Six Sigma ohjelmistokehityksessä

Harri Pirttinen

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaajat: Eleni Berki, Timo Poranen
Marraskuu 2010

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Harri Pirttinen: Six Sigma ohjelmistokehityksessä
Pro gradu -tutkielma, 62 sivua
Marraskuu 2010

Tutkielman tarkoituksena on selvittää, mikä Six Sigma on ja miten sitä voisi hyödyntää ohjelmistokehityksessä ohjelmistotuotantoprosessin ja laadun kehittämiseen. Six Sigma on levinnyt muilla teollisuudenaloilla prosessinkehityksessä ja laadunkehityksessä, eikä sen soveltamisesta ohjelmistokehitykseen ole tehty kattavaa tutkimusta.

Tutkimuksessa perehdytään Six Sigmaan kirjallisuuden ja toisten tutkijoiden löydöksien perusteella. Ohjelmistokehityksen sovellustapoja kootaan lähteistä ja niiden käytön hyödyllisyyttä arvioidaan.

Työssä havaittiin, että Six Sigma on sovellettavissa tietyiltä osilta ohjelmistokehitykseen, mutta se ei ole hopealuoti parempaan ohjelmistolaatuun. Se voi kuitenkin ajaa organisaatiota tilaan, jossa organisaatio on tietoinen tuotettujen virheiden syistä ja täten auttaa organisaatiota tekemään oikeita korjausliikkeitä. Tutkielmassa tehtiin havaintoja Six Sigman hyödyntämisestä Scrum-kehitysmalliin ja yksikkötestaukseen.

Tutkielmassa käsitellystä aiheesta on vähän tutkimustyötä ja tässä tutkielmassa esille tulleet ehdotukset ja pohdinnat vaatisivat lisätutkimusta, jotta niiden todellinen hyöty voitaisiin todentaa.

Avainsanat ja -sanonnat: Six Sigma, ohjelmistolaatu, ohjelmistotuotantoprosessin kehitys, CMMI, Scrum, PSP/TSP, yksikkötestaus, tilastollinen analyysi.

Sisällysluettelo

1. Johdanto

Laadukkuus on tavoiteltu piirre ohjelmistotuotteissa. Laadukkaita ohjelmistotuotteita kehittävät ohjelmistoyritykset erottuvat edukseen muista kilpailijoista asiakkaiden valitessa tuottajia. Ohjelmistoyrityksen siis kannattaa panostaa ohjelmistotuotteiden laatuun, sillä laatu on suora kilpailuetu markkinoilla. Asiakkaan näkökulmasta laatu on arvoa. Ohjelmistokehityksessä ohjelmiston arvo on eri muuttujien yhdistelmä. Tällaisia muuttujia ovat mm. ohjelmistotuotantoprosessi, lopputuotteen laatu, käyttäjien ja käyttötilanteiden ominaisuudet sekä markkinatilanne [Sheriff and Georgiadou, 2007].

Pysyvä korkea laatu ohjelmistokehityksessä on lähempänä tavoitetta kuin toteutunutta todellisuutta. Ohjelmistoyritykset luovat laatujärjestelmiä, joiden avulla laatuongelmien tulisi poistua tuotantoprosessista. Laatujärjestelmä muodostuu yrityksen työprosesseista, joilla yritetään täyttää organisaation ylimmän johdon määrittämät laatutavoitteet. Vaikka jokaisella ohjelmistoyrityksellä on laatujärjestelmä, joko dokumentoituna tai dokumentoimattomana [Haikala ja Märijärvi, 2003], ne eivät aina johda organisaation toimintaa tilaan, joka täyttäisi johdon määrittämän laatutason. Tämän takia laatua parantavat metodit kiinnostavat ohjelmistoyrityksiä ja laadunparannuskeinoja haetaan kilpailijoilta ja muilta tuotannon aloilta.

Usea suurempi ohjelmistoyritys [SixSigmaCompanies, 2010] on ottanut perinteisestä teollisuudesta ponnistaneen Six Sigman osaksi omaa laatujärjestelmää. Six Sigma –metodologia perustuu rakenteiseen metodologiaan, jossa käytetään hyväksi tilastotieteen analyysityökaluja virheitä tuottavien prosessien kehittämiseen. Six Sigman käyttöönotosta on raportoitu parannuksia tuotannon laadussa, sekä tehokkaamman tuotantoprosessin tuottamista säästöistä. Vaikka Six Sigma on saanut paljon huomiota, sitä ei ole kuitenkaan tutkittu paljon.

Tässä tutkielmassa perehdytään Six Sigmaan, sen toimintamalleihin ja piirteisiin. Tutkielmassa selvitetään, miten Six Sigmaa voisi soveltaa ohjelmistokehitykseen ja ohjelmistotuotantoprosessin laadun kehittämiseen. Tutkielma osoittaa myös, miten Six Sigmaa ei voi hyödyntää ohjelmistokehityksessä. Lisäksi tutkielmassa esitellään havaintoja Six Sigman hyödyntämisestä Scrum-kehitysmalliin ja yksikkötestaukseen.

Luvussa 2 esitellään ohjelmistolaatu ja ohjelmistolaadun parantamiseen käytettäviä työkaluja ja toimintatapoja. Kolmannessa luvussa esitellään erilaisia ohjelmistotuotannon kehitysmalleja ja laatujärjestelmiä, joita voi hyödyntää ohjelmistotuotantoprosessissa. Luvussa 4 esitellään ja kuvataan Six Sigma, sekä sen roolirakenne ja toimintamallit. Viidennessä luvussa pohditaan Six Sigman soveltamista ohjelmistokehitykseen kirjallisuuden ja aikaisempien tutkimusten perusteella.

2. Ohjelmiston laatu

Laadun kehittämiseen on kehitetty useita mittareita ja työkaluja. Tässä luvussa esitellään ohjelmistolaatua ja sen kehittämiseen liittyviä käsitteitä.

2.1. Ohjelmistolaatu

Laatu on käsitteenä hankala, sillä jokainen kokee laadun subjektiivisesti. Jos kuitenkin tietty ryhmä yksilöitä odottaa arvioitavan kohteen täyttävän samat vaatimukset, he voivat kokea laadun suhteellisen samalla tavalla. Edellinen huomio pätee myös käännetysti, eli puutteet voidaan kokea myös suhteellisen samalla tavalla.

Yleisesti voisi ajatella, että laadukas ohjelma tai ohjelmisto täyttää asiakkaan asettamat vaatimukset, toimii virheettömästi, valmistuu aikataulussa ja pysyy budjetissa. Britannican tietosanakirjassa [2009] laatu oli määritelty seuraavasti: perustavanlaatuinen aistittava ominaisuus, joka tekee siitä pohjimmiltaan erilaisen muista aistimuksista, erinomaisuuden vertailuaste (the attribute of an elementary sensation that makes it fundamentally unlike any other sensation, a degree of excellence). Loppukäyttäjä ei tosin voi aistia ohjelmistotuotetta kaikilla inhimillisillä aisteilla, koska se ei ole fyysinen objekti. Käyttäjä voi kuitenkin aistia laatua ohjelmiston käytettävyydestä ja toimivuudesta. Asiakas voi aistia laatua myös ohjelmistotuotantoprosessista, jolla asiakkaan järjestelmä tuotetaan. Ohjelmistotuotantoprosessista aistittavat laatutekijät liittyvät tapaan, jolla asiakas otetaan huomioon prosessin aikana, kuinka prosessi täyttää sille asetetut resurssirajoitukset ja kuinka hyvin asiakasvaatimukset täytetään.

Ohjelmiston laatuun sisältyy muitakin kuin aistittavia ominaisuuksia. Ohjelmiston laadun ominaisuuksia on määritelty erilaisten standardien muodossa. Esimerkkinä mainittakoon ISO 9126, jossa määritellään laatukäsitteet, kuten siirrettävyys (portability), tehokkuus (efficiency), luotettavuus (reliability), toiminnallisuus (functionality), käytettävyys (usability), ylläpidettävyys (maintainability) ja laajennettavuus (extensibility) [Al-Kilidar et al., 2005].

Siirrettävyys liittyy asioihin, jotka mahdollistavat tai rajoittavat järjestelmän, tai sen osan, siirtämisen toiseen ympäristöön. Ympäristöllä voidaan tarkoittaa poikkeavaa teknistä ympäristöä, kuten eri prosessorikannalla toimivaa laitteistoa tai eri käyttöjärjestelmää. Ympäristöllä voidaan myös tarkoittaa liitännäisjärjestelmien aikaisempia versioita, joiden kanssa uudemman version tulisi olla yhteensopiva. Tehokkuus liittyy fyysisten resurssien käyttömäärään, kun järjestelmää käytetään käyttötarpeeseen.

Luotettavuus liittyy järjestelmän toimintakykyyn tietyllä aikavälillä, jonka aikana järjestelmän tulisi toimia mahdollisimman virheettömästi ja toipua virheistä menettämättä toimintakykyään.

Toiminnallisuus liittyy järjestelmään tehtyjen toimintojen kykyyn täyttää käyttäjien todelliset tarpeet. Tämän lisäksi toiminnallisuus liittyy järjestelmän kykyyn täyttää vaaditut standardit ja lakimääräykset. Myös tietoturvallisuus on liitetty toiminnallisuuteen. Tietoturvallisuus liittyy järjestelmän kykyyn suojata käsiteltävä data käsittelyssä sekä tallennettuna.

Käytettävyys liittyy järjestelmän käytettävyyteen liittyviin tekijöihin. Käytettävyyteen kuuluvat mm. käytön ymmärrettävyys, oppimisnopeus, käytön tehokkuus ja käyttöliittymän puoleensavetävyys.

Ylläpidettävyys liittyy järjestelmään tehtävien muutosten työmäärään ja vaikeuteen. Mitä ylläpidettävämpi järjestelmä on, sitä nopeammin järjestelmään voidaan tehdä muutos, joka ei tuo mukanaan uusia järjestelmävirheitä.

Laajennettavuus liittyy siihen, kuinka hyvin järjestelmän suunnittelussa on otettu tulevaisuuden todennäköiset muutokset huomioon ja täten varauduttu niihin. Varautuminen näkyy tällöin esim. mahdollisina arkkitehtuuripäätöksinä tai muissa keskeisissä ratkaisuissa.

Edellä mainitut laatukäsitteet ovat sanoina kuvaavia ja käsitettävissä subjektiivisesti. Jotta laatu voidaan yksikäsitteisesti määrittää, se pitää pystyä mittaamaan. Tälle mittaamiselle on olemassa käsite ohjelmistometriikka (software metrics) tai lyhyemmin, metriikka. Metriikka tähtää tarkkaan laadun mittaamiseen ja tarkkojen laatumittareiden kehittämiseen. Metriikka edustaa epäsuoraa mittausta, eli laatua ei ikinä voida mitata suoraan, vaan jonkin laadun ilmentymän kautta. Mittarin määrittämisen hankaluus piilee laatumittarin ja todellisen ohjelmistolaadun suhteen välillä [Pressman, 2005, s. 465].

2.2. Ohjelmistotuotantoprosessin kehittämisen ja ohjelmistolaadun suhde

Määritellyt standardit määrittelevät joukon kehityskriteerejä, jotka ohjaavat tapaa, jolla ohjelmisto tuotetaan. Jos kriteerejä ei noudateta, seurauksena lähes aina ohjelmistolaatu heikkenee [Pressman, 2005, s. 463]. Edellisestä voi päätellä, että ohjelmistotuotantoprosessi vaikuttaa tuotetun ohjelmiston laatuun. Ohjelmistotuotantoprosessin tulisi varmistaa, että tuotettava ohjelmisto täyttää asiakasvaatimukset, toimii virheettömästi ja valmistuu aikataulussa ja suunnitelluilla resursseilla. Jos näin ei kuitenkaan käy, niin missä on vika? Aivan kuten muilla teollisuuden aloilla, myös ohjelmistokehityksen tuotantoprosessi tulee kehittää tilaan, missä se tuottaa mahdollisimman vähän virheitä ja on samalla suorituskyvyltään kilpailukykyinen kilpailijoihin nähden.

Ohjelmistotuotanto poikkeaa perinteisestä teollisuudesta merkitsevästi, joka vaikuttaa myös tapaan, jolla ohjelmistotuotteita tuotetaan. Ohjelmistotuotannon ja perinteisen teollisuuden keskeisiä eroja ovat [Pressman, 2005, s. 35–39]:

- Ohjelmistotuote ei ole kosketeltavissa, koska se on immateriaalituote. Tämän takia tuotetta ei voi tutkia tai mitata fyysisesti. Tämä tekee myös tuotteen todellisen valmiusasteen määrittämisestä vaikeaa.
- Ohjelmistotuotannolle ei ole standardia ohjelmistotuotantoprosessia. Ohjelmistotuotantoprosessit ovat organisaatiokohtaisia ja voivat poiketa toisistaan huomattavasti.
- Suuremmat ohjelmistotuotantoprojektit ovat ainutlaatuisia ja ainutkertaisia. Suurista projekteista opittuja asioita ei pakosta voi hyödyntää toisissa projekteissa, johtuen niiden ainutlaatuisuudesta.
- Ohjelmistotuotteet voivat olla erittäin monimutkaisia. Samalla niiltä voidaan vaatia erittäin korkeaa toimintavarmuutta. Tällaisia järjestelmiä ovat kaikki järjestelmät, jotka liittyvät tavalla tai toisella tilanteeseen, missä järjestelmävirhe voi aiheuttaa loukkaantumisia tai ihmisuhreja. Avaruusluotaimien ja –sukkuloiden järjestelmät ovat myös hyviä esimerkkejä monimutkaisista järjestelmistä, joiden toimintavarmuus on ehdottoman tärkeää.

Koska korkea ohjelmistolaatu on ohjelmistokehityksessä kilpailuetu, on ohjelmistokehitysprosessin kehittäminen kohti korkeampaa laatua perusteltua. Tämä seikka on tunnistettu yleisesti ja ohjelmistotuotantoprosessin kehittämiseen on kehitetty useita eri malleja.

Ohjelmistotuotantoprosessi on mallinnettavissa, jaksotettavissa ja ennen kaikkea, mitattavissa. Mittarit ovat yksiselitteisiä havaintoja prosessin toiminnasta, eli ne kertovat prosessin tehokkuuden ja sen kehittymisen tai rappeutumisen.

2.3. Prosessimittarit ja prosessin mittaus

Prosessi yleensä koostuu toistettavista tehtävistä, jotka suoritetaan tietyssä järjestyksessä. Prosessista voidaan tunnistaa kohtia, jotka vaikuttavat prosessin tulokseen ratkaisevasti. Ainut tapa tutkia prosessin kuntoa ja tehokkuutta on mitata sitä. Tällöin projektin kriittisiin pisteisiin liitetään prosessimittareita, joiden avulla prosessin suorituskyky ja tuotettu laatu voidaan todentaa. Prosessin mittausjärjestelmän tulee olla luotettava ja sen tulokset pitää pystyä toistamaan nopeasti [Keller, 2005, s. 86]. Nopea toistettavuus mahdollistaa prosessin muokkaamisen nopeasti, koska muutoksen tulokset saadaan nopeasti. Hyvä mittausjärjestelmä mahdollistaa väliaikaisten poikkeavuuksien tunnistamisen todellisista prosessin ongelmista.

Prosessimittarit ovat yksiselitteisiä numeerisia arvoja, jotka tuottavat prosessivaiheen toiminnasta analysoitavaa dataa. Prosessimittareista on tunnistettu kolme keskeistä ryhmää: laadun kannalta kriittiset (Critical To Quality), kustannusten kannalta kriittiset (Critical To Cost) ja aikataulun kannalta kriittiset mittarit (Critical To Schedule) [Keller, 2005, s. 87–93]. Laadun kannalta kriittiset mittarit riippuvat

tuotettavasta tuotteesta ja sen asiakasvaatimuksista. Tarkoituksena on varmistaa, että prosessi tuottaa asiakasvaatimukset täyttävän tuotteen. Kustannusten kannalta kriittiset mittarit mittaavat tekijöitä, joiden vaihtelu vaikuttaa kustannuksiin voimakkaasti. Kustannuksiin liittyvät mittarit voivat olla samalla laadullisia tai aikataulun kannalta kriittisiä mittareita. Aikataulun kannalta kriittiset mittarit mittaavat tavalla tai toisella prosessin ja prosessin vaiheiden kestoa. Nämä mittarit kertovat prosessin tehokkuudesta ja suorituskyvystä.

2.4. SIPOC-analyysi

SIPOC-lyhenne tulee sanoista Suppliers (toimittajat), Inputs (syötteet), Process (prosessi), Outputs (tulosteet) ja Customers (asiakkaat). Analyysia käytetään projektin osa-alueiden ja rajojen tunnistamiseen. Rajojen ja osa-alueiden tunnistaminen auttavat ymmärtämään ja hahmottamaan projektin merkitystä ja kulkua. SIPOC-analyysi vastaa kysymyksiin “Ketkä/Mitkä tuottavat syötteitä prosessiin?”, “Mitä määrittelyjä sijaitsee syötteissä?”, “Keitä ovat prosessin todelliset asiakkaat?” ja “Mitä ovat asiakkaan vaatimukset?” [iSixSigma, 2010].

Analyysi suoritetaan kuvaajana, missä edellä mainitut osa-alueet asetetaan taulukossa omiin sarakkeisiinsa ja niihin liittyviä asioita kirjataan osa-alueen sisään. Taulukossa 1 on kuvattu SIPOC-analyysi taulukkomuodossa. Taulukossa on kuvattu pinnallisesti ohjelmistotuotantoprosessista tunnistetut tekijät. Toimittajaksi on tunnistettu ohjelmistokehittäjät. Prosessi on ohjelmistotuotantoprosessi, joka vaatii syötteiksi esim. projektisuunnitelman, henkilöstöresurssit, määrittelydokumentit ja asiakasvaatimukset. Tulosteeksi on tunnistettu valmis ohjelmistotuote, sekä siihen liittyvä dokumentaatio. Ohjelmiston vastaanottavat loppukäyttäjät.

Toimittajat (Suppliers)	**Syötteet** (Inputs)	**Prosessi** (Process)	**Tulosteet** (Outputs)	**Asiakkaat** (Customers)
(Tarvittavien resurssien toimittajat)	*(Prosessin vaatimat resurssit)*	*(Ylimmän tason kuvaus toiminnasta)*	*(Prosessista syntyneet tuotokset)*	*(Tuotosten vastaanottajat)*
Ohjelmistokehittäjät	Henkilöstö, kalusto, suunnitelmat, määrittelyt ja vaatimukset	Ohjelmisto-tuotantoprosessi	Ohjelmistotuote, dokumentaatio	Loppukäyttäjät

Taulukko 1. SIPOC-analyysi [Redzic and Baik, 2006].

2.5. Virhekausaalianalyysi (Defect Causal Analysis)

Virheistä oppiminen on kehittymisen edellytys ihmisen toiminnassa. Jotta virheestä voi oppia, se on tunnistettava ja selvitettävä syy, josta virhe syntyi. Tästä syy-seuraussuhteen analysoinnista on kehitetty projektityökalu nimeltä virhekausaalianalyysi.

Virhekausaalianalyysi keskittyy syy-seuraussuhteiden ymmärtämiseen. Tällä systemaattisella menetelmällä on tarkoitus tunnistaa kausaalijärjestelmään epätoivotusti vaikuttavia tekijöitä ja minimoida niiden määrä [Card, 2006]. Kausaalianalyysi voi epäonnistua ja johtaa harhaan, jos tunnistettavia syy-seuraussuhteita ei ole määritelty. Cardin [2006] tunnisti kolme ehtoa syy-seuraussuhteelle:

1. Ehdotetulla syyllä ja seurauksella tulee olla assosiaatio tai korrelaatio.
2. Syyn tulee edeltää seurausta ajallisesti.
3. Syyn ja seurauksen yhdistävä mekanismi pitää pystyä tunnistamaan.

Cardin [2006] mukaan kausaalianalyysi on hyödyllisimmillään silloin, kun sen suorittavat samat henkilöt, jotka tuottavat virheitä, eli työskentelevät prosessissa. Toiminnan tueksi tulisi kehittää strategia, joka huomioi koulutuksen, parannusehdotusten toteuttamisen sekä tiedotuksen ja kommunikoinnin opituista asioista. Strategialla varmistetaan, että toistuvasti tehtävä analyysi tunnistaa pienetkin korjausta vaativat tekijät. Pienelläkin parannuksella on suuri merkitys pitkällä aikavälillä.

Syy-seuraussuhteissa on tunnistettu käsite juurisyy, joka on kokonaisen syy-seuraussuhdepolun ensimmäinen syy, josta koko polku saa alkunsa. Tämä tarkoittaa sitä, että syy voikin olla seuraus syvemmästä syystä [Rooney and Hopen, 2005]. Periaatteessa jos juurisyyn voi estää, sen seuraus ei aiheuta muita syitä ja kyseinen syy-seuraussuhde saadaan ehkäistyä.

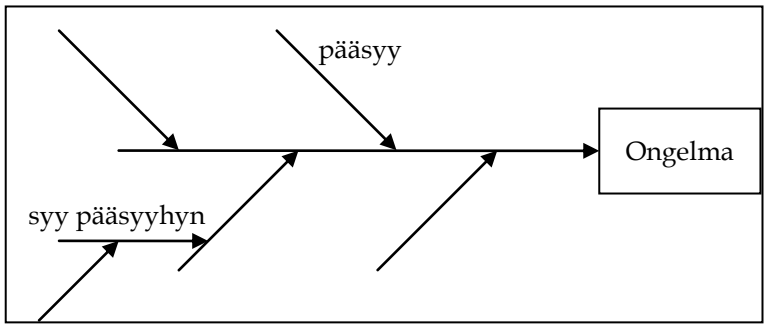
2.5.1. Five Whys

Virhekausaalianalyysin voi toteuttaa monella tavalla. Päätehtävänä on tunnistaa perimmäiset syyt epätoivotuille ilmiöille. Esimerkiksi Toyota-yrityksen analyyseista löytyi kyselytekniikka Five Whys, jonka mukaan juurisyyllä olisi yleensä neljä seurausta. Kysymys ”miksi” pitää kysyä viisi kertaa siten, että edellisen kysymyksen vastaus asetetaan uudelleen kyseenalaiseksi [Melton, 2005].

2.5.2. Ishikawa-kaavio

Virhekausaalianalyysin ehkä tunnetuin työkalu on Ishikawa-kaavio, joka tunnetaan myös nimellä kalanruotokaavio (fishbone diagram) [Kelley, 2000]. Ishikawa-kaaviossa ongelma merkitään vaakaviivalla, johon liitetään viistoilla viivoilla havaitut pääsyyt

ongelmaan. Pääsyihin liitetään uusia haaroja, toissijaisia syitä, jotka aiheuttavat pääsyyn. Ishikawa-kaavio on kuvattu kuvassa 1. Ishikawa-kaavio toimii hyvänä dokumentaationa analyysin tuloksista.

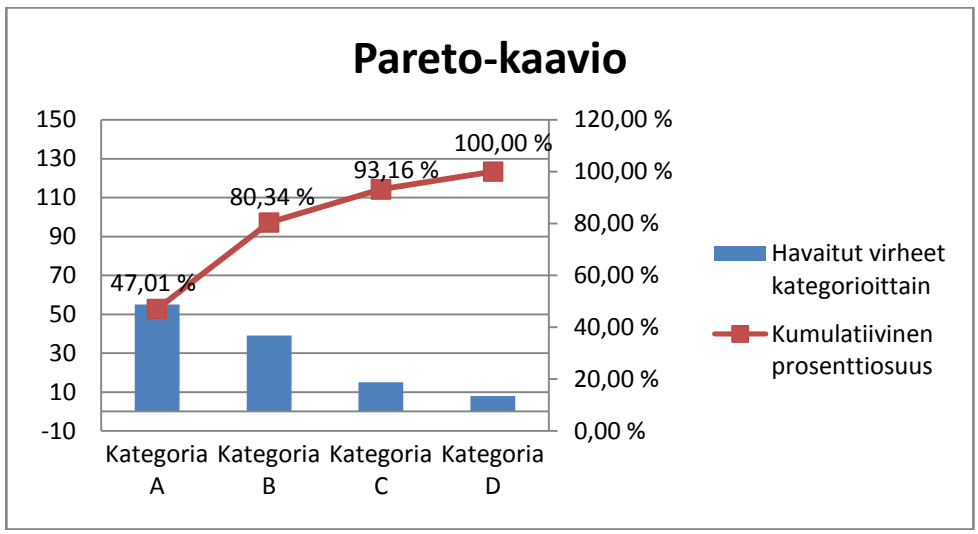


Kuva 1. Ishikawa-kaavio.

2.5.3. Pareto-analyysi

Pareto-kaavion tehtävänä on osoittaa, mikä syy tai virhe on yleisin. Tämän tiedon avulla kehitysresurssit voidaan suunnata kohteisiin, jotka aiheuttavat eniten ongelmia. Pareto-kaaviolla on kyky osoittaa 80/20-säännön toteutumisen, missä 80 % ongelmista johtuu 20 % mahdollisista syistä [Kelley, 2000].

Pareto-kaaviota varten ongelman syyt pitää kategorisoida. Ongelmaan johtavat tilanteet tulee mitata tietyn aikaa. Tämän jälkeen kategorioiden frekvenssit lasketaan ja asetetaan laskevaan järjestykseen pylväskuvaajaan. Pylväiden lisäksi samaan kuvaajaan lisätään käyrä, joka osoittaa kumulatiivisen kategorioiden prosenttiosuuden havaintojen kokonaispopulaatiosta. Pareto-kaavio on kuvattuna kuvassa 2.



Kuva 2. Pareto-kaavio.

2.6. Kano-malli

Asiakkaat arvioivat laatua eri näkökulmista ja tekijöistä. Tämän takia on tärkeätä kartoittaa, mitkä tekijät tuottavat asiakkaan mielestä enemmän laatua ja mitkä vähemmän. Kano-malli on kehitetty selittämään suhteet, jotka vallitsevat asiakastyytyväisyyden ja tuotteen kriteerien kesken. Kano-malli jakaa tuotekriteerit viiteen kategoriaan. Kategoriat ovat Chenin ja Chuangin [2008] mukaan seuraavanlaiset:

1. Pakollinen laatu (Must Be) sisältää ominaisuuksia, joiden huono toiminta tai puuttuminen laskee laatua. Ominaisuuksien keskimääräistä parempi toteutus ei kuitenkaan nosta asiakastyytyväisyyttä.
2. Yksiulotteinen laatu (One-dimenional) sisältää ominaisuuksia, joissa asiakastyytyväisyys kasvaa lineaarisesti toteutuksen laadun kanssa.
3. Vetovoimainen (Attractive) laatu sisältää ominaisuuksia, jotka tuottavat asiakastyytyväisyyttä eksponentiaalisesti toteutuksen laadun mukaan. Asiakastyytyväisyys ei kuitenkaan laske huonommassa toteutuksessa.
4. Yhdentekevä (Indifference) laatu sisältää ominaisuuksia, joiden keskimääräistä parempi toteutus ei nosta asiakastyytyväisyyttä.
5. Käänteinen (Reversal) laatu liittyy ominaisuuksiin, joiden laadukkaampi toteutus laskee asiakastyytyväisyyttä.

