

**OPINNÄYTETYÖ**

**TIINA MÄÄTTÄ 2010**

**PIRKKO RAUTIO 2010**

**TOIMINNALLINEN MÄÄRITTELY-  
DOKUMENTTI KETTERÄSTI**



**Rovaniemen  
ammattikorkeakoulu**  
University of Applied Sciences

**TIETOJENKÄSITTELYN KOULUTUSOHJELMA**

ROVANIEMEN AMMATTIKORKEAKOULU

LUONNONTIETEIDEN ALA

Tietojenkäsittelyn koulutusohjelma

Opinnäytetyö

## **TOIMINNALLINEN MÄÄRITTELYDOKUMENTTI KETTERÄSTI**

Tiina Määttä  
Pirkko Rautio

2010

Toimeksiantaja WAF-Solutions Oy

Ohjaaja Aarre Jortikka

Hyväksytty \_\_\_\_\_ 2010 \_\_\_\_\_

---

<b>Tekijä</b>	Tiina Määttä Pirkko Rautio	Vuosi	2010
<b>Toimeksiantaja</b>	WAF-Solutions Oy		
<b>Työn nimi</b>	Toiminnallinen määrittelydokumentti ketterästi		
<b>Sivu- ja liitemäärä</b>	62 + 5		

---

Opinnäytetyö käsittelee toiminnallisen määrittelydokumentin kirjoittamistapaa. Dokumentti kirjoitettiin ohjelmistotuotantoyritys WAF-Solutions Oy:lle, jolle ketterien menetelmien soveltaminen on muodostunut osaksi toimintatapaa. Opinnäytetyö tarkastelee dokumentin luomista ja toteuttamista.

Työn tavoitteena on vastata kysymykseen käytettiinkö tutkimuksen aineistona olevan toiminnallisen määrittelydokumentin kirjoittamisen yhteydessä Scott W. Amblerin määrittelemiä parhaita käytäntöjä ketterän dokumentin luomiseen sekä täyttyvätkö Amblerin kriteerit siitä, milloin dokumentti on ketterä. Tehtyä dokumenttia ja sen tekotapaa verrattiin edellä mainittuihin määritelmiin. Vertailun tuloksena saatiin hyvä vastaavuus määritelmiin.

Johtopäätöksenä voidaan sanoa, että dokumentti on ketterä ja että sen kirjoittamisessa noudatettiin parhaita ketterän dokumentin luomisen käytäntöjä.

Avainsana(t)

ohjelmistotuotanto, ketterät menetelmät, prosessimallit, toiminnallinen määrittely, dokumentointi

---

<b>Author</b>	Tiina Määttä Pirkko Rautio	<b>Year</b>	2010
<b>Commissioned by</b>	WAF-Solutions Ltd.		
<b>Subject of thesis</b>	Functional Specification Document Agility		
<b>Number of pages</b>	62 + 5		

---

The objective of this study was to examine the following: is the case Functional specification document an agile document. The document was commissioned by WAF-Solutions Ltd., a software engineering company. The Company used agile methods as part of their procedure. This research considered how the functional specification document was created and implemented.

The case was compared to Ambler's best practices for increasing the agility of documentation. The case was also analysed to see if the document met the criteria of an Ambler's agile document.

The results of the study showed that on the basis of the comparison and the analysis that the document was agile and also complied with best practice.

**Key words** software engineering, agile methods, process models, functional specification, documentation

## SISÄLLYS

KUVIOLUETTELO .....	1
1 JOHDANTO .....	2
2 OHJELMISTOTUOTANNON OSA-ALUEET .....	3
2.1 Ohjelmistotuotannon kehitysmenetelmät .....	3
2.2 Ohjelmiston prosessimallit .....	5
2.2.1 Vesiputousmalli .....	6
2.2.2 Prototyypimalli .....	9
2.2.3 Spiraalimalli .....	11
2.3 Dokumentointi ohjelmistotuotannossa .....	13
2.3.1 Toiminnallinen määrittely .....	15
2.3.2 Tekninen määrittely .....	18
3 KETTERÄ OHJELMISTOKEHITYS .....	19
3.1 Ketterästä ohjelmistokehityksestä yleisesti .....	19
3.2 Yleiset arvot ja periaatteet .....	20
3.3 Ketterän ohjelmistokehityksen menetelmät .....	22
3.3.1 Extreme Programming (XP) .....	23
3.3.2 Scrum .....	27
3.4 Dokumentointi ketterissä menetelmissä .....	30
3.4.1 Parhaita käytäntöjä ketterän dokumentin kirjoittamiseen .....	33
3.4.2 Amblerin ketterän asiakirjan kriteerit .....	38
4 TOIMINNALLISEN MÄÄRITTELYDOKUMENTIN LUOMINEN .....	40
4.1 Sisällönhallintajärjestelmä .....	40
4.2 Dokumentin työstäminen .....	41
4.3 Vertailua ja analysointia .....	51
5 YHTEENVETO .....	57
LÄHTEET .....	58
LIITTEET .....	62

## KUVIOLUETTELO

Kuvio 1. Ohjelmistotuotannon osa-alueet .....	3
Kuvio 2. Ohjelmistotuotannon kehittymisen ongelmanratkaisuja .....	4
Kuvio 3. Vesiputousmalli.....	6
Kuvio 4. Testauksen V-malli .....	9
Kuvio 5. Protoilumalli, jossa järjestelmä toteutetaan alusta alkaen.....	10
Kuvio 6. Prototyyppimalli, jossa prototyyppi kehitetään valmiiksi järjestelmäksi	11
Kuvio 7. Spiraalimalli .....	12
Kuvio 8. Toiminnallisen määrittelyn sisällysluettelo .....	16
Kuvio 9. XP:n elinkaaren prosessimalli.....	24
Kuvio 10. Scrum-malli.....	29
Kuvio 11. Mallien, dokumenttien, lähdekoodin ja dokumentoinnin suhteet. ....	33
Kuvio 12. Dokumentoinnin vaiheita ohjelmistokehityksen elinkaaren aikana.....	34
Kuvio 13. Määrittelydokumentin sisällysluettelo.....	42
Kuvio 14. Sisällysluettelo valmiista toiminnallisesta määrittelydokumentista .....	43
Kuvio 15. Järjestelmän sisäänkirjautumisikkuna.....	45
Kuvio 16. Järjestelmän sisäänkirjautumisvaihe .....	45
Kuvio 17. Käyttötapaus sisäänkirjautumisesta.....	46
Kuvio 18. Näkymä etusivulta.....	48
Kuvio 19. Editorinäkymä .....	49
Kuvio 20. Sivurakenteen toiminnot .....	50
Kuvio 21. Vertailun tuloksia .....	52

## 1 JOHDANTO

Tämä opinnäytetyö sai aiheensa siitä, kun suoritimme syventävää työharjoittelua ohjelmistotalo WAF-Solutions Oy:ssä. WAF-Solutions Oy on rovaniemeläinen ohjelmistotalo, jonka osaaminen painottuu Internetin hyödyntämiseen tuotteistettuihin palveluihin ja markkinointiin erityisesti matkailualalla. Internet-sivustot ja verkkokauppalpalvelut toteutetaan maailman suosituimpien ratkaisujen pohjalta yhdistettynä omaan tuotekehitykseen. Harjoittelun aikana kirjoitimme toiminnallisen määrittelydokumentin valmistumassa olevasta sisällönhallintajärjestelmästä.

Toiminnallisen määrittelydokumentin kirjoittaminen oli meille täysin uusi asia. Toimeksiantajamme ehdotti meille opinnäytetyön tekemistä aiheesta. Yritys on tehnyt alusta alkaen ohjelmistoja ketterän menetelmän mukaisesti – ei suunnitellusti, vaan menetelmä on muotoutunut luonnostaan. Tiivis ja pieni työyhteisö on antanut mahdollisuuden ja muokannut toimintamallia enemmän juuri ketterän kehityksen suuntaan. Vasta myöhemmin oman toimintansa esittelyn myötä yritys on ottanut käyttöön termin ketterä. Yritys ei noudata mitään tiettyä ketterien menetelmien mallia, vaan vuosien myötä muotoutuneessa omassa tuotantomallissa on eniten XP:n ja Scrum:n ketteriä piirteitä.

Opinnäytetyössä etsimme vastausta kysymykseen: noudattaako toiminnallinen määrittelydokumentti ja sen kirjoittaminen Amblerin (2009a) ketterän ohjelmistokehityksen dokumentoinnin parhaita käytäntöjä?

