Pasi Kiviluoto

# Vaatimusmäärittely ja vaatimusten priorisointi ohjelmistoprojekteissa

OPINNÄYTETYÖ

Kevät 2013

Tekniikan yksikkö, Seinäjoki

Teknologiaosaamisen johtamisen koulutusohjelma, YAMK

Seinäjoen ammattikorkeakoulu
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

## Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö, Seinäjoki

Koulutusohjelma: Teknologiaosaamisen johtamisen koulutusohjelma

Tekijä: Kiviluoto, Pasi

Työn nimi: Vaatimusmäärittely ja vaatimusten priorisointi ohjelmistoprojekteissa

Ohjaaja: Lahti, Markku

Vuosi: 2013 Sivumäärä: 78 Liitteiden lukumäärä: 0

Tämän opinnäytetyön aiheena on tutkia vaatimusmäärittelyprosessia ohjelmistoprojektissa, ja tarkoituksena on löytää ratkaisuja kohdeyrityksen, Finn-Power Oy:n, ongelmiin ohjelmistoprojekteissa. Yrityksessä ei ole käytössä sellaista vaatimustenhallintaohjelmistoa, joka olisi koko organisaation saatavilla, ja tämä hankaloittaa vaatimustenhallintaa sekä koko projektin sujumista. Eri sidosryhmiltä tulleita vaatimuksia ei koota hallitusti talteen, eikä ole selvää sääntöä siitä miten vaatimuksia tulisi priorisoida. Yritys haluaa kehittää ohjelmistokehitysprosessia, jotta projekteista tulisi helpommin hallittavia ja sujuvia.

Tutkimuksessa perehdytään ohjelmiston vaatimusmäärittelyprosessiin teorian, analyysin, kirjallisuuden ja aikaisempien tutkimusten avulla. Tutkimusmenetelmänä käytetään konstruktiivista eli soveltavaa tutkimusmenetelmää, joka soveltuu tähän työhön hyvin, sillä työssä kehitetään uusia ratkaisuja ohjelmistokehityksen vaatimusmäärittelyn ongelmiin. Yrityksen tarvitseman ohjelmiston valintaa varten perehdytään eri vaatimustenhallintaohjelmistoihin vertailemalla niiden ominaisuuksia.

Tutkimuksen johtopäätöksenä voidaan todeta, että yksinkertaiset ratkaisut ovat usein parhaita. Vaatimusten priorisointia varten kehitettiin pisteytysjärjestelmä, käytännössä yksinkertainen Excel-taulukko. Taulukon avulla vaatimuksia voidaan laittaa tärkeysjärjestykseen järkevällä ja oikeudenmukaisella tavalla. Parhaiten yrityksen tarpeisiin sopivaksi vaatimustenhallintajärjestelmäksi todettiin Atlassian JIRA. Koska JIRA oli yrityksessä käytössä projektinhallintatyökaluna jo valmiiksi, koitui siitä yritykselle huimat säästöt, sillä ei ollut tarvetta investoida uuteen ohjelmistoon. Vaatimusten hallittu kerääminen ratkaistiin keskittämällä niiden kerääminen vain tietyille henkilöille. Kaikki vaatimukset, myös hylätyt, tallennetaan JIRA-järjestelmään.

Avainsanat: vaatimusmäärittely, vaatimusten priorisointi, vaatimustenhallintaohjelmisto

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

## Thesis abstract

Faculty: School of Technology

Degree programme: Master´s Degree in Technology Competence Management

Author: Kiviluoto, Pasi

Title of thesis: Requirement specification and prioritization of the software projects

Supervisor: Lahti, Markku

Year: 2013 Number of pages: 78 Number of appendices: 0

The essence of this thesis is to study the requirement analysis process of the software project with the aim of finding solutions to the the problems of the software projects of the target company, Finn-Power Oy. The requirements are not collected in a controlled manner, and there is no clear rule for the prioritization. The company wants to develop its software development process, so that the projects would be more manageable and smooth. The aim was also to find the suitable requirement management software for Finn-Power Oy.

The thesis examines the software requirement specification process through the theory, analysis, literature and previous studies. The research method used was the constructive research method. The software for the company was examined by comparing the features of different software.

The study found that the simplest solutions are often the best. For the requirement prioritization a prioritization system was developed, a simple Excel spreadsheet, which allows the requirements to be put in order of importance in a rational and fair way. The most suitable requirement management software for the company was found to be Atlassian JIRA. Since JIRA already was in use in the company as a project management tool, there was no need to invest in new software. The requirement of the centralized collection was solved by concentrating their collection only on few certain people. In the future, all of the requirements, also the ones which got rejected, are stored in JIRA. Atlassian JIRA therefore proved to be a very good and flexible tool for the company's wide range of needs.

Keywords: requirement engineering, requirement prioritization, requirements management tool

# SISÄLTÖ

## Kuvio- ja taulukkoluettelo

## Käytetyt termit ja lyhenteet

| | |
|---|---|
| **Asiakas** | Projektissa valmistuvan tuotteen tai ohjelmiston tilaaja, käyttäjä ja vastaanottaja. |
| **FMS** | Joustava valmistusjärjestelmä. FMS on lyhenne sanoista Flexible Manufacturing System. |
| **IEEE** | The Institute of Electrical and Electronic Engineers on kansainvälinen tekniikanalan järjestö, joka mm. ylläpitää standardeja. |
| **JIRA** | Web-pohjainen projektinhallintaohjelmisto. |
| **Käyttäjä** | Henkilö, joka käyttää projektista syntynyttä tuotetta. |
| **Ohjausryhmä** | Ohjausryhmän tehtävänä on huolehtia projektien käynnistyksestä, ohjauksesta, koordinoinnista ja seurannasta. Ryhmään kuuluvat yrityksen eri osastojen päälliköt. |
| **Sidosryhmä** | Henkilö, ryhmä tai organisaatio, joka on osallisena projektissa. |
| **SVN** | Versiohallintajärjestelmä |
| **Tekninen määrittely** | Kuvaa tarkasti ohjelmiston teknisen arkkitehtuurin, mutta ei sisällä ohjelmistokoodia. |
| **Tulus®** | Prima Power -ohjelmistojen rekisteröity tuotemerkki. |
| **UML** | Mallinnuskieli. UML on lyhennys sanoista Unified Modeling Language. |
| **URL-osoite** | Yksilöllinen osoite internetissä olevalle tiedostolle. URL on lyhenne sanoista Uniform Resource Locator. |
| **Vaatimusmäärittely** | Prosessi, joka kattaa kaikki projektin elinkaaren liittyvät toimet. |

| | |
|---|---|
| **Vaatimustenhallinta** | Prosessi, jossa vaatimusmäärittelyssä esitetyt ominaisuudet viedään kauttaaltaan läpi kehitysprosessissa ja sen operatiivisen ajan. |
| **VPN** | Virtuaalinen erillisverkko. VPN on lyhennys sanoista Virtual Private Network. |

# 1 JOHDANTO

## 1.1 Työn tausta

Nykyään ohjelmistojen asema konejärjestelmien myynnissä ja toimituksissa on korostunut. Koska koneiden mekaaniset ratkaisut ovat hyvin lähellä toisiaan kaikilla konetoimittajilla, eroja saadaan järjestelmän älykkyydellä. Asiakkaalle on tärkeää, että hänen hankkimansa järjestelmä valmistaa tuotteet mahdollisimman vähin kustannuksin sekä nopeasti ja vähällä työvoimalla.

Ohjelmistokehityksessä on tärkeää tietää asiakkaan todelliset tarpeet ja vaatimukset. Vain tällä tavoin voidaan varmistaa, että asiakas todella saa haluamansa tuotteen ja toiminnallisuudet. Tällä tavoin myös reklamaatioiden määrä vähenee. Kun vaatimustenhallinta on organisoitu oikein, työnteko tehostuu.

Vaatimustenhallintaa kehittämällä pystytään paremmin suunnittelemaan projekteja ja hallitsemaan resursseja. Kun vaatimukset ovat selvillä hyvissä ajoin, voidaan oikealla priorisoinnilla jakaa resurssit tasaisesti. Näin projektin voi helposti ohjata sellaiselle henkilölle, jolla on tämän aiheen osaamista ja tietotaitoa. Henkilöiden resurssit voidaan perustellusti varata ja aikatauluttaa tarkemmin. Puutteellinen vaatimusmäärittelyprosessi aiheuttaa resurssien epätasaisen kuormittumisen ja tekee aikataulusta tiukan. Tällä tutkimuksella pyritään löytämään ratkaisu tähän ongelmaan.

Kun vaatimukset on kerätty, täytyy niitä pystyä keskitetysti hallinnoimaan ja niiden tilaa seuraamaan. Tätä varten tarvitaan ohjelmisto, jolla vaatimuksia hallitaan koko niiden elinkaaren ajan. Elinkaarella tarkoitetaan tässä tapauksessa koko ohjelmistokehityksen elinkaarta; vaatimuksesta valmiiksi ohjelmistotuotteeksi asti.

## 1.2 Opinnäytetyön tavoite ja rajaus

Tässä opinnäytetyössä tutkimuksen kohteena olevassa kohdeyrityksessä ei ole suunniteltua ja yhteisesti sovittua vaatimustenhallintajärjestelmää ohjelmistoprojekteissa. Vaatimuksia esittää moni eri taho, kuten myynti, koulutus, käyntiinajo ja

huolto. Kohdeyrityksessä ei ole tällä hetkellä käytössä mitään virallista toimintatapaa toteutusjärjestyksen määrittämiseksi. Toimeen ryhdytään vaikka faktat puuttuvat, ja tästä aiheutuu väärinkäsityksiä ja aikatauluongelmia. Myös resurssien varaaminen oikeaan aikaan on vaikeaa.

Asiakasprojekteissa ongelma saattaa ilmetä niin, että vasta lähellä toimitusta huomataan, että jokin tietty ominaisuus ei toimi oikealla tavalla tai jopa puuttuu kokonaan. Myyjät eivät kerro suunnitteluun riittävän tarkkoja määrityksiä siitä, mitä asiakas on todella tilannut. Vaatimusmäärittelyyn täytyy suunnitella ja toteuttaa prosessi, jolla asiakkaalta tulleet vaatimukset toimitetaan hallitusti suunnitteluun. Tällainen prosessi voi olla esimerkiksi myyjää varten laadittu tarkastuslista, jonka avulla myyjää ohjataan huomioimaan kaikki tarvittavat asiat. Myyntitilanteessa tehdyt virheet pienentävät katetta ja aiheuttavat laatuvirheitä, jotka voitaisiin välttää huolellisella vaatimusmäärittelyllä. Jos vaatimusmäärittelyä ei ole tehty kunnolla, oikean osaamistaustan omaavaa suunnittelijaa ei ole varattu toteuttamaan projektia ja tällöin aikaa ja rahaa kuluu toisen henkilön kouluttamiseen. Epävarmuus tulevista töistä vaikuttaa negatiivisesti henkilöiden motivaatioon ja työkuormaan. Jos henkilö joutuu tekemään mielestään turhia töitä, työn mielekkyys kärsii.

Uusien ohjelmisto-ominaisuuksien osalta on ongelmana, että vaatimuksia tulee monelta eri taholta ja jokaisen niistä toteuttamisessa on yleensä kiire. Ongelmana on myös se, että uusien vaatimusten toteuttamisen kannattavuutta ei tutkita tarkemmin.

Yksi vaihtoehto vaatimusmäärittelyn kehittämiseksi on toteuttaa ideapankki, jonne uudet vaatimukset kerätään. Tulleet vaatimukset pisteytetään ja priorisoidaan näiden annettujen pisteiden perusteella. Priorisoinnissa täytyy olla mukana henkilöitä yrityksen eri osastoilta.

Kun vaatimukset on dokumentoitu ja ne on päätetty toteuttaa, niiden säilyttäminen kaikkine tietoineen täytyy hallita. Myös ne ideat, jotka eivät ole olleet toteutuskelpoisia, säilytetään. Lisäksi päätöksen, miksi ideaa ei toteuteta, tulee jäädä näkyviin. Näin voidaan jälkeenpäin todeta, miksi jokin ominaisuus jätettiin tekemättä. On mahdollista, että hylätyt ominaisuudet tullaan myöhemmin tekemään, kun ajat

ja tarpeet muuttuvat. Kaiken tämän tietomassan hallitsemiseen tarvitaan toimiva järjestelmä.

Tämän opinnäytetyön tutkimusongelmana on selvittää, kuinka vaatimustenhallinta tulee toteuttaa, jotta se parantaa toimitusten ja ohjelmistotuotteiden laatua. Tutkimuskysymyksinä ovat:

1. Miten vaatimukset saadaan kerättyä sidosryhmiltä hallitusti talteen ja siirrettyä ne suunnitteluprosessiin?
2. Miten nämä vaatimukset priorisoidaan järkevällä ja oikeudenmukaisella tavalla?
3. Missä järjestelmässä vaatimukset säilytetään ja miten siitä saadaan avoin koko organisaatiolle?

Tässä työssä keskitytään vaatimusmäärittelyyn ja vaatimustenhallintaan ohjelmistoprojekteissa sekä tuotekehitys- tai asiakasprojekteissa. Työssä ei kuitenkaan käydä tarkasti läpi eri vaatimustenhallintatekniikoita, vaan tutkitaan mikä olisi paras ratkaisu kohdeyrityksen näkökulmasta.

### 1.3 Tutkimusmenetelmä

Tässä opinnäytetyössä käytetään konstruktiivista eli soveltavaa tutkimusmenetelmää. Konstruktiivinen tutkimus vastaa mm. rakentamisen, käyttöönoton, sovittamisen ja muutostoimenpiteiden arvioinnin kysymyksiin. Konstruktiivisessa tutkimuksessa on aina päätettävä, millainen lopputulos halutaan rakentaa. Tämä riippuu hyvin paljon päätöksentekijöiden arvoista. Tuloksen ei tarvitse aina olla konkreettinen tuote, vaan se voi olla myös prototyyppi tai esimerkiksi pelkkä suunnitelma. Tärkeää on, että tutkimuksen lopputulosta täytyy kuitenkin aina voida arvioida. (Järvinen & Järvinen 2000, 102.)

Konstruktiivinen tutkimusmenetelmä soveltuu hyvin tähän työhön, sillä työssä kehitetään uusia ratkaisuja ohjelmistokehityksen vaatimusmäärittelyn ongelmiin, käyttäen tukena jo olemassa olevaa teoriaa.

### 1.4 Työn rakenne

Tutkielma koostuu johdannosta, kohdeyrityksen esittelystä, teoriasta, tutkimuksesta ja yhteenvedosta. Johdannossa esitellään opinnäytetyön taustaa ja esitellään tutkimuskysymykset sekä tutkimuksen rakennetta.

Kohdeyrityksen esittelyn jälkeen perehdytään ohjelmistokehityksen teoriaan. Ohjelmistokehitystä tarkastellaan myös kohdeyrityksen näkökulmasta ja esitellään yrityksen ohjelmistotuotteita.

Teoriaosuus on jaettu kolmeen isompaan osaan: vaatimusmäärittely, vaatimusten priorisointi ja vaatimustenhallintaohjelmistot eli keskitytään tutkimuskysymyksiin liittyviin aiheisiin. Lisäksi on myös muuta teoriaa, joka tukee tätä tutkimusaihetta.

Tutkimusosuudessa kerättyä aineistoa analysoidaan ja sovelletaan käytännön ratkaisuihin sekä esitellään ratkaisuja tutkimusongelmiin.

# 2 KOHDEYRITYKSEN ESITTELY

Kohdeyrityksenä tässä opinnäytetyössä on Finn-Power Oy. Yrityksen perusti Jorma Lillbacka vuonna 1969. Yritys oli silloiselta nimeltä Lillbackan Konepaja. Vuonna 2002 yritys ja sen myötä koko levytyötekniikkaan keskittynyt osa Lillbacka-yhtiöistä siirtyi EQT-sijoitusrahaston omistukseen. Yrityskaupan jälkeen myös yrityksen nimi muutettiin Finn-Power Oy:ksi. Vuonna 2008 EQT-sijoitusrahasto myi Finn-Powerin koko osakekannan italialaiselle Prima Industrielle. Koko konsernin palveluksessa on noin 1300 henkilöä. Konsernin liikevaihto vuonna 2012 oli 349 m€. Prima Industrie -konserni käyttää tuotekehitykseen noin 5 - 6 % liikevaihdosta ja koko konsernissa työskentelee noin 90 suunnittelijaa.

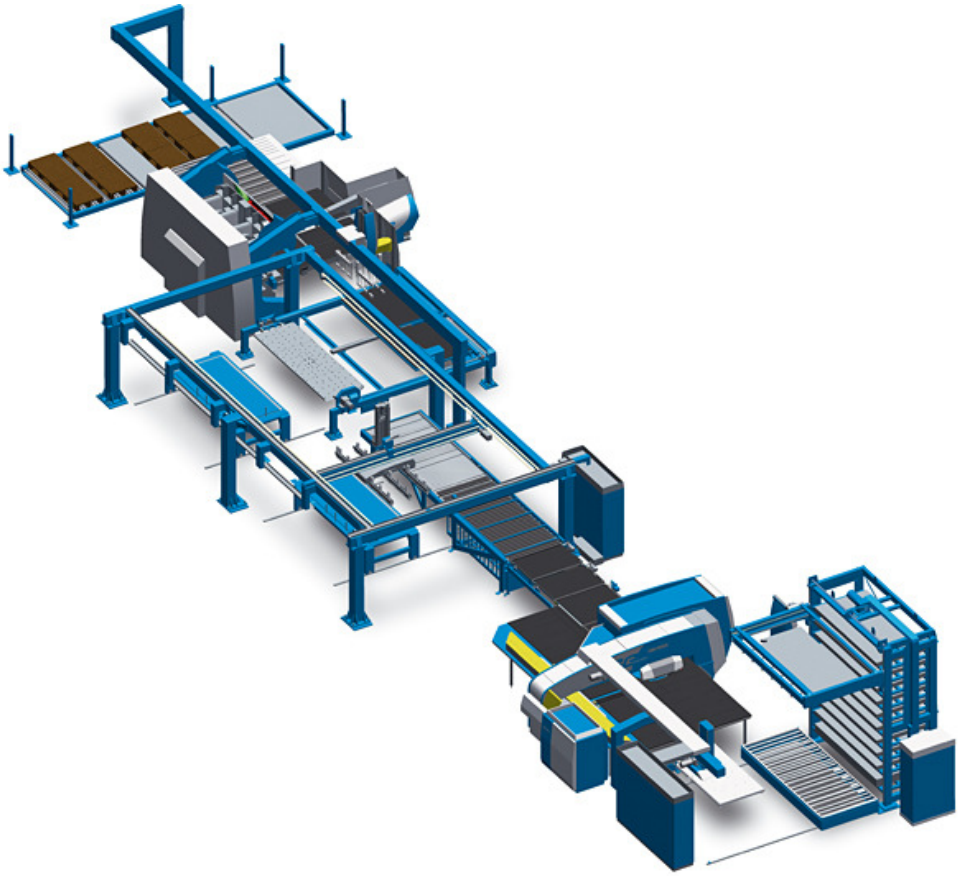

Kuvio 1. Finn-Power Oy

Finn-Power valmistaa työstökoneita ja valmistusjärjestelmiä, joilla ohutlevyä lävistetään, leikataan ja taivutetaan. Toimitusvalikoimaan kuuluu järjestelmiä kaikille automaatiotasoille, itsenäisistä levytyökeskuksista aina tehdaslaajuisiin joustaviin valmistusjärjestelmiin saakka. Tuotevalikoimasta löytyy koneita, joissa on pelkkä lävistys (kuvio 2) tai yhdistelmäkoneita, joissa on integroituna lävistys ja kulmaleikkuri (kuvio 3) tai lävistys ja laser (kuvio 4). Näitä koneita voidaan liittää erilaisiin automaattisiin varastojärjestelmiin, joko pelkkään materiaalivarastoon (kuvio 5) tai sitten FMS-järjestelmään (Flexible Manufacturing System) (kuvio 6). FMS-järjestelmässä materiaalit tulevat automaattisesti koneelle ja valmiit kappaleet siirretään automaattisesti varastoon.

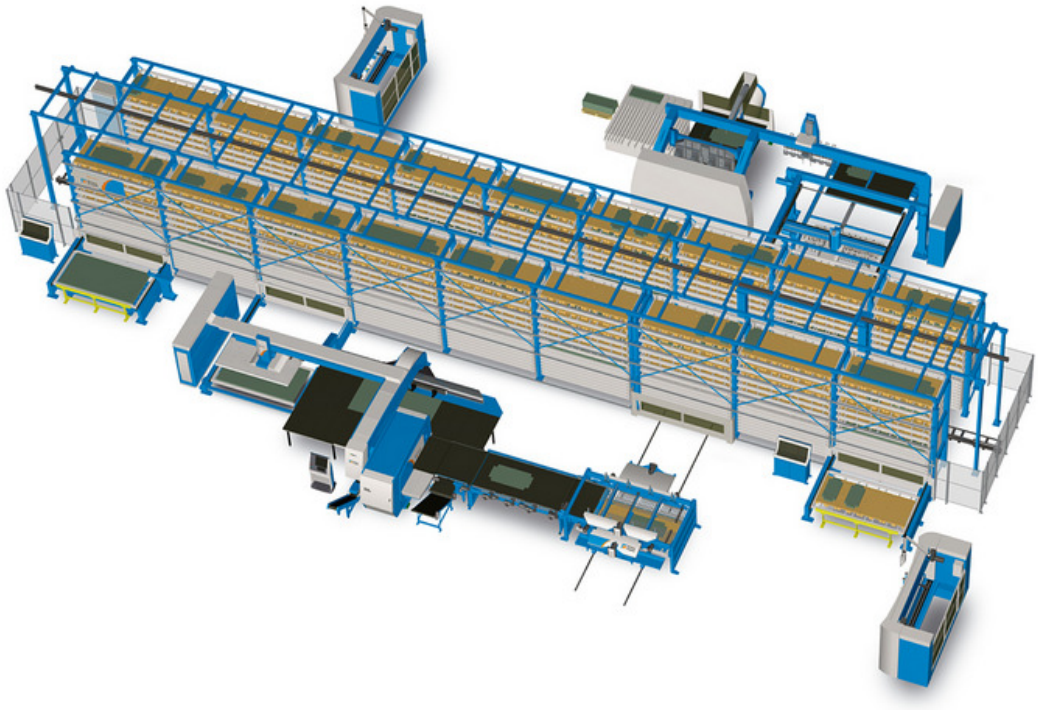

Kuvio 2. Prima Power -lävistyskone

Kuvio 3. Prima Power -lävistys- ja kulmaleikkurikone

Kuvio 4. Prima Power -lävistys- ja laserleikkauskone



Kuvio 5. Taivutusautomaattilinja materiaalivarastoliitynnällä



Kuvio 6. Prima Power Night Train FMS -automaattivarasto

# 3 OHJELMISTOKEHITYS

## 3.1 Ohjelmistokehityksen historiaa

Ohjelmistotuotanto on moniin teollisuudenaloihin verrattuna varsin nuori ala. Alkunsa se on saanut 1960-luvulla, jolloin ohjelmistoja alettiin kehittää laajamittaisesti. Jo samalla vuosikymmenellä huomattiin, että ohjelmistojen kasvaessa ja monimutkaistuessa projektien ennustettavuus alkoi olla hälyttävän huono. Tämä antoi sysäyksen systemaattisten ohjelmistotuotantomenetelmien kehittämiselle. (Haikala & Märijärvi 2002, 26.)

