösvaiheeseen*. Tässä vaiheessa järjestelmän lopullinen versio on valmis ja sille suoritetaan järjestelmätestaus. Lisäksi sille kirjoitetaan tarvittava dokumentaatio.

3.2.2 Ydinkäytännöt

Scrumin ydinkäytäntöjä ovat Abrahamssonin *& al.* (2002) mukaan edellisessä kohdassa kuvatut tuotteen ja sprintin työlista, sprintit, työmääräarvioiden tekeminen, sprintin suunnittelutapaaminen, päivittäiset palaverit sekä sprintin tarkastelutilaisuus.

Ydinkäytäntöjen luettelosta on havaittavissa, kuten aiemmin tässä luvussa jo todettiin, ettei Scrum tarjoa spesifisiä ohjelmistokehityksen menetelmiä toteutusvaiheeseen, vaan projektinhallinnallisen lähestymistavan prosessiin (Schwaber & Beedle, 2002).

3.2.3 Hyödyt ja haasteet

Scrum on kehitetty ketteräksi projektinhallintaan painottuvaksi prosessimalliksi pienille, alle 10 hengen projektiryhmille. Yli kymmenen hengen ryhmät tulisi jakaa alle 10 hengen osaryhmiksi, joiden yhteistoimintaa tulisi koordinoida hyvin, jotta läpiviennissä onnistuttaisiin hyvin (Rising & Janoff, 2000). Menetelmä on XP:n ohella tunnetuimpia ketterän kehityksen menetelmiä, ja sitä on tutkittu tieteellisessä kirjallisuudessa jonkin verran.

Schwaberin ja Beedlen (2002) mukaan Scrumia voidaan käyttää joustavana projektinhallintamallina, samalla käyttäen organisaation olemassa olevia ohjelmistokehityksen menetelmiä ja käytäntöjä. Scrum ei itsessään tarjoa mitään yksittäisiä kehitysmenetelmiä, vaan ainoastaan työkalut projektin läpivientiin ja vaiheistukseen.

Rising ja Janoff (2000) tutkivat Scrumin soveltuvuutta pienille projektiryhmille. He havaitsivat, että menetelmä toimii hyvin uuden tuotteen kehitysprojektissa sekä olemassa olevan tuotteen jatkokehitys- ja ylläpitoprojekteissa. Heidän mukaansa Scrum paransi projektiryhmän yhteistyötä, tiivisti projektiryhmää ja paransi yhteishenkeä yhteisen tavoitteen saavuttamiseksi. Kaikissa kolmessa projektiryhmässä, ryhmän jäsenet olivat erittäin tyytyväisiä Scrumiin. Mann ja Maurer (2005) havaitsivat projektiryhmän lisäksi myös asiakkaiden olleen erittäin tyytyväisiä Scrumiin onnistuneen projektin jälkeen. Esimerkkinä Scrumin mukanaan tuomista muutoksista olivat heidän mukaansa

mm. ylitöiden määrän merkittävä väheneminen, mikä osaltaan parantaa projektiryhmän työviihtyvyyttä merkittävästi.

Tämä ylityön määrän väheneminen oli mahdollista osittain hienojakoisemman projektinhallinnan, eli päivittäisen ohjauksen ja seurannan ansiota. Scrumin ansiosta työkuorma jakautui paremmin henkilöiden kesken, ja työn etenemisen jatkuva seuranta osaltaan vähensi työmäärän kasautumista tiettyyn kohtaan projektissa.

Koska Scrum mahdollistaa myös soveltamisen, on sen joustavuudessa omat etunsa prosessimallina. Yrityksen olemassa olevia käytäntöjä voidaan käyttää kaikessa, ja tehtyä työtä saadaan seurattua paljon tarkemmin. Projektiryhmälle jää vähemmän aikaa olla jouten tuottamattomana, kun projektia viedään eteenpäin päivätasolla. Tämä tuotannon käyttöasteen kasvattaminen on pienelle ohjelmistoyritykselle erittäin tärkeätä, jotta yrityksen toimintaa pystytään kannattavasti ylläpitämään ja kehittämään.

Scrumin käyttäminen yrityksen prosessimallina ei sulje pois mahdollisuutta käyttää myös muita malleja ja arviointikehyksiä. Tieteessä on useita tutkimuksia Scrumin käyttämisestä yhdessä muiden prosessimallien ja arviointikehyksien, kuten CMMI:n (Marçall *& al.*, 2008; Sutherland *& al.*, 2007) tai XP:n (Upender, 2005; Vriens, 2003) kanssa.

Sutherland *& al.* (2007) tutkivat Scrumin ja CMMI-mallin yhdistämisen vaikutuksia. He havaitsivat, että yhdistettyään ketterän menetelmän ja hyvin perinteisen CMMI-kehyksen, lopputulos oli CMMI:n ansiosta ennustettava, mitattava ja laadukas, mutta ketterä prosessimalli paransi merkittävästi tuottavuutta ja laatua verrattuna perinteisiin menetelmiin. Lopputulos oli edelleen CMMI-yhteensopiva, mutta toisaalta ketterä. He suosittelevatkin, että ketteristä menetelmistä saatavia hyötyjä tehostettaisiin käyttämällä lisäksi CMMI:n kolmannen tason yleisiä käytäntöjä.

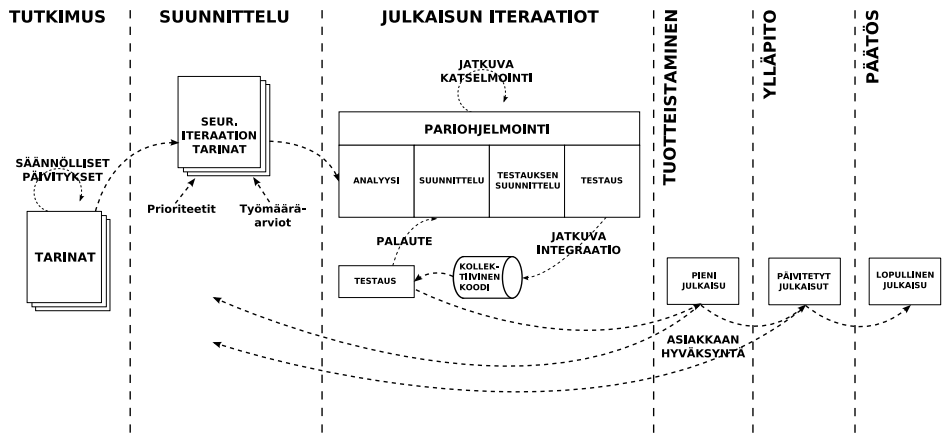
3.3 Extreme Programming (XP)

Extreme Programming (XP) on Scrumin ohella tunnetuin ja yleisimmin käytetty ketterä ohjelmistokehitysmenetelmä. Sen kehittivät Beck, Cunningham ja Jeffries 1990-luvulla (Boehm & Turner, 2003). Se on kehitetty ratkaisemaan perinteisen suunnitelmaohjautuvan ohjelmistotuotannon prosessimallien, kuten vesiputousmallin, mukanaan tuomia ongelmia ja haasteita (Beck, 1999). Tällaisia haasteita ovat esimerkiksi vaatimuksiin kohdistuvien muutoksien suuri vaikutus ohjelmistoprojektien työmääriin ja prosessin jäykkyys.

Abrahamssonin *& al.* (2002) mukaan Beck (1999) esittää XP:n perusajatukseksi sen, ettei ole olemassa prosessia, joka sopii sellaisenaan kaikkiin projekteihin. Sen sijaan on olemassa yksittäisiä käytäntöjä, jotka voidaan räätälöidä jokaiseen projektiin. XP on joukko tällaisia parhaita käytäntöjä yhdessä sovellettuna ja se on tarkoitettu pienille ja keskisuurille projektitiimeille.

3.3.1 Prosessin kuvaus

XP:n elinkaarimalli muodostuu viidestä vaiheesta: tutkimus (exploration phase), suunnittelu (planning phase), julkaisun iteraatiot (iterations to release phase), tuotteistaminen (productionizing phase), ylläpito (maintenance phase) sekä päätös (death phase).



Kuva 4: XP prosessin elinkaarimalli (Abrahamsson *& al.*, 2002)

Kuvassa 4 on esitetty XP:n ohjelmistoprosessin elinkaarimalli. Vaiheet suoritetaan vasemmalta oikealle, ja tietyissä kohdissa palataan takaisin aiempaan vaiheeseen.

Tutkimus

Projekti aloitetaan *tutkimusvaiheesta*, jossa asiakkaat kirjoittavat järjestelmän ensimmäiselle versiolle asettamansa ominaisuus- ja muut vaatimukset yksittäisiksi ja mahdollisimman yksinkertaisiksi *käyttötarinoiksi* (story cards). Kukin käyttötarina kuvaa tietyn yksittäisen, järjestelmään toteutettavan ominaisuuden (Abrahamsson *& al.*, 2002). Kun asiakas on saanut kirjoitettua käyttötarinat, ohjelmoijat analysoivat ne ja varmistavat, että niiden mukaiset ominaisuudet on mahdollista toteuttaa, ja että niiden työmäärän arviointi on mahdollista (Newkirk & Martin, 2000). Samalla projektiryhmä tutustuu projektissa käytettäviin välineisiin, teknologioihin ja käytäntöihin. Käytännössä tutkimusvaihe korvaa perinteisen vaatimusmäärittelyvaiheen. Tutkimusvaihe voi kestää muutamasta viikosta muutamiin kuukausiin (Abrahamsson *& al.*, 2002). Käyttötarinat elävät koko projektin ajan, eli niitä voidaan poistaa, lisätä ja tarkentaa toteutuksen edetessä.

Suunnittelu

Suunnitteluvaiheessa priorisoidaan käyttötarinoiden toteutusjärjestys ja sovitaan ensimmäiseen pienempään julkaisuun sisältyvät ominaisuudet. Kun ominaisuudet ensimmäiseen julkaisuun on valittu, ohjelmoijat antavat asiakkaalle työmääräarvion kunkin ominaisuuden toteuttamisesta. Asiakas päättää mukaan tulevat ominaisuudet sekä ensimmäiseen versioon toteutettavat ominaisuudet, ja aikataulu lukitaan (Abrahamsson *& al.*, 2002). Waken (2000) mukaan liian suuret käyttötarinat voidaan jakaa erillisiin pienempiin käyttötarinoihin. Työmääräarvioita helpottaakseen ohjelmoijat voivat toteuttaa nopean, muutamassa tunnissa tehdyn *prototyypin* (spike), jolla jonkin tietyn toteutustavan tai -teknologian käyttöä voidaan kokeilla ennen toteutuksen aloittamista.

Ensimmäinen julkaisu pyritään tuottamaan yleensä alle kahdessa kuukaudessa. Suunnitteluvaihe kestää yleensä muutaman päivän (Abrahamsson *& al.*, 2002) ja vastaa käytännössä perinteistä projektisuunnittelua.

Yhdessä tutkimus- ja suunnitteluvaihe muodostavat yhden XP:n ydinkäytännöistä, suunnittelupelin (planning game), johon osallistuvat asiakkaat ja ohjelmoijat. Suunnittelupelin tavoitteena on saada sekä asiakas, että ohjelmoijat ymmärtämään mitä ollaan tekemässä, ja valita ensimmäiseen julkaisuun suurimman vaikutuksen tuottavat ominaisuudet. Tässä asiakkaan ja ohjelmoijien välisessä tiiviissä kommunikaatiossa käytetään apuna käyttäjätarinoita (Wake, 2000).

Julkaisun iteraatiot

Julkaisun iteraatioissa julkaisun toteuttaminen jaetaan useampaan 1-3 viikon mittaiseen iteraatioon. Tämä tapahtuu iteraation suunnittelussa, jossa asiakas valitsee kuhunkin iteraatioon kokonaisuudessaan toteutettavat käyttötarinat ja lupaa olla muuttamatta tai lisäämättä ominaisuuksia, ennen kuin iteraatio on valmis. Toteutettavien käyttötarinoiden valitsemisen lisäksi asiakas toimittaa kullekin käyttötarinalle hyväksymistestauksen testitapaukset, jotka suoritetaan kunkin iteraation lopussa.

Kun käyttötarinoiden jaottelu iteraatioihin on valmis, ohjelmoijat analysoivat käyttötarinat ja jakavat kunkin käyttötarinan yksittäisiksi 1-2 päivän kestoisiksi tehtäviksi sekä määrittävät niiden suoritusjärjestyksen. Tämän jälkeen ohjelmoijat valitsevat tehtäviä suoritettavakseen ja tarkentavat näiden tehtävien työmääräarvioita omien kokemuksiensa perusteella. Jos tehtävien yhteenlaskettu työmäärä on suurempi, kuin käyttötarinalle alun perin arvioitu, asiakas päättää siirretäänkö niiden toteutusta seuraaviin iteraatioihin (Newkirk & Martin, 2000).

Kukin ominaisuus toteutetaan järjestelmään siten, että ennen varsinaista ohjelmointia suunnitellaan ja toteutetaan yksikkötestauksen testitapaukset, jotka voivat olla myös automaattisia yksikkötestejä. Varsinainen ohjelmakoodi tehdään sen jälkeen täyttämään nämä testitapaukset (Newkirk & Martin, 2000). Tämä *testiohjautuva kehittäminen* (test-driven development) onkin Waken (2000) mukaan yksi XP:n ydinkäytännöistä, kuten myös kaiken pitäminen mahdollisimman yksinkertaisena. Yksinkertaisuus näkyy käytännössä siten, että esimerkiksi kehitettävään ohjelmistoon ei lisätä mitään ylimääräistä ominaisuutta, tai että ohjelmakoodi pyritään pitämään mahdollisimman yksinkertaisena ja luettavana koko ajan.

Ohjelmointi ja suunnittelu tehdään pariohjelmointina, jolloin kaikki lähdekoodi on pareittain tehtyä. Työpari voi työskennellä yhdessä saman tietokoneen ääressä siten, että toinen kirjoittaa koodia, ja toinen kommentoi ja analysoi työtä vieressä. Tällä tavoin ohjelmakoodi on jatkuvan tarkastuksen alla, ja virheiden määrää saadaan pudotettua merkittävästi. Tarvittaessa ohjelmakoodia voidaan refaktoroida eli parantaa. Tämä menettelytapa tukee myös yksinkertaisuuden periaatetta (Mišić, 2006).

Kaikki kirjoitettu ohjelmakoodi viedään toimivana yhteiseen koodivarastoon (collective codebase), josta se edelleen viedään testattavaksi tai iteraation päätteeksi hyväksymistestiin. *Yhteinen koodivarasto* tarkoittaa käytännössä sitä, että siellä oleva ohjelmakoodi on ajatuksena jokaisen kehittäjän ”omaa”, eli jokainen pari voi testata, muokata, tai päivittää tuotanto- ja testikoodia milloin tahansa. Periaatteena tässä kuitenkin on se, että jatkuvan integroinnin ansiosta koodivarastosta on milloin tahansa otettavissa kääntyvä ja toimiva versio (Mišić, 2006).

Testauksen tulokset antavat palautetta suunnittelulle, jolloin havaittuja ongelmia voidaan ottaa huomioon seuraavissa tehtävissä. Onnistuneen hyväksymistestin jälkeen iteraatiossa kehitetty ohjelmistotuote on valmis julkaistavaksi tuotteistamisvaiheessa (Abrahamsson *& al.*, 2002).

Ensimmäisessä iteraatiossa kehitetään yleensä järjestelmän kokonaisarkkitehtuuri. Seuraavissa iteraatioissa järjestelmän ominaisuuksia toteutetaan inkrementaalisesti, kunnes viimeisen iteraation lopussa järjestelmä on valmis tuotantoon (Abrahamsson *& al.*, 2002). Waken (2000) mukaan asiakas voi iteraatioiden aikana lisätä uusia käyttötarinoita, muokata olemassa olevia tai jakaa niitä erillisiksi tarinoiksi. Iteraatiossa toteutuksen alla oleviin käyttötarinoihin ei kuitenkaan kosketa kesken iteraation.

Tuotteistaminen

Tuotteistamisvaiheessa ohjelmistolle tehdään vielä lisää toiminnallisuuteen ja suorituskykyyn liittyviä testejä. Tässäkin vaiheessa voi tulla vielä uusia muutostarpeita, mutta niiden mukaan ottamista valmistuvaan julkaisuun on harkittava, ja niistä on tehtävä päätös. Tässä vaiheessa suoritetaan tarvittaessa uusia iteraatiokierroksia, kunnes jul-

kaisu on valmis julkaistavaksi. Toteuttamatta jätetyt muutostarpeet listataan ylläpitovaiheessa mahdollisesti toteutettaviksi muutoksiksi (Abrahamsson *& al.*, 2002).

Ylläpito

Kun järjestelmän kehitysiteraation tuloksena julkaistaan uusi versio, aloitetaan valmistuneen version osalta ylläpitovaihe. Tässä vaiheessa ylläpidetään julkaistua versiota erilaisten ylläpitotoimintojen, kuten käyttäjätuen, virheiden korjaamisen ja teknisen tuen kautta. Julkaistuun järjestelmään voidaan kehittää lisää ominaisuuksia uusissa iteraatioissa (Abrahamsson *& al.*, 2002). Tällöin niistä tehdään uusia käyttäjätarinoita, jotka lisätään aiempien käyttötarinoiden joukkoon odottamaan toteuttamistaan.

Päätös

Kun kaikki asiakkaan projektin aikana antamat käyttötarinat on toteutettu järjestelmään ominaisuuksiksi, eikä uusia käyttötarinoita enää ole tulossa, järjestelmän kehittämisprosessi siirtyy päätösvaiheeseen. Tällöin järjestelmä täyttää kaikki sille asetetut vaatimukset, ja siitä on tehty *lopullinen julkaisu*. Tämän jälkeen järjestelmän ohjelmakoodiin, dokumentaatioon tai tekniseen arkkitehtuuriin ei enää tehdä muutoksia, ja järjestelmäversio jäädytetään. Päätösvaiheeseen voidaan päätyä myös siinä tapauksessa, jos järjestelmä ei täytä sille asetettuja vaatimuksia, tai sen kehittäminen on käynyt tarpeettomaksi tai liian kalliiksi (Abrahamsson *& al.*, 2002).

3.3.2 Ydinkäytännöt

XP pohjautuu neljään perusarvoon, jotka otetaan huomioon kaikessa toiminnassa koko ohjelmistoprosessin ajan. Nämä perusarvot ovat Boehmin ja Turnerin (2003) mukaan kommunikaatio, yksinkertaisuus, palaute ja rohkeus. Perusarvot konkretisoituvat mallin ydinkäytäntöjen kautta, jotka ovat käytössä koko edellä kuvatun kehitysprosessin aikana. Ydinkäytäntöjen joukko vaihtelee kirjoittajan mukaan, mutta tässä kohdassa on kuvattu Abrahamssonin *& al.* (2002) luettelemat käytännöt, jotka ovat lähes vastaavia Beckin (1999) luettelemien käytäntöjen kanssa.

Suunnittelupeli (Planning game)

Suunnittelupeliin osallistuvat asiakkaat ja ohjelmoijat. Siinä ohjelmoijat arvioivat kunkin käyttötarinan työmäärän, ja asiakas päättää julkaisuihin toteutettavat ominaisuudet. Sen tavoitteena on saada sekä asiakas, että ohjelmoijat ymmärtämään, mitä ollaan tekemässä, ja valita seuraavaan julkaisuun toteutettavat käyttötarinoiden mukaiset ominaisuudet (Abrahamsson *& al.*, 2002). Bowersin *& al.* (2007) mukaan priorisoinnissa otetaan huomioon sekä tekninen, että liiketoiminnallinen näkökulma.

Pienet julkaisut (Small/short releases)

Yksinkertainen järjestelmä tuotteistetaan nopeasti, vähintään kerran kahdessa tai kolmessa kuukaudessa. Uusia versioita julkaistaan tuotteistamisen jälkeen nopeassa tahdissa, jopa päivittäin (Abrahamsson *& al.*, 2002). Kuhunkin julkaisuun valitaan toteutettavaksi joukko yhteenkuuluvia ja kokonaisia käyttötarinoita (Bowers *& al.*, 2007).

Metafora (Metaphor)

Järjestelmä kuvataan *metaforana*, tai joukkona metaforia. Tämä voidaan nähdä jaettuna tarinana, joka ohjaa koko kehitystä kuvaamalla, kuinka järjestelmä toimii (Abrahamsson *& al.*, 2002). Waken (2000) mukaan metaforan ansiosta asiakkaalla ja kehittäjillä on yhteinen termistö ja näkemys kokonaisjärjestelmästä, ja tämä ohjaa kehittämistä oikeaan suuntaan. Esimerkiksi järjestelmäarkkitehtuuri voidaan ajatella metaforana, joka ohjaa kehitystä tiettyyn suuntaan.