Aluksi opinnäytetyössä käsitellään ohjelmistotuotantoa ja perinteisiä prosessimalleja. Seuraavaksi perehdytään ketterään ohjelmistokehitykseen sekä esitellään enemmän kahta ketterää menetelmää, jonka jälkeen esitellään dokumentointia ketterissä menetelmissä.

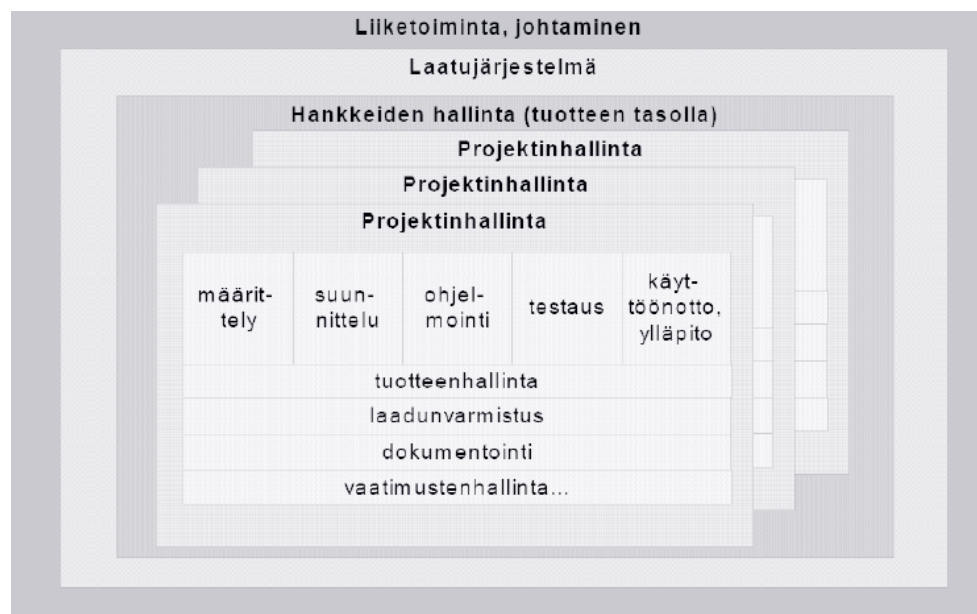
Lopuksi vertaamme tekemäämme toiminnallista määrittelydokumenttia Scott W. Amblerin (2009a) listaamiin ketterän dokumentin parhaisiin käytäntöihin. Analysoimme vastaako dokumentti noita käytäntöjä eli voidaanko sanoa, että kyseinen dokumentti on ketterä dokumentti.

## 2 OHJELMISTOTUOTANNON OSA-ALUEET

### 2.1 Ohjelmistotuotannon kehitysmenetelmät

Ohjelmistotuotannolla (Software engineering) tarkoitetaan ohjelmistotyötä, jonka tuloksena syntyvät tietojärjestelmät täyttävät käyttäjien toiveet aikatauluineen ja kustannuksineen. Software tarkoittaa kaikkea ohjelmistotyön tuloksena syntyvää materiaalia ja se suomennetaan yleensä termiksi ohjelmisto. Engineering tarkoittaa tekniikkaa ja sillä tarkoitetaan tieteellisen tiedon soveltamista käytännön ongelmiin. Software Engineering voidaan vapaasti suomentaa ohjelmistotuotannoksi tai ohjelmistokehitykseksi. (Haikala–Märijärvi 2004, 16.)

Ohjelmistotuotanto käsittää siis kaikki ohjelmistotuotantoprosessiin liittyvät osa-alueet (Haikala–Märijärvi 2004, 16). Toisinaan ohjelmistoprojektit voivat olla osa laajempaa tuotekehityshankkeen osaprojektia, jolloin kehitysprosessista voidaan erottaa ainakin määrittely-, suunnittelu-, ohjelmointi- ja testausvaiheet sekä käyttöönotto ja ylläpito. Projektiin liittyy koko projektin ja ohjelman elinkaaren ajan kestäviä tukitoimintoja, joista tärkeimpiä ovat laadunvarmistus, tuotteenhallinta ja dokumentointi (kuvio 1). (Haikala–Märijärvi 2004, 16, 35.)

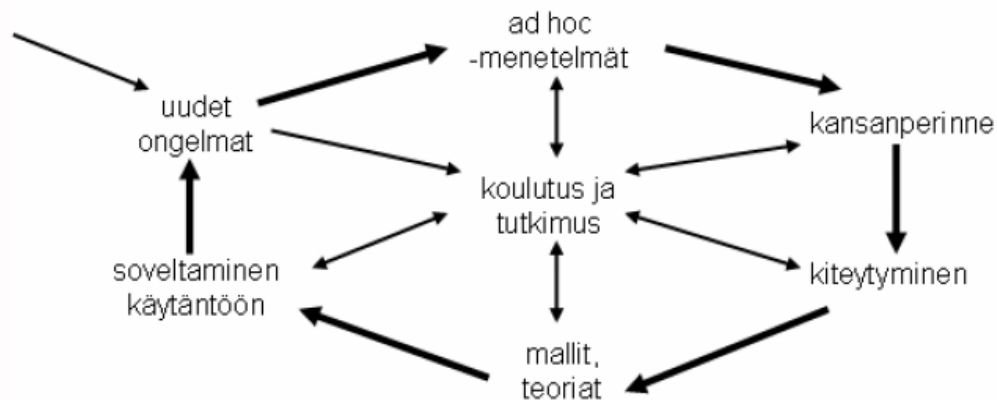


Kuvio 1. Ohjelmistotuotannon osa-alueet (Haikala–Märijärvi 2004, 35.)



Ohjelmistokehitys on insinööritieteenala, jossa käsitellään kaikkia ohjelmiston tuottamiseen liittyviä näkökulmia aina ohjelmiston suunnittelun alkuvaiheesta sen ylläpitoon asti. Kaikilla ohjelmistojen tuottamisen näkökulmilla tarkoitetaan teknisten ohjelmistokehitysmenetelmien lisäksi ohjelmistoprojektin hallintaan liittyviä työkaluja, menetelmiä ja teorioita, jotka tukevat ohjelmiston tuottamista. (Sommerville 2007, 7.)

Haikalan ja Märijärven (2004, 27–28) mukaan tekniikan kehittymisen esteitä ovat käytännön ongelmat, joihin haetaan ratkaisuja (kuvio 2). Ongelmaan haetaan ratkaisua millä tahansa toimivalla tavalla eli sovelletaan niin sanottuja ad hoc-menetelmiä. Löydetyistä ratkaisuista – jotka toimivat useammassa kuin yhdessä tapauksessa – syntyy kansanperinnettä. Kansanperinteenä välittyvä tietotaito taas kehittyy systemaattisemmaksi ja siitä syntyy tutkimus- ja työskentelymenetelmiä. Vähitellen nämä menetelmät kehittyvät riittävästi tukeakseen malleja ja teorioita, joita taas käytetään uusiin, entistä vaativampiin sovelluksiin. Ja kun sovelluksien valmiita ratkaisumalleja ei ole olemassakaan, niin on lähdettävä liikkeelle alusta ad hoc-menetelmistä.



Kuvio 2. Ohjelmistotuotannon kehittymisen ongelmanratkaisuja (Haikala–Märijärvi 2004, 27.)

Ohjelmistokehitysmenetelmät voidaan jakaa Huttusen (2006, 5–6) mukaan kahteen lähestymistapaan tuottamansa dokumentoinnin määrän perusteella: suunnitelmaohjautuviin kehitysmenetelmiin (plan-driven methods) ja ketteriin kehi-

tysmenetelmiin (agile methods). Suunnitelmaohjautuvissa kehitysmenetelmissä ohjelmiston kehitys alkaa tarpeiden määrittelystä ja päättyy ylläpitoon. Näissä menetelmissä tehtävät, merkkipaalat, määrittelyt ja arkkitehtuurisuunnitelmat sekä suunnitellaan että dokumentoidaan mahdollisimman tarkasti.

Vastapainoksi raskaille suunnitelmaohjautuville kehitysmenetelmille kehitettiin joukko kevyempiä menetelmiä, jotka perustuvat dokumentoinnin sijaan asiakasyhteistyön tuloksena syntyvään toimivaan ohjelmistoon. Tähän malliin kuuluvia menetelmiä kutsutaan ketteriksi ohjelmistokehitysmenetelmiksi (Huttunen 2006, 6), josta kerrotaan laajemmin luvussa kolme.