		Ominaisuuden vaillinainen toteutus				
		V1	**V2**	**V3**	**V4**	**V5**
Ominaisuuden täysi toteutus	**V1**	*L6*	*L1*	*L1*	*L1*	*L2*
	V2	*L5*	*L4*	*L4*	*L4*	*L3*
	V3	*L5*	*L4*	*L4*	*L4*	*L3*
	V4	*L5*	*L4*	*L4*	*L4*	*L3*
	V5	*L5*	*L5*	*L5*	*L5*	*L6*

Vastausasteikko	
V1	Tyytyväinen
V2	Asian pitäisi olla näin
V3	Yhdentekevää
V4	Pystyn elämään asian kanssa
V5	Tyytymätön

Laatukategoria	
L1	Pakollinen laatu
L2	Yksiulotteinen laatu
L3	Vetovoimainen laatu
L4	Yhdentekevä laatu
L5	Käänteinen laatu
L6	Ristiriitainen

Kuva 3. Kano-mallin analyysitaulukko [Chen and Chuang, 2008].

Chenin ja Chuangin [2008] mukaan ominaisuuksien laatukartoitus toteutetaan kahdella kyselyllä, joista ensimmäisessä oletetaan, että tuote sisältää halutun

ominaisuuden tai asiakasvaatimuksen. Asiakas vastaa kysymyksiin asteikolla ”tyytyväinen” (”satisfied”), ”asian pitäisi olla näin” (”it should be that way”), ”yhdentekevää” (”I am indifferent”), ”pystyn elämään asian kanssa” (”I can live with it”) ja ”tyytymätön” (”dissatisfied”). Toisessa kyselyssä asettelu käännetään päinvastoin, eli asiakas vastaa samalla asteikolla oletuksiin, että ominaisuus tai asiakasvaatimus puuttuu tuotteesta tai on vaillinainen. Kyselyjen tulokset analysoidaan kano-taulukolla, jonka avulla ominaisuudelle voi asettaa laatukategorian. Kano-taulukko on kuvattuna kuvassa 3. Taulukossa kahden eri kyselykierroksen tulokset paljastavat laatukategorian siten, että riveillä on ensimmäisen kyselyn vaihtoehdot, missä ominaisuus on toteutettu ja sarakkeissa toisen kyselyn vaihtoehdot, missä ominaisuuden toteutus on vaillinainen. Esimerkiksi, jos asiakas on tyytyväinen toteutukseen (V1) ja tyytyväinen siihen, että toteutus on vaillinainen (V1), niin kyseessä on ristiriitainen laatukategoria (L6). Laatukategoria on sama (L6), jos asiakas vastaa kummallakin kierroksella olevansa tyytymätön (V5).

2.7. Katselmointi ja auditointi

Katselmointi on laadunhallintaan kuuluva aktiviteetti, jossa katselmoinnin kohteena oleva toteutus tutkitaan ja validoidaan ohjelmistokehittäjien ja muiden tarpeellisten asianomaisten osalta. Katselmoinnit ovat muodollisia tilaisuuksia, joiden tarkoituksena on löytää virheitä toteutuksesta, varmistaa asiakasvaatimuksen täyttyminen tarkasteltavan toteutuksen osalta, varmistaa, että ohjelmistotuote täyttää määritetyt standardit, varmistaa ja että ohjelmisto kirjoitetaan yhtenäisellä tavalla [Pressman, 2005, s. 754].

Katselmointi onnistuu todennäköisemmin, jos se suunnitellaan etukäteen ja ohjataan läpi asianmukaisesti. Katselmoinnissa tulisi olla 3-5 henkilöä. Heidän tulisi valmistautua katselmointiin, mutta valmistautuminen saisi viedä enintään kaksi tuntia jokaiselta osanottajalta. Itse katselmoinnin tulisi kestää enintään kaksi tuntia [Pressman, 2005, s. 755]. Yksi osanottajista ottaa tallentajan roolin ja kirjoittaa ylös kaikki keskeiset huomiot katselmoinnista. Katselmoinnin päättyessä osanottajien on tehtävä päätös katselmoitavan toteutuksen suhteen. Heidän täytyy yksimielisesti antaa yksi seuraavista päätöksistä:

- Toteutus hyväksytään ilman muutoksia.
- Toteutus hylätään vakavien virheiden takia (vaatii uuden katselmoinnin).
- Toteutus hyväksytään pienin korjausvaatimuksin (ei vaadi uutta katselmointia).

Tiedonjakamisen kannalta katselmoinnit toimivat kehittäjille tilaisuuksina oppia tuntemaan järjestelmän toiminnallisuudet, joita he eivät ole olleet toteuttamassa.

Katselmoinnin kohteena olleen toteutuksen tuottaja saa katselmoinnista mahdollisesti uusia näkökulmia ongelmanratkaisuun ja täten kehittää yksilön kykyjä.

Auditointi on laadunvarmistuksen aktiviteetti, jossa varmistetaan, että todellinen toiminta vastaa suunniteltua ja määriteltyä toimintaa [McDonald, 2002]. Auditointi voi olla ulkoinen tai sisäinen. Ulkoisessa auditoinnissa organisaation ulkopuolinen, riippumaton tekijä tarkastaa todellisen toiminnan, esimerkiksi laatujärjestelmän osalta. Sisäinen auditointi on laadunvarmistusta, missä organisaation sisäinen elin varmistaa toiminnan vastaavuuden määritellylle toiminnalle. Tarkoituksena on siis selvittää, miten organisaatiossa oikeasti toimitaan. Tulokset voivat paljastaa organisaation toiminnasta ongelmakohtia, sekä tuloksista voi tunnistaa kehityskohteita.

2.8. Koodirivimäärä mittarina

Ohjelmistoprojektin koon arvioinnin kannalta on tärkeää pystyä mittaamaan ohjelmiston kokoa yksiselitteisellä ja vertailukelpoisella mittarilla. Koska alimmalla tasolla ohjelmisto koostuu koodiriveistä, koodirivien määrä tuntuu sopivalta kokomittarilta. Koodirivimäärä ilmoitetaan yleensä lyhenteillä LOC (Lines Of Code) tai KLOC (Kilo Lines Of Code). Koodirivimäärä ei ole kuitenkaan täysin vertailukelpoinen mittari. Pressman [2005] mainitsee, että koodirivimäärämittari on riippuvainen ohjelmointikielestä, se kohtelee kaltoin hyvin suunniteltuja lyhyempiä ohjelmia ja sen käyttö arvioinnissa vaatii tarkkaa matalan tason suunnittelua ohjelmistoprojektin alussa.

2.9. Tilastotieteen käsitteitä

Tässä kappaleessa esitellään tilastotieteen käsitteitä, joiden ymmärtäminen on edellytyksenä Six Sigman metodologian periaatteiden käsittämiselle. Tässä kappaleessa esitellyt käsitteet kuuluvat suurimmaksi osaksi tilastotieteen perusteihin, joten käsitteiden ymmärtäminen ei vaadi syvempää tilastotieteen tuntemista.

2.9.1. Keskiarvo

Keskiarvo, havaintojen aritmeettinen keskiarvo on eniten käytetty kvantitatiivisen muuttujan jakauman keskiluku [Grönroos, 2003]. Aritmeettinen keskiarvo on arvojoukon jäsenten summa jaettuna jäsenten lukumäärällä.

Keskiarvoja on useita erilaisia [Grönroos, 2003], mutta tässä tutkielmassa käsitellään ainoastaan aritmeettista keskiarvoa. Koko populaation keskiarvoa merkitään kreikkalaisella kirjaimella μ ja otoksen keskiarvoa merkillä $\bar{x}$.

2.9.2. Normaalijakauma

Normaalijakauma on jatkuva jakauma, joka on kaikkein yleisimmin käytetty ja tärkein jakauma tilastotieteessä [Grönroos, 2003]. Syy yleisyyteen on se, että jakauman käyttö voidaan perustella keskeisellä raja-arvolauseella. Tämän takia normaalijakaumaan

perustuvia malleja käytetään aina, kun niiden käyttö voidaan perustella empiirisesti tai teoreettisesti [Grönroos, 2003].

Suurin hankaluus, joka normaalijakaumaan liittyy, on se, että jakauman kertymäfunktio ei ole esitettävissä suljetussa muodossa. Tunnettuja riittävän tarkkoja laskukaavoja on useita, joten tietokoneita hyödyntämällä todennäköisyydet ovat helposti laskettavissa. Yleisimmän hyödynnetyn normaalijakauman, standardinormaalijakauman, arvot löytyvät taulukkokirjoista positiivisten pisteiden osalta.

2.9.3. Keskihajonta

Keskihajonta on luku, joka kuvaa arvojen ryhmittymistä keskiarvonsa ympärille. Mitä lähemmäs keskiarvoa ja siis myös toisiaan arvot ovat, sitä pienempi on keskihajonta. Hajallaan sijaitsevien, eli keskenään kovin erisuurien lukujen keskihajonta on iso. Tilastotieteessä keskihajontaa merkitään kreikkalaisella kirjaimella sigma (σ), mutta myös kirjaimella s. Keskihajonnan kaava on kuvattu kaavassa 1.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n-1}}$$

μ=keskiarvo
N=population koko

Kaava 1. Keskihajonnan kaava [Grönroos, 2003].

2.9.4. T-testi

Studentin t-testi on tilastollinen testi, joka noudattaa t-jakaumaa, kun nollahypoteesi on voimassa [Grönroos, 2003]. Nollahypoteesi on tilastotieteellisessä tutkimuksessa tehtävä olettamus, jonka mukaan tutkittavien ryhmien välillä ei ole perustavaa eroa, vaan ryhmien välillä olevat pienet erot johtuvat otannan satunnaisvaihtelusta. Vaikka tässä tutkielmassa ei paneuduta edellä mainittuun kuvaukseen syvemmin, t-testin käytöstä voidaan todeta, että sillä voi testata normaalijakautuneiden satunnaismuuttujien keskiarvoja. Yksi käyttötapa on verrata kahden riippumattoman otoksen keskiarvojen yhtäsuuruutta.

3. Ohjelmistojen kehitysmallit ja laatujärjestelmät

Kehitysmallit ja organisaatiota ohjaavat toimintamallit ovat tärkeitä tuotettavan laadun kannalta, koska ne ohjaavat ohjelmistotuotantoprosessin suoritusta. Tässä luvussa esitellään kehitysmalleja, jotka liittyvät suoraan ohjelmistotuotantoprosessiin tai muuten organisaation toiminnan ohjaamiseen.

3.1. Total Quality Management

Total Quality Management on integroitu lähestymistapa, jolla tavoitellaan lopputuotteen korkean laadun saavuttamista ja laadun ylläpitämistä keskittymällä prosessien huoltoon, jatkuvaan kehittämiseen ja virheiden ennaltaehkäisyyn kaikilla tasoilla ja organisaation toiminnoissa, jotta asiakasvaatimukset voidaan täyttää tai ylittää [Boaden, 1997].

TQM koostuu laadunhallinnan käytännöistä ja aatteista, joita siihen on liitetty 1950-luvulta lähtien. Toisin sanoen, TQM ei sisällä mitään omalaatuista tai ainutlaatuista, vaan se kokoaa jo hyväksi tunnistettuja asioita yhdeksi kokonaisuudeksi [Boaden, 1997]. TQM oli kuitenkin tunnistettu kilpailukyvyn edistäjäksi, jonka myötä se on ollut myös laajasti käytössä. TQM:n sisältö oli 1990-luvulle asti määrittelemätön, minkä takia sisällöstä oltiin eri mieltä sovelluspiireissä [Boaden, 1997]. Boaden [1997] tutki TQM-käsitettä ja listasi TQM:n yhteisesti hyväksytyt periaatteet:

- Asiakaslähtöisyys.
- Ylimmän johdon vastuu laadunkehitysprosessista.
- Työntekijöiden ja heidän tietämyksen kunnioittaminen.
- Työntekijät ovat aktiivisesti mukana laadunkehitysprosessissa.
- Prosessi- ja tuotehajonnan vähentäminen.
- Työntekijöiden jatkuvan koulutuksen ja harjoittelun mahdollistaminen.
- Tilastollisen ajattelutavan sisäänajo ja tilastollisten metodien hyödyntäminen koko organisaatiossa.
- Painoarvo virheiden ehkäisyllä tunnistamisen sijaan.
- Toimittajien näkeminen pitkäaikaisina yhteistyökumppaneina.
- Suorituskyvyn mittarit, jotka ovat johdonmukaiset organisaation tavoitteiden kanssa.
- Pitäytyminen parhaissa toimintatavoissa tehtävien suorittamiseen.
- Painoarvo laadulla tuotteiden ja palveluiden suunnittelussa.
- Organisaation kaikkien toimintojen yhteistyö ja osanotto.
- Sisäisten asiakkaiden tarpeiden tiedostaminen.
- Huomattava organisaatiokulttuurillinen muutos.

3.2. Lean-malli

Lean-mallin kehitti Taiichi Ohto Toyotalla 1950-luvulla [Hines et al., 2004]. Lean-malliin liittyy keskeisesti hukka-käsite, eli toiminto tai asia, joka kuluttaa resursseja lisäämättä arvoa prosessiin tai lopputulokseen. Lean-malliin liittyy loputon yritys poistaa tai vähentää hukka suunnittelusta, valmistamisesta, jakelusta ja asiakaspalvelusta [Durkee, 2008].

Lean-malliin liittyy arvovirtojen kartoittaminen, jossa tunnistetaan tekijät, jotka organisaation toiminnassa lisäävät arvoa lopputulokseen [Chaneski, 2002]. Jos sisäinen hukka vähenee, hukkaa tuottavat toiminnot ja niihin liittyvät kustannukset vähenevät ja asiakkaalle tarjolla oleva kokonaisarvo lisääntyy [Hines et al., 2004].

3.3. CMM ja CMMI

Software Engineering Institute on kehittänyt kaksi tunnettua kypsyysmallia, Capability Maturity Model –kypsyysmallin (CMM) sekä Capability Maturity Model Integration –kypsyysmallin (CMMI). Tässä kappaleessa esitellään edellä mainitut kaksi kypsyysmallia ja niiden keskeiset erot.

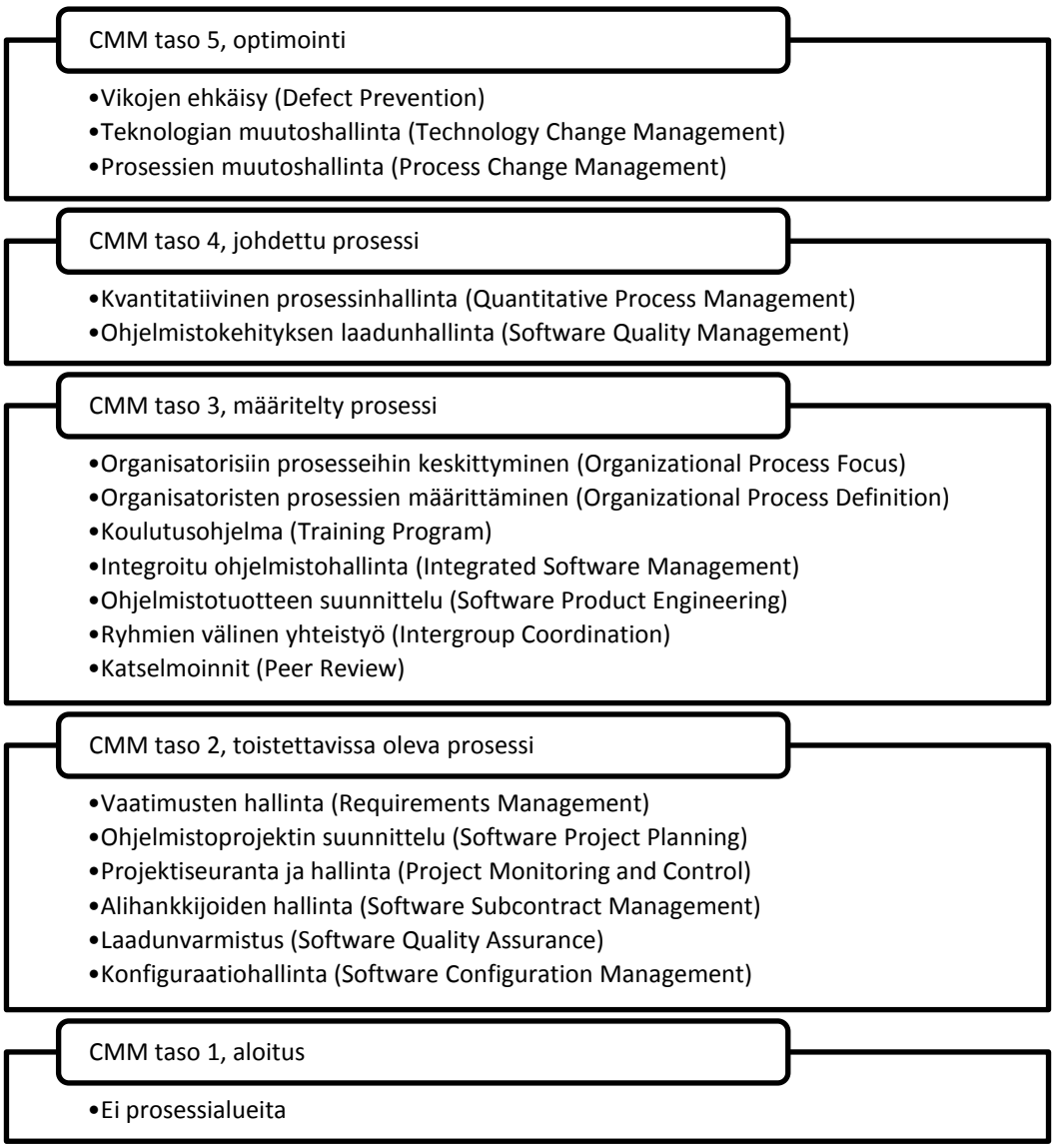
3.3.1. Capability Maturity Model

CMM-kypsyysmalli julkaistiin vuonna 1989 ja mallia kehitettiin vuoteen 1997 asti. CMM:n alkuperäinen tarkoitus oli toimia työkaluna ohjelmistotoimittajien toimituskyvyn arvioinnissa. CMM-mallia on erityisesti käytetty ohjelmistotuotantoprosessin kypsyysmallina. CMM-mallista kuitenkin kehittyi muille aloille erikoistuneita malleja, joten ohjelmistokehitykselle luotiin oma malli, jolle käytettiin tunnistetta SW-CMM (SoftWare-CMM).

Malli sisältää viisi kypsyystasoa, jotka ovat kuvattuna tasoittain kuvassa 4. Kuvasta ilmenee, mitkä avainprosessialueet pitää toteuttaa eri tasoja tavoiteltaessa. Kypsyystasot koostuvat avainprosessialueista (Key Process Area), joita CMM-mallissa on yhteensä 18 kappaletta [SEI, 2002]. Jokaiselle avainprosessialueelle määritellään tavoitteet (Goals), sitoutuminen (Commitment), kyvykkyys (Ability), mittaaminen (Measurement) ja todentaminen (Verification). Avainprossessialueiden sisällöt ovat seuraavat [SEI, 2002]:

- *Konfiguraatiohallinta*: Kaikkien projektin tuotteiden eheyden ylläpitäminen. Tämä tehdään seuraamalla ja hallinnoimalla artefaktien muutoksia ja versioiden hallintaa.
- *Laadunvarmistus*: Tuotteiden laatu on varmistettava, jotta ne täyttävät laatuvaatimukset ennen kuin ne luovutetaan asiakkaille. Tämä prosessialue kattaa laadunvarmistuksen kaiken tuotettavan osalta.

- *Alihankkijoiden hallinta*: Alihankkijasuhteita hallinnoidaan yllätysten ja riskien välttämiseksi. Lisäksi on tärkeää varmistaa, että alihankkija on kykenevä toimittamaan sovitusti, eli varmistaa aikataulullisten riippuvuuksien toteutuminen.



Kuva 4. SW-CMM -mallin kypsyystasot ja tasojen avainprosessialueet [SEI, 2002].

- *Projektiseuranta ja hallinta*: Sisältää suunnitelmaa vasten tapahtuvan edistymisen seuranta. Aikataulua, budjettia ja muita kriittisiä tekijöitä tulisi seurata säännöllisesti. Havaittuja poikkeavuuksia tulee myös seurata.
- *Ohjelmistoprojektin suunnittelu*: Projektin suunnittelu, missä arvioidaan tarvittava aika ja kustannus projektille. Työ pilkotaan pienempiin kokonaisuuksiin ja jokaiselle kokonaisuudelle tulisi määrittää resurssit ja aikataulu.

- *Vaatimusten hallinta:* Tunnistettuja vaatimuksia tarvitsee hallinnoida. Tehtävän työn, projektisuunnitelman ja vaatimusten välillä ei saa olla epäjohdonmukaisuuksia.
- *Katselmoinnit:* Verifioidaan, että toteutus vastaa määritettyjä vaatimuksia jokaisessa projektin vaiheessa.
- *Ryhmien välinen yhteistyö:* Kaikkien olennaisten asianomaisten sekä projektin jäsenten on pystyttävä tekemään tarpeen mukaan yhteistyötä. Tämä prosessialue kattaa yhteistyön mahdollistamisen kaikissa projektin eri vaiheissa.
- *Ohjelmistotuotteen suunnittelu*: Sisältää asiakasvaatimuksia täyttävien ratkaisujen suunnittelun. Tämä sisältää korkean ja matalan tason suunnittelun, eli moduulisuunnittelusta arkkitehtuurisuunnitteluun.
- *Integroitu ohjelmistohallinta*: Tarkoituksena on räätälöidä organisaation ohjelmistotuotantoprosessista ja vahvuuksista prosessi, jossa ohjelmistosuunnittelu ja hallintotoiminnot integroituvat yhdeksi.
- *Koulutusohjelma*: Tarkoitus on tunnistaa koulutustarpeet yksilötasolta organisaatiotasolle ja tarjota tarvittava koulutus.
- *Organisatoristen prosessien määrittäminen*: Tarkoitus on kehittää ja ylläpitää käytettävä joukko ohjelmistotuotantoprosessiin liittyviä vahvuuksia, jotka kehittävät prosessin tehokkuutta.
- *Organisatorisiin prosesseihin keskittyminen*: Tarkoituksena on perustaa organisatoriset vastuut ohjelmistotuotantoprosessin toiminnoille, jotka kehittävät koko organisaation kyvykkyyttä.
- *Ohjelmistokehityksen laadunhallinta*: Laadunhallintaan kuuluu sekä laadunohjaus, että laadunvarmistus. Laadunohjauksella tarkoitetaan niiden tekniikoiden ja toimintojen joukkoa, joita käyttämällä pyritään täyttämään laatuvaatimukset. Laadunvarmistuksen tavoitteena on saavuttaa sekä organisaation sisäisten että ulkopuolisten asiakkaiden ja viranomaisten luottamus tavoitteiden toteutumiseen. Tämän prosessialueen tarkoitus on määrittää ja ylläpitää laadunhallinnan toteutumista.
- *Kvantitatiivinen prosessinhallinta*: Tarkoituksena on perustaa suorituskykytavoitteet, suorituskyvyn mittaaminen ja analyysitapa ja näiden toimintojen pohjalta ohjata prosessia siten, että sen suorituskyky säilyy tai paranee.
- *Prosessien muutoshallinta*: Tarkoituksena on jatkuvasti kehittää olemassa olevaa ohjelmistotuotantoprosessia. Prosessialueen tehtäviin kuuluvat mm. kehitystavoitteiden määritys ja kehityskohteiden tunnistaminen, arviointi ja toteuttaminen ohjelmistotuotantoprosessissa.
- *Teknologian muutoshallinta*: Tarkoituksena on tunnistaa hyödyllisiä uusia teknologioita ja siirtää ne hallitusti organisaation toimintaan. Prosessialueen

tehtäviin kuuluu uusien teknologioiden tunnistaminen, valinta, evaluointi ja teknologian ajaminen organisaatioon.

- *Vikojen ehkäisy*: Tarkoituksena on estää virheitä tai vikoja syntymästä. Prosessialueeseen kuuluu aikaisempien virheiden analysointi juurisyyn tunnistamiseksi ja toimenpiteiden määrittely, joiden avulla virhe ei toistuisi enää tulevaisuudessa.

Organisaatio pystyy siis määrittelemään, miten se täyttää avainprosessialueiden vaatimukset. Kypsyystaso muuttuu ulkoisen sertifioinnin kautta, missä tarkastava osapuoli arvioi tavoiteltavan kypsyystason avainprosessialueiden vaatimusten täyttymisen.

3.3.2. Capability Maturity Model Integration

CMMI ei ollut ainoa kypsyysmalli 1990-luvun lopulla vaan eri aloille kehitettiin omat kypsyysmallit prosessien kehittämiseksi. Ongelmaksi muodostui, että esim. ohjelmistokehitykselle, suunnittelulle, henkilöstöhallinnalle ja riskien hallinnalle oli omat erilliset mallit. Malleista piti integroida yksi malli, joka kattaisi kaikki osa-alueet. Tuloksena syntyi CMMI. CMMI on malli, jossa on kokoelma prosessien ja tuotteiden kehittämisen parhaita toimintatapoja [Siviy et al., 2008]. CMMI:tä voi käyttää tiekarttana prosessien toteuttamisessa ja kehittämisessä. CMMI ei kuitenkaan kerro, miten mallia käytettäessä tulisi toimia, vaan se jättää toimintatavat määrittelemättä. Kuten CMM, myös CMMI koostuu prosessialueista, joilla on omat tavoitteet, sekä toimintatavat.

Prosessin kypsyyttä mitataan viisiportaisella asteikolla. Jokaiseen portaaseen kuuluu erityinen prosessialueiden joukko, joiden tavoitteet on täytettävä ansaitakseen kyseisen tason kypsyyden. Organisaation tulee määrittää toimintatavat, jotka ajavat organisaation toimintaa kohti prosessialueen tavoitetta. Prosessikypsyys varmennetaan auditoinnilla ja kypsyystaso sertifioidaan. Kypsyystason sertifiointi on osoitus markkinoille, että organisaation toiminta täyttää tietyt prosessialueet. Kypsyystasolla voi olla suuri merkitys tuottajan valinnassa. Lisäksi, tietyillä toimialueilla vaaditaan tiettyä kypsyystasoa, jotta voi edes osallistua tarjouskilpailuun. CMMI:n kypsyystasot ovat esiteltyinä kuvassa 5. Osa avainprosessialueista on pysynyt suhteellisen muuttumattomina CMM-mallista. CMMI-mallin avainprosessialueet ovat lyhyesti kuvattuina seuraavanlaiset [Siviy et al., 2008, s. 6-16]:

- *Konfiguraatiohallinta*: Prosessialue kattaa kaikkien projektin tuotteiden eheyden ylläpitämisen.
- *Prosessien ja tuotteiden laadunvarmistus*: Laadunvarmistukseen liittyvä työ, jolla varmistetaan, että tuote vastaa vaatimuksia ja standardeja.