Yksinkertainen suunnittelu (Simple design)

Suunnittelussa painotetaan yksinkertaisimman toteutettavissa olevan ratkaisun löytämistä. Tarkoituksena on täyttää käyttötarinalle asetetut vaatimukset, sekä läpäistä yksikkö- ja hyväksymistestit. Liiallinen monimutkaisuus ja ylimääräinen koodi poistetaan välittömästi esimerkiksi refaktoroinnin avulla. *Ohjelmakoodin yksinkertaistaminen* suoritetaan pariohjelmoinnissa siten, että toisen kirjoittaessa ohjelmaa, toinen analysoi kirjoitettua koodia ja ehdottaa parannuksia sekä kyseenalaistaa monimutkaista koodia (Abrahamsson *& al.*, 2002).

Testaaminen (Testing)

Kaikki ohjelmistokehitys on testiohjautuvaa, eli automaattiset yksikkötestit tehdään jo ennen ohjelmakoodia, ja niitä suoritetaan jokaisen käännöksen yhteydessä. Tällä varmistetaan, että koodivarastoon vietävä koodi on kääntyvää, ja sieltä otetun version jo toteutetut ominaisuudet toimivat oikein. Lisäksi jokainen kehitetty ominaisuus joutuu hyväksymistestiin, jonka testitapaukset ovat asiakkaan itse tai ohjelmoijien avustuksella luomia (Wake, 2000). Müllerin ja Padbergin (2003) mukaan jatkuva automaattinen testaaminen parantaa ohjelmiston laatua.

Refaktorointi (Refactoring)

Jotta ohjelmiston rakenteessa päästäisiin perusarvojen mukaiseen yksinkertaisuuteen, voidaan ohjelmakoodia tarvittaessa *refaktoroida*. Tämä tarkoittaa käytännössä ylimääräisen ja turhan koodin poistamista, yksinkertaisempien ratkaisutapojen hakemista tai kirjoitetun koodin muuta yksinkertaistamista, esimerkiksi järjestämällä koodia uudelleen. Refaktorointi parantaa koodin luettavuutta, ymmärrettävyyttä ja yksinkertaisuutta (Wake, 2000).

Pariohjelmointi (Pair programming)

Pariohjelmoinnissa kaksi ohjelmoijaa suorittaa yhdessä yhtä ohjelmointitehtävää siten, että toinen ohjelmoi, ja samalla toinen analysoi kirjoitettua koodia sekä työparin tekemiä ratkaisuja. Tarvittaessa analysoija antaa palautetta ja parannusehdotuksia kirjoittajalle. Näin koodista saadaan yksinkertaisempaa, ja turha monimutkaisuus saadaan karsittua pois (Wake, 2000). Müllerin ja Padbergin (2003) mukaan pariohjelmointi parantaa ohjelmoijien välistä kommunikaatiota, nopeuttaa kehittämistä sekä vähentää ohjelmakoodiin jääneiden virheiden määrää.

Yhteisomistus (Collective ownership)

Kaikki kehitetty ohjelmakoodi on kaikkien yhteisessä omistuksessa, eli jokainen pari voi muokata, päivittää ja parantaa koodivarastossa olevaa ohjelmakoodia milloin tahansa, jos huomaa parantamiselle tarvetta (Bowers *& al.*, 2007).

Jatkuva integraatio (Continuous integration)

Jotta koodivarastossa olisi aina kääntyvä ja toimiva versio, integroidaan kehitetty ohjelmakoodi heti valmistuttuaan siihen. Tätä käytäntöä kutsutaan *jatkuvaksi integraatioksi*, ja se tarkoittaa käytännössä sitä, että heti kun koodi on valmis, se integroidaan toimivaan versioon ja viedään koodivarastoon (Abrahamsson *& al.*, 2002). Tälle uudelle versiolle ajetaan kaikki integraatiotestit, jotka sen tulee läpäistä, jotta muutokset voidaan hyväksyä (Beck, 1999).

40-tuntinen työviikko (40-hour week)

Jotta työntekijät jaksavat paremmin tehdä työtänsä, heidän ei pitäisi työskennellä yli 40 tuntia viikossa tai tehdä liikaa ylitöitä. Jos ylitöitä joudutaan tekemään yli viikon verran, asiasta tulee ongelma, joka on ratkaistava. Liiallinen kiire ja ylityöt lisäävät stressiä, mikä puolestaan vähentää tiimin työtehoa, ja sitä kautta koko projektin läpimeno vaikeutuu (Wake, 2000).

Asiakas paikan päällä (On-site customer)

Jotta kommunikaatio asiakkaan ja projektiryhmän välillä olisi mahdollisimman tehokasta, olisi hyvä, että asiakas olisi koko ajan projektiryhmän ulottuvilla. Parhaiten tämä olisi saavutettavissa, jos asiakas on tarvittaessa paikan päällä vastaamassa kysymyksiin ja tekemässä päätöksiä liittyen järjestelmän kehittämiseen (Wake, 2000).

Ohjelmointikäytännöt (Coding standards)

Ohjelmakoodin luettavuuden ja yhteiskäyttöisyyden mahdollistamiseksi, koko projektitiimi käyttää yhtenäisiä ohjelmointikäytäntöjä. Näitä ovat esimerkiksi koodin muotoilusäännöt ja kommentointikäytännöt (Wake, 2000).

Avoimet työtilat (Open workspace)

Jotta kommunikaatio projektiryhmän sisällä olisi mahdollisimman tehokasta, työtilat voidaan järjestää avoimeksi siten, että suuri huone on jaettu pieniin, sermeillä erotettuihin työtiloihin, ja pariohjelmoijat ovat huoneen keskellä. (Beck, 1999).

Yleiset säännöt (Just rules)

Jotta projektiryhmän yhteiselämä sujuisi mutkattomasti, on yhteisöllä omat säännöt, joita jokainen noudattaa. Näitä sääntöjä ryhmä voi yhteisesti muuttaa, kunhan muutosten vaikutus on ensin arvioitu (Abrahamsson *& al.*, 2002).

3.3.3 Hyödyt ja haasteet

XP on kehitetty ohjelmistokehityksen malliksi pienille ja keskisuurille projektiryhmille, ratkaisemaan perinteisen suunnitelmaohjautuvan kehittämisen mukanaan tuomia ongelmia. Menetelmää on tieteellisestikin tutkittu, ja Abrahamssonin *& al.* (2002) mukaan suurin osa tutkimuksista kuvaa hyviä kokemuksia. Tällaisiin positiivisiin tuloksiin on Abrahamssonin mukaan päässyt mm. Anderson *& al.* (1998) ja Schuh (2001).

Müller ja Padberg (2003) ovat tutkineet XP:n vaikutuksia projektien kannattavuuden ja talouden kannalta. He havaitsivat, että vaikka pariohjelmointi lisäsikin henkilöstökuluja kaksinkertaiseksi, ja testiohjautuva kehittäminen hidasti ominaisuuksien kehittämistä, oli pariohjelmoinnin tuottavuus, ja sitä kautta myös kehitysnopeus, paljon parempi kuin perinteisen yksin ohjelmoinnin. Lisäksi pariohjelmoinnissa tuotettu ohjelmakoodi oli laadultaan parempaa, yksinkertaisempaa, ja siinä oli vähemmän virheitä kuin perinteisin menetelmin tehdyssä koodissa. Jatkuva yksikkötestaaminen paransi ohjelmakoodin laatua merkittävästi, kun virheet havaittiin paremmin ja nopeammin, ja ne voitiin korjata heti havaitsemisen yhteydessä.

He havaitsivat, että XP sopii tilanteisiin, jossa yrityksellä on kiire julkaista jokin järjestelmä tietyssä aikataulussa säilyttääkseen markkinaosuutensa, tai jos järjestelmä voidaan jakaa pienemmiksi osajulkaisuiksi. Inkrementaaliset pienet julkaisut mahdollistavat projektin laskutuksen iteraatioittain, jolloin yrityksen toiminta projektin aikana on taloudellisesti helpompaa.

Haasteina he näkivät esimerkiksi vaatimusmäärittely- tai korkean tason suunnitteludokumentaation puuttumisen. Kaikki järjestelmän ominaisuudet on kuvattu käyttötari-

noissa, eikä kokonaiskuvaa kokonaisuudesta ole välttämättä kenelläkään. Tämän vuoksi voi olla vaikeaa todentaa, täyttääkö valmis järjestelmä asiakkaan tarpeet (Müller & Padberg, 2003).

Myös Cockburn ja Williams ovat tutkineet tarkemmin pariohjelmoinnin tuomia hyötyjä. He havaitsivat, että pariohjelmoinnin tuloksena ohjelmoijat olivat tyytyväisempiä ja tuottivat selkeämpää ohjelmakoodia sekä parempia suunnitteluratkaisuja, kuin normaalisti. Jatkuva katselmointi vähensi virheitä, ja ohjelmoijat oppivat toisiltaan paljon. Lisäksi tiimin sisäinen yhteistyö parantui, ja tiimin jäsenet oppivat parempaa tiimityöskentelyä (Succi & Marchesi, 2001).

Müllerin ja Padbergin (2003) mukaan XP:n taloudellinen tehokkuus riippuukin loppujen lopuksi pariohjelmoinnilla saavutetun nopeuden ja virheiden vähenemisen määrästä.

4 Prosessin arvioinnin menetelmiä

Jotta ohjelmistoprosessien kattavuutta, tehokkuutta ja toimivuutta voitaisiin analysoida, käytetään ohjelmistoprosessien kehittämishankkeissa usein prosessin arvioinnin ja kehittämisen pohjana jotakin referenssimallia, kuten ISO 9001:2000. Nämä *referenssimallit* asettavat yleiset vaatimukset yrityksen laatujärjestelmälle ja ohjelmistoprosessille. Valitun referenssimallin tulisi tukea yrityksen liiketoimintasuunnitelmaa ja mahdollistaa siinä asetettujen tavoitteiden toteutuminen (Mutafelija & Stromberg, 2003).

Tässä luvussa on esitelty muutamia laajalti käytettyjä ohjelmistoprosessin arvioinnin menetelmiä, joita kutsutaan myös referenssimalleiksi tai *arviointikehyksiksi*. Ensin esitellään ISO 9001:2000, sen jälkeen CMMI ja lopuksi ISO/IEC 15504 (SPICE).

4.1 ISO 9001:2000

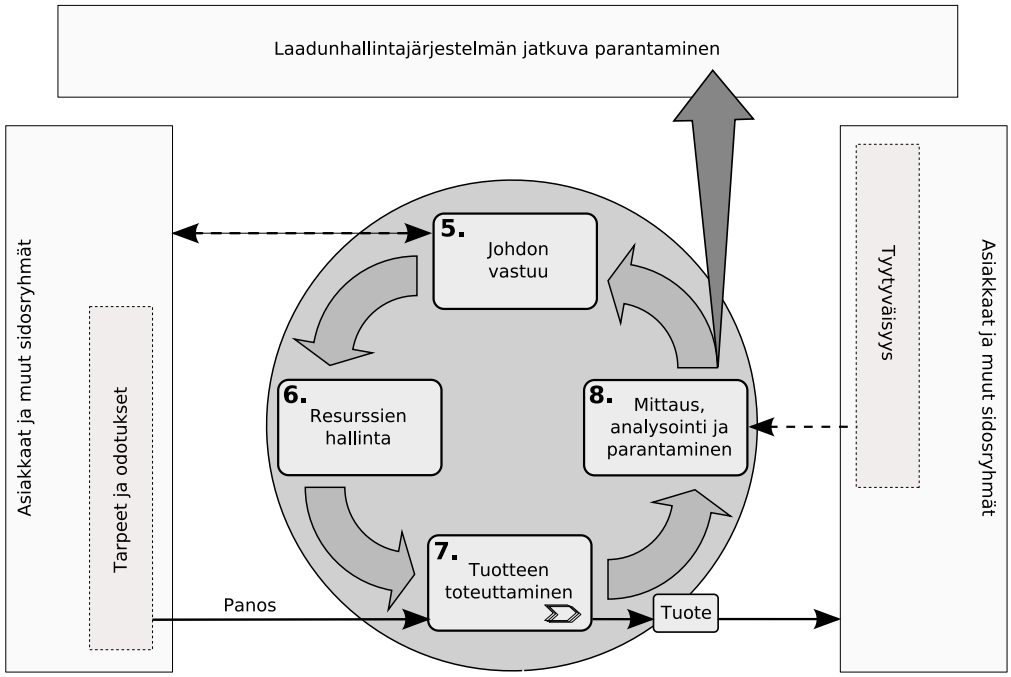
ISO 9001 on kansainvälisen standardointiorganisaatio ISO:n (the International Organization for Standardization) ylläpitämä kansainvälinen laadunhallinnan standardisarja, joka muodostuu useammasta toisiaan täydentävästä standardista. Tähän standardisarjaan olennaisimpina kuuluvat standardit ISO 9000:2000, 9001:2000 ja 9004:2000.

Johtuen siitä, että varsinaiset standardit ovat toimialariippumattomia ja varsin yleisellä tasolla, on näiden rinnalle tehty erilaisia toimialakohtaisia soveltamisohjeistuksia, kuten ISO 9000-3 ja British Standards Instituten TickIt!, jotka ovat kumpikin oppaita ISO 9001 -standardin soveltamiseen ohjelmistotuotannossa (Paulk, 1994).

Tärkein edellä mainituista standardeista ohjelmistotuotantoa harjoittaville yrityksille on soveltamisohjeistuksineen sertifioinninkin pohjana oleva ISO 9001:2000, joka kuvaa vaatimukset yrityksen prosessipohjaiselle laadunhallintajärjestelmälle (Mutafelija & Stromberg, 2003). Muut perheeseen kuuluvat standardit täydentävät tätä standardia siten, että ISO 9000:2000 sisältää laadunhallintajärjestelmien perusteet sekä sanaston, ja ISO 9004:2000 määrittää suuntaviivat suorituskyvyn jatkuvalle parantamiselle.

4.1.1 Standardin ydinkohdat

Laadunhallintajärjestelmän kehittäminen tämän standardiperheen ja ohjeistusten mukaisesti tarjoaa yritykselle paremman tavan organisoida, hallita, dokumentoida ja ohjata tekemiänsä asioita (Standards Australia, 2001). Standardi kattaa yrityksen kaikki prosessit ja toiminnot aina tuotteiden tai palveluiden suunnittelusta ja tuotannosta jakeluun saakka.



Kuva 5: Prosesseihin perustuvan laadunhallintajärjestelmän malli (Anttila & Vakkuri, 2001).

Kuvassa 5 on esitetty prosessipohjainen ja jatkuvaan laadun sekä prosessien parantamiseen tähtäävä laadunhallintajärjestelmän malli. Siinä on kuvattu ISO 9001:2000 laatustandardin viisi ydinkohtaa: laadunhallintajärjestelmä, johdon vastuu, resurssien hallinta, tuotteen toteuttaminen sekä mittaus, analysointi ja parantaminen (Anttila & Vakkuri, 2001). Keskellä oleva ympyrä kuvaa yrityksen toimintaa, ja siinä olevat standardin sisällysluettelon pääkohtien mukaisesti numeroidut laatikot peräkkäisessä järjestyksessä suoritettavia karkean tason ydinkokonaisuuksia.

Laadunhallintajärjestelmä

Standardin ensimmäinen ydinkohta - *laadunhallintajärjestelmä* - määrittää vaatimukset laadunhallintajärjestelmän perustamiselle, dokumentoinnille, ylläpitämiselle ja jatkuvalle parantamiselle (Hoyle, 2001). Haikalan ja Märijärven (2003) mukaan standardin keskeisiin laadunhallintajärjestelmän vaatimuksiin kuuluu, että kaikki prosessit sekä niiden syötteet ja tulokset on määritetty, dokumentoitu ja ohjeistettu. Lisäksi dokumenttien ja ohjeistusten on oltava ajan tasalla ja organisaation koko henkilöstön saatavilla.

Johdon vastuu

Yrityksen ylimmän johdon vastuulla on laadunhallintajärjestelmän rakentaminen, kehittäminen, julkaiseminen ja noudattaminen. Tämä vaatii johdolta vahvaa sitoutumista asiaan, jotta laadunhallintajärjestelmä ei olisi yrityksessä ainoastaan joukko erilaisista toimintaprosesseista kertovia dokumentteja (Mutafelija & Stromberg, 2003).

Johdon tulisikin määrittää yritykselle selkeät laadunhallintajärjestelmän tavoitteet, jotka linkittävät yrityksen liiketoimintasuunnitelman mukaisen mission ja vision yrityksen laatutavoitteisiin. Laatupolitiikan ansiosta yrityksen laadunhallintajärjestelmä tukee liiketoimintasuunnitelmaa, ja toiminnan jatkuva kehittäminen mahdollistuu (Standards Australia, 2001).

Resurssien hallinta

Jotta yritys voi toimia ja päästä tavoitteisiinsa, on sillä käytössään erilaisia resursseja, kuten henkilö-, materiaali- ja taloudellisia resursseja. Resurssienhallinta on yksi yritysten avainprosesseista. *Resurssienhallinta* pitää sisällään resurssisuunnittelun, uusien resurssien hankkimisen, sekä resurssien käytön, ylläpitämisen ja poistamisen käytöstä (Hoyle, 2001).

Tuotteiden toteuttaminen

Resurssien avulla yritys voi toteuttaa tuotteitansa, jotka voivat olla palveluita tai fyysisiä tuotteita. *Tuotteen toteuttamisen* prosessit kattavat kaiken asiakasvaatimuksien rea-

lisoimisesta tuotteeksi (Mutafelija & Stromberg, 2003). Standardin mukaan tuotanto voidaan nähdä prosessina, joka muodostuu alemman tason prosesseista ja alimmalla tasolla ovat yksittäiset tehtävät (Haikala & Märijärvi, 2003). Tuotteen toteuttamisen prosesseihin kuuluvat esimerkiksi tuotesuunnittelun, ostojen ja valmistuksen prosessit.

Jotta lopputulos täyttäisi asiakasvaatimukset, on sen laatu varmistettava tuotantoprosessin aikana erilaisilla *laadunvarmistustoimilla*, kuten katselmoinneilla ja testauksilla (Mutafelija & Stromberg, 2003). Tuotantoprosessissa laadunhallintajärjestelmän mukaisesta toiminnasta on jäätävä todisteita. Näitä todisteita ovat erilaiset tarkastuskertomukset, raportit ja dokumentaatio (Haikala & Märijärvi, 2003).

Mittaaminen, analysointi ja parantaminen

Yrityksen on ohjattava ja *mitattava* jokaista prosessiaan, jotta se pystyy takaamaan laadunhallintajärjestelmän mukaisen toiminnan (Haikala & Märijärvi, 2003). Hoylen (2001) mukaan tietoa kerätään prosessin aikana itse prosessista ja tuotantovaiheen jälkeen mukaan otetaan myös asiakastyytyväisyysmittaukset.

Kerättyä tietoa päästään *analysoimaan*, ja saatujen tulosten perusteella prosessia voidaan *parantaa* paremman tuotelaadun ja tehokkaampien prosessien saavuttamiseksi. Kertyneen mittaritiedon avulla nähdään myös, kuinka aiemmat parantamiset ovat vaikuttaneet prosessiin ja laatuun. Standardin mukaisesti prosessin parantamisen ohella parannetaan myös tarvittaessa koko laadunhallintajärjestelmää.

4.1.2 Standardin sovittaminen ohjelmistotuotantoon

ISO 9000-3 on ohjeistus ISO 9001 -standardin soveltamiseen ohjelmistotuotantoa harjoittaville yrityksille, jotka kehittävät, toimittavat, asentavat ja ylläpitävät ohjelmistoja (Coallier, 1994). Se perustuu olettamukselle, että noudattamalla määriteltyjä ja dokumentoituja prosesseja pystytään tuottamaan asiakkaille laadukkaampia ohjelmistotuotteita (Kehoe & Jarvis, 1996).

Ohjeistuksen sisältö muodostuu yleisellä tasolla kolmesta kokonaisuudesta. Se kuvaa kehyksen laadunhallintajärjestelmälle, ohjelmistoprosessin elinkaaren toiminnoille sekä prosessin tukitoiminnoille (Ben-Yaacov, 1995). Näiden kohtien sisältö voidaan jakaa tarkentamaan kutakin aiemmin kuvattua ISO 9001 -standardin pääkohtaa.

