OHJELMISTOTUOTTEEN LAADUN MITTAAMINEN

Ohjelmistotuotteen laadun mittaaminen ja seuraaminen yrityksen ohjelmistotuotantoprosessin aikana

Sami Riekkinen

Opinnäytetyö

Ylempi ammattikorkeakoulututkinto

Savonia
ammattikorkeakoulu

SAVONIA-AMMATTIKORKEAKOULU **OPINNÄYTETYÖ**
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala			
Koulutusohjelma Hyvinvointiteknologian koulutusohjelma			
Työn tekijä(t) Sami Riekkinen			
Työn nimi Ohjelmistotuotteen laadun mittaaminen.			
Päiväys	23.8.2011	Sivumäärä/Liitteet	42
Ohjaaja(t) lehtori Sami Lahti			
Toimeksiantaja/Yhteistyökumppani(t) laatuvastaava Ari Voutilainen / Procomp Solutions Oy			

Tiivistelmä

Tämän opinnäytetyön aiheena oli perehtyä ohjelmistojen mittaamiseen teoriaan ja toteuttaa laadun demomittari yrityksen käyttöön. Opinnäytetyössä pohditaan, minkälaisia laatumittareita yrityksen käyttöön voisi toteuttaa ja asettavatko yrityksen sisäiset prosessit esteitä näiden mittareiden toteuttamiselle.

Demomittari toteutettiin Java-ohjelmointikielellä ja kehitysympäristönä käytettiin NetBeans-välinettä. Mittaustietokantana käytettiin testiprojektia, joten yrityksen oikeisiin tuotteisiin ei viitata tässä työssä missään muodossa.

Opinnäytetyön tekemisen yhteydessä kävi ilmi, että yrityksen sisäisiä prosesseja tulee kehittää myös siten, että ne tukevat paremmin mittaamista. Todettiin myös, että muutosvaatimusten hallintavälineen vapaudet estävät omalta osaltaan luotettavien mittaustulosten saantia.

Työ osoittaa, että laadun mittaaminen on tärkeä osa yrityksen tuotekehitysprosessia. Mittaamisesta saatavalla tiedolla tuotetta voidaan hallita paremmin. Automaattinen laadun mittaaminen vapauttaa resurssit tulosten analysointiin.

Avainsanat
ohjelmistoprosessi, mittaaminen, ohjelmistotuote, laatumittari

SAVONIA UNIVERSITY OF APPLIED SCIENCES **THESIS**
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme of Health and Welfare Technology			
Author(s) Sami Riekkinen			
Title of Thesis Measuring Quality of Software.			
Date	23 August 2011	Pages/Appendices	42
Supervisor(s) Mr Sami Lahti, Lecturer			
Project/Partners Mr Ari Voutilainen, Quality Controller / Procomp Solutions Oy			

Abstract

The aim of this thesis was to research how to measure the quality of software and develop a quality measuring software demo for a company.

Measuring demo software was developed using the Java programming language and the used computing environment was NetBeans. Software uses a test database, so no references to the company's products or data are presented in this thesis.

The findings of this research showed that the company should improve their internal processes so that measuring is easier to accomplish. Also features of the change request management tool prevent receiving reliable measuring results.

The project showed that quality measuring has an important place in the company's development process. Information which is collected from the measuring process helps to improve products. Automatic quality measuring releases resources to result analyzing.

Keywords
software developing process, quality measuring, software, software quality

SISÄLTÖ

1 JOHDANTO

Ohjelmistotuotteen elinkaarta on vaikea ennustaa ja voidaankin ehkä sanoa, että ohjelmistotuote ei ole koskaan valmis. Asiakkaiden tarpeet muuttuvat esim. lainsäädännöllisistä tai käytännöllisistä syistä. Myös ohjelmistotuotteen tekniset ratkaisut tai ympäristöt voivat muuttua. Kaikki nämä aiheuttavat ohjelmistotuotteeseen muutoksia. Toimittaja pyrkii tekemään nämä muutokset ohjelmistotuotteeseen mahdollisimman hyvin ja kehittämään ohjelmistotuotetta laadukkaammaksi. Kehitystyötä tehdään koko ajan myös sisäisen laaduntarkkailun avulla. Tämä tarkoittaa sitä, että ohjelmistoa testataan sisäisesti ja raportoidaan löytyneistä virheistä ja puutteista. Myös nämä löydökset aiheuttavat muutoksia ohjelmistotuotteeseen.

Ohjelmistotuotteen laadun seuraaminen on tärkeää, koska siten pystytään vastaamaan paremmin asiakkaan tarpeisiin. Mahdolliset puutteet tai viat näkyvät ohjelmistotuotteessa heikkona laatuna. Laadun tilaa ohjelmistotuotteessa voi olla vaikea hahmottaa, koska tuote voi sisältää hyvin monia osa-alueita, jotka kuitenkin usein liittyvät toisiinsa jollakin tavoin. Ohjelmistotuotteessa on usein kriittisiä osia, joiden kehittämiseen kuluu paljon resursseja. Nämä kriittiset osat ovat yleensä myös virhealttiita osia, koska niissä on paljon toiminnallisuutta.

Demoprojektin tarkoituksena oli löytää mittari, jolla voidaan kuvata ohjelmistotuotteen tilaa. Mittariin tuleva data saadaan yrityksen käytössä olevasta järjestelmästä, johon kootaan järjestelmällisesti kaikki ohjelmistotuotteeseen kirjatut virheet ja kehityspiirteet. Kehittämisprojektin seurauksena saadaan yrityksen käyttöön demoympäristö, jolla tuotteen tai projektin tilaa voidaan tarkastella. Tuloksia analysoimalla voidaan seurata tuotteen tilan kehittymistä valitulla ajanjaksolla ja näin saadaan parempi yleiskuva tuotteen tilan muutoksista. Kehittämisprojekti on tarkoitus liittää osaksi Procomp Solutions Oy:n laatujärjestelmää.

Lopputyön ensimmäiset luvut käsittelevät laatua yleisellä tasolla. Pohditaan laatukäsitteen merkitystä ja sitä, mitä tarkoittaa laadukas ohjelmistotuote. Keskimmäinen osa työstä keskittyy itse mittaridemon toteutukseen esittelyyn ja tekniseen kuvaukseen. Aivan lopuksi pohditaan työn tuloksia ja mietitään, millaisia muita mittareita olisi hyödyllistä rakentaa yrityksen tuotteiden mittaamiseen.

2 OHJELMISTON LAATU JA SEN MITTAAMINEN

2.1 Procomp ja laatu

Procomp Solutions Oy on ohjelmistotalo, joka on perustettu Oulussa vuonna 1995. Tällä hetkellä yrityksessä työskentelee noin 70 ohjelmistoalan ammattilaista. Yritys toimii Oulussa ja Kuopiossa. Procompilla on omia ohjelmistotuotteita, joista tunnetuimpia ovat Rahti ja Aikajana. Rahti on logistiikan tietojärjestelmä ja Aikajana puolestaan työajanhallinnan ohjelmisto. Procomp toimittaa myös räätälöityjä tietojärjestelmiä. (Procomp Solutions Oy 2011.)

Oletettavasti kaikkien ohjelmistoyritysten lähtökohtaisena tavoitteena on tuottaa laadukkaita ohjelmistoja asiakkaittensa käyttöön, ja tämä pätee myös Procomp Solutions Oy:öön. Yrityksen verkkosivuilla missioksi on kerrottu asiakkaan kilpailukyvyn parantaminen tietotekniikan keinoin. Laatumittareiden avulla yritys voi parantaa tuotteidensa laatua. Yksi keino laatumittareiden löytämiseen on johtaa ne suoraan yrityksen tavoitteista (Lecklin 2006, 39 - 41).

Yrityksen laatukäsikirjassa (Voutilainen 2009) sisäiseksi laatupolitiikan tavoitteeksi on asetettu Procompin kilpailukyvyn ja kannattavuuden parantaminen. Laatukäsikirjassa laadun seuraamisesta sanotaan seuraavasti: Laatutavoitteita ja laatupolitiikkaa arvioidaan säännöllisesti ja päätetään mahdollisista muutoksista. (Voutilainen 2009.)

Laatuun liittyviä panostuksia seurataan, jotta yrityksen toimintaa voidaan kehittää tasapainoisesti. Yksi laadun seuraamisen keino on laatumittarit. Laittamalla mittarit mittaamaan oikeita, kehitettäviä asioita saadaan kerättyä laadun parantamiseen liittyvää tietoa, jonka pohjalta tuotteita ja prosesseja voidaan kehittää laadukkaammiksi.

Procomp uusi yritysilmeensä keväällä 2011. Yrityksen verkkosivuilta löytyy mm. seuraavia toteamuksia: laadukkaita ohjelmistoja, jatkuva kehittyminen ja sustainable value. Sustainable value suomeksi käännettynä tarkoittaa pysyvää arvoa tai kestävää arvoa. Yrityksen verkkosivuilla sitä kuvataan seuraavasti*: "Sustainable Value" -sloganilla haluamme myös viestiä tahtoamme olla asiakkaamme pitkäaikainen kumppani, jolle järjestelmämme tuovat jatkuvaa lisäarvoa ja tehostamalla*

resurssien käyttöä ne edistävät myös kestävää kehitystä". (Procomp Solutions Oy 2011.)

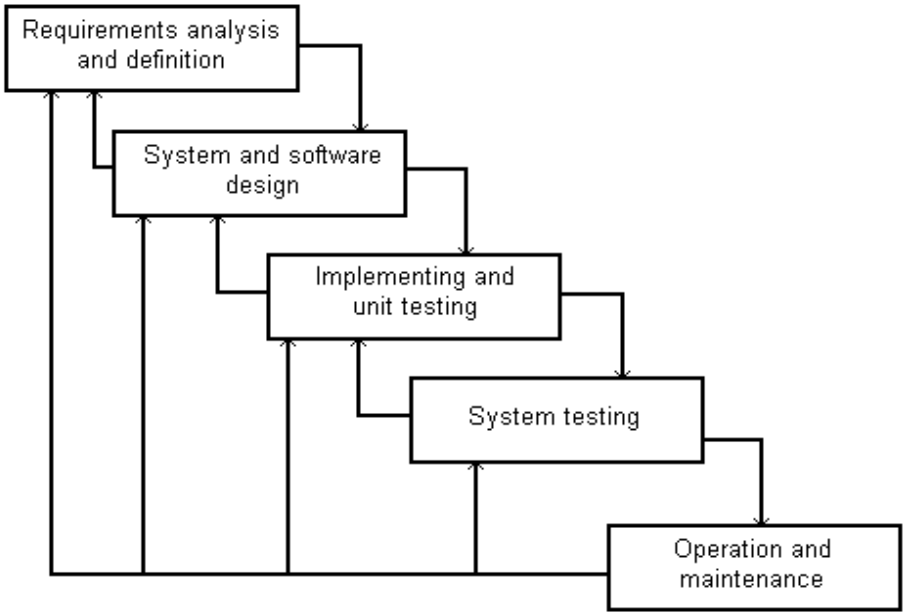
Procompin tavoitteeksi määritelty ”sustainable value” käsitteenä on laaja ja epäselvä, joten sitä ei voi mitata. Käsitteestä voi kuitenkin löytää tarkentavia ja mitattavia asioita, jotka ovat osa tätä käsitettä tai liittyvät tähän läheisesti. Tällaisia voisivat olla esimerkiksi asiakastyytyväisyys ohjelmistotuotteisiin ja asiakassuhteen kesto vuosina. Ohjelmistotuotteen laadun käsitteistä on kerrottu tarkemmin luvussa 2.3.1.

Ohjelmiston teknistä laadukkuutta voidaan mitata siten, että mitataan ohjelmiston sisältämiä virheitä. Jatkuva kehittyminen liittyy prosessien jatkuvaan parantamiseen. Jatkuva lisäarvo ja kestävä kehitys pohjautuvat myös samaan periaatteeseen. Kestävän kehityksen malli on kuvattu tarkemmin luvussa 2.3. Ohjelmiston teknisen laadun käsitteestä ja merkityksestä on kerrottu tarkemmin luvussa 2.3.1.