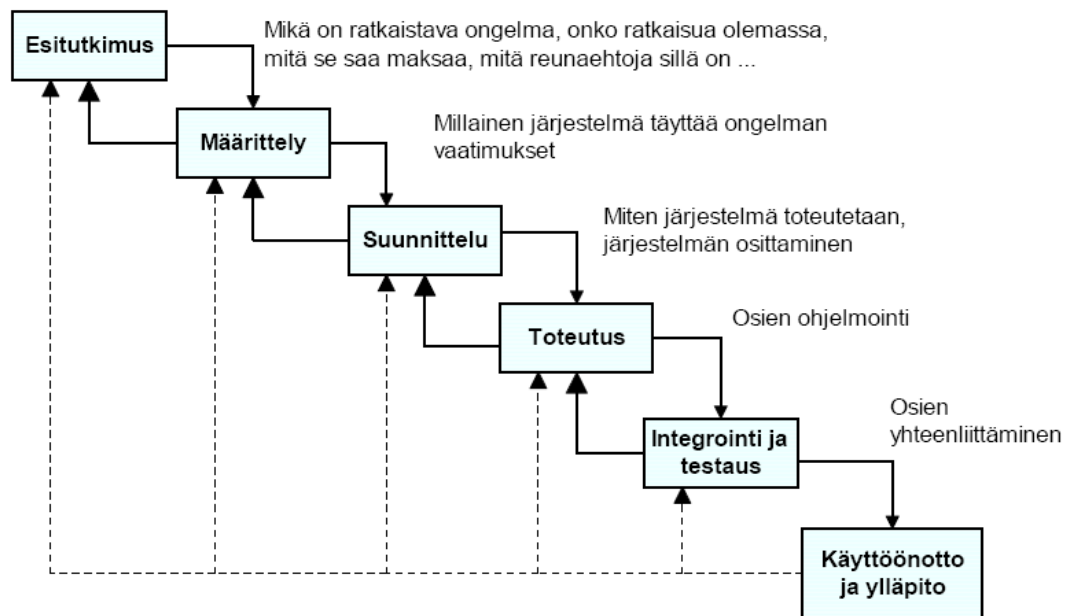
## 2.2 Ohjelmiston prosessimallit

Ohjelmiston elinkaari (life cycle) tarkoittaa aikaa, joka kuluu ohjelmiston kehittämisen aloittamisesta sen käytöstä poistamiseen. Elinkaarimalli on tapa, jolla ohjelmiston kehitystyö tai koko elinkaari jaetaan vaiheisiin. (Haikala–Märijärvi 2004, 36.) Oikean elinkaarimallin valinnalla voi vaikuttaa ratkaisevasti ohjelmistoprojektin kehitykseen, koska sillä voidaan sekä parantaa projektin kehittämisenopeutta, laatua ja hallintaa että minimoida turhaa työtä ja riskien ottamista. Väärällä elinkaarimallin valinnalla tai elinkaarimallin valitsematta jättämisellä voidaan aiheuttaa jatkuvaa tarpeetonta työtä, joka taas hidastaa projektia ja johtaa helposti turhautumisen tunteeseen. (McConnell 2002, 134.)

Tunnetuin ohjelmiston elinkaarimalli on vesiputousmalli, jossa suunnittelu- ja toteutusprosessi etenee vaihe vaiheelta alaspäin kuten vesiputouksessa. Vesiputousmallin ongelmana on suunnitella koko tuote kerralla toteutuskuntoon, koska käytännössä tuotantoprosessi on usein *iteratiivinen* eli suunnittelua ja toteutusta tehdään pienissä osissa ja prosessia toistetaan. Ohjelmistolle on tyypillistä *inkrementaalinen* kehitys eli ohjelmisto kasvaa koko ajan kohti lopullista muotoaan. Tästä tarpeesta ovat syntyneet useat iteratiiviset prosessimallit kuten spiraalimalli. (Wikipedia 2009a.) Spiraali- ja vesiputousmallin lisäksi seuraavissa alaluvuissa kerrotaan myös prototyypimallista, jotka ovat yhdessä kolme yleisintä käytössä olevaa elinkaarimallia.

### 2.2.1 Vesiputousmalli

Ensimmäinen ja tunnetuin elinkaarimalli on vesiputousmalli (waterfall model), joka kehitettiin 1960-luvun lopulla fyysisten prosessimallien pohjalta (Pohjonen 2002, 40). Mallista on olemassa erilaisia muunnelmia, mutta yleensä kaikista löytyvät ainakin määrittely-, suunnittelu- ja toteutusvaiheet. Määrittelyvaihetta edeltää usein esitutkimus- tai tarvekartoitusvaihe. (Haikala–Märijärvi 2004, 37.)



Kuvio 3. Vesiputousmalli (Tepsa 2006.)

Vesiputousmallissa ohjelmiston kehittämisen vaiheet (kuvio 3) seuraavat suora-  
viivaisesti toisiaan alkaen esitutkimuksesta ja päättyen ylläpitoon (Pohjonen 2002, 40). Ohjelmistoprojektissa pidetään katselmus jokaisen vaiheen lopussa ja jossa päätetään, onko projekti valmis jatkamaan seuraavaan vaiheeseen. Jos katselmuksessa todetaan, että projekti ei ole vielä valmis seuraavaan vaiheeseen, niin se pysyy paikoillaan niin kauan, kunnes on valmis. (McConnell 2002, 136.) Vesiputousmallissa tieto siirtyy vaiheiden välillä suurina kokonaisuuksina, yleensä dokumenttien muodossa (Nykopp–Koskela 2006, 6).

Vesiputousmalli kuvaa huonosti ohjelmiston iteratiivisuuden (Pohjonen 2002, 40). Vesiputousmallin perusongelmana pidetään sitä, että kussakin vaiheessa rakennetaan olettamuksia olettamusten päälle, josta syntyneet virheet ja väärinkäsitykset huomataan vasta testausvaiheessa. Tällöin projekti on jo viimeistelyvaiheessa, joten virheiden korjaaminen on hankalaa eikä tehtävään ole enää riittävästi aikaa. Vesiputousmallin ongelmien taustalla onkin tarpeiden ja vaatimusten epävarmuus, koska projektin tarpeista ei ole alussa riittävän tarkkaa tietoa. Monia yksityiskohtia selviää vasta projektin aikana ja valmiin järjestelmän käyttökokemusten perusteella tahdotaankin erilainen järjestelmä. Vaikka projektin aikataulu ja budjetti pitäisivätkin, niin laatu kärsii. Useimmiten toteutetaankin sellainen järjestelmä, joka vastaa vanhaa tavoitetilaa vaikka sen pitäisi tukea nykyisiä tarpeita. (Nykopp–Koskela 2006, 6.)

Vesiputousmalli toimii hyvin projekteissa, jotka toisaalta ovat hyvin ymmärrettyjä ja toisaalta taas monimutkaisia, koska monimutkaisuudesta selvitään säännönmukaisesti toimimalla. Projekti toimii hyvin, kun laatuvaatimukset ovat tärkeämpiä kuin kustannukset ja aikataulu. Vesiputousmalli toimii erityisen hyvin sellaisessa tilanteessa, jossa henkilökunta on kokematon eikä hallitse teknistä osaamista, koska se tarjoaa projektille rakenteen joka vähentää turhaa työtä. (McConnell 2002, 137.)

Vesiputousmallin *esitutkimusvaiheen* tehtävänä on asettaa yleiset vaatimukset toteutettavalle järjestelmälle. Esitutkimus vastaa kysymykseen, miksi järjestelmä tai ohjelmisto tehdään tai miksi sitä ei kannata tehdä. Esitutkimusvaiheen suurin ongelma on asiakkaan todellisten tarpeiden selvittäminen ja niiden tulkitseminen, sillä vääristä asiakasvaatimuksista ei rakenneta toimivaa ohjelmistoa. (Haila–Märijärvi 2004, 37.)