Laadunhallintajärjestelmä

ISO 9000-3 kuvaa tarkemmat vaatimukset ohjelmistotuotantoa harjoittavan yrityksen laadunhallintajärjestelmälle. Sen mukaan yrityksen tulisi täydentää laadunhallintajärjestelmän dokumentaatio koskettamaan myös ohjelmistotuotannon toimintoja, esimerkiksi tunnistamalla, määrittämällä ja dokumentoimalla ohjelmistokehityksen ja ylläpidon prosessinsa (ISO, 2003a). Paulk (1994) esittääkin laadunhallintajärjestelmän olevan integroitu osa yrityksen elinkaarimallin toimintoja.

Sen lisäksi, että laatudokumentaatio on olemassa, tulee sen käytöstä ja tehokkuudesta olla näyttöä, kuten erilaisia dokumentoituja testituloksia, muutospyyntöjä ja katselmointiraportteja (ISO, 2003a). Näitä varsinaisen laatukäsikirjan ja ohjeistusten ulkopuolisia laadunhallintaan liittyviä materiaaleja kutsutaan yhteisesti *laatutietokannaksi*.

Haikalan ja Märijärven (2003) mukaan yrityksen laadunhallintajärjestelmän dokumentaatio voidaan jakaa neljään eri tasoon. Ylimmällä tasolla on varsinainen *laatukäsikirja*, joka sisältää esimerkiksi yrityksen toiminnan yleiskuvauksen ja laatupolitiikan kuvauksen. Laatukäsikirjassa otetaan kantaa kuhunkin standardin kohtaan, mutta sen luonne on enemmänkin korkean tason dokumentti, josta viitataan tarkemman tason erillisohjeisiin ja muihin dokumentteihin. Toisen tason muodostavat eri työvaiheiden suorittamisen ohjeistavat työohjeet, kuten esimerkiksi erilaiset määrittely-, suunnittelu- tai katselmointiohjeet. Nämä voivat olla erillisinä dokumentteina yrityksen laadunhallintajärjestelmässä. Kolmannella tasolla ovat eri työvaiheissa sovellettavat dokumentointikäytännöt, standardilomakkeet ja dokumenttipohjat. Alimmalla tasolla Haikalan ja Märijärven mukaan ovat *viitemateriaalit*, kuten työvälineiden manuaalit.

Johdon vastuu

Johdon vastuun osalta ohjeistus tarkentaa ISO 9001 -standardia ainoastaan laadunhallintajärjestelmän suunnittelun ja sisäisten auditointien osalta. Siinä luetellaan muutamia yleisimpiä ohjelmistotuotantoa harjoittavan yrityksen laadunhallintajärjestelmän kehittämisen tehtäviä. Tällaisia tehtäviä ovat esimerkiksi prosessin tuotosten, käytettävien työkalujen sekä käytettävän elinkaarimallin määrittäminen.

Kehittämisen lisäksi laadunhallintajärjestelmää ja sen tehokkuutta asetettuja laatutavoitteita vastaan tulisi mitata ja arvioida sisäisillä auditoinneilla sekä ulkopuolisen arvioijan toimesta sopivin väliajoin. Esimerkiksi kohdassa 4.3 kuvattu standardi ISO/IEC 15504 (SPICE) tarjoaa yhden tällaisen laadunhallintajärjestelmän arviointikehyksen. Sisäisiä auditointeja tai hyväksymistestauksia tulisi suorittaa myös kaikille tuotteille, erilaisten laatu- ja asiakasvaatimusten täyttymisen varmistamiseksi (ISO, 2003a).

Resurssienhallinta

Jotta ohjelmistoyritys voi toimia ja tuottaa asiakasvaatimukset täyttäviä tuotteita tai palveluita, täytyy sen hallita resurssejansa tehokkaasti. ISO 9000-3 tarkentaa ISO 9001 -standardia henkilöresurssien koulutustarpeiden ja käytettävien teknologioiden ja menetelmien arvioinnin osalta. Sen mukaan henkilöstön tulisi osata projektissa käytettävät tekniikat ja menetelmät, kuten vaatimusten keruu tai tietyn ohjelmointikielen hallinta. Tarvittaessa henkilöstöä on koulutettava valittuihin tekniikoihin ja menetelmiin. Tämän lisäksi resurssienhallintaan kuuluu ohjeistuksen mukaan käytettävien ohjelmointityökalujen, ohjelmistokirjastojen ja muiden infrastruktuuriin kuuluvien tekijöiden hallitseminen (ISO, 2003a).

Tuotteiden toteuttaminen

Merkittävin ero ohjelmistotuotantoa harjoittavan yrityksen toiminnassa verrattuna jonkin toisen toimialan yritykseen on tuotteiden toteuttamisen prosesseissa. Tämän vuoksi tuotteiden toteuttamisen prosessit ovatkin ISO 9000-3 ohjeistuksen pääsisältöä.

Tuotteen toteutusprosessi aloitetaan tuotteen toteuttamisen suunnittelusta. Yrityksen on suunniteltava ja kehitettävä tuotekohtainen toteutusprosessi tuotteen toteuttamista varten. Se voi olla sovitus jonkun toisen samankaltaisen tuotteen toteutusprosessista tai yksilöllinen tuotekohtainen prosessi. Tuotteen toteutusprosessin pitäisi pohjautua johonkin elinkaarimalliin, ja sen valinnassa tulisi ottaa huomioon mm. projektin koko, monimutkaisuus, turvallisuusvaatimukset ja riskit.

Elinkaarimallin lisäksi tuotteen toteutusta suunniteltaessa laaditaan *laatusuunnitelma*. Se kuvaa kuinka laadunhallintajärjestelmää sovitetaan tätä tuotetta kehitettäessä. Laatusuunnitelma voidaan kirjoittaa itsenäisenä dokumenttina tai se voi olla osana esimerkiksi projektisuunnitelmaa. Tällaisia sovitusohjeita ovat esimerkiksi laatuvaatimukset tuotteelle ja prosesseille sekä valitut muutosten- ja kokoonpanonhallinnan menetelmät.

ISO 9000-3 ohjeistus tarkentaa standardia sen osalta, että tuotteen vaatimukset tulisi määritellä ja katselmoida yhdessä asiakkaan tai loppukäyttäjien kanssa. Tämän lisäksi asiakkaan tulisi hyväksyä vaatimukset ennen toteutuksen aloittamista, jotta päästään parempaan asiakastyytyväisyyteen ja odotettuun lopputulokseen.

Ennen tuotantosopimuksen solmimista yrityksen tulisi arvioida määritettyjen vaatimusten toteutettavuus. ISO 9000-3 kuvaa joukon tässä yhteydessä arvioitavia asioita, kuten esimerkiksi määriteltyjen vaatimusten toteutettavuus ja käyttökelpoisuus, käytettävät suunnittelu- ja kehitysmenetelmät sekä riskienhallinta (ISO, 2003a).

Kun asiakkaan vaatimukset on saatu tunnistettua ja dokumentoitua, aloitetaan projektisuunnittelu, joka tuottaa *projektisuunnitelman* (development plan). Tämä dokumentti kuvaa, kuinka kehitetään asiakkaan vaatimukset täyttävä tuote. Projektisuunnitelmassa kuvataan esimerkiksi kussakin vaiheessa käytettävät resurssit, aikataulutus, kunkin vaiheen syötteet ja tuotokset sekä tuotosten verifiointimenetelmät. Tällaisia *verifiointimenetelmiä* ovat esimerkiksi vaihetuotteiden testaaminen ja erilaiset katselmointikäytännöt (Kehoe & Jarvis, 1996).

ISO 9000-3 kuvaa ohjelmistotuotannon asettamia vaatimuksia varsinaisen toteutuksen (design and development) vaiheiden syötteille ja tuotoksille sekä niiden laadun varmis-

tamiselle. Ohjeistuksen mukaan vaihesyötteet tulisi analysoida ennen vaihetta esimerkiksi vaatimusten toteutettavuuden ja epätarkkuuksien varalta. Vaihesyötteiden analysoinnin lisäksi ohjeistuksessa on lueteltu esimerkkinä muutamia ohjelmistotuotannolle ominaisia vaihetuotteita, kuten suunnittelu- ja testausdokumentaatio, arkkitehtuurimalli, valmis tuote sekä käyttöohjeet (ISO, 2003a).

Näiden syötteiden ja vaihetuotteiden laadunvarmistus tulisi suorittaa katselmointien, verifioinnin ja validoinnin kautta. Katselmoinneissa varmistetaan esimerkiksi, että valittuja menetelmiä käytetään oikein (Kehoe & Jarvis, 1996). Katselmoinnit käsittelevät tuotoksia ja prosessia yleisemmällä tasolla, ja varsinainen vaihetuotteen laadun varmistaminen tapahtuu verifiointien ja validointien kautta. *Verifioinnissa* varmistetaan, että vaiheen tuotos vastaa vaihesyötteenä saatuja, sille asetettuja vaatimuksia. *Validoinnissa* puolestaan varmistetaan, että vaihetuote vastaa käyttötarkoitustaan. Ohjeistuksen mukaan vaihetuotteiden validointi tarkoittaa käytännössä niiden testaamista esimerkiksi erilaisilla yksikkö- ja integraatiotesteillä. Valmiille lopputuotteelle tulisi suorittaa yhdessä asiakkaan kanssa hyväksymistesti, jossa nähdään täyttääkö valmis lopputuote alkuperäiset asiakasvaatimukset (ISO, 2003a).

Tuotteen toteuttamisen prosessiin standardissa kuuluu myös muutostenhallinta, mutta ISO 9000-3 sijoittaa ne osaksi kokoonpanonhallintaa, jota käsitellään myöhemmin.

Kehitysprojektissa voidaan joutua käyttämään ostettuja komponentteja, työkaluja tai palveluita, kuten alihankintaresursseja. Tämän vuoksi ISO 9000-3 tarkentaa ostoprosesseja ostettavien tuotteiden tai palveluiden hallinnan, tuotetietojen ja verifioinnin osalta. Se luettelee muutamia yleisimpiä ostettavia tuotteita, kuten esimerkiksi valmisohjelmistot, räätälöidyt ohjelmistot ja alihankintaostot sekä ohjeistaa standardin vaatimien tuotetietojen sovittamisen ohjelmistotuotteeseen.

Huomattava seikka on, että ISO 9000-3 käsittelee avoimen lähdekoodin (open source) ilmaisia sovelluksia tai kirjastoja ostotuotteina ja ne tulisi käsitellä saman prosessin kautta kuin muutkin järjestelmäostot. Jos avoimen lähdekoodin kirjastoja tai ohjelmistoja käytetään osana tuotetta, tulisi niitä käsitellä tuotannossa samalla tavalla kuin mitä tahansa yrityksen itse kehittämää ohjelmistokomponenttia. (ISO, 2003a)

Kaikki ostetut tuotteet tulisi verifioida ennen käyttöönottoa esimerkiksi hyväksymistestauksen avulla. Jos tällainen hyväksymistestaus ei ole mahdollista, tulisi varmistaa, että ostettu tuote on testattu toimittajan toimesta ennen hankinnan tekemistä (ISO, 2003a).

Kun tuote on valmis, sille järjestetään hyväksymistestaus, jonka jälkeen tuote saadaan julkaistavaksi. ISO 9000-3 tarkentaa standardia tuotteen julkaisu- ja toimitustoimenpiteiden sekä toimituksen jälkeisten toimenpiteiden osalta. Ohjeistuksen mukaan julkaisutoimenpiteisiin kuuluvat käännökset, julkaisut ja replikointi. Julkaisun yhteydessä suoritetaan toimitus, johon kuuluvia toimenpiteitä ovat esimerkiksi toimitus ja asentaminen. Toimituksen jälkeen aloitetaan ylläpito ja perustetaan asiakasta varten tekninen- ja käyttötuki, joka voidaan toteuttaa esimerkiksi help desk -puhelinpalveluna. Myös nämä prosessit tulisi katselmoida ja validoida niiden soveltuvuus tarkoitukseensa, jotta asiakastyytyväisyyttä saataisiin parannettua (ISO, 2003a).

Jotta kokoonpanonhallinta olisi mahdollista, tulisi tuote kehittää siten, että kukin sen osakomponentti on tunnistettavissa ja kaikki siihen tehdyt ominaisuudet ja muutokset on jäljitettävissä määrityksiin ja vastaaviin muutospyyntöihin. ISO 9000-3 tarkentaa standardia kokoonpanonhallinnan, jäljitettävyyden ja tunnistettavuuden sekä muutostenhallinnan osalta. Sen mukaan jäljitettävyys ja tunnistettavuus saadaan toteutettua oikean kokoonpanonhallinnan avulla (ISO, 2003a).

Projektin aikana voidaan tarvita asiakkaan toimittamia dokumentteja, laitteita tai ohjelmistoja. Ohjeistus listaa muutamia mahdollisia asiakkaan toimittamia asioita, kuten ohjelmistotuotteet, kehitystyökalut tai rajapintamääritykset. Näitä asiakkaan toimittamia materiaaleja tulisi säilyttää yhtä huolellisesti ja samalla tavalla kuin omia vastaavia materiaaleja. ISO 9000-3 kuvaa tuotteen säilyttämiseen liittyviä asioita, jotka tulisi ottaa huomioon omia tai asiakkaan materiaaleja säilytettäessä. Se ottaa kantaa esimerkiksi ohjelmistotuotteen säilyttämiseen, tietosuojaan ja varmuuskopioihin (ISO, 2003a).

Projektin aikana tulisi mitata ja monitoroida laadunhallintajärjestelmän toimivuutta ja tehokkuutta, sekä prosessien ja tuotteen laatua. Standardin mukaan tukitoimiin kuuluu myös näissä mittauksissa käytettävien mittaus- ja monitorointilaitteiden hallinta ja ka-

librointi. Koska ohjelmistotuotannossa mittaaminen ja monitorointi eivät yleensä vaadi laitteita, tarkentaa ISO 9000-3 näiden tarkoittavan ohjelmistotuotannossa esimerkiksi testisyötteitä, simulaatio- tai muita testauksen ohjelmistotyökaluja (ISO, 2003a).

Mittaaminen, analysointi ja parantaminen

Laadunhallintajärjestelmää sovelletaan projektikohtaisesti valitsemalla esimerkiksi soveltuva elinkaarimalli ja sen mukana vaihesyötteet ja -tuotokset. Tämä sovellusohje on kuvattu projektisuunnitelmassa tai erillisessä laatusuunnitelmadokumentissa. Mittaamiseen ja analysointiin kuuluu tämän sovitetun laatusuunnitelman ja sen toteutuneen laadun sisäinen katselmointi aika ajoin verrattuna laadunhallintajärjestelmään (ISO, 2003a).

Kehoen ja Jarvisin (1996) mukaan toimittajalla tulisikin olla kvantitatiiviset mittarit tuotanto- ja toimitusprosesseille. Näiden mittareiden tulisi kuvata, kuinka hyvin kehitysprosessi on viety läpi verrattuna asetettuihin aikataulu- ja laatutavoitteisiin, sekä kuinka tehokkaasti prosessi vähentää virheiden määrää tuotteessa ennen ja jälkeen julkaisun. Kun mittaritietoa kerätään ja raportoidaan säännöllisesti, voidaan tunnistaa suorituksen taso kullakin mittarilla. Jos mitatut arvot alittavat etukäteen sovitut alarajat, voidaan suorittaa korjaavia toimenpiteitä. Saman mittaritiedon pohjalta voidaan lisäksi perustaa tavoitteita laadunparantamishankkeille.

Prosessien lisäksi myös tuotteen laatua tulisi mitata ja monitoroida. Käytettyjen mittareiden tulisi olla olennaisia tuotettavalle tuotteelle, ja niiden tulosten tulisi olla vertailukelpoisia eri tuotteiden vastaavien mittareiden kanssa. Tällaisia mittareita ovat Kehoen ja Jarvisin (1996) mukaan esimerkiksi havaittujen virheiden määrä, vakavuus ja keskimääräinen korjaamiseen käytetty aika.

Mittaamiseen ja monitorointiin kuuluu varsinaisten prosessien, laadunhallintajärjestelmän ja ohjelmistotuotteen laadun mittaamisen ohella myös asiakastyytyväisyyden mittaaminen. ISO 9000-3 tarkentaa keinot mitata ohjelmistotuotantoa harjoittavan yrityksen asiakastyytyväisyyttä. Se voidaan suorittaa esimerkiksi mittaamalla *käytönaikaista laatua* (quality-in-use) erilaisilla mittareilla asiakkaiden suoran ja epäsuoran palautteen

pohjalta (ISO, 2003a). Tällaisia käytönaikaisen laadun mittareita on kuvattu esimeriksi teknisessä raportissa ISO/IEC TR 9126-4 (ISO, 2001b).

Jos tuotteen kehittämisessä käytetään valmiita kirjastoja tai ohjelmistoja, jotka eivät täytä standardin ja ohjeistuksen vaatimuksia, tulisi niiden käyttöä valvoa erityisen tarkasti ja mieluiten pyrkiä korjaamaan standardin vastaiset ominaisuudet pois tai olla käyttämättä niitä (ISO, 2003a).

Standardin vaatima jatkuva laadun parantaminen on mahdollista kerätyn mittaritiedon analysoinnin ja sen pohjalta suunniteltujen korjaavien toimenpiteiden kautta. ISO 9000-3 suosittelee, että jatkuvasta parantamisesta tehtäisiin yksi koko elinkaaren mittainen tukiprosessi, joka kulkee varsinaisten tuotantoprosessien rinnalla koko ajan.

4.1.3 ISO 9001 -sertifiointi

Sen lisäksi, että ISO 9001 tarjoaa ohjeistuksen ja vaatimukset yrityksen laadunhallintajärjestelmälle, ja sitä kautta myös ohjelmistoprosessille, voidaan yrityksen laadunhallintajärjestelmä sertifioida *ISO 9001 -sertifikaatilla*.

Sertifiointi tarkoittaa yrityksen laadunhallintajärjestelmän virallista tutkimista ja arviointia (assessment), jonka tarkoituksena on selvittää, täyttääkö yrityksen toiminta valittujen prosessien osalta standardin vaatimukset (Haikala & Märijärvi, 2003). Coallierin (1994) mukaan ainoa merkityksellinen sertifikaatti ohjelmistoalan yritykselle on ISO 9001/9000-3.

Sertifikaatti voidaan hakea koko yritykselle tai jollekin sen osalle. Sertifiointeja suorittavat auktorisoidut *sertifiointiorganisaatiot*, kuten Suomen Standardoimisliitto SFS ja Det Norske Veritas DNV (Haikala & Märijärvi, 2003). Sertifioinnin suorittavat ISO 9001 -standardeihin koulutetut arvioijat, mutta ongelmaksi Paulkin (1995) mukaan voi nousta heidän kokemattomuutensa ja vähäinen tietämyksensä ohjelmistotuotantoon liittyvistä asioista. Tämän vuoksi arvioijissa tulisi olla vähintään yksi, jolla on tarkempaa tietämystä ohjelmistotuotannosta.

Yrityksen sertifiointiprosessi muodostuu Ben-Yaacovin (1995) mukaan seuraavista askeleista. Yritysjohdon on oltava sitoutunut kehittämään laadunhallintajärjestelmää ja hankkimaan sertifikaatin. Tätä varten on perustettava erityinen *ISO-ohjausryhmä*. Ensimmäiseksi arvioidaan nykyinen laadunhallintajärjestelmä ja valmistellaan ISO-hankkeen toteutussuunnitelma. Tässä vaiheessa voidaan käyttää myös ulkoisia asiantuntijoita. Tämän jälkeen koulutetaan yritysjohto, henkilöstö ja sisäiset auditoijat sekä toteutetaan ja dokumentoidaan uudet prosessit. Kun toteutus, dokumentointi ja koulutus on saatu tehtyä, aloitetaan uusien prosessien, dokumentaatioiden ja sisäisten auditointien käyttäminen.

Kun uudet käytännöt on saatu käyttöön ja niiden käytöstä alkaa olla todisteita, valitaan jokin sertifiointeja suorittava sertifiointiorganisaatio ja jatketaan edelleen uusien prosessien, dokumentaatioiden ja sisäisten auditointien käyttämistä. Kun uudet menetelmät on omaksuttu ja sisäistetty riittävän hyvin, kutsutaan sertifiointiorganisaatio suorittamaan ennakkoarviointi sertifiointia varten (Ben-Yaacov, 1995). Haikalan ja Märijärven (2003) mukaan tässä *ennakkoarvioinnissa* käydään läpi samat asiat kuin varsinaisessakin *arvioinnissa*, jotta mahdolliset puutteet laadunhallintajärjestelmässä saadaan poistettua jo ennakkoon. Ennakkoarvioinnin jälkeen suoritetaan korjaavat toimenpiteet, ja niiden jälkeen suoritetaan varsinainen arviointi (Ben-Yaacov, 1995).