2.2 Ohjelmistotuotantoprosessi

Tuotettavan ohjelmiston taustalta löytyy aina jonkinasteinen ohjelmistotuotantoprosessi, jonka mukaan ohjelmisto tehdään. Prosessilla tarkoitetaan yleisesti jonkin asian edistymistä. Ohjelmistotuotantoprosessi on yksi prosessin muoto, joka pyrkii siis tuottamaan ennakkoon suunnitellun ohjelmistotuotteen. Sommerville (2001, 43) kuvaa ohjelmistotuotantoprosessia seuraavasti: ”Ohjelmistotuotantoprosessi on sarja toimintoja ja siihen liittyviä tuloksia, jotka johtavat ohjelmistotuotteeseen”. Prosessikuvaus on abstrakti malli siihen, miten ohjelmistoprosessi viedään alusta loppuun. Prosessikuvaus auttaa hahmottamaan, missä vaiheessa prosessia ollaan menossa kullakin hetkellä.

Ohjelmistotuotantoprosessin suorittamisen pohjana on käyttää erilaisia malleja. Perinteisin malleista on lineaarinen malli, josta yleisemmin käytetään nimitystä vesiputousmalli. Sommerville (2001) esittää vesiputousmallin kuvan 1 tavalla.



KUVA 1. Vesiputousmalli (Sommerville 2001, 45)

1) *Requirement analysis and definition*: Vaatimusten määrittelyvaiheessa järjestelmän palvelut ja tavoitteet määritellään. Tästä muodostuu vaatimusmäärittelydokumentti.

2) *System and software design*: Suunnitteluvaiheessa kuvataan järjestelmän riippuvuudet muihin mahdollisiin järjestelmiin ja tuotetaan ohjelmistosuunnitelma, jonka pohjalta ohjelmointityö toteutetaan.

3) *Implementing and unit testing*: Toteutusvaihe sisältää varsinaisen ohjelmointityön. Ohjelmoija suorittaa ohjelmoidessaan samalla yksikkötestauksen, jolla hän varmistaa, että koodi toimii oikein.

4) *System testing*: Integrointi ja testausvaihe pitävät sisällään järjestelmän kokonaisvaltaisen testauksen, jossa testataan kattavasti järjestelmän toimintaa. Tämän vaiheen tukidokumenttina on testaussuunnitelma.

5) *Operation and maintenance*: Tuki- ja ylläpitovaiheessa ohjelmisto on toimitettu loppukäyttäjälle ja on siirrytty vaiheeseen, jossa asiakas käyttää ohjelmistoja ja virheen löytyessä siitä raportoidaan ohjelmiston toimittajalle ja virhe korjataan. Tässä vaiheessa on tärkeä verrata raportoitua virhettä alkuperäiseen vaatimusmäärittelydokumentissa esitettyyn vaatimukseen, onko kysymyksessä oikeasti virhe vai mahdollinen muutos.

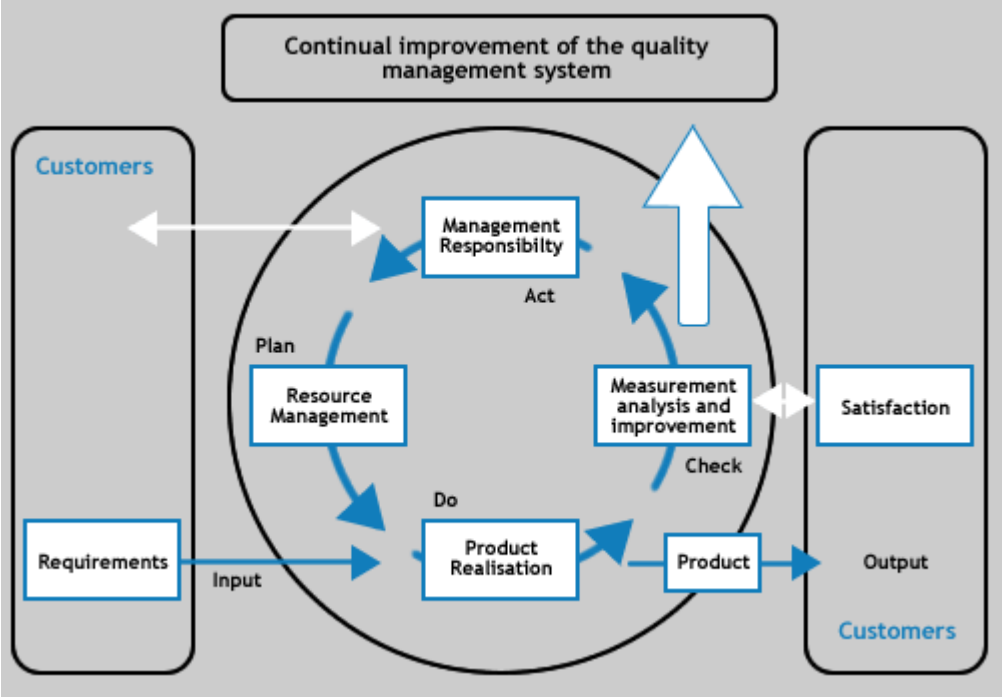
Lineaarisessa mallissa työvaiheet etenevät vaiheittain eteenpäin. Aina edellisen vaiheen valmistuttua siirrytään seuraavaan vaiheeseen. Tosielämässä näin harvoin kuitenkaan tapahtuu, vaan suunnitemaa joudutaan usein muuttamaan vaatimusten tarkentuessa ja muuttuessa.

Nykyään on tarjolla useita muita malleja perinteisen vesiputousmallin rinnalle. Nykypäivänä yksi käytetyimmistä menetelmistä on ns. ketterä ohjelmistokehitys. Ketterän menetelmän erona perinteiseen ohjelmistokehitykseen on se, että siinä tekemistä jaksotetaan lyhyisiin pätkiin, joiden kesto on muutamia viikkoja. Jokaisen jakson päätteeksi on tavoitteena tuottaa julkaisukelpoinen ohjelmisto. Kun jakso on päättynyt, suunnitellaan seuraavan jakson julkaisu etukäteen. Ketterällä ohjelmistokehityksellä pyritään tiiviiseen, ohjelmistolähtöiseen kehitykseen, jossa painotetaan suoraa, kasvotusten tapahtuvaa viestintää dokumentaation kirjoittamisen sijaan. (Agile Manifesti 2001; Agile Manifestin Periaatteet 2001.)

2.3 Laadun kehittäminen ohjelmistoyrityksessä

Jotta ohjelmistoyrityksen tuotteet voisivat pärjätä markkinoilla vuodesta toiseen, sen täytyy vaalia tuotteidensa laatua. Laaduntarkkailuun voidaan käyttää monenlaisia välineitä ja kanavia. Voidaan käyttää mm. kyselyitä, joilla koetetaan selvittää asiakastyytyväisyyttä ja näin parantaa yrityksen toimintaa. Tässä työssä keskitytään siihen, miten ohjelmiston laatua voidaan parantaa laatumittareiden avulla. Laatumittarit antavat konkreettista numerotietoa tuotteen tilasta, ja täten tieto on tilastoitavissa ja tulkittavissa erilaisista grafiikoista.

PDCA (Plan-Do-Check-Act) on ratkaisumalli ja kehittämismenetelmä, joka tunnetaan myös nimellä Demingin laatuympyrä. PDCA:n toiminto perustuu neljään eri vaiheeseen (Plan, Do, Check, Act). Suunnitteluvaiheessa (Plan) määritellään lopputuloksen tuottavat prosessit ja tavoitteet, joihin pyritään. Tekovaiheessa (Do) tavoitteet ja suunnitellut prosessit toteutetaan. Tarkistusvaiheessa (Check) seurataan ja mitataan prosesseja sekä raportoidaan tuloksia. Toimintavaiheessa (Act) ryhdytään toimenpiteisiin, joilla pyritään parantamaan suoritettua prosessia. (Lecklin 2006, 48 - 49.)



KUVA 2. Jatkuvan kehittämisen malli (ISO 2008)

2.3.1 Laadun käsite ohjelmistokehityksessä

Että voisimme miettiä sitä, kuinka muutamme tai voimme muuttaa ohjelmistoa laadukkaampaan suuntaan, on hyvä ensin pohtia, mitä ohjelmiston laatu yleensä tarkoittaa. Laatu on sanana ja käsitteenä epäselvä ja määrittelemätön asia, joten jokaisella ohjelmistoprosessissa työskentelevällä on todennäköisesti erilainen kuva siitä, mitä ohjelmiston laatu tarkoittaa. Käytämme laatu-sanaa päivittäisessä elämässä eri tarkoituksiin, joten sen merkitys on hämärtynyt ja ehkä kulunutkin. Lillrank (1998) on kuvannut laatua siten, että laatu on sitä, mistä itse pitää. Joseph Juran (Juran Instituutin perustaja, 2011), jota pidetään yhtenä laatuajattelun oppi-isänä, on sanonut, että laatu on tuotteen tai palvelun sopivuutta käyttötarkoitukseen. (Sarala U; Sarala A, 1996.)

Ohjelmistosuunnittelija voi mieltää laadun olevan selkeää hyvin jäsenneltyä koodia, jossa ei ole virheitä ja se on lisäksi hyvin kommentoitua. Ohjelmiston testaajan mielestä kattavasti testattu ohjelmisto voi olla laadukas. Myyjä saattaa ajatella, että markkinoilla pärjäävä ja hyvin myyvä ohjelmisto on laadukas. Asiakkaan mielestä usein laadukas ohjelmistotuote on sellainen, joka tekee ennalta määritellyt asiat toivotulla tavalla. Fentonin ja Pfleegerin mukaan ohjelmiston laatu on moni-

tahoinen, eikä sitä voi tarkastella pelkästään yhdestä näkökulmasta (Fenton ym. 1997).

Ohjelmistoprojektissa laatu kirjoitetaan yhteistyössä toimittajan ja asiakkaan kanssa sovelluksen ominaisuuksia määriteltäessä. Jos ohjelmisto täyttää kaikki määritellyt ominaisuudet, niin sen voidaan sanoa olevan määritysten mukaisesti laadukas. Jos ominaisuudet esitellään mitattavina, niin jälkeenpäin voidaan esittää, että asetettu tavoite on täyttynyt. Tätä ajattelua tukee myös Fenton ja Pfleeger kuvatessaan ”tee-se-itse-laatumallin” periaatteita. Mallin mukaan ohjelmisto toimitetaan asteittain tilaajalle. Ohjelmiston tilaaja määrittelee ohjelmiston tärkeysalueet ja avainominaisuudet. Avainominaisuudet kuvataan siten, että niitä voidaan mitata. (Fenton ym. 1997.)

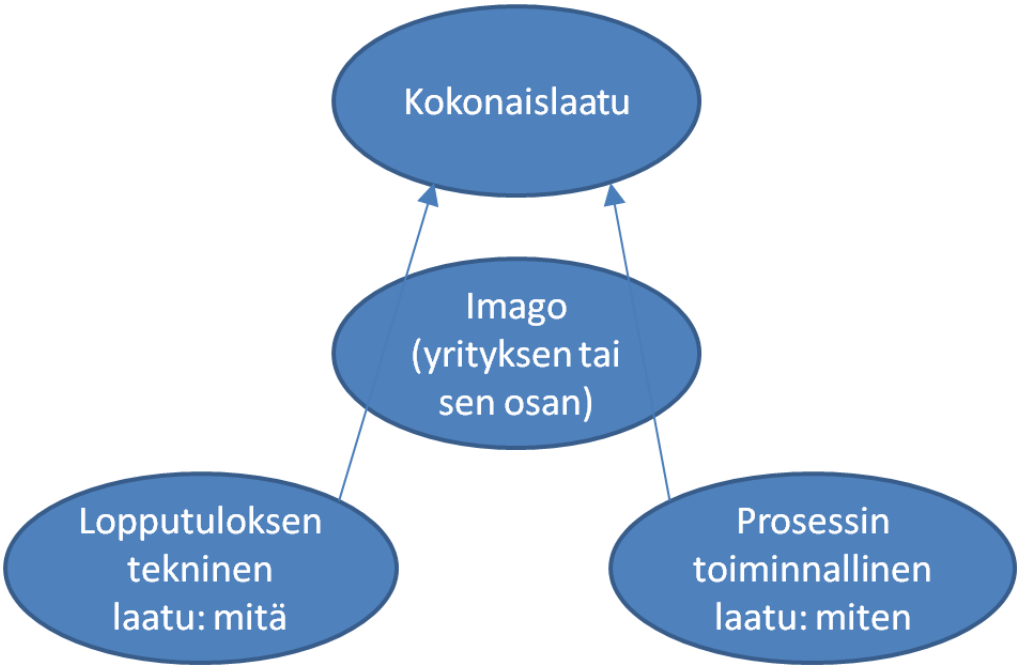
Fentonin ja Pfleegerin mukaan laadukasta ohjelmistoa voidaan kuvata seuraavasti: ohjelmisto sopii käyttötarkoitukseensa, ohjelmisto vastaa määrityksiään, ohjelmisto on luokitukseltaan erinomainen ja ohjelmistolla on pitkä käyttöikä. (Fenton ym. 1997.)