Vaatimukset määrittelevät eri sidosryhmien tarpeet järjestelmää kohtaan mutta eivät ota kantaa siihen, miten ne käytännössä toteutetaan (Pohjonen 2002, 28). Tätä vaihetta kutsutaan *määrittelyvaiheeksi* tai *vaatimusmäärittelyksi* (requirement analysis, requirement specification, system analysis). Määrittelyn tulokse-

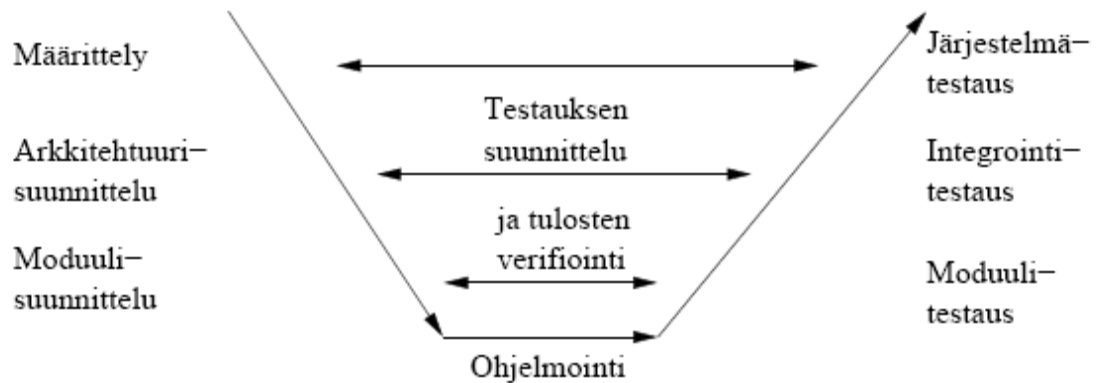
na syntynyttä dokumenttia kutsutaan toiminnalliseksi määrittelyksi. (Haikala–Märijärvi 2004, 38.)

Vaatimukset luokitellaan toiminnallisiin (functional) ja ei-toiminnallisiin vaatimuksiin (non-functional requirements). Toiminnalliset vaatimukset kertovat, mitä järjestelmän odotetaan tekevän ja ei-toiminnalliset vaatimukset taas sen, miten järjestelmä toteuttaa toiminnalliset vaatimukset, esimerkiksi vasteaika, kapasiteetti ja käytettävyys. (Pohjonen 2002, 28.)

*Suunnitteluvaiheessa* (design) suunnitellaan, miten järjestelmä toteutetaan ja sen tarkoituksena on muuntaa asiakkaan tarpeisiin mukautuva toiminnallinen määrittely tekniseksi määrittelyksi (technical specification), joka kuvaa järjestelmän toteutuksen (Pohjonen 2002, 32). Suunnitteluvaihe voidaan jakaa kahteen osaan: arkkitehtuurisuunnitteluun (architectural design) ja moduulisuunnitteluun (module design). Arkkitehtuurisuunnittelussa järjestelmä pilkotaan pieniin, toisista riippumattomiin osiin eli moduuleihin. Moduulisuunnitteluvaiheessa (module design, detailed design) suunnitellaan jokaisen moduulin sisäinen rakenne. (Haikala–Märijärvi 2004, 40.)

*Toteutusvaiheessa* (ohjelmointivaiheessa) kirjoitetaan varsinainen ohjelmakoodi (Haikala–Märijärvi 2004, 40). Lopuksi ohjelmamoduulit integroidaan eli kasataan toimivaksi kokonaisuudeksi (Pohjonen 2002, 34).

*Testausvaiheen* (testing) tarkoituksena on löytää ohjelmistosta virheitä. Ohjelmistojen testaus tapahtuu yleensä ns. V-mallin mukaisesti. V-mallissa testaus jaetaan kolmeen osaan: moduulitestaukseen, integrointitestaukseen ja järjestelmätestaukseen (kuvio 4). Moduulitestauksessa etsitään yksittäisten moduulien vikoja, integrointitestauksessa vikaa etsitään kaikkien moduulien yhteistoiminnoista sekä järjestelmätestauksessa koko järjestelmän toiminnoista ja suorituskyvystä. Järjestelmätestauksen suunnittelu tehdään alustavasti jo määrittelyvaiheessa ja testaus tehdään vertaamalla valmista järjestelmää sen määritellyyn. (Haikala–Märijärvi 2004, 40.)



Kuvio 4. Testauksen V-malli (Haikala–Märijärvi 2004, 289.)

*Ylläpitovaiheessa* (maintenance) korjataan ohjelman tekemiä virheitä, muutetaan ohjelma vastaamaan ympäristön muuttuneita vaatimuksia ja parannetaan ohjelmaa lisäämällä tai muuttamalla sen toiminnallisuutta (Haikala–Märijärvi 2004, 41).

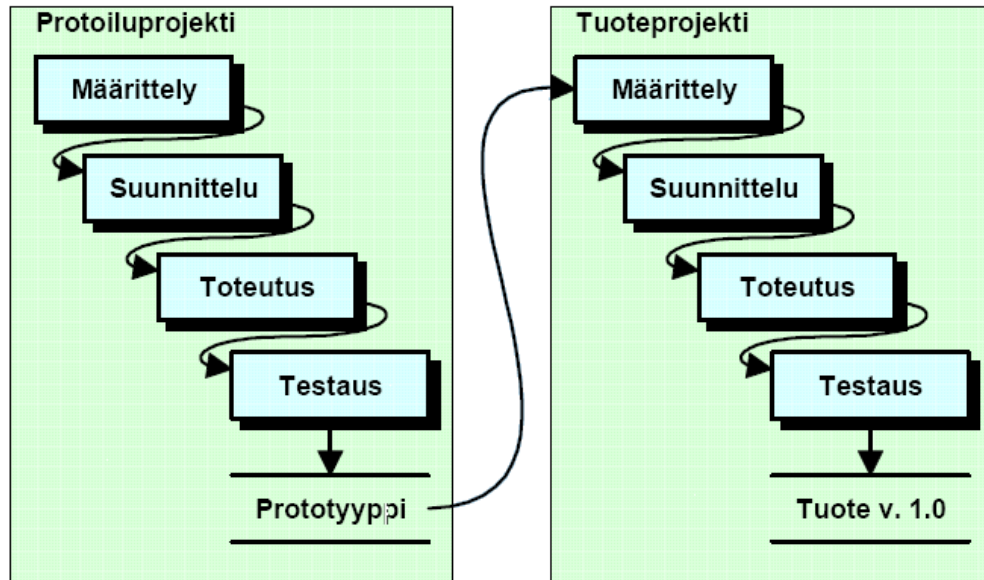
### 2.2.2 Prototyyppimalli

Prototyyppimallilla (protoilumallilla) voidaan tarkoittaa melkein mitä tahansa työskentelymallia, jossa jotain tuotteen piirrettä kokeillaan ennen varsinaisen tuotteen rakentamista. Prototyyppimallit soveltuvat erityisesti uuden teknisen ratkaisun vaatiman kokeilun tekemiseen tai etsimään epäselviä asiakasvaatimuksia. (Haikala–Märijärvi 2004, 42; Pohjonen 2002, 41.)

Haikalan ja Märijärven (2004, 42) mukaan valmis prototyyppi voidaan jakaa kahteen pääkäyttövaihtoehtoon:

- valmistuneen prototyypin perusteella määritellään toteutettava järjestelmä, joka toteutetaan alusta asti uudelleen
- prototyyppi kehitetään valmiiksi järjestelmäksi.

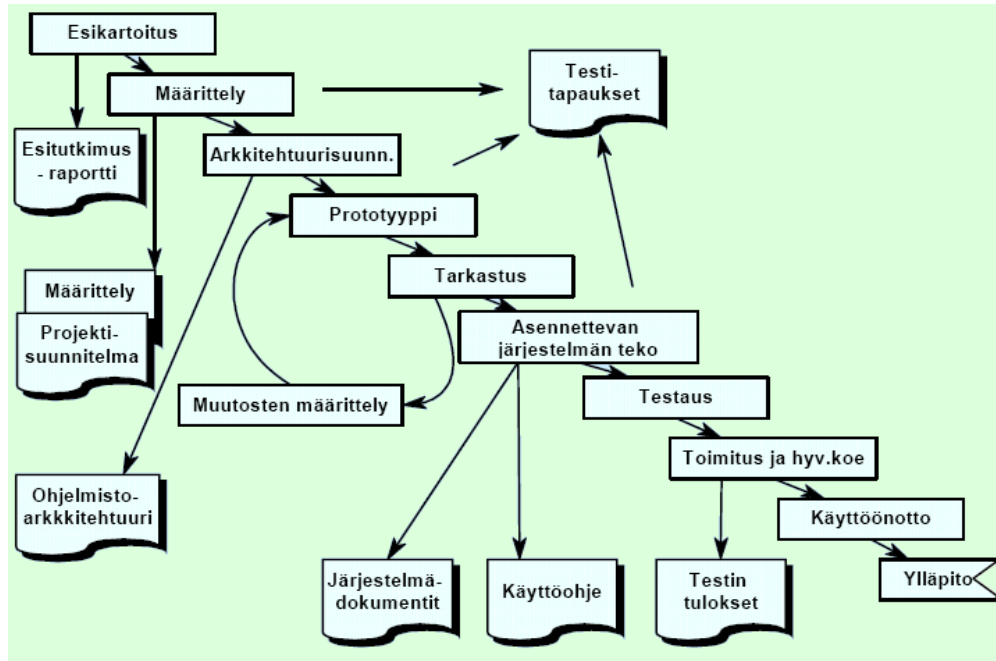
Prototyyppimallin esimerkki (kuvio 5) kuvaa rinnakkain sekä valmistunutta prototyyppiä että järjestelmän toteutusta eli ensimmäistä pääkäyttövaihtoehtoa. (Haikala–Märijärvi 2004, 42.)