Sovellusten ymmärtäminen ja toteuttaminen on kehittynyt yksinkertaisien välineiden kautta kohti laajempia kokonaisuuksia. Alussa sovellukset olivat lähinnä rekistereihin ja muistipaikkoihin tallennettujen lukujen ja merkkijonojen käsittelijöitä. Tämän jälkeen alettiin kehittää rakenteellisia tietotyyppejä. Sitten ohjelmointikielissä alettiin käyttää aliohjelmia ja myöhemmin kehitettiin tietoabstraktioita eri muodoissa. (Koskimies & Mikkonen, 2005, 15-16.)

Seuraavaksi havaittiin tarve suurempien kokonaisuuksien muodostamiselle loogisesti yhteenkuuluvista palveluista, jotka esiteltiin tietyn rajapinnan toteuttavana komponentteina. Tämän seurauksena sovelluksissa alettiin käyttää samoja komponentteja ja komponenttikirjastoja. Ohjelmistoalustat kehittyivät, kun huomattiin että ohjelmistoilla saattoi olla samankaltainen perusarkkitehtuuri. Sovellukset saattoivat olla keskenään samankaltaisia, joko rakenteen tai toiminnallisuuden kannalta. Tästä alettiin tunnistaa sovellus- tai tuoteperheitä, joita varten olioparadigman sisällä kehitettiin sovelluskehyksen käsite. (Koskimies & Mikkonen, 2005, 16.)

## 3.2 Ohjelmistokehityksen elinkaari ja prosessimallit

Projektin suunnitteluprosessin perustana ovat vaatimukset. Niiden pohjalta valitaan sopiva ohjelmiston kehityskaari sekä arvioidaan resurssi- ja aikataulutarpeet. Projektia suunniteltaessa saatetaan esimerkiksi huomata, että tietyn ominaisuuden

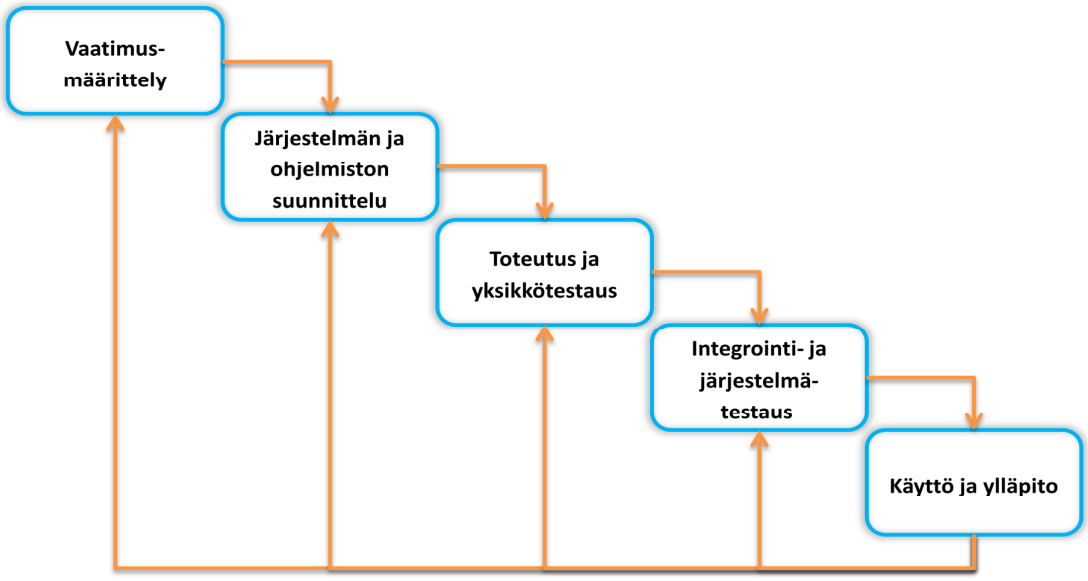
toteutus ei onnistu saatavilla olevilla resursseilla tai halutussa ajassa. Tällöin suunnittelun tuloksena projektin laajuutta saatetaan supistaa tai päädytään julkaisemaan ominaisuuksia vaiheittain. (Wiegers 2003, 383.)

Ohjelmiston prosessimalli on abstrakti kuvaus ohjelmistoprosessista. Jokainen prosessimalli esittää prosessin tietystä näkökulmasta ja tarjoaa siten osittaista tietoa kyseisestä prosessista. (Sommerville 2004, 65.)

Yleisimmät prosessimallit ovat vesiputous-, evoluutio- ja komponenttiperustainen malli (Sommervillen 2004, 65).

### 3.2.1 Vesiputousmalli

Käytössä on erilaisia ohjelmistoprosessimalleja. Käytännössä kaikissa niissä esiintyy jossain muodossa neljä toimintoa (aktiviteettia): määrittely, kehittäminen, vahvistaminen ja evoluutio. Vesiputousmalli (*Waterfall model*, kuvio 7) esittää nämä toiminnot erillisinä peräkkäisinä prosessin vaiheina. Sommervillen (2006, 65) mukaan vesiputousmalli voidaan jakaa viiteen elinkaaren vaiheeseen:




Kuvio 7. Vesiputousmalli

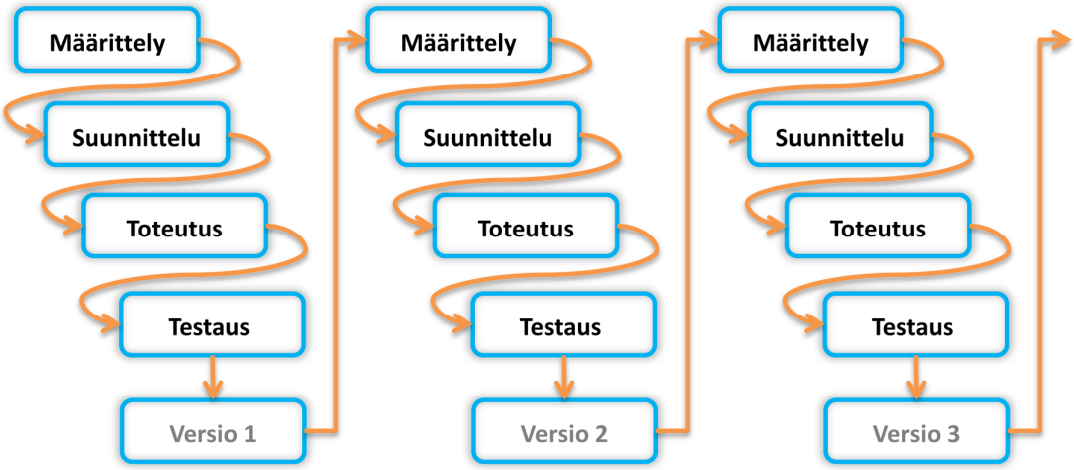
Sommervillen (2006, 67) mukaan vesiputousmalli voidaan jakaa viiteen elinkaarivaiheeseen:

1. Vaatimusmäärityksvaiheessa selvitetään järjestelmän rajoitteet, palvelut ja tavoitteet järjestelmän käyttäjien kanssa ja laaditaan niistä yksityiskohtainen määrittelydokumentti.
2. Järjestelmän ja ohjelmiston suunnitteluvaiheessa rakennetaan yleinen arkkitehtuuri sekä määritellään ja kuvataan ohjelmiston perusabstraktiot ja niiden suhteet.
3. Toteutus- ja yksikkötestausvaiheessa varmistetaan, että ohjelmaryhmät ja -yksiköt toimivat kuten pitääkin.
4. Integrointi- ja järjestelmätestausvaiheessa ohjelmayksiköt integroidaan järjestelmään ja testataan. Testauksen jälkeen järjestelmä toimitetaan asiakkaalle.
5. Käyttö- ja ylläpitovaihe on yleensä ohjelmiston elinkaaren pisin. Tällöin järjestelmä on asennettu ja otettu käyttöön. Tässä vaiheessa korjataan virheitä ja parannetaan palveluja sitä mukaa, kun uusia vaatimuksia löydetään.

Teoriassa jokaisesta vaiheesta tuloksena on yksi tai useampi dokumentti, eikä seuraava vaihe ala ennen kuin edellinen vaihe on valmis. Käytännössä nämä vaiheet etenevät limittäin ja antavat tietoa toisilleen. Suunnittelun aikana paljastuvat vaatimusvaiheessa tehdyt virheet ja ohjelmointivaiheessa paljastuvat suunnitteluvaiheessa tehdyt virheet jne. Ohjelmistoprosessi ei ole yksinkertainen lineaarinen malli, vaan se kehittyy aina toistojen myötä. (Sommerville 2006, 67.)

### 3.2.2 Evoluutiomalli

Evoluutiomallissa *(Evolutionary development,* kuvio 8) prosessin eri vaiheet kuten määrittely, kehittäminen ja vahvistaminen tapahtuvat samanaikaisesti tai ainakin osin rinnakkain. Ensimmäinen järjestelmä on nopeasti kehitetty abstraktista määrittelystä. Tämä järjestelmä jalostetaan asiakkaan palautteen mukaan ja kehitetään asiakkaan tarpeen täyttäväksi järjestelmäksi. (Sommerville 2006, 65.)




Kuvio 8. Evoluutiomalli

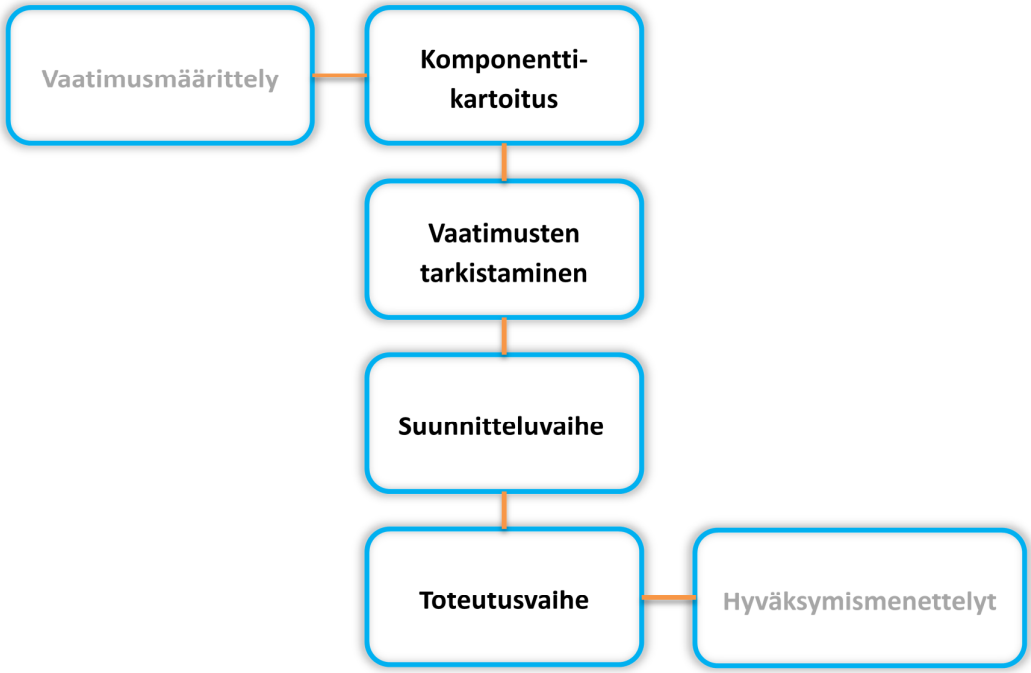
Evoluutiomallissa tuote julkaistaan tarkoituksella ns. raakileena, josta se käyttäjien antamien kommenttien ja monen version myötä jalostuu lopulliseksi tuotteeksi. Evoluutiokehitys voidaan jakaa tutkivan kehityksen tyyppiin ja prototyyppiseen kehitysmalliin. Tutkivan kehityksen malli perustuu asiakkaan kanssa käytävään yhteistyöhön ja sitä kautta selvitettäviin vaatimuksiin aina lopputuotteeseen saakka. Kehitystyö alkaa niiltä osin kuin vaatimukset on ymmärretty ja uusia ominaisuuksia lisätään asiakkaan toivomuksesta. Prototyyppinen kehitysmalli perustuu asiakkaiden vaatimusten ymmärtämiseen ja sitä kautta niiden parantamiseen. Prototyypeillä testataan niitä vaatimuksia, joita ymmärretään huonosti. (Sommerville 2006, 68.)

Evoluutiomallin on todettu olevan ohjelmistojen kehityksessä tehokkaampi kuin vesiputousmallin, sillä evoluutiomallissa asiakkaan välittömät tarpeet tyydytetään heti ja määrittelyt voidaan tehdä vaiheittain, ymmärryksen kasvaessa. (Sommerville 2006, 68-69.)

Tällaisessa kehitysmallissa on myös ongelmansa, esimerkiksi kehityksen mittaaminen on vaikea prosessin ollessa näkymätöntä ja jokaisen ohjelmistoversion dokumentointi on kallista ja työlästä. Jatkuva ohjelmiston muokkaaminen vaurioittaa ohjelmiston rakennetta ja muutoksista tulee kalliita ja vaikeita toteuttaa. (Sommerville 2006, 69.)

### 3.2.3 Komponenttiperustainen malli

Komponenttiperustaiset mallit *(component-based software engineering,* kuvio 9) perustuvat merkittävään määrään olemassa olevia uudelleenkäytettäviä komponentteja. Järjestelmäsuunnittelijat keskittyvät integroimaan näitä komponentteja järjestelmään pikemmin kuin kehittämään niitä tyhjästä. (Sommerville 2006, 65.)




Kuvio 9. Komponenttimalli

Komponenttikartoituksessa pyritään tunnistamaan olemassa olevien komponenttien joukosta ne, joilla vaatimukset pystytään toteuttamaan mahdollisimman yksinkertaisesti. (Sommerville 2006, 70.)

Vaatimuksien oikeellisuus tarkistetaan ja vaatimusten tärkeydestä keskustellaan asiakkaan kanssa. Jos komponenttikartoituksessa ei ole löytynyt sopivia komponentteja vaatimuksen toteuttamiseen, täytyy selvittää uusien komponenttien toteuttamisesta aiheutuvat kustannukset sekä asiakkaan tarpeet ja niiden prioriteetit. Tässä vaiheessa voidaan asiakkaan vaatimuksia muokata tai todeta, että täytyy kehittää täysin uusia komponentteja. (Sommerville 2006, 70.)

Suunnitteluvaiheessa suunnitellaan komponenttien integroinnissa käytettävä runko. Jos sopiva runko on jo olemassa, hyödynnetään se. Samalla suunnitellaan olemassa oleviin komponentteihin mahdollisesti tehtävät parannukset. Tarvittaessa toteutetaan täysin uudet komponentit. (Sommerville 2006, 70.)

Toteutusvaiheessa tehdään komponentteihin liittyvät muutos- ja kehitystyöt. Komponentteja integroimalla muodostetaan asiakkaan vaatimukset täyttävä uusi järjestelmä. (Sommerville 2006, 70.)

Hyväksymismenettelyt ovat komponenttiperustaisen mallin ulkopuolisia toimenpiteitä. Komponenttiperustainen malli ei ota kantaa hyväksymismenettelyjen toteutustapaan. (Sommerville 2006, 70.)

Komponenttiperustaisen mallin edut liittyvät työn kustannuksiin ja riskien hallintaan. Mallin avulla vähennetään ohjelmointityötä ja siitä aiheutuvia kustannuksia. Mallin haittoja ovat vaatimusten tarkistamisvaiheessa tehtävä asiakasvaatimusten kyseenalaistaminen. Tämä saattaa johtaa kompromissiin, jonka seurauksena asiakas saattaa saada huonosti hänen tarpeisiinsa soveltuvan järjestelmän. (Sommerville 2006, 70.)

### 3.3 Sidosryhmät

Projekteissa on mukana useita eri sidosryhmiä, jotka kaikki tulee ottaa huomioon. Wiegers (2003, 4-5) listaa sidosryhmät seuraavasti:

- Asiakkaat, jotka rahoittavat projektin tai hankkivat tuotteen. Tuotteen tulee täyttää organisaation liiketoiminnalliset tavoitteet.
- Käyttäjät, jotka suorasti tai epäsuorasti ovat tekemisissä tuotteen kanssa (asiakkaiden asiakkaat).
- Vaatimusten analysoijat, jotka laativat vaatimukset ja välittävät ne kehittäjille.
- Kehittäjät, jotka suunnittelevat, toteuttavat ja ylläpitävät tuotetta.
- Testaajat, jotka tutkivat toimiiko tuote siten, kuten sen on tarkoitus.
- Dokumenttien kirjoittajat, jotka kirjoittavat mm. käyttöohjeen ja koulutusmateriaalin.

- Projektipäälliköt, jotka suunnittelevat projektin ja pitävät lankoja käsissään, jotta projekti onnistuisi.
- Juristit, jotka pitävät huolta, että tuote on kaikkien asiaankuuluvien lakien ja säännösten mukainen.
- Tuotantoihmiset, jotka valmistavat ne tuotteet, johon ohjelmisto kuuluu.
- Myynti, markkinointi, tuki kentällä, neuvontapiste ja kaikki muut ihmiset, jotka työskentelevät tuotteen ja sen asiakkaiden parissa.

### 3.4 Ohjelmistokehitys kohdeyrityksessä

Ohjelmistokehitys kohdeyrityksessä keskittyy levytyökeskusten käyttöliittymien ja asiakkaan valmistuksen ohjauksen suunnitteluun ja toteutukseen. Tästä opinnäytetyössä ei käsitellä koneiden ohjauksiin tarvittavaa NC/PLC-ohjelmistosuunnittelua.

Ohjelmointityökaluna käytetään Microsoft Visual Studio 2010 -ohjelmistokehitysympäristöä, ohjelmointikielenä on C#. Versiohallintasovelluksena on Subversion eli SVN, jossa kaikki lähdekoodi on tallessa.

Atlassian JIRA-järjestelmällä hallitaan ohjelmistoprojekteja ja testaustoimintaa. JIRA sisältää myös palautejärjestelmän, jonka kautta esimerkiksi käyntiinajo kirjaa ylös ongelmatilanteet ja parannusehdotukset. Kaikkien ohjelmistosuunnittelijoiden työt löytyvät JIRA-järjestelmästä, jolloin töiden seuranta on keskitetty yhteen paikkaan. Sellaista työtä ei ole, jota ei tähän järjestelmään olisi kirjattu. Myös testauksessa löytyneet virheet kirjataan samaan järjestelmään, jolloin niiden siirtäminen tehtäviksi onnistuu helposti. JIRA on web-pohjainen sovellus ja on helposti räätälöitävissä ja laajennettavissa eri toimintoihin.

Koneiden käyttöliittymät on toteutettu siten, että ohjelmistosta on aina vain yksi versio, joka tukee kaikkia eri valmistusteknologioita. Näin lävistys-, laser- ja taivutusteknologioita käyttäviin koneisiin voidaan asentaa sama käyttöliittymä. Eri ohjelmistokehityshaaroja ei tarvita, ja uudet ominaisuudet tulevat samalla kerralla kaikkiin koneisiin. Ohjelmistokehityshaara tarkoittaa, ettei ohjelmistoista ole montaa eri konemallista riippuvaa versiota. Tämä lisää haasteita vaatimustenhallintaan

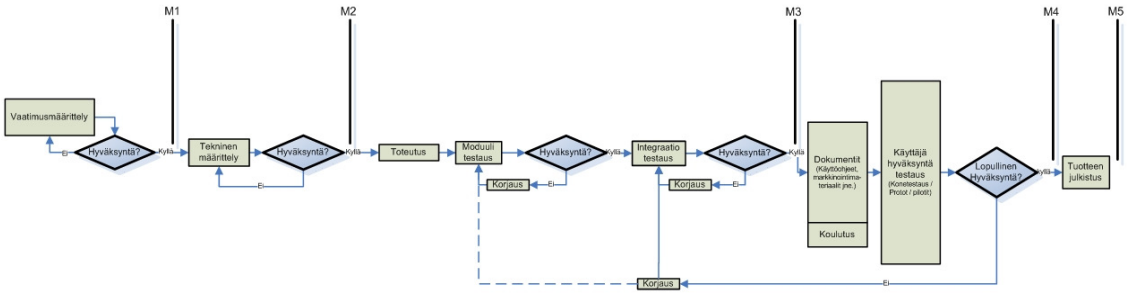
ja suunnitteluun. Töiden aikataulutus ja ohjelman versiointi on haastavaa, koska eri teknologioita käyttäviä koneita toimitetaan asiakkaille jatkuvasti. Konemalleja kehitetään eri aikaan, joten samoja vaatimuksia ei pystytä toteuttamaan heti kaikkiin konemalleihin.

Ohjelmistojen julkaisuaikataulut määritetään etukäteen noin kahden vuoden ajanjaksolle. Tähän aikatauluun määritellään isoimmat uudet ominaisuudet, jotka ovat tiedossa hyvissä ajoin. Näitä ovat esimerkiksi uusien konemallien tai lisälaitteiden julkistaminen. Aikatauluun määritellään myös joitain koneista riippumattomia, uusia ohjelmisto-ominaisuuksia, jotka on todettu tarpeellisiksi. Koska resurssit ovat rajalliset, korostuu tässä uusien ominaisuuksien priorisoinnin tärkeys.

### 3.5 Ohjelmiston elinkaari kohdeyrityksessä

Kohdeyrityksessä ohjelmiston elinkaari perustuu vesiputousmalliin, ja samalla sovelletaan komponenttimallia. Uuden ominaisuuden kanssa pyritään aina hyödyntämään jo valmiita komponentteja ja niitä laajennetaan tarvittaessa. Tässä kappaleessa ei syvennytä elinkaarimallin yksityiskohtiin, koska opinnäytetyö keskittyy vain osaan tätä mallia eli vaatimusmäärittelyyn. Jotta kokonaisuus on helpompi hahmottaa, on elinkaarimallin prosessia kuitenkin syytä hieman selvittää.