Ohjelmistoyrityksissä laatu mielletään helposti vain osaksi teknistä toteutusta, koska yritykset ovat hyvin tekniikkakeskeisiä. Jos laatua lähdetään parantamaan määrittelemättä tarkemmin, että mitä laatu on ja kuinka sitä voidaan edistää, niin laadun kehittäminen on jo tässä vaiheessa menossa huonoon suuntaan. Kun laatu määritellään liian kapeasti, niin myös laatuhankkeet ovat vaarassa jäädä kapea-alaisiksi. (Grönroos 2009, 100.)

Laatu on kuitenkin myös paljon muutakin kuin pelkkää teknistä toteuttamista. Yrityksen asiakkaat mieltävät laadun paljon laajemmin kuin todellisuudessa ajatellaan. Asiakkaiden laatukokemukset pohjautuvat usein aivan muihin asioihin kuin teknisiin yksityiskohtiin. Laatu on tärkeää sellaisena kuin asiakas sen kokee. (Grönroos 2009, Lecklin 2006, Hölttä ym. 1997.)

Asiakkaiden kokemalla palvelun laadulla on kaksi eri ulottuvuutta (kuva 3), *tekninen eli lopputulosulottuvuus ja toiminnallinen eli prosessiulottuvuus*. Grönroos kuvaa näitä käsitteitä sanoilla ”mitä” ja ”miten”. Teknisellä ulottuvuudella tarkoitetaan konkreettista asiaa, jonka asiakas saa, kun vuorovaikutustilanne asiakkaan välillä on ohi. Tällöin puhutaan lopputuloksen teknisestä laadusta. Toiminnallisella

laadulla tarkoitetaan laatuasioita, jotka tapahtuvat sillä hetkellä kun vuorovaikutustilanne asiakkaan kanssa on käynnissä. (Grönroos 2009, Hölttä ym. 1997.)



KUVA 3. Kaksi palvelun laatu-ulottuvuutta (Grönroos, 2009)

Kumpi laadun ulottuvuuksista on sitten tärkeämpi? Jos yritys vastaa tähän kysymykseen väärin, niin se saattaa ohjata tekemisiään vääriin asioihin ja menettää näin kilpailukykyään. Laatuasioissa tuijotetaan liian usein vain teknisiin laatunäkökohtiin. Teknisen laadun ylivertaisuus kilpailijoihin nähden konkretisoituu vain silloin, kun yritys onnistuu tekemään sellaisen teknisen ratkaisun, johon kilpailijat eivät kykene. Näin tapahtuu nykypäivänä yhä harvemmin. Yritykset toteuttavat suurin piirtein samanlaista teknistä laatua. Teknisellä laadulla kilpaileminen on kallista ja kilpailuedun hetkellinen saavuttaminen ei ole tavoittelemisen arvoista. (Grönroos 2009, 104.)

Asiakkaaseen vaikuttaa myös se, että miten hän saa tilaamansa tuotteen tai palvelun ja millaiseksi hän kokee läpi käydyn prosessin. Jos asiakkaalla on myönteinen kuva palveluntarjoajasta, he antavat luultavasti pienet virheet anteeksi. Jos virheitä sattuu usein, imago kärsii, ja jos imago on kielteinen, mikä tahansa virhe vaikuttaa suhteellisesti enemmän. Imagoa voi pitää laadun kokemisen suodattimena. (Grönroos 2009, Hölttä ym. 1997.)

Toiminnallinen laatu on yhä tärkeämpää myös ohjelmistoliiketoiminnassa. Tekniikoilla ei saavuteta merkittävää kilpailuetua, joten laatu muodostuu suurelta osin toiminnallisesta laadusta. Kilpailijat voi lyödä tarjoamalla asiakkaille enemmän ja parempia palveluita, joissa korostetaan toiminnallista laatua. Teknisen laadun merkitystä ei voi tietenkään väheksyä. Teknisen laadun on oltava kunnossa. Se toimii perustana, jonka päälle toiminnallista laatua voidaan toteuttaa. (Grönroos 2009, 100 – 104.)

Laadun parannus on sisällytettävä yrityksen prosesseihin, eikä sitä voi ajatella erillisenä hankkeena. Jos laadun parantamista ajatellaan pelkästään läpi vietävänä ohjelmana, niin epäonnistumisen riski on hyvin suuri. Laadun parantaminen on oltava jatkuva prosessi. Jokaiselta organisaation jäseneltä vaaditaan laadun merkityksen arvostusta. Lecklinin mukaan laatuajattelu on vietävä koko yrityksen läpi (Lecklin 2006). Yrityksen johdon tulee osallistua aktiivisesti laadun kehittämiseen ja pitää sitä kaiken aikaa yllä. (Grönroos 2009, 141.)

2.3.2 Miten hyvä laadun tulisi olla?

Yritys usein tavoittelee optimaalista laatutasoa tuotteidensa ja palveluittensa suhteen. Liian korkean laatutason tavoittelu aiheuttaa lisäkustannuksia, mutta taas liian alhainen laatutaso aiheuttaa epäluottamusta asiakkaissa. Yleisesti voidaan sanoa, että odotusten ja kokemusten vastatessa toisiaan, laatu koetaan hyväksi. (Grönroos 2009, 142; Krishnamurthy 2008, 195.)

Ohjelmiston laatutavoite kannattaa asettaa hieman korkeammalle kuin mitä ohjelmiston tilaaja vaatii. Hyväksyttävä laatu tyydyttää asiakasta, mutta ei saa häntä välttämättä pitämään suhdetta palveluntarjoajaan vaalimisen arvoisena. Myönteisesti yllättynyt asiakas kertoo saamistaan kokemuksista muillekin. (Grönroos 2009, 141 - 142.)

Laatutavoitteiden ylittämisellä ajatellaan olevan myös negatiivisia puolia. Laatutavoitteiden ylittäminen aiheuttaa ylimääräisiä kustannuksia (Lecklin 2006). Asiakas voi tottua myös vaatimaan koko ajan enemmän. Myönteisen asian ei välttämättä tarvitse maksaa yhtään mitään, mutta se herättää asiakkaassa positiivisen tunteen. Positiivisia tunteita pitäisi pystyä herättämään toistuvasti, eikä pyrkiä teke-

mään sitä vain kerran. Pettymyksestä koituu suurempia kielteisiä vaikutuksia kuin yhden kerran koetut positiiviset vaikutukset olivat. (Grönroos 2009, 142.)

2.3.3 Laatumallit

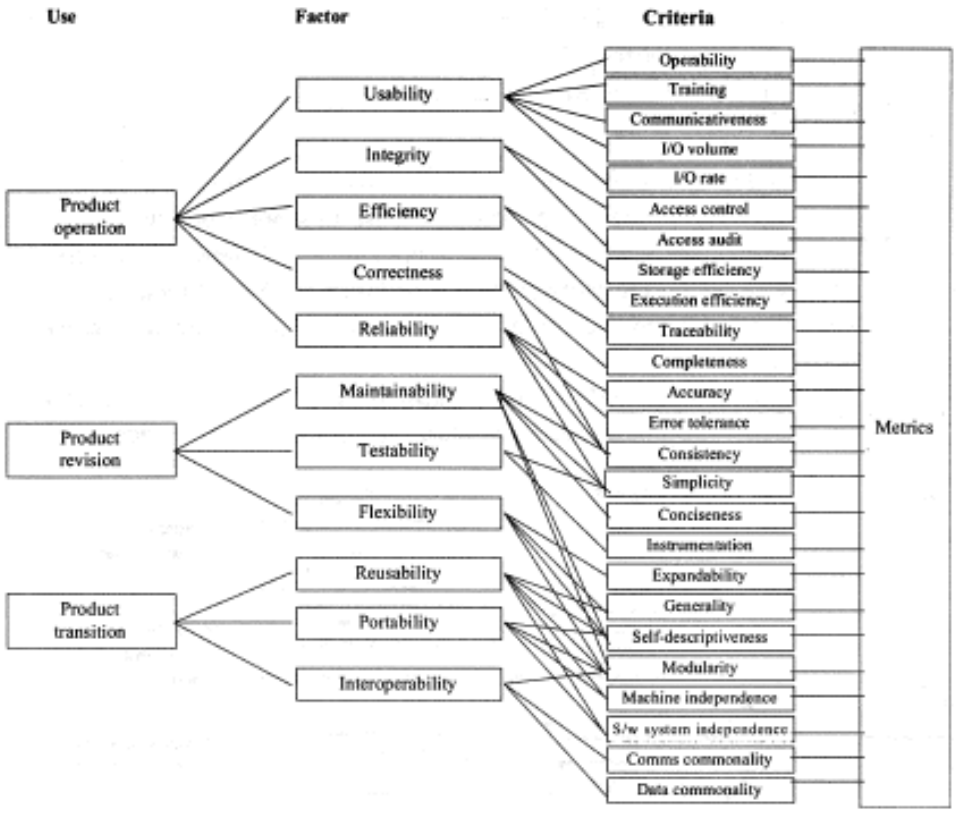
Ohjelmistotuotannossa laatua pyritään kuvaamaan erilaisten mallien avulla, koska laatu muodostuu eri ominaisuuksista. Ominaisuudet voidaan Fentonin ja Pfleegerin mukaan pääpiirteittäin jakaa kahteen ryhmään: sisäisiin ja ulkoisiin. Sisäisiin ominaisuuksiin kuuluvat esimerkiksi: ohjelmiston koko, uudelleenkäytettävyys ja muokattavuus. Ulkoisia ominaisuuksia puolestaan ovat esim. luotettavuus, käytettävyys ja ylläpidettävyys. Sisäiset ominaisuudet ovat yleensä helpommin mitattavia ja niitä voidaan mitata aikaisemmin kuin ulkoisia ominaisuuksia. Ohjelmiston kokonaislaatu koostuu ulkoisten ja sisäisten ominaisuuksien yhteisvaikutuksesta. (Fenton ym. 1997.)

Fenton ja Pfleeger esittelevät ohjelmiston laadun mallintamiseen kaksi mallia, jotka ovat McCallin (kuva 4) ja Boehmin (kuva 5) mallit. Kummatkin mallit on esitelty 1970-luvulla. Näissä malleissa on kuvattu erilaisia tekijöitä, jotka liittyvät ohjelmiston laatuun. Tekijöitä pilkotaan pienempiin osiin siten, että pilkkomisen seurauksena päästään lopulta käsiksi konkreettisesti mitattaviin asioihin. Kummatkin mallit käsittelevät laatua hyvin yleisellä ja abstraktilla tasolla. McCallin mallia on käytetty ISO 9126 -standardin perustana. (ISO 2008; Fenton ym. 1997, 339.)