Kuvio 5. Protoilumalli, jossa järjestelmä toteutetaan alusta alkaen (Haikala–Märijärvi 2004, 42.)

Kuviossa 6 oleva prototyyppi kehitetään valmiiksi järjestelmäksi. Prototyyppimalli sisältää jo kaikki tärkeimmät toiminnot lukuun ottamatta virhetarkastuksia, opastuksia ja tehokkaasti toimivaa tietokantaa. Ennen järjestelmän lopullista toteutusta varmistetaan järjestelmän sisältävän kaikki tarpeelliset toiminnot ja otetaan huomioon käyttäjäliittymässä, että se on asiakkaan toiveiden mukainen ja helppokäyttöinen. Prototyyppimalli on osoittautunut erityisen hyödylliseksi juuri käyttäjäliittymämäärittelyn yhteydessä. (Haikala–Märijärvi 2004, 42–43.)

Prototyyppimallin ongelma on se, että asiakas saattaa luulla järjestelmää lähes valmiiksi sen ulkoasun perusteella, vaikka valtaosa työstä olisi vielä tekemättä. Protoilumallia ei siis kannata tehdä viimeistellyn näköiseksi, jotta asiakas huomaa sen olevan keskeneräinen. (Haikala–Märijärvi 2004, 43.)



Kuvio 6. Prototyypimalli, jossa prototyyppi kehitetään valmiiksi järjestelmäksi (Haikala-Märijärvi 2004, 43.)

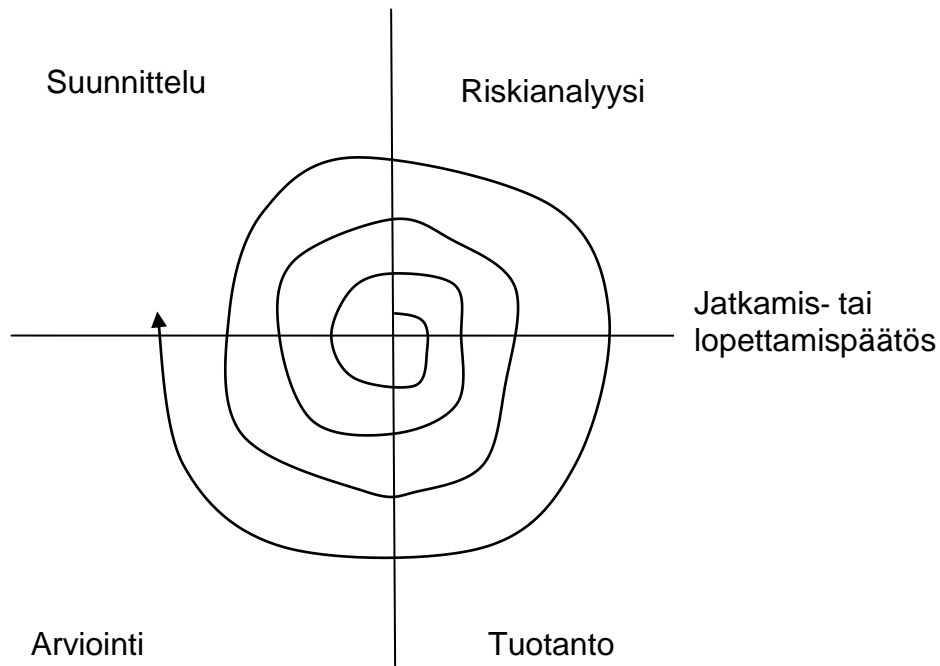
### 2.2.3 Spiraalimalli

Spiraalimalli on riskeihin suuntautunut elinkaarimalli, joka pilkkoo ohjelmistoprojektin pienemmiksi osaprojekteiksi. Jokainen pienempi osaprojekti keskittyy yhteen tai useampaan riskiin, kunnes kaikki pääriskit on hoidettu. Riskeillä tarkoitetaan tässä yhteydessä muun muassa huonosti ymmärrettyjä ohjelmiston vaatimuksia ja arkkitehtuuria sekä ohjelmistoon liittyviä suorituskykyongelmia. (McConnell 2002, 141.) Spiraalimallille on siis tyypillistä iteratiivisuuden lisäksi riskien jatkuva analysoiminen sekä prosessin uudelleen ohjaaminen riskianalyysin tuotosten mukaan ja tämä onkin spiraalimallin suurin ero verrattuna muihin prosessimalleihin (Pohjonen 2002, 42; Sommerville 2007, 73).

Spiraalimalli jakaantuu neljään eri vaiheeseen, joita toistetaan jatkuvasti tarkentaen niin kauan, kunnes järjestelmä on valmis (kuvio 7). Ensimmäinen vaihe on suunnittelu eli määrittellään rakennettavalle ohjelmistolle tavoitteet, vaihtoehdot ja rajoitukset. Toisessa vaiheessa tehdään riskianalyysi, jossa arvioidaan eri vaihtoehtoihin liittyviä ongelmia. Tämän jälkeen ohjelmisto menee tuotantoon, josta syntyy uusi ohjelmaversio. Lopuksi asiakas suorittaa arvioinnin, joka toimii



tarkastuspisteenä ennen seuraavaa iteraatiota ja jonka perusteella tehdään ohjelmiston jatkamis- tai lopettamispäätös. (Pohjonen 2002, 42.)



Kuvio 7. Spiraalimalli (Pohjonen 2002, 42.)

Spiraalimalli ei edellytä jotain tiettyä menetelmää tuotantovaiheessa, vaan sallii esimerkiksi vesiputousmallin tai prototyyppilähestymistavan soveltamisen. Mitä eteenpäin spiraalin ulompia kehiä mennään, sitä tarkemmaksi järjestelmän yksityiskohdat tarkentuvat. Prosessi on mahdollista keskeyttää, jos jossain vaiheessa havaitaan riskien kasvaneen liian suuriksi, mutta tavoitteena on kuitenkin pienentää analyysin avulla riskejä joka iteraatiolla. (Pohjonen 2002, 43.)

Spiraalimalli on elinkaarimalli, josta on vähän käytännön kokemuksia. Mallin ongelmina pidetään sitä, että asiakkaat olisi saatava aktiivisemmin mukaan prosessiin, mallin soveltaminen vaatii riskianalyysin hallitsemista sekä iteratiivinen prosessi on aikaavievä. (Pohjonen 2002, 42.)

### 2.3 Dokumentointi ohjelmistotuotannossa

Dokumentti on yleiskielessä synonyymi käsitteille asiakirja, todistuskappale. Perinteisesti dokumentilla tarkoitetaan kirjoitettuun tai tulostettuun paperiin tai artefaktiin, joka tuo esiin todistusvoimaista informaatiota, kuten passi, ajokortti, sopimus, lasku ja niin edelleen. Laajemmin käsitettynä dokumentti sanalla voidaan viitata mihin tahansa kirjalliseen tuotokseen; kirjaan, artikkeliin, kirjeeseen tai mihin tahansa, jolla on informatiivista arvoa. Dokumentti on tallennettu jollakin tallennusvälineellä ja sitä voidaan havainnoida ja käsitellä yhtenä kokonaisuutena. (Wikipedia 2009b.)

Dokumentointi on osa ohjelmistotuotantoa. Valitettavasti se hyvin usein jää tekemättä. Usein ohjelmiston suunnittelussa on kiireinen aikataulu. Dokumentointia ei pidetä niin tärkeänä, vaan aika käytetään mieluummin tuotteen koodaamiseen. Tosiasia on kuitenkin, että hyvin ja tarkasti tehty dokumentointi helpottaa tulevaisuudessa ohjelmiston päivittämistä uudempaan versioon. Toiminnallinen dokumentointi luo hyvän pohjan myös käyttöohjeiden tekemiselle.