Kuviossa 10 on kuvattu elinkaarimallin prosessin kulku vaatimusmäärittelystä alkaen. Kuviosta on poistettu joitain vaiheita, jotka toteutetaan jo ennen vaatimusmäärittelyä, kuten esitutkimus ja esisuunnittelu. Riippuen projektin suuruudesta nämä vaiheet eivät ole aina käytössä. Jos kyseessä on pieni projekti, mikä ei vaadi näitä toimenpiteitä, riittää kuviossa kuvattu prosessimalli.




Kuvio 10. Kohdeyrityksen ohjelmiston elinkaarimalli

Uuden projektin suunnitteluprosessi aloitetaan esitutkimuksella. Esitutkimuksessa selvitetään projektin sisältö ja kartoitetaan projektin tarve. Tämän jälkeen tehdään ennakkosuunnittelu, jossa sisältöä kootaan tarkemmin sekä laaditaan dokumentit. Dokumenttien avulla projekti esitellään ryhmälle, joka lopulta antaa luvan projektin toteutukselle. Nämä vaiheet ovat käytössä projektin ollessa niin iso, että se vaatii tarkemman selvityksen ja hyväksynnän.

Kun projekti on saanut hyväksynnän, aloitetaan vaatimusmäärittely. Kun vaatimukset on hyväksytty, aloitetaan tekninen määrittely. Tässä vaiheessa tehdään tekninen suunnitelma vaatimuksissa tulleista ominaisuuksista. Molemmista vaiheista on erillinen hyväksymiskäytäntö. Suunnitelmat katselmoidaan erikseen kerätyn ryhmän kanssa. Tässä ryhmässä täytyy olla mukana eri osastojen henkilöitä, että asiasta saadaan mahdollisimman kattava näkemys.

Teknisen määrittelyn jälkeen alkaa ohjelmiston toteutus, jota katselmoidaan määrätyin välein. Ohjelmistosuunnittelija tekee ohjelmistoa teknisen määrittelyn pohjalta. Tähän vaiheeseen kuuluu myös moduulitestaus, jonka suorittaa pääasiassa ohjelmistosuunnittelija. Kun ohjelmisto on valmis, siirrytään integraatiotestaukseen. Integraatiotestausvaiheen suorittaa sellainen henkilö, joka ei ole ollut toteutusvaiheessa mukana. Integraatiotestauksessa uusi ominaisuus liitetään siihen kokonaisuuteen jossa se tulee toimimaan. Tässä vaiheessa ilmenevät yleensä sellaiset viat, joita ei ole osattu ohjelmiston tekovaiheessa ottaa huomioon. Tämä testaus suoritetaan yleensä simulaatioympäristössä, joka on mahdollisimman lähellä oikeaa lopullista ympäristöä.

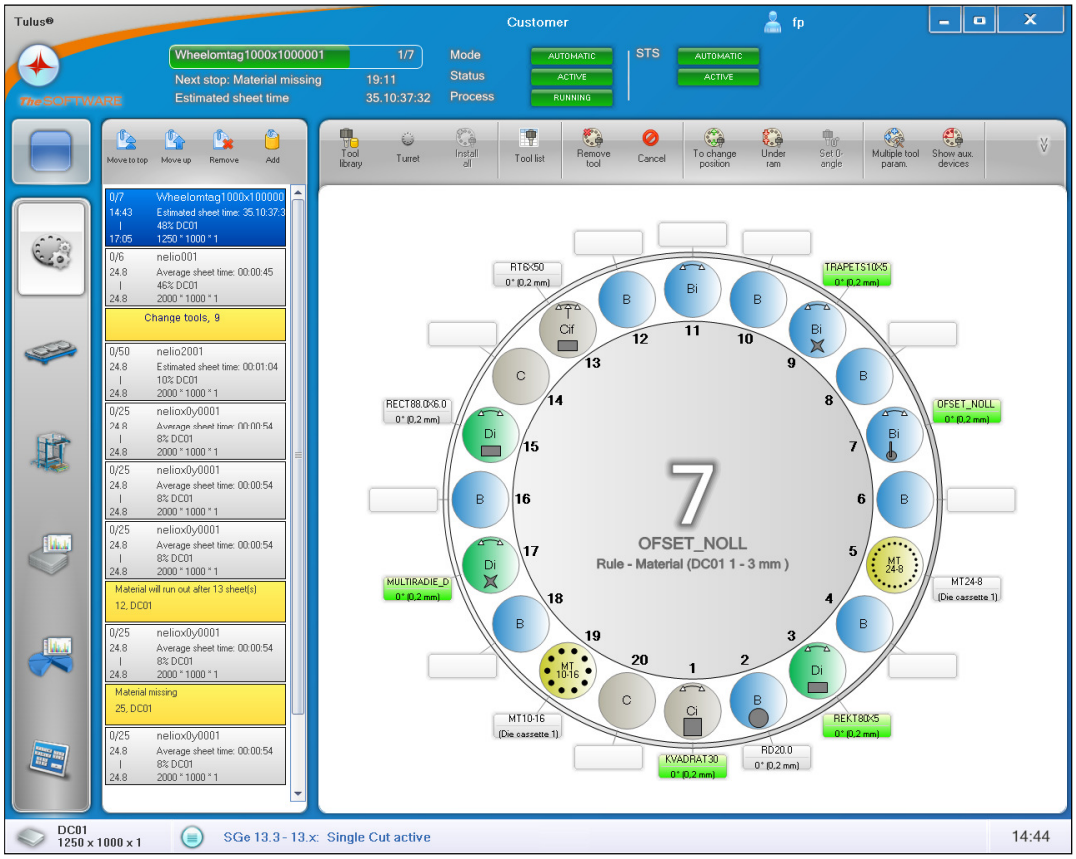
Dokumentteja, kuten asennus- ja käyttöohjeita, aletaan laatia samaan aikaan integraatiotestauksen kanssa. Tässä vaiheessa ohjelmisto on jo niin valmis, että ohjeita voidaan tehdä. Ohjeissa täytyy olla selkeät kuvat ohjelmiston näytöistä ja näytöissä oikeanlaista dataa. Simulaatioympäristössä on helppo kertoa ja näyttää dokumentoijalle, miten uusi ominaisuus toimii ja kuvakaappausten ottaminen on vaivatonta.

Kun integraatiotestaus on hyväksytty, ohjelmisto voidaan siirtää varsinaiseen ympäristöönsä. Tämä tapahtuu joko käyntiinajossa, jossa testaus voidaan suorittaa tuotannossa olevassa järjestelmässä tai asiakkaalla, jossa ominaisuutta testataan oikeassa tuotannossa asiakkaan järjestelmällä. Varsinaiseen ympäristöön siirto riippuu siitä, millainen ominaisuus on kysymyksessä. Lopullisessa hyväksymisessä ominaisuus käydään läpi. Jos kaikki toimii sovitusti, ohjelmisto hyväksytään tuotantoon.

### 3.6 Tulus®-ohjelmistoperhe

Tulus® on kohdeyrityksen luoma ohjelmistokonsepti, jolla hallitaan tuotantoprosessia. Tulus® on paitsi moderni koneen käyttöliittymä, mutta myös yhteensopiva edellisen ohjelmistosukupolven kanssa.

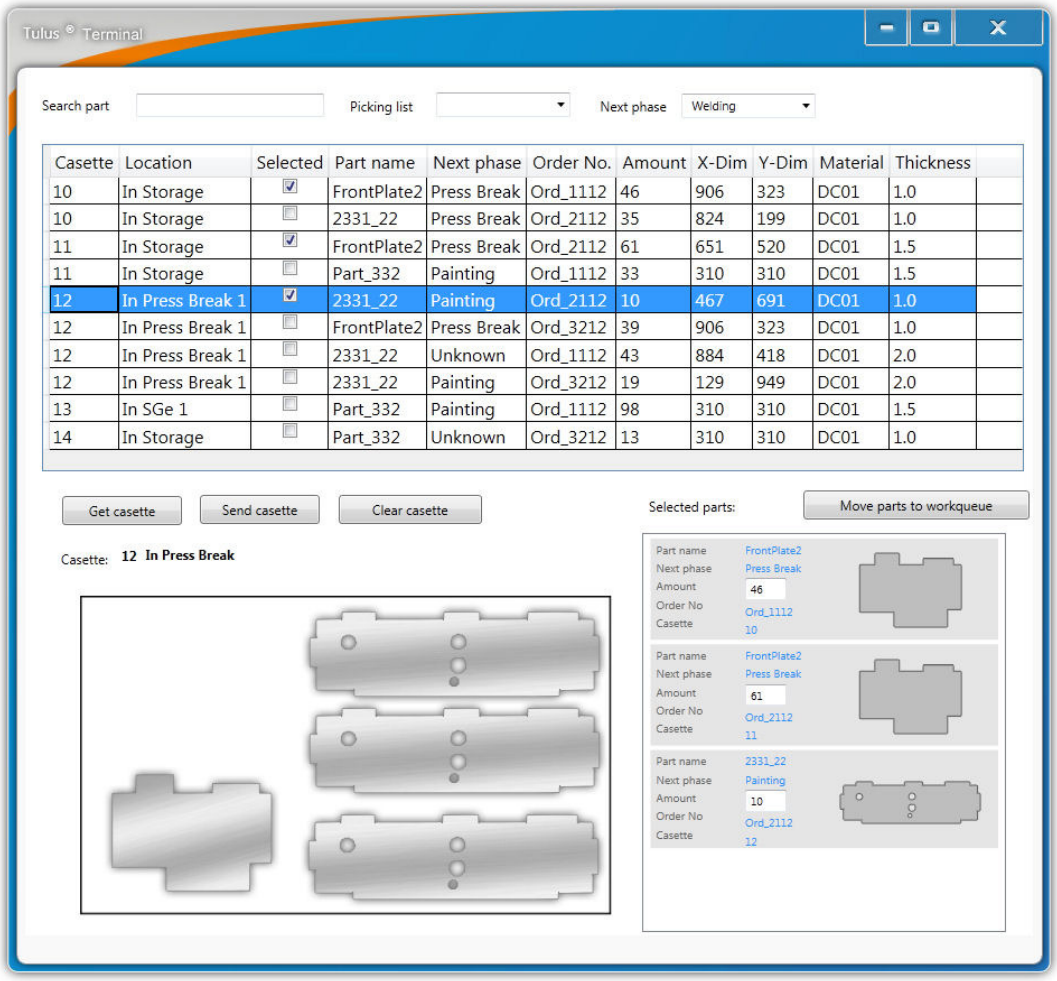
**Tulus® Cell** (kuvio 11) on koneen käyttöliittymä, joka hallitsee koneen toimintaa, laser- ja lävistystyökaluja, työstöjärjestystä ja valmiiden kappaleiden lajittelua. Tuluksen työkalunhallinta sisältää revolveri- ja työkalunhallinnan sekä työkalukirjaston, jossa ovat kaikki Tuluksen tietokannassa olevat työkalut sekä niiden parametrit. Koneen työstöjärjestystä hallitaan tehtävälistan avulla. Tehtävälista näyttää kaikki koneen automaattiset tehtävät, kuten tuotantotilauksen ajon sekä manuaaliset tehtävät, kuten työkalunvaihdon revolveriin ja tarvittavan materiaalin lisäämisen. Tehtävälistaa voi muokata myös tuotantoajon aikana. Tulus® Cell sisältää myös lajittelunhallinnan, joka laskee automaattisesti lavoille sijoitettavien kappaleiden paikat sekä ajat, jolloin lavausalueet on tyhjennettävä käsin tai varastoliitynnässä automaattisesti. Tulus® Cell näyttää koneiden ja laitteiden tilatiedot koneen käyttämistä, testausta ja diagnostiikkaa varten.




Kuvio 11. Tulus® Cell

**Tulus® Power Processing** on valmistuksenohjausjärjestelmä, joka voidaan integroida tuotannonohjausjärjestelmän kanssa. Power Processing valvoo ja optimoi tuotantoprosessia, kuten tuotannon ajoa, kappaletilauksia ja konekuormitusta sekä varastosaldoja. Kappaleet voidaan asetella levylle ilman CAMia ja kappale voidaan reitittää tiettyihin työvaiheisiin halutussa järjestyksessä ja seurata koko ajan, missä vaiheessa valmistettava kappale on.

**Tulus® Terminal** kuvio 12 on yksittäistä työvaihetta, esimerkiksi maalausta tai hitsausta varten suunniteltu käyttöliittymä. Tulus® Terminal näyttää työvaiheen tietoja ja työohjeita, esimerkiksi maalin tiedot, sekä työvaiheen oman tehtävälistan.



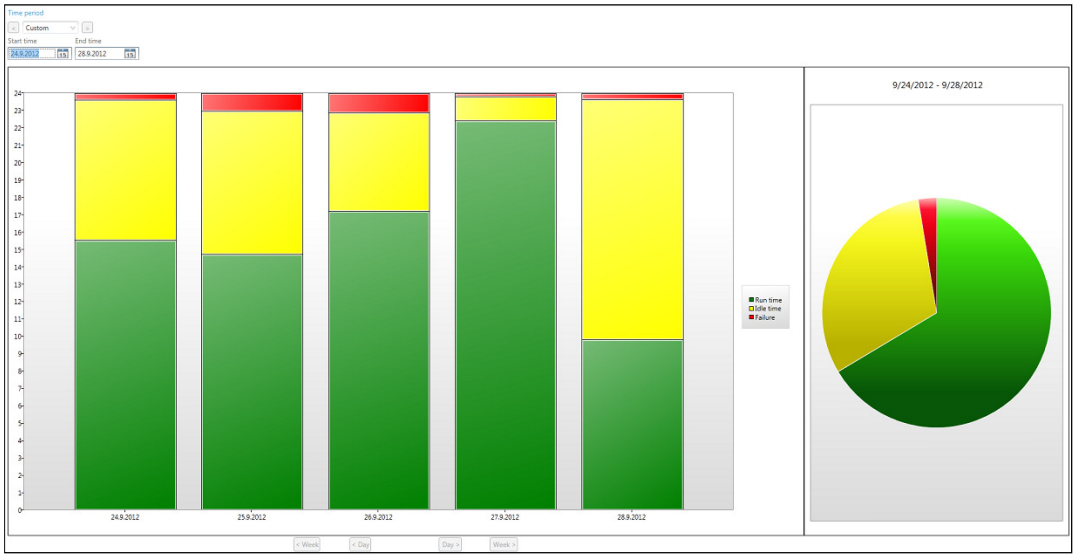

Kuvio 12. Tulus® Terminal

**Tulus® Storage** näyttää kappaleen, kasetin ja materiaalisaldon sekä siinä voi tarkkailla varaston tilaa.

**Tulus® Bend** näyttää taivutusautomaatin tiedot. Sen avulla voi seurata taivutusprosessia, hallita taivutusautomaatin tehtävälistaa sekä työkuormaa.

**Tulus® Performance Reporting** (kuvio 13) antaa tietoa koneen suorituskyvystä ja käyttöasteesta. Sen raporttien avulla voi analysoida, miten tuotannonsuunnittelua voisi parantaa ja esimerkiksi milloin työkaluja kannattaisi huoltaa. Se raportoi kaiken koneajan, mukaan luettuna työstö-, odotus- ja häiriötilanneajat. Hälytysraportin avulla ongelmakohdat voidaan nopeasti tunnistaa ja jäljittää päivämäärän ja ajankohdan avulla ja huolto-osastot voivat ennalta reagoida toistuviin hälytyksiin. Tulus® Performance Reporting voidaan asentaa

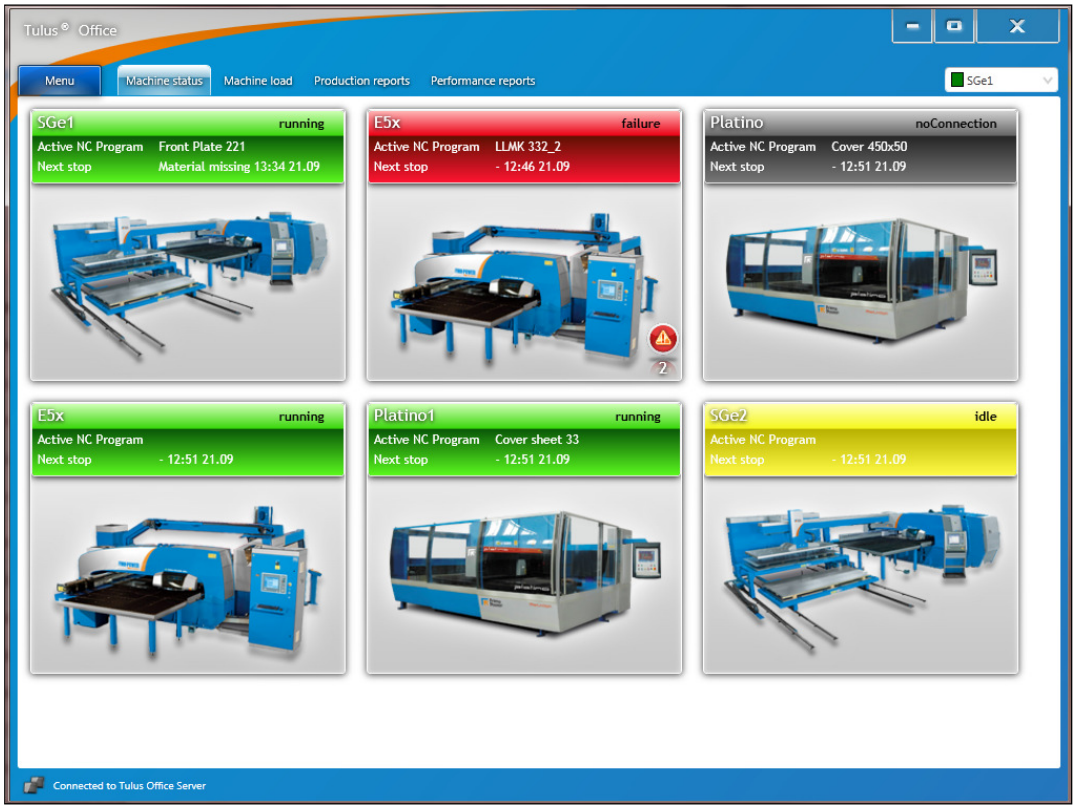
koneen tietokoneelle ja Tulus® Office -tietokoneelle. Se voidaan myös asentaa erillisenä sovelluksena mille tahansa tietokoneelle.




Kuvio 13. Tulus® Performance Reporting

**Tulus® Production Reporting** luo raportteja valmistuneesta tuotannosta. Tuotantotiedot sisältävät mm. käytetyt ohjelmat, komponentit ja materiaalit. Raportit voidaan tulostaa, jakaa tai raportoida takaisin toiminnanohjausjärjestelmään ja ne voidaan myös tallentaa tietokoneelle HTML-, CSV- tai XML-muodossa.

**Tulus® Office** (kuvio 14) on työn- ja tuotannonsuunnittelun sekä konekapasiteetin hallinnan työkalu. Sillä voi seurata useiden koneiden tilaa ja tehtävälistoja ja nähdä tehtaan kaikkien koneiden tapahtumat yhdessä näytössä. Tulus® Office voidaan yhdistää kaikkiin koneisiin, jotka tukevat Tulus® Office -rajapintaa. Tulus® Officessa on koneen aktiivinen tehtävälista koko ajan saatavilla, mikä helpottaa koneiden tehtävien aikataulutusta. Tehtäviä voi lisätä koneen tehtävälistalle ajon aikana tai voi suunnitella esimerkiksi kokonaisen päivän, viikon tai kuukauden tehtävät etukäteen. Tulus® Office sisältää tuotantoraportointi- ja suoritusraportointioptiot. Näillä työkaluilla voidaan rakentaa erilaisia raportteja tuotannonsuunnittelun avuksi ja analysoida, missä olisi parantamisen varaa.




Kuvio 14. Tulus® Office

**Tulus® Mobile Information System** (kuvio 15) lähettää tietoja koneen toiminnasta matkapuhelimeen. Tällaisia tietoja ovat mm. tieto aktiivisesta hälytyksestä ja tulevat manuaaliset tehtävät sekä tilatiedustelut. Käyttäjät voidaan nimetä eri työvuoroihin, jolloin aina työvuorossa oleva henkilö saa tiedot tapahtumista. Konfigurointiasetusten avulla Tulus® Mobile Information System tunnistaa työvuorot ja niiden aikana tavoitettavissa olevat matkapuhelimet. Käyttäjä päättää, mistä hälytyksistä ja tapahtumista tekstiviestejä lähetetään; näin turhia viestejä ei lähetetä. Mobiilikäyttäjä voi myös lähettää koneelle kyselyn ja saada tietoa koneen sen hetkisestä tilasta.


Kuvio 15. Tulus® Mobile Information System

**Tulus® Web Information System** (kuvio 16) antaa tietoja koneesta ja siitä, koska käyttäjän toimenpiteet ovat tarpeen. Sitä käytetään internetselaimella esimerkiksi tietokoneella, älypuhelimella tai taulutietokoneella. Web Information Systemin käyttö vaatii yhteyden tehtaaseen esimerkiksi VPN:n kautta. Web Information Systemin avulla koneen tietoja voi katsella, mutta ei muokata.


Kuvio 16. Tulus® Web Information System

# 4 VAATIMUSMÄÄRITTELY

## 4.1 Vaatimusmäärittely kirjallisuudessa

Aika ajoin huomataan, että ohjelmisto- ja IT-projekteissa tapahtuu hyvin paljon virheitä. Projektien aikataulut venyvät, budjetit ylittyvät ja lopulta toimitetaan vähemmän, mitä alun perin on haluttu. (Meredith & Mantel 2012, 243.)

Vaatimusmäärittely on ohjelmistosuunnittelun haara, joka kuvaa reaalimaailman tavoitteita toiminnoissa ja rajoituksissa koskien ohjelmistojärjestelmiä. Aiheena vaatimustenhallinta on laaja, monitieteinen ja avoin. (Zave 1997.) Vaatimukset tarjoavat navigaatiokartan, jonka avulla projekti ohjataan haluttuun päämäärään. Hyväksytty vaatimusmäärittely tarjoaa pohjan suunnittelulle ja sen toteutukselle. Vaatimukset auttavat hallitsemaan riskejä mahdollisimman aikaisessa vaiheessa suunnittelua. (Hull, Jackson & Dick 2011, 2.)