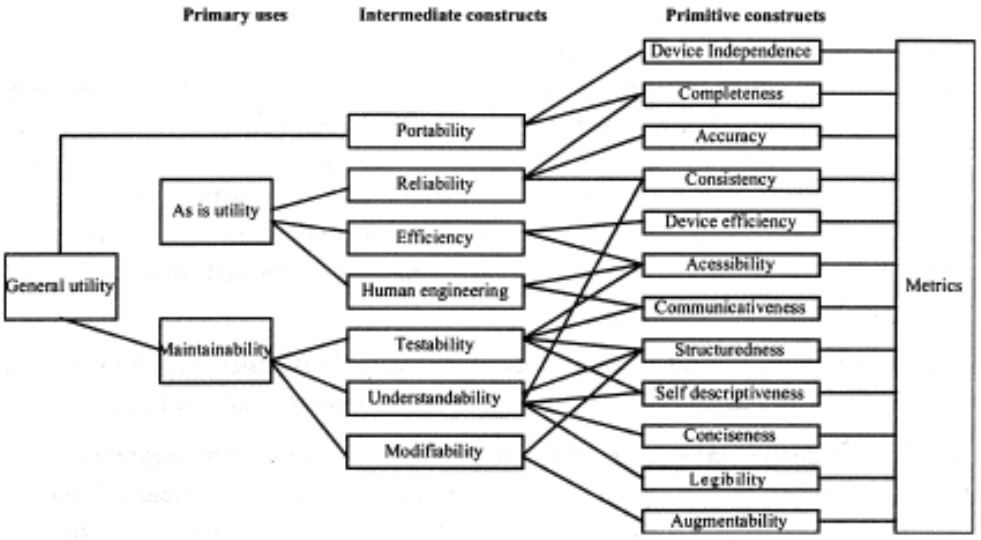
Jim McCall esitteli laatumallinsa vuonna 1977. McCall kehitti laatumallin alun perin Yhdysvaltojen ilmavoimien käyttöön järjestelmän kehittäjien ja järjestelmän kehitysprosessin tueksi. Laatumallissa on kolme näkökulmaa (use), jotka ovat tuotteen versiointi, tuotteen siirrettävyys ja tuotteen ominaisuudet. Nämä päätasot jakautuvat laatutekijöihin (factor), jotka puolestaan jakautuvat laadun kriteereihin (criteria). Kriteerit johtavat suoraan mitattaviin laatutekijöihin (metrics). McCall kehitti tämän mallin, jotta ohjelmistojen kehittäjät ja käyttäjät voisivat paremmin ymmärtää toisiaan. McCallin malli on esitetty kuvassa 4. (Fenton ym. 1997.)

Tutkitaan McCallin mallia parin tekijän (factor) kannalta, jotka ovat virheettömyys (correctness) ja luotettavuus (reliability). Näitä tekijöitä tutkimalla nähdään, että tekijöitä yhdistävät osittain samat laatukriteerit. Ohjelmiston virheettömyydellä tarkoitetaan sitä, kuinka hyvin ohjelmisto täyttää sille asetetut vaatimukset. Luotettavuudella tarkoitetaan ohjelmiston kykyä toimia luotettavasti siten, että suoritus ei

pääty virheeseen. Virheettömyydellä ja luotettavuudella on sama laatukriteeri: yhtenäisyys (consistency). McCallin laatumallin idea on, että yhdistettävät laatutekijät antavat kokonaisvaltaisen kuvan ohjelmistotuotteen laadusta. Yksittäinen laatumittari saadaan, kun vastataan kyllä ja ei -kysymyksiin. Kysymyssarjan vastauksia verrataan niihin laatukriteereihin, joista tämä kyseinen kriteeri on riippuvainen. (Fenton ym. 1997.)



KUVA 4. McCallin laatumalli (Fenton ym. 1997, 339)



KUVA 5. Boehmin laatumalli (Fenton ym. 1997, 339)

2.3.4 Laadunhallintajärjestelmät

Laadunhallintajärjestelmällä tarkoitetaan tapaa, jolla yritys johtaa ja ohjaa laatuun liittyvää toimintaa. Yleisesti ottaen se käsittää organisaatiorakenteen sekä sen suunnittelun, prosessit, resurssit ja dokumentaation, joita käytetään laatutavoitteiden saavuttamiseksi. (Lecklin 2006.)

Procompin laatukäsikirjassa kuvataan ohjelmistotuotannon eri prosessit ja niiden vaiheet. Laatukäsikirjan avulla pyritään yhdenmukaistamaan ja vakauttamaan toimintatapoja eri tilanteissa ja siten parantamaan laatua yrityksen ohjelmistotuotantoprosessissa.

Ohjelmistotuotteiden laadunmittaus osa Procompin laadunhallintajärjestelmää. Opinnäytetyö tulee olemaan osa tätä kokonaisuutta.

2.3.5 Sertifiointi ja standardointi

Toisen maailmansodan aikana laatuongelmat toivat konkreettisia ongelmia mm. Britannian korkeanteknologian yrityksiin, kuten esimerkiksi tehtaisiin, jotka valmistivat ammuksia ja pommeja. Koska valmistusta ei seurattu mitenkään, tehtaissa tapahtui useita räjähdyksiä. Tämä johti siihen, että tehtaat pakotettiin ottamaan käyttöön kirjallinen valmistusprosessin selvitys. Tämän lisäksi vaadittiin, että pro-

sessista pidetään kirjanpitoa. Standardin nimeksi tuli BS 5750, ja se tunnettiin nimellä *management standard*, sen johtamisprosesseihin liittyvän ohjeistuksen vuoksi. Vuonna 1987 Britannian hallitus suostutteli ISO:n ottamaan BS 5750:n kansainväliseksi standardiksi, jolle annettiin nimi ISO 9000. (QSL 2011.)

ISO 9000 -standardiperheeseen kuuluu erilaisia laatustandardeja, jotka soveltuvat kaikenkokoisiin ja kaikenlaisiin palvelualoihin. ISO 9000 -standardit käsittelevät laadunhallintaa. ISO 9000 -standardien tarkoitus ei ole yhdenmukaistaa yritysten järjestelmiä ja menetelmiä, vaan kukin yritys voi laatia itselleen parhaiten sopivan järjestelmän. Sen tulee kuitenkin soveltuvin osin ottaa huomioon standardin asettamat vaatimukset. Kansainvälinen Standardoimisliitto on sitoutunut päivittämään ISO 9000 -sarjaa. Päivittäminen tapahtuu katselmusten, parannusten ja standardien virtaviivaistamisen avulla. Päivittäminen takaa, että myös tulevaisuudessa yrityksille on käytettävissä tehokkaita johtamismalleja. (Lecklin 2006; Hölttä ym. 1997; SFS 2009.)

ISO 9001 -standardi määrittelee laadunhallintajärjestelmän vaatimukset mille tahansa organisaatiolle. Standardi on laadittu käyttäjäystävälliseen muotoon ja sellaisin termein, jotka ovat helposti tunnistettavissa kaikilla elinkeinoelämän sektoreilla. ISO 9001 -standardi on ainoa ISO 9000 -sarjan standardi, jonka vaatimusten pohjalta ulkopuolinen laitos voi sertifioida laatujärjestelmän. (SFS 2009.)

ISO 9004 -standardia käytetään läheisesti ISO 9001 -standardin kanssa. ISO 9004 -standardin avulla ISO 9001 -standardia laajennetaan siten, että sillä voidaan tavoittaa suurempi joukko osapuolia. ISO 9004 -standardi on harmonisoitu siten, että siirtyminen standardista toiseen olisi sujuvaa. (SFS 2009.)

ISO 9126 -standardi on laatustandardi, jonka tavoitteena on määritellä yhdenmukaiset mittaustavat ohjelmistojen laatuvaatimuksille. Tämä mahdollistaa sen, että eri ohjelmistotuotteita on helpompi verrata toisiinsa. McCallin laatumalli (Fenton ym. 1997, 339) on ollut perustana tälle standardille. ISO 9126 -standardi määrittelee, että ohjelmistotuotteen laatu voidaan jakaa kuuteen tekijään: funktionaalisuus, luotettavuus, käytettävyys, tehokkuus, ylläpidettävyys ja siirrettävyys. (ISO 2008.)

Procompin tavoitteena on standardoida heidän laatujärjestelmänsä ISO 9001 -standardin mukaiseksi. Laatujärjestelmän laatukäsikirjaa kehitetään edelleen ja laatukäsikirjan prosesseja tarkennetaan. (Voutilainen 2009.)

2.4 Ohjelmistotuotteen laadun ja puutteiden mittaaminen

Mittausjärjestelmän kehittäminen kerralla valmiiksi on käytännössä mahdotonta. Vaikeuksia aiheuttavat mittareiden valinnat ja mahdolliset muutokset liiketoiminnassa saattavat aiheuttaa tarvetta muuttaa seurattavia mittareita. Tämän vuoksi mittareita ei pidä kehittää staattisiksi. (Kankkunen, Matikainen, Lehtinen 2006.)

Mittausjärjestelmän kehittäminen on hidas prosessi. Teoreettisella tasolla mittareiden ja mittaamisen suunnittelu on vielä kohtuullisen helppoa. Mittareiden käytäntöön saattaminen vie kuitenkin paljon aikaa ja tässä vaiheessa kiinnostus mittaamista kohtaan saattaa vähentyä. Järjestelmän toimeenpanoon kuluva aika pitää kuitenkin aina ottaa huomioon. Mittariston lanseeraaminen ja käyttöönotto päätöksenteon tueksi voi vaatia 12–18 kuukautta. Mittausjärjestelmän ottaminen kokonaisvaltaisen johtamisfilosofian osaksi saattaa kestää vieläkin kauemmin. (Kankkunen ym. 2006.)

2.4.1 Puutteiden määrittely ja luokittelu

Ohjelmiston ongelmista puhuttaessa yleisesti käytetään termiä *puute*. IEEE:n standardin (982.2) mukaan puutteella tarkoitetaan poikkeamaa ohjelmistossa, jonka vuoksi ohjelmisto toimii virheellisesti. (IEEE 1998.)

IEEE:n standardissa 982.2 (IEEE 1998; Kipponen 2005) puutteet määritellään seuraavasti:

- *puute* (defect): Poikkeama tuotteessa. Puutteeksi luokitellaan esimerkiksi kehityksen alkuvaiheen laiminlyönnit.
- *virhe* (error): Ihmisen toiminto, jonka seurauksena ohjelmistoon syntyy vika. Virhe voi olla seurausta epätäydellisestä vaatimusmäärittelystä tai laiminlyönnistä ohjelmointivaiheessa.
- *häiriö* (failure): Häiriöllä tarkoitetaan tilaa, jossa esim. ohjelmamoduuli ei pysty suorittamaan vaadittua toimintoa tietyssä aikarajassa tai ollenkaan. Häiriö voi syntyä kun vika kohdataan.
- *vika* (fault): Vika on odottamaton tila, joka aiheuttaa toiminnalliselta yksiköltä esim. ohjelmamoduulilta vaaditun toiminnon epäonnistumisen. Vika voi aiheuttaa häiriön.

Ohjelmistotuotteen puutteet aiheutuvat erilaisista virheistä ohjelmistoprosessin aikana. Virheet ovat usein inhimillisiä ja yleensä suurimmat virheet tapahtuvat määrittelyvaiheessa, kun ohjelmiston toimintaa analysoidaan. Ohjelman toiminta voidaan käsittää väärin tai ohjelman määritys tehdään huolimattomasti. (Kipponen 2005.)

Fenton ja Pfleeger määrittelevät yhdessä IEEE:n kanssa ohjelmiston laadun puutteiden terminologiaa kuvan 6 esittämällä tavalla. (Fenton 1997 ym.; IEEE 1998.)

KUVA 6. Ohjelmiston laadun puutteiden terminologia (Fenton ym. 1997)

Vika on ihmisen virheen ilmentymä ohjelmistossa ja aiheutuu ohjelmistoon inhimillisestä virheestä. Ohjelmiston suunnittelija saattaa ymmärtää väärin vaatimuksen ja ohjelmoi logiikan väärin. Väärin ymmärretty ja ohjelmoitu koodi aiheuttaa viallisen ohjelmiston. Viallisesta ohjelmistosta tehdään viallinen käyttöohje, joten ohjelmistosuunnittelijan inhimillinen virhe tuottaa monta erillistä vikaa. (Kipponen 2005.)

Viat voivat aiheuttaa tietyissä olosuhteissa häiriön, jonka sattuessa ohjelmiston toiminta poikkeaa sen vaaditusta toiminnasta. Kaikki viat eivät johda häiriöihin, koska vian sisältämää ohjelmakoodia ei ehkä koskaan suoriteta. Viat ovat ongelmia, jotka sovelluskehittäjät havaitsevat, kun taas häiriöt ovat ongelmia, jotka käyttäjät havaitsevat. (Kipponen 2005.)