Kun yritys saa sertifikaatin, se voi julkistaa sertifiointinsa esimerkiksi mainosmateriaaleissaan ja lehtiartikkeleissaan, mutta ei kuitenkaan tuotepakkauksissaan (Ben-Yaacov, 1995). Saavutettuaan sertifikaatin, yrityksen täytyy todistetusti toimia standardin mukaisesti. Sertifikaatin voimassa pysyminen edellyttää uusinta-arviointeja tavallisesti muutamia kertoja vuodessa. Jos jossakin tällaisessa uusinta-arvioinnissa havaitaan, ettei yritys toimikaan dokumentoimallaan ja väittämällään tavalla, menettää se sertifikaattinsa (Haikala & Märijärvi, 2003). Tämä tarkoittaa käytännössä suuren työn valumista hukkaan, koska Ben-Yaacovin (1995) mukaan sertifiointiprosessi on pitkä ja kallis prosessi ja voi kestää 6-18 kuukautta.

4.1.4 Hyödyt ja haasteet

Standardinmukaisen laadunhallintajärjestelmän kehittäminen ja käyttöönotto tuo mukanaan monia haasteita, mutta myös merkittäviä hyötyjä. Ben-Yaacovin (1995) mukaan sertifioidun laadunhallintajärjestelmän vaatiminen toimittajalta yleistyy koko ajan. Hän listaa artikkelissaan elokuussa 1993 suoritetun Yhdysvaltalaisen tutkimuksen tuloksia. Tutkimuksessa kysyttiin ISO 9000 -sertifioinnin mukanaan tuomia hyötyjä, ja mukana oli 620 Yhdysvaltalaista yritystä.

Lähes puolet (32,4%) vastanneista koki parantaneensa dokumentaation tasoa. Edelleen reilu neljännes (25,6%) oli tullut tietoisemmaksi laatuasioista. Muita koettuja sisäisiä hyötyjä olivat positiivinen muutos yrityskulttuuriin (15%), tuottavuuden ja tehokkuuden paraneminen (9%) sekä sisäisen kommunikaation paraneminen (7,3%).

Ulkoisina hyötyinä reilu kolmannes (33,5%) kertoi laadun paranemisesta, ja reilu neljännes (26,6%) kertoi asiakastyytyväisyyden parantuneen. Kilpailuasemien paraneminen oli havaittu noin viidenneksessä (21,6%) yrityksistä. Tämän lisäksi asiakkaan tekemien auditointien määrä oli pudonnut noin kymmenesosassa (8,5%) vastanneista yrityksistä, ja noin viisi prosenttia vastanneista kertoi parantaneensa markkinaosuuksiaan paremman laadun ja sertifioinnin ansiosta.

Corbett *& al.* (2002) tutkivat ISO 9000 -sertifioinnin vaikutuksia yritysten toimintaan. Heidän mukaansa ISO 9000 -sertifiointi tuo mukanaan merkittävän määrän prosessien suunnittelu-, toteutus- ja dokumentointityötä, joista muodostuu merkittävästi kuluja palkkojen, konsulttipalkkioiden ja sertifiointimaksujen myötä. Standardiluonteensa mukaisesti ISO 9001 on erittäin vaikealukuinen, ja sen tulkitsemiseen tarvitaan erilaisia ohjeistuksia, kirjallisuutta ja konsultteja. Onneksi ISO 9000-3 tarkentaa standardin vaatimuksia ohjelmistotuotantoon. Tarvittavan koulutuksen määrä on joka tapauksessa merkittävä.

He havaitsivat kuitenkin, että huolimatta sertifioinnin tuomista lisäkuluista ja -työstä, sillä oli positiivinen vaikutus yrityksen talouteen. Jo ISO 9001 -sertifikaatin hankkimisprosessi ennen varsinaista sertifiointia paransi yrityksen laatua ja taloudellista kan-

nattavuutta, parantuneen laadun ja suorituskyvyn myötä, verrattuna yrityksiin, jotka eivät tavoitelleet sertifikaattia. Sertifioinnin jälkeen tilanne pysyi kohtuullisen vakaana samalla tasolla, ja he esittävätkin artikkelissaan, että ISO 9001 -sertifioitu laadunhallintajärjestelmä onkin enemmän suorituskykyä ylläpitävä järjestelmä, kuin varma tie jatkuvaan laadun ja tuottavuuden parantumiseen.

Tästä voidaan vetää johtopäätös, että huolimatta siitä, että standardia noudattavan laatujärjestelmän kehittäminen ja sen sertifiointi on aikaa vievää ja kallista, ovat sen tuomat hyödyt merkittäviä. Ennen sertifiointia suoritetaan varsinainen suurempi laadun parantaminen, ja sertifioinnin jälkeen pidetään saavutettua suoritustasoa yllä ja sitä pyritään jatkuvasti parantamaan.

Lisääntyneen työmäärän ja korkeiden toteutuskustannusten vuoksi ISO 9001 -laatujärjestelmää ei kuitenkaan voi suositella pienille aloittaville ohjelmistoyrityksille.

4.2 CMMI

Capability Maturity Model-Integrated, eli *CMMI* on Software Engineering Instituten (SEI) kehittämä yrityksen tuotekehityksen *kypsyysmalli*. Sen avulla voidaan mitata organisaation tai sen osan kypsyystasoa ja prosessien kyvykkyyttä. Mallia voidaan käyttää esimerkiksi toimittajan arvioinnissa. Saadut arviointitulokset toimivat kehittämisen ja laadunparantamisen pohjana esimerkiksi projektissa, yksikössä tai koko organisaatiossa (Chrissis *& al.*, 2006).

CMMI-kypsyysmallista on tehty sovitukset tuotekehityksen (SEI, 2006) ja hankinnan (SEI, 2007) prosessien arviointiin ja kehittämiseen. Tulossa on myös sovitus palveluprosesseja varten (Chrissis *& al.*, 2006). Tässä tutkielmassa on kypsyysmallia käsitelty tuotteen kehittämisen sovituksen kautta.

CMMI-kypsyysmallia voidaan noudattaa kahdella tavalla, joko jatkuvana (continuous representation) tai vaiheittaisena (staged representation). Ensin mainittua lähestymista-

paa käytetään, kun halutaan parantaa yhtä tai useampaa prosessialuetta. Tällöin eri prosesseja voidaan kehittää eri tahtiin yrityksen liiketoimintatavoitteiden asettaman prioriteetin mukaisesti. Vaiheittaisessa lähestymistavassa parannetaan organisaation kypsyystasoa järjestelmällisesti kehittämällä tiettyä *kypsyystason* mukaista prosessijoukkoa kerrallaan. Kypsyystason noustessa saadaan mallista aina uusi joukko prosesseja kehitettäväksi. Tällä tavoin saadaan kehitettyä yrityksen menetelmiä kokonaisuutena verrattuna jatkuvan lähestymistavan yksittäisten prosessien parantamiseen (Chrissis *& al.*, 2006).

4.2.1 Prosessialueet

CMMI jakaa yrityksen prosessit neljään *prosessikategoriaan* ja edelleen 22 prosessialueeseen. *Prosessialueet* ovat yhteenkuuluvien käytäntöjen ryhmiä, jotka yhdessä toteutettuna täyttävät tietyn osa-alueen prosessin parantamisen tavoitteet. Prosessialueet on ryhmitelty prosessinhallinnan (Process Management), projektinhallinnan (Project Management), tuotannon (Engineering) ja tukiprosessien (Support) kategorioihin.

Prosessialueita ovat esimerkiksi prosessinhallinnan kategoriaan kuuluva organisaation koulutus (OT Organizational Training), projektinhallinnan kategoriaan kuuluva projektisuunnittelu (PP Project Planning), tuotannon kategoriaan kuuluva vaatimusten hallinta (REQM Requirements Management) sekä tukiprosessien kategoriaan kuuluva konfiguraationhallinta (CM Configuration Management).

Kukin prosessialue on kuvattu CMMI-kypsyysmallissa hyvin yksityiskohtaisesti. Prosessialueeseen kuuluu joukko komponentteja, jotka on jaettu vaadittuihin (required), odotettuihin (expected) ja informatiivisiin (informative) komponentteihin.

Vaaditut komponentit kuvaavat, mitä yrityksen tulee saavuttaa täyttääkseen kyseisen prosessialueen vaatimukset. Vaadittuja komponentteja ovat prosessikohtaiset spesifiset ja yleiset kaikille prosessialueille yhteiset tavoitteet. Näiden tavoitteiden todistettu täyttyminen yrityksen prosesseissa ja toiminnassa on myös CMMI-arviointien pohjana. Esimerkkinä spesifisistä tavoitteista on projektisuunnittelun prosessialueeseen kuu-

luva projektisuunnitelman tekeminen. Yleisiä tavoitteita ovat esimerkiksi hallitun ja määritellyn prosessin perustaminen.

Odotetut komponentit kuvaavat, mitä organisaatio voi tehdä täyttääkseen vaaditun komponentin mukaiset vaatimukset. Niitä voidaan käyttää prosessien parantamisen pohjana. Odotettuja komponentteja ovat spesifiset ja yleiset käytännöt, jotka toteuttamalla prosessialueen vaaditun komponentin tavoitteet täyttyvät. *Spesifisiä käytäntöjä* ovat esimerkiksi projektisuunnittelun prosessialueeseen kuuluva projektin elinkaarimallin määrittäminen ja projektin resurssisuunnittelu. *Yleisen tason käytäntöihin* kuuluu esimerkiksi prosessin suunnitteleminen.

Informatiiviset komponentit antavat lisätietoa vaadittujen ja odotettujen tavoitteiden täyttämisen toimenpiteistä. Tällaista lisätietoa ovat esimerkiksi alikäytännöt, työvaiheiden tuotokset ja kirjallisuusviitteet. Informatiivisiin komponentteihin kuuluu esimerkiksi prosessialueen olemassa olon perusteleva tavoitelauseke (Chrissis *& al.*, 2006).

4.2.2 Prosessien kypsyys ja kyvykkyys

CMMI-malli käyttää prosessien kypsyyden ja kyvykkyyden arviointiin viisi- tai kuusiportaista asteikkoa. Arviointiasteikon määrittää valittu lähestymistapa, eli joko jatkuvan parantamisen tai vaiheittaisen parantamisen lähestymistapa.

Jatkuvan parantamisen lähestymistapa

Jatkuvan parantamisen lähestymistavan tarkoituksena on parantaa valittua joukkoa yksittäisiä prosessialueita. Tällöin asteikolla arvioidaan yksittäisten prosessialueiden *kyvykkyyttä*, eli niille asetettujen yleisten tavoitteiden täyttymistä. Kyvykkyyttä mitataan kuusiportaisella asteikolla (0 - Incomplete, 1 - Performed, 2 - Managed, 3 - Defined, 4 - Quantitatively Managed, 5 - Optimizing). Tässä lähestymistavassa voidaan valita prosessialueiden joukosta kehitettävät alueet vapaasti.

Nollatasolla olevaa prosessialuetta ei suoriteta ollenkaan, tai se suoritetaan ainoastaan osittain. Käytännössä yhtä tai useampaa spesifistä tavoitetta ei ole täytetty.

Ensimmäisellä kyvykkyystasolla oleva prosessi täyttää kaikki spesifiset tavoitteensa, ja lisäksi sen mukaisia toimenpiteitä suoritetaan todistetusti yrityksen toiminnassa.

Toisella kyvykkyystasolla oleva prosessi on suunniteltu, ohjeistettu, ja sen työvaiheiden tuotokset ovat hallittuja.

Kolmannen kyvykkyystason saavuttaminen vaatii, että prosessi on määritetty sekä hallittu, ja tämän lisäksi sen tulee olla räätälöity organisaation *standardiprosesseista*. Toisin sanoen, yrityksellä täytyy olla joukko määritettyjä vakioprosesseja, joista käytettävä prosessi räätälöidään projektikohtaisesti.

Neljännen kyvykkyystason prosessialueen tulee olla hallittu kvantitatiivisten ja tilastollisten menetelmien avulla. Käytännössä yrityksen on kerättävä mittaritietoa ja analysoitava prosessien suorituskykyä.

Ylimmällä, eli viidennellä tasolla olevan prosessialueen suorituskykyä tulee mitata jatkuvasti kvantitatiivisten ja tilastollisten menetelmien avulla. Kerättyä mittaritietoa tulee analysoida, ja analysoidun tiedon perusteella etsiä mittaritiedosta poikkeamia, ja poistaa niiden syitä prosessista tarvittaessa. Tavoitteena on saavuttaa mahdollisimman vakaat, luotettavat ja toimivat prosessit jatkuvan kehittämisen avulla (Chrissis *& al.*, 2006).

Vaiheittaisen parantamisen lähestymistapa

Vaiheittaisen parantamisen lähestymistavassa arvioidaan koko organisaation *kypsyyttä* yleisten ja spesifisten tavoitteiden täyttymisen kautta. Organisaation kypsyyttä mitataan viisiportaisella asteikolla (1 - Initial, 2 - Managed, 3 - Defined, 4 - Quantitatively Managed, 5 - Optimizing). Vaiheittaisessa lähestymistavassa kukin taso tuo mukanaan joukon prosessialueita, joiden yleiset ja spesifiset vaatimukset yrityksen on täytettävä saavuttaakseen kyseisen kypsyystason.

Ensimmäisellä tasolla olevan yrityksen prosessit ovat yleensä kaoottisia ja improvisoituja, ja menestyminen perustuu "sankarikoodaajien"pätevyyteen. Tällaiset yritykset pystyvät tuottamaan toimivia tuotteita ilman prosessia. Ongelmana kuitenkin ovat jat-

kuvat budjettien ja aikataulujen ylitykset. Ilman määritettyä prosessia yritykset hylkäävät usein kriisitilanteessa kaikki prosessit ja tekevät asiat, niin kuin parhaiten pystyvät. Tällä tasolla ei vaadita minkään prosessialueen vaatimusten täyttämistä.

Toiselle tasolle siirtyminen vaatii, että yrityksen prosessit on suunniteltu, määritetty ja suoritetaan yrityksen käytäntöjen mukaisesti. Tällä tasolla olevan yrityksen projektit viedään läpi projektisuunnitelman mukaisesti. Tuotokset täyttävät niille asetetut vaatimukset, ja sen lisäksi niille on suoritettu asiaankuuluvat laadunvarmistustoimet. Aiemmin tehtyjen samanlaisten projektien suoritus pystytään toistamaan. Prosessien määrittäminen voi hieman vaihdella projektikohtaisesti, mutta niitä noudatetaan koko ajan, myös kriisitilanteessa. Tällä tasolla olevan yrityksen tulee täyttää mm. vaatimustenhallinnan (REQM Requirements Management), projektisuunnittelun (PP Project Planning) sekä mittaamisen ja analysoinnin (MA Measurement and Analysis) prosessialueiden yleiset ja spesifiset vaatimukset.

Kolmannella tasolla olevan yrityksen kaikki prosessit on määritettyjä, ja niitä pyritään tehostamaan. Tällä tasolla olevan yrityksen laadunhallintajärjestelmässä on määritetty prosessit, käytännöt, työkalut ja työohjeet. Yrityksellä on joukko vakioprosesseja, joista räätälöidään projektikohtaisesti parhaiten tilanteeseen sopivat prosessit. Kolmannella tasolla prosessit on määritetty huomattavasti tarkemmin kuin toisella tasolla, eli prosessista on kuvattu selkeästi esimerkiksi prosessin tarkoitus, syötteet, aktiviteetit, laadunvarmistuksen askeleet ja tuotokset. Prosessien suorituskyky on kvantitatiivisesti ennustettavissa oleva. Tällä tasolla olevan yrityksen tulee täyttää mm. tuoteintegraation (PI Product Integration), verifioinnin (VER Verification) ja validoinnin (VAL Validation) sekä riskienhallinnan (RSKM Risk Management) prosessialueiden yleiset ja spesifiset vaatimukset.

Neljännelle tasolle siirtyminen vaatii, että yritys asettaa mitattavissa olevat tavoitteet laadulle ja prosessien suorituskyvylle sekä käyttää saatuja tuloksia prosessien hallintaan. Mitattavat tavoitteet voivat perustua esimerkiksi asiakkaiden tai loppukäyttäjien tarpeisiin. Tällä tasolla oleva yritys ymmärtää laadun ja prosessien suorituskyvyn tilastollisten menetelmien kautta, ja prosesseja mitataan ja hallitaan koko ajan. Kerättyä

mittaritietoa varastoidaan yrityksen laatutietokantaan ja sitä käytetään päätöksenteon tukena. Syyt prosessien suorituskyvyn vaihteluun pyritään tunnistamaan ja tarvittaessa poistamaan, ja prosessien suorituskyky on tilastollisten ja kvantitatiivisten menetelmien avulla hyvin ennustettavissa oleva. Tällä tasolla olevan yrityksen tulee täyttää organisaation prosessien suorituskyvyn (OPP Organizational Process Performance) ja kvantitatiivisen projektinhallinnan (QPM Quantitative Project Management) prosessialueiden yleiset ja spesifiset vaatimukset.

Ylimmällä, viidennellä tasolla oleva yritys parantaa jatkuvasti prosessiensa suorituskykyä ja parantamisessa käyttämiään teknologioita inkrementaalisesti ja innovatiivisesti. Prosessien parantaminen perustuu mitattavissa oleviin prosessinparantamistavoitteisiin, joiden järkevyyttä ja soveltuvuutta tarkkaillaan jatkuvasti. Näiden tavoitteiden tulee tukea yrityksen liiketoimintasuunnitelman mukaisia tavoitteita. Kaikkien prosessinparantamisen toimenpiteiden vaikutuksia mitataan ja arvioidaan parantamistavoitteita vastaan. Tällä tasolla olevan yrityksen tulee täyttää organisaation innovatiivisuuden (OID Organizational Innovation and Deployment) ja syy-seuraus-suhteiden analysoinnin ja selvittämisen (CAR Causal Analysis and Resolution) prosessialueiden yleiset ja spesifiset vaatimukset (Chrissis *& al.*, 2006).

4.2.3 CMMI-arvioinnit

CMMI-kypsyystasomallin mukaisesti yrityksen prosessien kyvykkyyttä ja organisaation tai sen osan kypsyystasoa voidaan arvioida joko sisäisesti (assessment), tai ulkoisen arvioijan toimesta (appraisal, evaluation). Arviointien perimmäisenä tarkoituksena on yleensä selvittää, millä tasolla organisaation prosessit ovat verrattuna CMMI:n parhaisiin käytäntöihin, ja tunnistaa parantamista tarvitsevat kohdat. Lisäksi arvioinneilla pyritään tiedottamaan asiakkaille organisaation prosessien taso ja toisaalta täyttämään asiakkaiden sopimusvaatimukset (Chrissis *& al.*, 2006).

Assesmoinnit ovat yrityksen sisäisesti suorittamia prosessien kyvykkyyden ja organisaation kypsyyden arviointeja. Näiden arviointien tarkoituksena on yleensä parantaa

yrityksen prosesseja sisältä päin (Kulpa & Johnson, 2003). Ulkopuolisten suorittamilla arvioinneilla ei ole yleensä ollut läheskään niin suuri vaikutus yrityksen toimintaan sekä laadun ja tuottavuuden parantamiseen, kuin sisäisillä assesmoinneilla (Bush & Dunaway, 2005).

Arvioinnit ovat ulkoisten arvioijien suorittamia toimenpiteitä. Niiden tarkoituksena on valitusta lähestymistavasta riippuen tarkastella joko yksittäisten prosessialueiden kyvykkyyttä tai vaihtoehtoisesti organisaation tai sen osan kypsyystasoa. Lisäksi arvioinneissa haetaan organisaation prosessien vahvuudet ja heikkoudet laadunparantamisen pohjaksi. Arvioinnit suoritetaan kyselyjen, haastatteluiden ja kohdeorganisaation luovuttaman dokumentaation perusteella.

Evaluaatiot ovat vastaavanlaisia kuin arvioinnit, mutta niissä organisaatioon tulee ulkoinen ryhmä, joka arvioi organisaation prosesseja tulevan päätöksenteon tueksi. Tällaisia evaluointeja ovat esimerkiksi alihankkijan prosessien arviointi.