- *Mittausanalyysi*: Prosessialue kattaa tehtäviä, kuten mittareiden määrittäminen, mittausdatan säilöminen, raportointi ja analyysi. Tämä prosessialue tukee tarpeellisen tiedon keräämistä, joka auttaa päätöksenteossa.

Kuva 5. CMMI-mallin kypsyystasot ja tasojen avainprosessialueet [Siviy et al., 2008].

- *Toimittajienhallinta*: Prosessialue kattaa alihankkijasuhteiden hallinnoinnin. Tähän liittyy sopimuksien hallinta, kuten myös alihankkijoiden hankkimiseen liittyvien riskien hallinta.
- *Projektiseuranta ja hallinta*: Prosessialue kattaa projektien etenemisen seurannan mm. aikataulun ja resurssien kannalta. Tämä alue määrittää toimintatavat projektiseurannalle.
- *Projektisuunnittelu*: Projektisuunnitelman muodostaminen.
- *Vaatimustenhallinta*: Vaatimustenhallintaan liittyvä työ.
- *Päätöksenteon analyysi*: Päätöksenteon määrittely. Tähän liittyy päätöksentekijän tunnistaminen päätöksentekotilanteissa ja formaalin päätöksentekotavan muodostaminen.
- *Riskienhallinta*: Potentiaalisten riskien, eli asioiden, jotka voivat estää projektia saavuttamasta tavoitettaan, tunnistaminen ja niihin varautuminen. Lisäksi tämä prosessialue sisältää riskien kehittymisen seurannan, eli muuttuvatko riskit todennäköisemmiksi projektien edetessä.
- *Integroitu projektinhallinta*: Prosessialueen tehtävänä on integroida projektin asianomaiset projektiin mukaan siten, että kaikki voivat tuoda oman osuutensa projektiin.
- *Organisatorinen koulutus*: Prosessialue kattaa organisaation koulutussuunnittelun. Tarkoituksena on tunnistaa organisaation henkilöstön henkilökohtaiset koulutustarpeet tehtävien mukaan.
- *Organisatoristen prosessien määrittäminen*: Organisaation prosessien ja toimintatapojen muodollinen kirjaus. Nämä prosessit ja toimintatavat muodostavat pohjan organisaation toiminnan ymmärtämiselle.
- *Organisatorisiin prosesseihin keskittyminen*: Määrittää organisaatioon elimen, joka vastaa organisaation toimintaprosessin kehittämisestä. Tämä elin ajaa parannuksia prosessiin syvän ymmärryksen ja tiedon kautta.
- *Validointi*: Varmistetaan, että tuote on rakennettu siten, että se täyttää alkuperäisen käyttötarpeen.
- *Varmentaminen*: Prosessialue määrittää, miten jokaisessa tuotteen tuotantovaiheessa varmennetaan, että tuote on rakennettu oikein.
- *Tuoteintegraatio*: Prosessialue kattaa tuotteen kokoamisen osakomponenteista. Kokoamisen lisäksi tämä alue kattaa integraatiotestauksen.
- *Tekninen ratkaisu*: Prosessialue kattaa teknisen toteutuksen suunnittelun, eli miten ja millä teknisellä ratkaisulla asiakasvaatimukset saadaan täytettyä.
- *Vaatimusten kehitys*: Prosessialue kattaa vaatimusten määrittelyn. Vaatimusten kehityksellä viitataan asiakkaan kanssa tehtyyn yhteistyöhön, jossa vaatimukset kehittyvät alustavasta tarpeesta määritellyksi vaatimukseksi.

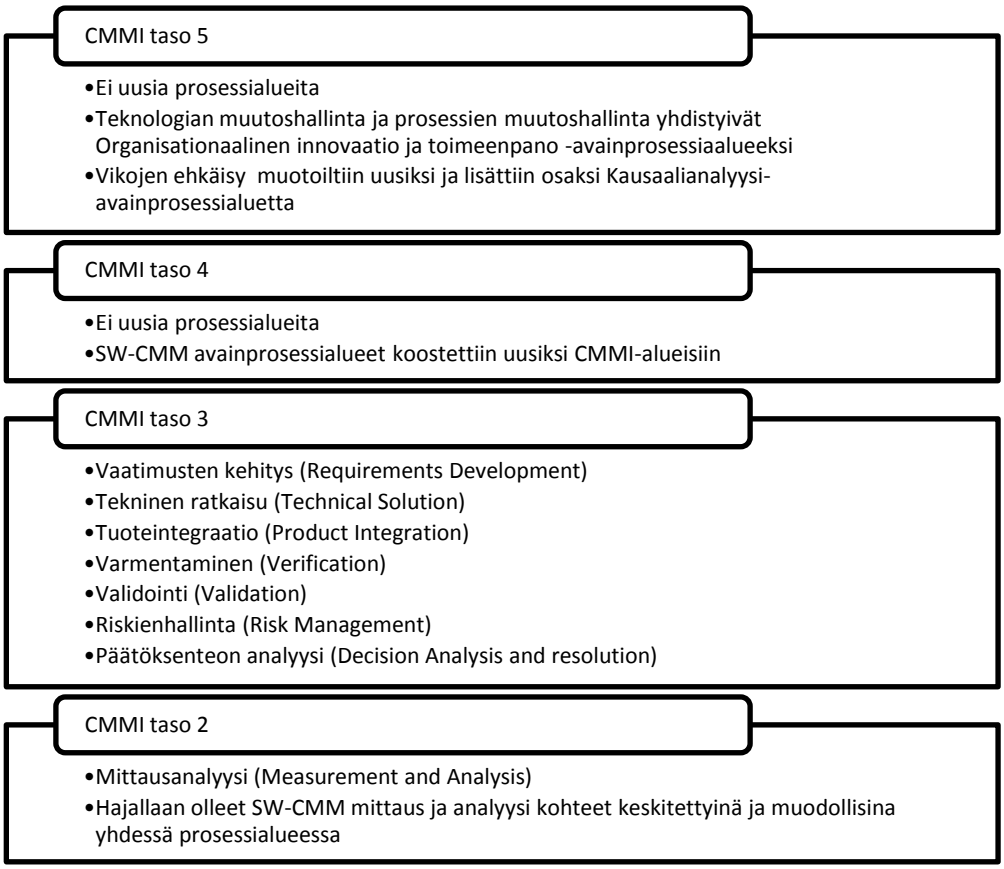
- *Kvantitatiivinen projektinhallinta*: Prosessialue kattaa organisaation tavan edetä kohti tiedon ajamaa päätöksentekoa, missä jokainen päätös perustuu mitattuun tietoon. Prosessialue määrittelee mittausprosessit ja tavan, jolla mittausprosessi liittyy projektin ja liiketoiminnan suorituskykyyn ja laatuun.
- *Organisatoristen prosessien suorituskyky*: Prosessialue kattaa organisaation prosessien suorituskykyjen kvantitatiivisen mittauksen. Prosessialue kattaa myös kerätyn tiedon tallentamisen, tiedon jakamisen ja prosessien mallintamisen.
- *Kausaalianalyysi*: Prosessialue kattaa juurisyiden tunnistamisen ongelmille, kuten myös menestystekijöille. Tämä prosessialue on keskeisesti yhteydessä kvantitatiivisen tiedon käyttämiseen päätöstenteossa.
- *Organisationaalinen innovaatio ja toimeenpano*: Prosessialue kattaa organisaation prosessin parantamisen tunnistettujen tarpeiden mukaan. Prosessin kypsyessä, siitä tunnistetaan parannuskohteita. Prosessialue tunnistaa, arvioi, valikoi ja toteuttaa parannukset hallitusti.

3.3.3. SW-CMM -mallin ja CMMI-mallin ero

CMMI:n tarkoituksena oli yhdistää parhaat puolet olemassa olevista kypsyysmalleista ja toimia SW-CMM:n seuraajana. Keskeisimmät erot SW-CMM:n ja CMMI: kesken ovat lisätyt prosessialueet, modernit parhaaksi huomatut toimintatavat ja geneerinen toteutustavoite, joka koskee jokaista prosessialuetta [SEI, 2004]. Kypsyystasojen erot on esitelty kuvassa 6. Prosessialueiden sisällöt on kuvattu edellisessä kappaleessa.

SEI esittelee SW-CMM -mallista CMMI-malliin päivittämisen keskeisimmät edut seuraavasti [SEI, 2004]:

- Hallinnollisten aktiviteettien sekä kehitysaktiviteettien yhdistäminen liiketoiminnallisiin tavoitteisiin.
- Parantunut näkyvyys tuotteen elinkaareen sekä kehitysaktiviteetteihin, jotta tuote tai palvelu täyttää asiakasvaatimukset.
- Lisätehon hakeminen lisäprosessialueiden parhaista toimintatavoista, kuten esim. mittauksesta, riskienhallinnasta ja toimittajien hallinnoinnista.
- Vankemmat korkeampien kypsyystasojen toimintatavat.
- Parantunut näkyvyys organisaation toimintoihin, jotka ovat kriittisiä tarjottujen tuotteiden tai palveluiden kannalta.
- Tiukempi pariuttaminen keskeisten ISO standardien kanssa.



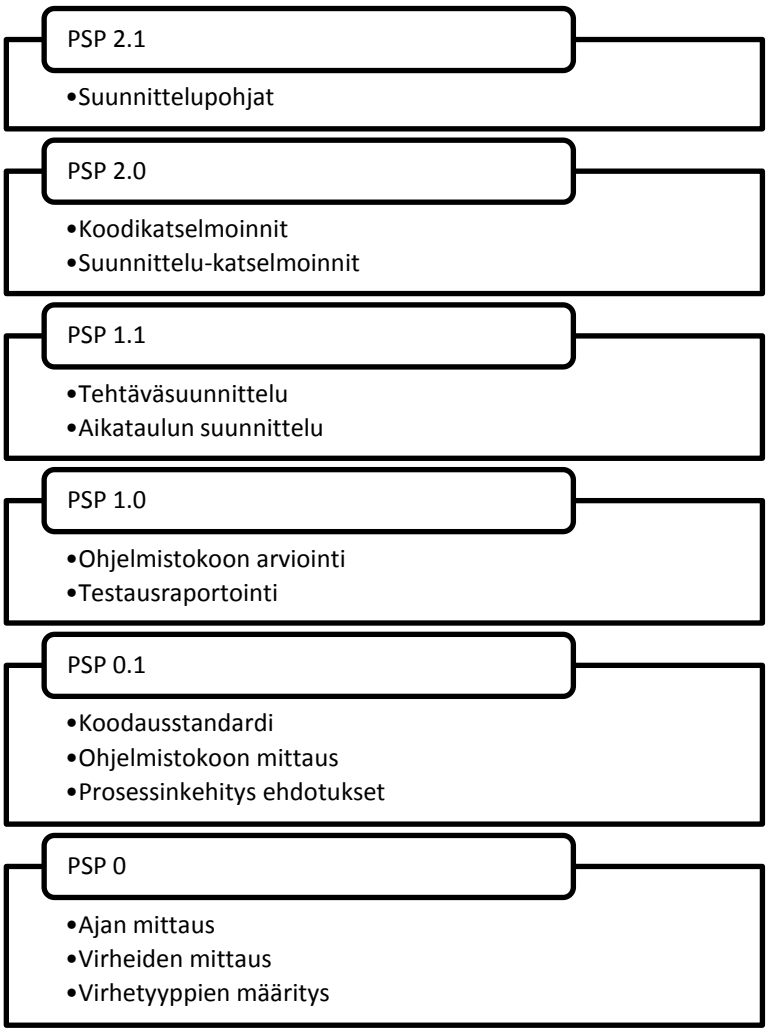
Kuva 6. SW-CMM ja CMMI-kypsyysmallien erot [SEI, 2004].

3.4. Henkilökohtainen ja ryhmäohjelmistoprosessi, PSP/TSP

Henkilökohtainen ja ryhmäohjelmistoprosessi (Personal Software Process, Team Software Process) ovat SEI:n [2009] käsitteitä, jotka perustuvat CMM/CMMI-kypsyystasomallin periaatteiden soveltamiseen henkilökohtaiseen ja ryhmätason käytäntöön. Henkilökohtainen ohjelmistotuotantoprosessi, eli PSP, näyttää, miten ohjelmistokehittäjä voi hallita tuotteiden laatua, tehdä luotettavia suunnitelmia ja perustella ne datan perusteella. Humphreyn [2000a] mukaan PSP perustuu seuraaviin periaatteisiin:

- Jokainen kehittäjä on erilainen. Ollakseen mahdollisimman tehokas, jokaisen kehittäjän on itse suunniteltava työnsä ja suunnitelmien pitää perustua henkilökohtaiseen toimintatehokkuuteen.
- Kehittääkseen jatkuvasti omaa tehokkuutta, kehittäjien on käytettävä hyvin määriteltyjä ja mitattuja prosesseja.
- Tuottaakseen laadukkaita tuotteita, kehittäjien pitää tuntea henkilökohtaista vastuuta laadusta.
- On halvempaa löytää ja korjata virheitä aikaisemmassa ohjelmistotuotantoprosessin vaiheessa verrattuna myöhempään.

- On tehokkaampaa ehkäistä virheiden syntyä kuin etsiä ja korjata niitä.
- Oikea tapa suorittaa tehtävä on nopein ja halvin tapa.



Kuva 7. PSP-prosessin vaiheet [Humphrey , 2000a].

PSP koostuu kuudesta prosessista, joiden sisällöt ovat lyhyesti kuvattuina kuvassa 7. Tämä prosessin on tarkoitus kouluttaa ohjelmistokehittäjä kehittämään omaa toimintaansa. Ohjelmistokehittäjä, joka haluaa soveltaa PSP:tä aloittaa 0–tasolta ja siirtyy prosesseissa ylöspäin täyttäessään edellisen tason vaatimukset. 0-taso esittelee tapoja, joilla henkilökohtaiseen ohjelmistoprosessiin saadaan muodollisuutta ja kurinalaisuutta. Tämän lisäksi 0-taso vaatii henkilökohtaisen prosessin mittaamista aika-, virhe- ja kokomääreiden kannalta. Tämän jälkeen 0-tason oppeja laajennetaan lisäämällä 0.1-tason tuomat käsitteet, kuten koodausstandardin määrittely ja standardin mukainen ohjelmointi. Tämän lisäksi ohjelmistokokoa aletaan alustavasti mitata. Myös käytettyä ohjelmistotuotantoprosessia pyritään arvioimaan ja kehittämään.

Taso 1.0 esittelee arvioinnin, siten että kehittäjän on arvioitava uusien ohjelmien koot. Arvioiden tulee perustua edellisten projektien henkilökohtaisiin arvioihin.

Kaikkien projektien toteutuneet koot ja kestot tallennetaan. Tasoa 1.0 jatketaan 1.1-tasolla, jossa arviointeja aletaan hyödyntää tehtäväsuunnittelussa ja aikataulujen suunnittelussa.

Taso 2 esittelee katselmoinnit [Pressman, 2005, s. 754] ohjelmistokoodin ja -suunnittelun kannalta. Tämän lisäksi 2-taso esittelee virheiden ehkäisyn, jonka takia kehittäjät arvioivat ja mittaavat luotujen ja poistettujen virheiden lukumäärää tuotantoprosessin kaikissa vaiheissa. Kun taso 2 on hallussa, ohjelmistokehittäjällä on PSP:n kannalta tarvittava ymmärrys kokonaissuunnitteluun. Tasolla 2.1 luodaan tarpeellisia suunnittelupohjia, jotka ottavat kantaa PSP:n osoittamiin näkökulmiin ohjelmistoprojektin suunnittelussa ja dokumentoinnissa.

Ryhmäohjelmistoprosessi TSP kuvaa, miten ja millainen projektiryhmä tulisi koota ja miten sitä ohjataan. TSP ja PSP kulkevat käsikkäin, sillä TSP vaatii kaikilta jäseniltään kykyä PSP-prosessin soveltamiseen. Humphreyn [2000b] kuvauksen mukaan ryhmän muodostus perustuu yleisiin periaatteisiin, joita käytetään hyvän tehokkaan ryhmän muodostamisessa. Tehokkaan ryhmän muodostaminen vaatii ryhmän jäseniltä todellista ymmärrystä työstä, johon he sitoutuvat. Heidän tulee olla samaa mieltä tavasta, jolla työ tehdään ja uskoa, että heidän suunnitelma on mahdollista toteuttaa onnistuneesti. TSP-ryhmä muodostetaan seuraavien ehtojen mukaan [Humphrey , 2000b]:

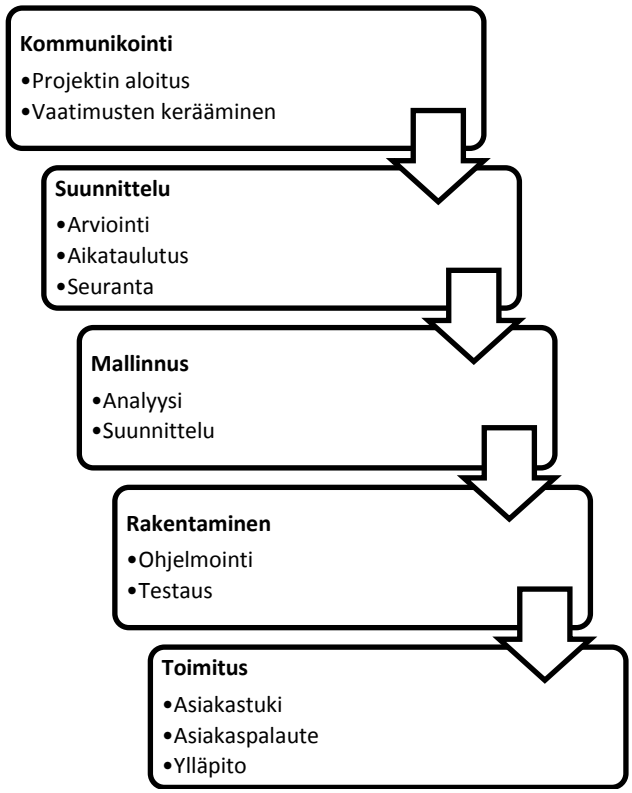
- Ryhmän jäsenet muodostavat yhteisen tavoitteen ja määrittävät omat roolinsa.
- Ryhmä muodostaa yhteisesti hyväksytyn strategian ja määrittää yleisen prosessin, joka ohjaa heidän työtä.
- Kaikki jäsenet ottavat osaa suunnitelman tuottamiseen ja jokainen tuntee oman henkilökohtaisen roolinsa suunnittelussa.
- Ryhmä esittelee suunnitelman johdolle ja johto katselmoi ja hyväksyy, mahdollisesti neuvotellun, suunnitelman.
- Ryhmä tekee työnsä suunnitelman mukaan.
- Ryhmä kommunikoi vapaasti ja usein, josta pitäisi seurata koheesiotila ja tehokkaampi yhteistyö.
- Ryhmän ohjelmistokehittäjät tuntevat tilanteensa ja saavat palautetta työstään.
- Ryhmää johdetaan tukemalla heidän motivaatiota.

TSP-ryhmä muodostetaan ja laukaistaan toimintaan yhteisen suunnitelman perusteella. Tähän suunnitelmaan on määritetty tehtävät ja niiden arvioidut koot ja aikataulut. TSP on iteratiivinen menetelmä, sillä yhden aloituksen sijaan, prosessi laukaistaan alusta väliajoin. Uudelleenlaukaisut ovat tarpeen, jotta projektin eri vaiheet voidaan suunnitella edellisten vaiheiden datan perusteella. Lisäksi uudelleenlaukaisun

yhteydessä ohjelmistokehittäjät päivittävät henkilökohtaiset PSP-suunnitelmat ja arviot, jotka ovat tarkkoja vain muutaman kuukauden ajan.

3.5. Vesiputousmalli

Vesiputousmallia pidetään perinteisenä ohjelmistokehitysmallina ja Pressmanin [2005, s. 79] mukaan se on ohjelmistokehityksen vanhin paradigma. Vesiputousmalli on lineaarinen ohjelmistokehitysmalli. Mallin mukaan ohjelmistokehitys etenee lineaarisesti projektin aloittamisesta suunnitteluun, mallinnukseen, kehitykseen ja toimitukseen. Kehitysmallin eteneminen ja vaiheiden sisältö on kuvattuna kuvassa 8.



Kuva 8. Vesiputousmalli [Pressman, 2005, s. 79].

Vesiputousmalli on ollut yleisesti käytössä ohjelmistokehityksen alkuajoista asti. Ajan kuluessa asiakas- ja järjestelmävaatimukset ovat muuttuneet monimutkaisemmiksi ja kompleksisuus on lisääntynyt. Vesiputousmalli on kohdannut kritiikkiä nykyaikaisen ohjelmistokehityksen vaatimusten edessä. Pressman [2005, s. 79–80] mainitsee kolme ongelmaa, jotka voi kohdata vesiputousmallin käytössä:

1. Harvat projektit voivat seurata lineaarista mallia. Vaikka malli tukee iterointia, se ei onnistu luonnollisesti.
2. Asiakkaan on usein mahdotonta lausua kaikkia vaatimuksia eksplisiittisesti projektin alussa. Vesiputousmalli vaatii tätä ja mallilla on vaikea hallita luonnollista epävarmuutta, joka on olemassa projektien alkuvaiheissa.

3. Ohjelmistotuotteen ensimmäinen toimiva versio on saatavilla vasta projektin loppuvaiheessa. Projektin loppuvaiheeseen päässeet huomaamattomat virheet tuottavat vakavia ongelmia projektin onnistumiselle.

3.6. Scrum

Scrum on hallinto-, kehitys- ja ylläpitometodologia olemassa olevalle järjestelmälle tai tuoteprototyypille [Schwaber, 1995]. Scrum kuuluu ketteriin menetelmiin, jotka pyrkivät tuomaan sopeutumiskykyä ja joustavuutta ohjelmistotuotantoprosessiin.

Scrum-projektissa tuotettava ohjelmisto pilkotaan havaituiksi ominaisuuksiksi, joista muodostuu projektin ominaisuuslista, product backlog. Product backlog rakennetaan yleensä käyttäjäkertomusten pohjalta (user story). Käyttäjäkertomuksesta tulisi ilmetä, millainen käyttäjä vaatii toimintoa, jotta käyttäjä saisi tarpeensa täytettyä. Cohn [2009] mainitsee esimerkiksi muotoilun ”Käyttäjäroolissa x haluan y, jotta z”, missä x on käyttäjän rooli, y on käyttäjän tavoite ja z on syy tavoitteelle. Esimerkiksi ”käyttäjäroolissa henkilöstöhallitsija, haluan hakea käyttäjiä sosiaaliturvatunnuksen perusteella, jotta löydän nopeasti yksilöidyn käyttäjän”. Uudet tunnistetut tarpeet voidaan lisätä product backlogiin missä tahansa vaiheessa projektia. Tuotteen omistajan (product owner, product manager) tehtävänä on määrittää jokaiselle product backlogilla olevalle asialle prioriteetti ja järjestää lista prioriteetin mukaan. Tuotteen omistaja voi muokata prioriteetteja tarpeidensa mukaan missä vaiheessa tahansa projektia.

Ohjelmistokehittäjät muodostavat Scrum-ryhmän, jossa tulisi olla kolmesta kuuteen henkilöä [Schwaber, 1995]. Ryhmään kuuluu Scrum Master, jonka tehtävänä on poistaa esteet ryhmän jäsenten työstä, sekä suojata heitä projektin ulkopuolisilta häiriöiltä. Scrum Masterilla ei ole suoraa käskyvaltaa ryhmän jäseniin.

Scrum-projekti etenee iteraatioina, joita kutsutaan sprinteiksi. Sprintin tulisi olla pituudeltaan sopivan lyhyt, jotta ketteryys säilyy ja suunnittelu sekä arvioinnit ovat tehtävissä ilman arvailuja. Sprintin sisältö suunnitellaan sprint backlogiin. Tuotteen omistaja pyytää Scrum-ryhmää toteuttamaan hänen priorisoinnin kannalta tärkeimpiä ominaisuuksia. Scrum-ryhmä pilkkoo pyydetyt ominaisuudet n. yhden työpäivän pituisiksi toteutuskokonaisuuksiksi, jotta todellinen ominaisuuden työarvio voidaan arvioida [Cohn, 2009]. Tämä arvio annetaan pisteinä siten, että pisteet kertovat kuinka työläs arvioinnin kohde on toisiin tehtäviin verrattuna. Tämän tiedon pohjalta scrum-ryhmä voi luvata tietyn määrän toteutusta seuraavalle sprintille. Sprint backlog sisältää siis yhden sprintin aikana toteutettavat tehtävät, jotka on saatu pilkkomalla product backlogin ominaisuuksia pienemmiksi osiksi. Scrum-ryhmä lupautuu toteuttamaan sovitun sisällön valmiiksi sprintin aikana.

Sprintin jälkeen Scrum-ryhmä esittelee tuotteen omistajalle toteutetun sisällön, mistä asiakas pääsee antamaan palautetta heti. Edellistä sprintiä tarkastellaan

kehittämisen näkökulmasta tunnistamalla mikä sprintin aikana meni hyvin ja mikä meni huonosti. Projektin etenemistä seurataan jokaisesta sprintistä kerätyn velocity-arvon perusteella, joka kertoo, kuinka monta pistettä projektiryhmä pystyy toteuttamaan sprintissä. Tämän jälkeen seuraava sprint suunnitellaan ja uusi iteraatio alkaa alusta.

Scrumiin siirtyminen ei ole pelkästään toimintatapojen muutosta vaan se vaatii myös filosofista muutosta ryhmän jäseniltä ja organisaatiolta. Lisätietoa onnistuneesta tavasta toteuttaa Scrumia saa Cohnin [2009] havainnoista.

3.7. Automaattinen yksikkötestaus

Yksikkötestaus on yksi testausstrategia, joka perustuu komponenttien ja moduulien testaukseen, siten että testin laajuus rajoittuu komponentin sisäisen prosessin ja tietorakenteiden testaamiseen [Pressman, 2005, s. 394–395]. Yksikkötesti ohjelmoidaan siten, että se on riippumaton kaikesta testikohteen ulkopuolisesta toteutuksesta, sekä sen ajamista ei rajoita mikään, eli sen voi ajaa nopeasti, milloin vain. Tämä mahdollistaa yksikkötestien nopean ja riippumattoman toistettavuuden.