Tyypillistä ohjelmistotyölle on, että projektin aikana kertynyttä tietoa kirjataan dokumentin muotoon. Ohjelmistotuotantoon kuuluvan dokumentoinnin keskeisinä tuotedokumentteina ovat toiminnallinen määrittely eli määrittelydokumentti ja tekninen määrittely eli suunnitteludokumentti. Myös testaussuunnitelma olisi hyvä olla olemassa. Minimidokumentoinnin pitäisi sisältää ainakin edellä mainitut dokumentit. Lisäksi projektisuunnitelma kuuluu minimidokumentointiin. (Haikala-Märijärvi 2004, 51.)

Testaamista käytetään virheiden eliminointikeinona. Ohjelmistojen testaus on suunnitelmallista virheiden etsintää, mikä tapahtuu ohjelmaa tai sen osaa suorittamalla. Testaaminen on siis suunnitelmallista ja etsimistä. Testaus voi tapahtua umpimähkäisesti kokeilemalla tai se voidaan suorittaa järjestelmällisesti. Järjestelmällisesti voi käydä koko ohjelman läpi toiminto toiminnolta. Testaaminen tapahtuu siis suorittamalla ohjelmaa. Testauksen avulla on mahdollista osoittaa ohjelmassa olevia virheitä. Ohjelman virheettömyyttä ei testauksella ole mahdol-

lista osoittaa. Ohjelman toimivuuteen ei kannata liiaksi luottaa, vaikka testitulokset olisivatkin hyvät. Testauksen yhteydessä virhe on poikkeama spesifikaatiosta. Eli testaus ilman spesifikaatiota on mahdotonta, koska silloin lopputuloksen oikeellisuutta ei voi todeta. Toiminnallinen määrittely ja tekninen määrittely ovat tavallisimmin testauksessa käytettävät spesifikaatiot. (Haikala–Märijärvi 2004, 283–287.)

Kun projekti päättyy, muutetaan tuotedokumentointi ylläpitoa palvelevaan muotoon ylläpidodokumentoinniksi. Tekniseen dokumenttiin kootaan kaikki tieto, jota ylläpidossa tarvitaan: toiminnallinen määrittely, tekninen määrittely sekä testaukseen ja tuotteen hallintaan liittyvät ohjeistukset. Kun ohjelmistoa korjataan ja muutetaan ylläpidon ongelmana voi olla se, että dokumentoinnin päivittäminen voi jäädä tekemättä. Tästä syystä dokumentointi kannattaa tehdä mahdollisimman helposti ylläpidettävään muotoon. Dokumentointia varten kannattaa luoda dokumenttimallit. Dokumenttien ulkoasu kannattaa tehdä yhdenmukaiseksi. (Haikala–Märijärvi 2004, 51, 75–76.)

Dokumentit kannattaa tehdä huolellisesti, sillä muutoin ne muuttuvat aikaa myöten täysin hyödyttömiksi. Jos ohjelmistoon tehdään muutoksia jotka kirjataan vain pintapuolisesti, ei dokumenteista pitemmän päälle ole hyötyä. Projektin aikana tulisi olla säännöllisesti tarkastuspisteitä. Tällaisella tarkastusmenettelyllä voidaan varmistaa, että dokumentit syntyvät. Samalla myös varmistetaan, että dokumenttien laatu on riittävä. Projektin koko ja monimutkaisuus määrittää dokumenttien määrän. Perusdokumentointi on kuitenkin oltava pienessäkin projektissa, vaikka dokumentointitarve olisi vähäinen. Määrittely ja suunnittelu -dokumenttien puuttumisesta voi olla seurauksena että, jossain vaiheessa koko tuote joudutaan suorittamaan uudelleenkirjoitus. Keskeisimpiä tuotedokumentteja ohjelmistoprojektissa ovat siis toiminnallinen määrittely, tekninen määrittely ja testausdokumentit. (Haikala–Märijärvi 2004, 70–72.)

Toiminnallinen määrittelydokumentti syntyy vaatimusmäärittelyn pohjalta. Vaatimusmäärittelyn synonyymeja ovat termit analyysi ja määrittely. (Haikala–Märijärvi 2002, 78.) Vaatimusmäärittelyn tehtävänä on projektin tarpeellisuuden

ja toteuttamiskelpoisuuden selvittäminen, vaatimusten ja tavoitteiden asettaminen sekä ratkaisumallin laatiminen. Vaatimusmäärittely siis on dokumentti, joka kertoo eri sidosryhmien tarpeet kehitettävälle järjestelmälle. (Pohjonen 2002, 28.)

Vaatimusmäärittelydokumentista tulisi löytyä ainakin seuraavat asiat (Pohjonen 2002, 30–31):

- kuvaus hankkeen toimeksiannosta
- yleiskuvaus organisaation nykytilanteesta
- kuvaus kohdejärjestelmästä ja sille asetetuista tavoitteista
- jokaisen toiminnallisen ja ei-toiminnallisen vaatimuksen kuvaus
- jokaisen rajoitteen kuvaus
- vaatimukset ja rajoitteet numeroituna ja tärkeysjärjestyksessä
- mahdolliset lisäselvitykset.

### **2.3.1 Toiminnallinen määrittely**

Toiminnallisessa määrittelyssä (functional specification) kuvataan ohjelmiston toiminnot (operations, functions) sekä toteutukselle asetettavat ei-toiminnalliset vaatimukset ja rajoitukset. Ei-toiminnallisia vaatimuksia ovat esimerkiksi käytettävyys, vasteaika ja suoritusteho. Rajoituksia ovat muun muassa toteutus tietyllä ohjelmointikielellä sekä käytettävissä oleva muistitila. Ohjelmistolla toteutettavat ominaisuudet, käyttöliittymä ja kommunikointi muiden järjestelmien kanssa määritellään toimintojen yhteydessä. Toiminnallinen määrittely sisältää suunnitteluvaiheen vaatimukset. (Haikala–Märijärvi 2004, 79–81.)

<b>1. JOHDANTO</b>	<b>5. ULKOISET LIITTYMÄT</b>
1.1 TARKOITUS JA KATTAVUUS	5.1 LAITTEISTOLIITTYMÄT
1.2 TUOTE	5.2 OHJELMISTOLIITTYMÄT
1.3 MÄÄRITELMÄT, TERMIT JA LYHENTEET	5.3 TIETOLIKENNELIITTYMÄT
1.4 VIITTEET	<b>6. MUUT OMINAISUUDET</b>
1.5 YLEISKATSAUS DOKUMENTTIIN	6.1 SUORITUSKYKY JA VASTEAJAT
<b>2. YLEISKUVAUS</b>	6.2 KÄYTETTÄVYYS, TOIPUMINEN, TURVALLISUUS, SUOJAUKSET
2.1 YMPÄRISTÖ	6.3 YLLÄPIDETTÄVYYS
2.2 TOIMINTA	6.4 SIIRRETTÄVYYS JA YHTEENSOPIVUUS
2.3 KÄYTTÄJÄT	6.5 KÄYTTÄJÄN YLLÄPITOTOIMET
2.4 YLEISET RAJOITTEET	<b>7. SUUNNITTELURAJOTTEET</b>
2.5 OLETUKSET JA RIIPPUVUUDET	7.1 STANDARDIT JA SUOSITUKSET
<b>3. TIEDOT JA TIETOKANNAT</b>	7.2 LAITTEISTORAJOTTEET
3.1 TIETOSISÄLTÖ	7.3 OHJELMISTORAJOTTEET
3.1.1 <i>Käsite X (kukin omana alakohtanansa)</i>	7.4 MUUT RAJOITTEET
3.2 KÄYTTÖINTENSITEETTI	<b>8. HYLÄTYT RATKAISUVAIHTOEHDOT</b>
3.3 KAPASITEETTIVAATIMUKSET	<b>9. JATKOKEHITYSAJATUKSIA</b>
3.4 TIEDOSTOT JA ASETUSTIEDOSTOT	<b>10. VIELÄ AVOIMET ASIAT</b>
<b>4. TOIMINNOT</b>	
4.1 YLEISTÄ (TAI JOKU MUU SOPIVA OTSIKKO)	
4.2 JÄRJESTELMÄN TOIMINNOT	

Kuvio 8. Toiminnallisen määrittelyn sisällysluettelo (Haikala 2009a.)

Toiminnallisen määrittelydokumentin johdanto-osassa kerrotaan järjestelmän tavoitteiden lisäksi, miksi tuote on tehty ja kenelle se on tarkoitettu (kuvio 8). Johdannossa voi tarvittaessa esitellä määritelmiä, termejä ja lyhenteitä. Kerrotaan myös, mitä muita dokumentteja on asiasta olemassa. Johdanto-osassa kuvataan lisäksi dokumentin rakenne. Dokumentin rakenteen kuvauksen tarkoituksena on antaa lukijalle riittävästi tietoa, mitkä kohdat hänen kannattaa lukea. (Haikala–Märijärvi 2004, 80.)