Ensimmäinen askel ohjelmistojen kehittämisessä on vaatimusmäärittely (Ul-Arif, Khan & Gahyyur 2010, 41). Vaatimusmäärittely on prosessi, jossa tunnistetaan sidosryhmät ja heidän tarpeensa (Nuseibeh & Easterbrook 2000). Vaatimukset kerätään asiakkaalta ja dokumentoidaan. Tämä toiminta on erittäin tärkeää hankkeen menestyksen kannalta, koska kaikki muu toiminta, kuten suunnittelu, toteutus, testaus, käyttö ja kunnossapito, riippuvat vaatimusmäärittelystä. (Ul-Arif, Khan & Gahyyur 2010, 41). Käytännössä on huomattu, että vaatimusmäärittely on hyvä pohja myös testaussuunnitelman tekemiselle. Sen avulla voidaan testausvaiheessa tarkistaa, toimiiko ohjelmisto juuri niin kuin on tarkoitettu ja kuten asiakas on toivonut. Myös kaikkien ominaisuuksien, jotka on listattu vaatimusmäärittelyssä, täytyy löytyä ohjelmistosta. Kuten Wiegers (2003, 383) toteaa, käyttäjän asettamat vaatimukset sekä toiminnalliset vaatimukset ovat tärkeitä tietolähteitä järjestelmän testauksessa. Jos ei ole tarkasti määritelty, miten ohjelmiston tulee toimia erilaisissa olosuhteissa, testaajien on vaikea löytää siitä vikoja ja on hankala todentaa, että suunnitellut toiminnallisuudet on toteutettu kuten pitääkin.

Vaatimusmäärittelyssä määritellään se ongelma, jonka ohjelmiston tulee ratkaista. Eli toisin sanoen määritellään tarkalleen, mitä tämän ohjelmiston tulee tehdä. (Cheng & Atlee 2007). Jos vaatimuksia ei tarkasti määritellä, ohjelmiston kehittäjät eivät tiedä mitä tehdä ja käyttäjät eivät tiedä, mitä ohjelmistolta odottaa. Ilman tarkkoja vaatimuksia ei voida arvioida, tyydyttääkö ohjelmisto käyttäjän tarpeet. Vaatimukset ovat keskeisessä roolissa sekä toimittajan että asiakkaan näkökulmasta. (Saiedian & Dale 2000, 419.) Vaatimusmäärittelyn tärkeyden ymmärtävät kaikki ohjelmistokehitysorganisaatiot, eikä sen hallintaan ja määrittelyyn liittyvien tehtävien tärkeyttä vähätellä (Juristo, Moreno & Silva 2002, 70).

Yksi esimerkki vaatimuskäsitteelle löytyy IEEE:n sanastosta (IEEE Standard Glossary of Software Engineering Terminology 1990, 62):

1. Edellytykset tai kyvyt, joiden avulla käyttäjä ratkaisee ongelman tai saavuttaa tavoitteen.
2. Edellytykset tai kyvyt, joihin järjestelmän tai sen osan tulee kyetä vastaamaan täyttääkseen sopimuksen, standardin, määrittelyn tai muun virallisesti säädetyn dokumentin.
3. Kohtien 1 ja 2 edellytykset ja kyvyt täytyy olla dokumentoituna.

Nopeasti muuttuvat teknologiat ja kasvava kilpailu asettavat paineita prosessien kehittämiselle. Tehokas vaatimusmäärittely helpottaa organisaation kykyä selvitä koko ajan lisääntyvästä monimutkaisuudesta. Vaatimukset ovat perusta jokaisessa projektissa, jotka määrittelevät sidosryhmien eli käyttäjien, asiakkaiden ja suunnittelijoiden vaatimukset tulevalle järjestelmälle ja mitä järjestelmän tulee tehdä tyydyttääkseen nämä tarpeet. (Hull, Jackson & Dick 2011, 1-2.)

Wiegers (2003, 12) toteaa, että vaatimusmäärittely ei sisällä yksityiskohtaista tietoa esimerkiksi toteutuksesta, projektin suunnittelusta tai testauksesta. Tällaiset kannattaakin eriyttää vaatimusmäärittelystä, jotta voidaan keskittyä olennaiseen, eli siihen mitä on tarkoitus rakentaa ja luoda. Projektilla on omat vaatimuksensa, kuten vaikkapa aikataulu- ja budjettivaatimukset, eikä näitä tule sekoittaa tuotteen vaatimuksiin.

## 4.2 Vaatimusmäärittelyn haasteet

Yleisimpiä haasteita vaatimuksia määriteltäessa ovat rajalliset aikataulut ja budjetit, sidosryhmien erilaiset toiveet ja tarpeet sekä asenne.

### 4.2.1 Aikataulut ja budjetit

Korkealuokkaisten ja suurten ohjelmistojen suunnittelun haaste on aikataulujen ja budjettien rajoitukset. Muutokset, jotka sisältävät uusia ja muuttuvia ominaisuuksia, aiheuttavat vielä tätäkin suuremman haasteen. (Collofello & Gosalia 1993, 1095.)

### 4.2.2 Dokumentit

Useat tutkimukset osoittavat, että määrittelyprosessissa tehdyt virheet tuottavat huonolaatuisia dokumentteja tai aiheuttavat ongelmia esimerkiksi sovellusten integroinnissa. Vaatimusmäärittelyyn liittyviä tutkimuksia on tehty jo vuosia ja monia siihen liittyviä ongelmia on tunnistettu ja ongelmiin on löydetty myös ratkaisuja. Tästä huolimatta, vaatimusmäärittelyongelmat jatkuvat edelleen. (Juristo, Moreno & Silva 2002, 70.)

Juristo, Moreno ja Silva (2002, 71) ovat huomanneet, että yleensä asiakaskohtaisissa projekteissa dokumentaatio on erittäin virallista ja yksityiskohtaista, mutta markkinalähtöisissä projekteissa ne ovat hyvin epämuodollisia. Yksi syy puutteelliseen dokumentaatioon on se, että henkilöt jotka niitä laativat, eivät ole perillä ohjelmistotekniikasta. Tämä voi johtua organisaatiorakenteesta, jossa määrittelyjä tekevillä tekijöillä ei ole kokemusta.

### 4.2.3 Kommunikointi

Epäsuora kommunikointi tarkoittaa sellaista tilannetta, joissa asiakas ja kehittäjä eivät keskustele keskenään. Kaikki kommunikointi tapahtuu markkinoinnin ja

myynnin kautta. Tiedon välittäjällä, joka määrittelee asiakkaan tavoitteet ja tarpeet suunnittelijoille ja kehittäjille, ei ole välttämättä täydellistä ymmärrystä asiakkaan tarpeista. Tämän seurauksena voivat tiedot suodattua ja vääristyä, tahallisesti tai tahattomasti. (Saiedian & Dale 2000, 421.) Yleinen ongelma teknisten asioiden kommunikoinnissa on se, että käytetään sellaista terminologiaa jota toinen osapuoli ei ymmärrä. Ammattilaiset käyttävät mielellään runsaasti ammattikieltä ja lyhenteitä. Tällaisen terminologian liikakäyttö voi hämmentää, ärsyttää tai pelotella asiakasta. Keskustelussa on tärkeää asettua asiakkaan teknisen ymmärryksen tasolle. (Saiedian & Dale 2000, 422.)

### 4.2.4 Tarpeet ja tavoitteet

Sidosryhmien tarpeet saattavat olla moninaiset ja voivat olla jopa ristiriidassa keskenään. Ilman vakaata vaatimuspohjaa projekti epäonnistuu. (Hull, Jackson & Dick 2011, 2.)

Myös Nuseibeh ja Easterbrook (2000) toteavat, että prosessissa voi olla useita vaikeuksia. Sidosryhmiä saattaa olla useita ja ne ovat hajautettuja. Tavoitteet voivat vaihdella, näkökulmat voivat riippua ympäristöstä, jossa sidosryhmät työskentelevät. Sidosryhmien tavoitteet eivät ole samat tai henkilöillä voi olla vaikeuksia ymmärtää toisiaan.

### 4.2.5 Asenne

Yhtenä ongelmana on asenne. Käyttäjää pidetään yksinkertaisena, mikä on loukkaavaa käyttäjää kohtaan. Saatetaan ajatella, että asia on liian monimutkaista selittää. Asiakkaan kouluttaminen tuo kuitenkin paljon etuja. Se pakottaa ymmärtämään ongelman ja sen miten sovellus ongelman ratkaisee, mutta nimenomaan selvittää tämän myös asiakkaalle. (Saiedian & Dale 2000, 422.) Asiakkaat eivät yleensä osaa kertoa tarkasti, minkälaisia vaatimuksia heillä on laadun suhteen. Tuotteen halutaan olevan esimerkiksi käyttäjäystävällinen ja nopea, mutta asiakas ei osaa esittää vaatimuksiaan tarkasti. Tärkeää onkin selvittää tarkemmin nämä laatuvaatimukset, jotta ymmärretään, mitä asiakas

todella haluaa. Laatu on monitahoinen käsite ja siksi sen täytyykin tulla määritellyksi sekä asiakkaan että ohjelmistoa rakentavan, testaavan ja ylläpitävien henkilöiden taholta. Haastattelemalla asiakasta saadaan selville ne hiljaiset toiveet ja odotukset, ja näin ollen on paremmat mahdollisuudet saada aikaan onnistunut tuote, joka täyttää kaikki odotukset. (Wiegers 2003, 216.)

### 4.3 Vaatimusmäärittelyprosessin vaiheet

Cheng ja Atlee (2007) jaottelevat vaatimusmäärittelyprosessin viiteen eri vaiheeseen.

- Vaatimusten tunnistaminen (*Elicitation*)
- Mallintaminen (*Modeling*)
- Vaatimusanalyysi (*Requirements analysis*)
- Vahvistaminen ja varmentaminen (*Validation and verification*)
- Vaatimustenhallinta (*Requirements management*)

#### 4.3.1 Vaatimusten tunnistaminen ja mallintaminen

Vaatimusten tunnistamien tarkoittaa toimenpiteitä, joiden tarkoituksena on ymmärtää hankittavan ohjelmiston päämäärä ja tavoitteet (Cheng & Atlee 2007). Ensimmäinen vaihe ohjelmiston kehittämisessä on määritellä tarkasti, mitä ohjelmiston tulee tehdä. Tämän vaiheen aikana käyttäjät alkavat ymmärtää paremmin omia tarpeitaan ja se auttaa heitä arvioimaan tehokkaammin eri vaihtoehtoja ja ymmärtämään päätöstensä seurauksia. Ohjelmiston kehittäjille muodostuu käsitys siitä, mikä on ydinongelma tai -asia, joka ohjelmiston avulla pyritään ratkaisemaan. (Saiedian & Dale 2000, 420.)

On vaikea tietää, koska vaatimuksia on riittävästi tunnistettu ja kyseinen osa projektityöstä on valmis. Koska uusia ideoita tulee mieleen jatkuvasti, jossain vaiheessa niiden etsiminen täytyy vain lopettaa, jotta projekti voisi edetä. Vaatimustyön voidaan päätellä olevan valmis, jos asiakkaan esittämät vaatimukset ovat jo kertaalleen käsiteltyjä tai eivät liity aiheeseen. Yksi tapa on laatia projektille tarkistuslista, jossa on listattuna tärkeimmät kohdat, kuten esimerkiksi virheiden

etsintä, varmuuskopiointi ja palautus, raportointi ja käyttäjäasetukset. Tarkistuslistan avulla voidaan varmistaa, että kaikki tarpeellinen on tullut huomioiduksi. Toisaalta, vaikka vaatimuksia mietitään kuinka tarkasti tahansa, niitä joudutaan aina jonkin verran muokkaamaan projektin edetessä. (Wiegers 2003, 129.)

Mallintamisessa tavoitteet todennetaan mallien avulla. Tämän vaiheen aikana pyritään tarkempiin ja yksiselitteisempiin malleihin kuin mitä tunnistamisvaiheessa on ollut mahdollista tehdä. Mallintaminen auttaa saamaan selville yksityiskohtia, jotka ehkä puuttuivat vielä kokonaan tunnistamisvaiheesta. (Cheng & Atlee 2007.)

UML (Unified Modeling Language) on mallinnuskieli, jonka avulla ohjelmistojen osia visualisoidaan, määritetään, havainnollistetaan, rakennetaan sekä dokumentoidaan. Mallintamisessa käytetään erilaisia kaavioita. UML-kaaviotyyppejä ovat luokka-, olio-, käyttötapaus-, sekvenssi-, yhteistyö-, tila-, toiminto-, komponentti- ja sijoituskaavio. (Rumbaugh, Jacobson & Booch 1999, 3.)

### 4.3.1 Vaatimusten analysointi

Vaatimusten analysoinnissa varmistetaan, että kaikki sidosryhmät varmasti ymmärtävät määritetyt vaatimukset. Vaatimuksia parannetaan poistamalla mahdolliset virheet ja puutteet. Analysoinnissa suuret vaatimukset hajotetaan pienemmiksi yksityiskohdiksi, rakennetaan prototyyppi, arvioidaan toteutettavuus ja päätetään prioriteeteista. Tarkoitus on saada riittävän laadukkaat ja tarkat vaatimukset, jotta niiden perusteella voidaan määritellä projektille arviot ja voidaan jatkaa suunnittelu-, rakennus- ja testaustyötä. (Wiegers 2003, 50.)

Hyväksi vaatimusten analysoijaksi kasvetaan, ei opiskella. Analysointityöhön ei ole tarkkaa työnkuvausta tai opetussuunnitelmaa, vaan se vaatii ennen kaikkea ihmisten parissa olemista. (Wiegers 2003, 71.)

Joskus projektiin osallistuvien henkilöiden välit saattavat olla kireät, eikä toisten motiiveihin luoteta tai toisten tarpeita arvosteta. Vaatimusten analysoijan vastuulla on saada kaikki projektin osapuolet toimimaan yhteistyössä. Hyvä analysoija arvostaa projektin tuomia haasteita ja kohtelee kaikkia osapuolia kunnioituksella.

Hän ohjaa projektia kohti onnistunutta vaatimusten hyväksyntää, jossa kaikki osapuolet voivat olla tyytyväisiä; asiakkaat pitävät tuotteesta, kehitysorganisaatio on tyytyväinen yrityksen tulokseen ja kehitystiimin jäsenet ovat ylpeitä työstään. (Wiegers 2003, 71.)

### 4.3.2 Vaatimusten vahvistaminen

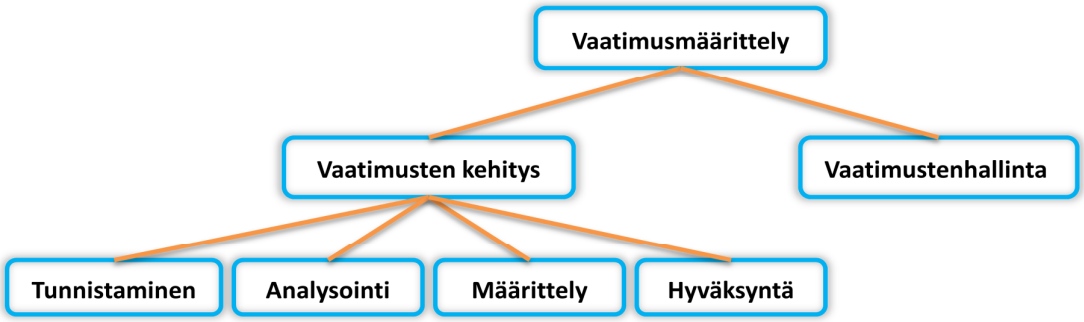
Vaatimusten vahvistaminen varmistaa, että vaatimukset ovat oikein. Vaatimusten tulee täyttää halutut laatuominaisuudet ja tyydyttää asiakasta. Vaatimus, joka vaikutti hyvältä määrittelyssä, saattaakin olla käytännössä työläs tai vaikea kehittäjille. Testitapausten kirjaaminen vaatimusten pohjalta paljastaa epämääräisyyksiä ja epäselvyyksiä. Tällaiset ongelmat tulee selvittää, jotta vaatimukset voivat olla luotettava perusta suunnittelulle ja lopulliselle järjestelmän arvioinnille. (Wiegers 2003, 53.)

Vaatimusten vahvistamiseen ja varmentamiseen on olemassa melko vähän menetelmiä. Joitain käytettyjä menetelmiä ovat esimerkiksi vertaisarvioinnit, katselmoinnit ja skenaariot, lisäksi myös päätösten ja niiden perustelujen kirjaaminen on tärkeää. (Hofmannin & Lehnerin 2001, 59.)

### 4.3.3 Vaatimustenhallinta

Hoffmann, Kuhn, Weber ja Bittner (2004) toteavat vaatimustenhallinnan olevan vaatimusten tunnistamiseen, analysointiin, koordinointiin, versiointiin ja jäljittämiseen liittyvän tiedon organisoimista ja hallintaa koko ohjelmiston elinkaaren ajan.

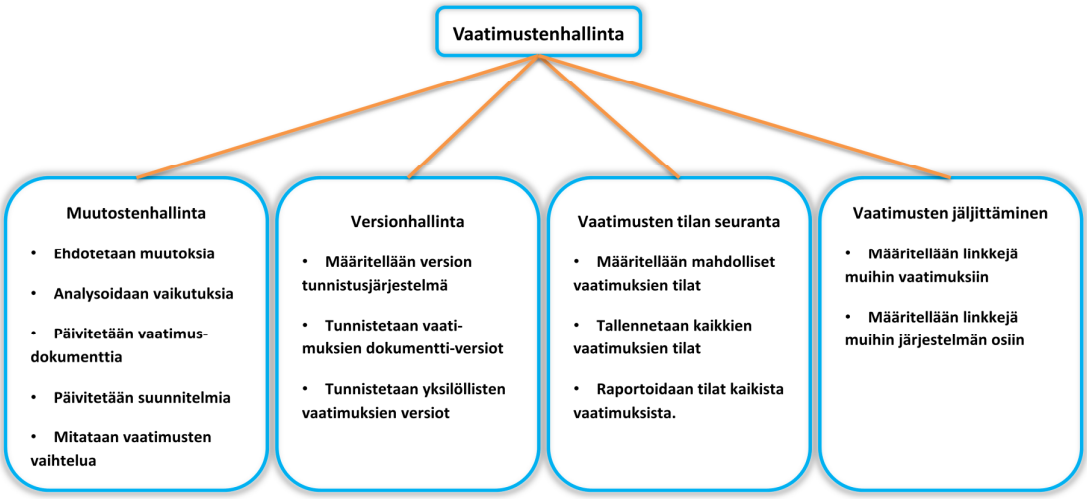
Wiegers (2003) kuvaa vaatimusmäärittelyn osa-alueet. Kuvio 17 havainnollistaa miten vaatimustenhallinta sijoittuu vaatimusmäärittelyprosessissa.




Kuvio 17. Vaatimustenmäärittelyn osiot (Wiegers 2003, 13)

Vaatimustenhallinnan ydintoimintoja ovat muutosten- ja versionhallinta sekä vaatimusten tilojen seuranta ja jäljittäminen (Wiegers 2003, 314.).

Wiegerssin (2003, 314-315) mielestä organisaation täytyy määrittää aktiviteetit projektiryhmille, mitä heiltä odotetaan tehtäväksi, hallitakseen vaatimukset. Kuviossa 18 Wiegers kuvaa yleisimmät vaatimustenhallinnan aktiviteetit.




Kuvio 18. Yleisimmät vaatimustenhallinnan aktiviteetit (Wiegers 2003, 314)

### 4.4 Koostaminen ja dokumentointi

Mallinnus, analysointi sekä eri osapuolten välillä käytyjen neuvottelujen ja vaatimusten dokumentointi on tärkeää. Sen avulla voidaan todentaa, että

neuvotellut ja dokumentoidut vaatimukset vastaavat toisiaan. (Cheng & Atlee 2007.)

Wiegers (2003, 395) määrittelee prosessissa tarvittavat dokumentit seuraavasti:

- Tarkistuslista listaa tehtävät, tuotokset tai muut huomattavat kohteet. Tällä varmistetaan, ettei mitään tärkeää tehtävää unohdeta.
- Suunnitelma, missä hahmotellaan miten tavoite voidaan saavuttaa ja mitä sen saavuttamiseksi tarvitaan.
- Pääperiaate, joka ohjaa johdon toimintaa, käytöstä ja toimintatapoja.
- Menettelytapa, jossa tehtävien järjestys on kuvattuna vaihe vaiheelta, step-by-step. Tässä dokumentissa on kuvattuna tehtävä sekä roolit, jotka suorittavat sen tässä projektissa. Ohjetekstiä tehtävän suorittamista varten voi olla lisädokumenttina.
- Prosessikuvaus sisältää kuvauksen projektin tavoitteesta sekä siitä, mitkä ovat sen tärkeimmät välitavoitteet. Lisäksi siinä kerrotaan mm. projektin osapuolet, yhteydenpidon tiheys sekä prosessissa käytettävät työkalut.
- Kaava, jolla prosessi toteutetaan. Valmis kaava auttaa huomioimaan myös ne kohdat ja kysymykset, jotka muuten saattaisivat jäädä huomioimatta.

Cheng ja Atleen (2007) mielestä vaatimusten koostaminen luo ymmärrystä päämäärästä ja tavoitteista sekä siitä, miksi ohjelmisto tehdään. Koostaminen määrittelee ne vaatimukset, jotka kehitettävän järjestelmän on täytettävä, jotta päämäärään päästään.

Vaatimusten koostamiseksi on kehitetty erilaisia tekniikoita, jotta päästään mahdollisimman tarkkaan ja yksityiskohtaiseen tulokseen:

- Sidosryhmien tunnistaminen varmistaa, että jokainen taho, jota ohjelmisto koskettaa, tulee kuulluksi (Sharp, Finkelstein & Galal 1999).
- Analogiset tekniikat, kuten metaforat ja persoonat auttavat sidosryhmiä ajattelemaan vaatimuksia syvällisemmin ja tarkemmin, jotta yksityiskohdat tulisivat huomioiduiksi (Cooper 2004).
- Sidosryhmien vaatimuksia analysoidaan asiayhteyden, ohjelmiston ympäristön sekä ohjelmistoa käyttävien henkilöiden pohjalta. Tällä varmistetaan, että tuleva ohjelmisto sopii ympäristöönsä ja niihin

tarpeisiin, johon sitä ollaan luomassa. (Sutcliffe, Fickas, & Sohlberg 2006.)

- Erilaiset aivoriihet ja ideatyöpajat luovat ja löytävät sellaisia vaatimuksia, jotka eivät välttämättä ole olennaisia mutta tekevät valmiista tuotteesta, tässä tapauksessa siis ohjelmistosta, houkuttelevamman ja myyvemmän. (Maiden & Robertson 2005.)
- Havainnollistamistekniikat, kuten simulointi, mallinnus ja kuvalliset kertomukset, auttavat positiivisten ja negatiivisten palautteen saamisessa jo varhaisessa vaiheessa. (France & Rumpe 2007, Thompson, Heimdahl & Miller 1999.)