Automaattinen yksikkötestaus on käytettävissä kaikissa kehitysmalleissa, mutta se on erityisen tärkeä työkalu ketterissä menetelmissä. Ketterissä menetelmissä yleensä edetään iteratiivisesti ja järjestelmään rakennetaan toiminto tai osakokonaisuus kerrallaan. Tällöin tulee tilanteita, että aikaisempaa toteutusta on muutettava, jotta uuden toiminnon voi toteuttaa. Yksikkötestit antavat kehittäjälle varmuuden tehdä tuon muutoksen, sillä kehittäjä näkee yksikkötestien tuloksista, onko hän rikkonut järjestelmän muutoksellaan. Samalla kun järjestelmää muutetaan, pitää myös muutoksen testaavat yksikkötestit päivittää vastaamaan uutta oikeellista toteutusta. Tällainen kehitystapa vaatii korkeata testikattavuutta, sillä virheitä ei huomata koodista, jolle ei ole kirjoitettu yksikkötestiä.

Testaukseen ja erityisesti yksikkötestaukseen liittyy käsite testikattavuus. Testikattavuudella tarkoitetaan prosenttiosuutta ohjelmakoodiriveistä, jotka ajetaan yksikkötesteissä. Sataprosenttinen testikattavuus tarkoittaa, että jokainen ohjelmakoodirivi käydään läpi yksikkötestien yhteydessä. Testikattavuus osoittaa yleensä järjestelmän arkkitehtuuritasoja tai komponentteja, joiden testikattavuus ei ole kehittynyt yksikkötestejä kirjoitettaessa.

Varsinkin ketterissä menetelmissä käytetään ns. jatkuvan integroinnin palvelua (Continous Integration) [Duvall et al., 2007]. Yleensä tämä palvelu liitetään versionhallintaan siten, että kun versionhallintaan siirretään uusi versio jostain rakennettavan järjestelmän osasta, jatkuvan integroinnin palvelu ajaa kaikki yksikkötestit versionhallinnan koodipohjaa vasten ja ilmoittaa, jos versionhallinnassa on rikkinäinen versio rakennettavasta järjestelmästä. Tämä mahdollistaa nopean reagoinnin virheeseen ja estää virheen leviämisen. Lisäksi jatkuvan integroinnin palveluihin voi liittää esim. automaattisen testikattavuuden määrittämisen ja muita

automaattisesti laskettavia koodiin liittyviä mittareita, esim. koodirivien määrän laskemisen. Esimerkkinä jatkuvan integraation palvelusta mainittakoon Hudson [Hudson, 2010], joka on avoimeen lähdekoodiin perustuva jatkuvan integraation palvelin, johon voi liittää erilaisia mittauskomponentteja.

4. Six Sigma

Six Sigma on eri asiayhteyksissä yhdistetty mm. laadunhallintaan, projektinhallintaan ja liiketoiminnan tehostamiseen, joista yhtäkään se ei yksittäisesti edusta. Six Sigma on myös saanut tilastollisen menetelmän leiman, johtuen metodologiassa hyödynnettävistä tilastollisista työkaluista. Kyseessä on kokonaisuus, joka tulee helposti luokiteltua käytön tarkoitusperän tai tavoitteiden perusteella. Six Sigmasta on olemassa lukuisia kuvauksia, kirjallisuutta, itseopiskelumateriaalia, mutta ei kattavaa empiiriseen tutkimuksen tuottamaa pohjaa. Schroeder ja muut [2008] tutkivat Six Sigman piirteitä ja ominaisuuksia muodostaakseen perusmääritelmän, joka sisältäisi Six Sigman teoreettiset näkökulmat. Heidän määritelmä suomennettuna on seuraava:

> Six Sigma on järjestelmällinen rinnakkaisrakenne, jolla on tarkoitus vähentää organisatoristen prosessien variaatiota, käyttämällä kehittämiseen erikoistunutta henkilöstöä, rakenteista metodia ja suorituskykyjen mittausta strategisten tavoitteiden saavuttamiseen.

Kyseinen määritelmä kattaa keskeiset piirteet metodologiasta. Six Sigma – metodologialla on omat roolinsa ja henkilöstöhierarkia, joista seuraa rinnakkaisuus organisaatiorakenteen kanssa. Roolit vaativat koulutusta, joka erikoistaa henkilöstön metodologian vaatimiin tehtäviin. Six Sigma on vaiheistettu, rakenteen omaava metodologia, joka hyödyntää mittaustuloksia kehityksen varmistamisessa ja päätösten teossa. Zu ja muut [2008] käyttävät Six Sigmasta seuraavaa määritelmää:

> Six Sigma on organisoitu, systemaattinen metodi strategiselle prosessin kehittämiselle, uusien tuotteiden, sekä palveluiden kehittämiselle, joka luottaa tilastotieteen metodeihin ja tieteelliseen metodiin tehdä dramaattisia vähennyksiä asiakkaan määrittämien kohteiden virhemäärissä.

Zun ja muiden [2008] käyttämä määritelmä painottaa eri asioita kuin Schroederin ja muiden [2008] määritelmä. Edellä mainittu määritelmä laajentaa Six Sigman käyttötarkoituksia myös uusien tuotteiden ja palveluiden kehittämiseen. Määritelmästä tulee myös ilmi asiakaslähtöisyys, joka sitoo metodologian tulokset asiakkaan asettamiin kohteisiin.

Six Sigma kehitettiin Motorolalla 1980-luvun puolivälissä. Tuolloin käytettiin yleisesti Total Quality Management –hallintostrategiaa. TQM otti huomioon laadun prosesseissa ja toi laadun keskeisesti mukaan päätöksentekoon. Six Sigma on omaksunut TQM:n piirteitä ja onkin selkeästi TQM:n seuraaja. Six Sigmaa on kuitenkin kritisoitu siitä, että se ei tuo mitään uutta laadunhallintaan, vaan esittelee jo

olemassa olevat tekniikat uudessa paketissa. Mitä uutta Six Sigma toi sitten mukanaan? Schroeder ja muut [2008] esittelevät neljä keskeistä Six Sigman tuomaa lisäystä suhteessa TQM-hallintostrategiaan:

Six Sigmassa on lähes ainutlaatuinen tapa kohdistaa taloudelliset ja liiketoiminnalliset tavoitteet. Yleensä jokaiselta Six Sigma –projektilta ja täysiaikaisesti Six Sigma –roolissa toimivalta henkilöltä vaaditaan taloudellisia tuloksia, esimerkiksi säästöjen tai tuottavuuden muodoissa. Tämä tarkoittaa, että organisaation taloudelliset tavoitteet ovat projektitasolla. Tuloksia seurataan projektia edeltävien ja projektien jälkeisten auditointien avulla.

Six Sigma vaatii jäsennetyn metodin tarkkaa seuraamista, täysiaikaisten Six Sigma –rooleissa toimivien henkilöiden intensiivistä koulutusta ja tilastollisten ja muiden metodologiaan kuuluvien työkalujen integrointia organisaation toimintaan. Erityisen kehitysprosessin käyttäminen kehittämiseen erikoistetulla henkilöstöllä vaatii organisaatiolta tukea ja sitoutumista metodologian käyttämiseen.

Six Sigmassa on erityisesti metodologialle määritettyjä mittareita. Nämä erityiset mittarit, kuten sigma-taso ja DPMO (Defects Per Million Opportunities), konkretisoivat parannustulokset ja prosessin kunnon metodologian tuomien tulosten kautta. Mittarit korostavat prosessin parantamista ja rohkaisevat vaativiin, mutta realistisiin tavoitteisiin. Lisäksi, Six Sigman mittarit varmistavat, että asiakasvaatimukset huomioidaan, kun kehitysprosessia ruvetaan käyttämään. Six Sigman keskeiset mittarit esitellään luvussa 4.2.

Useiden täysiaikaisten kehitykseen erikoistuneiden toimihenkilöiden käyttäminen on uutta monille organisaatioille. Organisaatiot ovat harvemmin valmiita panostamaan täysiaikaiseen kehityshenkilöstöön. Kehityshenkilöstön lisäksi organisaation henkilöille tulee kouluttaa jäsennetyn kehitysmetodin toiminta.

Six Sigman vaikutuksista laadunhallintaan on tehty vähän empiiristä tutkimusta. Zu ja muut [2008] tutkivat empiirisesti Six Sigman vaikutusta laadunhallintaan 226 yhdysvaltalaisen tuotanto-organisaation kokemuksista Six Sigman ja perinteisen laadunhallinnan yhdistämisestä. Tutkimuksessa tunnistettiin perinteisen laadunhallinnan käytännöiksi ylemmän johdon tuki, asiakassuhteet, tavarantoimittajasuhteet, työvoiman johto, laatuinformaatio, tuote- ja palvelusuunnittelu, sekä prosessinhallinta. Six Sigman laadunhallinnasta tunnistettuja käytäntöjä olivat roolirakenne, jäsentynyt kehitysprosessi ja mittaamisen keskeisyys. Tutkimuksessa ilmeni myönteisiä suhteita Six Sigman käytäntöjen ja perinteisen laadunhallinnan käytäntöjen välillä. Havaintojen perusteella tutkijat tekivät olettamuksia Six Sigman myönteisistä suhteista perinteisen laadunhallinnan yhteydessä. Positiiviset suhteet on esitetty taulukossa 2.

	Six Sigman roolirakenne	Six Sigman jäsentynyt kehitysprosessi	Six Sigman keskeinen mittaaminen
Ylemmän johdon tuki	+		+
Työvoiman johto	+		
Laatuinformaatio			+
Tuote- ja palvelusuunnittelu		+	+
Prosessinhallinta		+	+
Six Sigman roolirakenne		+	
Six Sigman keskeinen mittaaminen		+	

Taulukko 2. Six Sigman ja perinteisen laadunhallinnan käytäntöjen positiiviset suhteet [Zu et al., 2008].

4.1. Six Sigma toimintamallit

Six Sigma -metodologia perustuu järjestelmälliseen, iteroitavaan toimintaan. Toimintamallin seuraaminen on ehdotonta projektiryhmälle, sillä ryhmän päätöksenteko pohjautuu toimintamallin aikana kerättävään dataan. Six Sigman toimintamalleja on kehitetty useita ja jokainen organisaatio voi luoda oman mallin tai käyttää organisaatiolle sopivaa mallia. Six Sigma –kirjallisuudessa ja tutkimuksissa korostuu kaksi keskeistä toimintamallia: Define, Measure, Analyze, Improve, Control (DMAIC) ja Design For Six Sigma (DFSS). Suurin osa muista malleista on muunneltuja DMAIC-malleja.

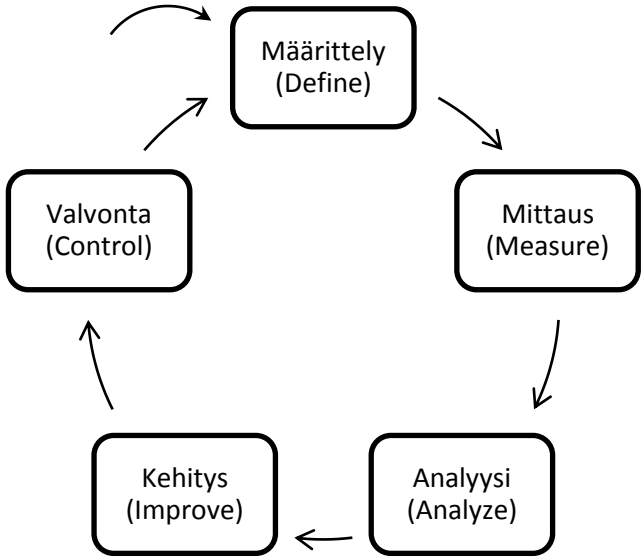
4.1.1. Define, Measure, Analyze, Improve, Control (DMAIC)

Six Sigman toimintamalleista yleisin on DMAIC ja sitä kutsutaankin perinteiseksi tavaksi hyödyntää Six Sigmaa. Kirjallisuudessa Six Sigma kuvataan järjestelmällisenä prosessina DMAIC-mallin kautta. DMAIC-malli on suunniteltu Six Sigma –projektiin, jossa tuote on suunniteltu ja tuotteen tuotantoprosessi on jo olemassa. Tällöin lähtökohtana on jo olemassa olevan tuotantotavan tavoitehakuinen kehittäminen.

DMAIC koostuu viidestä eri vaiheesta: määrittely-, mittaus-, analyysi-, kehitys- ja valvontavaiheesta. Toimintamalli käy vaiheet läpi sykleinä siten, että edellisen iteraation tuottamasta informaatiosta voidaan määritellä seuraavan iteraation tavoitteet. Iteraatio on kuvattu kuvassa 9. Uusi iteraatio tähtää tuotantomuutoksilla edellistä iteraatiota kehittyneempään toimintaan. Iteratiivinen toiminta ei tosin ole vaadittua, vaan prosessi voi päättyä saavutetun kehityksen ylläpitoon ja valvontaan.

Ennen Six Sigma –projektin aloittamista, olemassa oleva prosessi tutkitaan ja sen kehitysmahdollisuudet arvioidaan. Jos arvioiden mukaiset säästöt tai muut tavoitteet

ovat saavutettavissa Six Sigman avulla, voidaan aloittaa uusi Six Sigma –projekti DMAIC-toimintamallilla. Seuraavissa kappaleissa käydään toimintamalli vaiheittain läpi Kellerin [2005] ja Tayntorin [2007] kuvausten pohjalta.



Kuva 9. DMAIC-prosessi.

Määrittely (Define)

Määrittelyvaihe on DMAIC-toimintamallin aloittava vaihe. Vaiheeseen tullessa organisaatio on päättänyt ottaa Six Sigman käyttöön kyseiseen projektiin, jolloin se voidaan leimata Six Sigma –projektiksi. Vaiheen tavoitteena on määrittää projekti, sekä ymmärtää käytössä oleva tuotantoprosessi ja ratkaista siinä havaitut ongelmat.

Vaihe käynnistyy projektin määrittelyllä. Kaikki vaiheen aikana esiin tulevat projektiin liittyvät asiat kirjataan projektin perustuskirjaan. Perustuskirjaan kirjataan ainakin projektin tarkoitus, ongelman määrittely, liiketaloudelliset ja muut tavoitteet, projektin laajuus, projektin jäsenien roolit, vastuualueet, muut resurssit ja virstanpylväät. Perustuskirjaan kirjatut tavoitteet määrittävät perustan, jota vasten projektin onnistuminen arvioidaan.

Käytössä oleva tuotantoprosessi kartoitetaan korkealla tasolla. Tässä vaiheessa on tärkeää tunnistaa prosessin ja sen aliprosessien rajat, syötteet, tulokset ja asianomaiset. Prosessikartoituksen tarkoituksena on auttaa projektin määrittelyä, sekä projektiryhmän muodostamista.

Projektiryhmä tulisi muodostaa siten, että jokainen organisaatioyksikkö, joka on osana tuotantoprosessia, olisi edustettuna ryhmässä. Projektiryhmän muodostaminen on tärkeä asia, sillä ryhmän toimivuus ja osaaminen ovat keskeisimmät menestystekijät, jotka ratkaisevat Six Sigma –projektin onnistumisen. Anand ja muut [2010] tutkivat tiedon luonnin kannalta Six Sigmaa ja totesivat, että projektiryhmän hiljaisen tiedon

muuttaminen eksplisiittiseksi tiedoksi vaikuttaa positiivisesti Six Sigma -projektin onnistumiseen. Tämä on järkeenkäypää, kun ottaa huomioon, että jokaisessa organisaatioyksikössä on oma tietosisältö.

Mittaus (Measure)

Mittausvaiheeseen tullessa projekti on määritelty ja tuotantoprosessi tunnetaan korkealla tasolla. Mittausvaiheessa tuotantoprosessi mallinnetaan mahdollisimman tarkasti. Jokainen prosessin osa vaikuttaa kokonaisuuteen, jolloin jokainen osaprosessi saattaa olla vaikuttava tekijä kokonaisprosessin laatuvirroissa. Mitä tarkempi tuotantoprosessin kartoitus on, sitä enemmän Six Sigma -metodologiasta voi olla hyötyä.

Kun tuotantoprosessi tunnetaan tarkasti, voi sen tehokkuutta ja kuntoa mitata prosessimittareilla. Mittaukseen on olemassa erilaisia lähestymistapoja. Prosessiin voidaan asettaa mittareita, jotka kertovat toiminnan tehokkuudesta ja teoreettisesta potentiaalista. Prosessiin voidaan asettaa Six Sigmalle keskeisiä hajontamittareita, joilla etsitään keskihajontaa kasvattavia tekijöitä. Mittaukseen voi myös liittää muita hyväksi huomattuja mittareita, esimerkiksi yhdistämällä Lean-mallin tapaisen lähestymistavan mittareita, joiden avulla yritetään tunnistaa resursseja hukkaavia tekijöitä. Yleisimmät mittauksen kohteet liittyvät tavalla tai toisella kustannuksiin, laatuun ja aikatauluun [Keller, 2005, s. 85].

Kun mittarit on tunnistettu ja dokumentoitu, niille suoritetaan mittaus. Mittauksessa kerätään dataa toiminnassa olevasta tuotantoprosessista. Mittaus koostuu otoksesta, jossa tuotantoprosessia mitataan toistuvasti, jotta dataa on riittävästi alustavien analyysien tekemiseen. Mittauksen jälkeen olemassa olevalle tuotantoprosessille lasketaan Six Sigman keskeisimmät arviointimääreet, eli prosessin suorituskyky ja Sigma–taso. Edellä mainitut mittarit selitetään tarkemmin kohdassa 4.2.

Analyysi (Analyze)

Analyysivaiheessa projektiryhmällä on käytettyjen mittareiden mittaustulokset toiminnassa olevasta tuotantoprosessista. Heillä on myös tuotantoprosessille määritellyt tavoitteet, joiden perusteella mittarit on valittu. Projektiryhmän tehtävänä on tunnistaa mittaustulosten perusteella tuotantoprosessin vaiheet, jotka eivät toimi määritysvaiheessa määritetyllä tavoitetasolla. Analyysivaiheessa tehdyissä huomioissa on otettava huomioon, että tuotantoprosessin vaiheet voivat kärsiä ajoittaisista tai pysyvistä hajontaa aiheuttavista tekijöistä. Kun tuotantoprosessin heikot vaiheet on tunnistettu, on projektiryhmän löydettävä juurisyyt, jotka aiheuttavat vaiheiden heikkoudet. Juurisyiden tunnistamiseen on olemassa erilaisia menetelmiä. Juurisyytä selvitettäessä voidaan käyttää esimerkiksi syy-seuraus –analyysia tai kyselytekniikkaa,

jossa kysymyksen vastaus asetetaan kyseenalaiseksi toistuvasti, kunnes pohjimmainen syy tunnistetaan.

Projektiryhmän tulee keksiä ratkaisuehdotuksia juurisyiden ratkaisemiseen. Ratkaisuehdotusten tulee poistaa tai vähentää juurisyyn aiheuttamaa hajontaa prosessissa. Kehitettyjä ratkaisuehdotuksia verrataan toisiinsa ja arvioidaan, mitkä ehdotukset kehittävät tuotantoprosessia siten, että se tuottaa asiakasvaatimusten mukaista tuotetta mahdollisimman tehokkaasti.

Kehitys (Improve)

Kehitysvaiheessa projektiryhmällä on analyysivaiheessa tunnistetut ratkaisuehdotukset. Ratkaisuehdotuksista on tarkoitus muodostaa konkreettisia prosessimuutoksia, jotka tähtäävät optimoituun tuotantotoimintaan.

Oikeiden muutosten tunnistaminen ja tekeminen ovat vaativia tehtäviä. Jos muutos ei korjaa prosessin kuntoa paremmaksi, hyöty jää saavuttamatta. Muutostoimenpiteet voivat myös heikentää prosessia entisestään. Korjaavien muutosten tunnistamiseen on kuitenkin erilaisia analyysityökaluja. Kokeilujen sijasta prosessia kannattaa tutkia, voisiko sitä parametrisoida ja täten simuloida eri arvoilla. Parametrisoinnissa prosessista tunnistetaan avainmuuttujia, joita muuttamalla prosessin kunto muuttuu. Simuloinnilla voi järjestelmällisesti löytää optimaaliset säädöt prosessiin.

Yksi yritysmaailmassa yleisesti tunnettu tapa parantaa omaa prosessia on verrata omaa tuotantoprosessia alan johtavien organisaatioiden tuotantoprosesseihin ja hakea sieltä kehitysajatuksia. Tällaista toimintaa kutsutaan termillä benchmarking [Stewart, 2010]. Benchmarking ei vaadi Six Sigmalle tyypillisiä analyyttisia työkaluja, mutta Six Sigma tuo mukanaan työkalut, joilla voidaan mitata ja varmistaa muutos.

Valvonta (Control)

Valvontavaiheessa muutoksista on muodostettu uusi standardoitu tuotantoprosessi, joka siirretään valvotusti käytäntöön. Tarkoituksena on saada muutokset pysyvästi tuotantoprosessiin aiheuttamalla mahdollisimman vähän haittoja. Muutos vaatii valvontaa, sillä muutos saattaa aiheuttaa itsessään ongelmia, joita projektiryhmä ei ennalta osannut huomioida. Tämän takia vaihetta varten tulisi laatia valvontasuunnitelma, jossa määritetään prosessin eri osille valvontatavat. Muutos yleensä vaatii prosessissa työskenteleviltä työntekijöiltä paljon, sillä he saattavat joutua muuttamaan pitkäaikaisia työrutiinejaan. Muutos voi viedä aikaa, joten valvontavaiheeseen kannattaa varata resursseja ja se tulisi suunnitella huolellisesti.

Valvontavaiheessa prosessin kuntoa mitataan lyhyemmillä aikaväleillä. Lyhyen aikavälin tulokset kertovat tuotantoprosessin muutoksesta nopeasti, jolloin sitä voidaan

vielä ohjata oikeaan suuntaan. Kun tuotantoprosessin toiminta alkaa vakiintua, voidaan mittausten aikaväliä suurentaa.

Dokumentointi kuuluu valvontavaiheeseen oleellisena osana. Vaiheen aikana tulisi dokumentoida asioita, joista voidaan oppia jatkossa, sekä kehittää tulevien valvontavaiheiden toteuttamista. Valvontavaiheen dokumentointiin tulisi kuulua ainakin seuraavat asiat [Keller, 2005, s. 164]:

- Projektin perustuskirja kertoo lukijalle projektin ongelman, tavoitteet ja suunnitelman.
- Jokaisesta DMAIC-vaiheesta tulisi olla yhteenveto tavoitteista ja tuloksista.
- Liitteeksi tulisi liittää kerätty raakadata, sekä datasta tehdyt analyysit aikajärjestyksessä.
- Projektin toteuttamiseen tarvitut menoerät tulisi listata. Tähän listaukseen kuuluvat esim. väliaikaisesti menetetty suorituskyky, materiaali- ja työvoimakulut.
- Projektin tuloksena syntyneet säästöt tulisi kirjata kuluneeseen päivään asti, sekä arvioida vaikutukset tuleville lähivuosille.
- Projektin nykyinen tilanne ja valvontasuunnitelma.
- Suositukset tuleville projekteille, jotka liittyvät kyseiseen prosessiin.
- Suositukset tulevien projektien johtajille ja tukijoille, jotka pohjautuvat kyseisen projektin aikana opittuihin asioihin.

4.1.2. Design For Six Sigma (DFSS)

Design For Six Sigma –toimintamallia käytetään uuden tuotteen ja sen tuotantoprosessin suunnitteluun. DFSS-mallissa pyritään vaikuttamaan laatuun vaikuttaviin asioihin ennen tuotantoa, kun taas DMAIC-mallissa pyritään tehostamaan olemassa olevaa tuotantoprosessia. Tayntorin [2007] mukaan useat organisaatiot pysähtyvät 4,5:n Sigma-tasoon, koska he eivät ole suunnitelleet tuotetta DFSS-toimintamallin mukaan. Suunnittelussa yhdistetään asiakasvaatimukset, sekä tuotantoprosessin suorituskyky. Lopputuloksen tulisi vastata asiakasvaatimuksia ja määritellä prosessi tai tuote, joka voidaan tuottaa Sigma-tasolla 6 [Tayntor, 2007].

DFSS-toimintamallista on kehitetty useita eri lailla vaiheistettuja kehyksiä [Tayntor, 2007]. DFSS-toimintamallin kehyksille on yhtenäistä se, että kehykset kääntävät asiakasvaatimukset vankaksi suunnitteluksi. Tämän mahdollistavat metodologian suunnittelupainotteiset analyyttiset työkalut ja niitä käyttämään koulutettu Six Sigma -ryhmä [Siviy et al., 2008]. Edellä mainittuja työkaluja ovat mm. Quality Function Deployment [Chan and Wu, 2002], mallinnus, kokeellinen suunnittelu, simulointi ja tilastollinen optimointi.

Yksi esimerkki DFSS-kehyksestä on IDDOV, jota mm. Zhao ja muut [2008] soveltavat mallissaan ohjelmistokehityksen vaatimustenhallintaan. IDDOV koostuu viidestä vaiheesta: tunnistus- (Identify), määrittely- (Define), suunnittelu- (Develop), optimointi- (Optimize) ja tarkistusvaihe (Verify). Kehys alkaa tunnistusvaiheella (Identify, Define), joka vastaa sisällöltään läheisesti DMAIC-toimintamallin määrittelyvaihetta. Tämä johtuu siitä, että kehyksen alussa on perustettava Six Sigma –projekti, sekä tunnistettava projektin ongelma ja mahdollisuudet korjata ongelma. Vaiheen aikana on tarkoitus tunnistaa asiakasvaatimukset, jotka liittyvät uuteen tuotteeseen. Tässä vaiheessa sovelletaan SIPOC-analyysia (Supplier, Input, Process, Output, Customer), jonka avulla tunnistetaan eri osapuolien syötteet, syötteiden prosessointi, prosessoinnista syntyvä tuloste ja tulosteen saava asiakas. Vaatimukset jaetaan ryhmiin niiden asiakkaalle tuottaman arvokkuuden mukaan. Kano-mallin avulla voidaan tunnistaa arvokkuutta tuottavat vaatimukset ja arvioida niiden toteutukseen tarvittavat resurssit. Tällaisen analyysin avulla voidaan esim. tunnistaa suurta arvokkuutta tuovat piirteet, joiden toteuttamiseen tarvitaan suhteellisen vähän resursseja. Tällaisia vaatimuksia kutsutaan ns. matalalla roikkuviksi hedelmiksi. Vaatimusten analysoinnin jälkeen Six Sigma –ryhmä tunnistaa vaatimukset, jotka ovat kriittisiä laadun kannalta. Tunnistamisen tarkoituksena on erotella pois vaatimukset, jotka eivät tuota asiakkaalle arvokkuutta. Vaatimuksia, jotka eivät tuota arvoa, ei tule toteuttaa.