Dokumentin toisessa luvussa annetaan yleiskuva järjestelmän toiminnasta. Kuvataan järjestelmän liittymät ympäristöön ja myös ympäristöä kuvataan riittävällä tarkkuudella. Kuvataan myös järjestelmän käyttäjiä ja käyttöympäristöä sekä yleisiä rajoitteita. Esitetään oletukset, joiden voimassa ollessa määrittely on voi-

massa. Oletukset voivat koskea esimerkiksi käytettävissä olevan laitteiston tehoa tai muiden osaprojektien tuottamia tuotteita. (Haikala–Märijärvi 2004, 80.)

Dokumentin kolmannessa luvussa kuvataan järjestelmän käsittelemien tietojen ja tietokannan tietosisältö, tiedon pysyvyysvaatimukset, kapasiteetti- ja saantiaikavaatimukset ja niin edelleen. (Haikala–Märijärvi 2004, 80.) Neljännessä luvussa määritellään järjestelmän toiminnot. Alakohdissa jokainen järjestelmän toiminto täsmennetään. Jokaisesta toiminnosta kuvataan sen tarkoitus, mitä syötteitä toiminto tarvitsee, miten käsittely tapahtuu ja mitä vaikutuksia toiminnolla on. Eli mitkä ovat siis toiminnan tulosteet. (Haikala–Märijärvi 2004, 80–81.)

Viidennessä luvussa täsmennetään toisessa kohdassa yleisesti kuvatut liittymät järjestelmän ympäristöstä. Tässä vaiheessa voidaan jo tarkasti kuvata käyttöliittymä. Se voidaan myös esittää vain yleisellä tasolla, jolloin tarkentaminen jää suunnitteluvaiheen tehtäväksi. Muita tyypillisiä osia ovat liittymät oheislaitteisiin ja tietoliikenneyhteyksiin. Kuudennessa luvussa kuvataan järjestelmän eitoiminnalliset osat, joita ovat suorituskky, vasteajat, ylläpidettävyys ja käytettävyys. Seitsemännessä luvussa kuvataan rajoitteet. Rajoitteita ovat standardit, ohjelmistorajoitteet ja laiterajoitteet. (Haikala–Märijärvi 2004, 81.)

Toiminnallinen määrittelydokumentti kuvitetaan eri notaatioilla tehdyillä kuvauksilla. Esimerkiksi yleiskuvausta toisessa luvussa havainnollistamaan voidaan käyttää käyttötapauksia, liittymäkaaviota, ylimmän tason tietovirtakaaviota ja luokkakaaviota sekä erilaisia käyttöliittymäkuvauksia. Dokumentin kolmannessa luvussa käytettäviä kuvaustekniikoita ovat tietohakemistokuvaukset ja luokkakaaviot. Toimintojen määrittelyssä apuna voidaan käyttää esimerkiksi tietovuokaavioita, tila-automaatteja ja toiminnanmäärittelytekniikoita. (Haikala–Märijärvi 2004, 81.)

Toiminnalliseen määrittelyyn kuuluu olennaisena osana käyttötapaukset. Käyttötapauksista on muodostunut keskeinen osa projektikehitystä ja suunnittelua. Järjestelmän käyttötapauksilla tarkoitetaan vuorovaikutusta ohjelmiston kanssa. Käyttötapaukset edustavat ulkoista näkökulmaa järjestelmään. Käyttötapaus-

kaavio (use case diagram) on UML-kaavio, jolla voidaan havainnollistaa käyttötappauksia. Käyttötapausten käyttäminen ei kuitenkaan edellytä kaavioita, vaikka ne koetaankin hyödyllisiksi. Käyttötapaukset ovat muodostuneet korvaamattomiksi työkaluiksi niin vaatimusten kartoittamisessa kuin iteratiivisten projektien suunnittelussa ja hallinnassa. Fowler sanoo kuitenkin arvostavansa sitä vähemmän käyttötapauskaavioita mitä enemmän käyttötapauksista saa kokemusta. Fowlerin mielestä käyttötapauksissa kannattaakin laatia mieluummin tekstikuvauksia kuin piirtää kaavioita. (Fowler–Scott 2004, 35, 37, 42–43.)

### 2.3.2 Tekninen määrittely

Tekninen määrittely kuvaa järjestelmän toteutuksen. Teknisessä määrittelyssä tulisi Pohjosen (2002, 32) mukaan olla tiivistelmä järjestelmän tarkoituksesta, sekä kuvaukset seuraavista osa-alueista:

- järjestelmän sovellusalue ja järjestelmän osuus siinä
- järjestelmän laitteisto- ja ohjelmistoympäristö
- järjestelmän toteutuksen keskeiset reunaehdot
- järjestelmän ja sen ympäristön vuorovaikutus
- järjestelmän arkkitehtuuri
- yksittäiset moduulit ja alijärjestelmät
- toteutusrajoitteiden määrittelyt
- erityiset tekniset ratkaisut
- vaihtoehtoiset tai hylätyt ratkaisut sekä mahdolliset muut toteutukseen vaikuttavat seikat.

### 3 KETTERÄ OHJELMISTOKEHITYS

#### 3.1 Ketterästä ohjelmistokehityksestä yleisesti

Menetelmää, jossa suunnittelu perustuu dokumentoinnin sijasta ihmisten välisellä keskustelulla tapahtuvaan tiedonhankintaan, kutsutaan ketteräksi ohjelmistokehitysmenetelmäksi (Boehm 2002, 64–69). Ketterille menetelmille (agile methods, agile development) on yhteistä yksilöiden välinen vuorovaikutus, ohjelmiston toimivuus, suora viestintä ja nopea muutoksiin reagoiminen (Nykopp–Koskela 2006, 7.)

Ohjelmistoon tehdään koko ajan pieniä muutoksia ja julkaistaan uusia versioita nopeassa tempossa (Abrahamsson–Salo–Ronkainen–Warsta 2002, 98). Ketterässä kehityksessä ei pyritä siihen, että määriteltäisiin kaikki yksityiskohtat kerralla, vaan kehitystyö etenee pieninä sarjoina. Silloin jokainen iteraatio luo jotakin uutta tietoa tai uusia yksityiskohtia. (Eriksson–Penker 2000, 240.) Käytännössä ketterässä tuotantoprosessissa suunnittelua ja toteutusta tehdään pienimmissä osissa ja prosessia toistetaan. Näin ohjelmisto kehittyy koko ajan kasvavaan kohti lopullista muotoaan. Ketterä ohjelmistokehitys nojaa korkeaan tekniseen taitoon ja se pyrkii minimoimaan muutoksista aiheutuvat kustannukset. (Klemetti 2007.) Ketterät menetelmät ovat hyvin suoraviivaisia ja mukautuvia, sillä menetelmät ovat helposti opittavia, muokattavia ja hyvin ohjeistettuja sekä mahdollistavat viime hetken muutokset. (Abrahamsson ym. 2002, 98.)

Ketterää ohjelmointia perustellaan muun muassa sillä, että liiketoiminnassa niin yksityisellä kuin julkisellakin sektorilla vaaditaan yhä enemmän ketteryyttä, myös teknologian muutokset ovat nopeita. Koska työelämässä ollaan jatkuvassa muutoksessa, niin myös ICT-projektien pitää sopeutua nopeasti muuttuviin tilanteisiin. Tarvitaan uudenlaisia ohjelmistonkehitysprosesseja, jotka ovat nopeita, mutta silti laadukkaita. (Nykänen 2009.)

Käyttäjäkeskeisissä menetelmissä kaikki suunnittelu ja kehitys dokumentoidaan perusteellisesti, kun taas ketterät menetelmät eivät korosta etukäteissuunnittelua ja dokumentointi on kevyttä (Nykänen 2009). Projektin aikaisten dokumentti-



en ainoa tehtävä on vähentää epävarmuustekijöitä ja tehostaa kommunikaatiota. Muutoin ne ovat turhia. (Ketterät käytännöt 2009a.)

### 3.2 Yleiset arvot ja periaatteet

Vuonna 2001 USA:ssa kokoontui seitsemäntoista ketterän kehityksen puolesta-puhujaa ja kokoontumisen seurauksena otettiin käyttöön termi ketterä (agile), joka kuvaa eri ohjelmistomenetelmien yhteisiä piirteitä. Tässä samassa tapahtumassa julkaistiin ketterän kehityksen manifesti (Agile Alliance 2009a), joka kuvaa ketterän kehityksen yleiset arvot ja periaatteet.