Madridin ammattikorkeakoulussa tehtyjen tutkimusten mukaan vaatimustekniikoita ei hyödynnetä tarpeeksi määrittelyn koostamisessa eikä neuvotteluissa. Vaikka nämä tekniikat ovat olleet saatavilla jo vuosia, organisaatiot eivät vieläkään niitä tunne. Tästä johtuen tieto ei välity tehokkaasti eri osapuolien välillä. Puutteelliset ohjelmistovaatimusdokumentit aiheuttavat paljon lisätyötä. Dokumenteissa käytetty kieli ei saa olla esteenä korkealaatuisen dokumentin synnylle tai analysointityökalujen käytölle dokumentin arvioinnissa. Vaatimusmäärittelydokumentin korkea taso saadaan käyttämällä tyylioppaita, huolimatta siitä millä kielellä dokumentti on kirjoitettu. (Juristo, Moreno & Silva 2002, 74.)

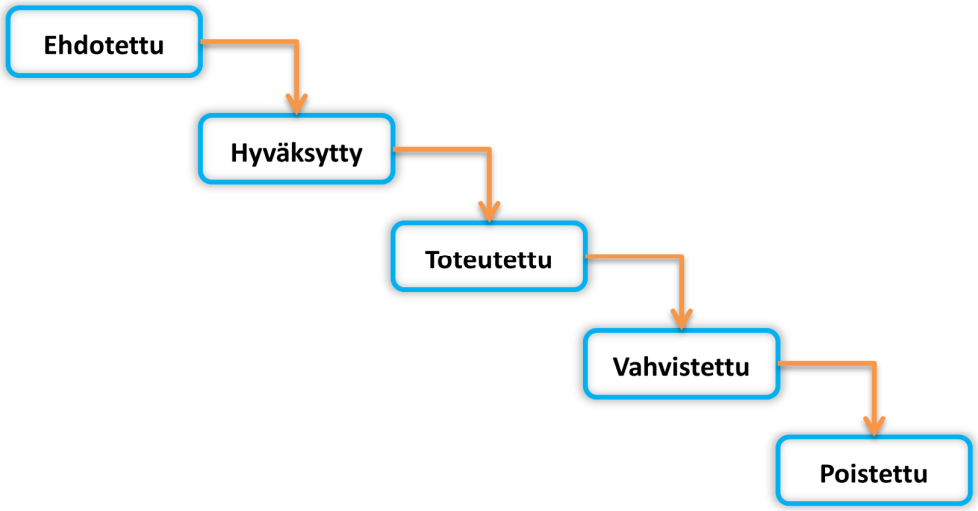
Kun vaatimukset muuttuvat, niiden hallinta ei ole vain dokumenttien päivitystä, vaan muutokset vaikuttavat aina paljon laajemmin. Tämä vaatii vaatimusten tunnistamista, riskien uudelleenarvioimista ja järjestelmän arvioimista operatiivisessa ympäristössään. Jokainen muutos tulee arvioida jo olemassa olevien vaatimusten ja arkkitehtuurin ehdoilla, jotta sen kustannus saadaan selville. Tyypilliset muutokset ovat uusien vaatimusten lisäämistä, jotka ovat jääneet jostain syystä pois alkuperäisestä. Vaatimuksia lisätään myös jos sidosryhmien tarpeet muuttuvat. Vaatimuksia saatetaan poistaa kehitystyön aikana, näin pyritään kustannus- ja aikataulusäästöihin. (Nuseibeh & Easterbrook 2000.). Kaikkia vaatimuksia ei ole mahdollista määritellä etukäteen, sen vuoksi ohjelmiston elinkaaren aikana kehittyy aina uusia vaatimuksia. Muutoksia ei voi välttää, joten niitä pitää pystyä hallitsemaan. (Wiegers 2003, 328.)

## 4.5 Vaatimusten tilojen seuranta

Vaatimusten tilojen seuranta on yksi vaatimustenhallinnan toiminnoista (Wiegers 2003, 314).

Ohjelmistokehittäjät ovat joskus liian optimistisia raportoidessaan tehtävän valmiusastetta. He antavat luotettavaa tietoa siitä, että työt on aloitettu, mutta eivät tehtävän valmistumisesta. Työ on usein 90 prosenttisesti valmis. Jäljittämällä jokaisen erillisen vaatimuksen tilaa tarjoaa se paljon varmemman mittarin projektin kulusta ja sen tilasta. (Wiegers 2003, 321.)

Vaatimusten tilojen seurantaa pidetään yhtenä vaatimustenhallinnan osa-alueista. Tiloja ovat esimerkiksi ehdotettu, hyväksytty, toteutettu, vahvistettu ja poistettu (kuvio 19). (Wiegers 2003, 322–323.)




Kuvio 19. Vaatimusten tiloja

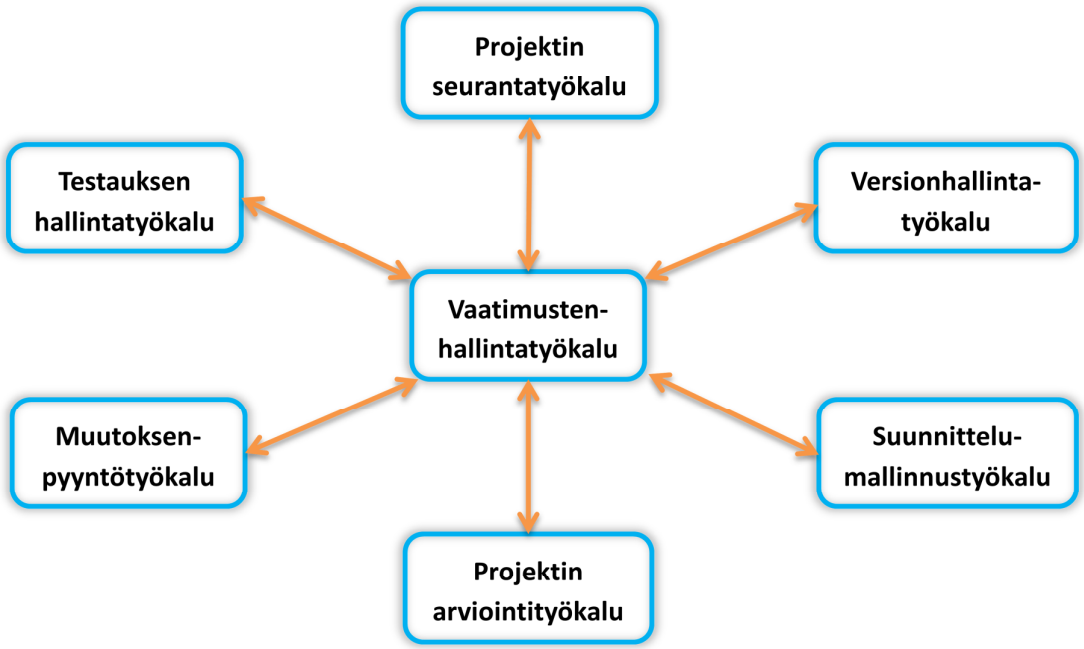
## 4.6 Vaatimustenhallinnan ohjelmistoja

Vaatimustenhallintaan tarkoitettuja ohjelmistoja käytetään yhä yleisemmin ja ne ovat auttaneet parantamaan ohjelmistojen laatua ja niiden käyttö on levinnyt teollisuudessa (Ruhe, Eberlein & Pfahl 2002).

Vaatimustenhallintatyökalun käyttämisellä projektissa on monia hyötyjä. Sen avulla on helppo hallita versioita, muutoksia sekä sitä kenellä on oikeus päästä lukemaan tai muokkaamaan projektin tietoja. Vaatimustenhallintatyökaluun voi tallentaa vaatimusten attribuutteja ja sen avulla voi analysoida mitä vaikutuksia tulee olemaan, jos yksittäiseen vaatimukseen tehdään muutoksia. Sen avulla voi seurata vaatimuksen tilaa ja kommunikoida eri sidosryhmien kanssa. Vaatimusten tallentaminen tietokantaan mahdollistaa myös niiden uudelleenkäyttämisen. (Wiegers 2003, 370-371.)

Projektin vaatimusten tallentaminen tietokantaan sekä niiden ajan tasalla pitäminen, attribuuttien ja jäljitettävyys linkkien määritys, käyttäjäryhmien oikeuksien määritys sekä käyttäjien koulutus vaatii paljon työtä. Tämä täytyy ottaa huomioon ja näiden tekemiseen tulee varata riittävät resurssit. Koko organisaation tulee sitoutua käyttämään valittua vaatimustenhallintatyökalua, ettei se jää vain kalliiksi ja turhaksi hankinnaksi. (Wiegers 2003, 378.)

Wiegers (2003) kuvaa (kuvio 20) sitä, kuinka vaatimustenhallintatyökalu integroituu muiden ohjelmistojen kanssa.




Kuvio 20. Ohjelmistotyökalujen integraatio (Wiegers 2003, 373.)

Vaatimustenhallintaohjelmistot ovat kehittyneet nopeasti. Kysynnän joustavuus, ketterä ohjelmistokehitys, maailmanlaajuinen yhteistyö sekä kehittyneet ohjelmistot ja järjestelmien ekosysteemit muuttuvat aiheuttaen haasteita vaatimustenhallinnalle. Esimerkiksi ketterän ohjelmistokehityksen tiimit ovat vähemmän dokumenttikeskeisiä ja enemmän koodisuuntautuneita. He odottavat, että vaatimukset liittyvät suoraan koodimuutoksiin ja vaatimustenhallintaohjelmisto tulisi olla kevyt. Toisaalta ohjelmistokehitystiimit tarvitsevat helposti ja kattavasti määritellyt vaatimukset ja niiden jäljitettävyyden koko niiden elinkaaren ajan aikana. (Carrillo 2010, 86.)

Vaatimustenhallintaohjelmistojen markkinat muuttuvat nopeasti. Klassiset ohjelmistot, jotka hallitsevat markkinoita, ovat yhä monimutkaisempia ja vaikeita käyttää. Monet kalliit ohjelmistot eivät ole riittävän avoimia muille toimittajille. Tämä kannustaa tulokkaita esittelemään mielenkiintoisia ominaisuuksia, joita voidaan käyttää joustavammin yhteistyössä. Vaatimustenhallintaohjelmiston sopivuutta on syytä arvioida huolellisesti omassa ympäristössä. On varauduttava maksamaan, koska halvimmat välineet eivät sisällä kehittyneempiä ominaisuuksia. (Carrillo 2010, 91.) Suurin ongelma useiden vaatimustenhallintaohjelmistojen käyttöönotossa on ohjelmistojen integraation puute. Tähän ei näytä olevan mitään selvää ratkaisua, vaikka koko ajan kehitetään sellaisia ohjelmistoja, jotka tarjoavat enemmän mahdollisuuksia. (Juristo, Moreno & Silva 2002, 74.)

Projektin dokumentteja on käytännössä vaikea pitää aina ajan tasalla ja synkronoituina. Pienikin muutos tarkoittaa dokumentin päivittämistä ja kirjaaminen on työlästä. Jos esimerkiksi jokin tietty ominaisuus siirtyykin ohjelmistoversiosta toiseen, myöhemmin toteutettavaksi tai kokonaan toiseen tuotteeseen, täytyy ominaisuuden vaatimukset siirtää dokumentista toiseen. Vaatimusten tietojen muokkaaminen dokumentteihin saattaa olla hankalaa, varsinkin jos projektin osapuolet ovat maantieteellisesti kaukana toisistaan. Ongelmana ovat myös jo kirjatut, mutta sittemmin hylätyt vaatimukset, sillä niiden tallentamiseen ei usein löydetä mitään sopivaa paikkaa. (Wiegers 2003, 367-368.)

Dokumentoinnin haasteisiin vastaa parhaiten vaatimustenhallintaohjelmisto, joka tallentaa tiedot sellaiseen tietokantaan, jonne on pääsy kaikilla tarvittavilla osapuolilla. Pienissä projekteissa voidaan käyttää taulukkolaskentaohjelmaa tai

pientä tietokantaa, jonne vaatimukset sekä niiden selitykset ja attribuutit tallennetaan. Isoissa projekteissa on hyödyllistä käyttää jotain markkinoilla olevista vaatimustenhallintaohjelmistoista, joissa vaatimukset voidaan tuoda tietokantaan suoraan lähdedokumenteista. Lisäksi tietoja voidaan hakea ja suodattaa suoraan tietokannasta sekä viedä ja tallentaa tietoa eri formaateissa. Vaatimuksia voidaan yhdistää ja linkittää muihin nimikkeisiin, jotka ovat tallennettuina muissa ohjelmistokehitystyökaluissa. (Wiegers 2003, 368.)

Wiegers (2003, 368) kutsuu ohjelmistoja vaatimustenhallintaohjelmistoiksi, ei vaatimustenkehitysohjelmistoiksi. Ohjelmistot eivät kerää vaatimuksia projektia varten tai määrittele mahdollisia tuotteen käyttäjäryhmiä, vaan ovat vaatimusten hallintaa helpottavia ohjelmistoja. Niiden avulla vaatimusten muutosten hallinta on joustavaa käyttäen vaatimuksia suunnittelun, testauksen ja projektinhallinnan perustana. Nämä ohjelmistot eivät korvaa projektissa vaatimusten tunnistamis- ja määrittelyprosessia, eivätkä prosessin, kurin tai kokemuksen puuttumista.

### 4.7 Vaatimusten priorisointi

Kaupallisten ohjelmistojen kehittämisessä on kasvava tarve menetelmälle, jolla voidaan priorisoida vaatimukset. Kaikkia vaatimuksia ei voida yleensä toteuttaa, johtuen käytettävissä olevasta ajasta ja resurssien rajallisuudesta. Asiakkaat suuremmassa määrin vaativat järjestelmältä rahoilleen vastinetta. Priorisoimalla vaatimukset voidaan jakaa eri versioihin. Tehokkaan ja luotettavan menetelmän löytyminen vaatii paljon harjoittelua. Lupaavia tuloksia tähän tarkoitukseen antaa kustannusarvoon perustuva lähestymistapa. (Karlsson, Wohlin & Regnell 1998.)

Firesmith (2004) pitää usein vaikeana tehtävänä priorisoida vaatimuksia niin, että korkeimman prioriteetin vaatimukset voidaan toteuttaa heti osana projektin aikataulua.

Pelkästään priorisointi ei voi määrittää tehtävää työjärjestystä, vaan saattaa olla myös muita asioita, jotka vaikuttavat tähän. Jokin tehtävä on vain teknisesti pakko tehdä ensin, esimerkiksi siksi että jälkeenpäin sen tekeminen tulee kalliimmaksi. Myös Firesmith (2004) viittaa tähän luetellessaan eri tapoja miten vaatimuksia

voidaan priorisoida. Ensimmäiseksi määritellään välttämättömät ja tärkeät vaatimukset. Vaikka kaikki vaatimukset olisivatkin pakollisia, toiset ovat silti kriittisempiä kuin toiset. Tarpeettomat vaatimukset poistetaan yhteisymmärryksessä ja sen jälkeen vaatimusten toteutus aikataulutetaan.

Firesmith (2004) toteaa, että suurissa ja monitahoisissa ohjelmistoprojekteissa on tavallisesti valtavan paljon yksittäisiä vaatimuksia. Hyvin usein asiakas haluaa jokaisen niistä myös toteutettavan. Tällaiset projektit eivät kuitenkaan voi välttyä ongelmilta. Näkemyserot vaatimusten tärkeydessä saattavat poiketa. Se mikä on tärkeää yhdelle sidosryhmälle voi olla vähemmän tärkeää toiselle. Kaikilla projekteilla on rajallinen määrä resursseja, kuten budjetti, työntekijät ja aikataulu. Kaikkien vaatimusten toteuttaminen on käytännössä mahdotonta, ainakin järjestelmän julkaisun aikana. Suuret ohjelmistojärjestelmät toteutetaankin usein vaiheittain. Pitkät aikataulut ovat ongelmallisia ja kehitykseen kuluu kuukausia, usein jopa vuosia. Ajan kuluessa yrityksen ympäristö ja tarpeet muuttuvat ja uusia vaatimuksia tulee ilmi. Vaatimuksia tulee siis tarkistaa ja päivittää.

Karlsson, Wohlin ja Regnell (1997, 940) toteavat, että priorisointimenetelmät ohjaavat tekijöitä heidän tehtävässään analysoida vaatimuksia ja järjestää ne tärkeysjärjestykseen. Priorisointi-istunnossa voi olla kolme peräkkäistä vaihetta:

1. Valmisteluvaiheessa vaatimukset priorisoidaan valittujen kriteerien mukaisesti. Tiimi ja tiiminvetäjä on valittu istuntoa varten ja kaikki tarvittavat tiedot on toimitettu.
2. Toteuttamisvaiheessa, jossa tekijät tekevät vaatimusten priorisoinnin käyttäen apuna valmisteluvaiheessa saatua tietoa. Arvioinnin kriteerit täytyy hyväksyä, ennen kuin toteuttamisvaihe aloitetaan.
3. Esitysvaiheessa tulokset esitellään asianosaisille. Joihinkin priorisointimenetelmiin liittyy erilaisia laskelmia, jotka tehdään ennen kuin tulokset voidaan esittää.

Firesmith (2004) toteaa, että vaatimusten kunnollinen priorisointi tuo projektille valtavasti hyötyjä:

- Aikataulun muokattavuus. Koska kehityssykli on Iteratiivinen eli suunnittelua ja toteutusta tehdään pienissä osissa ja prosessia toistetaan,

antaa se aikataululle joustoa. Sekä projektipäällikkö että asiakas voivat muokata projektin aikataulua, jotta rajallisten resurssien ja kiinteiden aikataulujen kanssa selvitään.

- Parempi asiakastyytyväisyys. Asiakas on tyytyväinen, kun hänelle tärkeät vaatimukset toteutetaan ensin.
- Riski projektin peruuntumiselle on pienempi, sillä jokaisessa inkrementointivaiheessa voidaan osoittaa todeksi, että projekti etenee ja kehitystä on tapahtunut. Vaikka projekti peruuntuisikin ennen lopullista valmistumistaan, tehty työ ei olisi mennyt hukkaan, sillä tärkeitä toiminnallisuuksia on kuitenkin jo toteutettu ja toimitettu asiakkaalle.
- Kaikkien sidosryhmien vaatimukset tulevat huomioiduiksi.
- Priorisointi antaa karkean arvion eri vaatimusten eduista.
- Priorisointi auttaa resurssien kohdentamisessa, mikä on erityisen tärkeää, sillä resurssit ovat aina rajalliset.

Menestyksekkään ohjelmiston mittari on ensisijaisesti se, missä määrin ohjelmisto täyttää ne odotukset, joita siltä vaaditaan ja kuinka se soveltuu ympäristöönsä (Nuseibeh & Easterbrook 2000; Cheng & Atlee 2007). Ohjelmiston tekovaiheessa tulisi aina muistaa, ketä varten ohjelmistoja tehdään. Asiakkaan tilaaman tuotteen täytyy soveltua juuri siihen tarkoitukseen, mihin se on tilattu.

Firesmith (2004) toteaa, että vaatimuksia voidaan priorisoida erilaisin mittarein, jotka voivat olla jopa toistensa vastakohtia. Esimerkkejä mittareista ja niiden luokittelusta ovat:

- Sidosryhmien omat, henkilökohtaiset mieltymykset.
- Toiset vaatimukset tarjoavat enemmän arvoa liiketoiminnalle kuin toiset. Toiset vaatimukset ovat liiketoiminnan kannalta elintärkeitä, kun toiset puolestaan ovat vähemmän tärkeitä, joskin kuitenkin pakollisia.
- Vahinkojen välttäminen. Mitä vahinkoa koituu, jos vaatimusta ei toteuteta. Tämä kriteeri toteutuu varsinkin turvallisuusvaatimuksissa.
- Vaatimuksia priorisoidaan niiden toteutukseen liittyvien riskien perusteella.

- Eri vaatimusten toteuttamisella on erilaiset kehitys- ja elinkaarikustannukset. Tämä kriteeri on tärkeä ja saattaa olla jopa määräävä vaatimuksia priorisoitaessa.
- Vaatimukset voidaan toteuttaa vaikeusasteen perusteella, esimerkiksi niin, että ensin toteutetaan helpoimmat vaatimukset ja vasta sitten siirrytään vaikeimpiin.
- Kiireellisyys saada tuote markkinoille. Tiukassa markkinatilanteessa, jossa kilpailijat tarjoavat vastaavanlaisia tuotteita, saattaa nopeus olla tärkeä kriteeri.
- Vaatimusten vakaus. Jotta ylimääräiseltä työltä vältyttäisiin, on järkevää toteuttaa ensin sellaiset vaatimukset, jotka eivät tule muuttumaan.
- Riippuvuudet täytäntöönpanossa. Suuria järjestelmiä kehitettäessä järjestelmän tietyt komponentit ovat riippuvaisia toisistaan, esimerkiksi perus- ja infrastruktuurikomponentteihin liittyvät vaatimukset tulee toteuttaa ennen muita vaatimuksia.
- Eri vaatimuksia tulee lähestyä eri priorisointikriteerein. Ei-toiminnalliset vaatimukset voidaan priorisoida suoraan, kun taas toiminnallisia vaatimuksia voidaan priorisoida epäsuorasti käyttötapauksien ja skenaarioiden kautta.
- Vaatimukset voidaan priorisoida tärkeämmiksi, jos ne on määrätty laissa tai säädöksissä tai jos niitä määrittävät esimerkiksi kansainväliset standardit.
- Toiminnalliset vaatimukset saatetaan priorisoida tärkeämmiksi, jos niillä oletetaan olevan suuri käyttöaste.
- Jos vaatimusta voidaan uudelleen käyttää tuotesarjassa, on se hyvä priorisoida korkealle, ettei tuotesarjassa tapahdu viivästyksiä.

# 5 VAATIMUSTENHALLINTAOHJELMISTO

Vaatimustenhallintaohjelmistoja on tarjolla paljon ja niiden ominaisuuksien ero on suuri. Paljon ominaisuuksia sisältävät ohjelmistot ovat kalliita ja edullisemmat ohjelmistot ovat ominaisuuksiltaan vaatimattomampia. Turhista ominaisuuksista ei kannata maksaa, mutta tarvittavista ominaisuuksista kylläkin.

Ohjelmisto on vain apuväline, työntekijöiden tulee hallita työ ja tarvittavat tekniikat. Tietojen ja dokumenttien koottu säilyttäminen on kuitenkin tärkeää.

Tutkimuksen tavoitteena on löytää kohdeyritykselle sellainen vaatimustenhallintaohjelmisto, joka sopii yrityksen tarpeisiin, mutta on myös integroitavissa muihin ohjelmistokehitysprosesseihin. Ohjelmiston tulee integroitua seuraaviin ohjelmistoihin:

- Projektinseurantaohjelmisto
- Versionhallintaohjelmisto
- Muutoksenpyyntöohjelmisto
- Testauksenhallintaohjelmisto
- Palauteohjelmisto.

Kaikki edellä mainitut ohjelmistot käsittelevät vaatimuksia tavalla tai toisella. Ohjelmistosta joko tulee vaatimuksia tai sen avulla toteutetaan vaatimuksesta uusi ominaisuus.