Suunnitteluvaiheessa (Design) hahmotetaan ratkaisu, joka täyttää kriittiset asiakasvaatimukset. Samalla yritetään havaita tekijöitä, jotka aiheuttaisivat hajontaa ratkaisua sovellettaessa. Ratkaisu, eli uusi tuote, mallinnetaan tarkasti, jotta sitä voitaisiin simuloida ongelman ratkaisuna. Uuden tuotteen valmistusprosessiin asetetaan laatumittarit, joita seuraamalla voidaan varmistaa, että uutta tuotetta valmistetaan asiakasvaatimusten mukaiseksi.

Optimointivaiheessa (Optimize) tuotantoprosessin suorituskyky optimoidaan. Optimointi suoritetaan parametrisoimalla tuotantoprosessi ja tutkimalla eri parametrien vaikutusta tuotantoprosessiin. Tätä kautta tuotantoprosessille löydetään teoreettiset, optimaaliset parametrit, joilla tuotantoprosessin pitäisi toimia tehokkaimmillaan myös käytännössä.

Tarkistusvaiheessa (Verification) kehitetty tuote ja tuotteen valmistusprosessi todennetaan valmistamalla tuotteesta prototyyppi ja pilotoimalla tuotantoprosessi. Tuotteen prototyyppiä käyttämällä tutkitaan, täyttääkö tuote sille asetetut asiakasvaatimukset. Tuotantoprosessin pilotoinnilla varmistetaan, että tuotantoprosessilla voidaan luoda tuotetta asiakasvaatimusten mukaiseksi mahdollisimman tehokkaasti.

4.2. Six Sigma –metodologian keskeiset mittarit ja analyysi

Six Sigma –projektissa päätöksenteko pohjautuu tuotantoprosessista mitattuun dataan, mikä kuvaa prosessin kuntoa. Tuotantoprosessin mittaaminen suunnitellaan Six Sigma –toimintamallien alkuvaiheissa. Metodologian mittarit kertovat tuloksia erityisesti metodologian näkökulmasta.

4.2.1. Sigma-taso

Sigma (σ) on kreikkalainen kirjain, joka tilastotieteessä on keskihajonnan symboli. Sigma-taso kertoo, kuinka paljon mitatun kohteen tulos vaihtelee ja kuinka prosessin mitattava kohde pysyy sille määritetyllä kontrollialueella.

Keskihajonta ei itsessään esitä konkreettista näkökulmaa prosessin kunnosta. Sigma-taso on rinnastettu virheiden suhteelliseen määrään miljoonasta virheen mahdollisuudesta. Virheellä tarkoitetaan prosessin toimintaa, joka ei pysy prosessille määritettyjen rajojen sisällä. *DPMO*-mittari (*Defects Per Million Opportunities*) tuo näkökulman, josta saa keskihajontaa selkeämmän kuvan, kuinka usein prosessi tuottaa virheitä. Taulukossa 3 on kuvattu, miten Sigma-tasot vastaavat virheiden määriä miljoonassa virheen mahdollisuudessa.

Sigma-tason laskeminen perustuu yleiseen normaalijakaumaan. Normaalijakauman käyttö mahdollistaa otosten käyttämisen prosessin kokoaikaisen toiminnan arvioimisessa. Käytännössä Sigma-taso kertoo virheettömän toiminnan alueen normaalijakaumassa hajontojen lukumääränä keskiarvosta laskettuna. Virheellisen toiminnan todennäköinen osuus määritetään yleisen normaalijakauman avulla. Prosessimittarille on siis määritettävä arvoalue, jonka sisällä prosessin tai sen osan toiminta on hyväksyttävää. Alempi määrittelyraja, *LSL* (*Lower Specification Limit*), määrittää alimman hyväksytyn tuloksen ja ylempi raja, *USL* (*Upper Specification Limit*), määrittää ylimmän hyväksytyn tuloksen. Jos prosessimittari tuottaa tuloksen, joka ei ole alemman ja ylemmän määrittelyrajan välissä, se tulkitaan virheeksi. Yleensä LSL ja USL määritellään asiakasvaatimuksista.

USL = ylempi määrittelyraja
LSL = alempi määrittelyraja
μ = keskiarvo
σ = keskihajonta

$$z_l = \frac{LSL - \mu}{\sigma} \qquad z_u = \frac{USL - \mu}{\sigma}$$

Kaava 2. Sigma-tason laskukaava.

Sigma-taso lasketaan mittaustulosjoukosta, joka on tässä yhteydessä tilaston populaatio. Populaatiosta lasketaan keskiarvo μ, sekä keskihajonta σ. Määrittelyrajojen, populaation keskiarvon ja keskihajonnan avulla voidaan laskea Sigma-taso z ylemmälle ja alemmalle määrittelyrajalle kaavoilla, jotka ovat kuvattuina kaavassa 2.

Arvolla Z_u voidaan tutkia, kuinka suuri osa koko populaatiosta on todennäköisesti pienempi kuin USL tai z_l-arvolla suurempi kuin LSL. Aivan kuten tilastotieteessä, myös Six Sigman yhteydessä normaalijakauman todennäköisyyksiä tutkitaan valmiiksi lasketuista taulukoista. Yleisen normaalijakauman todennäköisyystaulukosta voidaan hakea todennäköisyysarvo z-arvon avulla, joka on arvoltaan välillä [0,1].

Sigma-taso ei kuitenkaan tule aivan suoraan tuloksesta z. Motorolan projekteista kerättyjen tietojen perusteella metodologian kehittäjät tekivät huomion, että Sigma-taso ei pysy mitatulla tasolla pitkällä aikavälillä [Keller, 2005, s. 4]. Normaalijakauman keskitettyyn +/-6σ sisään kuuluu 99,9999998 % populaatiosta, eli Sigma-tasolla 6 DPMO on 0,002 eikä usein kirjallisuudessa esitetty 3,4 (esim. [Keller, 2005, s. 4] ja [Pressman, 2005, s. 761]). Motorolan havaintojen mukaan pitkällä aikavälillä prosessin suorituskyky on 1,5σ heikompi. Motorola on selittänyt siirtymää esimerkiksi materiaalivariaatioilla, valmistusmenetelmillä ja luonnollisella kulumisella. Toisin sanoen, keskitettyä tulosta siirretään 1,5σ. Sigma-tason 4,5 sisään kuuluu 99,9996602 % populaatiosta, eli DPMO on 3,398 ≈ 3,4. Tavoiteltu Sigma-taso 6 on siis tilastotieteen näkökulmasta Sigma-taso 4,5.

Sigma-taso	DPMO 1,5 siirtymän jälkeen	Virheettömien osuus 1,5 siirtymän jälkeen	DPMO ilman 1,5 siirtymää	Virheettömien osuus ilman 1,5 siirtymää
0	933 193	6,68%	500 000	50,00%
1	691 462	30,85%	158 655	84,13%
2	308 537	69,15%	22 750	97,72%
3	66 807	93,32%	1349	99,87%
4	6 210	99,38%	31	99,9997%
5	233	99,9767%	0,29	99,99997%
6	3	99,9997%	0,001	99,99999990%

Taulukko 3. Sigma-tasojen ja DPMO-määrien vertailu.

Puolentoista sigman siirtymä ei ole pieni asia. Sigma-tasolla 3 keskitetyn populaation DPMO on 1349 ja 1,5σ siirtymän jälkeen DPMO on 66807. Six Sigma on saanut kritiikkiä kyseisestä siirtymästä, koska se yleistetään metodologiaan Motorolan projekteissa havaituista siirtymistä. Siirtymää ei voi tilastotieteen kautta yleistää, joten siirtymää ei voi perustella yleistettäväksi tieteellisesti. Siirtymän eroa on kuvattu 1,5

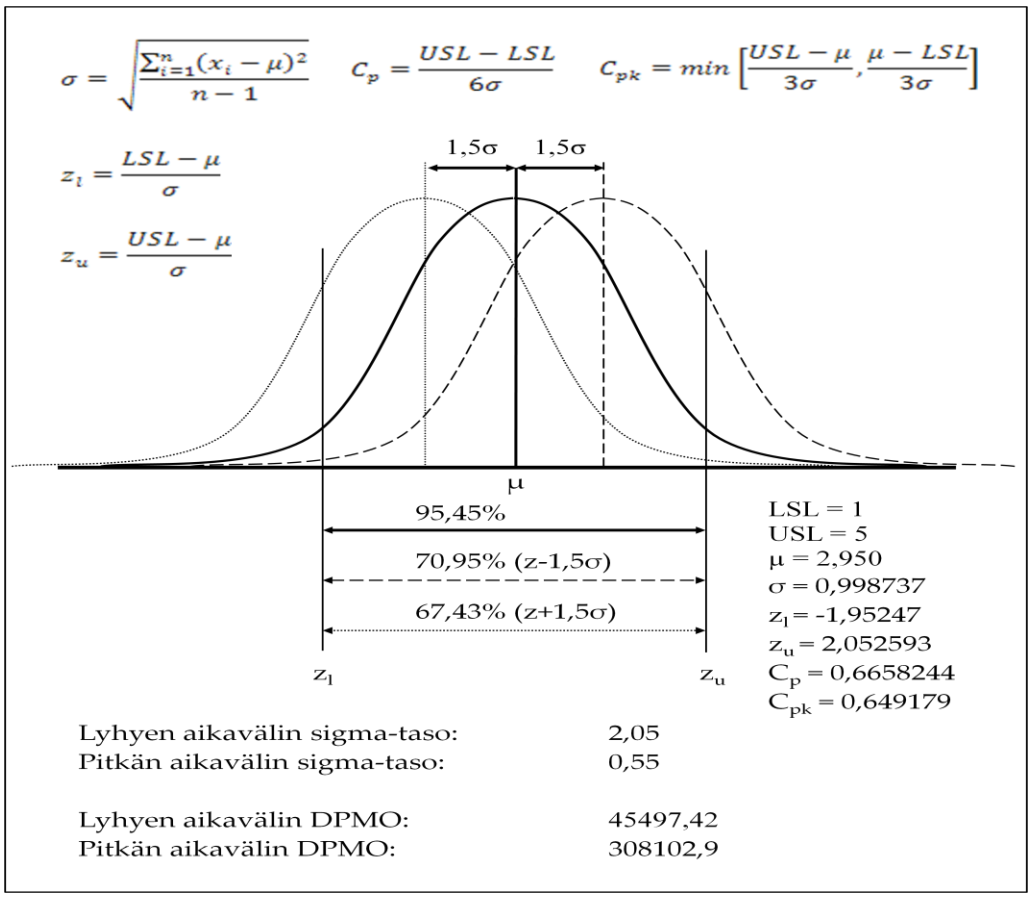
sigma-tasolla taulukossa 3. Taulukosta ilmenee, kuinka suuri merkitys siirtymällä on esim. virheettömien osuuteen.

Esimerkki 1

Prosessille on määritetty mittari, jonka ylempi määrittelyraja z_u on viisi ja alempi määrittelyraja z_l on yksi. Prosessin toiminnasta otetaan sadan mittauksen otos, jonka tulokset ovat kuvattuina taulukossa 4. Taulukon tuloksissa esimerkiksi arvo 0 on ilmentynyt kerran ja arvo 2 on ilmentynyt 25 kertaa.

Arvo	Frekvenssi
0	1
1	4
2	25
3	47
4	16
5	6
6	1

Taulukko 4. Esimerkin populaatio.

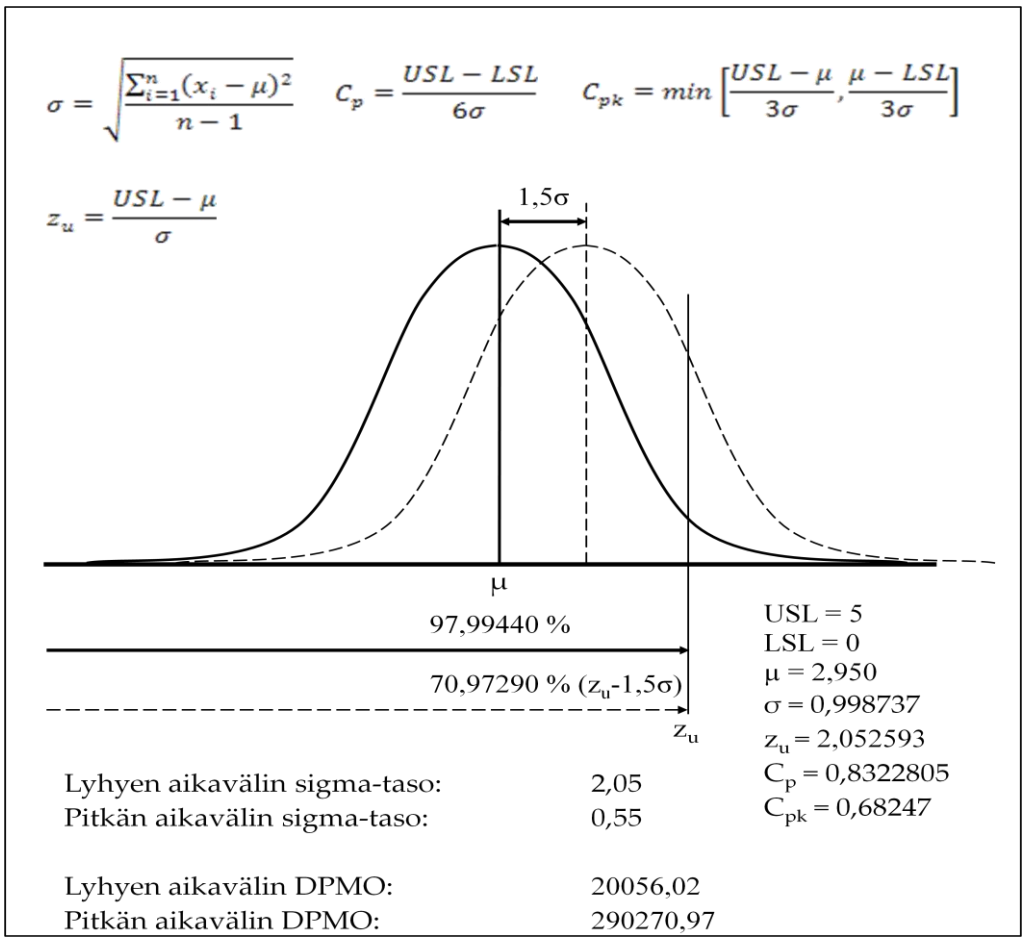


Kuva 10. Esimerkin 1 tulos.

Otokselle lasketaan keskiarvo μ, sekä keskihajonta σ. Tässä vaiheessa tiedetään kaikki tekijät sigma-tason laskemista varten. Sigma-taso lasketaan sekä lyhyelle, sekä pitkälle aikavälille. Sigma-taso määrittelyvälille [1,5] on kuvattuna kuvassa 10.

Esimerkki 2

Toisessa esimerkissä käytetään samaa populaatiota, mutta tarkoituksena on kuvata määrettä, joka ei saa negatiivisia arvoja. Ylempi määrittelyraja z_u on 5, eli kaikki arvon 5 saaneet mittaustulokset ja sitä pienemmät ovat hyväksyttyjä. Tämä esimerkki eroaa edellisestä siten, että pienempää määrittelyrajaa ei käytetä normaalijakauman todennäköisyyslaskentaan. Sigma-taso määrittelyvälille [-∞, 5] on kuvattu kuvassa 11.



Kuva 11. Esimerkin 2 tulos.

4.2.2. Prosessin suorituskyky

Six Sigma -organisaatiot käyttävät kahta indeksiä kuvaamaan prosessin suorituskykyä. Suorituskykyindeksi C_p kuvaa prosessin kykyä tuottaa yhdenmukaista tulosta. Mitä yhdenmukaisempaa prosessin toiminta on, sitä suurempi C_p on. Suorituskykyindeksi C_{pk} kuvaa kuinka hyvin prosessin tulokset täyttävät tavoitteensa. Mitä paremmin prosessin tulokset sopivat ylemmän ja alemman määritysrajan väliin, sitä lähempänä

C_{pk} on arvoa 1,5. Prosessi on Sigma-tasoltaan n. 6 kun C_p >= 2,0 ja C_{pk} = 1,5. Suorituskykyindeksien C_p ja C_{pk} arvot on laskettu sigma-tason esimerkeissä, kuvissa 10 ja 11. Arvojen C_p ja C_{pk} –arvojen kaavat on esitelty kaavassa 3.

USL = ylempi määrittelyraja
LSL = alempi määrittelyraja
μ = keskiarvo
σ = keskihajonta

$$C_p = \frac{USL - LSL}{6\sigma}$$

$$C_{pk} = min\left[\frac{USL - \mu}{3\sigma}, \frac{\mu - LSL}{3\sigma}\right]$$

Kaava 3. Prosessin suorituskyvyn laskentakaava.

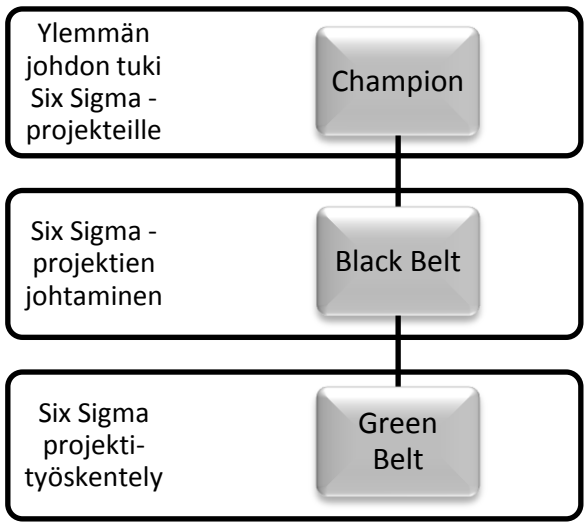
4.2.3. Six Sigma –mittareiden vaatimuksia

Yksi eniten käytetyimmistä Six Sigman dogmista on ”mittaa mitä arvostat” [Tayntor, 2007]. Edellä mainitun lauseen voi myös kääntää asiakaslähtöiseksi lauseeksi ”mittaa mitä asiakkaasi arvostaa”. Asiakasvaatimukset ovat keskeinen lähde mittareille. Asiakasvaatimuksia hyödyntämällä tiedetään, että mittareilla mitataan juuri niitä asioita, joita asiakas arvostaa. Mittareiden tulisi myös tuottaa tietoa liiketoiminnan analysoinnin näkökulmasta. Mittarit kannattaa luoda yhteistyössä prosessiin kuuluvien organisaatioyksiköiden kesken, jolloin saadaan katettua tärkeimmät mittauskohteet jokaisen yksikön kannalta.

Ylempi johto ei kerkeä perehtymään yksittäisten mittareiden tuloksiin, vaan he tarvitsevat laajemman kokonaiskuvan prosessista. Mittareiden tulisi siis tuottaa johdolle dataa, josta selviää prosessin kokonaistehokkuus, sisältäen heikkoudet ja vahvuudet. Prosessiin voidaan asettaa mittareita, jotka kertovat heikkouden tai vahvuuden omalta alueeltaan. Prosessista voidaan muodostaa funktio, jossa koko prosessin tehokkuus muodostuu osaprosessien tehokkuusarvoista. Six Sigma projekteissa puhutaan tällöin isoista ja pienistä y-kirjaimista. Yleensä abstraktiotasoa nostava funktio kirjataan muodossa $Y=\{y_1, y_2, y_3, \ldots, y_n\}$, jossa jokainen mittarin tulos vaikuttaa funktioon. Tietyn tason isot Y:t voidaan kuvata uutena funktiona korkeammalle organisaatiotasolle. Tällä lailla ylemmälle organisaatiotasolle voidaan antaa helpommin tulkittavaa dataa, jonka pohjalta on helpompi tehdä päätöksiä. Prosessin valvonnan kannalta on tärkeää, että mittarin tuottama data voidaan käsitellä heti tulkittavaksi, jolloin prosessia voidaan nopeasti ohjata toivottuun suuntaan.

4.3. Six Sigma -projektin roolit

Kuten jo aikaisemmin on mainittu, Six Sigman määritelty roolirakenne on ainutlaatuista nykyaikaisessa laadun tai toiminnan hallinnassa [Schroeder et al., 2008]. Roolirakenteessa on hierarkia, joka toimii organisaation eri tasoilla. Seuraavissa kappaleissa kuvataan Six Sigman keskeiset roolit ja niiden vastuualueet Kellerin [2005] ja Tayntorin [2007] kuvausten pohjalta. Six Sigman roolirakenne on hierarkkisesti kuvattuna kuvassa 12 siten, että hierarkia laskee kuvasta katsottuna ylhäältä alas.



Kuva 12. Six Sigman roolihierarkia.

4.3.1. Champion

Champion-roolissa toimivan henkilön päätehtävänä on varmistaa, että organisaatio pystyy tukemaan Six Sigman käyttöönottoa. Champion on vahva ja esillä oleva Six Sigman puoltaja. Champion-rooli asetetaan henkilölle, joka toimii organisaation ylemmässä tai keskijohdossa.

Champion-roolissa oleva henkilö toimii auktoriteettina organisaationsa Six Sigma –ryhmille. Champion vastaa Six Sigma –ohjelman kehittämisen, projektien valinnan ja resurssien varaamisen kautta siitä, että organisaatiolla on mahdollisuus käyttää Six Sigma –metodologiaa ja onnistua kehityksessä [Keller, 2005, s. 27]. Champion-roolissa oleva henkilö edustaa organisaation johdon tukea Six Sigmalle ja vakuuttaa organisaatiolle, että Six Sigma on pitkäaikainen valinta, eikä väliaikainen yritys. Yleensä Champion käy aluksi Green Belt –koulutuksen, jotta hän ymmärtää Six Sigma –projektien toiminnan ja pystyy siten tukemaan Six Sigma –toimintaa paremmin.

4.3.2. Black Belt

Black Belt –roolissa olevat henkilöt toimivat yleensä täysiaikaisesti Six Sigma –kehitystehtävissä. Black Belt on Six Sigma –työkalujen ekspertti, joka hallitsee Six Sigmaan liitetyt analyysi-, tilastotiede- ja ongelmanratkaisutyökalut. Edellä mainitut

työkalut liittyvät DMAIC-prosessin hallitsemiseen, prosessin Sigma-tason määrittämiseen ja juurisyiden tunnistamiseen. Tilastotiedetyökaluilla tarkoitetaan erilaisia tilastollisia tapoja analysoida mittaridataa. Black Belt –henkilöt toimivat projektipäällikköinä samanaikaisesti yhdessä tai useammassa Six Sigma –projekteissa.

Black Belt –henkilöt ovat muutoksenhallitsijoita. Heidän tehtävänä on saada nostettua organisaation Six Sigma -projektien toiminnan Sigma-taso tasolle 6. Six Sigma –tason korottaminen 6. tasolle vaatii Black Belt -henkilöiltä syvää ymmärrystä organisaation toiminnasta, sekä kulttuurista.

4.3.3. Green belt

Green belt –henkilöt ovat organisaation työntekijöitä, joilla on perustiedot Six Sigma –metodologiasta ja sen työkaluista. Six Sigma –organisaatio kouluttaa ajan mittaan suurimman osan työntekijöistään Green Belt –henkilöiksi, jolloin heillä on pätevyys olla mukana Six Sigma –projektiryhmässä.

Green Belt –henkilöt työskentelevät heille osoitetussa projektissa Black Belt –henkilön alaisina. Heidän tehtävänä on tuoda tarkkaa tietoa parannettavasta prosessista projektiryhmälle. Green Belt –henkilöt vastaavat projektiryhmän aivoriihistä ja mittausdatan keräämisestä.

4.3.4. Yellow belt

Koulutusyritykset tarjoavat Six Sigman roolirakenteen mukaisia koulutuksia. Koulutuksiin on ilmestynyt myös Yellow belt –koulutus [SixSigmaOnline, 2010]. Koska tämä käsite ei ilmene todellisena roolina Six Sigman roolirakenteessa, sitä ei kutsuta tässä tutkielmassa Six Sigman roolirakenteeseen kuuluvaksi rooliksi. Yellow belt –koulutus on johdanto Six Sigmaan ja sen hyödyntämiseen. Lisäksi koulutus auttaa tunnistamaan kohteita, joissa Six Sigmaa voisi hyödyntää ja miten Six Sigma –projektia seurattaisiin.

5. Six Sigma ja ohjelmistotuotanto

Six Sigmaa hyödynnetään laajasti perinteisessä tuotannossa [SixSigmaCompanies, 2010], missä samaa tuotetta tuotetaan massoittain vakaan tuotantoprosessin kautta. Tuotantoprosessin kautta syntyvien tuotteiden tulee täyttää tuotteelle asetetut vaatimukset ja laatumääritykset, sekä tuotantokulujen tulisi pysyä samana tuotekappaleiden välillä. Tässä luvussa tutkitaan, miten Six Sigma soveltuu ohjelmistotuotantoon.

5.1. Kritiikkiä Six Sigman soveltuvuudesta ohjelmistotuotantoon

Six Sigman väitetään olevan teknisesti riippumaton toimialasta [Siviy et al., 2008]. Väite perustellaan siten, että kaikki ihmisten ja organisaatioiden toiminta voidaan käsitellä prosesseina ja Six Sigma käsittelee prosesseja. Kaikki eivät kuitenkaan koe Six Sigmaa ohjelmistotuotantoon soveltuvaksi, vaikka ohjelmistoyritykset ovat raportoineet säästöistä ja kehittymisestä, jotka ovat olleet Six Sigman ansiota. Zun ja muiden [2008] mukaan osa raportoineista yrityksistä on omannut heikon laatutason, joka olisi parantunut millä tahansa tuotantoprosessin kehittämismenetelmällä. Soveltumista on katseltava ohjelmistotuotantoprosessin piirteiden näkökulmista, joihin Six Sigma vaikuttaisi, sekä tuotettavan tuotteen piirteiden näkökulmasta.

Binder [1997] perustelee kolme syytä, joiden perusteella Six Sigma ei sovellu ohjelmistokehitykseen, vaan tulisi jättää perinteisen teollisuuden työkaluksi. Ensimmäiseksi, ohjelmistotuotantoprosessi ei noudata samoja kurinalaisuuksia kuin fyysisen tuotteen tuotantoprosessi. Ohjelmistoprosessin käyttäytyminen on amorfinen möykky verrattuna objektin rajoitettuun ja rajalliseen käyttäytymiseen. Toiseksi, ohjelmiston ominaisuudet eroavat fyysisestä tuotteesta. Ohjelmiston arvollisia ominaisuuksia ei voi ilmaista tarkasti ordinaalisena toleranssina. Toisin sanoen, ohjelmiston virheettömyyttä tai virheellisyyttä ei voi mitata etäisyyksinä, painoina tai muina fyysisinä ominaisuuksina. Ohjelmisto joko täyttää vaatimukset tai ei täytä. Tämän takia pienemmän tai suuremman täyttymistoleranssin määrittely on merkityksetöntä. Ohjelmistovikojen ja havaittujen virheiden suhde on monesta moneen. Lisäksi, kaikki havaitut virheet eivät ole ohjelmistovikoja. Osa havaituista virheistä voi johtua infrastruktuurista, jonka päällä ohjelmisto toimii. Tällaisia infrastruktuurin osia ovat esimerkiksi käyttöjärjestelmä ja laitteistoajurit. Kolmanneksi, ohjelmisto on ainutlaatuinen, eikä massatuote. Ohjelmisto ei ole fyysinen objekti, eikä konkreettinen palvelu, vaan ohjelmisto on sarja binäärikoodia, joka sijaitsee tallenteella. Ohjelmistotuote toimii kaikissa yhteensopivissa teknisissä ympäristöissä samalla lailla, joten samaa tarvetta täyttävää ohjelmistotuotetta ei tarvitse tuottaa samaan ympäristöön toista kertaa.