2.4.2 Yleistä mittaamisesta

Ohjelmiston mittaaminen on yleensä koettu lisäominaisuutena tai ylellisyytenä ohjelmistokehityksessä. Sille ei monestikaan katsota olevan tarvetta, mutta Fenton ja Pfleeger näkevät asian toisin. DeMarco on Fentonin, Pfleegerin ja Kipposen mukaan todennut seuraavasti: ”Et voi kontrolloida sitä, mitä et voi mitata.” Tämä yksiselitteisesti kertoo sen, että ohjelmistotuotteen laadun seuraamisen kannalta

tuotteeseen kannattaa liittää mittareita, että sitä voidaan kontrolloida ja hallitusti ohjata laadukkaammaksi. (Fenton ym. 1997; DeMarco 1982; Kipponen 2005.)

Mittaamisen tarkoitus on, että käsitteet saadaan näkyvimmäksi, ymmärrettävämpään muotoon ja ohjattavammaksi. Mittaaminen auttaa hahmottamaan kokonaiskuvaa siitä, onko ohjelmistotuotteen tila hyvä vai huono. (Fenton ym. 1997; DeMarco 1982; Kipponen 2005.)

Mittaaminen on tärkeä osa prosessien hallintaa ja niiden kehittämistä. Oikeiden asioiden mittaaminen auttaa ymmärtämään prosessia ja ennaltaehkäisemään jopa tulevia virheitä. Kovin tavallista on väittää, että yrityksen sisällä kyllä tiedetään ilman mittaamistakin, että missä mennään. Tuntumat ovat kuitenkin usein aivan vääriä. (Hokkanen, Strömberg 2006, 48 - 49.)

2.4.3 Mitattavaksi tekeminen

Galileo Galilei (1564–1642) on todennut: ”Tee mitattavaksi se, jota et voi mitata”. Tällä Galilei on tarkoittanut sitä, että jos haluaa saada jostain asiasta lisätietoja, tulee selvittää, miten kohteen voi saattaa mitattavaan muotoon. Meidän tulee siis etsiä ja mitata semmoisia asioita, jotka meitä kiinnostavat. Mittausten avulla saamme tutkimistamme asioista lisätietoa ja opimme ymmärtämään mitattua asiaa paremmin. Mitata ei siis kannata vain mittaamisen ilosta, näistä mittaustuloksista ei ole meille mitään hyötyä.

Farrell-Vinay, sekä Fenton ja Pleeger esittelevät GQM (Goal-Question-Metric) -tyyppisen lähestymistavan ohjelmistomittareiden löytämiseksi. (Farrell-Vinay 2008; Fenton ym. 1997.)

GQM-mallintaminen jakautuu kolmeen osa-alueeseen, jotka ovat:

- Päämäärät (Goal): Määritellään päämäärät. Mitä yritetään tehdä? Yritetään karsia ohjelmistotuotteesta virheitä ja yritetään löytää ohjelmiston ongelmapaikat.
- Kysymykset (Question): Määritellään kysymykset, joka ohjaavat päämääriin. Kuinka saadaan pienennettyä virheiden määrää ohjelmistossa? Kuinka löydetään virhealttiimmat paikat ohjelmiston koodista?

- Mittarit (Metric): Mittarista luetut tiedot ovat ratkaisuja edellä kuvattuihin kysymyksiin. Mittareita voisivat olla esim. ”Virheiden määrä moduuleittain projektissa x”., tai ”Virheiden kokonaismäärä projektissa x”.

GQM on joustava ja käyttökelpoinen tapa löytää mittauskohteita ohjelmistotuotteesta. Ohjelmistoprojektin määrittelyvaiheessa on myös mahdollista suoraan määritellä mitattavia kohteita.

2.4.4 Ohjelmistomittarit

Ohjelmistomittarit kattavat Fentonin ja Pfleegerin (Fenton ym. 1997) mukaan seuraavat osa-alueet:

- Hinta ja työmääräarviot: Työmäärien arviointi on keskeinen osa projektin suunnittelua, koska tätä käytetään projektin hinnan laskemisessa. Myös henkilöiden resursointi on sidottu työmäärien arviointiin, koska työmääräarvioiden avulla kiinnitetään projektin resurssit. Työmäärien arvioiminen on sitä helpompaa, mitä tarkemmin kaikki halutut vaatimukset on purettu auki ja määritelty. Työmäärien ennakkoon laskemiseen on tehty erilaisia malleja, kuten COCOMO II ja Albrechtin toimintopistemalli. (COCOMO II 1991, Albrecht 1979.)

- Tuottavuusmittarit ja mallit: Analysoidaksemme mallia ja tuottavuutta, meidän tulee mitata yhtä resurssiominaisuutta, kuten esimerkiksi henkilöstöä. Yrityksen johdon yksi mielenkiinnon kohde projektissa on usein henkilön tuottavuus, eli kuinka hyvin resursoitu henkilö vastaa hänelle asetettuihin työmääriin. (Fenton ym. 1997; Kipponen 2005.)

- Mittaustiedon kerääminen: Yksinkertaisimmatkin mallit tarvitsevat taustalleen tarkat mittaukset. Tallennuspaikkana kannattaa suosia mittaustietokantaa, koska sen avulla saadaan kerättyä kaikki tiedot yhteen ja samaan paikkaan. Mittaustietokannan käyttö helpottaa myös historiatiedon hakemista käsittelyä. (Fenton ym. 1997; Kipponen 2005.)

- Laatumallit ja mittarit: Jos tuotteen laatu ei ole hyvä, niin tuottavuus menettää merkityksensä. Tämän vuoksi on kehitetty laatumalleja, joiden mittaustuloksia voidaan yhdistellä tuottavuusmallien kanssa. (Fenton ym. 1997; Kipponen 2005.)

- Luotettavuusmallit: Useimmat laatumallit sisältävät luotettavuuden yhtenä ohjelmiston laadun komponenttina, kuten esimerkiksi McCall tekee laatumallissaan (kuva 4). Toisinaan ohjelmiston luotettavuutta on tarpeen mitata ja ennustaa myös itsenäisenä osana, jolloin tuotteen ymmärrettävyyttä ja kontrollia voidaan parantaa. (Fenton ym. 1997; Kipponen 2005.)

- Suorituskyvyn arviointimallit: Suorituskyvyn arviointimalleilla pyritään arvioimaan ja parantamaan tuotteen suorituskykyä. Suorituskyvyn ominai-

suuksia ovat esimerkiksi vasteaika, suoritusnopeus ja virheestä toipumisnopeus. (Fenton ym. 1997; Kipponen 2005.)

- Rakenne ja kompleksisuusmallit: Ohjelmistoprojektissa olisi hyvä pystyä ennustamaan mitkä kehitettävän järjestelmän osat tai moduulit voivat olla virhealttiimpia tai monimutkaisempia kuin toiset. Monimutkaisuus tarkoittaa yleensä myös sitä, että näiden moduuleitten ylläpitoon tullaan käyttämään enemmän aikaa, kuin yksinkertaisempien moduuleitten ylläpitoon. (Fenton ym. 1997; Kipponen 2005.)

- Johtaminen mittareiden avulla: Mittaamisesta on tullut tärkeä osa ohjelmistoprojektin johtamista. Asiakkaat ja kehittäjät haluavat nähdä ja varmistua mittausten pohjalta luotujen kaavioiden ja käyrien avulla, että heidän projektinsa etenee suunnitelmien mukaisesti. Kaavioiden pohjalta on myös helpompi ymmärtää projektin todellinen tila, koska raportteihin helposti sisällytetään teknistä sanastoa ja tätä voi olla vaikea ymmärtää. (Fenton ym. 1997; Kipponen 2005.)

- Menetelmien ja työkalujen arviointi: Markkinat tarjoavat koko ajan uusia menetelmiä ja keinoja, että kuinka yritys voi parantaa organisaationsa tai projektiensa tuottavuutta. Näiden menetelmien ja välineiden toimivuuden todentaminen on kuitenkin kohtuullisen vaikeaa. Tämän vuoksi eri yritykset ja organisaatiot testaavat, että kuinka jokin väline toimii heidän organisaatiossaan, ennen kuin kyseinen väline otetaan lopullisesti käyttöön. Testaukset vievät aikaa ja kuluttavat resursseja, mutta yleensä niiden tekeminen kannattaa, koska testaamattoman välineen ottaminen käyttöön sisältää myös riskinsä. (Fenton ym. 1997; Kipponen 2005.)

- Kyvykkyyden ja kypsyyden arviointi: Yhdysvaltain ohjelmistotuotannon instituutti (US Software Engineering Institute, SEI) julkaisi 1980-luvulla mallin, jolla pyritään arvioimaan organisaation kyvykkyyttä ohjelmistontoimittajana. Mallissa yritys pisteytetään asteikolla 1-5. Mallin antaman pisteytyksen perusteella pystytään mm. arvioimaan, että kuinka hyvin yrityksen ohjelmistosuunnittelijat ovat sisäistäneet valmistettavan ohjelmistotuotteen ja kuinka hyvin he pystyisivät sen toteuttamaan. (Fenton ym. 1997; Kipponen 2005.)

Ohjelmistomittarit voidaan korkealla tasolla jakaa Kipposen ja Fentonin ja Fleegerin (Kipponen 2005; Fenton ym. 1997) mukaan seuraaviin ryhmiin:

- Prosessimittarit: Prosessimittarit kuvaavat prosessin tilaa. Haluamme usein tietää, että kuinka kauan jonkin toiminnon tekemiseen menee aikaa tai kuinka paljon se tulee maksamaan. Prosessimittareilla voidaan myös suhteuttaa prosessin eri osia keskenään. (Fenton ym. 1997.)

- Tuotemittarit: Tuotemittarit kuvaavat ohjelmiston ominaisuuksia, joita ovat esimerkiksi koko, monimutkaisuus, suunnittelupiirteet, suorituskyky ja laatu. Tuotemittareilla mitataan ohjelmistoprosessin tuottamia ohjelmiston ominaisuuksia. (Fenton ym. 1997.)

- Resurssimittarit: Resurssimittareilla kuvataan projektiin liittyviä resursseja. Projektin resursseja ovat esim. henkilöstö, materiaali ja välineet. Resurssien mittaamisen avulla voimme selvittää esim. projektin henkilöstön koon ja

> projektin henkilöstölle maksettavat palkat. Resurssimittarien avulla tarkkailemme projektille asetettuja syöttötietoja, joiden avulla projekti tuottaa ulos prosessin mukaista lopputuotetta. Jos lopputuote ei vastaa esim. laadultaan asetettuja vaatimuksia, niin asetetuissa resursseissa voi olla puutteita. (Fenton ym. 1997.)

Kaikissa edellä mainituissa ryhmissä mitattavat ominaisuudet voidaan jakaa sisäisiin ja ulkoisiin ominaisuuksiin. Sisäiset ominaisuudet voidaan mitata tutkimalla tuotetta, prosessia tai resurssia itsessään erillään sen käyttäytymisestä. Ohjelmakoodin sisäisiä ominaisuuksia ovat esim. ohjelmarivien määrä ja ohjelmakomponenttien määrä. Ohjelmamoduulien ulkoisia ominaisuuksia voidaan mitata vain kun ohjelmakoodia suoritetaan. Ulkoisia ominaisuuksia tässä tapauksessa ovat esim. virheestä palautumisen nopeus ja käyttäjän kohtaamien virheiden määrä ohjelmistossa. (Kipponen 2005.)

2.4.5 Laatumittarit

Ohjelmiston laatumittarit keskittyvät ainoastaan ohjelmistotuotteen laadun mittaamiseen. Ohjelmiston laatumittarit ovat joukko mittareita, joilla mitataan tuotteen, prosessin ja projektin laatunäkökohtia. Laatumittarit voidaan Kai'in (Kan 2003) mukaan jakaa kolmeen kategoriaan:

- kehitysprosessin laadun mittarit
- lopputuotteen laadun mittarit
- ylläpidon laadun mittarit.

2.4.6 Mittauksen kohdistaminen

Eri organisaatiotasoilla käytettävät mittarit eivät voi olla täysin samanlaisia. Jos mittari on tehty alemmalle portaalle, niin sitä ei voi välttämättä käyttää ylemmällä organisaatiotasolla. Aina, kun mittaria ollaan viemässä pois omasta organisaatioympäristöstä, niin mittarin mielekkyys voidaan kyseenalaistaa. (Kankkunen ym. 2006.)