Vaihtoehdot kohdeyrityksen vaatimustenhallintaohjelmaksi ovat Atlassian JIRA, IBM® Rational® RequisitePro® ja Microsoft Visual Studio Team Foundation Server Requirements Engineering.

## 5.1 Atlassian JIRA

JIRA ei ole varsinainen vaatimustenhallintaohjelmisto, vaan projektinhallintaan ja jäljittämiseen tehty web-pohjainen ohjelmisto. JIRA-ohjelmistoon on helppo lisätä ominaisuuksia laajennusosien avulla. Niitä on tarjolla ilmaisia ja maksullisia, riippuen tasosta ja laajuudesta. JIRA on myös helppo räätälöidä omiin tarpeisiin

sopivaksi, joten sen joustavan arkkitehtuurin vuoksi se on taipuisa työkalu moneen eri tehtävään.

### 5.2 IBM® Rational® RequisitePro®

IBM® Rational® RequisitePro® on vaatimusten ja käyttäjätapausten hallinta- ja analysointityökalu, jolla on myös tietokantamaisia ominaisuuksia, kuten muutosten jäljitettävyys ja vaikutusten arviointi. Koska relaatiot ovat ohjelmistossa näkyvillä, sillä on helppo hallita laajoja kokonaisuuksia ja ymmärtää riippuvuussuhteita. Skaalautuvan web-käyttöliittymän avulla projektin sidosryhmät voivat pitää yhteyttä missä päin maailmaa tahansa. Projektiin tehtyjä muutoksia voidaan jäjittää versioiden vertailutyökalulla. IBM® Rational® RequisitePro® -ohjelmisto voidaan integroida monien ohjelmistojen kanssa, esimerkiksi IBM Rational -ohjelmistojen ja Microsoft® Word -ohjelman kanssa.

IBM® Rational® RequisitePro® on kattava ohjelmisto vaatimusten hallintaan. Tuoteperheeseen on myös tarjolla paljon muita ratkaisuja ohjelmistojen kokonaisvaltaiseen kehitykseen ja suunnitteluun. Kohdeyrityksessä on käytössä Rational Functional Tester -ohjelmisto, jota käytetään automaattisen ohjelmistotestauksen työvälineenä. Tästä syystä IBM® Rational® RequisitePro® -ohjelmisto on valittu yhdeksi vaihtoehdoksi työkalun valinnassa.

### 5.3 Microsoft Visual Studio Team Foundation Server Requirements Engineering

Microsoft Requirements Engineering on hyvin monipuolinen ohjelmisto, joka tarjoaa työkalut kattavaan projektinhallintaan ja ohjelmistosuunnitteluun. Ohjelmisto mahdollistaa mm. vaatimusten jäljittämisen, analysoinnin, koostamisen, vaikutusten analysoinnin sekä integroinnin kolmannen osapuolen ohjelmistoihin.

### 5.4 Ohjelmiston valinta

Tarkoituksena oli löytää oikeanlainen työkalu, joka sopisi kohdeyrityksen prosessiin. Työkalun tuli olla kohtuullisen hintainen sekä helppokäyttöinen. Alusta alkaen vahvana ehdokkaana oli Atlassian JIRA, joka on jo käytössä ohjelmistokehityksen projektityökaluna ja myös testauksessa, virheiden kirjauksessa ja jäljityksessä. JIRA on myös konsernitasolla käytössä muissa suunnitteluosastoissa sekä asennus- ja huoltopalautteen hallinnassa. Tutkimuksessa selvitettiin muita vaihtoehtoja, jottei mitään oleellista jäisi huomaamatta. Tärkeä valintakriteeri ohjelmistolle oli, ettei järjestelmien määrä kasva, vaan yritys pyrki laajentamaan olemassa olevia järjestelmiä uusilla ominaisuuksilla.

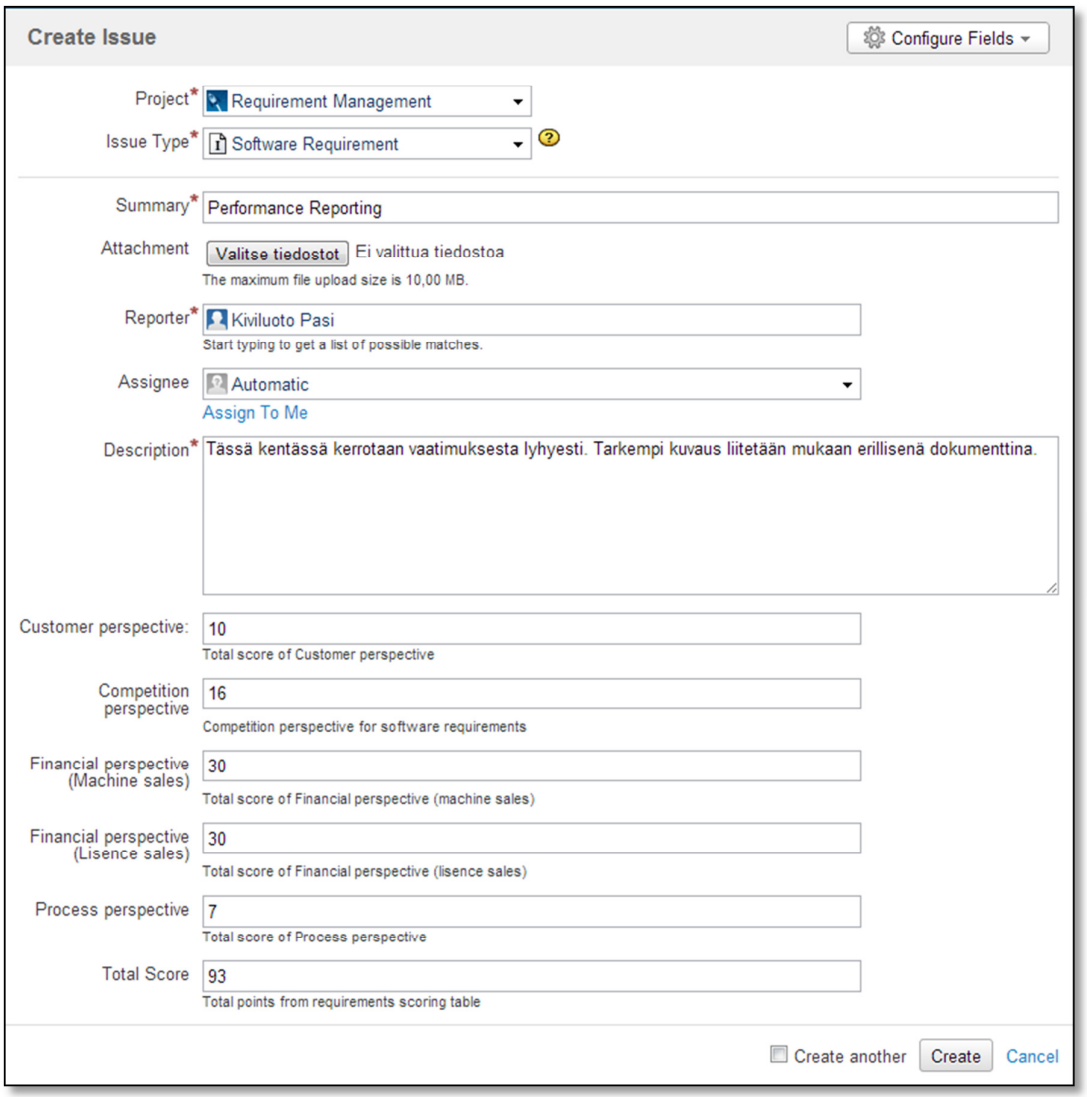
Käytettävän ohjelmiston valinta oli helppo, sillä jo aikaisempien kokemusten perusteella JIRA tarjosi hyvät mahdollisuudet vaatimusten hallitsemiseen. Tärkeitä JIRA:n tukemia ominaisuuksia olivat myös vaatimustyyppinen tehtävä ja pisteytyksen arvokentät. Päätökseen myös vaikutti se, että kaikki työt löytyvät JIRA:sta, jolloin hyväksytyt vaatimukset voidaan helposti siirtää työn alle samasta järjestelmästä. Vaatimuksen antaja voi helposti seurata, miten työ etenee kyseisen vaatimuksen kohdalla. JIRA integroituu kaikkiin vaadittuihin ohjelmistoihin, joita aikaisemmin listattiin.

Microsoft Visual Studio Team Foundation Server Requirements Engineering -ohjelmisto sopisi hyvin kohdeyrityksen ohjelmistokehitykseen, sillä yrityksessä on jo käytössä Microsoft Visual Studio. Ohjelmisto hylättiin kuitenkin heti liian korkean hinnan vuoksi. Koska yrityksessä tarvitaan useita lisenssejä, ei kallis ohjelmisto sopinut budjettiin.

IBM® Rational® RequisitePro® -ohjelmisto todettiin liian monipuoliseksi ja kalliiksi kohdeyritykselle. Ohjelmistossa on paljon sellaisia ominaisuuksia, joita yritys ei tulisi käyttämään. Runsas ominaisuustarjonta myös monimutkaisti käytettävyyttä.

## 5.5 Vaatimustenhallintaa JIRA:ssa

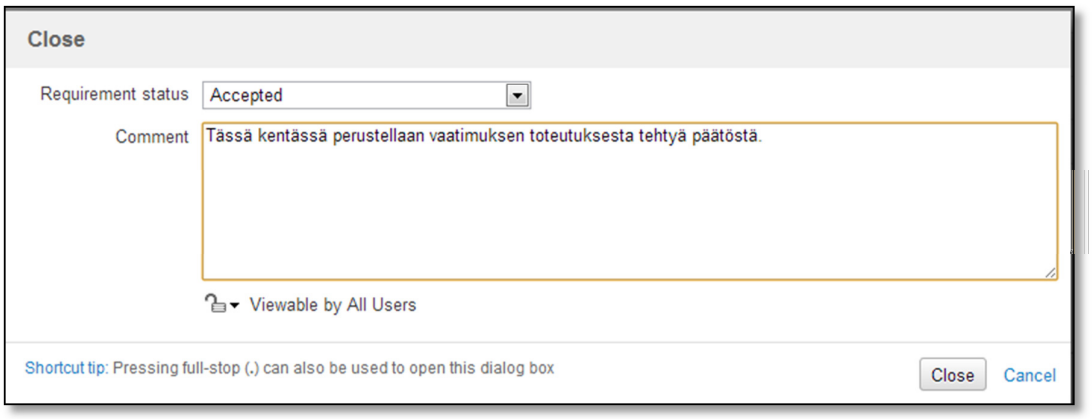
Vaatimuksen tiedot kirjataan JIRA-järjestelmään (kuvio 21). Uutta vaatimusta kirjatessa, lisätään ensin perustiedot, kuten vaatimuksen nimi ja kuvaus. Tarkempi selvitys lisätään mukaan erillisenä liitteenä. Kun vaatimus on käynyt priorisointiprosessin läpi, lisätään tietoihin vaatimuksen saamat yhteispisteet.




Kuvio 21. Uuden vaatimuksen lisääminen JIRA-järjestelmään

Kun vaatimuksen toteutuksesta on tehty päätös (Hyväksytty/Hylätty), se kirjataan JIRA-järjestelmään (kuvio 22). Jos vaatimus on hyväksytty, kirjataan tietoihin vaatimuksen hyväksyntä ja mahdollinen toteutusajankohta. Jos vaatimus hylätään,

kommentti-kentässä kerrotaan hylkäämisperusteet. Näin tieto päätöksestä tallentuu JIRA-järjestelmään.




Kuvio 22. Päätös vaatimuksen toteutuksesta

# 6 VAATIMUSMÄÄRITTELY KOHDEYRITYKSESSÄ

Tämän tutkimuksen yhtenä tutkimuskysymyksistä on selvittää, miten vaatimukset saadaan kerättyä sidosryhmiltä hallitusti talteen ja siirrettyä ne suunnitteluprosessiin. Koska sidosryhmien taustat ovat hyvin erilaiset, pelkästään yksi tapa tähän ei ole riittävä. Asiakkaan ja kohdeyrityksen organisaation taustojen ero voi olla huomattava, mutta myös organisaation sisällä erot saattavat olla suuret. Jos verrataan esimerkiksi myyntiä ja koulutusta, myynnillä ei ole niin tarkkaa tietoa ominaisuuksista kuin kouluttajilla. Kouluttajat ovat tekemisissä ohjelmistojen kanssa kouluttaessaan asiakkaita, myyjät keskittyvät enemmän ohjelmiston pääominaisuuksiin.

## 6.1 Nykytila ja haasteet

Oikeiden henkilöiden valitseminen projektiin on yksi vaatimusten tunnistamisen haasteista. Kommunikointiongelma on yksi yleisimmistä ongelmista ohjelmistoprojekteissa. Kun vaatimuksia määritellään asiakkaan kanssa, läsnä ei välttämättä ole henkilöt, jotka pystyisivät kommunikoimaan asiakkaan kanssa samalla tietotasolla. Tästä saattaa seurata se, että myyjä voi luvata sellaisia ominaisuuksia, jotka on mahdotonta toteuttaa. Näitä luvataan esimerkiksi helposti sen vuoksi, että kauppa ei menisi kilpailijalle. Aina on syytä miettiä miten paljon voidaan joustaa asiakkaiden vaatimuksista, jos kyseessä on huomattavan iso kauppa. On kuitenkin tärkeää, että voidaan ohjata asiakasta saamaan haluamansa ominaisuus, joka saattaa löytyä jo valmiina ominaisuutena. Tällainen ominaisuus saattaa toteutua hiukan toisella tavalla, mutta toiminta on sama. Näitä myyjät eivät aina huomaa, tästä syystä voidaan joutua tekemään sama ominaisuus toisella tavalla.

Yksi tapa parantaa kommunikointia sidosryhmien välillä on, että keskusteluissa on mukana sellaiset henkilöt, jotka pystyvät kommunikoimaan samalla tasolla. Tällä vältetään väärinymmärryksiä ja keskusteluista saadaan tehokkaampia.

Tietotekniikassa on tyypillistä, että ohjelmistonkehittäjät käyttävät sellaista terminologiaa, jota toinen osapuoli ei tunne. Tuntemattomien termien käyttö

helposti sekoittaa ja aiheuttaa väärinymmärryksiä sidosryhmien kesken. Tämä on hyvin yleinen ongelma aina, jos keskustelukumppanit eivät ole samalla teknisen osaamisen tasolla. Toisaalta vastuu ymmärtämisestä on myös kuulijalla, sillä jos ei ymmärrä mitä toinen tarkoittaa, on syytä pyytää selittämään tarkemmin. Väärinymmärryksiä syntyy, jos kuulija antaa sen mielikuvan, että hän ymmärtää asian vaikka todellisuudessa näin ei olisi. Tämä johtaa siihen, että kertoja olettaa viestin tulleen ymmärretyksi vaikka oikeasti se onkin väärin ymmärretty.

Vaatimuksiin vaikuttaa monta eri tekijää ja näiden tekijöiden eri näkökulmat. Asiakkaalla on oma näkemyksensä asioista ja ohjelmistojen toimittajalla on omansa. Näkemykseen vaikuttaa se, että asiakas ei ole välttämättä ohjelmistoalan ammattilainen. Asiakkaalla ei ehkä ole riittävän tarkkaa näkemystä siitä, mitä kaikkea tulisi ottaa huomioon ohjelmistohankinnan yhteydessä. Syy tähän saattaa olla se, että ohjelmisto ei ole kauttaaltaan konkreettisesti nähtävää tai ajatellaan ohjelmistojen olevan helposti muokattavissa jälkeenpäin. Näin tietyin rajoituksin onkin, mutta muutokset johtavat kuitenkin siihen, että virheiden ja väärinymmärrysten määrä kasvaa. Kun lisätöitä ja muutoksia tehdään, kustannukset kasvavat ja aikataulut venyvät.

Ilman vaatimusmäärittelyä on asiakkaan hankala jälkeenpäin vaatia toimittajalta korvauksia tai lisäyksiä ohjelmistoon, jos asiakkaan mielestä toimitus on puutteellinen. Vaatimusmäärittelyä voisi pitää jopa takuuna siitä, että ohjelmiston toimittaja toimittaa juuri sellaisen ohjelmistotuotteen, mitä on tilattu. Asia on sama myös toisinpäin, eli toimittaja voi varmistaa toimituksen oikeellisuuden vaatimusmäärittelyn avulla. Asiakkaan kanssa yhdessä laadittu vaatimusmäärittely on hyvä olla osana sopimusta.

Kun kaikki vaatimukset listataan ja aikataulutetaan, selkeyttää se myös työntekijöille päämäärää ja tavoitteita. Eri henkilöiden työkuormaa voidaan arvioida paremmin, ja työjärjestystä voisi optimoida tiedon pohjalta. Työntekijöille on motivoivaa, kun työ on järjestelmällisempää ja omaa työtään voi paremmin organisoida. Työntekijällä on suurempi mahdollisuus vaikuttaa omien tavoitteidensa asettamiseen, kun tulevaisuus on selkeämpi ja tulevat työt ovat aikaisemmin nähtävillä.

Työn kokonaisuuden kannalta tällä on suuri vaikutus, koska työntekijä näkee jo alussa mihin uudella ominaisuudella pyritään ja miten ominaisuus on hyväksytty toteutettavaksi. Selkeät työlistat auttavat tekijää hahmottamaan, mitä töitä on tulossa ja alussa laaditut vaatimukset auttavat arvioimaan työn onnistumisen ja seuraamaan sen etenemistä.

### 6.2 Vaatimusten analysointi ja vahvistaminen

Kun vaatimukset on kerätty, ne analysoidaan. Tässä vaiheessa vaatimukset käydään läpi tarkemmin ja varmistetaan, että jokainen sidosryhmän henkilö on ne ymmärtänyt. On mahdollista, että vaatimuksesta rakennetaan esimerkiksi prototyyppi, jolla pystytään tuomaan tarkasti vaatimuksen sisältö esiin. Prototyypin rakentaminen on tärkeä, jos vaatimuksen toteuttamiseen vaaditaan merkittävän iso projekti. Jos vaatimus ei ole työmäärältään merkittävä ja prototyypin rakentamiseen menisi enemmän resursseja kuin itse ominaisuuteen, kannattaa analysointi hoitaa toisella tavalla. Tällaisessa tapauksessa riittää esimerkiksi pelkästään keskustelu, jossa mukana osapuoli, jolta vaatimus on tullut.

Teoria ei anna tarkkaa ohjetta analysoinnista, vaan se on ihmisten välistä kommunikointia. Tässä kohden on kaikkien osapuolten oltava tarkkana, ettei mikään jää huomaamatta ja tehdä vääriä ratkaisuja.

Kohdeyrityksen tapauksessa analysointi on paras toteuttaa siinä vaiheessa, kun priorisointia käydään läpi. Vaatimus käydään perusteellisesti läpi sidosryhmän kanssa ennen sen pisteytystä. Vaatimuksen analysointia varten vaatimuksen esittelijän on valmistauduttava huolellisesti ja tuotava tilaisuuteen tarvittavat dokumentit. Näiden avulla varmistetaan, että vaatimus on selkeä kaikille osapuolille.

Vaatimusten vahvistamisvaiheessa varmistetaan, ettei käytännön työssä tule ikäviä yllätyksiä, esim. ominaisuus on vaikeampi toteuttaa mitä alun perin kuviteltiin.

Vaatimukset vahvistetaan esimerkiksi vertaisarvioinnilla. Vertaisarvioinnissa vaatimuksen dokumentteja ja ominaisuuksia arvioivat eri henkilöt, jolloin näkemys

laajenee. Vertaisarviointia suoritetaan siinä vaiheessa, kun vaatimuksia pisteytetään. Prosessissa on mukana eri alueiden asiantuntijat, jotka omien näkemystensä mukaan vahvistavat ominaisuuden. Tämä tapa soveltuu parhaiten kohdeyrityksen tarpeeseen ja tämän tutkimuksen tuloksena toteutuneen prosessin vaiheena. Muita menetelmiä ovat esimerkiksi katselmoinnit ja skenaariot.

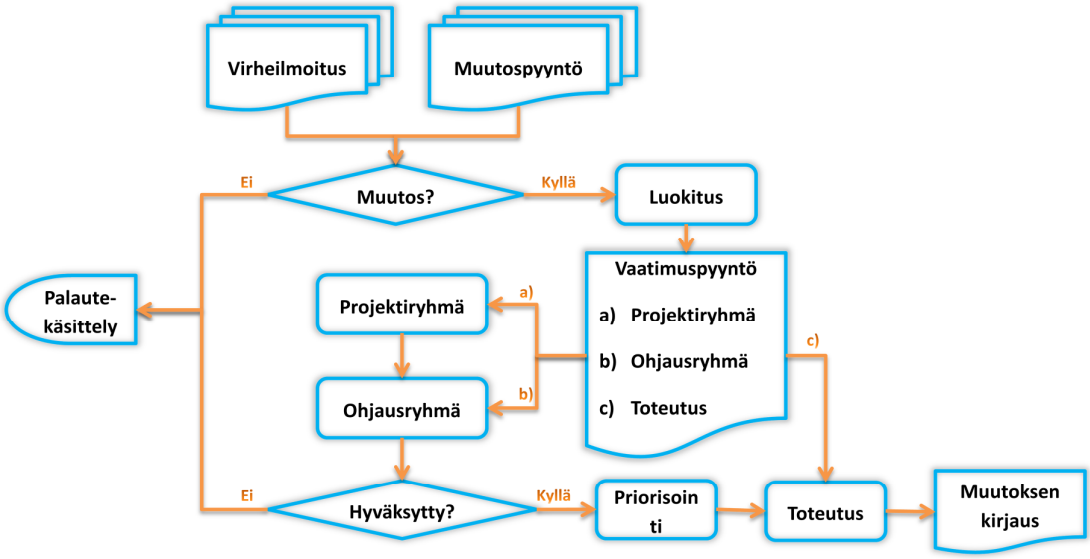
Koska vertaisarviointi sopii myös analysointiin, vaatimusten analysointi ja vahvistaminen suoritetaan samalla kun vaatimuksia priorisoidaan. Tällöin paikalla ovat oikeat henkilöt ja näiden vaiheiden avulla saadaan tietoa jaettua kaikille mukana oleville henkilöille.

### 6.3 Vaatimustenhallinta

Kun vaatimukset on määritelty, alkaa niiden hallintavaihe. Tässä vaiheessa keskitytään pääasiassa muutosten- ja versionhallintaan ja seurataan vaatimusten tiloja. Tämä tarkoittaa tiedon organisoimista ja hallintaa koko ohjelmiston elinkaaren ajan.