Motorolan havaintojen perusteella materiaalivariaatiot, valmistusmenetelmät ja luonnollinen kuluminen aiheuttavat prosessin suorituskyvyn heikkenemisen pitkällä

aikavälillä, joka näkyy sigma-tason laskentakaavassa. Ohjelmistotuotannossa ei ole materiaalivariaatioita, koska tuotteet eivät koostu materiaalista. Tietokoneet eivät myöskään tee virheitä ohjelmistotuotteisiin kulumisen takia, eivätkä menetä laskentatehoa käytön takia. Motorolan perustelut siirtymälle eivät näyttäisi pätevän suoranaisesti ohjelmistotuotannossa. Siviy ja muut [2007] kuitenkin olettavat, että ohjelmistokehityksessä on olemassa tekijöitä, jotka mahdollistaisivat siirtymän myös ohjelmistokehityksessä. Tällaisia syitä ovat esimerkiksi prosessin mukaisen toiminnan laiminlyönti pitkällä ajalla, oppimiskäyrä ja koko ajan muuttuvat työkalut ja teknologiat.

Tuote koostuu komponenteista ja osista, jonka takia Six Sigma -metodologiassa tuotetta tutkitaan myös alimmalla tasolla. Ohjelmistotuotteen alin komponentti on koodirivi. Jokaisella koodirivillä on yksilöllinen tehtävä ohjelmistotuotteessa, eli jokaisella rivillä on omat ominaisuudet. Koodirivi voi liittyä eri tavoin useisiin asiakasvaatimuksiin [Romanchik, 2004]. Perinteisessä tuotannossa tuotteella on pysyvät ominaisuudet ja asiakasvaatimukset, eli samasta tuotantolinjasta tuotettujen tuoteyksiköiden tulisi olla mahdollisimman samanlaisia. Koodirivi ei siis edellä mainitusta näkökulmasta ole verrannollinen perinteisen tuotantolinjan tuoteyksikköön. Voi siis olla, että Six Sigma ei sovellu ohjelmistotuotantoon yhtä hyvin kuin perinteiseen teollisuuteen, koska ohjelmistotuotannon alimman tuotantoyksikön, eli koodirivin, ominaisuudet eroavat aina tuotantoyksikköjen välillä.

DMAIC-toimintamallissa tuotteesta löytyvä virhe on analysoitava ja virheelle tulee määrittää sen juurisyy. Perinteisessä teollisuudessa virhe liittyy usein tuotteen mitattaviin ominaisuuksiin, esimerkiksi pituuteen tai paksuuteen. Kun tuotteen ominaisuudessa havaitaan hajontaa, tuotantoprosessista tutkitaan osa-alueet, jotka vaikuttavat kyseiseen ominaisuuteen. Mekaanisessa tuotantoprosessissa edellä kuvaillun virheen juurisyy voi löytyä esimerkiksi koneiston kalibroinnista tai kulumisesta. Ohjelmistotuotteen virheen syy voi johtua esimerkiksi määrittelystä, vaatimustenhallinnasta, moniselitteisestä asiakasvaatimuksesta, testauksesta tai ohjelmointivirheestä. Kun juurisyy ilmenee joltain ohjelmistotuotantoprosessin osa-alueelta, syy todennäköisesti on jo yleisesti tiedossa oleva ohjelmistotuotannon luonteeseen liittyvä ongelma, johon ei ole kehitetty universaalia ratkaisua. DMAIC-toimintamallin kehitysvaiheen parannusehdotusten tulisi pureutua ongelmiin, joihin ei välttämättä löydy pieniä konkreettisia muutoksia.

Nicoletten [2005] mielestä Six Sigma soveltuu vain sellaisille aloille, joissa toiminta muistuttaa luonteeltaan teollista tuotantoa. Nicolette perustelee eri tavoin, että ohjelmistokehitys muistuttaa luonteeltaan rakennusalaa. Hyvänä esimerkkinä ominaisuuksien samanlaisuudesta ovat suunnittelumallit (design patterns) [Gamma et al., 1994], jotka on huomattu hyviksi työkaluiksi ratkaisemaan ohjelmistokehityksen ongelmia. Suunnittelumalleja voidaan käyttää äärettömän monta kertaa eri ongelmissa,

joihin ne sopivat, mutta aina erityisellä tavalla. Samalla lailla vuosituhansien saatossa rakennusalalla on opittu parhaita toimintatapoja, joilla voidaan luoda toimivia ratkaisuja rakentamisen ongelmiin.

Aivan kuten rakennuksella, ohjelmakoodilla on arkkitehtuuri, jonka tulee varmistaa, että nykyiset vaatimukset voidaan toteuttaa ohjelmaan. Tämän lisäksi ohjelmistoarkkitehtuurin tulisi olla joustava tuleville muutoksille ja lisäyksille. Ohjelmistoarkkitehtuuri onkin avaintekijä ohjelmakoodin ylläpidettävyyteen. Ohjelmistoarkkitehtuurin arviointi on tärkeä osa ohjelmiston laadun määritystä. Arkkitehtuuri ei ehkä näy asiakkaalle suoranaisesti, mutta hyvän arkkitehtuurin tuomat edut vaikuttavat asiakkaan kokemaan laatuun. Tulevat muutokset ohjelmakoodiin voidaan tehdä olemassa olevan rakenteen perusteella, jolloin ylläpidettävyys säilyy. Lisäksi selkeä rakenne tukee muutosten jäljitettävyyttä, jolloin ohjelmaan jää todennäköisesti vähemmän ohjelmointivirheitä. Arkkitehtuuria ei pysty arvioimaan Six Sigman keskihajontaan perustuvalla menetelmällä. Ohjelmistoarkkitehtuuri suunnitellaan järjestelmän, sekä sen ympäristön vaatimusten perusteella, käyttäen hyväksi suunnittelumalleja ja ongelmanratkaisukykyä.

Ohjelmistotuotanto vaatii ohjelmistokehittäjiltä loogista päättelykykyä ja ongelmanratkaisukykyä. Ohjelmistotuotantoprosessi koostuu pitkälti inhimillisestä luovasta toiminnasta, jonka lopputulosta ei voi tarkalleen määrittää. Luovan toiminnan prosessia on vaikea kartoittaa tai mitata. Siksi, jo pelkästään ihmisen kognitiivisessa prosessissa tapahtuva variaatio hankaloittaa Six Sigman soveltamista ohjelmistotuotantoprosessiin.

5.2. Six Sigman soveltaminen ohjelmistotuotantoon

Vaikkakin Six Sigman soveltumista ohjelmistokehitykseen on kritisoitu voimakkaasti, usea ohjelmistoyritys on ottanut Six Sigman käyttöön kehittääkseen ohjelmistotuotantoprosessejaan [SixSigmaCompanies, 2010] [Antony and Fergusson, 2004]. Kuten kritiikissä mainitaan, Six Sigman suoraviivainen soveltaminen ohjelmistokehitykseen ei todennäköisesti toimisi kunnolla, johtuen ohjelmistokehityksen erityispiirteistä. Ohjelmistoyritykset ovatkin sopeuttaneet Six Sigmaa ohjelmistokehitykseen, eivätkä ohjelmistokehitystä Six Sigmaan. Antonyn ja Fergussonin [2004] mukaan kriittisimmät onnistumiseen vaikuttavat tekijät Six Sigman käyttöönottamisessa ohjelmistotuotantoon ovat vankkumaton ylimmän johdon tuki, organisaatiokulttuurin muutos, Six Sigman liittäminen liiketoimintastrategiaan sekä liiketoimintatavoitteisiin ja asiakasvaatimusten ymmärrys sekä niiden vaikutus Six Sigman toimintamalleihin.

Six Sigman sovellustapoja on monia, mutta niitä yhdistää ohjelmistotuotantoprosessin mittaaminen, mittaustulosten analysointi ja

ohjelmistotuotantoprosessin kehittäminen. Tässä kappaleessa kuvataan raportoituja Six Sigman sovellustapoja ohjelmistokehityksessä.

5.2.1. Six Sigman soveltaminen raportoitujen virheiden perusteella

Redzik ja Baik [2006] tutkivat DMAIC-toimintamallin soveltamista ohjelmistokehityksen laadun kehittämiseen. Mallin kehitys tapahtui Six Sigman kotikentällä, sillä malli kehitettiin Motorolan toimesta. Heidän mallissaan Six Sigmaa sovelletaan tarkasti ja sen tilastotyökaluja hyödynnetään monin eri tavoin. Redzikin ja Baikin [2006] tutkimuksessa olleen tapauksen DMAIC-iteraation tuli olla valmis kuudessa kuukaudessa ja sen avulla oli tarkoitus parantaa ohjelmistojen laatua huomattavasti seuraavan kahden vuoden aikana. Tavoitteena oli kehittää prosessia siten, että ohjelmistotuotteisiin jäisi vähemmän kriittisiä virheitä. Ohjelmistokehitys suoritettiin vesiputousmallin mukaisesti. Seuraavaksi kuvataan Redzikin ja Baikin [2006] malli Six Sigman soveltamisesta ohjelmistokehitykseen.

Määrittelyvaiheessa Six Sigma –projekti määriteltiin projektin perustuskirjaan. Six Sigma –ryhmän jäsenet valittiin siten, että jokaisesta ohjelmistotuotantoprosessin vaiheisiin liittyvistä organisaatioista otettiin edustaja, jolla olisi aikaa sitoutua tehtävän suorittamiseen oman osastonsa osalta. Projektiin piti saada edustajat esim. tuotannosta, testauksesta, markkinoinnista ja ylemmästä johdosta. Six Sigma –ryhmän jäsenten keskinäiset suhteet ovat tärkeitä projektin onnistumisen kannalta, joten ryhmä kokoontui viikoittain keskustelemaan yhteisistä odotuksista, sekä tavoiteltavista eduista. Ryhmän tulee pystyä mittaamaan kehitys yhteisellä tavalla siten, että jokainen ymmärtää yhteisen edun ja sen merkityksen omalle osastolle. Ryhmä jaettiin viiteen aliryhmään, joista jokainen sai erikoistumisalueensa. Erikoistumisalueita olivat front-end, ohjelmistokehitys, järjestelmätestaus, esijulkaisutestaus ja nopeat voitot. Yhteinen kehitysmittari oli ohjelmiston luotettavuuden kasvu testausvaiheessa. Tämä mittari perustuu Goelin ja Okumoton eksponentiaaliseen malliin [Goel and Okumoto, 1979], joka suhteuttaa testauksen aikana löydetyt virheet testausaikaan ja määrittelee arvion vielä löytämättä olevien virheiden määrästä. Redzikin ja Baikin [2006] raportoimassa tapauksessa, useiden Motorolan ohjelmistojulkaisujen arvioitujen virhemäärien ja raportoitujen virheiden määrät korreloivat. Tällä tavoin voitiin todentaa, että mittari voisi ilmaista todellisen laatukehityksen.

Nykyinen prosessi määritettiin SIPOC-analyysin avulla. Samassa yhteydessä määritettiin prosessin vaiheet, joista saadaan syötteet käytettäville Six Sigma -mittareille. Asiakkaiden ääntä ja odotuksia analysoitiin useiden vuosien asiakastyytyväisyyskyselyjen perusteella. Kano-mallia varten data koostettiin kolmeen kategoriaan: perustarpeisiin, tyydyttäviin tekijöihin ja ilahduttaviin tekijöihin. Perustarpeet ovat ominaisuuksia, joiden oletetaan olevan ohjelmistossa, vaikka niitä ei erikseen pyydetä. Esimerkiksi ohjelmistolaatu on perustarve, sillä asiakas olettaa, että

järjestelmä toimii virheettömästi. Tyydyttävät tekijät ovat ominaisuuksia, joita asiakas odottaa ja joiden toteuttaminen kasvattaa asiakastyytyväisyyttä. Ilahduttavat tekijät ovat ominaisuuksia, joita asiakas ei erikseen osannut vaatia, eikä siten myös maksa, mutta joiden toteutus täyttäisi jälkikäteen tulevia tarpeita. Liiketoiminnan ääni määritti kriittisten virheiden lukumäärän, jota kehityksen onnistuessa yksikään osasto ei ylittäisi.

Nopeisiin voittoihin erikoistunut ryhmä tutki ohjelmistotuotantoprosessiin kuuluvien osastojen prosesseja. Tarkoituksena oli löytää ns. matalalla roikkuvia hedelmiä, eli prosesseja, joita olisi helppoa ja nopeaa kehittää tuottamaan korkean arvon suhteessa sijoitukseen.

Mittausvaiheessa määritettiin mittarit jokaiselle osastolle ja mittausjärjestelmä arvioitiin. Mittarit asetettiin SIPOC-analyysin mukaan syötteille, itse tuotantoprosessille, sekä tulosteille. Syötteiden mittarina toimi kuukausittainen vaihehallintadata, josta tutkittiin erityisesti syötteiden yhdenmukaisuutta. Vaihehallintadatan mittarit ovat ohjelmistotuotantoprosessin eri vaiheissa käytettäviä mittareita, jotka kertovat vaiheen toimintakyvystä, sekä kyvystä tunnistaa kyseisessä vaiheessa tuotettu virhe, sekä käsitellä aikaisemmassa vaiheessa tuotettu virhe. Tuotantoprosessin tehokkuutta mitattiin virhetiheydellä. Viikoittaisten asiakkaiden raportoimien kriittisten vikojen määrä toimi tulostemittarina. Nykyisen prosessin toimintakyky mitattiin ja vertailupohja muodostettiin. Mittausdatan keräämiselle luotiin suunnitelma.

Analyysivaiheessa tutkittiin syitä mittarituloksiin hyödyntämällä erilaisia työkaluja. Julkaisuihin päätyvät virheet ovat keskeisin ongelma, joten asiakkaiden raportoimien virheiden juurisyyt on analysoitava huolella. Tähän käytettiin Pareto-analyysia, jonka avulla oli tarkoitus tunnistaa, kuinka paljon ohjelmistotuotantoprosessin eri vaiheissa tuotetaan virheitä. Vaiheen prosessi on toimintakyvyltään huono, jos se tuottaa systemaattisesti virheitä. Virheitä syntyy suhteessa ohjelmiston kokoon, joten koko on otettava huomioon analyysissa. Virheet jaettiin kriittisyyden perusteella kolmeen ryhmään, joille jokaiselle laskettiin korrelaatio ohjelmiston kokoon verrattuna. Virheiden lisäksi testausajan tehokkuus analysoitiin. Tarkoituksena oli arvioida testausajan suhde julkaisuaikatauluun.

Kehitysvaiheessa analyysin tuloksia tutkittiin syy-seuraus-kaavion avulla. Syiden ja seurauksien pohjalta laadittiin parannusehdotuksia, joiden avulla virheiden syyt voitaisiin poistaa. Kehitysideoiden tehokkuus arvioitiin, sekä arvioitu hinta- ja hyötysuhde dokumentoitiin. Tässä tapauksessa parannukset liittyivät uusien teknologioiden käyttöönottoon, joiden valinnoissa käytettiin asiantuntijoita.

Valvontavaiheessa ryhmän jäsenille jaettiin vastuualueet kyseisen vaiheen osa-alueista. Vastuualueita olivat uusien kehitystapojen käyttöönotto, kehitystapojen arviointi, kehitystapojen aiheuttamien muutosten standardisointi, sekä jatkuvan mittauksen datan keräys ja mittaustulosten generointi. Toimintakykyä valvottiin

katselmoinneilla, joissa tarkistettiin mittareiden tulokset, mahdollisten ongelmien tunnistaminen, sekä pakollisten muutosten priorisointi. Tuotantoprosessin laatua katselmoitiin myös kuukausittaisissa laatukatselmoinneissa.

Redzik ja Baik [2006] dokumentoivat, että kehitysiteraatio onnistui ja tavoitteeksi asetetut laatuparannukset saavutettiin mallin avulla. Tutkijat olivat vakuuttuneita, että Six Sigman avulla voi saavuttaa ohjelmistolaadun kehityksen suunnitellusti.

Zhao ja muut [2008] tekivät myös mallin DMAIC-toimintamallin hyödyntämisestä ohjelmistotuotantoprosessin laadun kehittämiseen. He tutkivat, miten DMAIC-prosessin avulla voitaisiin ehkäistä ohjelmistotuotantoprosessin tuottamia virheitä. Tuloksena syntyi kolmiosainen ohjelmistotuotantoprosessin hallintamalli, jonka kolme osa-aluetta on DMAIC-prosessin käyttö tuotantoprosessin kehittämiseen, IDOV-kehyksen käyttö vaatimustenhallintaan ja Six Sigma laadunhallintatyökalujen hyödyntäminen.

Mallin kuvaus alkaa vaatimustenhallinnasta, missä IDOV-kehyksen askelten mukaan tunnistetaan asiakasvaatimuksia sekä kriittisiä laatutekijöitä. Myös tässä mallissa käytetään SIPOC-analyysia, jonka avulla hahmotetaan eri asianomaiset, heidän syötteet ja tulosteet nykyisestä ohjelmistotuotantoprosessista. SIPOC-analyysin jälkeen kehitysryhmä analysoi nykyisen ohjelmistotuotantoprosessin tilaa, sekä määrittää prosessin kehityskohteet. Samalla ohjelmistotuotantoprosessista etsitään epäkohtia, sekä yritetään poistaa niitä. Myös tässä mallissa ohjelmistotuotantoprosessin toimintaa tutkitaan hajottamalla asiakasvaatimukset Kano-mallin avulla. Kano-mallin avulla voi nähdä, pyrkiikö nykyinen toiminta tekemään ohjelmiston, jota asiakas erityisesti haluaa.

Mittausvaiheessa nykyisen prosessin toimintakyky määritetään mittareiden avulla. Zhaon ja muiden [2008] mallissa keskeisin mittari ohjelmistotuotantoprosessille on kuukausittaisten raportoitujen asiakasongelmien määrä. Kokonaismäärään kuuluvat sekä ohjelmavirheestä johtuvat ilmoitukset, sekä ilmoitukset, jotka eivät liity ohjelmavirheisiin. Mittarin avulla ohjelmistotuotteiden historiatiedoista voidaan luoda nykyisen prosessin sigma-taso.

Kuten Redzikin ja Baikin [2006], myös Zhaon ja muiden [2008] mallin analyysivaiheessa asiakasongelmien juurisyiden selvittämiseen käytetään Pareto-analyysia. Pareto-analyysin tuloksista voidaan todeta osa-alueet, jotka aiheuttavat eniten asiakasongelmia. Ongelmien lähteet on tunnistettava ohjelmistotuotantoprosessin vaiheista. Ongelman juurisyitä analysoidaan tarkemmin syy-seuraus –analyysilla.

Kehitysvaihe etenee perinteisen DMAIC-toimintamallin mukaisesti, eli projektiryhmä ideoi kehitysehdotuksia havaittujen ongelmien korjaamiseksi. Valvontavaiheessa projektiryhmä ottaa kehitysmuutokset mukaan ohjelmistotuotantoprosessiin ja valvoo prosessin toimintakykyä suhteessa kuukausittaisten asiakasongelmien määrään.

Zhaon ja muiden [2008] mallin avulla pystyttiin kehittämään ohjelmistotuotantoprosessin sigma-tasoa n. 2,5 sigmasta 3,5 sigmaan. Ohjelmavirheisiin liittyvien asiakasongelmien kuukausittainen keskiarvo oli ennen kehitystä 30,5 ja kehityksen jälkeen 23,1. Zhao ja muut raportoivat käyttäneensä malliaan vain yhdessä organisaatiossa.

Zhaon ja muiden [2008] malli on suunnattu organisaatioille, joilla on omia ohjelmistotuotteita, joista tehdään uusia julkaisuja säännöllisin väliajoin. Tämä johtuu siitä, että keskeisin mittari on kuukausittaisten asiakasongelmien määrä, jonka pitäisi kertoa julkaisun laadusta. Mallissa tuote päätyy asiakkaille, jotka varmentavat tuotteen laadun käyttämällä tuotetta. Jos käyttäjä havaitsee ongelman tuotteen käytössä, hän tekee siitä ilmoituksen. Käytetty mittari ei siis kerro, kuinka virheetön tuote on, vaan kuinka paljon asiakkaiden ongelmiin on liittynyt ohjelmointivirheitä. Raportissa ei kerrota, kuinka asiakasongelmat jaettiin virheellisiin ja ei-virheellisiin kategorioihin, mutta tämä herättää kysymyksen, mitä ei-virheellisiin asiakasongelmiin kuului? Voi olla mahdollista, että ei-virheellisten joukkoon voisi päätyä esimerkiksi käytettävyyteen liittyviä asiakasongelmia, jotka liittyvät erityisesti asiakkaan kokemaan laatuun ja arvoon. Vaikka ohjelmointivirheisiin liittyvät asiakasongelmat vähenisivätkin, tulisi myös asiakasongelmien kokonaismäärän laskea samalla tavalla. Voi myös olla, että kaikki asiakkaat eivät ilmoita kaikista ongelmistaan, vaan toteavat tuotteen laadun heikkouden ja vaihtavat toimittajaa. Tosin, jos asiakkaalle asti päätyvät ohjelmointivirheet vähenevät, tuotantoprosessi on kehittynyt ja laatu jossain määrin parantunut. Tämä malli tosin ei kata laadunhallintaa yksin, vaan tueksi tarvitaan kattava paketti ohjelmistokehityksen laatumittareita. Laatu on monitahoinen käsite, eikä sitä voi määrittää yhdellä mittarilla.

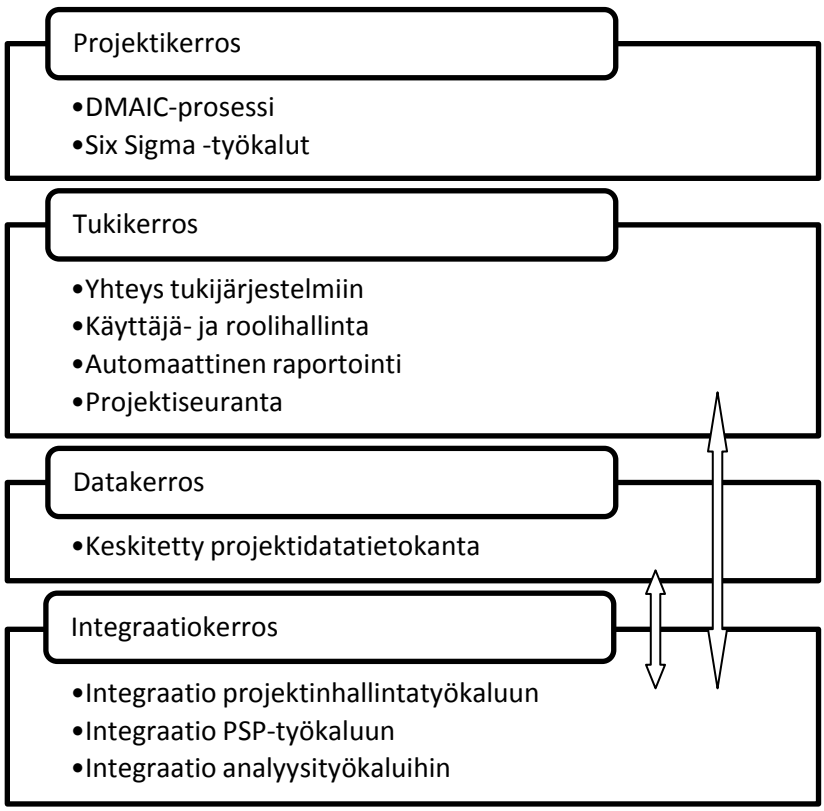
5.2.2. Laadunkehitys henkilökohtaisen ja ryhmäohjelmistoprosessin perusteella

CMM/CMMI-kypsyystasomallin mukaan prosessin mittaaminen ja valvonnanalainen hallinta ovat perusta jatkuvalle prosessikehitykselle. Ongelmana on, että kypsyystasomalli ei kerro, miten mallia sovelletaan käytännössä. Eräs tapa on soveltaa PSP/TSP–menetelmää, jonka käyttö on kuvattu ja se on yhteensopiva CMMI:n tavoitteiden kanssa.

Park ja muut [2006] tutkivat Six Sigman yhdistämistä PSP:hen kartoittamalla Six Sigman tilastollisia työkaluja PSP-prosessien eri vaiheisiin. He huomasivat, että PSP:n oleellisimpiin toimintoihin kuuluivat kolme ohjelmistokehittäjien toimintoa.

1. Ohjelmistokehittäjien tulee arvioida ohjelmakoot tehtävistä, sekä tallentaa arviointi ja toteuma.
2. Ohjelmistokehittäjien tulee tallentaa käytetty aika kaikista tehtävistä.
3. Ohjelmistokehittäjien tulee tallentaa kaikki viat, sekä seurata niiden elämänkaarta.

Esimerkkinä Park ja muut [2006] käyttivät PSP 1.0 -prosessin jälkiseuranta-vaihetta, jossa tarkastellaan ohjelmistokoon ja varattavien resurssien arvioinnin toteutumista. Ohjelmistokehittäjät arvioivat tehtävään kuluvan ajan, sekä koon koodiriveinä, eli LOC-arvona (Lines Of Code). Arvioista muodostetaan suhdeluku LOC/tunti, joka kertoo, kuinka paljon ohjelmistokehittäjä on arvioinut kirjoittavansa tuotantokelpoista ohjelmakoodia tunnissa. Jokaisesta tehtävästä jää historiatietona arvio ja toteuma, jotka auttavat arvioiden muodostamista. Arvioiden ja toteutumien normaalijakauman noudattaminen tutkitaan, jotta t-testin soveltuminen voidaan todentaa. keskiarvoa verrataan toteutumien keskiarvoon käyttämällä parillista t-testiä 95 % luottamusvälillä. Testi osoittaa, kuinka hyvin arviot ovat onnistuneet. Tilastotyökalut voivat arvioida datasta eroja, jotka auttavat parempien arvioiden tekemiseen. Parempien arvioiden kautta projektien aikataulut vastaavat tarkemmin todellisuutta ja ohjelmistokehittäjät saavat tarpeeksi aikaa tuottaa laadukasta koodia. Park ja muut [2006] toteavat, että Six Sigman ja PSP:n välillä on synergiaa ja että PSP ja sen mittarit antavat perustan Six Sigman hyödyntämiseen. Ryhmätason kehitys mahdollistetaan TSP:n avulla kerätystä datasta. Näin prosessin kehitys nousee alemmilta organisaatiotasoilta korkeammille ja koko ohjelmistotuotantoprosessin kehitys on ohjattavissa Six Sigman soveltamisella.