Kaikissa arviointitavoissa arvioijat tekevät päätöksen CMMI-kypsyysmallissa esitettyjen prosessialuekohtaisten spesifisten ja yleisten tavoitteiden toteutumisen perusteella. Jos tietyn prosessialueen yleiset ja spesifiset vaatimukset täyttyvät, pidetään kyseistä prosessialuetta toteutuneena. Vaiheittaisessa lähestymistavassa kullakin tasolla on tietty joukko prosessialueita, joiden vaatimukset tulee täyttää, jotta organisaatio on kyseisellä kypsyystasolla.

SEI on määrittänyt arvioinnin kolme tasoa (A, B ja C). Eniten käytetty ja hyvän menetelmätuen omaava A-taso on ainoa taso, joka voi tuottaa *kypsyys- tai kyvykkyystasoluokituksen* (rating). Käytännössä tasojen kattavuus pienenee alusta loppuun päin. A-tasolla kerätään mahdollisimman paljon tietoa sekä todisteita prosesseista, ja arvioinnin tulee olla auktorisoidun pääarvioijan suorittama, kun taas C-tasolla tietoa ei tarvitse kerätä paljoakaan, ja sen voi suorittaa henkilö, joka on koulutettu aiheeseen ja omaa kokemusta.

Arviointimenetelmät on ohjeistettu SEI:n toimesta kahdessa ohjedokumentaatiossa. Ensimmäinen on kaikkien kolmen edellä mainitun arviointitason vaatimukset sisältävä

"Appraisal Requirements for CMMI (ARC)"ja toinen on ainoastaan A-tason menetelmän ohjeistava "Standard CMMI Appraisal Method for Process Improvement (SCAMPI) Method Description Document (MDD)". Jälkimmäinen on laajalti käytetty ja toisaalta ainoa menetelmä, jonka avulla voidaan tuottaa kypsyys- tai kyvykkyystasoarvioita (Kulpa & Johnson, 2003).

Westin (2004) mukaan ei ole olemassa virallisia arviointeja tai CMMI-sertifikaatteja, vaan ainoastaan SEI:n auktorisoimia arvioijia, joilla on oikeus suorittaa arviointeja ja toisaalta SEI:n vaatimia arviointimenetelmiä, joiden mukaisesti arviointi tulisi suorittaa. Kypsyys- ja kyvykkyystasojen arvioinnin pystyy suorittamaan vain SEI:n auktorisoima arvioija, joka voi olla ulkoinen asiantuntija tai yrityksen sisäinen resurssi, kunhan hän ei ole osa arvioitavaa organisaation osaa.

4.2.4 Hyödyt ja haasteet

SEI on koonnut eri lähteistä tietoa CMMI-kypsyysmalliin perustuvan laadunparantamisen tuomista hyödyistä. Yritykset ja yhteisöt saivat lähettää heille tietoa CMMI:n käyttöönoton vaikutuksista kustannuksiin, aikatauluihin, laatuun, asiakastyytyväisyyteen, sijoitetun pääoman tuottoasteeseen (ROI) tai muihin suorituskykymittareihin. Osallistuneiden yritysten joukossa oli mukana myös suuria ja kansainvälisiä organisaatioita, kuten Boeing Company, Reuters, General Motors ja Accenture.

Osallistuneet organisaatiot kertoivat saaneensa merkittäviä hyötyjä CMMI:n käyttämisestä. Tärkeimpinä hyötyinä oli koettu tuottavuuden, tehokkuuden ja laadun merkittävä parantuminen, projektien ennustettavuuden ja aikataulujen pitävyyden parantuminen, projektien läpimenoaikojen lyheneminen sekä laatukustannusten ja havaittujen ohjelmistovirheiden määrän pieneneminen. Muita saavutettuja hyötyjä olivat alentuneiden kulujen tuomat merkittävät säästöt, aikatauluvaihteluiden pieneneminen, asiakastyytyväisyyden paraneminen sekä *sijoitetun pääoman tuottoasteen* (ROI) kasvaminen.

Ilmaista prosessien kehittäminen ei kuitenkaan ole. Kuten muidenkin referenssimallien osalla, myös CMMI-mallin avulla kehittäminen maksaa omaa sekä arvioijien työaikaa ja erilaisia arviointimaksuja (SEI, 2008).

Useimmilla pienillä yrityksillä on ongelmia CMMI:n kanssa, koska sen käytäntöjä ei Brodmanin ja Johnsonin (1994) mukaan ole tarkoitettu sovellettavaksi pieniin projekteihin, jotka ovat pienillä ohjelmistoyrityksillä yleisimpiä. Pienillä yrityksillä ja organisaatioilla ei myöskään ole mahdollisuutta muodostaa CMMI:n mukaisia hallinnollisia rakenteita, kuten erillisiä konfiguraationhallinnan tai laadunvarmistuksen organisaatioita. Tämän lisäksi pienille yrityksille ominainen käytettävien työkalujen, prosessien ja aihealueiden jatkuva muuttuminen vaikeuttaa hallittavan prosessin tarvitseman historiatiedon syntymistä. Historiatietoa syntyy, mutta ne eivät ole keskenään vertailukelpoisia.

Brodmanin ja Johnsonin (1994) mukaan pienet yritykset pelkäävät etteivät kehittämiseen käytetty työpanos, resurssit ja raha tuo vastaavaa hyötyä takaisin. Prosessien kehittäminen referenssimallin mukaisesti vie paljon resursseja, eikä yritys välttämättä voi irrottaa avainresurssejaan pitkään laatuhankkeeseen kriittisemmistä liiketoiminnoistaan. Ehkä juuri siksi, pienten yritysten yrityskulttuuri ei tue suuria laadunparantamishankkeita.

Wilkien *& al.* (2005) mukaan pienet ohjelmistoyritykset keskittyvätkin enemmän tuotelaadun varmistamiseen kuin varsinaisten tuotantoprosessien laadun varmistamiseen. Tähän tarkoitukseen jatkuvan parantamisen lähestymistapa yksittäisien prosessialueiden parantamisineen on pienelle yritykselle hyvä valinta.

4.3 ISO/IEC 15504 (SPICE)

SPICE (Software Process Improvement and Capability dEtermination), eli standardi ISO/IEC 15504 on standardointiorganisaatio ISO:n ja SPICE-projektin ylläpitämä objektiivinen ohjelmistoprosessien parantamisen ja prosessien kyvykkyyden arvioinnin viitekehys. Sen avulla pyritään määrittämään erikokoisten yritysten valittujen proses-

sien kyvykkyyden nykytila sekä soveltuvuus yrityksen toimintaan, ja toisaalta tarjoamaan suuntaviivat prosessien parantamiseen saatujen tulosten perusteella (von Wangenheim *& al.*, 2006).

Standardi ei vaadi käyttämään jotakin tiettyä arviointimallia, vaan kuvaa minimivaatimukset ISO/IEC 15504 -yhteensopivalle ohjelmistoprosessin arviointimallille. ISO/IEC 15504-5 kuvaa yhden esimerkin tällaisesta arviointimallista pohjautuen ISO/IEC 12207 -standardissa kuvattuun elinkaarimalliin (ISO, 2003b). Arvioinnissa voidaan käyttää myös muita ISO/IEC 15504 -yhteensopivia arviointimalleja, kuten RAPID (Rout *& al.*, 2000) tai MARES (Anacleto *& al.*, 2004). Tässä tutkielmassa käsitellään standardin tarjoamaa esimerkkimallia.

ISO/IEC 15504 ei ota kantaa organisaatiorakenteeseen, johtamisfilosofioihin, ohjelmiston elinkaarimalleihin tai toteutusteknologioihin. Se perustuu kaksiulotteiseen arviointimalliin, jonka ulottuvuuksina ovat *prosessi-* ja *kypsyysulottuvuus* (El Emam *& al.* 1997).

4.3.1 Prosessiulottuvuus

Standardin soveltamisesimerkki, ISO 15504-5 jakaa elinkaariprosessit kolmeen pääryhmään, varsinaiset elinkaariprosessit (primary processes), tukea antavat elinkaariprosessit (supporting processes) ja organisaation elinkaariprosessit (organizational processes). Pääluokat jakautuvat edelleen *prosessiryhmiin* ja siitä edelleen *prosesseihin*, jotka jaetaan vielä yksittäisiksi *peruskäytännöiksi* (von Wangenheim *& al.*, 2006).

ISO 15504-5 -ohjeistuksen esimerkissä käytetyn ISO/IEC 12207 -standardin (ISO, 2001a) mukaan *varsinaisiin elinkaariprosesseihin* kuuluvat hankintaprosessit (ACQ), toimitusprosessit (SPL), ohjelmistojen kehittämisprosessit (ENG) sekä operationaaliset prosessit (OPE). Näihin prosessiryhmiin kuuluvia prosesseja ovat esimerkiksi hankinnan valmistelu (ACQ.1), sopimuksen solmiminen (SPL.2), arkkitehtuurisuunnittelu (ENG.3) sekä asiakaspalvelu (OPE.2). Kunkin prosessin alla on vielä joukko peruskäytäntöjä, kuten esimerkiksi järjestelmäarkkitehtuurin suunnittelu (ENG.3.BP1).

Tukea antaviin elinkaariprosesseihin kuuluvat läpi ohjelmiston elinkaaren suoritettavat tukiprosessit. Näihin tukiprosesseihin kuuluvat konfiguraationhallinnan (CFG), laadunvarmistuksen (QUA) sekä tuotelaadun hallinnan prosessit (PRO). Näihin prosessiryhmiin kuuluvia prosesseja ovat esimerkiksi dokumentointi (CFG.1), verifiointi (QUA.2) ja tuotteen evaluointi (PRO.2). Prosessien peruskäytäntöjä ovat esimerkiksi dokumentin kirjoittaminen (CFG.1.BP5) ja verifioinnin suorittaminen (QUA.2.BP3).

Kolmas elinkaariprosessien prosessiryhmä on SPICE-jaottelun mukaan *organisaation elinkaariprosessien* ryhmä. Siihen kuuluvat hallinnon (MAN), prosessin parantamisen (PIM), resurssien ja infrastruktuurin hallinnan (RIN) sekä uudelleenkäytön prosessit (REU). Näihin prosessiryhmiin kuuluvat esimerkiksi projektinhallinnan (MAN.3), prosessin parantamisen (PIM.3), henkilöresurssien hallinnan (RIN.1) ja uudelleenkäyttöohjelman hallinta (REU.2). Peruskäytännöistä esimerkkinä voidaan mainita esimerkiksi prosessimuutosten toteuttaminen (PIM.3.BP6).

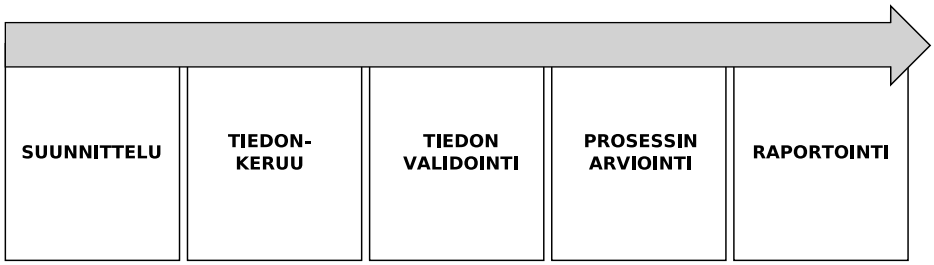
4.3.2 Kypsyysulottuvuus

ISO/IEC 15504-2 -standardin (ISO, 2002) mukaan kypsyysulottuvuudella tarkoitetaan sitä, että kunkin prosessin ja sen sisältämien toimintojen kypsyystaso arvioidaan kuusiportaisella asteikolla siten, että tasolla 0 (incomplete) on puutteellinen prosessi, tasolla 1 (performed) prosessin mukaisia toimia suoritetaan, tasolla 2 (managed) suoritukset ja työtuotokset tehdään hallitusti, ja tasolla 3 (established) prosessit on jo määritelty, ja määrityksien mukaiset prosessit on otettu käyttöön yleisesti yrityksessä. Tasolla 4 (predictable) prosesseja mitataan ja ohjataan, ja korkeimmalla tasolla 5 (optimizing) prosesseja kehitetään jatkuvasti ja optimoidaan innovatiivisesti. Kullakin tasolla on yksi tai useampia prosessiattribuutteja (process attributes), jotka määrittävät prosessin jonkin tietyn kypsyysaspektin tason. Tällaisia prosessiattribuutteja ovat esimerkiksi tasolla 2 oleva prosessin suorituskyvyn hallinta (PA 2.2). Kunkin prosessin kypsyyttä arvioidaan näiden prosessiattribuuttien toteutumisen kautta neliportaisella asteikolla - täysin saavutettu F (Fully achieved), pääosin saavutettu L (Largely achieved), osittain saavutettu P (Partially achieved) ja ei saavutettu N (Not achieved).

Näiden osa-arvosanojen perusteella määritetään kokonaisten peruskäytäntöjen ja prosessien kypsyystasot.

4.3.3 Prosessien kypsyyden arviointi ja parantaminen

Von Wangenheimin *& al.* (2006) mukaan prosessin kypsyyden arviointi eli assesmointi suoritetaan kuvan 6 mukaisesti viisivaiheisena.



Kuva 6: Arviointiprosessin vaiheet SPICE-mallissa

Arviointiprosessi aloitetaan suunnitellulla, eli luodaan mm. arvioinnin laajuuteen kantaa ottava arviointisuunnitelma. Tässä vaiheessa valitaan *arvioinnin kohteena olevat prosessit* (target process profiles). Näitä arviointikohteita voivat esimerkiksi olla aiemmin kuvatut ohjelmistotuotannon prosessit (ENG). Kohdeprosesseja voidaan valita yksi tai useampia. Yleensä valitut prosessit ovat yrityksen toiminnan kannalta erittäin olennaisia ja siksi priorisoitu korkealle arvioinnissa.

Suunnitteluvaiheen jälkeen kerätään suunnittelun yhteydessä arvioitavaksi valituista prosesseista järjestelmällisesti kaikki olennainen tieto. Tällaista tietoa ovat esimerkiksi ohjeistukset, mallipohjat sekä todisteet prosessin mukaisesta toiminnasta.

Keruuvaiheen jälkeen kerätty tieto analysoidaan ja validoidaan. Tässä vaiheessa varmistetaan tiedon oikeellisuus ja riittävyys arviointiin. Kun tietoa on riittävästi ja se on oikeellista, voidaan aloittaa varsinainen prosessien arviointi kohdassa 4.3.2 kuvattujen kypsyystasomittareiden perusteella.

Viimeiseksi dokumentoidaan arvioinnin tulokset arviointiraporttiin ja toimitetaan raportti yrityksen edustajille. Tämä arviointiraportti toimii yrityksen prosessinparannushankkeen perustana, ja tämän prosessin ansiosta laadunhallintajärjestelmä on hyvien laatukäytäntöjen mukaisesti jatkuvan parantamisen kohteena.

4.3.4 Hyödyt ja haasteet

Ohjelmistoprosessin arviointikehyksen käyttäminen prosessien kyvykkyyden arvioinnissa ja parantamisessa on perusteltua. Niiden käyttämisessä on sekä hyviä, että huonoja puolia. Von Wangenheim *& al.* (2006) tutkivat ISO/IEC 15504 -standardin soveltuvuutta pienille ohjelmistoyrityksille erään ISO/IEC 15504 -yhteensopivan prosessien kyvykkyyden MARES-arviointimallin (Anacleto *& al.*, 2004) avulla Brasiliassa.

Tutkimuksessa selvisi, että arvioinnin jälkeen kaikki osallistuneet yritykset kokivat saaneensa hyötyä arvioinnista ja oppineensa paremmin ymmärtämään ohjelmistoprosessejansa. Arvioinnin tuloksena moni yrityksistä lähti kehittämään ja mallintamaan avainprosessejansa, kuten esimerkiksi toimitus-, asiakaspalvelu- ja testausprosesseja. Lisäksi moni yritys aloitti järjestelmällisen projektinhallinnan.

Arvioinnin tuloksien pohjalta tehdyt parantamistoimet johtivat parempaan tuotelaatuun, alentuneisiin kuluihin ja vähentyneeseen työn virheiden vuoksi uudelleen tekemiseen. Useimmat yritykset kokivat lisäksi saaneensa prosessinparantamiseen sitoutumisensa johdosta lisää vakuuttavuutta asiakasneuvotteluihinsa. Huolimatta tulosten positiivisuudesta, useat tutkimukseen osallistuneet alle viiden hengen yritykset aloittivat arvioinnin jälkeen prosessinparantamistoimia, mutta eivät koskaan vieneet niitä tuotantoon. Hyvin pienten yritysten mielestä parantaminen kannattaisi jättää myöhempään vaiheeseen, kun yrityksen koko on kasvanut ensin.

Von Wangenheimin *& al.* johtopäätös kuitenkin ISO/IEC 15504 -standardin soveltumisesta pienille ohjelmistoyrityksille on, että standardin mukaisen arvioinnin suorittaminen myös pienissä ohjelmistoyrityksissä on järkevää ja tuo mukanaan merkittäviä hyötyjä.

4.3.5 Arviointikehyksen luotettavuuden arviointi

Koska prosessien kyvykkyyden arviointi perustuu arvioijakohtaiseen subjektiiviseen näkemykseen, on ISO/IEC 15504 -standardin luotettavuudesta tehty empiirisiä tutkimuksia. Arviointikehyksen luotettavuuden, yhtenäisyyden ja käytettävyyden sekä vaatimustenmukaisuuden varmistamiseksi SPICE-projektissa perustettiin joukko empiirisiä tutkimuksia (*Spice Trials*), jotka jaettiin kolmeen vaiheeseen.

Ensimmäisessä vaiheessa pyrittiin validoimaan alkuperäisen ohjeistusdokumentaation laatimisessa tehdyt suunnittelupäätökset sekä testaamaan ydindokumenttien käytettävyyttä. Toinen vaihe perustui ISO/IEC 15504 PDTR-versioon (Proposed Draft Technical Report). Nyt arvioitiin kaikkien ohjeistusdokumenttien soveltuvuus sekä tehdyt suunnittelupäätökset. Yksi vaiheen tavoitteista oli tuottaa ohjeistuksia standardin tehokkaaseen käyttöön. Kolmannen vaiheen tavoitteena oli validoida kehitetty arviointikehys standardin kokonaistavoitteita ja -vaatimuksia vastaan (Jung *& al.*, 2001).

El Emamin (1998) mukaan ohjelmistoprosessin arvioinnin luotettavuutta voidaan arvioida kahdella tavalla - sisäisen yhtenäisyyden (internal concistency) ja arvioijien välisen yhteneväisyyden (interrater agreement) kannalta.

Arviointien luotettavuuden arviointi *sisäisen yhteneväisyyden* perusteella tarkoittaa käytännössä sitä, että tutkitaan, tuottaako sama mittaustapa saman tuloksen uudelleenmittauksen yhteydessä. *Arvioijien välisen yhteneväisyyden* perusteella arviointi tarkoittaa käytännössä sitä, että tutkitaan, tuottaako samojen prosessien arviointi samoilla lähtötiedoilla eri arvioijilla erilaisia arvioita.

Sisäisen yhteneväisyyden osalta tutkimuksissa kävi ilmi, että peräkkäisillä arviointikerroilla päästään lopputuloksen kannalta samaan tulokseen käytettäessä prosessiattribuuttien arvioinnissa neliportaista, aiemmin kohdassa 4.3.2 kuvattua, F, L, P, N -arvoasteikkoa. Tämä arvoasteikko tasoittaa varianssit siten, ettei heittoja arviointituloksissa eri kerroilla juurikaan tule. Tutkimuksessa huomattiin myös, ettei tuota neliportaista asteikkoa pysty enää parantamaan.

Arvioijien välisen yhteneväisyyden osalta havaittiin samoin, että arviointien yhteneväisyys on eri arvioijien välillä hyvä. Edelleen neliportainen prosessiattribuuttien arvoasteikko tasoitti varianssit siten, ettei heittoja arviointituloksissa eri henkilöiden välillä tule. Tässäkin kohdassa huomattiin, ettei neliportaista asteikkoa enää voida parantaa.

Edellä mainittujen ohella näissä empiirisissä tutkimuksissa tutkittiin mm. tulosten ennustettavaa oikeellisuutta, jossa varmistettiin, että arviointimallissa määritetyt käytännöt todella ovat hyviä käytäntöjä, ja että niiden toteuttaminen todella parantaa suorituskykyä (El Emam & Birk, 2000a, b). Lisäksi tutkimuksissa arvioitiin ISO/IEC 15540-5 -ohjeistuksen tarjoamaa esimerkkiarviointimallia mm. käytettävyyden ja ymmärrettävyyden osalta (El Emam & Jung, 2001) sekä tutkittiin tekijöitä, jotka vaikuttavat arvioijien arviointiin käyttämään aikaan (Jung & Hunter, 2001).