Vaatimuksille tulee lähes poikkeuksetta joitain muutoksia koko niiden elinkaaren ajan. Vain hyvin harvoin pystytään määrittelyvaihe viemään läpi niin, ettei mitään muutoksia tarvitse jälkeenpäin tehdä. Koska muutoksia tulee, niiden hallintaan täytyy osata varautua. Kaikki muutospyynnöt eivät tarkoita sitä että ne toteutetaan, vaan pyynnöt täytyy hyväksyä hyväksymisprosessin kautta.

Kuviossa 23 on määritelty kohdeyrityksen muutosprosessi.




Kuvio 23. Muutosprosessi

Kun sidosryhmältä tulee palautetta, sen sisältö analysoidaan. Palaute voi olla esimerkiksi virheilmoitus tai muutospyyntö. Jos kyseessä on virheilmoitus, se käsitellään normaalin palautekäsittelyprosessin mukaan. Jos kyseessä on muutospyyntö, luokitellaan ensin muutoksen tärkeys ja määritellään kuinka isosta muutoksesta on kyse. Tarvittaessa vaatimus käsitellään projektiryhmässä, jonka jälkeen ohjausryhmä hyväksyy sen. Jos vaatimusta ei hyväksytä, se siirtyy normaaliin palautekäsittelyyn, aivan kuten virheilmoituksetkin. Jos on kyse pienestä muutoksesta, se ohjataan suoraan toteutukseen. Isot muutokset menevät priorisointivaiheen läpi. Lopuksi kun muutos on toteutettu, kirjataan muutos järjestelmään.

Vaatimusten versionhallinta toteutetaan samaan tapaan kuin lähdekoodien ja muiden dokumenttien versiointi. Dokumentit säilytetään SVN-versiohallintajärjestelmässä ja viimeisimmät dokumentit on linkitetty JIRA-järjestelmään. Tällä tavalla jokainen, joka on kiinnostunut kyseisestä vaatimuksesta, pääsee helposti sitä tarkastelemaan.

Koska JIRA:ssa pidetään päivitettynä viimeisimmät tiedot ja vaiheet vaatimuksesta, voidaan sitä käyttää hyvin myös vaatimusten tilan seurantaan ja

jäljittämiseen. JIRA:ssa on mahdollista linkittää eri vaatimuksia keskenään ja laatia isompia vaatimuskokonaisuuksia ja ne on jaettu pienempiin kokonaisuuksiin.

### 6.4 Mistä vaatimuksia tulee?

Se mistä vaatimuksia tulee, selvitettiin käyttämällä kokemuspohjaista tietoa aikaisemmista projekteista. Seuraavassa listassa on lueteltuna sidosryhmät, joilta vaatimuksia tulee eniten:

- asiakkaat
- huolto
- käyntiinajo
- maahantuojat/tytäryhtiöt
- myynti
- suunnittelu.

Vaatimuksia tulee paljon ja ne ovat eritasoisia. Jotkut vaatimukset ovat pitkälle jalostettuja, kun toiset ovat vain muutamalla sanalla kerrottuja ideoita. Vaatimukset saattavat olla alun perin joko asiakkaan tai yrityksen henkilöiden löytämiä. Vaatimusten yleisin toimitustapa on sähköposti, josta ne on helppo kerätä talteen. Yrityksessä on käytössä palautejärjestelmä, jonne esimerkiksi huolto ja käyntiinajo raportoivat asiakkaalla esiintyneitä tilanteita. Palautejärjestelmää käytetään usein myös uusien vaatimusten esittämispaikkana.

Ehdotetut vaatimukset kerätään talteen JIRA-järjestelmään odottamaan niiden priorisointia. Vaatimuksen esittäjältä tiedustellaan kaikki tarpeellinen tieto kyseisestä ominaisuudesta, jotta lähtötiedot saataisiin mahdollisimman kattavaksi. Alkuperäisen raportoijan tietojen täytyy näkyä järjestelmässä, jotta on mahdollista jälkeenpäin jäljittää kuka idean alun perin esitti. Tämä tehdään esimerkiksi sen vuoksi, että asioita voidaan kysyä suoraan esittäjältä itseltään.

Vaatimukset pyritään toimittamaan samojen henkilöjen kautta keskitetysti, jolloin vaatimusten hallinta on helpompaa. Näin voidaan jo alkuvaiheessa ilmoittaa, että tämä vaatimus on ollut listoilla ja on tullut hylätyksi jonkin syyn takia. Vaatimuksen laatija ei aina tiedä kenelle asia kuuluu, mutta kun vaatimus saapuu

ohjelmistosuunnitteluun, se ohjataan oikealle henkilölle. Keskitetty kirjaaminen vähentää päällekkäisyyttä.

### 6.5 Vaatimustenhallinta osana osaamisen johtamista

Tässä kappaleessa tutkitaan kohdeyrityksen vaatimustenhallinnan kehittämistä osana ohjelmistokehitystä ja osaamisen johtamista. Lisäksi perehdytään siihen, voiko projektin hallintaa ja työnohjausta parantaa sillä, että vaatimukset ovat hallinnoitu etukäteen paremmin. Voiko työn aikatauluttaa jo vaatimusten määrittelyvaiheessa tietylle suunnittelijalle, vaikka vaatimusta ei heti toteutetakaan. Lisäksi tarkastellaan myös näiden asioiden vaikutusta motivaatioon ja yleisiin toimintatapoihin.

Vaatimustenhallinnan yksi keskeisin tavoite on tarvittavien resurssien ennakointi ja oikea osaaminen oikeaan aikaan. Näin projektin voi helpommin ohjata henkilölle, jolla on tämän osa-alueen osaamista ja tietotaitoa. Näiden henkilöiden resurssit voi perustellusti varata ja aikatauluttaa. Jos tällaista järjestelmää ei ole, resurssit kuormittuvat epätasaisesti ja projektit viivästyvät. Työn jakautuminen ei ole oikeudenmukaista, mikä vaikuttaa myös työmotivaatioon.

Kun aikataulutetaan resurssi, jolla on halutunlaista osaamista, laaditaan resurssianalyysi. Resurssianalyysin avulla pystytään jakamaan resursseja ja tietoa henkilöiden tietotaidosta. Resurssianalyysiä täytyy ylläpitää aktiivisesti, koska henkilöt kehittyvät ja tietotaidon määrä kasvaa. Resurssianalyysia voi käyttää henkilöstön kouluttamistarpeen selvittämiseen, jos esimerkiksi yrityksen rauhallisempana aikana halutaan kouluttaa henkilökuntaa. Rekrytoitaessa uutta henkilökuntaa voi resurssianalyysin perusteella määritellä hakijan oikean tyyppinen osaaminen. Tällä tavoin osaamista voi laajentaa ilman erillistä koulutusta. Projektien käynnistyksen aikana ei ole aikaa/mahdollisuutta rekrytoida uusia työntekijöitä, vaan henkilön on löydyttävä jo olemassa olevista resursseista.

Taulukossa 1 esitetään osaamisen dokumentointia ja tietojen ylläpitoa. Taulukko näyttää resurssin (henkilön) nimen, toimenkuvan ja erikoisosaamisen. Resurssien roolien täytyy olla selvät. Yhdellä henkilöllä voi olla enemmän kuin yksi rooli. Jos

kyseessä on prosessi, missä resurssi on ns. omistaja, hän vastaa kyseisestä osasta ohjelmistoa. Omistaja määrittää tehtävät projektin eri henkilöille, aikataulun sallimissa rajoissa. Taulukkoa hyödynnetään osaamistarpeen kartoittamisessa sekä rekrytointitilanteissa että koulutusta suunniteltaessa. Taulukosta voi helposti nähdä, minkä osa-alueen osaamiselle on lisätarvetta ja mitä puolestaan on jo olemassa.

Taulukko 1. Resurssianalyysitaulukko

| **Matti Meikäläinen** | Software designer |
|---|---|
| ***Special know-how:***<br><br>Communication with controls<br>Back process<br>Old systems (ControlLink, LaserLink etc.)<br><br>Lasers<br><br>***Roles:***<br><br>Project team member<br><br>**Process owner:**<br>– DeviceScreens.dll<br>– DeviceScreens.xml<br>– Alarms.dll<br>– ControlConnector.dll<br>– ControlLink, LaserLink | |
| **Maija Teikäläinen** | Documentation specialist |
| ***Special know-how:***<br><br>Documentation<br><br>***Roles:***<br><br>Project team member<br><br>**Process owner:**<br>– User manuals<br>– Technical documents | |
| **Pekka Testaaja** | Testing engineer |
| ***Special know-how:***<br><br>Testing<br>Development of testing<br><br>***Roles:***<br>Test lead<br>Integration test lead | |

System test lead
User Acceptance Test Lead (UAT)
Project team member

**Process owner:**
- Testing

Kun uusia vaatimuspyyntöjä tulee, joko asiakasprojektien myötä tai jollain muulla tavalla, ne käsitellään projektiryhmän kesken läpi. Hyvä tapa tähän on pitää viikoittainen palaveri, missä kaikkien henkilöiden työt käydään läpi ja samalla käsitellään tulleet uudet vaatimukset. Palaverissa mietitään, kenelle kyseinen työ sopisi parhaiten. Kriteerinä tähän voi olla kyseisen henkilön tietotaito sekä hänen aikataulunsa. Jos vaadittavaa osaamistaustaa omaava resurssi löytyy, mutta resurssin aikatauluun ei työ sovi, valitaan toinen resurssi ja mietitään miten koulutus suoritetaan.

# 7 VAATIMUKSIEN PRIORISOINTI

Tutkimuksen yhtenä tutkimusongelmista oli se, miten vaatimukset priorisoidaan järkevällä ja oikeudenmukaisella tavalla. Tässä kappaleessa haetaan ratkaisua tähän ongelmaan ja esitellään vaatimusten priorisointiin kehitettyä taulukkoa. Taulukon avulla voidaan vaatimuksia järkiperustein laittaa tärkeysjärjestykseen.

Kaikkia vaatimuksia on mahdoton toteuttaa johtuen rajallisesta aikataulusta sekä raha- ja henkilöstöresursseita. Syy miksi vaatimus hylätään voi olla esimerkiksi se, että vaatimus on toteuttamiskelvoton. Vaatimuksen hylkääminen täytyy aina selvittää ja perustella.

Tietyt vaatimukset voidaan toteuttaa ilman priorisointiprosessia. Tällaisia vaatimuksia ovat esimerkiksi järjestelmän käyttämiseen liittyvät vaatimukset. Vaatimukset saattavat olla sidoksissa koneen mekaaniseen ominaisuuteen, joka tarvitsee ohjelmiston tukea toimiakseen. Nämä vaatimukset on käyty läpi siinä vaiheessa, kun kyseinen koneen ominaisuus on päädytty toteuttamaan. Näitä vaatimuksia ei tässä tutkimuksessa käsitellä, vaan keskitytään pelkkiin ohjelmistovaatimuksiin.

Kohdeyrityksessä asiakkailta tulevat vaatimukset priorisoidaan korkealle, jos vaatimukset vain ovat perusteltuja. Tämä johtuu siitä, että ohjelmistot eivät ole päätuote, vaan aina osa isompaa toimitusta. Ohjelmiston osuus koneen hinnasta on noin 5 prosenttia, riippuen koneen tai järjestelmän laajuudesta. Ohjelmistoilla tuetaan koneiden myyntiä ja usein joudutaan osittain tinkimään vaatimusten kriteereistä. Tähän pätee sama sääntö kuin kaikille muillekin vaatimuksille, vaatimus täytyy hyväksyä toteutettavaksi jonka jälkeen se priorisoidaan. Yhtään vaatimusta ei tuoda tämän prosessin ohi.

## 7.1 Vaatimusten pisteytysprosessi

Kun vaatimukset on kerätty järjestelmään ja ne on tarpeeksi hyvin dokumentoitu, aloitetaan pisteytysprosessi. Prosessiin osallistuu tarkasti valittu ryhmä henkilöitä, joilla on kokemusta ja näkemystä monilta eri osaamisalueilta. Ryhmässä täytyy

olla läsnä henkilöitä ainakin suunnittelu-, huolto-, käyntiinajo- sekä koulutusosastolta. Tällä varmistetaan, ettei mitään tärkeää osa-aluetta unohdeta. Tätä ryhmää ei kuitenkaan kutsuta koolle, jos vaatimuksia on vain muutama.

Jokainen vaatimus esitellään ryhmälle yksitellen ja selvitetään tarkasti mitä sillä tarkoitetaan. Tämän esittelyn tekee joko vaatimuksen ehdottanut osapuoli tai henkilö, joka on siihen huolella paneutunut ja sen sisäistänyt. On tärkeää että henkilö on todella tietoinen mitä vaatimuksella tavoitellaan, että hän osaa sen selittää muille. Jos vaatimus ymmärretään väärin, voidaan tehdä sen perusteella väärä päätös.

Esittelyn jälkeen jokainen ryhmän jäsen pisteyttää vaatimukset pisteytystaulukon mukaisesti. Pisteytystaulukko on kuvattu tarkemmin seuraavassa kappaleessa. Tässä vaiheessa voi myös esittää tarkentavia lisäkysymyksiä. Pisteiden antaminen on suoraviivaista ja esitäytetyt kohdat helpottavat valintaa.

Kun vaatimukset on pisteytetty ja annetut pisteet on yhteenlaskettu, vaatimukset siirretään JIRA-järjestelmään. Näin vaatimukset ja niiden saamat pisteet säilyvät tallessa. Vaikka vaatimus hylätään, se jää talteen järjestelmään ja jälkeenpäin voidaan todeta syyt miksi se tuli aikoinaan hylätyksi.

Pisteytyksen jälkeen vaatimukset esitellään ohjausryhmälle. Ohjausryhmä tekee lopullisen ratkaisun siitä, toteutetaanko kyseinen vaatimus vai ei. Jos vaatimus toteutetaan, määritetään sen prioriteetti. Tähän käytetään apuna pisteytystaulukosta tulleita pisteitä ja taloudellisia lukuja. Kun vaatimus on hyväksytty, sen tila muutetaan JIRA-järjestelmässä ”tehtäväksi” ja sille määritetään resurssi ja aikataulu. Vaatimuksen seuraava prosessivaihe on tekninen määrittely.

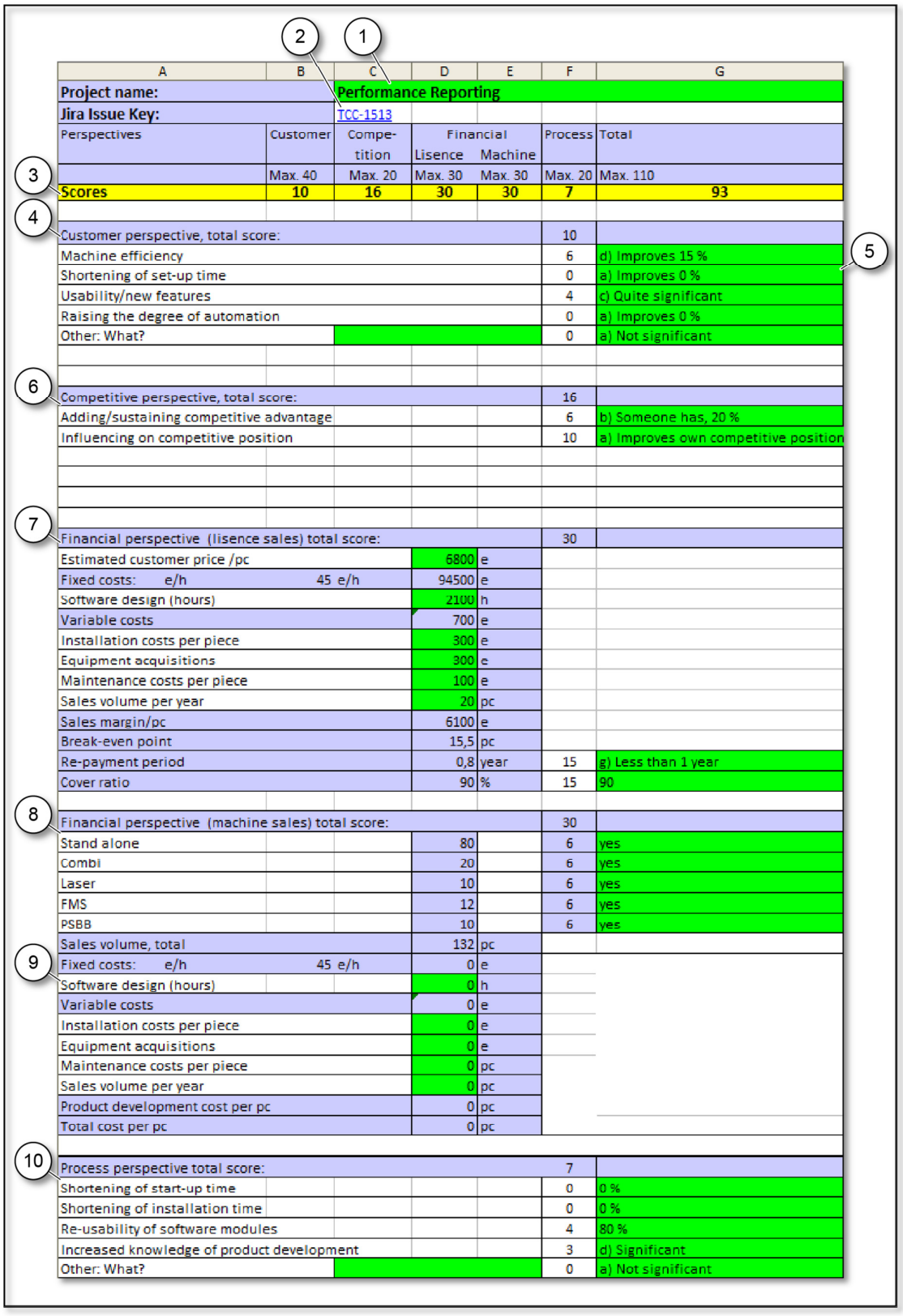
### 7.2 Vaatimuksien pisteytystaulukko

Vaatimusten pisteytystaulukon tehtävänä on tarjota työkalu, jolla määritellään vaatimuksen tärkeys ja tarpeellisuus. Ilman mitään työkalua sitä on vaikea määrittää. Tässä tutkimuksessa käytetty työkalu on Microsoft Excel -taulukkolaskentaohjelmalla tehty yksinkertainen taulukko (kuvio 24), jossa käyttäjä

kirjoittaa arvot taulukon vihreänvärisiin, esitäytettyihin kenttiin. Taulukko laskee arvot automaattisesti yhteen ja tiedot on helppo kerätä talteen. Pisteytyksen jälkeen taulukko voidaan liittää JIRA-järjestelmään.

Pisteytystaulukon (kuvio 24) pääkohdat ovat:

1. Vaatimuksen tai projektin nimi
2. Linkki Atlassian JIRA-järjestelmään
3. Yhteispisteet
4. Asiakasnäkökulma
5. Arviointikenttä
6. Kilpailunäkökulma
7. Lisenssimyynnin taloudellinen näkökulma
8. Konemyynnin taloudellinen näkökulma
9. Vaatimuksen taloudellisen näkökulman aputyökalu (ei vaikuta pisteisiin)
10. Prosessinäkökulma.



| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| | **Project name:** | | **Performance Reporting** | | | | |
| | **Jira Issue Key:** | | TCC-1513 | | | | |
| | Perspectives | Customer | Compe-tition | Financial Lisence | Financial Machine | Process | Total |
| | | Max. 40 | Max. 20 | Max. 30 | Max. 30 | Max. 20 | Max. 110 |
| 3 | **Scores** | **10** | **16** | **30** | **30** | **7** | **93** |
| 4 | Customer perspective, total score: | | | | | 10 | |
| | Machine efficiency | | | | | 6 | d) Improves 15 % |
| | Shortening of set-up time | | | | | 0 | a) Improves 0 % |
| | Usability/new features | | | | | 4 | c) Quite significant |
| | Raising the degree of automation | | | | | 0 | a) Improves 0 % |
| | Other: What? | | | | | 0 | a) Not significant |
| 6 | Competitive perspective, total score: | | | | | 16 | |
| | Adding/sustaining competitive advantage | | | | | 6 | b) Someone has, 20 % |
| | Influencing on competitive position | | | | | 10 | a) Improves own competitive position |
| 7 | Financial perspective (lisence sales) total score: | | | | | 30 | |
| | Estimated customer price /pc | | | 6800 | e | | |
| | Fixed costs: e/h 45 e/h | | | 94500 | e | | |
| | Software design (hours) | | | 2100 | h | | |
| | Variable costs | | | 700 | e | | |
| | Installation costs per piece | | | 300 | e | | |
| | Equipment acquisitions | | | 300 | e | | |
| | Maintenance costs per piece | | | 100 | e | | |
| | Sales volume per year | | | 20 | pc | | |
| | Sales margin/pc | | | 6100 | e | | |
| | Break-even point | | | 15,5 | pc | | |
| | Re-payment period | | | 0,8 | year | 15 | g) Less than 1 year |
| | Cover ratio | | | 90 | % | 15 | 90 |
| 8 | Financial perspective (machine sales) total score: | | | | | 30 | |
| | Stand alone | | | 80 | | 6 | yes |
| | Combi | | | 20 | | 6 | yes |
| | Laser | | | 10 | | 6 | yes |
| | FMS | | | 12 | | 6 | yes |
| | PSBB | | | 10 | | 6 | yes |
| | Sales volume, total | | | 132 | pc | | |
| 9 | Fixed costs: e/h 45 e/h | | | 0 | e | | |
| | Software design (hours) | | | 0 | h | | |
| | Variable costs | | | 0 | e | | |
| | Installation costs per piece | | | 0 | e | | |
| | Equipment acquisitions | | | 0 | e | | |
| | Maintenance costs per piece | | | 0 | pc | | |
| | Sales volume per year | | | 0 | pc | | |
| | Product development cost per pc | | | 0 | pc | | |
| | Total cost per pc | | | 0 | pc | | |
| 10 | Process perspective total score: | | | | | 7 | |
| | Shortening of start-up time | | | | | 0 | 0 % |
| | Shortening of installation time | | | | | 0 | 0 % |
| | Re-usability of software modules | | | | | 4 | 80 % |
| | Increased knowledge of product development | | | | | 3 | d) Significant |
| | Other: What? | | | | | 0 | a) Not significant |

Kuvio 24. Vaatimuksien pisteytys Excel-taulukossa

**Pisteytystaulukon kohta 1: Priorisoitavan vaatimuksen tai projektin nimi** (kuvio 25). Nimen tulee olla selkeä ja kuvaava, jotta sen avulla sidosryhmät tietävät heti mistä on kyse. Kuviossa on esimerkkinä vaatimus nimeltään Performance Reporting. Vaatimus voi olla laaja tai jokin yksittäinen, pienempi ominaisuus.