Kuva 13. Six Sigma kehyksen arkkitehtuuri [Pan et al., 2007].

Pan ja muut [2007] jatkoivat ajatusta Six Sigman ja PSP:n yhdistämisestä ohjelmistolaadun kehittämisen tarpeisiin. He tekivät huomion, että PSP data pitää kerätä manuaalisesti, mikä hankaloittaa Six Sigman hyödyntämistä. Lisäksi manuaalinen datan keruu voi vääristää tuloksia inhimillisten virheiden tai mahdollisen vastarinnan kautta. Tässä korostuu Six Sigman heikkous datan keruun suhteen, sillä se ei määritä tapoja, joilla dataa voisi kerätä prosessista. Tämän vuoksi Pan ja muut [2007] lähtivät kehittämään ohjelmistokehitykseen kehystä, joka yhdistäisi automaattisen datan keruun ja DMAIC-toimintamallin hallinnan. Tuloksena syntyi neljä kerroksinen kehys, joka on kuvattu kuvassa 13.

Projektikerros tarjoaa DMAIC-toimintamallin vaiheiden hallintatyökalut. Lisäksi projektikerros sisältää toteutuksia Six Sigma –analyysityökaluista, joita käytetään DMAIC-toimintamallin eri vaiheissa. Tukikerros tarjoaa käyttäjä- ja roolihallinnan, raporttien generoinnin, projektiseurannan, sekä tuen integraatiotasolle. Datakerros sisältää tietovaraston, johon tallennetaan keskitetysti Six Sigma –projektien tuottamat datat. Integraatiotaso mahdollistaa automaattisen datan keruun toisten sovellusten kautta. Pan ja muut [2007] käyttivät kahta erillistä ohjelmistoa projektidatan ja PSP-datan keräämiseen. PSP-työkalu [Shin et al., 2007] keräsi automaattisesti ohjelmistokehittäjien työpäätteiltä PSP-dataa, kuten yksikkötestien virheitä, ohjelmistokokoa rivimäärissä, kääntämiseen kuluvaa aikaa ja resurssienkäyttöä. Lisäksi PSP-työkalu tuki PSP0 ja PSP0.1 toimintoja, kuten suunnittelua ja ohjelmistokoon arviointia. Projektidataa kerättiin CMMI-kehykseen perustuvalla projektinhallintatyökalulla [Symphony, 2010]. Sen lisäksi, että projektinhallintatyökalulla pystyi tukemaan projektinhallintaa, sillä pystyi muodostamaan nykyisen toimintaprosessin ja resurssinhallinnan sen pohjalle.

5.2.3. Asiakkaan prosessin kehittäminen

Sen sijaan, että Tayntorin [2007] sovellusmalli keskittyisi ohjelmistoyrityksen ohjelmistotuotantoprosessin kehittämiseen, keskitytäänkin Six Sigman työkalujen avulla kehittämään asiakkaalle tuote, joka parantaisi asiakkaan tuotantoprosessia mahdollisimman tehokkaasti. Kuvauksessa sovelletaan Six Sigmaa ns. perinteiseen ohjelmistokehityksen elämänkaareen, eli vesiputousmalliin. Tayntorin kuvauksessa tietojärjestelmän tuottava taho on organisaation IT-osasto ja asiakkaana toimii jokin saman organisaation toinen osasto. Vaikka kuvaus ei vastaa markkinoiden tyypillistä tilannetta, missä ohjelmistokehitykseen erikoistunut yritys kauppaa osaamistaan toisille organisaatioille, tässäkin kuvauksessa voidaan tunnistaa erillinen tuottaja ja asiakas. Tosin, kuvauksen tapainen asiakkaan ja tuottajan välinen suhde ei toteudu yleisesti eri organisaatioiden välillä, koska erillisen IT-osaston tehtävä on mahdollistaa omaa organisaatiota tukeva tietojärjestelmä ja ylläpitää sitä. Tällaisessa suhteessa ei ole markkinatilannetta, jossa olisi useita kilpailevia tuottajia. Tämän tapainen kehittäminen

soveltuu IT-osaston toimintaan paremmin kuin ulkoisen ohjelmistoyrityksen toimintaan, koska ulkoinen ohjelmistoyritys ei todennäköisesti pystyisi seuraamaan ja tutkimaan asiakkaan prosesseja jatkuvasti pitkällä aikavälillä. Tayntorin kuvaus on pitkä, joten koko mallin kuvaamisen sijaan tässä kohdassa on tarkoituksena kertoa piirteistä, jotka Six Sigma tuo mukanaan perinteiseen vesiputousmalliin.

Tayntor käyttää mallissaan kuusivaiheista kuvausta vesiputousmallista. Mallin vaiheet ovat projektin aloitus, järjestelmäanalyysi, järjestelmäsuunnittelu, rakentaminen, testaus ja käyttöönotto. Six Sigman DMAIC-toimintamalli muistuttaa rakenteeltaan jokseenkin vesiputousmallia. Tayntor vaiheistaakin DMAIC-mallin ja vesiputousmallin vaiheet yhteen. Vaiheistus on kuvattuna taulukossa 5 siten, että ensimmäisessä sarakkeessa on Tayntorin tunnistama vesiputousmallin vaihe ja toisessa sarakkeessa vesiputousmallin vaihetta vastaava Six Sigman DMAIC-vaihe.

Vesiputousmallin vaihe	**Six Sigman DMAIC-toimintamalli**
Projektin aloitus	Määrittely, mittaus, analyysi
Järjestelmäanalyysi	Määrittely, mittaus, analyysi
Järjestelmäsuunnittelu	Analyysi
Ohjelmointi	Kehitys
Testaus	Kehitys
Käyttöönotto	Kehitys ja hallinta

Taulukko 5. DMAIC-toimintamallin vaiheistus vesiputousmallissa [Tayntor, 2007].

Projektin aloitusvaiheessa yhdistetään vesiputousmallin projektisuunnittelu, sekä Six Sigma –projektin määrittely. Tayntorin kuvauksessa Champion-roolin tulisi tulla asiakkaan puolelta, joka tarkoittaa, että asiakkaan edustaja on Six Sigma –projektin suurin tukija. Projektihenkilöstö muodostetaan Champion-roolin ja tuottajapuolen vastuuhenkilön yhteistyönä. Jos prosessiin kuuluu muitakin osapuolia, heiltä tulisi saada edustajat projektiin. Kun projektiryhmä on muodostettu, ryhmä tutkii projektin alustavia tekijöitä. Ryhmän tulee perustaa Six Sigma –projekti määrittämällä ratkaistava ongelma ja projektin tavoite. Tässä vaiheessa tulisi myös muodostaa korkean tason ymmärrys asiakkaan nykyisestä prosessista, sekä dokumentoida se. Kun projektin alustavat tekijät ovat selviä, projektiryhmän tulisi suorittaa soveltuvuustutkimus, jossa tutkitaan ovatko projektiryhmän lähestymistapa, sekä Six Sigman käyttö soveltuvia projektille. Jos projekti on toteutuskelpoinen, sille haetaan hyväksyntä. Six Sigma –projekti hyväksytään, jos jokainen projektissa mukana oleva osapuoli hyväksyy projektin. Tällä on tarkoitus vahvistaa osapuolien yhteistyötä sekä yhteisvastuuta projektin onnistumisesta. Tällä tavoin kaikki osapuolet ovat myös tietoisia projektin sisällöstä, eikä yllätyksiä pääse syntymään.

Six Sigman kannalta järjestelmäanalyysi alkaa prosessikartoituksella, jossa asiakkaan prosessi tutkitaan mahdollisimman tarkasti. Yleensä joku asiakkaan edustaja tarjoaa oman näkemyksensä heidän prosessista, mutta projektiryhmän tulee itse tutkia se. Voi olla mahdollista, että asiakkaan edustaja ei ole syvällisesti perillä prosessista, tai prosessi voi todellisuudessa olla toisenlainen, kuin mitä prosessikartoituksesta voisi ymmärtää. Prosessikartoituksen jälkeen prosessista yritetään tunnistaa piirteitä, jotka voidaan luokitella virheiksi. Tässä vaiheessa virheet ovat viiveitä, jotka hidastavat asiakkaan prosessia, sekä toimintoja, jotka tuottavat epätasaista tulosta. Alkuvaiheen virheiden tutkimisella vahvistetaan kuvaa ratkaistavasta ongelmasta, sekä kartoitetaan toimintoja, joissa muutokset aiheuttaisivat suurimman kehityksen. Lisäksi alkuvaiheessa asiakkaan prosessista voidaan tunnistaa aliprosesseja, joita muokkaamalla voidaan ehkäistä havaittuja virheitä mahdollisesti paremmin kuin tuottamalla tietojärjestelmään muutos.

Järjestelmäanalyysin tarkoituksena on muodostaa vaatimusjoukko, joka määrittää toteutettavan järjestelmän. Ohjelmistokehityksessä on yleisesti ymmärretty vaatimustenhallinnan ja –määrittelyn tärkeys projektin onnistumisen kannalta. Myös Tayntorin mallin onnistuminen perustuu vaatimusten täsmällisyyteen ja oikeellisuuteen. Tosin, Six Sigma ei näyttäisi tuovan tähän vaiheeseen mitään, mikä huomattavasti eroaisi työkaluista, joita käytetään muutenkin ohjelmistokehityksen vaatimusmäärittelyssä. Mallissa painotetaan vaatimusten tunnistamista, vaatimusten priorisointia, sekä nykyisen prosessin täydellistä ymmärtämistä. Six Sigma –projektin ongelman ratkaisu jatkuu aktiivisesti järjestelmäanalyysissa. Vaatimusten pohjalta yritetään tunnistaa potentiaalisia kehityskohteita nykyisestä prosessista. Kehityskohteiden tunnistamiseen ja vaikutuksen arvioimiseen voi hyödyntää esimerkiksi Quality Function Deployment- [Chan and Wu, 2002] ja Critical To Quality –analyyseja [Bond and Fink, 2001], joilla voidaan perustella valinnat. Tässä vaiheessa ratkaisuiksi kehityskohteisiin käyvät myös muut keinot kuin uuden ohjelmiston luominen. Kehityskohteiden muutosten kautta luodaan tulevan prosessin prosessikartta ja riskiarviot muutoksien vaikutuksista. Uudelle prosessille määritetään mittarit, joiden avulla tullaan valvomaan prosessin toimintakykyä.

Tayntor jakaa järjestelmäsuunnitteluvaiheen kolmeen osaan: toiminnallisuuden määrittelyyn, tekniseen suunnitteluun ja komponenttisuunnitteluun. Six Sigma ei tuo mitään uutta toiminnalliseen määrittelyyn. Teknisessä suunnittelussa käsitellään virheiden ehkäisyä katselmoinnin muodossa. Tarkoituksena on katselmoida järjestelmäsuunnittelun tulokset kaikkien projektin osapuolien kanssa. Virheiden löytyessä niistä tulee tutkia juurisyyt, jotta virheiltä voidaan välttyä seuraavan kerran. Tässä toimenpiteessä ei ole mitään uutta, sillä katselmointien vaikutus laatuun on tunnistettu ja katselmoinnit voidaan ajatella kuuluvan laadukkaaseen

ohjelmistokehitykseen. Six Sigma ei myöskään vaikuta ohjelmistoarkkitehtuuriin tai komponenttisuunnitteluun.

Ohjelmointivaihe vastaa kehitysvaihetta Six Sigman kannalta. Tämä johtuu siitä, että ohjelmointivaiheessa luodaan tietojärjestelmä, joka kehittää asiakkaan prosessia siten, että Six Sigma –projektin ongelma korjautuu. Tayntor mainitsee, että ohjelmointivaiheessa tehdyt virheet tulisi dokumentoida ja virheiden juurisyyt tulisi analysoida. Tällä voitaisiin kehittää samalla ohjelmointivaihetta. Tayntor samalla mainitsee, ettei Six Sigmassa ole erityisiä työkaluja ohjelmistokehityksen virheiden mittaamiseen.

Testausvaihe on aina ollut laadunhallinnan kannalta oleellinen piste todentaa, että ohjelmisto täyttää toiminnalliset vaatimukset sekä laatuvaatimukset. Tayntorin mallissa ei kuitenkaan esitellä Six Sigmaan liittyviä toimenpiteitä, jotka muuttaisivat yleisiä vesiputousmallin testaustapoja.

Käyttöönoton jälkeen projekti arvioidaan. Arvioinnin tarkoituksena on todeta, kuinka projekti on onnistunut tavoitteissaan. Six Sigma tuo arviointiin mukaan valvontasuunnitelman laatimisen, jonka avulla voidaan valvoa saavutetun kehityksen vakautta. Valvontaan käytetään järjestelmäanalyysin yhteydessä luotuja mittareita.

Tayntorin mallilla on tarkoitus parantaa asiakkaan prosessia, ei ohjelmistotuotantoprosessia. Vaikka mallin eri vaiheissa mainitaan tapoja, joilla voidaan parantaa myös ohjelmistotuotantoprosessia, mikään niistä ei ole uusi metodi ohjelmistokehitykselle. Keskeisin Tayntorin mainitsema metodi virheiden ehkäisylle on katselmointi, joka on huomattu osaksi hyvää laadunhallintaa. Mallissa tuotettiin useita dokumentteja, joissa perusteltiin miksi muutospäätökset oli tehty ja miten niiden tulisi vaikuttaa prosessinkehitykseen. Raskaiden dokumenttien tuottaminen tulisi eristää pois ohjelmistokehitysryhmältä, sillä heidän tulisi keskittyä itse tuotteen suunnitteluun ja toteutukseen niiden vaatimusten pohjalta, jotka perustuvat tehtyihin päätöksiin.

Mallin käyttöönotosta tai tuloksista ei ole tietoa, eikä mallia ole tutkittu empiirisesti. Tayntorin mallia voisi mahdollisesti hyödyntää laatuvaatimusten mittaamisessa sellaisissa tapauksissa, joissa tietojärjestelmän tulee nopeuttaa tai muuten tehostaa toimintaa. Tällöin tehostaminen voidaan valita Six Sigma –projektin ratkaistavaksi ongelmaksi. Ongelmalle voidaan määrittää mittarit, joilla mitataan toiminnan tehokkuus ennen ja jälkeen uuden järjestelmän käyttöönoton. Jos mittarin arvo pysyy toleranssissa, sekä hajonta pienenee ennalta määritetyllä aikavälillä, tietojärjestelmä on täyttänyt sille asetetun laatuvaatimuksen. Voi olla, että jossain vaiheessa järjestelmän toimintaympäristö tai siihen liittyvä prosessi muuttuu, eikä järjestelmä pysy enää toleranssissa. Sama mittari voi myöhemmässä vaiheessa osoittaa, että järjestelmää tulisi muuttaa siten, että se vastaa tehokkaammin muuttuneeseen tilanteeseen. Edellä mainitulla tavalla Six Sigmaa voidaan käyttää laadun validointiin ja sitä kautta ohjelmistokehityksen laadunvarmistukseen.

5.3. Six Sigma ja CMMI

CMMI:n kolme ensimmäistä kypsyystasoa sisältävät lähes kaikki prosessialueet projektinhallinnasta, teknisestä osaamisesta ja tukitoimista. Toisin sanoen, CMMI kypsyystason 3 saavuttanut organisaatio omaa huolellisesti suunnitellun laatujärjestelmän ja vakaan ohjelmistotuotantoprosessin, joka täyttää huomattavasti korkeammat laatuvaatimukset, kuin CMMI:n 1. tai 2. taso pystyisi täyttämään. CMMI:n kypsyystaso 3 vaatiikin yleensä useiden vuosien panostuksen ohjelmistotuotantoprosessin kehittämiseen. CMMI:n kolmannen kypsyystason saavuttaminen ei ole itsestäänselvyys, eivätkä kaikki kypsyystasoa tavoittelevat organisaatiot sitä saavuta.

CMMI:n neljäs kypsyystaso muodostuu prosessialueista, jotka kiinnittävät huomiota organisatoristen prosessien suorituskykyyn ja kvantitatiiviseen projektinhallintaan. Siviyn ja muiden [2008] mukaan neljännellä kypsyystasolla organisaation tulee muodostaa odotukset organisatorisille prosesseille, sekä ymmärtää prosessin suorituskyky mittaamisen kautta. Organisaation tulee ylläpitää tietokantaa, johon tallennetaan suorituskykydata. Tietokannan avulla saadaan aikaiseksi vertailukohta ja malleja suorituskyvystä, joita käytetään organisaatio- ja projektitason kvantitatiivisessa hallinnassa. Lisäksi projektien tulee pystyä käyttämään suorituskykydataa verratakseen omaa suorituskykyä organisaation tai projektien räätälöityihin prosesseihin. Kun organisaatio saa asiantuntemusta mittausdatan ohjaamasta päätöksenteosta, organisaation hallinta ja kulttuuri kokee muutoksen. Prosesseja ohjataan kvantitatiivisen hallinnan tavoitteiden mukaan, jotka ovat yhteydessä projektin ja liiketoiminnan suorituskykyihin.

Vaikkakin Siviyn ja muiden [2008] mukaan Six Sigmaa voi hyödyntää kaikilla CMMI:n kypsyystasoilla, Six Sigman kokonaisvaltainen käyttöönotto on edellytys neljännen kypsyystason saavuttamiseksi. Tosin, neljännen kypsyystason kvantitatiivinen hallintamenetelmä voi olla muukin kuin Six Sigma. Six Sigma vastaa ideologialtaan ja työkaluiltaan neljännen kypsyystason vaatimuksiin. Ne muistuttavat toisiaan jopa niin, että yleisenä uskomuksena on ollut, että Six Sigman käyttöönotto vastaisi CMMI:n neljättä kypsyystasoa [Siviy et al., 2008]. Näinhän ei tietenkään ole, koska Six Sigman käyttöönotto ei takaa, että kolmen ensimmäisen kypsyystason vaatimukset tulee täytettyä. Siviy ja muut [2008] ovat tehneet havaintoja ohjelmistoalan organisaatioista, jotka ovat ottaneet käyttöön CMMI:n sekä Six Sigman ja heidän mielestä CMMI ja Six Sigma toimivat synergiassa ja mahdollistavat ohjelmistolaadun kehittämisen. Antony ja Fergusson [2004] tekivät samankaltaisen huomion Six Sigman soveltamisesta CMM-malliin. Seuraavassa listassa esitellään Siviyn ja muiden [2008] havaintoja Six Sigman ja CMMI:n käyttöönotosta:

- Organisaatiot, jotka huomaavat hyötyvän Six Sigman käyttöönotosta, hyödyntävät kaikkia Six Sigman työkaluja, menetelmiä ja filosofiaa.
- Hyvä päätöksenteko korreloi Six Sigman käytön kanssa organisaatioissa.
- Six Sigma tarjoaa yhdistävän kielen, jonka kautta ylempi johto, jotka eivät tunne ohjelmistokehitystä ja ohjelmistokehittäjät ymmärtävät tapoja päästä todellisiin liiketoimintatavoitteisiin.
- Six Sigma mahdollistaa CMMI:n käyttöönoton sekä siirtymisen CMM- mallista CMMI-malliin.
- Six Sigma nopeuttaa CMMI:n käyttöönottoa.
- Six Sigma mahdollistaa CMMI:n käytön tavalla, jolla sitä on tarkoitettu käytettävän, sillä Six Sigma vaatii organisaatiota yhdistämään kaiken toiminnan liiketoimintaan ja asiakkaisiin.
- Six Sigman työkalujen ja menetelmien kautta saadut tulokset antavat kuvan siitä, mitä tarvitaan CMMI:n korkeaan kypsyyteen.
- CMMI:hin pohjautuvat organisaatioresurssit mahdollistavat Six Sigma – projekteihin pohjautuvan opitun tiedon jakamisen ohjelmistokehitysorganisaatioiden kesken ja tätä kautta mahdollistaa Six Sigman institutionalisoinnin.

CMMI-kypsyysmallilla on yhteys prosessinkehitykseen ja sitä kautta kehittyvään ohjelmistolaatuun. Six Sigman huomattavin synergiaetu CMMI:n kanssa on kvantitatiivinen hallinta, joka tulee keskeiseksi viimeistään CMMI:n neljännellä kypsyystasolla. Suurin osa ohjelmistokehityksen yleisistä laatuongelmista kuitenkin liittyy CMMI:n kolmeen ensimmäiseen kypsyystasoon. Watson [2002], sekä Murugappan ja muut [2003] huomioivat CMMI:n esimallista CMM:stä, että Six Sigman synergiaedut tulevat esille vasta kypsillä ohjelmistoyrityksillä, joiden ohjelmistokehitysprosessi on keskivertoa vakaampi ja tuottaa jo valmiiksi hyvää ohjelmistolaatua.

5.4. DFSS-toimintamallin soveltaminen ohjelmistoprojektiin

Ohjelmistokehityksessä on yleisesti tiedossa, että mitä aikaisemmassa vaiheessa ohjelmistotuotantoprosessia virheet tunnistetaan, sitä vähemmän niiden korjaus tulee maksamaan. Varsinkin vaatimustenhallinnalla on suuri vaikutus alkuvaiheessa tehtävien virheiden tunnistamiseen. DFSS-toimintamallin soveltaminen suoraan ohjelmistokehitykseen voi olla hankalaa, sillä osa asiakasvaatimuksista voi muuttua ohjelmistotuotantoprosessin aikana, tai osa vaatimuksista tulee esille myöhemmissä vaiheissa. Ketteriä menetelmiä käytettäessä asiakasvaatimukset vakiintuvat kehitysiteraatioiden mukaan. Voidaan siis olettaa, että kaikkia asiakasvaatimuksia ei todennäköisesti voida ottaa huomioon DFSS-toimintamallia käytettäessä

ohjelmistokehitykseen. Jos kuitenkin osa määrittelyn aikana esille tulleista asiakasvaatimuksista on erityisen keskeisiä tuotantoprosessille, niin DFSS-analyysin voisi mahdollisesti tehdä keskeisimmillä asiakasvaatimuksilla. Analyysi voisi esimerkiksi vaikuttaa kehitysmallin, kuten vesiputousmallin tai sopivan ketterän mallin valintaan.

5.5. Six Sigma ja ketterät menetelmät

Redzic ja Baik [2006] sovelsivat tutkimuksissaan Six Sigmaa vesiputousmalliin. Kuitenkin vesiputousmalli on todettu ongelmalliseksi nykyajan ohjelmistokehityksessä, jossa on reagoitava nopeasti muutoksiin. Ketterät menetelmät ovat vaikuttaneet nykyajan ohjelmistokehitykseen huomattavasti, mutta tutkijat eivät ole suuremmin esittäneet Six Sigman soveltamista ketteriin menetelmiin. Tämä on merkillistä, sillä vesiputousmalli ei yleensä ole paras vaihtoehto projekteihin, joissa asiakas halutaan pitää lähellä. Toisin sanoen, Six Sigma tuskin yleistyy keskeisesti ohjelmistokehityksessä, jos sitä on vaikeata soveltaa ketterissä menetelmissä. Todennäköisesti Six Sigman tarkka prosessin määrittäminen ja ilmeinen orgaanisen toiminnan kääntäminen mekaaniseen ovat pitäneet Six Sigman sivussa ketteristä menetelmistä.

Ketterissä menetelmissäkin on piirteitä, joihin voisi soveltaa joitain osa-alueita Six Sigmasta. Hyvänä esimerkkinä on Scrum, jossa ohjelmistokehitys perustuu lyhyihin iteraatioihin, joiden päättyessä tuotettavasta ohjelmistotuotteesta julkaistaan uusi versio, joka sisältää edellisten iteraatioiden ja viimeisimmän iteraation tulokset. Asiakas on sidottu tarkistamaan ja hyväksymään jokaisen sprintin, eli iteraation tulokset, jolloin asiakas validoi myös laadun ja virheettömyyden. Ero vesiputousmalliin on suuri, sillä Scrumissa asiakas näkee tuotteen ominaisuudet kehitysvaiheessa, eikä vasta hyväksymistestauksen yhteydessä. Ohjelmistokehittäjät saavat myös väliajoin palautteen tehdystä työstä ja tuotetuista ohjelmistovirheistä. Tällainen iteratiivinen kehittäminen voitaisiin nähdä sopivaksi myös Six Sigman kannalta, kun tuotantoyksiköksi määritettäisiin yhden iteraation aikana tuotettu ohjelmistosisältö.

Ketterää menetelmää ei kannata rajoittaa tiukoilla prosesseilla, sillä se voi vaikuttaa nopeaan sopeutumiseen, joka on ketterien menetelmien keskeisin etu. Silti esim. Scrum-kehitysmallista löytyisi tilaa Six Sigmalle ja mahdollisuuksia kehittää ohjelmistolaatua. Scrumiin voisi soveltaa osittain Redzicin ja Baikin [2006] mallin toimintatapoja, missä raportoitujen virheiden juurisyyt ja ohjelmistotuotantovaihe selvitetään. Tällöin jokaisen lyhyen iteraation jälkeen saadaan palaute, missä vaiheessa ja miten ohjelmistovirheet päätyvät ohjelmistotuotteeseen. Juurisyiden avulla Scrum-ryhmä voi kehittää toimintatapoja, joilla juurisyihin voitaisiin vaikuttaa siten, että ohjelmistotuotteeseen päätyisi vähemmän virheitä. Sprinttien ohjelmistolaatua voisi verrata mittarilla, jossa sprintin sisällön arvioitu työmäärä, eli tehtävien pisteytyksen

kokonaismäärä jaettaisiin iteraatiosta löydettyjen virheiden määrällä. Tämä mittari suhteuttaisi virhetiheyden sprintin arvioituun ohjelmistokokoon. Lisäksi, virhetiheyden voi viedä tehtäväkohtaiselle tasolle, jossa voidaan tutkia tehtävän pisteytyksen ja siitä löytyneiden virheiden syitä ja suhteita. Virheiden määrä voi myös ilmaista, että ohjelmistokehittäjät ovat arvioineet sprintin sisällön ohjelmistokoon alakynteen, jolloin he ovat joutuneet toteuttamaan kiireessä luvatut tehtävät. Asiakas ei todennäköisesti kerkeä tutkimaan iteraation tuloksia niin nopeasti, että löydetyt virheet voitaisiin raportoida iteraation vaihtuessa. Virheet voisi raportoida yhden iteraation viiveellä ja ne voitaisiin käsitellä sprintin katselmoinnin jälkeen. Hyödyllisin tapa olisi katselmoida ohjelmointivirheet ja yhdessä määrittää, sekä dokumentoida syy-seuraussuhteet virheiden syntymiselle. Virheet, jotka kuuluvat aikaisempien iteraatioiden tuloksiin, voidaan jäljittää ja merkata oikean toteutusiteraation virheeksi. Virheiden suhde sprintin kokoarvion yksikköä kohtaan on mittarina verrannollinen eri Scrum-ryhmien välillä.