5 Aloittavan ohjelmistoyrityksen ominaisuudet

Aloittavat ohjelmistoyritykset ovat luonteeltaan hyvin samankaltaisia kuin aloittavat pienet yritykset yleensäkin. Yrityksen on perustanut yksi tai useampia tietyn alan asiantuntijoita, jotka alussa muodostavat koko yrityksen henkilöstön. Pienen kokonsa ja tuoreutensa vuoksi aloittavan ohjelmistoyrityksen on kohdattava ja selvitettävä monia haasteita selviytyäkseen kasvuun ja sitä kautta toimintansa turvaamiseen.

Yleensä aloittavilla ohjelmistoyrityksillä on jokin tuoteaihio, jota lähdetään kehittämään edelleen. Aloittavien yritysten alkupääoma on usein pieni, eikä mittaville tuotekehityksille ole taloudellisia resursseja. Tämän vuoksi yritykset päätyvätkin rahoittamaan tuotekehitystään ja alkuvaiheen liiketoimintaansa tekemällä alihankintaa suuremmille yrityksille, joko toteuttamalla suuremman kokonaisuuden osakomponentteja, tai tarjoamalla henkilöresursseja päämiehen ohjelmistoprojekteihin.

Tämä lähestymistapa on yleensä toimiva, koska yrityksen perustajilla yleensä on tarvittava asiantuntemus toteuttaa tarjottuja palveluita. Yrittäjien sitominen projekteihin tuottaa yritykselle tulorahoitusta, mutta toisaalta sitoo yrittäjiä pois myynnin ja markkinoinnin sekä hallinnon tehtävistä. Alussa yrittäjät ovat mukana kaikessa toiminnassa sopimusneuvotteluista suunnitteluun, toteutukseen ja käyttökoulutukseen. Tässä tilanteessa henkilöresurssien oikeanlainen kuormittaminen on erittäin tärkeätä, jotta yritystä voidaan viedä haluttuun suuntaan.

Kuitusen *& al.* (2003) mukaan yrityksen toiminta alkuvaiheessa perustuukin yrittäjien henkilökohtaiseen osaamiseen, innostukseen ja suhdetoimintaan. Usein tärkeimpänä tavoitteena on työllistää itsensä ja toisaalta mahdollistaa yrityksen kehittyminen ja kasvaminen tulorahoituksen avulla. Aloittavan yrityksen alkutaipaletta kutsutaankin usein eloonjäämistaisteluksi.

Tilanne konkretisoituu Tekesin (2004) esityksessä alkavien yritysten kehittymisestä. Esitettyjen tilastojen mukaan aloittavista yrityksistä kuolee vuosittain noin 10 %. Selvinneistä yrityksistä 7-8 % kehittyy kasvuyrityksiksi, jotka vastaavat valtaosasta työllistävää vaikutusta.

Yrityksen tuoreus tai nuoruus vaikuttaa merkittävästi sen mahdollisuuksiin ja uskottavuuteen. Pihkalan (2001) mukaan rahoituslaitokset ja sijoittajat eivät helposti lähde rahoittamaan aloittavaa yritystä, ellei sillä todella ole jokin tekninen innovaatio, josta sijoitettu raha saataisiin takaisin. Aivan alkuvaiheessa, kun yrityksellä ei ole ensimmäistäkään asiakasta, ei sillä myöskään ole tulorahoitusta. Tässä kohtaa ollaan tekemisissä aloittavan yrityksen ehkä merkittävimmän haasteen kanssa.

Yrityksen olisi löydettävä asiakas tai asiakkaita, jotta se pystyisi rahoittamaan toimintansa muuten, kuin syömällä sijoitettua alkupääomaa. Asiakkaan löytämiseksi tarvitaan toimiva ratkaisu, paljon myynti- ja markkinointityötä sekä liiketoiminnallista osaamista. Kun sopiva asiakaskandidaatti on löytynyt, aloitetaan vakuuttaminen, että ratkaisu todella täyttää asiakkaan tarpeet. Yrityksen onneksi, yleensä asiakas tässä tilanteessa käyttää sumeilematta hyväkseen aloittavien yrittäjien tilannetta ja suostuu ostamaan tuotteen tai palvelun hyvin alhaisella hinnalla riskien minimoimiseksi. Tämä on toisaalta yrityksen kannalta hyvä, koska ensimmäinen asiakas on ensimmäinen referenssi, ja seuraava asiakas on jo helpompi vakuuttaa.

Kun asiakas on löydetty, kommunikaatio asiakkaan kanssa aloittavassa ohjelmistoyrityksessä on läheistä, ja asiakkaan toivomuksiin reagoidaan yleensä välittömästi, sen enempää miettimättä. Tämän vuoksi aloittavia ohjelmistoyrityksiä pidetään joustavina verrattuna suuriin yrityksiin, joissa kaikki asiakaskommunikaatio hoidetaan avainasiakkuuspäälliköiden ja projektipäälliköiden kautta. Toisaalta Kuitusen *& al.* (2003) mukaan pienten ohjelmistoyritysten toiminnassa ei ole mitään sellaista kilpailuetua, mitä suuremmat yritykset eivät voisi saavuttaa.

Kun tulorahoitusta saadaan tasaisesti ja riittävästi mahdollistamaan liiketoiminta yrittäjävetoisesti, aletaan yleensä harkita lisäresurssien hankkimista tuotantoon. Tällä saadaan joko lisättyä tuotantokapasiteettia tai vapautettua yrittäjiä muihin, liikkeenjohdollisiin tehtäviin. Tämä *orgaaninen kasvu* on yleensä itseään ruokkivaa, eli kun tuotantokapasiteetti lisääntyy, myyjät voivat myydä suurempia projekteja tai enemmän pieniä projekteja, mikä puolestaan nostaa kapasiteetin käyttöastetta ja mahdollisten lisäresurssien tarvetta.

Tässä vaiheessa on erityisen tärkeätä kiinnittää huomiota yrityksen kasvuvauhtiin. Vaikka kasvulle näyttäisi olevan tarvetta, kannattaa kasvun kannattavuutta miettiä pidemmällä tähtäimellä, koska yleensä projektien jälkeen on ajanjakso, jolloin edellinen projekti on jo päättynyt ja uutta ei vielä ole aloitettu. Tällainen kuoppa syö yrityksen taloudelliset resurssit vahvankin kasvun jälkeen nopeasti.

Kuitusen *& al.* (2003) mukaan yksi tyypillisimpiä käynnistysvaiheen jälkeisiä haasteita on kasvun myötä vastaan tullut yrityksen muuttuminen yrittäjävetoisesta epämuodollisesti johdetusta tiimistä johdetuksi organisaatioksi, jossa vastuuta on luovutettu muillekin kuin yrittäjille. Kuun muutos tehdään hallitusti, jää yrityksen johdolle aikaa keskittyä myyntiin, markkinointiin ja yrityksen strategian kehittämiseen. Sisäisen organisoitumisen lisäksi aloittavalla yrityksellä olisi hyvä olla kumppaniverkosto, jonka avulla yritys voisi tarjota palveluitansa laajempina kokonaisuuksina, täydentäen omaa osaamista kumppaneiden osaamisella.

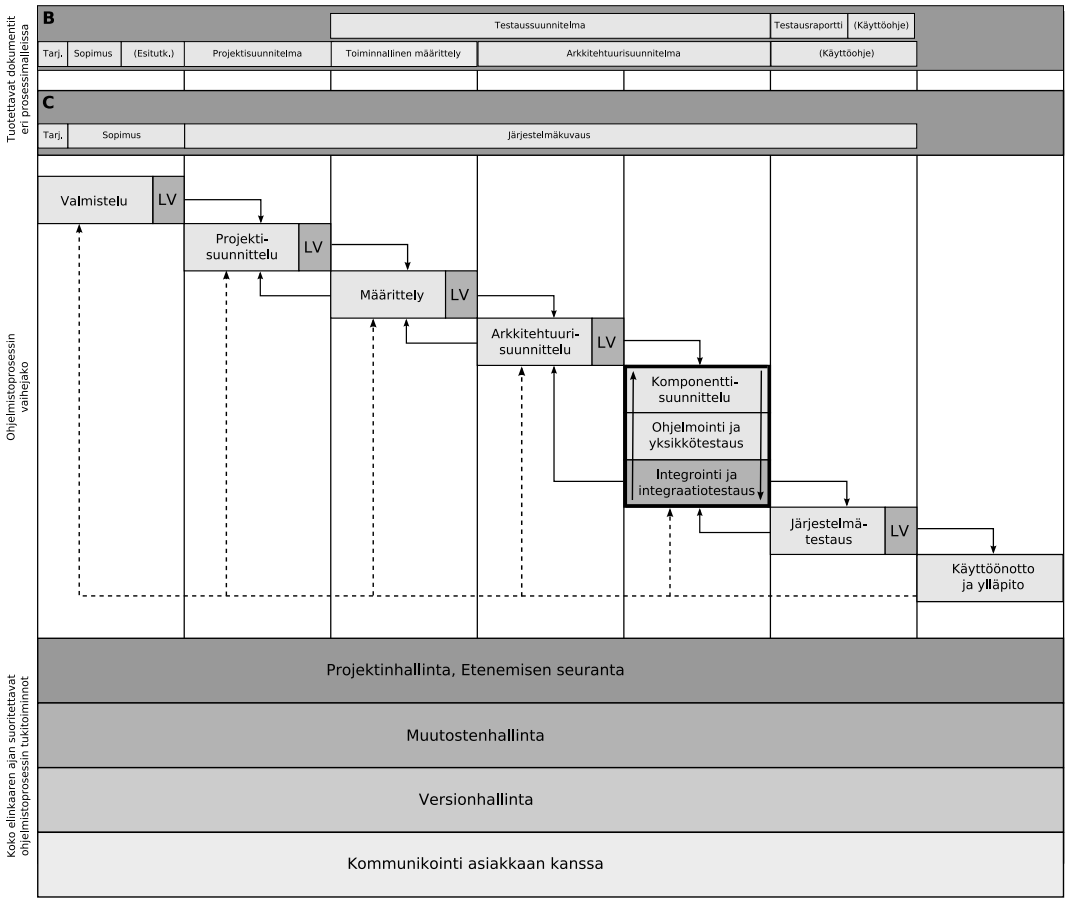
Sen lisäksi, että yrityksen täytyy löytää ensimmäiset asiakkaansa, täytyy yrityksen tuottaa jotakin sellaista palvelua tai tuotetta, josta asiakkaille on hyötyä. Kuten aiemmin tässä tutkielmassa ohjelmistoprosessin kohdalla mainittiin, yritys tarvitsee ohjelmistotuotteiden tuottamiseen ohjelmistoprosessin. Prosessi koostuu joukosta toimintoja, joiden onnistuneen toteutuksen tuloksena syntyy valmis ohjelmistotuote.

Demirorsin ja Demirorsin (1998) mukaan pienellä ohjelmistoyrityksellä ei useinkaan ole määritettyä ohjelmistoprosessia. Toistettavan mallin sijaan aloittavassa yrityksessä luotetaan yksittäisiin lahjakkaisiin ohjelmoijiin, jotka pystyvät ratkaisemaan minkä tahansa ongelman. Prosessit ovat käytännössä kaoottisia ad-hoc -tyyppisiä. Jokainen projekti on erilainen, ja jokaisessa projektissa asiat tehdään hieman eri tavalla riippuen tekijästä.

Jotta aloittavalla ohjelmistoyrityksellä olisi paremmat mahdollisuudet selviytyä elinkaarensa alussa, tarvitaan edes jonkinlainen määritetty ohjelmistoprosessi. Seuraavassa luvussa on kuvattu esimerkki yhdestä aloittavaan ohjelmistoyritykseen soveltuvasta ohjelmistoprosessista.

6 Ohjelmistoprosessi aloittavassa yrityksessä

Tähän mennessä tutkielmassa on käsitelty ohjelmistoprosessia ja erilaisia valmiita prosessimalleja sekä prosessien arviointikehyksiä. Lisäksi on tarkasteltu aloittavan ohjelmistoyrityksen luonnetta ja tilaa. Edellisessä luvussa todettiin, että myös aloittava ohjelmistoyritys tarvitsee ohjelmistoprosessin, joten tässä luvussa esitellään yksi aloittavassa ohjelmistoyrityksessä kehitetty ja toimivaksi havaittu iteratiivinen suunnitelmaohjautuva prosessimalli. Tätä mallia voidaan käyttää aloittavan ohjelmistoyrityksen prosessimallin kehittämisen perustana.



Kuva 7: Iteratiivinen suunnitelmaohjautuva prosessimalli

6.1 Prosessin kuvaus

Esimerkkiprosessimallissa ohjelmiston elinkaari jakautuu seitsemään vaiheeseen - valmistelu, projektisuunnittelu, määrittely, arkkitehtuurisuunnittelu, iteratiivinen toteutus, järjestelmätestaus sekä käyttöönotto ja ylläpito. Malli kuvaa vaiheistuksen lisäksi myös kussakin vaiheessa tuotettavan dokumentaation. Jokaisen vaiheen päätteeksi suoritetaan laadunvarmistus (LV), jossa varmistetaan, että vaiheessa tuotettu materiaali on asianmukaisessa kunnossa. Koko elinkaaren ajan vaiheiden rinnalla kulkee ohjelmistoprosessin tukitoiminnot, kuten projektin- ja muutostenhallinta, joiden tehtävänä on varmistaa projektin onnistuminen ja tuottaa tärkeätä laatudokumentaatiota.

Kaikki projektissa tuotettu materiaali viedään laadunvarmistuksen jälkeen versionhallinnan alaisuuteen. Käännetyt ohjelmistot ja osa dokumenteista toimitetaan myös asiakkaalle. Osa näistä asiakkaalle toimitettavista dokumenteista hyväksytetään asiakkaalla ennen jatkamista seuraavaan vaiheeseen.

Prosessimalli sisältää kaksi soveltamisvaihtoehtoa - prosessimallit B ja C. Ensin mainittu soveltuu hyvin suuriin ja keskisuuriin tietojärjestelmäprojekteihin, joissa tarkat ja spesifiset määrittelyt ovat erittäin tärkeässä osassa, eikä mahdolliseen järjestelmän kehityksen aikaiseen uudelleensuunnitteluun ole resursseja. Malli B tuottaa kattavan ja tarkan dokumentaation kehitettävästä järjestelmästä. Jälkimmäinen soveltuu paremmin pienempiin projekteihin, jotka ovat työmäärältään muutamasta viikosta kuukauteen tai kahteen, ja asiakas ei halua maksaa liiallisen dokumentaation tuottamisesta. Malli C tuottaa yhden tiivistetyn *järjestelmäkuvausdokumentin*.

Alun perin rinnalla oli myös kattavin prosessimalli A, jossa määrittelyvaiheessa tuotettiin ennen toiminnallista määrittelyä erillinen vaatimusmäärittelydokumentti. Kun vaatimukset myöhemmin prosessin kehittyessä yhdistettiin toiminnalliseen määrittelyyn, voitiin erillinen vaatimusmäärittelydokumentti poistaa käytöstä, jolloin malli A poistettiin käytöstä turhan raskaana.

Tässä esimerkissä käsitellään prosessimallin sovittamista asiakasprojektiin. Sisäisessä tuotekehitysprojektissa soveltaminen eroaa hieman kuvatusta.

Valmistelu

Prosessi alkaa *valmisteluvaiheella*. Tässä vaiheessa asiakkaan lähettämien materiaalien ja tarjouspyynnön perusteella arvioidaan kehitettävää järjestelmää käytettävien tekniikoiden, resurssien, aikataulun ja budjettikysymysten valossa. Tämän vaiheen päätarkoitus on tuottaa asiakkaalle tarjouspyyntöä vastaava tarjous, sekä alustavasti määrittää järjestelmään toteutettavat ominaisuudet. Lisäksi alustava projektisuunnitelma (karkean tason vaiheisiin ja tehtäviin jako sekä aikataulutus ja resurssit) muotoutuu tässä vaiheessa. Koko tämän vaiheen suorittamisessa käytetään hyväksi aiempia kokemuksia ja osaamista vastaavista projekteista.

Kun sopimus on tehty, asiakas saattaa haluta esitutkimuksen suorittamista ennen varsinaisen projektin toteuttamista. Esitutkimuksen tarkoitus on selvittää projektissa käytettävät tekniikat, resurssit ja muut ratkaisut. Esitutkimus selvittää lisäksi, onko projektia ylipäätään mahdollista toteuttaa annetuissa rajoissa. Tarjouksen laatimisen ohessa tehdyt muut tuotokset voidaan suoraan käyttää *esitutkimusraportin* laatimiseen. Esitutkimus suoritetaan yleensä ainoastaan B-mallin projekteissa.

Projektisuunnittelu

Perusteellisen *projektisuunnittelun* tarkoituksena on organisoida koko projektin elinkaari. Tässä vaiheessa tuotettava projektisuunnitelma sisältää koko projektin suunnittelun, kuten esimerkiksi kunkin projektiin osallistuvan henkilön panoksen, projektiorganisaation, sekä projektin valvonnan ja hallinnan.

Projektin suunnittelu aloitetaan määrittämällä projektin sisältö ja tavoitteet sekä vaatimukset projektin onnistumiselle. Lisäksi huomioon tulee ottaa mm. projektin arvioitu koko, tarvittavien resurssien määrä ja resursointi, projektin aikataulu, välietapit sekä projektin hallinta. Hallitun ohjelmistoprojektin tulisi aina seurata projektisuunnitelmaa. Tässä vaiheessa projekti perustetaan myös version-, projektin- ja muutostenhallintajärjestelmiin.

Prosessimalli B tuottaa projektisuunnittelun tuloksena projektisuunnitelman, joka viedään laadunvarmistuksen jälkeen versionhallinnan alaisuuteen. Prosessimalli C puo-

lestaan tuottaa kevyen vapaamuotoisen kuvauksen projektin aikataulusta, vaiheistuksesta, resursseista sekä etapeista. C-mallin dokumentti voidaan esimerkiksi toteuttaa sisältönä projektin verkkosivuille. Lisäksi kumpikin malli tuottaa projektin tarkemman tehtäväjaon projektinhallintajärjestelmään, jota kautta tarkempi projektin etenemisen seuranta suoritetaan.

Määrittely

Määrittely on ohjelmistoprojektin perustavin osa. Siinä kerätään ja määritetään vaatimukset järjestelmän ominaisuuksille, toiminnoille ja muille projektiin liittyville tekijöille. Määrittelyvaiheessa muutetaan asiakkaan vaatimukset järjestelmän toiminnallisiksi ja tietosisällöllisiksi ominaisuuksiksi. Vaatimusten muuttaminen myöhemmin tulee usein hyvin kalliiksi, koska muutosten vaikutukset koskevat helposti kaikkea tuotettua materiaalia. Tämän vuoksi määrittelyssä on oltava huolellinen. Määrittelyn pohjalta suunnitellaan lisäksi järjestelmätestauksen testitapaukset, jotka kirjataan testaussuunnitelmaan tai järjestelmäkuvausdokumentin testaussuunnitelmaosioon.

Prosessimalli B tuottaa määrittelyn tuloksena toiminnallisen määrittelyn sekä testaussuunnitelman järjestelmätestauksen testitapaukset. Prosessimalli C puolestaan tuottaa järjestelmäkuvausdokumentin tietosisällöllisiä ja toiminnallisia ominaisuuksia, sekä järjestelmätestauksen testitapauksia kuvaavat luvut. Valmis dokumentaatio viedään laadunvarmistuksen jälkeen versionhallinnan alaisuuteen.

Tämän jälkeen määrittelydokumentti toimitetaan asiakkaalle, ja häneltä pyydetään kommentit ja muutospyynnöt määrittelyyn. Hyväksyttäminen suoritetaan ennen teknisen suunnittelun aloittamista. Mahdollisten korjausten jälkeen asiakas hyväksyy määrittelyn esimerkiksi allekirjoittamalla dokumentin. Tämä vähentää merkittävästi asiakkaan halua muuttaa vaatimuksia toimittajan kustannuksella kesken projektin.

Arkkitehtuurisuunnittelu

Kun määrittely on tehty ja hyväksytetty asiakkaalla, aloitetaan järjestelmän tekninen arkkitehtuurisuunnittelu, jossa suunnitellaan määrittelyssä kuvatut vaatimukset täyttävän järjestelmän arkkitehtuuri. Arkkitehtuurisuunnittelussa suunnitellaan käytettävät

teknologiat sekä järjestelmän kokonaisarkkitehtuuri. Kokonaisarkkitehtuuriin kuuluvat mm. järjestelmän jako komponentteihin, ja komponenttien välisten rajapintojen suunnittelu.