Mittaus on mielellään kohdistettava aina tiimitasolle. Jos tiimitasolle kohdistaminen ei ole mahdollista, niin mittaus on kohdistettava vastaavan tiimin johtajaan. Yksilötasolla tehtävä mittaaminen voi johtaa lyhyen tähtäimen optimointiin, joka voi haitata tiedon siirtoa yksilöiden välillä. (Kankkunen ym. 2006.)

2.4.7 Mittauksen automatisointi

Vaikka ohjelmiston tilan seuraaminen on tärkeää, niin se voi myös tuntua rasitteelta, jos mittaustoimenpiteet lankeavat aina saman ihmisen tehtäviksi. Mikäli mittaaminen aiheuttaa paljon lisätyötä, henkilöstö kokee sen vaikeaksi ja vastenmieliseksi. (Kankkunen ym. 2006.)

Mittauksen automatisoinnilla vapautetaan henkilöt itse mittaustoimenpiteestä ja näin voidaan keskittyä tulosten analysointiin. Koska mittauksen automatisoinnilla päästään siihen, että mittaustuloksia (raportteja) syntyy automaattisesti ja tietyn väliajoin, niin mittaamiselle saadaan säännöllistä jatkumoa ja tuloksia voidaan tilastoida paremmin.

Mittauksen automatisointi tarkoittaa mittaustoiminnallisuuden automaattista suorittamista. Mittaustoiminto itsessään ei ole uusi, vaan mittausprosessi suoritetaan pelkästään automaattisesti. Tämä tarkoittaa sitä, mittaustoiminto voidaan esim. ajastaa suoritettavaksi tiettynä ajanhetkenä, esimerkiksi jokaisen kuukauden viimeisenä päivänä. (Kankkunen ym. 2006.)

2.4.8 Mittaaminen päätöksenteon tukena

Mittaamista voidaan käyttää yrityksessä päätöksenteon tukena. Päätöksentekoa ohjaavat mittarit voidaan jakaa seuraavasti:

- parametrien kontrollointi
- säännöllinen toiminnan arviointi
- strategian kyseenalaistaminen

Kontrolloimisen tarkoituksena on tarkkailla jatkuvasti toiminnan kannalta olennaisia parametreja, joiden tavoitetasojen rikkominen ei ole hyväksyttävää. Tällaisten mittareiden ensisijainen tarkoitus on varoittaa mahdollisista vaikeuksista, joita kohti ollaan menossa. (Kankkunen ym. 2006.)

Toiminnan arvioinnin tarkoituksena on tietyin väliajoin tarkistaa, ovatko yrityksen pitkän tähtäimen tavoitteet toteutumassa suunnitelman mukaisesti. Toimintakatsaus tehdään säännöllisesti, esimerkiksi kerran kuukaudessa. (Kankkunen ym. 2006.)

Mittausjärjestelmän tuloksilla voidaan myös kyseenalaistaa yrityksen strategiaa. Strategian toteutumista voidaan seurata oikein valituilla mittareilla. Jos strategia ei näytä toteutuvan, on toimintaa syytä tarkistaa tai tehdä uusi toteutettavissa oleva strategia. Kyseenalaistamisella haetaan vastausta kysymykseen: ”Jos yritys pärjää strategisesti määritellyillä mittareilla, onko se pitkällä tähtäimellä kannattava.” Kyseenalaistava mittaus ja arviointi tulisi tehdä säännöllisesti, mutta kohtuullisen harvoin. Arvioinnissa yrityksen ylemmän johdon tulisi olla läsnä. (Kankkunen ym. 2006.)

3 LAATUMITTARIDEMO-PROJEKTIN TOTEUTUS

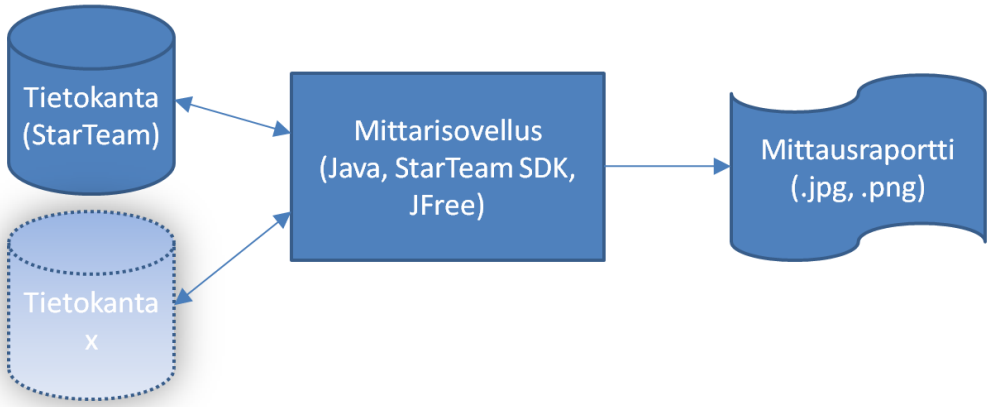
3.1 Tausta ja tavoitteet

Projektin lähtökohtana oli toteuttaa demo, jolla voidaan tehdä yksinkertainen raportti datasta, jota säilötään yrityksen normaalin tuotekehitysprosessin seurauksena tietokantaan. Lähtökohtaisena tavoitteena oli käyttää tietolähteenä kahta paikkaa: muutoksenhallinnan tietokantaa ja tuntikirjaussovelluksen tietokantaa.

Lopullinen mittarisovellus toteutetaan myöhemmin omana projektinaan, jos tälle nähdään tarvetta. Demoprojektin tuloksena saadaan erilaisia koodiesimerkkejä siitä, miten lähdetietoihin päästään käsiksi ja kuinka niitä pystytään ohjelmallisesti rajaamaan. Projektin tekemisen ohessa mietitään myös muita mittauskohteita ja pyritään selvittämään sisäisten prosessien epäkohtia. Tällaisia epäkohtia ovat esimerkiksi asiat, joita ei pysty mittaamaan sisäisten prosessien toimimattomuuden vuoksi.

3.2 Järjestelmän yleiskuvaus

Toteutettu sovellus käsittää kolme pääosa-aluetta, jotka ovat lähdetiedot, mittarisovellus ja mittausraportti. Järjestelmän yleiskuva on nähtävissä kuvassa 7. Järjestelmän lähdetiedot sijaitsevat tietokannassa, joka toimii mittarin mittaustietokantana. Mittarisovellus hakee tietokannasta tietoja halutuilla parametreilla ja generoi niistä ulos raportin, joka tallennetaan kuvatiedostomuotoon. Mittarin lähdetiedoista on kerrottu tarkemmin luvussa 3.5 (Lähdetiedot ja liittymät) ja mittarisovelluksen teknisestä toteutuksesta luvuissa 3.3 (Koodaustekniikka ja ympäristöt) ja 3.4 (Rajapinnat ja kirjastot).



KUVA 7. Laatumittaridemo-sovelluksen arkkitehtuuri

3.3 Koodaustekniikka ja ympäristöt

Mittarisovellus päätettiin toteuttaa Java-ohjelmointikielellä, koska yrityksen intranet-käyttöön ollaan ottamassa lähitulevaisuudessa Liferay-ympäristö. Liferay on Java-pohjainen ympäristö, joten mittaridemon toteuttaminen Javalla tuntui luontevalle vaihtoehdolle. Tietovarastona toimivaan StarTeamiin (StarTeam 2011) on myös saatavilla Java-kielellä toteutettu luokkakirjasto.

Ohjelmointivälineeksi valittiin NetBeans (NetBeans 2011), koska se tarjoaa hyvän välineen Java-projektien toteutukseen. NetBeansin käytöstä on myös aikaisempia kokemuksia yrityksessä, joten senkin puolesta valinta oli perusteltu.

Demoympäristössä oli käytettävä 32-bittistä JDK:ta, koska 64-bittinen ei ole yhteensopiva Starteamin ohjelmiston kanssa. Tästä aiheutuvat ongelmat tulivat esille projektin tekemisen aikana ja kehityskoneelle oli asennettava myös 32-bittinen Java. (Java Development Kit 2011.)

3.4 Rajapinnat ja luokkakirjastot

Ohjelmointityössä käytettiin StarTeam-välineen mukana asentuvaa SDK (Software Development Kit) -ohjelmistopakettia. Tätä SDK-pakettia käyttämällä pääsee käsiksi StarTeamin tietovarastoon. SDK tarjoaa tarvittavat rajapintafunktiot, joita käyttämällä voidaan muodostaa yhteys tietokantaan ja hakea tietoa halutuilla parametreilla. (StarTeam Software Development Kit 2011.)

Raporttien tekemisessä käytettiin apuna JFreeChart-luokkakirjastoa. Valittu kirjasto on ilmainen ja tarjoaa valmiita tulosten esitystapoja. Valittavissa on esimerkiksi projektissa käytetty pylväsdiagrammi. Tämän lisäksi käytettävissä ovat myös esimerkiksi piirakkadiagrammi ja viivadiagrammi. Luokkakirjasto valittiin, koska se oli ilmainen ja täyttää laatumittaridemon raporttien tulostamiseen vaaditut kriteerit. (JFreeChart 2011.)

3.5 Lähdetiedot ja liittymät

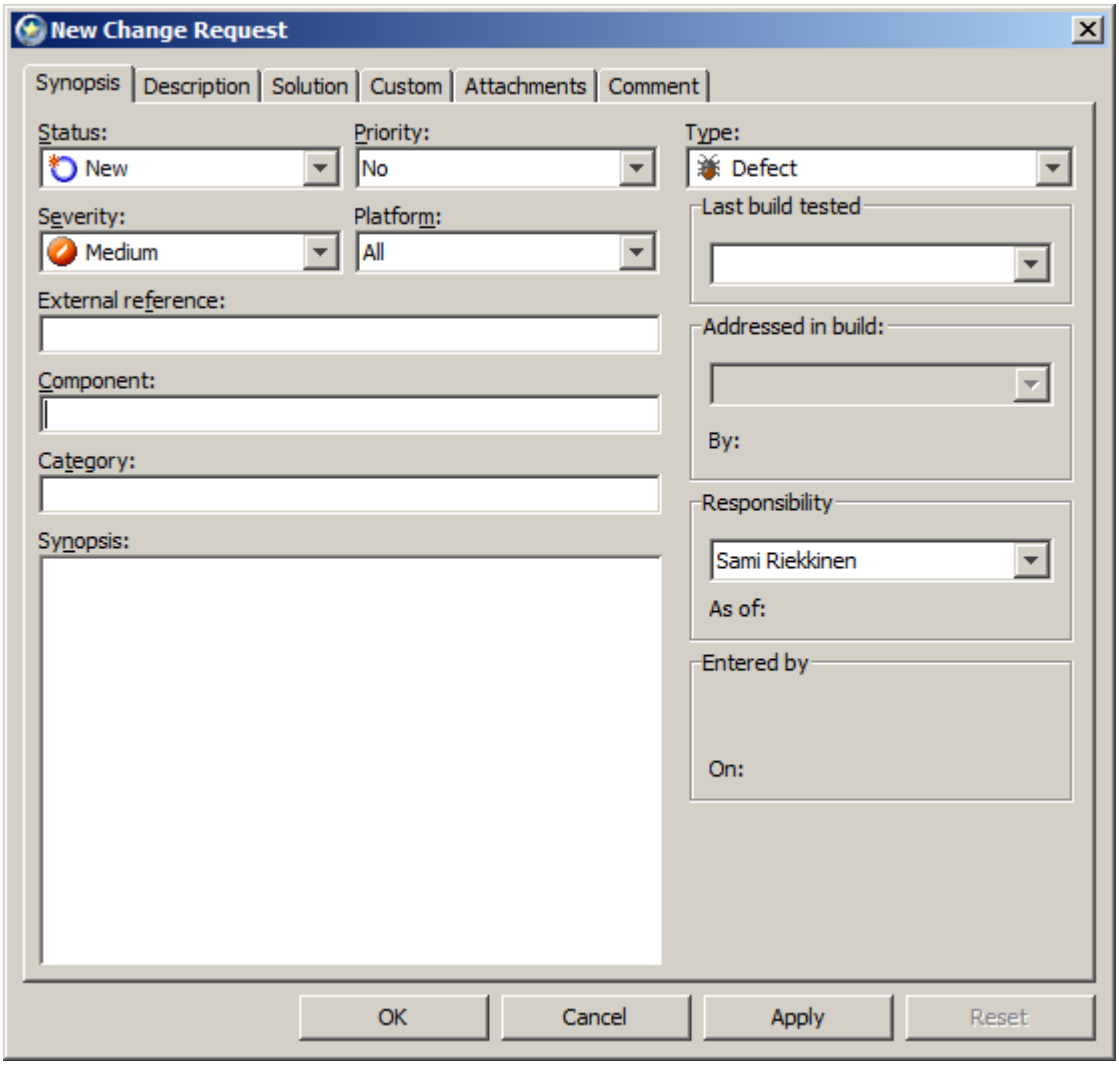
Mittaridemo saa lähdetietonsa StarTeam-tietokannasta, jonka pohjalta raportti muodostuu. StarTeam on väline, jota Procomp käyttää pääasiallisesti kolmeen tarkoitukseen: versionhallintaan, julkaisujenhallintaan ja muutoksenhallintaan. Versionhallintaan tallennetaan esimerkiksi yhden tuotteen kehitykseen sisältyvät dokumentit ja lähdekoodit. Näitä tiedostoja ylläpidetään Starteam-välineen avulla.