Scrum-projektin seurantaa varten voisi myös kehittää mittarin, jolla mitataan toteutusnopeuden vakautta. Scrum-projektin velocity-arvon, eli sprintin aikana valmiiksi saatujen arviopisteiden määrän, pitäisi vakiintua, mitä pidemmälle projekti etenee. Velocity-arvoa voisi alkaa tarkkailemaan esim. ensimmäisen kolmen sprintin keskiarvon perusteella. Mittariin voisi liittää alemman ja ylemmän määrittelyrajan siten, että sprint onnistuu mittarin osalta, jos sen velocity ei poikkea tiettyä prosenttimäärää aiemmin määritellystä kolmen sprintin keskiarvosta. Koska Scrum-projektin jäsenet kehittyvät projektikohtaisissa asioissa, mittarin vertailukohtaa voisi muokata projektin edetessä siten, että vertailukohta määritettäisiin aina viimeisimmän kolmen sprintin keskiarvon perusteella. Jos koko projektille on arvioitu kokonaispistemäärä, projektin valmistuminen voidaan arvioida velocity-arvosta. Jos velocity pysyy suhteellisen vakaana, kokonaispistemäärään katsoen on varmempaa, että projekti valmistuu ajallaan. Jos velocity heittelee paljon, siihen on syytä löytää juurisyyt, sillä ennustettavuus katoaa ja todennäköisesti projektissa on kohdattu odottamattomia ongelmia.

Ohjelmistoprojektit ovat luonteeltaan erilaisia, jonka takia myös eri projekteissa voi olla erilaiset ongelmat. Lisäksi projektien vaihtuessa projektiryhmät muuttuvat. Tässä kappaleessa kuvatun soveltamisen eduksi voidaan laskea se, että projekteissa havaitut kehitystoimenpiteet siirtyvät kehittäjien mukana seuraaviin projekteihin. Lisäksi, projekti ei ole riippuvainen edeltävistä projekteista, sillä se voi sopeutua omaan yksilölliseen projektiin iteraatioiden aikana koettujen havaintojen kautta, jolloin ketteryys säilyy.

Tässä kappaleessa esitellyssä sovellustavassa tosin sivuutetaan Six Sigman keskeiset mittarit ja toimintamallit. Voidaan myös kyseenalaistaa, että onko toimintatavassa sovellettu Six Sigmaa laisinkaan. Toisaalta, yhtälailla voidaan myös kyseenalaistaa DPMO:n, sigma-tason ja prosessikykyindeksien arvokkuus ketterässä ohjelmistokehityksessä.

5.6. Six Sigma ja automatisoitu yksikkötestaus

Perinteinen testaus on hankala kohde Six Sigmalle, koska olemassa olevien virheiden kokonaismäärää ei voida tietää ja täten testauksen todellinen tehokkuus ja onnistuminen jäävät arvioiksi. Yksikkötestaus sen sijaan on sopiva mittauskohde Six Sigmalle, sillä testi joko läpäistään hyväksytysti, tai ei. Lisäksi testiajon voi toistaa automaattisesti ja nopeasti, mikä mahdollistaa päivittäiset ajot. Päivittäiset ajot taas luovat dataa ohjelmiston eheydestä ja itse testauksesta koko kehitysvaiheen ajan. Tämä tarjoaa dataa erilaisille Six Sigma –mittareille, jotka kertovat ohjelmistotuotantoprosessin ja tuotettavan ohjelman laadusta.

Selkein mittari yksikkötestaukselle on testien läpäisy, esimerkiksi siten, että jokaisen yksittäisen testin ajo on tuotantoyksikkö ja testin läpäisy määrittelee virheellisyyden. Tällainen mittari kertoo siitä, kuinka usein ohjelmistoon ohjelmoitu logiikka täyttää testille määritetyt ehdot. Yleensä virhe yksikkötestauksessa osoittaa sen, että jossain komponentissa tapahtuneen koodimuutoksen vaikutuksia muihin komponentteihin ei ole huomioitu, tai sitten logiikka on muuttunut, eikä testiä ole muistettu päivittää uuden toteutuksen mukaiseksi.

Kun testit menevät läpi, ohjelmisto toimii oikein, mutta vain niiltä osin, mihin yksikkötestit on kirjoitettu. Testikattavuuden säilyttäminen onkin tärkeä osa yksikkötestausta ohjelmistoprojektin aikana. Tähän voisi soveltaa Six Sigma –mittaria siten, että joka päivä, kun koodirivien testikattavuus laskee alle määritetyn prosenttiosuuden ohjelmiston koodirivimäärästä, mittaustulos osoittaa virheen. Tämä mittari osoittaa, kuinka hyvin ohjelmoijat pystyvät ylläpitämään vaadittua testikattavuutta koko projektin ajan. Ylempi johto voi asettaa esimerkiksi bonusehdon projektin yksikkötestauksen sigma-arvolle, joka motivoi ohjelmoijia kirjoittamaan kattavat yksikkötestit. Mittari ei tosin takaa sitä, että yksikkötestit testaavat ohjelman logiikan oikein tai sitä, että yksikkötestit ovat yksikkötestauksen kannalta hyviä testejä. Nämä seikat riippuvat ohjelmoijan taidoista ja yksikkötestauskokemuksesta. Joka tapauksessa, jos testikattavuus pysyy hyvällä tasolla, moni koodimuutoksista johtuva virhe saadaan kiinni ennen kuin se päätyy tuotantoon. Virheiden väheneminen on taas ohjelmistotuotantoprosessin ja ohjelman laadun parantumista.

Automaattiset rasitustestit testaavat järjestelmän suorituskykyä, jota yleensä mitataan vasteajassa, suoritusajassa tai muistinkulutuksessa. Jokainen edellä mainituista määreistä sopii Six Sigma –mittariksi, koska sekä suoritusajalle että muistinkäytölle voidaan määrittää asiakasvaatimuksesta ylempi määrittelyraja. Esimerkiksi verkkosovellusta voidaan testata erilaisilla käyttäjämäärillä ja laskea sigma-tasot pyyntöjen vasteajoista. Jos Sigma-taso tippuu liian alas, jokin vaikuttaa heikentävästi järjestelmän suorituskykyyn. Mittarin voi myös asettaa hieman tiukemmalle, jolloin Sigma-taso varoittaa suorituskyvyn heikentymisestä ennen vaaditun suorituskykytason

menettämistä. Tällä tavoin korjaustoimenpiteitä voidaan suunnitella ja toteuttaa ennen kuin tilanne on huomioitavissa käyttäjäkokemuksissa tai järjestelmän vakaudessa.

5.7. Johtopäätöksiä Six Sigman soveltamisesta ohjelmistokehitykseen

Tässä luvussa on esitelty sovellustapoja, joissa Six Sigman avulla on yritetty parantaa ohjelmistotuotantoprosessia siten, että se tuottaa parempaa ohjelmistolaatua. Kuten viitatuista tutkimuksista ilmenee, Six Sigmaa voi soveltaa ohjelmistokehitykseen eri tavoin. Tosin metodologian mukaan ottaminen ei itsessään kehitä ohjelmistolaatua, vaan kehitys tulee ohjelmistokehittäjien tavasta käyttää Six Sigma –työkaluja virheiden tunnistamiseen, sekä virheiden juurisyiden selvittämiseen. Six Sigman mukaan ottaminen siis pakottaa ohjelmistokehittäjät pohtimaan, miksi, sekä missä virheitä syntyy ja miten niitä voisi välttää. Voi olla, että ohjelmistoyritykset eivät varaa aikaa virheiden analysoinnille tai ohjelmistoyritykset eivät omaa toimivaa toimintatapaa virheiden analysointiin, jolloin Six Sigman soveltaminen paikkaa puutteita ja voi vaikuttaa ohjelmistolaadun kehittämiseen positiivisesti.

Park ja muut [2006] osoittivat useita Six Sigma –työkalujen käyttötapoja ohjelmistokehityksessä. Käyttötavat osoittivat, että keskeisin mittauksen kohde olisi raportoidut ohjelmistoviat. Tämä ei ole yllättävää, sillä raportoidut ohjelmistoviat ovat konkreettisia tuloksia ohjelmistolaadusta ja vika on keskeinen käsite Six Sigmassa. Toinen huomattava mittauksen kohde on arvioiden tarkkuus. Arvioiden ja virheiden mittaaminen voi auttaa kehittämään ohjelmistolaatua koko ohjelmistotuotantoprosessin osalta. Jos arviot tarvittavista resursseista ja ohjelmiston kompleksisuudesta kehittyvät, toteutusvaiheeseen ja testaukseen varataan tarpeeksi aikaa ja osaamista, jolloin projektin valmistuminen aikataulussa on todennäköisempää. Ohjelmistoviat taas kertovat ohjelmistotuotantoprosessin kunnosta, sillä virheitä syntyy kaikissa tuotantovaiheissa ja niiden syntyperä voidaan jäljittää mm. kausaliteettianalyysilla. Tosin virheiden alkuperien jäljittäminen vaatii resursseja ja täten ylemmän johdon hyväksyntää ja vankkumatonta tukea. Ylemmän johdon hyväksyntää auttaisi Six Sigman rinnakkainen organisaatiorakenne, jossa Champion-rooli varmistaisi, että virheanalyyseja tuetaan.

Toki Six Sigmaa voi hyödyntää muihinkin mittauskohteisiin sekä perinteisiin ohjelmistotuotannon mittareihin, mutta tällöin kannattaa huomioida, että sigma-taso kertoo hajonnasta ja että mittauksia tulisi tehdä pitkällä aikavälillä. Sigma-tason tulisi olla arvokas tieto projektipäällikölle ja johdolle verrattuna esimerkiksi prosenttiosuuksien arvoon.

Six Sigman käyttöä ei ole määritelty tiukasti, minkä takia organisaatiot voivat soveltaa sitä omien toimintatapojen ja tarpeiden mukaan. Organisaatioiden toimintatavat eivät ole julkista tietoa, sillä ne saattavat liittyä kilpailukykyyn. Todennäköisesti näiden kahden syyn takia Six Sigmasta ei ole dokumentoitua de facto

–toimintatapaa ainakaan ohjelmistokehitykseen. Six Sigman käyttöä ei valvo tai auditoi ulkoinen organisaatio, joka vahvistaisi Six Sigman käytön kattavuuden tai soveltuvuuden tuotantoprosessiin. Jos Six Sigmaa hyödynnetään osana CMMI:tä, Six Sigma voi joutua validointiin kypsyystasoa arvioitaessa.

SEI:n mukaan CMMI-kypsyysmallin ja Six Sigman välillä on synergiaa. Edellisen perusteella CMMI-kypsyysmallia hyödyntävien kannattaisi integroida Six Sigma ja päinvastoin. CMMI:n ja Six Sigman soveltaminen ei ole pieni asia, vaan vaatii organisaatiolta panostusta ja voimavaroja. Voi siis olla, että pienempien organisaatioiden on resurssien valossa vaikeampaa ryhtyä soveltamaan molempia malleja. Pelkästään Six Sigman Black Belt –roolissa toimivat henkilöt käyttävät kaiken aikansa Six Sigma –projektien valvomiseen. Voi siis olla, että pelkästään Six Sigman soveltaminen perinteisellä tavalla on liian suuri resurssivaraus pienille organisaatioille, jotka saattavat tämän takia hylätä metodologian. Tosin mittareiden tilastollinen analysointi onnistuu pienemmilläkin resursseilla, mutta jos työkaluja sovelletaan poikkeavin menetelmin, saavutetaanko tällöin Six Sigman hyötyjä ja onko silloin enää kyseessä Six Sigma?

Six Sigma vaikuttaa valmiilta paketilta, joka oikeissa käsissä kehittää prosesseja kohti täydellisyyttä. Six Sigma ei kuitenkaan ole hopealuoti parempaan ohjelmistolaatuun. Six Sigma ei ratkaise syitä, joiden takia ohjelmistoviat syntyvät tai ohjelmistoprojektit epäonnistuvat. Six Sigma voi kuitenkin ajaa organisaatiota tilaan, jossa se keksii ratkaisuja organisaatiokohtaisiin ongelmiin, jotka ovat juurisyitä huonoon ohjelmistolaatuun.

6. Yhteenveto

Six Sigma –metodologia on ollut perinteisessä teollisuudessa 80-luvulta asti osana organisaatioiden laadun- ja prosessinhallintaa. Myös osa ohjelmistoyrityksistä on ottanut Six Sigman osaksi organisaationsa toimintaa ja samalla osaksi ohjelmistotuotantoprosessia.

Vaikka Six Sigmasta on kirjoitettu paljon oppaita ja dokumentoitu kokemuksia, siitä ei ole tehty kattavaa empiiristä tutkimusta. Tämän tutkielman tehtävänä oli selvittää, miten Six Sigmaa käytetään ja miten se soveltuu ohjelmistokehitykseen.

Six Sigman keskeisin piirre on tarjota toimintatapa, jonka avulla prosessista voidaan mitata syntyneiden virheiden määrää sekä selvittää virheiden juurisyyt. Näiden tietojen avulla prosessia voi muokata kohti tilaa, jolloin prosessin hajonta pienenee ja prosessi tuottaa vähemmän virheitä.

Perinteisessä teollisuudessa virheitä mitataan saman tuotantolinjan tuotantoyksiköistä, joiden hyväksymiskriteerit ovat samat ja ne yleensä liittyvät tuotantoyksikön fyysisiin piirteisiin. Ohjelmistokehityksessä vastaavan mittauskohteen määrittäminen on hankalaa, sillä ohjelmakoodilla ei ole fyysisiä ominaisuuksia ja koodiriveihin liittyy eri vaatimuksia, jolloin niillä ei ole yhteistä hyväksymiskriteeriä. Six Sigma ei siis sovellu ohjelmistokehitykseen aivan ongelmitta. Six Sigmaa pitää soveltaa ohjelmistokehitykseen.

Six Sigman soveltamisesta ohjelmistokehitykseen on olemassa hieman tutkimusta. Soveltamisessa keskeisintä on löytää mittauskohde, jonka avulla voi mitata haluttua muutosta ja täten kehittää ohjelmistotuotantoprosessia. Tässä tutkielmassa esiteltiin kaksi erilaista sovellustapaa, jotka perustuvat aikaisempiin julkaisuihin. Ensimmäisessä sovellustavassa mittari perustui asiakkaan raportoimiin virheisiin julkaisun yhteydessä. Toinen sovellustapa on yhdistää Six Sigma ja PSP/TSP-menetelmä ja kehittää ohjelmistotuotantoprosessia yksilöiden ja ryhmien toiminnasta syntyvän datan perusteella. Sovellustapojen lisäksi esiteltiin tapa varmentaa asiakasvaatimusten täyttyminen mittaamalla asiakkaan prosessia ennen järjestelmämuutosta ja mittaamalla prosessin toiminta järjestelmämuutoksen jälkeen.

CMMI-kehys sisältää osa-alueita, jotka vaativat tilastollista analyysia ja prosessien toimintakyvyn mittaamista. Tilastollisen analyysin ja mittaamisen osalta Six Sigma ja CMMI toimivat synergiassa. Sekä CMMI että Six Sigma pyrkivät stabiloimaan tuotantoprosessin ja parantamaan sen tehokkuutta sekä tuotettua laatua. CMMI ja Six Sigma kummatkin vaativat organisaatiolta lisäresursseja kehitystoiminnan kunnolliseen ylläpitämiseen. Tämän takia synergiaetu on todennäköisempää tavoittaa suurissa ohjelmistokehitysorganisaatioissa.

Jotta Six Sigma yleistyisi ohjelmistokehityksessä, sitä tulisi pystyä käyttämään ketterien menetelmien yhteydessä. Six Sigman soveltamisesta ketteriin menetelmiin ei ollut tutkimuksen tekovaiheessa tarpeeksi tutkimustietoa, mutta tässä tutkielmassa

tunnistettiin tapoja hyödyntää Six Sigman työkaluja Scrum-projekteissa. Scrum-projektista löytyi mittareita, joiden seuranta onnistuu Six Sigman kannalta ja ne kertovat prosessin sekä tuotettavan tuotteen laadusta, joten tiedon perusteella laadunkehityksen tulisi olla mahdollista.

Six Sigma ei sovellu suoraan perinteiseen testaukseen, koska Six Sigman pitäisi tietää, kuinka monta virhettä järjestelmässä on. Tässä tutkielmassa esiteltiin Six Sigma –mittareita yksikkötestaukseen, joka on myös yleinen testausstrategia ketterien menetelmien yhteydessä. Six Sigmalla voisi tukea koko kehitysvaiheen kestävää testikattavuuden mittaamista tai yleistä testien läpäisyastetta.

Tässä tutkimuksessa esiteltiin soveltamistapoja, joiden todellinen vaikutus laadun kehittämiseen vaatii lisätutkimusta. Jotta Six Sigman vaikutus ohjelmistoprojektin laadun kehittämisessä esiteltyjen toimintatapojen osalta voitaisiin todentaa, soveltamistapojen vaikutus tulisi todentaa ja dokumentoida kattavassa otoksessa, joka koostuisi eri organisaatioiden ohjelmistoprojekteista.

Viiteluettelo

[Al-Kilidar et al., 2005] Hiyam Al-Kilidar, Karl Cox and Barbara Kitchenham, The use and usefulness of the ISO/IEC 9126 quality standard. *In: Proceedings of International Symposium on Empirical Software Engineering*, 2005, 7.

[Anand et al., 2010] Gopesh Anand, Peter T. Ward and Mohan V. Tatikonda, Role of explicit and tacit knowledge in Six Sigma projects: An empirical examination of differential project success. *Journal of Operations Management*, **28**, 4 (2010), 303-315.

[Antony and Fergusson, 2004] Jiju Antony and Craig Fergusson, Six Sigma in the software industry: results from a pilot study. *Managerial Auditing Journal* **19**, 8 (2004), 1025-1032.

[Binder, 1997] Robert V. Binder, Can a manufacturing quality model work for software? *IEEE Software* **14**, 5 (1997), 101-102, 105.

[Boaden, 1997] Ruth J. Boaden, What is total quality management… and does it matter? *Total Quality Management*, **8,** 4 (1997), 153-172.

[Bond and Fink, 2001] Edward U. Bond III and Ross L. Fink, Meeting the customer satisfaction challenge. *Industrial Management*, **43**, 4 (2001), 26.

[Britannica, 2009] Encyclopædia Britannica, Inc. Available at *http://www.britannica.com/* (11.1.2009).

[Card, 2006] David N. Card, Myths and strategies of defect causal analysis. *In: Proceedings of Pacific Northwest Software Quality Conference*, 2006, .

[Chan and Wu, 2002] Lai-Kow Chan and Ming-Lu Wu, Quality Function Deployment: A comprehensive review of its concepts and methods. *Quality Engineering* **15,** 1 (2002), 23–35.

[Chaneski, 2002] Wayne S. Chaneski, Mapping a path to lean manufacturing. *Modern Machine Shop*, **75**, 5 (2002), 46.

[Chen and Chuang, 2008] Chun-Chih Chen and Ming-Chuen Chuang, Integrating the Kano model into a robust design approach to enhance customer satisfaction with product design. *International Journal of Production Economics*, **114**, 2 (2008), 667-681.

[Cohn, 2009] Mike Cohn, *Succeeding With Agile*. Addison-Wesley Professional, 1st edition, 2009.

[Durkee, 2008] J. Durkee, Just what is 'lean manufacturing' anyway? *Metal Finishing*, **106**, 12 (2008), 44-46.

[Duvall et al., 2007] Paul Duvall, Steve Matyas and Andrew Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*. 1st edition, Addison-Wesley Professional, 2007.

[Gamma et al., 1994] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st edition, Addison-Wesley Professional, 1994.

[Goel and Okumoto, 1979] L. A. Goel and K. Okumoto, Time-dependent error-detection method for software reliability and other performance measure. *IEEE Transactions on Reliability*, **R-29** (1979), 206-211.

[Grönroos, 2003] Matti Grönroos, *Johdatus tilastotieteeseen; Kuvailu, mallit ja päättely*. Tammer-Paino Oy, 2003.

[Haikala ja Märijärvi, 2003] Ilkka Haikala ja Jukka Märijärvi, *Ohjelmistotuotanto*. 9. painos, Talentum Media Oy, 2003.

[Hines et al., 2004] Peter Hines, Matthias Holweg and Nick Rich, Learning to evolve: A review of contemporary lean thinking. *International Journal of Operations & Production Management*, **24**, 10 (2004), 994-1011.

[Hudson, 2010] Hudson continuous integration. Available at: *http://hudson-ci.org/* (14.10.2010).

[Hughes and Cotterell, 2006] Bob Hughes and Mike Cotterell, *Software Project Management*. McGraw-Hill education, 2006, 4th edition.

[Humphrey, 2000a] Watts S. Humphrey, The Personal Software Process. Technical Report CMU/SEI-2000-TR-022, 2000.

[Humphrey, 2000b] Watts S. Humphrey, The Team Software Process. Technical Report CMU/SEI-2000-TR-023, 2000.

[iSixSigma, 2010] iSixSigma, SIPOC Diagram. Available at *http://www.isixsigma.com/tt/sipoc/* (1.2.2010).

[Keller, 2005] Paul Keller, *Six Sigma Demystified*. McGraw-Hill, 2005.

[Kelley, 2000] D. Lynn Kelley, More new twists on traditional quality tools and techniques. *Journal for Quality & Participation,* **23,** 4 (2000), 30-31.

[McDonald, 2002] James McDonald, Software project management audits – update and experience report. *Journal of Systems and Software*, **64,** 3 (2002), 247.

[Melton, 2005] T. Melton, The Benefits of lean manufacturing. *Chemical Engineering Research and Design* **83**, 6 (2005), 662-673.

[Murugappan and Keeni, 2003] Mala Murugappan and Gargi Keeni, Blending CMM and Six Sigma to meet business goals. *IEEE Software*, **20**, 2 (2003), 42-48.

[Nicolette, 2005] Dave Nicolette, Six Sigma and Agile Development. 5.12.2005. Available at *http://davenicolette.net/articles/six_sigma.html* (15.7.2008).

[Pan et al., 2007] Zhedan Pan, Hyuncheol Park, Jongmoon Baik and Hojin Choi, A Six Sigma Framework for software process improvements and its implementation. *Proceedings of the 14th Asia-Pacific Software Engineering Conference*, 2007, 446-453.

[Park et al., 2006] Youngkuy Park, Hyuncheol Park, Hojin Choi and Jongmoon Baik, A study on the application of Six Sigma tools to PSP/TSP for process improvement. *In: Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06)*, 2006.

[Park et al., 2007] Youngkyu Park, Hojin Choi and Jongmoon Baik, A framework for the use of Six Sigma tools in PSP/TSP. *In: Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007)*, 2007, 807-814.

[Pressman, 2005] Roger S. Pressman, *Software Engineering, A Practitioner's Approach*. 6th edition, McGraw-Hill, 2005.

[Redzic and Baik, 2006] Cvetan Redzik and Jongmoon Baik, Six Sigma approach in software quality improvement. *In: Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*, 2006, 396-406.

[Romanchik, 2004] Dan Romanchik, Six Sigma's not for everyone. *Application Development Trends*, **1** (2004), 16-17.

[Rooney and Hopen, 2005] James J. Rooney and Deborah Hopen, On the trail to a solution. *Journal for Quality & Participation,* **28,** 2 (2005), 15-21.

[Schroeder et al., 2008] Roger G. Schroeder, Kevin Linderman, Charles Liedtke and Adrian S. Choo, Six Sigma: Definition and underlying theory. *Journal of Operations Management* **26**, 4 (2008), 536-554.

[Schwaber, 1995] Ken Schwaber, SCRUM development process. *OOPSLA'95, In: Proceedings of the Tenth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, 1995.

[SEI, 2002] Software Engineering Institute. The internal consistency of key process areas in the capability maturity model (CMM) for software (SW-CMM). 2002. Available at *http://www.sei.cmu.edu/reports/02tr037.pdf* (20.1.2010).

[SEI, 2004] Software Engineering Institute, Upgrading from SW-CMM to CMMI. 25.2.2004. Available at *http://www.sei.cmu.edu/library/assets/upgrading.pdf* (20.1.2010).

[SEI, 2009] Software Engineering Institute, Overview of team software process and personal software process. 3.3.2009. Available at *http://www.sei.cmu.edu/tsp* (20.4.2009).

[Sheriff and Georgiadou, 2007] Mohamed A Sheriff and Elli Georgiadou, Scenarios of software value: Putting software quality in context. *In: Proceedings of the Fifteenth International Conference Proceedings on Software Quality Management* (SQM 2007), 133-145, 2007.

[Shin et al., 2007] Hyunil Shin, Ho-Jin Choi and Jongmoon Baik. Jasmine: a PSP supporting tool. *In: Proceedings of the international conference on Software process*, 2007, 73-83.

[Siviy et al., 2008] Jeannine M. Siviy, M. Lynn Penn and Robert W. Stoddard, *CMMI and Six Sigma*. Pearson Education Inc., 2008.

[SixSigmaOnline, 2010] sixsigmaonline.org. Available at http:// www.sixsigmaonline.org/six-sigma-yellow-belt-training (10.10.2010).

[SixSigmaCompanies, 2010] sixsigmacompanies.com. Available at www.sixsigmacompanies.com (14.5.2010).

[Stewart, 2010] Theodor J. Stewart, Goal directed benchmarking for organizational efficiency. *Omega*, **38**, 6 (2010), 534-539.

[Symphony, 2010] Project Symphony. Available at http://www.projectsymphony.com (4.10.2010).

[Tayntor, 2007] Christine B. Tayntor, *Six Sigma Software Development*. Taylor & Francis Group, 2007.

[Watson, 2002] Gregory H. Watson, Breakthrough in Delivering Software Quality: Capability Model and Six Sigma. *Lecture Notes in Computer Science: Software Quality – ECSQ* 2002. Springer Berlin / Heidelberg, 36-41, 2006.

[Zhao et al.,2008] Xiasong Zhao, Zhen He, Fangfang Gui and Shenqing Zhang, Research on the application of Six Sigma in software process improvement. *In: Proceedings of the 2008 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 937-940, 2008.

[Zu et al., 2008] Xingxing Zu, Lawrence D. Fredendall and Thomas J. Douglas, The evolving theory of quality management: The role of Six Sigma. *Journal of Operations Management* **26**, 5 (2008), 630-650.