Kuvio 25. Vaatimuksen tai projektin nimi

**Pisteytystaulukon kohta 2: Linkki Atlassian JIRA -tehtävään** (kuvio 26). JIRA-linkki avaa tehtävän tarkemmat tiedot. Koska JIRA on web-sovellus, linkki on URL-osoite *(Uniform Resource Locator)* sovelluksen palvelimelle. Vaatimusten tietoja ja dokumentteja ylläpidetään JIRA:ssa ja myös tämä pisteytystaulukko voidaan tallentaa JIRA-järjestelmään. Tämän linkin avulla voidaan myös antaa lisätietoja tai kommentteja vaatimuksesta.


Kuvio 26. Linkki Atlassian JIRA -tehtävään

**Pisteytystaulukon kohta 3: Yhteispisteet** (kuvio 27)**.** Yhteispisteisiin kerätään kaikkien osa-alueiden yhteenlasketut pisteet. Eri osa-alueiden pisteet ovat painotettu siten, että jostain osa-alueesta saa enemmän pisteitä kuin toisesta. Esimerkiksi talouden näkökulmasta annetut pisteet ovat painoarvoltaan suuremmat kuin prosessin näkökulmasta tulleet pisteet. Muita painoarvoltaan suuria osa-alueita ovat asiakkaan ja kilpailun näkökulmasta tulleet pisteet. Mitä suurempi pistemäärä on, sitä tärkeämpänä ominaisuutta pidetään. Yhteispistemäärää verrataan kaikkien priorisoitavien vaatimusten yhteispisteisiin ja tällä tavoin asetetaan prioriteettijärjestys. Kun kaikkien henkilöiden antamat pisteet on laskettu yhteen, pisteet siirretään JIRA-järjestelmään.

| Perspectives | Customer | Compe-tition | Financial | | Process | Total |
|---|---|---|---|---|---|---|
| | | | Lisence | Machine | | |
| | Max. 40 | Max. 20 | Max. 30 | Max. 30 | Max. 20 | Max. 110 |
| Scores | 10 | 16 | 30 | 30 | 7 | 93 |

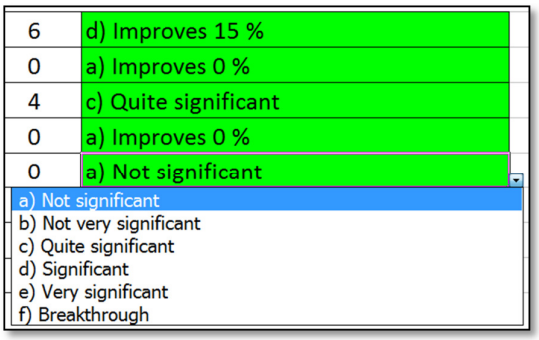
Kuvio 27. Yhteispisteet

**Pisteytystaulukon kohta 4: Asiakasnäkökulma** (kuvio 28). Asiakasnäkökulmassa esitellään ne edut, jotka tuovat lisäarvoa asiakkaalle, esimerkiksi lisää tuotannon tehokkuutta, lyhentää asetusaikoja, parantaa käytettävyyttä tai lisää automaatioastetta.

| Customer perspective, total score: | | 10 | |
|---|---|---|---|
| Machine efficiency | | 6 | d) Improves 15 % |
| Shortening of set-up time | | 0 | a) Improves 0 % |
| Usability/new features | | 4 | c) Quite significant |
| Raising the degree of automation | | 0 | a) Improves 0 % |
| Other: What? | | 0 | a) Not significant |

Kuvio 28. Asiakasnäkökulma

**Pisteytystaulukon kohta 5: Arviointikenttä** (kuvio 29). Arviointikenttään annetaan jokaiselle kohdalle priorisointipisteet sen mukaan, kuinka tärkeänä kohtaa pitää. Priorisointipisteet ovat kentissä esitäytettyinä ja niiden arvot on määritelty jo taulukon kehitysvaiheessa. Arvojen määrittelyssä on ollut mukana henkilöitä yrityksen eri osastoilta, varmistaen kattavan näkemyksen aiheesta. Esitäytetty priorisointipisteytys auttaa käyttäjää tekemään päätöksen pistemäärästä.




Kuvio 29. Arviointikenttä

**Pisteytystaulukon kohta 6: Kilpailunäkökulma** (kuvio 30)**.** Kilpailunäkökulma-kentässä määritellään pisteet sen mukaan, onko kilpailijoilla vastaavaa ominaisuutta tarjolla. Kilpailua tarkastellaan kahdesta näkökulmasta, ensinnäkin siitä, kuinka monella prosentilla kilpailijoista tämä ominaisuus jo on tarjolla ja kuinka paljon ominaisuus parantaa kilpailukykyä. On tärkeää, että tuotteessa pystytään tarjoamaan vähintään samat ominaisuudet kuin kilpailija, ja tietenkin mieluiten enemmänkin. Tästä johtuen tällä näkökulmalla on paljon painoarvoa.

| | | | | |
|---|---|---|---|---|
| Competitive perspective, total score: | | | 16 | |
| Adding/sustaining competitive advantage | | | 6 | b) Someone has, 20 % |
| Influencing on competitive position | | | 10 | a) Improves own competitive position |

Kuvio 30. Kilpailunäkökulma

**Pisteytystaulukon kohta 7: Lisenssimyynnin taloudellinen näkökulma** (kuvio 31). Tässä näkökulmassa lasketaan onko ominaisuuden toteutus taloudellisesti kannattavaa. Jokaiselle vaatimukselle tätä ei pystytä tekemään, koska kaikkia ominaisuuksia ei lisensioida. Jos vaatimus on sellainen, jota voidaan myydä omana tuotteenaan erikseen, se pisteytetään taloudellisen näkökulman mukaisesti. Tässä kohdassa saatuja pisteitä pidetään tärkeämpänä kuin muissa kohdissa saatuja pisteitä. Jos ominaisuus esimerkiksi maksaa itsensä takaisin lyhyessä ajassa ja sen jälkeen alkaa tuottaa, se kannattaa tehdä.

Taloudellisessa näkökulmassa huomioidaan arvioitu asiakashinta kappaleelta ja ominaisuuden tekemiseen käytetyt tunnit ja siitä syntyneet kulut. Lisäksi otetaan huomioon muita kuluja, joita syntyy esimerkiksi asennuksesta, laitteista ja ylläpidosta. Myyntivolyymi täytyy myös voida ennustaa ja sen avulla voidaan laskea takaisinmaksuaika. Myyntivolyymin ennustaminen ei ole aina kovin helppoa, joten sillä voidaan pisteytys johtaa harhaan. Myyntivolyymin selvittämiseen voidaan kerätä tietoa esimerkiksi myynniltä ja mahdollisesti tehdä asiakaskyselyitä eli markkinaselvitystä.

| Financial perspective (lisence sales) total score: | | | 30 | |
|---|---|---|---|---|
| Estimated customer price /pc | 6800 | e | | |
| Fixed costs: e/h 45 e/h | 94500 | e | | |
| Software design (hours) | 2100 | h | | |
| Variable costs | 700 | e | | |
| Installation costs per piece | 300 | e | | |
| Equipment acquisitions | 300 | e | | |
| Maintenance costs per piece | 100 | e | | |
| Sales volume per year | 20 | pc | | |
| Sales margin/pc | 6100 | e | | |
| Break-even point | 15,5 | pc | | |
| Re-payment period | 0,8 | year | 15 | g) Less than 1 year |
| Cover ratio | 90 | % | 15 | 90 |

Kuvio 31. Lisenssimyynnin taloudellinen näkökulma

**Pisteytystaulukon kohta 8: Konemyynnin taloudellinen näkökulma** (kuvio 32)**.** Toinen taloudellinen näkökulma on se, miten moneen konemalliin tai järjestelmään vaatimus vaikuttaa. Konemyynnin taloudellisessa näkökulmassa ei tarkastella vaatimusta tuoton kautta, vaan sitä miten moneen konemalliin se voidaan asentaa. Tarkoitus on selvittää vaatimuksen kattavuutta. Jos tuki on useammalle koneelle, sen tarve on suurempi.

| Financial perspective (machine sales) total score: | | | 30 | |
|---|---|---|---|---|
| Stand alone | 80 | | 6 | yes |
| Combi | 20 | | 6 | yes |
| Laser | 10 | | 6 | yes |
| FMS | 12 | | 6 | yes |
| PSBB | 10 | | 6 | yes |
| Sales volume, total | 132 | pc | | |

Kuvio 32. Konemyynnin taloudellinen näkökulma

**Pisteytystaulukon kohta 9: Vaatimuksen taloudellisen näkökulman aputyökalu** (kuvio 33). Taloudellisen näkökulman aputyökalu on tarkoitettu helpottamaan määrittelyä kannattavuuden näkökulmasta. Se kertoo mm. vaatimuksen takaisinmaksuajan ja mahdolliset tuotot. Tähän määritellyt arvot eivät vaikuta loppupisteisiin, vaan vain ohjaavat pisteyttäjää oikeaan suuntaan.

| | | | |
|---|---|---|---|
| Fixed costs: e/h | 45 e/h | 0 | e |
| Software design (hours) | | 0 | h |
| Variable costs | | 0 | e |
| Installation costs per piece | | 0 | e |
| Equipment acquisitions | | 0 | e |
| Maintenance costs per piece | | 0 | pc |
| Sales volume per year | | 0 | pc |
| Product development cost per pc | | 0 | pc |
| Total cost per pc | | 0 | pc |

Kuvio 33. Taloudellisen näkökulman aputyökalu

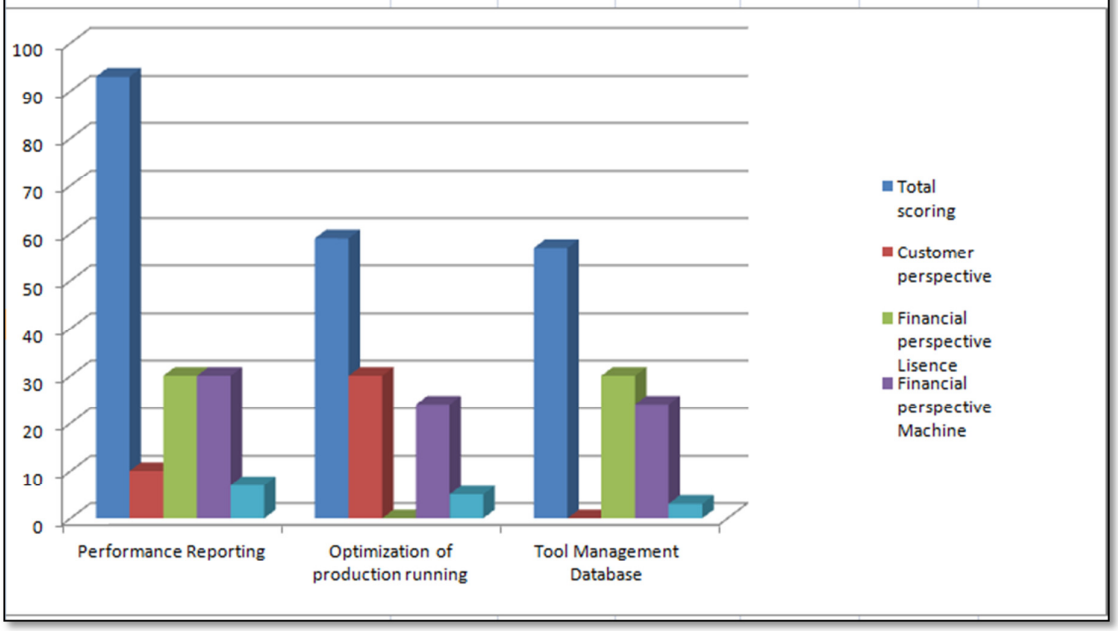
**Pisteytystaulukon kohta 10: Prosessinäkökulma** (kuvio 34). Prosessinäkökulmalla tarkoitetaan kohdeyrityksen omaa tuotantoa. Jos vaatimus lyhentää käyntiinajoaikaa tai asennusaikaa, se saa enemmän pisteitä, sillä muutokset vaikuttavat suoraan kustannuksiin. Tärkeitä ominaisuuksia ovat myös uudelleenkäytettävyys, jolloin säästöjä saadaan valmiilla moduuleilla. Jo olemassa olevia moduuleja muokkaamalla tai laajentamalla saadaan uusi ominaisuus tehtyä. Jos uusi ominaisuus lisää tietotaitoa, on sillä merkitystä pisteytyksessä.

| | | |
|---|---|---|
| Process perspective total score: | 7 | |
| Shortening of start-up time | 0 | 0 % |
| Shortening of installation time | 0 | 0 % |
| Re-usability of software modules | 4 | 80 % |
| Increased knowledge of product development | 3 | d) Significant |
| Other: What? | 0 | a) Not significant |

Kuvio 34. Prosessinäkökulma

**Yhteenvedossa** (kuvio 35) nähdään kooste kaikkien vaatimusten pisteistä. Koska kaikkien pisteytettävien vaatimusten lopulliset pistemäärät ovat näkyvissä, voidaan helposti päättää priorisointijärjestyksestä. Pisteet on jaettu jokaisen osa-alueen mukaan ja yhteenlaskettu summa on näkyvillä. Koska pisteissä on valmiiksi huomioitu painoarvot, ovat yhteispisteet suoraan vertailukelpoisia.

| Projects | Total scoring | Customer perspective | Competitive perspective | Financial perspective Lisence | Financial perspective Machine | Process perspective |
|---|---|---|---|---|---|---|
| **Performance Reporting** | 93 | 10 | 16 | 30 | 30 | 7 |
| **Optimization of production running** | 59 | 30 | | 0 | 24 | 5 |
| **Tool Management Database** | 57 | 0 | | 30 | 24 | 3 |




Kuvio 35. Yhteenveto

# 8 PISTEYTYSTAULUKON JATKOKEHITYS

Taulukossa tulee olla huomioituna myös asiakastarpeen tunnistaminen tai sen pisteytys. Jos idea on tullut jo suoraan asiakkaalta, tarpeen tunnistaminen on helpompaa, mutta vaatimuksen täytyy olla myös hyvin perusteltu. On paljon eri tahoilta tulleita vaatimuksia, joita ei ole riittävän hyvin mietitty asiakkaan näkökulmasta. On syytä tutkia saatu materiaali tarkasti läpi, jotta kaikki tarpeellinen on huomioitu.

Asiakastarpeen tunnistamisessa voidaan tehdä myös asiakaskyselyä tai haastatteluja. Vaatimuksen perusteella voidaan rajata tietyt asiakkaat, kenelle uusi ominaisuus sopisi. Paras tapa asiakastarpeen tunnistamiseen on kuitenkin vierailla asiakkaan luona. Asiakkaan tuotantoon tutustuminen ja koneen käyttäminen antavat tarkkaa tietoa asiakkaan tuotannosta ja mahdollisesta ongelmasta. Saattaa olla, että asiakas esimerkiksi vain käyttää konetta väärin. Tällöin asiakkaalle voidaan helposti näyttää parempi tapa toimia, eikä ohjelmistoon tarvitse tehdä muutoksia. Asiakkailla vierailut ovat kuitenkin kallis tapa, koska kohdeyrityksen tapauksessa asiakkaat ovat usein Suomen rajojen ulkopuolella. Tästä johtuen asiakkaalla vieraillaankin melko harvoin.

Vaatimusten pisteytyksessä on käytetty hyvin yksinkertaista tapaa laskea kustannukset ja tuotto-odotukset. Tästä kuitenkin huomataan se, onko kyseisen vaatimuksen toteuttaminen taloudellisesti kannattavaa. Jos vaatimus tulee asiakkaalta, on huomioitavaa että vaatimuksen toteuttamiseen kuluva summa saadaan takaisin jo kyseiseltä asiakkaalta, vaikka ominaisuudelle ei olisikaan muuten kysyntää.

Vaatimusten pisteytyksissä käytetyissä laskelmissa ei ole huomioitu nykyarvolaskentaa. Se ei ole aina välttämätöntä, koska joidenkin vaatimusten tekemiseen menee aikaa päivästä muutamaan viikkoon. On kuitenkin vaatimuksia, joiden tekemiseen menee useita kuukausia ja kustannuksia kertyy enemmän. Näissä tapauksissa olisi hyvä tehdä laajemmin laskelmat ja huomioida myös markkinointikustannukset.

# 9 YHTEENVETO

Tämän opinnäytetyön tutkimuskysymyksinä oli löytää ratkaisut siihen, miten vaatimukset saadaan hallitusti kerättyä ja priorisoitua sekä löytää kohdeyrityksen tarpeisiin sopiva, koko organisaatiolle avoinna oleva vaatimustenhallintaohjelmisto. Vaatimusten hallittu kerääminen ratkaistiin siten, että kaikki vaatimukset tulevat suunnitteluosastolle aina tietyille henkilöille. Nämä henkilöt kirjaavat tulleet vaatimukset JIRA-järjestelmään, jolloin ne tulevat tarkasti ja keskitetysti koottua talteen.

Vaatimusten priorisointia varten kehitettiin pisteytystaulukko, jolla priorisointia tekevien henkilöiden työtä helpotetaan. Yrityksen linjauksen mukaan määritettyjen painoarvojen perusteella henkilöä ohjataan oikeaan suuntaan priorisointia tehdessä. Pisteytystaulukko on Excel-taulukko, jota on yksinkertaista käyttää ja sen toteutus on edullista.

Vaatimustenhallintaohjelmistovaihtoehdoista kohdeyrityksen tarpeisiin sopi parhaiten Atlassian JIRA. JIRA todettiin joustavaksi työkalu, johon on helppo lisätä ominaisuuksia ja jota voi räätälöidä omiin tarpeisiin sopivaksi. Koska JIRA oli yrityksessä jo käytössä, ei yrityksen tarvinnut investoida uuteen järjestelmään ja ohjelmiston käyttö oli tuttua. JIRA on web-pohjainen ohjelmisto, joten se on myös avoin koko yrityksen organisaatiolle.

Vaatimustenhallintaa on tutkittu paljon ja siitä on esitetty lukuisia teorioita. Tätä tutkimusta varten löytyi runsaasti lähdemateriaalia ja tiedon etsiminen runsauden keskeltä oli työlästä. Tutkimuksen tuloksista on ollut paljon hyötyä kohdeyritykselle ja se on auttanut vaatimusten kokonaisvaltaisessa hallinnassa ohjelmistoprojekteissa. Vaatimusten hallinnan kehittämisessä on ollut selvästi hyötyä myös projektien läpiviemisessä, sillä se on auttanut oikean resurssin valitsemisessa sekä säästänyt rahaa ja aikaa.

# LÄHTEET

Booch, G., Rumbaugh, J. & Jacobson, I. 1999. The unified modeling language user guide. Reading, Massachusetts: Addison-Wesley Longman, Inc.

Carrillo, J.M., Nicolás, J., Fernández J.L., Alemán, T. A., Ebert, C. & Vizcaíno, A. 2010. Requirements Engineering Tools. IEEE Computer society

Cheng B.H.C. & Atlee J.M. 2007. Research Directions in Requirements Engineering. International Conference on Software Engineering, Future of Software Engineering (FOSE´07) Minneapolis, Minnesota, May 20-26. Washington DC: IEEE

Collofello, J.S. & Gosalia, B.P. 1993. An Application of Causal Analysis to the Software Modification Process. Software-Practice and Experience. Vol 23(10)

Cooper A. 2004. Inmates Are Running the Asylum, The: Why High-Tech Products Drive Us Crazy and How to Restore the Sanity. Sams Publishing.

Fellows, L. & Hooks I. 1998. A Case for Priority Classifying Requirements.

Firesmith, D. 2004. Prioritizing Requirements. Software Engineering Institute. U.S.A, ETH

France R. & Rumpe B. 2007. Model-driven development of complex systems: A research roadmap. Future of Software Engineering 2007. IEEE-CS Press.

Haikala, I. & Märijärvi, J. 2002. Ohjelmistotuotanto. 8. painos. Talentum Media Oy.

Hofmann, H.F. & Lehner, F. 2001. Requirements engineering as a success factor in software projects. IEEE Software 18(4).

Hoffmann, M., Kuhn, N., Weber, M. & Bittner, M. 2004. Requirements for requirements management tools. Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04) Kyoto, Japan, Syyskuu 6–11. IEEE Computer Society.

Hull, E., Jackson, K. & Dick, J. 2011. Requirements Engineering. 3. painos. Springer.

IEEE Computer Society 1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12–1990.

Juristo, N., Moreno, A.M. & Silva, A. 2002. "Is the European Industry moving toward Solving Requirements Engineering Problems?". Spain: IEEE

Järvinen, P. & Järvinen, A. 2000. Tutkimustyön metodeista, Tampere: Opinpajan kirja.

Karlsson, J., Wohlin, C. & Regnell, B. 1997. An evaluation of methods for prioritizing software requirements. Focal Point AB, Teknikringen 1E, SE-583 30 Linköping, Sweden

Koskimies, K. & Mikkonen, T. 2005. Ohjelmisto-arkkitehtuurit. Helsinki: Talentum.

Maiden, N. & Robertson, S. 2005. Integrating creativity into requirements processes: Experiences with an air traffic management system. Proc. of the IEEE Int. Req. Eng. Conf. (RE)

Meredith, J.R. & Mantel, S.J. 2012. Project Management: A Managerial Approach. 8. pianos. Wiley.

Nuseibeh, B. & Easterbrook, S. 2000. Requirements Engineering: A Roadmap.

Ruhe, G. Eberlein, A. & Pfahl, D. 2002. Quantitative WinWin – A New Method for Decision Support in Requirements Negotiation. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering Ischia, Italy, July 15–19. New York: ACM.

Saiedian, H. & Dale, R. 2000. Requirements engineering: making the connection between the software developer and customer. Information and Software Technology.

Sharp, H., Finkelstein, A. & Galal, G. 1999. Stakeholder identification in the requirements engineering process. Proc. of the 10th Int. Work. on Datab. & Exp. Sys. Appl. s. 387-391.

Sommerville, I. 2004. Software Engineering. 7. painos. Harlow: Pearson Education

Sutcliffe, A., Fickas, S. & Sohlberg, M.M. 2006. PC-RE a method for personal and context requirements engineering with some experience. Req. Eng. J., 11(3):1-17.

Thompson, J.M., Heimdahl, M.P.E. & Miller, S.P. 1999. Specification-based prototyping for embedded systems. Proc. of ACM SIGSOFT Found. on Soft. Eng. (FSE), s.163-179.

Ul-Arif, S., Khan, Q. & Gahyyur, S. A. K. 2010. Requirements Engineering Processes, Tools/Technologies, & Methodologies, International Journal of Reviews in Computing.

Wiegers, K. E. 2003. Software Requirements. 2. painos. Microsoft Press

Zave, P. 1997. Classification of Research Efforts in Requirements Engineering. ACM Computing Surveys (CSUR)