Mittaridemossa keskitytään muutoksenhallintaan ja siihen liittyviin muutosvaatimuksiin (Change Request). Projektin lopputuotteena syntyvässä raportissa esitetään eri tilassa olevien muutosvaatimusten lukumäärät. Muutosvaatimukset kirjataan käsin järjestelmään, joka tallentuu lopulta tietokantaan. Uuden muutosvaatimuksen keskeisiä tietoja ovat mm. muutoksen kirjaushetki, vastuuhenkilö, muutosta koskettava ohjelman kohta (moduuli) ja muutosta koskevat havainnot.

Muutosvaatimus voi olla tyypiltään ehdotus tai virhe. Ehdotuksiksi kirjataan yleensä muutokset, jotka nähdään kehittämispiirteinä. Virheiksi kirjataan muutosvaatimukset, joissa ohjelma toimii väärin tai ohjelman toiminta nähdään muulla tavoin puutteelliseksi. Fentonin, Pfleegerin ja Kipposen (Fenton ym. 1997; Kipponen 2005) mukaan tärkeimmät ohjelmistossa esiintyvästä puutteesta kerättävät tiedot ovat seuraavat:

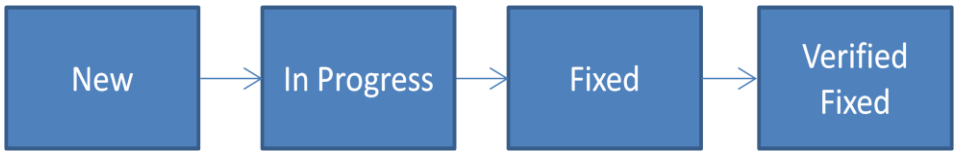
- *sijainti*: muutetun moduulin tai dokumentin yksilöllinen tunniste
- *aika*: muutoksen tekoaika
- *oire*: muutoksen tyyppi
- *lopputulos*: muutoksen onnistuminen, kirjataan miten muutos testattiin
- *mekanismi*: kuka teki muutoksen ja miten
- *syy*: korjaava, mukauttava, ehkäisevä vai täydentävä
- *vakavuus*: kuinka paljon muutos vaikuttaa systeemiin
- *hinta*: muutoksen tekemiseen ja testaamiseen kulunut työaika.

Ideaalinen muutosvaatimuksen elinkaari on esitetty kuvassa 9. Muutosvaatimus syötetään järjestelmään, jolloin se saa tilan uusi (new). Uuden muutosvaatimuksen syöttöikkuna StarTeam-välineessä on esitetty kuvassa 8.



KUVA 8. Uuden muutosvaatimuksen lisääminen Starteam-välineellä

Ohjelmistosuunnittelija ottaa vaatimuksen toteuttavakseen ja asettaa sen tilaan, joka kertoo sen olevan tekeillä (in progress). Kun vaatimus on korjattu, se asetetaan tilaan korjattu (fixed). Testauksen avulla tarkistetaan, että vaatimus toimii, kuten se on määritelty. Muutosvaatimuksen tilaksi asetetaan lopuksi hyväksytysti korjattu (verified fixed), jos muutos läpäisee testausprosessin.



KUVA 9. Ideaalinen muutosvaatimuksen elinkaari

Muutosvaatimuksen elinkaari kuitenkaan harvoin toteutuu kuvan 9 esittämällä tavalla. Tämä johtuu joko siitä, että vaatimuksen toteutus ei vastaa määrityksiä tai vaatimusta ei voida toteuttaa, koska lähtötiedot vaatimuksen toteuttamiselle ovat puutteelliset. Tällöin vaatimus joudutaan asettamaan tilaan avoin (open). Avoimessa tilassa olevat vaatimukset odottavat lisätietoja ja tarkennuksia, että ne voidaan toteuttaa tai korjata. Jos muutosvaatimuksia joudutaan asettamaan toteutuksen aikana paljon avoimeen tilaan, saattaa olla kyse määrittelytyön osittaisesta epäonnistumisesta tai määrittelyn puutteellisuudesta. Yllä esitellyt muutosvaatimusten tilat ovat tyypillisimpiä tiloja, joita muutosvaatimuksille asetetaan projektin aikana.

Alkuperäisen suunnitelman mukaan tavoitteena oli tehdä mittari myös tuntikirjaussovelluksen tiedoista, mutta tämä osa suunnitelmasta hylättiin, koska muutoksenhallintajärjestelmän ja tuntikirjaussovelluksen tietoja ei voitu yhdistää toisiinsa. Tuntikirjaussovelluksella ei ole suoraa yhteyttä muutoksenhallintajärjestelmään, vaan se on itsenäinen ja erillinen ohjelmisto. Jos tuntikirjauksella halutaan seurata työtehtävälle toteutunutta aikaa, kaikki projektin työtehtävät pitää kirjata järjestelmään ja seurata niille määriteltyjä työmääriä muulla tavoin. Tämän vuoksi katsottiin, että tuntikirjausjärjestelmän mittaaminen on hyödytöntä, ennen kuin tuntikirjaussovellus saadaan liitettyä osaksi muutoksenhallintaa.

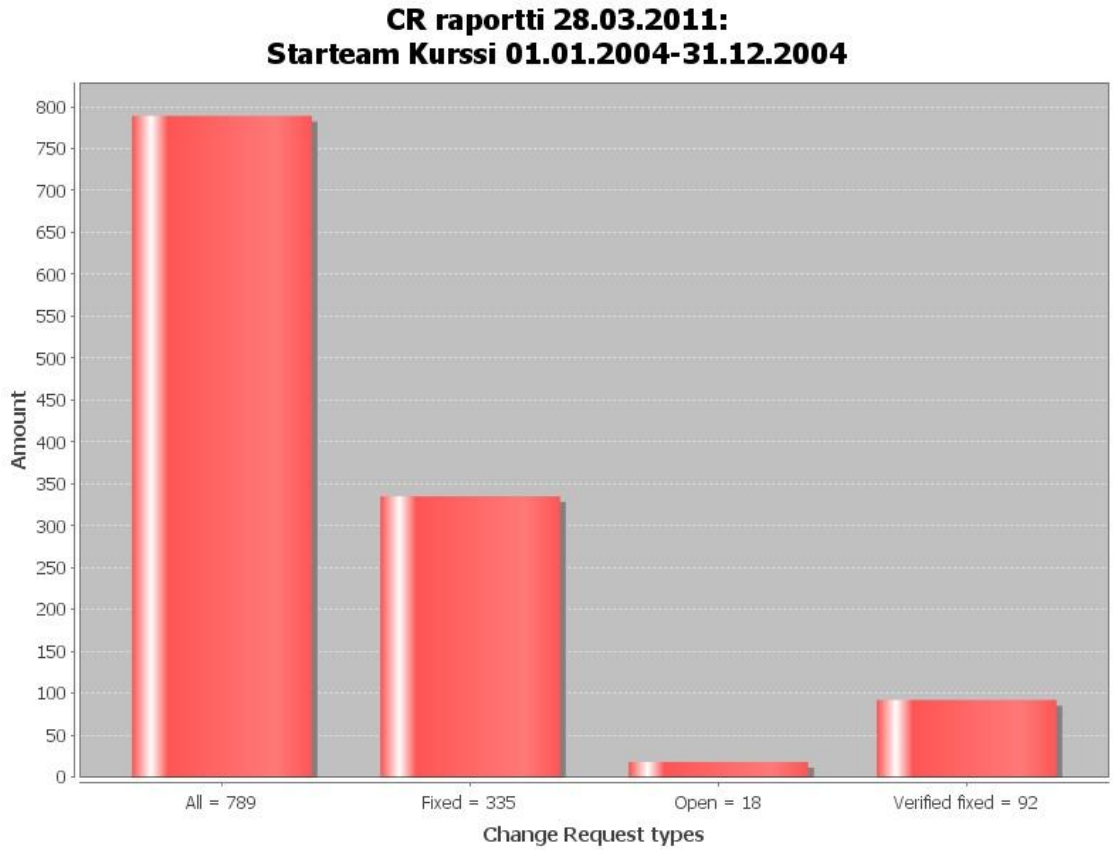
4 TULOKSET JA TULOSTEN TARKASTELU

4.1 Toteutetut mittarit

Projektin tavoitteena oli toteuttaa mittaridemo, jossa voidaan testata rajapinnat käytössä oleviin järjestelmiin. Projektin edetessä päätettiin luopua tuntikirjaussovelluksen mittarista, koska se toimii erillisenä järjestelmänä eikä näin tarjoa suoraa yhteyttä muutosvaatimuksiin. Tuotteen tilan mittaaminen oli selkeä kokonaisuus, ja tämän vuoksi demomittarin toteutuksessa kannatti keskittyä muutosvaatimusten tarkasteluun.

Toteutettu mittari kertoo, kuinka paljon tietynlaisia virheitä tietyssä projektissa on listattu haetulle aikavälille. Kuvassa 10 näkyy ohjelman tuottama lopputuote eli raportti. Ohjelmalle on annettu lähtötietoina halutun projektin nimi, josta tietoja halutaan, ja haluttu aikaväli.

Ohjelman suoritus tuottaa JPG-kuvan, jossa näkyy projektin nimi, aikaväli ja muutosvaatimusten tilat. ”All”-tilassa olevat vaatimukset pitävät sisällään kaikki muutosvaatimukset (Change Request), jotka on syötetty järjestelmään haetulla aikavälillä. ”Fixed”-tilassa olevat vaatimukset ovat muutosvaatimuksia, jotka ohjelmistosuunnittelija on korjannut. ”Open”-tilassa olevat muutosvaatimukset ovat avoimessa tilassa olevia vaatimuksia, joten niiden korjaamiseen tarvitaan lisätietoja tai niiden vaatimus on muuten epäselvä tai puutteellinen. ”Verified fixed” -tilan muutosvaatimus on korjattu ja tarkistettu testausprosessin vaatimalla tavalla.



KUVA 10. Virheet ja niiden tilat haetulla aikavälillä

Mittari kuvaa tuotteen tilaa haetulla aikavälillä. Kun mittaus suoritetaan säännöllisesti, esim. kerran kuukaudessa, voidaan tarkastella, mihin suuntaan tuotteen tila on kehittymässä. Avointen (open) muutosvaatimusten tilojen pysyessä suurena projektin tekemisen aikana, määrittelytyö on mahdollisesti tehty puutteellisesti tai epätarkasti (IEEE 1998; Kipponen 2005). Kuvassa 10 näkyvä mittarin tulos on otettu testiprojektista ”Starteam Kurssi”.

Mittausprosessin seurauksen tulokset tallennetaan paikkaan, josta ne ovat kaikkien niiden henkilöiden saatavilla, jotka ovat oikeutettuja raportteja tarkastelemaan.

4.2 Laatumittaridemon materiaali

Laatumittaridemolle luotiin versionhallintaan oma projekti (Laatu – QualityMeasurerDemo), jonne tullaan sijoittamaan projektiin kuuluva dokumentaatio ja lähdekoodit. Kansiorakenteesta löytyvään ”Future”-kansioon tehdään ohjelmistosuunnitelma mittaridemoa seuraavalle versiolle, johon voidaan suunnitella uusia ominaisuuksia.