Teknisen arkkitehtuurisuunnittelun tuloksena saadaan kehykset ja menettelytavat ohjelmointityölle. Tarkka ja yksityiskohtainen yleisarkkitehtuurin suunnittelu nopeuttaa varsinaisten komponenttien suunnittelua ja toteutusta. Ilman hyvää arkkitehtuurisuunnitelmaa toteutus tehdään helposti tehottomalla tavalla, ja toteutuksen aikana kuluu turhaan aikaa arkkitehtuurin suunnittelemiseen uudelleen. Pahimmassa tapauksessa huono arkkitehtuurisuunnittelu voi johtaa suuriin ongelmiin. Tämä on tilanne esimerkiksi silloin, kun järjestelmä hidastuu tai sen käyttö muuttuu mahdottomaksi, johtuen huonosti suunnitellun arkkitehtuurin skaalautumattomuudesta suurille käyttäjämäärille.

Prosessimalli B tuottaa arkkitehtuurisuunnittelun tuloksena *arkkitehtuurisuunnitelman*, ja prosessimalli C puolestaan järjestelmäkuvausdokumentin arkkitehtuuria kuvaavat osat. Tarvittaessa päivitetään myös testaussuunnitelman testitapauksia. Valmis dokumentaatio viedään laadunvarmistuksen jälkeen versionhallinnan alaisuuteen.

Iteratiivinen toteutus

Toteutusvaiheessa tuotetaan laaditun määritys- ja arkkitehtuuridokumentaation perusteella toimiva järjestelmä, joka täyttää määrittelydokumentissa kuvatut, asiakkaan hyväksymät vaatimukset.

Tässä vaiheessa toteutettavat ominaisuudet voidaan jakaa ketterien prosessimallien mukaisesti iteraatioihin, joiden tuloksena asiakas saa joko välijulkaisun tai lopullisen version. Toteuttaminen tapahtuu peräkkäisissä iteraatioissa, joissa on kolme alivaihetta - komponenttisuunnittelu, ohjelmointi ja yksikkötestaus sekä integrointi ja integraatiotestaus. Näitä kolmea vaihetta iteroidaan toteutusvaiheessa, kunnes iteraatioon valitut ominaisuudet on toteutettu. Jokaisen iteraation alussa suoritetaan *iteraatiosuunnittelu*, jossa valitaan iteraatiossa toteutettavat ominaisuudet. Iteraatioiden kesto on projektin koosta riippuen viikosta kuukauteen. Ensimmäisissä iteraatioissa toteutetaan järjestelmäarkkitehtuurin kannalta olennaiset komponentit.

Iteraatio aloitetaan *komponenttisuunnittelulla*, jossa suunnitellaan toteutettavat komponentit tai rajapinnat tarkemmalla tasolla. Suunnittelun yhteydessä suunnitellaan myös automaattisen yksikkötestauksen testiluokat ja päivitetään tarvittaessa arkkitehtuurisuunnitelmaa. Komponenttisuunnittelun perusteella suoritetaan varsinainen komponenttien, rajapintojen ja testiluokkien ohjelmointi sekä lähdekoodin dokumentointi. Tuotetulle ohjelmakoodille suoritetaan jokaisen käännöksen yhteydessä automaattisen yksikkötestauksen testitapaukset. Valmis ja toimiva toteutus integroidaan heti toimivaan versioon, suoritetaan integraatiotestaus, ja viedään ohjelmakoodi versionhallinnan alaisuuteen. Tavoitteena olisi, että versionhallinnassa oleva versio olisi aina kääntyvä, ja kaikki siihen toteutetut ominaisuudet olisivat toimivia ja testattuja.

Prosessimallissa B tarkennetaan tarvittaessa arkkitehtuurisuunnitelmaa ja testaussuunnitelmaa. Mallissa C puolestaan päivitetään järjestelmäkuvausdokumentin komponenttien toteutusta ja testausta kuvaavat osat. Lisäksi kummassakin mallissa tuotetaan ohjelmakoodi riittävän hyvin dokumentoituna, jotta automaattisella dokumentointityökalulla luotu luokka- ja rajapintakuvausdokumentti voidaan laittaa liitteeksi arkkitehtuurisuunnitelmaan tai järjestelmäkuvausdokumenttiin. Valmis dokumentaatio viedään laadunvarmistuksen jälkeen versionhallinnan alaisuuteen.

Kun iteraatiossa toteutettavat ominaisuudet on integroitu ja integraatiotestattu, suoritetaan seuraavassa vaiheessa vielä ennen julkaisua järjestelmätestaus.

Järjestelmätestaus

Järjestelmätestausvaiheessa järjestelmän toiminta testataan kokonaisuutena ennen julkaisua. Se suoritetaan, kun kaikki iteraation julkaisuun valitut osakomponentit on toteutettu, integroitu sekä yksikkö- ja integraatiotestattu yhdeksi kokonaisuudeksi. Järjestelmätestauksessa testataan järjestelmän lisäksi myös yhteistoiminta mahdollisten laitteiden ja muiden järjestelmien kanssa, sekä varmistetaan järjestelmän dokumentaation olemassa olo ja ulkoasu. Järjestelmätestauksessa testataan käytännössä kaikkien järjestelmän ominaisuuksien olemassa olo ja toiminta käyttämällä järjestelmää. Järjestelmätestaus suoritetaan testaussuunnitelman perusteella, ja sen testitapaukset pohjautuvat yleensä pääasiassa määrittelydokumenttiin.

Järjestelmätestauksen testaustulokset kirjataan jokaiselta testauskerralta erikseen testausraporttiin, joka on prosessimalli B:n osalta erillinen testausraporttidokumentti, ja C:n osalta liitteinä järjestelmäkuvausdokumentissa. Valmis dokumentaatio viedään laadunvarmistuksen jälkeen versionhallinnan alaisuuteen.

Kun järjestelmätestaus on suoritettu, voidaan iteraation julkaisu tehdä ja toimittaa asiakkaalle. Tällöin toteutetun järjestelmän osalta siirrytään käyttöönottoon ja ylläpitoon. Jos järjestelmän ominaisuuksien toteuttaminen on jaettu erillisiin iteraatioihin, jatketaan järjestelmän kehittämistä seuraavassa iteraatiossa, valitsemalla ensin iteraatiossa toteutettavat ominaisuudet.

Järjestelmätestauksen yhteydessä kirjoitetaan tarvittaessa järjestelmän käyttöohjedokumentaatio. Kummassakin mallissa voidaan tarvittaessa tuottaa erillinen käyttöohjedokumentti.

Järjestelmätestauksen jälkeen voidaan tarvittaessa suorittaa *hyväksymistestaus*, joka on asiakkaan järjestämä testaustilaisuus, jossa asiakas varmistaa, että järjestelmään on toteutettu hänen haluamansa ja määrittelyssä kuvatut ominaisuudet, ja että tulos vastaa haluttua. Testitapauksina voidaan käyttää järjestelmätestauksen testitapauksia. Hyväksymistestauksen yhteydessä kirjoitetaan hyväksymistestauskertomus, jonka asiakas allekirjoittaa kommentein tai ilman. Jos asiakas haluaa toteutettuun järjestelmään vielä jotakin muutoksia ennen hyväksymistä, kirjaa hän haluamansa kommentit hyväksymistestauskertomukseen, ja projektiryhmä toteuttaa muutokset. Muutosten jälkeen asiakas hyväksyy julkaisun. Tässä vaiheessa kannattaa huomioida, ettei asiakas enää voi antaa uusia vaatimuksia järjestelmälle, vaan kysymys muutoksissa on enemmänkin havaituista puutteista järjestelmässä tai dokumentaatiossa.

Käyttöönotto ja ylläpito

Käyttöönottovaiheessa testattu ja varmistettu ohjelmistotuote toimitetaan asiakkaalle ja tarvittaessa suoritetaan asennus. Asennuksen jälkeen alkaa yleensä takuuaika, jonka aikana järjestelmässä havaitut puutteet tulee viipymättä ja veloituksetta korjata, kunhan voidaan todistaa, ettei vika johdu käyttäjästä tai virheellisestä käyttötavasta. Ylläpidon

aikana järjestelmätoimittaja tarjoaa yleensä puhelin- ja sähköpostituen sekä huolehtii korjauksista ja päivitysversioista. Ylläpitovaiheessa havaitun puutteen korjaaminen voi aiheuttaa muutoksen osalta palaamisen kehitysprosessissa takaisin aiempaan vaiheeseen.

6.2 Ydinkäytännöt ja välinetuki

Esitetty prosessimalli yhdistää suunnitelmaohjautuvien ja ketterien prosessimallien käytäntöjä. Tässä kohdassa on esitelty mallin olennaisimmat ydinkäytännöt, joista suuri osa on prosessin tukitoimintoja. Lisäksi joidenkin käytäntöjen osalta on esitelty olemassa oleva välinetuki, silmällä pitäen aloittavaa ohjelmistoyritystä. Tämän vuoksi välinetuen esimerkit painottuvat enemmän ilmaisiin, avoimen lähdekoodin järjestelmiin.

Tarkastukset

Kaikki tuotettu materiaali käy läpi laadunvarmistuksen. Tarkastuksia suoritetaan dokumenteille ja lähdekoodille. *Tarkastus* on projektiryhmän sisäisesti suorittama tietyn tuotoksen laadullinen arviointi. Tarkastusten tavoitteena on löytää virheet mahdollisimman aikaisessa vaiheessa ja varmistua siitä, että kaikki ymmärtävät asiat samalla tavalla, ja että työ on ennalta asetettujen vaatimusten mukainen. Tarkastus antaa muodollisen päätöksen työvaiheelle, ja sen ansiosta seuraavan vaiheen syötteiden oikeellisuus on aina varmistettu ennen vaiheen aloittamista. Lisäksi tarkastukset tuottavat mitattavaa tietoa tuotteesta ja sen laadusta (tSoft, 2008b).

Erilaisia tarkastuskäytäntöjä ovat katselmointi ja läpikäynti. *Katselmoinnissa* tuotos tarkastetaan, ja havaitut puutteet kirjataan puuteluetteloon. Katselmointi suoritetaan virallisessa katselmointitilaisuudessa, joka pidetään heti tuotoksen valmistumisen jälkeen. Tilaisuudesta kirjoitetaan tarkastusraportti, ja havaitut puutteet kirjataan muutostenhallintaan erillisinä tai yhteen niputettuina muutospyyntöinä. Kun havaitut puutteet on korjattu, tarkastaja allekirjoittaa tarkastusraportin hyväksyttynä. Tämän jälkeen tuotos on tarkastettu ja laadunvarmistus suoritettu.

Läpikäynti on epämuodollinen katselmointi, jossa tarkastajat antavat ainoastaan huomautukset puutteista, ja tekijä kirjaa ne itselleen muistiin. Raporttia tai virallista puuteluetteloa ei kirjoiteta. Puutteet kuitenkin viedään muutostenhallintaan niputettuna yhteen tai useampaan ryhmään. Yleensä läpikäyntiä käytetään ohjelmakoodin tarkastamismenetelmänä, johtuen ohjelmakoodin luonteesta.

Automaattinen yksikkötestaus

Automaattinen yksikkötestaus mahdollistaa komponentin kattavan testaamisen jokaisen käännöksen yhteydessä. Tällä varmistetaan, että komponentti toimii niin kuin pitääkin, ja toisaalta varmistetaan, ettei ohjelmakoodiin tehty muutos ole rikkonut mitään komponentissa. Automaattisen yksikkötestauksen testausluokat kehitetään kaikille olennaisille komponenteille. Testausluokat on hyvä toteuttaa määrittelyjä vastaan jo ennen kyseisen komponentin toteuttamisen aloittamista.

Automaattisessa yksikkötestauksessa käytettävät työkalut riippuvat käytettävästä ohjelmointikielestä ja välineestä. Riippuen käytetystä toteutusteknologiasta, käytettävissä on koko joukko erilaisia *automaattisen yksikkötestauksen testauskehyksiä*, kuten JUnit (Java), CUnit (C/C++), NUnit (.NET), PHPUnit (PHP) ja SQLUnit (SQL). Yhteisesti näitä ilmaisia automaattisen yksikkötestauksen testauskehyksiä kutsutaan nimellä *xUnit*. Näiden lisäksi on olemassa vielä joukko kaupallisia testauskehyksiä. Lisätietoja automaattisesta yksikkötestauksesta, sekä eri testauskehysten asennuspaketit löytyvät osoitteesta (`http://www.opensourcetesting.org`).

Jatkuva integraatio

Automaattisen yksikkötestauksen jälkeen valmis komponentti integroidaan järjestelmän kehitysversioon, ja sille suoritetaan integraatiotestaus. Jatkuvan integraation ansiosta versionhallinnassa on aina kääntyvä ja toimiva versio, jonka kaikki toteutetut ominaisuudet on testattu ja ne toimivat oikein. Järjestelmätuen jatkuvalle integraatiolle antaa esimerkiksi ilmainen, avoimen lähdekoodin Apache Maven Continuum (`http://continuum.apache.org`) tai kaupallinen Atlassian Bamboo (`http://www.atlassian.com/software/bamboo`).

Muutostenhallinta

Virheitä ja muutoksia kutsutaan yhteisesti muutoksiksi. *Virhe* on vika tai puute järjestelmän toiminnassa tai dokumentaatiossa, ja *muutos* on esimerkiksi parannusehdotus tai jonkin toimivan ominaisuuden muokkaamispyyntö. Muutospyynnöt voivat tulla asiakkaalta, loppukäyttäjiltä, projektiryhmältä tai toimittajan muulta henkilökunnalta riippuen siitä, kuka järjestelmää käyttää ja testaa. Muutospyyntö voidaan vastaanottaa missä muodossa tahansa, kunhan se kirjataan muutostenhallintajärjestelmään heti vastaanottamisen jälkeen.

Muutostenhallinnalla varmistetaan, että kaikki kehitettävään järjestelmään tehtävät muutokset ovat harkittuja, ja toimittajan, sekä tarvittaessa myös asiakkaan hyväksymiä. Harkinta ja hyväksyttäminen muutosten osalta on tärkeää, koska hyvin usein muutokset voivat venyttää projektin toteutusaikataulua.

Kunnollisen muutostenhallintajärjestelmän käyttäminen varmistaa sen, että kaikki *muutospyynnöt* tulevat käsiteltyä ja kirjattua, eikä havaittuja kriittisiä puutteita unohdu korjaamatta. Kun muutostenhallintajärjestelmän ansiosta jokaisella muutospyynnöllä on oma yksilöllinen tunnisteensa, voidaan niiden aiheuttamat vaikutukset jäljittää aina dokumentaatioon ja ohjelmakoodiin asti käyttämällä yhtenäistä merkintätapaa muutoksia toteuttaessa. Hyviä muutostenhallinnan työkaluja ovat esimerkiksi ilmainen, avoimen lähdekoodin Bugzilla (`http://www.bugzilla.org`) tai kaupallinen Atlassian Jira (`http://www.atlassian.com/software/jira`).

Projektinhallinta ja etenemisen seuranta

Projektien hallittu läpivienti ja etenemisen tarkka seuranta mahdollistavat projektin läpiviennin aikataulussa ja onnistuneesti. Projektien läpivienti perustuu tehtyyn projektisuunnitelmaan, jota päivitetään tarvittaessa. Projektien etenemistä seurataan projektinhallintajärjestelmän ja viikoittaisten projektipalavereiden avulla.

Näissä palavereissa kukin projektiryhmän jäsen kertoo menneen jakson työtehtävät ja etenemisensä verrattuna aikatauluun. Lisäksi kerrotaan suunnitelmat tulevan jakson työtehtävistä ja mahdollisesti ilmenneistä riskeistä, jotka voivat venyttää projek-

tin aikataulua tai muutoin vaikuttaa projektin etenemiseen. Mahdollisia keinoja riskien välttämiseksi voidaan myös pohtia. Tällä tavoin projektin johto on koko ajan selvillä projektin etenemisestä ja mahdollisista aikatauluun vaikuttavista asioista.

Jos projektinhallinta halutaan viedä tarkemmalle tasolle, voidaan ottaa käyttöön kohdassa 3.2 kuvatut Scrum-mallin päivittäiset tapaamiset. Tällöin etenemistä seurataan päivittäisellä n. 15 minuutin mittaisella palaverilla, jossa käsitellään projektin etenemistä, edellisen palaverin jälkeen tehtyä työtä, meneillään olevia tehtäviä, mahdollisia ongelmia niiden suorittamisessa, sekä seuraavaan palaveriin mennessä suoritettavaa työtä. Jos tätä lähestymistapaa käytetään, ei erillisille viikkopalavereille ole välttämättä tarvetta.

Projektinhallinnalle ja etenemisen seurannalle on olemassa hyvä välinetuki. Työasemapohjaisia järjestelmiä ovat esimerkiksi ilmainen, avoimen lähdekoodin OpenProj (`http://openproj.org/openproj`), sekä kaupallinen Microsoft Project (`http://office.microsoft.com/project`). Verkkopohjaisia ratkaisuja puolestaan ovat ilmainen, avoimen lähdekoodin dotProject (`http://www.dotproject.net`), sekä kaupalliset Planmill PSA (`http://www.planmill.com`) ja Severa PSA (`http://www.severa.com/fi/product.htm`). Kumpikin jälkimmäisistä on pääasiallisesti toiminnanohjausjärjestelmä, mutta ne sisältävät hyvän tuen projektinhallinnalle, työn etenemisen seurannalle sekä projektin kustannuslaskennalle.

Version- ja kokoonpanonhallinta

Kaikki tuotettu, tarkastettu ja hyväksytty dokumentaatio sekä lähdekoodi viedään versionhallinnan alaisuuteen. Versionhallinnassa olisi jokaisella hetkellä oltava kääntyvä ja toimiva versio, johon on integroitu viimeisimmät valmiit toiminnot. Versionhallinnan ansiosta päällekkäisten muutosten mahdollisuus estetään, ja ohjelman muutoshistoria on koko ajan nähtävillä.

Versionhallinta on osa laajempaa kokonaisuutta, kokoonpanonhallintaa (konfiguraationhallinta), jolla hallitaan julkaistujen versioiden sisältöä ja rakennetta. *Kokoonpa-*

nonhallinnan ansiosta julkaistavan tuotteen versio osataan rakentaa oikeista komponenttiversioista, ja toisaalta tiedetään tarkasti, mistä komponenttiversioista asiakkaalle asennettu versio muodostuu. Kun tieto on olemassa, voidaan versionhallinnasta ottaa milloin tahansa ulos jokin tietylle asiakkaalle asennettu kokoonpano ja testata tai kehittää sitä eteenpäin.

Hyvän välinetuen version- ja konfiguraationhallinnalle antavat esimerkiksi ilmainen, avoimen lähdekoodin Subversion (SVN) (`http://subversion.tigris.org`) tai kaupallinen BitKeeper (`http://www.bitkeeper.com`).

Asiakaskommunikaatio

Asiakaskommunikaatio on yksi projektin onnistumisen kulmakivistä. Ilman riittävää kommunikointia asiakkaan kanssa, ei voida olla varmoja lopputuloksen soveltuvuudesta asiakkaalle. Lisäksi yhteydenpito asiakkaaseen mahdollistaa nopean reagoinnin mahdollisten muutosten vaikutukseen aikatauluihin ja tarvittaviin resursseihin.

Tämän vuoksi projektilla on hyvä olla sisäisten viikkopalaverien lisäksi, esimerkiksi kerran kuukaudessa pidettävä johtoryhmän tapaaminen, jossa myös asiakkaan edustaja on mukana. Tässä tapaamisessa asiakkaalle esitellään projektin tilanne. Tarvittaessa myös järjestelmän kehitysversiota voidaan esitellä, jos kaikki siihen jo toteutetut ominaisuudet toimivat oikein.

Näiden palaverien lisäksi, myös iteraation suunnittelussa voidaan ottaa huomioon asiakkaan toiveet julkaisujen sisällöstä. Tällöin asiakas otetaan mukaan iteraatiosuunnittelupalaveriin kunkin iteraation alussa.