4.3 Laatumittaridemon käyttöönotto

Laatumittaridemoa ei asenneta yrityksen palvelimelle, vaan sitä tullaan käyttämään alkuvaiheessa työasemilta.

5 POHDINTA JA JATKOTOIMENPITEET

5.1 Projektin onnistumisen ja toteutuksen arviointia

Alkuperäinen tarkoitus oli toteuttaa laadun demomittarit yrityksen kahteen eri järjestelmään, mutta tuntikirjaussovelluksen mittarista luovuttiin, koska sovelluksella ei ole suoraa yhteyttä vaatimustenhallintajärjestelmään. Suunnitteluvaiheessa oli tiedostettu mahdollinen järjestelmien yhteensopimattomuusongelma, joten ratkaisuna tuntikirjaussovelluksen mittari hylättiin, kun asialle saatiin varmuus.

Demomittareiden kehittämisen lisäksi tavoitteena oli etsiä muita mittauskohteita, joita käyttämällä voitaisiin tarkkailla ohjelmistotuotteiden laadun kehittymistä. Myös automaattista laadun mittaamista tukittiin tämän projektin aikana.

Projektin tekninen toteutus oli hetkittäin vaikeaa, koska StarTeam-välineen ohjelmointirajapinnoista löytyi vain vähän koodiesimerkkejä, joten ohjelmointityötä oli tehtävä tutkimalla ja kokeilemalla. Tekninen toteutus vei esimerkkien puutteen vuoksi suunniteltua enemmän aikaa. Tämä turhautti ajoittain, koska apua ei ollut saatavilla mistään lähteestä. Opinnäytetyön tekninen toteutus on herättänyt mielenkiintoa myös toisen projektin sisällä, joten tutkimustyöstä on varmasti apua tulevaisuudessa.

Opinnäytetyön tekeminen oli haastava prosessi, mutta koin sen myös mielenkiintoiseksi tavaksi tutustua alan kirjallisuuteen ja opiskella ohjelmistotuotteiden mittaamista. Kirjallisuutta löytyy valtavasti ja keskeisimpien teoksien valintaan oli käytettävä aikaa. Työn tekemisen mielekkyyttä lisäsi, että koin tekeväni itselleni selvitystyötä mahdolliseen mittaamisen jatkoprojektiin.

Projektin rajaaminen onnistui hyvin ja hyvin aikaisessa vaiheessa opinnäytetyötä minulle oli selvää, että mitä tulen tekemään. Projektin määrittelyvaiheessa keskustelimme useasti yrityksen laatuvastaavan kanssa ja yhteinen ymmärrys projektin sisällöstä muodostui varsin nopeasti.

5.2 Mittareiden jatkokehitys

Tässä luvussa esitetään mittareita, jotka ovat mahdollisia jatkokehityskohteita. Esitetyt mittarit saattavat vaatia muutoksia yrityksen sisäisiin prosesseihin. Sisäisten prosessien korjausehdotuksia ei esitetä, vaan kuvataan pelkästään ne ongelmat, jotka estävät mittareiden toteuttamisen.

Mahdollinen mittaridemon jatkokehitys tehdään erillisenä projektina. Projektin resursointi tehdään uuden määrittelytyön ja projektisuunnitelman pohjalta.

5.2.1 Ongelmalliset komponentit

Mitkä komponentit (moduulit) työllistävät eniten tuotteiden sisäisesti? Tästä mittarista saataisiin esille, mille komponentille kohdistuu eniten työtunteja. Ongelmana on se, että komponentin nimen voi syöttää StarTeamiin käsin (ks. kuva 8, component), joten komponentin hakeminen ei ole luotettavaa. StarTeamissa pitäisi olla ominaisuus, jolla CR:ien syöttö tapahtuisi siten, että komponentit valittaisiin ennalta laaditusta listasta. Täten päästäisiin tilanteeseen, jossa komponenttien nimiä käyttämällä voitaisiin suorittaa hakuja.

Mittari olisi hyödyllinen, kun arvioidaan ja tarkastellaan tuotteen sisältä löytyviä ongelmakohtia. Ohjelmistoissa usein suuri osa logiikkaa keskittyy johonkin komponenttiin, ja komponentista voi tulla ongelmallinen, jos se on suunniteltu huonosti. Suuri määrä logiikkaa tarkoittaa sitä, että komponentti sisältää myös suuren määrän ohjelmakoodia ja yhä useampi toiminnallisuus kohdistuu tähän komponenttiin. Mittarin pohjalta nähtäisiin ne komponentit, joille kohdistuu eniten muutosvaatimuksia, ja voitaisiin miettiä, voitaisiinko komponentti suunnitella uudelleen tai mahdollisesti jakaa toiminnallisuutta joillekin muille komponenteille.

5.2.2 Tehokkuuden mittaaminen

Tehokkuuden mittaamiseksi täytyy tarkastella johonkin projektin tehtävään kuluvaa aikaa ja verrata sitä siihen aikaan, joka sen tekemiseen on alun perin arvioitu menevän. Procompin käytössä olevalla tuntienseurantajärjestelmällä ei ole suoraa yhteyttä vaatimustenhallintajärjestelmään.

Tehokkuuden mittaamiseksi täytyisi luoda yhtenäinen kokonaisuus syötettävien tehtävien ja näitä seuraavan tuntikirjauksen välille. Näin projektin eri vaiheita ja vaihekokonaisuuksia pystyttäisiin seuraamaan paremmin. Työmäärien seuraaminen helpottuisi ja ongelmallisista paikoista voitaisiin oppia paremmin. Nykyisen käytäntöjen mukaan projektien tarkastelu jälkikäteen voi olla vaikeaa ja ehkä osittain mahdotontakin.

5.3 Loppusanat

Projekti sai alkunsa laadun mittaamisen tarpeesta ja projektin lopputuotteena syntyi mittari, joka toimii esimerkkinä tuotteiden laadun mittaamiselle. Lopputyön tavoitteena oli toteuttaa demomittari, jolla saadaan tietoa ohjelmistotuotteen tai projektin tilasta tietyllä aikavälillä.

Koin opinnäytetyön monivaiheisena prosessina, joka sisälsi mm. ohjelmointia, mittauksen teoriaa, omiin ympäristöihin tutustumista ja kaikkien näiden soveltamista yhteen. Kukin vaihe asetti oman haasteensa työlle, mutta hyvin rajattuna mistään osasta ei muodostunut liian suurta tai haastavaa palaa ja työ saatiin vietyä kokonaisuudessaan päätökseen. Kirjoitusprosessi vei aikaa suunniteltua enemmän, koska koin kirjallisiin lähteisiin tutustumisen työlääksi ja joskus niitä lukiessa eksyin osittain sivuun varsinaisesta aiheesta.

Opinnäytetyön aikana todettiin, että käytetyn vaatimustenhallintavälineen kehittämisen avulla olisi mahdollista lisätä raporttien luotettavuutta. Myös eri järjestelmien syvempi integraatio helpottaisi tiedon hyödyntämistä.

Demomittarin ideointi tehtiin yhdessä yrityksen laatuvastaavan kanssa. Yrityksen edustajan näkemykset ja mielipiteet olivat tärkeitä ja kannustavia projektin aikana. Myös yritysjohto osoitti kiinnostustaan laadun mittaamiseen ja mahdollisiin jatkokehitysprojekteihin.

Demomittarin ohella on tärkeää keskustella siitä, millaisia muita laatumittareita tarvittaisiin. Laadun mittaaminen on tärkeä osa yrityksen laadunvalvontaa. Tom DeMarco (1982) on sanonut, ”Et voi kontrolloida sitä, mitä et voi mitata.” Tähän mielipiteeseen on helppo yhtyä tämän projektin tekemisen myötä. Laatumittareilla tullee näin ollen olemaan paikkansa osana Procompin laaduntarkkailua.

LÄHTEET

Agile Manifesti 2001. *Manifesto for Agile Software Development* [verkkosivu]. [viitattu16.3.2010]. Saatavissa: http://agilemanifesto.org.

Agile Manifestin Periaatteet 2001. *Principles behind the Agile Manifesto* [verkkosivu]. [viitattu 2.3.2011]. Saatavissa: http://www.agilemanifesto.org/principles.html.

Albrecht, A.J. 1979. *Measuring Application Development.* Proceedings of IBM Applications Development Joint SHARE/GUIDE Symposium. Monterey, CA.

COCOMO II, 1991. *Constructive Cost Model II.* Etelä-Kalifornian yliopisto [verkkosivu]. [viitattu 30.1.2011]
Saatavissa: http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html.

DeMarco, T. 1982. *Controlling Software Projects.* New York: Yourdon Press.

Farrell-Vinay P. 2008. *Manage Software Testing.* Boca Raton: Taylor & Francis Group, LLC.

Fenton, N. E., Pfleeger S.L. 1997. *Software Metrics: A Rigorous & Practical Approach.* London: International Thomson Publishing Europe.

Grönroos C. 2009. *Palvelujen johtaminen ja markkinointi.* Juva: WS Bookwell Oy.

Hokkanen, S., Strömberg, O. 2006. *Laatuun johtaminen*. Jyväskylä: Sho Business Development Oy.

IEEE. 1988. *IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software*. *IEEE Std 982.2*. New York:IEEE.

ISO 9001. 2008. *ISO 9001:2008.* Helsinki: Suomen Standardoimisliitto.

Java Development Kit. 2011. [verkkosivu]. [viitattu 15.2.2011].
Saatavissa: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

JFreeChart. 2011. [verkkosivu]. [viitattu 15.2.2011].
Saatavissa: http://www.jfree.org/jfreechart/.

Juran Instituutin perustaja. 2011. [verkkosivu]. [viitattu 6.6.2011].
Saatavissa: http://www.juran.com/about_juran_institute_our_founder.html.

Kan, S.H. 2003. *Metrics and Models in Software Quality Engineering.* Boston: Addison-Wesley.

Kankkunen K., Matikainen E., Lehtinen L. 2006. *Mittareilla menestykseen*. Helsinki: Talentum Media Oy.

Kipponen, T. 2005. *Ohjelmiston laadun parantaminen puutemittareiden avulla.* Pro gradu -tutkielma. Joensuu: Tietojenkäsittelyn laitos. Joensuun yliopisto.
Saatavissa: ftp://ftp.cs.joensuu.fi/pub/Theses/2005_MSc_Kipponen_Teemu.pdf.

Krishnamurthy N., Saran A. 2008. *Building Software, A Practitioner's Guide.* New York: Taylor & Francis Group.

Lecklin O. 2006. *Laatu yrityksen menestystekijänä. 5. uudistettu painos.* Hämeenlinna: Talentum Media Oy.

Lillrank P. 1998. *Laatuajattelu*. Helsinki: Otava.

NetBeans. 2011. [verkkosivu]. [viitattu 15.2.2011].
Saatavissa: http://netbeans.org/.

Procomp Solutions Oy. 2011. [verkkosivu]. [viitattu 15.2.2011].
Saatavissa: http://www.procomp.fi/.

QSL. 2011. *The history of the international standards organization.* [verkkosivu]. [viitattu 27.5.2011].
Saatavissa: http://www.isoqsltd.com/html/iso_history.html.

Sarala U., Sarala A. 1996. *Oppiva organisaatio*. Tampere: Helsingin yliopiston Lahden tutkimus- ja koulutuskeskus.

SFS. 2010. *ISO 9000 standardisarja.* [verkkosivu]. [viitattu 12.1.2010]. Saatavissa: http://www.sfs.fi/iso9000/.

Sommerville I. 2001. *Software Engineering.* Harlow: Pearson Education Limited.

StarTeam. 2011. [verkkosivu]. [viitattu 15.2.2011]. Saatavissa: http://www.borland.com/us/products/starteam/.

StarTeam Software Development Kit. 2011. [verkkosivu]. [viitattu 15.2.2011. Saatavissa: http://techpubs.borland.com/starteam/2009/en/sdk_documentation/guide/index.html.

Voutilainen A. 2009. *Procomp laatukäsikirja. Versio 1.0.* Kuopio: Procomp Solutions Oy. Ei julkaistu.

www.savonia.fi

Savonia
ammattikorkeakoulu