6.3 Hyödyt ja haasteet

Vaikka tässä luvussa kuvattu prosessimalli on havaittu käytännössä toimivaksi malliksi, kärsii se osittain suunnitelmaohjautuvien prosessimallien perusongelmista, kuten etupainotteisesta suunnittelusta. Kehityksen aikaiset muutokset vaatimuksiin tuovat projekteihin edelleen suuren määrän lisätyötä. Toisaalta tehokas muutostenhallinta ja C-

mallin kevennetty dokumentaatio helpottaa ja nopeuttaa prosessia muutosten kohdalla. Kevyemmän dokumentaation ansiosta, aikaa ei käytetä turhiin välivaiheisiin ja useisiin eri laadunvarmistuksiin. Ketterä ja iteratiivinen toteutusvaihe toisaalta keventävät mallin raskautta ja mahdollistavat joustavuuden.

Prosessimallin kaksi eri sovitustapaa mahdollistavat prosessin käyttämisen erikokoisissa ja erilaisissa projekteissa. Tämä prosessimallipari toimii niin aloittavilla, kuin jo toimintaansa pidemmälle vieneillä ohjelmistoyrityksillä. Prosessimalli B on suunnattu enemmänkin toiminnassaan pidemmällä oleville yrityksille, joiden projektien kokoluokka on jo suurempi. Aloittavalle yritykselle, tai pienempiin, alle kaksi kuukautta kestäviin projekteihin sopii huomattavasti paremmin kevennetty prosessimalli C. Sen avulla saadaan varmistettua, että kaikki tulee tehtyä, laatu varmistettua ja tuotokset dokumentoitua, ilman liiallista dokumentaation tuottamista.

7 Yhteenveto

Aloittava ohjelmistoyritys toimii haasteellisessa tilanteessa, jossa sen on toisaalta luotava perustuksensa ja selviydyttävä taloudellisesti päivätasolla, sekä toisaalta luotava uskottava ja vakuuttava referenssipohja ensimmäisten asiakkaiden avulla. Ensimmäisten asiakkaiden vakuuttaminen on usein vaikeata, ja toimiva prosessimalli voi auttaa vakuuttamisessa. Ensimmäisen referenssiasiakkaan jälkeen seuraavan vakuuttaminen on jo huomattavasti helpompaa.

Prosessimalli kuvaa projektin vaiheet ja niiden aikana suoritettavat tehtävät. Aloittavan ohjelmistoyrityksen kannattaa valita käytettävä prosessimalli jostakin valmiista prosessimallista, ja tarvittaessa soveltaa sitä omaan toimintaan sopivaksi. Prosessimallit ovat yleensä joko suunnitelmaohjautuvia, tai ketteriä malleja. Yleisellä tasolla kummankin suunnan prosessimallit toimivat sekä pienissä, että suurissa yrityksissä.

Suunnitelmaohjautuvat prosessimallit ovat perinteisiä ja hyvin laajalti käytettyjä prosessimalleja. Ne pohjautuvat etupainoiseen suunnitteluun ja tiukkaan vaiheistukseen. Osaltaan juuri tämä etupainoinen suunnittelu on osoittautunut suunnitelmaohjautuvien prosessimallien kompastuskiveksi. Asiakas ei pysty antamaan kaikkia vaatimuksia kerralla heti alussa, vaan ajatus yleensä hioutuu asiakkaan päässä kehitysprosessin aikana, jolloin järjestelmän vaatimuksiin tulee muutoksia. Muutosten tekeminen aiheuttaa palaamisen takaisin määrittelyvaiheeseen, sekä kaikkien muiden vaiheiden läpikäynnin ja laadunvarmistuksen ennen jatkamista. Tämä tulee usein kalliiksi ja viivästyttää projekteja.

Ketterät prosessimallit toisaalta ovat hyvin joustavia, muutokselle suopeampia ja parantavat kehitysnopeutta. Niiden perusajatuksena on yleensä iteratiivinen ja inkrementaalinen toteutus, jossa järjestelmään toteutetaan jokaisessa iteraatiossa tietty joukko ominaisuuksia. Jokaisen iteraation jälkeen järjestelmästä tehdään julkaisu, joka voidaan antaa asiakkaalle. Tämä lyhyen julkaisuvälin käytäntö on suuressa kontrastissa perinteisen vesiputousmallin lopussa tapahtuvaan toimitukseen. Lisäksi ketterät prosessimallit tuovat mukanaan myös koko joukon ohjelmistotuotannon parhaita käytän-

töjä, kuten automaattisen yksikkötestauksen, jatkuvan integraation ja pariohjelmoinnin. Ketterien prosessimallien haasteena on kuitenkin järjestelmädokumentaation puuttuminen. Tästä johtuen järjestelmän kokonaiskuvaa ei välttämättä ole kenelläkään, ja järjestelmästä tulee helposti kokoelma ominaisuuksia. Samoin dokumentaation puutteesta johtuen, ei välttämättä pystytä sanomaan, täyttääkö järjestelmä asiakasvaatimukset.

Kummallakin prosessimallityypillä on siis omat hyvät ja huonot puolensa, ja useissa tapauksissa näiden prosessimallien parhaita puolia on pyritty yhdistelemään. Luvussa 6 kuvattu prosessimalli on juuri tällainen suunnitelmaohjautuvan ja ketterän prosessimallin sekoitus.

Käytettävän prosessimallin valintaan vaikuttaa yleensä yrityksen koon lisäksi projektin ja projektiryhmän koko, sekä projektin dokumentaatio- ja kriittisyysvaatimukset. Lisäksi valintaan voi vaikuttaa jokin noudatettava standardikehys, kuten kohdassa 4.1 esitelty ISO 9001. Kantavana ajatuksena prosessimallin valinnassa kannattaa kuitenkin pitää se, että mitä pienempi projekti, sitä kevyempi on prosessimallin oltava, jotta projekti voidaan viedä läpi kannattavasti. Toisaalta mitä kevyempi prosessimalli, sitä joustavammin, ja todennäköisesti nopeammin työ saadaan valmiiksi, aikataulussa ja kannattavasti. Projektien kannattavuuteen kannattaakin kiinnittää erityistä huomiota, jotta projektien katteet pysyvät järkevinä ja yrityksen toiminta kannattavana.

Prosessimalleja voidaan arvioida ja kehittää erilaisten arviointikehyksien ja standardien avulla. Tällaisia arviointikehyksiä ovat esimerkiksi kohdassa 4.3 kuvattu ISO/IEC 15504 (SPICE), sekä kohdassa 4.2 kuvattu CMMI. Näiden arviointikehyksien käyttäminen tuo merkittäviä hyötyjä, mutta usein arvioinitikehykset ovat kuitenkin soveltumattomia aloittavan pienen ohjelmistoyrityksen toimintaan. Ongelmaksi nousevat esimerkiksi organisaation liian pieni koko mallien vaatimien hierarkioiden ja prosessien perustamiseksi tai liian suuret kustannukset ja resurssivaatimukset. Tämän vuoksi olisikin suositeltavaa käyttää arviointikehyksiä yksittäisten olemassa olevien prosessien parantamiseksi ja kehittämiseksi tai prosessin käytäntöjen rakentamisen pohjana. Tällöin arviointikehyksistä saadaan täysi hyöty irti myös aloittavissa yrityksissä.

Laadunhallintajärjestelmiä voidaan sertifioida erilaisten standardien mukaiseksi. Tällöin ohjelmisto- ja muiden prosessien tulee noudattaa standardin vaatimuksia, ja tuotelaadusta saadaan tasaisempaa. Tällaisia standardeja ovat esimerkiksi kohdassa 4.1 kuvattu ISO 9001:2000, sekä sen ohjelmistotuotannon sovitusohjeistus ISO 9000-3.

Huolimatta hyödyistä, nämä standardit ovat aloittavalle ohjelmistoyritykselle kuitenkin aivan liian raskaita, koska sertifiointi tuo mukanaan merkittävän määrän prosessien suunnittelu-, toteutus- ja dokumentointityötä, joista muodostuu merkittävästi kuluja palkkojen, konsulttipalkkioiden ja sertifiointimaksujen myötä. ISO 9001 -sertifiointi on järkevä vasta suuremmille yrityksille, jotka ovat olleet toiminnassaan jo pidemmän aikaa, ja esimerkiksi asiakkaan vaatimuksesta tarvitsevat ISO 9001 -sertifikaatin.

Parasta onkin kehittää omaan yritykseen ja tyypillisimpiin projekteihin soveltuva prosessimalli, käyttäen pohjana jotakin valmista prosessimallia, tai esimerkiksi luvussa 6 kuvattua, aloittavassa ohjelmistoyrityksessä käytössä ollutta, vesiputousmallista jalostettua, suunnitelmaohjautuvaa ja iteratiivista prosessimallia. Prosessin mittaaminen ja kehittäminen ovat erittäin tärkeitä, mutta niihin kannattaa keskittyä vasta, kun itse ohjelmistoprosessi on saatu juurrutettua yritykseen riittävän hyvin. Projektipäälliköllä on merkittävä vastuu käyttöönoton onnistumisessa, koska käyttöönotossa vaikeinta on itse tuotantoon vienti, eli saada ohjelmoijat ja suunnittelijat luopumaan käyttämistään ad-hoc -menetelmistä sekä noudattamaan määritettyä ja dokumentoitua ohjelmistoprosessia.

Loppujen lopuksi tärkeintä on, että aloittavallakin yrityksellä on määritetty ja dokumentoitu ohjelmistoprosessi. Tämän tutkielman kirjoittaja on itse ollut perustamassa ja rakentamassa alkavaa ohjelmistoyritystä, ja on edelleen yrittäjänä samassa yrityksessä. Esimerkkinä kuvattu aloittavan ohjelmistoyrityksen prosessimalli on jalostunut kuluneiden vuosien aikana raskaasta vesiputousmallin mukaisesta prosessimallista todellisessa käytössä.

Viitteet

Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. (2002). *Agile software development methods: Review and analysis*. VTT Publications 478, VTT, Espoo.

Acuña, S. T., Ferré, X. (2001). Software process modelling. *ISAS-SCI '01: Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics Development*, IIIS, 237-242.

Agile Manifesto (2008). *Manifesto for agile software development*. WWW-sivusto, `http://agilemanifesto.org` (26.4.2008).

Anacleto, A., von Wangenheim, C. G., Salviano, C. F., Savi, R. (2004). A method for process assessment in small software companies. *SPICE 2004: Proceedings of the 4th International SPICE Conference on Process Assessment and Improvement*, SPICE User Group and Critical Software SA, Lissabon, 69-76.

Anttila, J., Vakkuri, J. (2001). *Luovan johtajan ISO 9000*. Sonera Oyj, Helsinki.

Beck, K. (1999). Embracing change with extreme programming. *IEEE Computer* **32**(10), 70-77.

Ben-Yaacov, G. (1995). Reap the rewards of quality with ISO 9000. *IEEE Computer Applications in Power* **8**(4), 26-30.

Boehm, B., Turner, R. (2003). *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Professional, Boston.

Bowers, A. N., Sangwan, R. S., Neill, C. J. (2007). Adoption of XP practices in the industry - A survey. *Software Process Improvement and Practice* **12**(3), 283-294.

Brodman, J. G., Johnson, D. L. (1994). What small businesses and small organizations say about the CMM. *ICSE '94: Proceedings of the 16th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos, 331-340.

Bush, M., Dunaway, D. (2005). *CMMI®Assessments: Motivating Positive Change*. Addison-Wesley Professional, Boston.

Chrissis, M. B., Konrad, M., Shrum S. (2006). *CMMI®: Guidelines for Process Integration and Product Improvement (2nd Edition)*. Addison-Wesley Professional, Boston.

Coallier, F. (1994). How ISO 9001 fits into the software world. *IEEE Software* **11**(1), 98-100.

Corbett, C. J., Montes, M. J., Kirsch, D. A., Alvarez-Gil, M. J. (2002). Does ISO 9000 certification pay?. *ISO Management Systems* (July-August), 31-40.

Demirors, O., Demirors, E. (1998). Software process improvement in a small organization: Difficulties and suggestions. *Software Process Technology* (1487), 1-12.

El Emam, K., Drouin, J.-N. (ed.), Melo, W. (ed.) (1997). *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*. IEEE Computer Society Press, Los Alamitos.

El Emam, K. (1998). The internal consistency of the ISO/IEC 15504 software process capability scale. *Metrics 1998: Proceedings of the Fifth International Software Metrics Symposium*, IEEE Computer Society Press, Washington DC, 72-81.

El Emam, K., Birk, A. (2000a). Validating the ISO/IEC 15504 measures of software development process capability. *Journal of Systems and Software* **51**(2), 119-149.

El Emam, K., Birk, A. (2000b). Validating the ISO/IEC 15504 measure of software requirements analysis process capability. *IEEE Transactions on Software Engineering* **26**(6), 541-566.

El Emam, K., Jung, H.-W. (2001). An empirical evaluation of the ISO/IEC 15504 assessment model. *Journal of Systems and Software* **59**(1), 23-43.

Haikala, I., Märijärvi, J. (2003). *Ohjelmistotuotanto*. Talentum, Helsinki.

Hoyle, D. (2001). *ISO 9000 Quality Systems Handbook, Fourth Edition*. Butterworth-Heinemann, Oxford.

ISO (2001a). *ISO/IEC 12207. Information Technology - Software Life Cycle Processes*. FDAM ISO/IEC JTC1/SC7/WG7, International Standards Organisation, Geneva.

ISO (2001b). *ISO/IEC 9126-4. Software Engineering – Software Product Quality - Part 4: Quality In Use Metrics*. DTR Ballot ISO/IEC JTC1/SC7/WG6, International Standards Organisation, Geneva.

ISO (2002). *ISO/IEC 15504-2. Information technology - Process Assessment - Part 2: Performing an Assessment*. FDIS ISO/IEC JTC1/SC7/WG10, International Standards Organisation, Geneva.

ISO (2003a). *ISO/IEC 9000-3. Software Engineering - Guidelines for the Application of ISO 9001:2000 to Computer Software*. FDIS ISO/IEC JTC1/SC7/WG18, International Standards Organisation, Geneva.

ISO (2003b). *ISO/IEC 15504-5. Information Technology - Process Assessment - Part 5: An Exemplar Process Assessment Model*. CD ISO/IEC JTC1/SC7/WG10, International Standards Organisation, Geneva.

Jung, H.-W., Hunter, R. (2001). The relationship between ISO/IEC 15504 process capability levels, ISO 9001 certification and organization size: an empirical study. *Journal of Systems and Software* **59**(1), 43-55.

Jung, H.-W., Hunter, R., Goldenson, D. R., El Emam, K. (2001). Findings from Phase 2 of the SPICE trials. *Software Process: Improvement and Practice* **6**(4), 205-242.

Kehoe, R., Jarvis, A. (1996). *ISO 9000-3: A Tool for Software Product and Process Improvement*. Springer-Verlag, New York.

Koch, A. S. (2005). *Agile Software Development: Evaluating the Methods for Your Organization*. Artech House, Boston.

Kuitunen, K., Ilomäki, S.-K., Simons, M., Valjakka, T. (2003). *Kehity kasvuun. Pk-yrityksen kasvu ja kehittäminen*. Työelämän kehittämisohjelma, Raportteja 29, Työministeriö, Helsinki.

Kulpa, M. K., Johnson, K. A. (2003). *Interpreting the CMMI®: A Process Improvement Approach*. CRC Press, Boca Raton.

Mann, C., Maurer, F. (2005). A case study on the impact of Scrum on overtime and customer satisfaction. *ADC '05: Proceedings of the Agile Development Conference*, IEEE Computer Society Press, Washington DC, 70-79.

Marçall, A. S. C., de Freitas, B. C. C., Soares, F. S. F., Furtado, M. E. S., Maciel, T. M., Belchior, A. D. (2008). Blending Scrum practices and CMMI project management process areas. *Innovations in Systems and Software Engineering* **4**(1), 17-29.

McConnell, S. (1998). Best practices. *IEEE Software* **15**(3), 119-120.

Mišić, V. B. (2006). Perceptions of extreme programming: An exploratory study. *ACM SIGSOFT Software Engineering Notes* **31**(2), 1-8.

Morien, R. (2005). Agile management and the Toyota way for software project management. *INDIN '05: Proceedings of the 3rd IEEE International Conference on Industrial Informatics*, IEEE Computer Society Press, Washington DC, 516-522.

Mutafelija, B., Stromberg, H. (2003). *Systematic Process Improvement Using ISO 9001:2000 and CMMI®*. Artech House, Boston.

Müller, M. M., Padberg, F. (2003). On the economic evaluation of XP Projects. *ACM SIGSOFT Software Engineering Notes* **28**(5), 168-177.

Newkirk, J., Martin, R. C. (2000). Extreme programming in practice. *OOPSLA '00: Addendum to the 2000 Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications*, ACM, New York, 25-26.

Paulk, M. C. (1994). *A Comparison of ISO 9001 and the Capability Maturity Model for Software*. Technical Report CMU/SEI-94-TR-12, Software Engineering Institute, Pittsburgh.

Paulk, M. C. (1995). How ISO 9001 compares with the CMM. *IEEE Software* **12**(1), 74-83.

Pihkala, T., (2001). *Entrepreneurial capability and new venture formation. A study on entrepreneurs´ start-up practices*. Väitöskirja. Acta Wasaensia No. 84, University of Vaasa, Vaasa.

Pressman, R. S. (2000). *Software Engineering: A Practitioners Approach, Fifth Edition*. McGraw-Hill, Berkshire.

Rising, L., Janoff, N. S. (2000). The Scrum software development process for small teams. *IEEE Software* **17**(4), 26-32.

Robillard, P. N., Kruchten, P., d'Astous, P. (2002). *Software Engineering Process with the UPEDU*. Addison-Wesley, Boston.

Rout, T. P., Tuffley, A., Cahill, B., Hodgen, B. (2000). The rapid assessment of software process capability. *SPICE 2000: Proceedings of the 1st International SPICE Conference on Process Assessment and Improvement*, SPICE User Group, Limerick, 47-55.

Royce, W. W. (1987). Managing the development of large software systems: concepts and techniques. *ICSE '87: Proceedings of the 9th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos, 328-338.

Salo, O. (2006). *Enabling Software Process Improvement in Agile Software Development Teams and Organisations*. VTT Publications 618, VTT, Espoo.

Schwaber, K., Beedle, M. (2002). *Agile Software Development With Scrum*. Prentice-Hall, Upper Saddle River.

Schwaber, K. (2004). *Agile Project Management With Scrum*. Microsoft Press, Redmond.

SEI (2006). *CMMI®for Development, Version 1.2*. Technical Report CMU/SEI-2006-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

SEI (2007). *CMMI®for Acquisition, Version 1.2*. Technical Report CMU/SEI-2007-TR-017, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

SEI (2008). *CMMI Performance Results*. WWW-sivusto, `http://www.sei.cmu.edu/cmmi/results.html` (8.3.2008).

Standards Australia (2001). *AS1540 Software development guide to ISO 9001-2000*. Final CDR ISO/IEC JTC1/SC7/WG18, Standards Australia, Sydney.

Succi, G., Marchesi, M. (2001). *Extreme Programming Examined*. Addison-Wesley Professional, Boston.

Sutherland, J., Jakobsen, C. R., Johnson, K. (2007). Scrum and CMMI level 5: The magic potion for code warriors. *AGILE '07: Proceedings of the AGILE 2007*, IEEE Computer Society Press, Washington DC, 272-278.

Tekes (2004). *Alkavat yritykset*. `http://www.tekes.fi/tekes/esittely/A_teknopol/Alkavat_yritykset.ppt` (27.4.2008).

tSoft (2008a). *Ohjelmistotuotantoprosessin parantaminen*. WWW-sivusto, `http://www.cs.joensuu.fi/tSoft/prosessinparantaminen.htm` (12.4.2008).

tSoft (2008b). *Laadun hallinta*. WWW-sivusto, `http://www.cs.joensuu.fi/tSoft/laadunhallinta.htm` (24.5.2008).

Upender, B. (2005). Staying agile in government software projects. *ADC '05: Proceedings of the Agile Development Conference 2005*, IEEE Computer Society Press, Washington DC, 153-159.

Vriens, C. (2003). Certifying for CMM Level 2 and IS09001 with XP@Scrum. *ADC '03: Proceedings of the Agile Development Conference 2003*, IEEE Computer Society Press, Washington DC, 120-124.

Wake, W. C. (2000). *Extreme Programming Explored*. Addison-Wesley Professional, Boston.

von Wangenheim, C. G., Anacleto, A., Salviano, C. F. (2006). Helping small companies assess software processes. *IEEE Software* **23**(1), 91-98.

West, M. (2004). *Real Process Improvement Using the CMMI®*. Auerbach Publications, Boston.

Wilkie, F. G., McFall, D., McCaffery, F. (2005). An evaluation of CMMI process areas for small- to medium-sized software development organisations. *Software Process Improvement and Practice* **10**(2), 189-201.