Ketterän kehityksen neljä yleistä arvoa ovat (Agile Alliance 2009b):

- **yksilöt ja vuorovaikutus** *ovat tärkeämpiä kuin prosessit ja työkalut*
- **toimiva ohjelmisto** *on tärkeämpää kuin kattava dokumentointi*
- **asiakasyhteistyö** *on tärkeämpää kuin sopimusneuvottelut*
- **muutokseen reagoiminen** *on tärkeämpää kuin suunnitelman noudattaminen.*

Ketterissä menetelmissä arvostetaan vasemman puoleisia (**lihavoidut**) arvoja enemmän kuin oikeanpuoleisia (*kursivoidut*), mutta tämä ei kuitenkaan tarkoita etteivätkö prosessit, työkalut, dokumentoinnit, sopimukset ja suunnitelmat olisi tärkeitä. Oikeanpuoleiset arvot vaan rinnastetaan yleisen ohjelmistomenetelmien ajattelutapaan – kun taas ketterät menetelmät suosivat enemmän asiakkaan vaatimuksen mukaisena syntyviä lopputuloksia. (Agile Alliance 2009b.)

Manifesti (Agile Alliance 2009c) kuvaa ketteryyden 12 periaatetta seuraavasti:

1. Tärkeintä on täyttää asiakkaan tarpeet jatkuvilla ja riittävän aikaisilla ohjelmistotoimituksilla.

2. Vaatimusten muuttumiset ovat tervetulleita ja ne hyväksytään jopa myöhemmissä kehitysvaiheissa. Ketterät menetelmät valjastavat muutoksen asiakkaan kilpailueduksi.
3. Toimivia ohjelmaversioita toimitetaan säännöllisesti mielellään lyhyellä aikavälillä (muutamasta viikosta muutamaan kuukauteen).
4. Liiketoiminnan ammattilaisten ja kehittäjien täytyy työskennellä yhdessä päivittäin koko projektin ajan.
5. Projektit rakennetaan motivoituneiden ihmisten ympärille. Annetaan heille ympäristö ja tuetaan heidän tarpeitaan, sekä luotetaan siihen, että he saavat työnsä tehtyä.
6. Kaikkein tehokkain tapa välittää tietoa kehitystiimille ja kehitystiimin sisällä on kasvokkain tapahtuva keskustelu.
7. Toimiva ohjelmisto on ensisijainen työn edistymisen mittari.
8. Ketterät menetelmät suosivat kestäväää kehitystä. Rahoittajien, kehittäjien ja käyttäjien tulisi pitää jatkuvasti yllä tasaista työtahtia.
9. Jatkuva huomion kiinnittäminen tekniseen osaamiseen ja hyvään suunnitteluun lisää ketteryyttä.
10. Yksinkertaisuus – taito maksimoida tekemätön työ – on olennaista.
11. Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat nousevat itseorganisativista tiimeistä.
12. Tiimi arvioi tasaisin väliajoin, kuinka tulla entistä tehokkaammaksi ja muokkaa toimintaansa sen mukaisesti.

### 3.3 Ketterän ohjelmistokehityksen menetelmät

Tunnetuimpia ketteriä menetelmiä ovat Scrum ketterään projektinhallintaan ja Extreme Programming (XP), joka keskittyy enemmänkin ohjelmistoteknisiin toteutuskäytäntöihin (Abrahamsson–Salo 2007, 4; Ketterät käytännöt 2009b).

Kun käytetään raskaita ohjelmistokehitysmenetelmiä, on niissä suunnan muuttaminen hankalaa sen jälkeen, kun sopimus on allekirjoitettu. Kehitystiimin tehtävänä on toteuttaa sellainen tuote, kuin sopimukseen on kirjattu. Jos realiteetit muuttuvat tai käyttäjät haluavatkin erilaisen ohjelmiston, on muutosta hankala toteuttaa. Arkkitehtuurin uudelleen muokkaaminen ja alkuperäisen aikataulun venyminen ovat esimerkkejä hankaluuksista, joita suunnan muuttaminen kesken projektin voi aiheuttaa. Kerran aloitettu ohjelmistokehitys on kuin valtamerialus, jonka suuntaa ei noin vain muuteta. Näin siis tapahtuu perinteisissä kehitysprosesseissa. Ohjelmistokehitysprosesseissa on myös pantu merkille, että oikeiden vaatimusten määrittäminen ei onnistu heti, vaan ne kertyvät ajan myötä. (van Vliet 2008, 54.)

Ketterät menetelmät ottavat käyttäjät mukaan kaikissa vaiheissa. Kehityssyklit ovat pieniä ja vähittäisiä. Kehityssykliden sarjoja ei ole laajasti suunniteltu eteenpäin, mutta uudet tilanteet tarkistetaan jokaisen syklin lopussa. Ne sisältävät jotain suunnitelmia seuraavalle syklille, eivät kuitenkaan liikaa. Jokaisen syklin lopussa järjestelmän pitäisi toimia. Eli syklin lopussa asiakkaalle on tarjolla toimiva ohjelma, vaikka se ei olisikaan aivan sitä, mitä asiakas on alun perin halunnut. (van Vliet 2008, 55.)

Ketterissä menetelmissä ei käytetä laajaa arkkitehtuuria tai etukäteissuunnittelua. Ei pidetä järkevänä tuhlaa työtä suunnitteluun, jos se todennäköisesti on hukkaan heitettyä aikaa. Paljon tehokkaampaa on suunnitella vain se, mitä välttämättä tarvitaan seuraavaan vaiheeseen. Usein ketterissä menetelmissä on erillisiä toimintoja, esimerkiksi lähdekoodin parantelu, joilla parannetaan suunnittelua jokaisen vaiheen jälkeen. Ketterä menetelmä onkin enemmän ihmisorientoitunutta kuin prosessisuuntautunutta. Tiimihenki mielletään erittäin tärkeäksi.

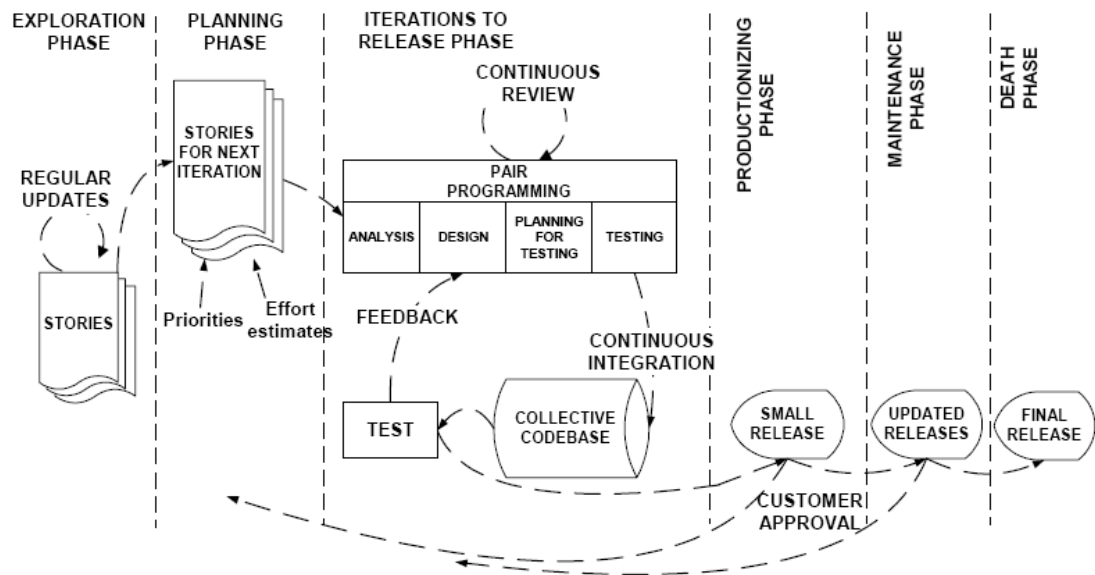
Tiimin suhteet ovat läheisiä ja usein tiimi työskentelee yhdessä isossa huoneessa. Ketterissä menetelmissä pidetään lyhyitä tiedonantoja suunnittelijoiden ja käyttäjien kesken kierrosten välissä, sekä suunnittelijat myös keskenään. (van Vliet 2008, 55.)

### 3.3.1 Extreme Programming (XP)

Tunnetuin ja käytetyin ketterien menetelmien tapa on Extreme Programming (XP). Se on myös ainoa ketterien menetelmien tapa, joka keskittyy pääasiassa ohjelmistokehityksen ohjelmointipuoleen. (Schuh 2004, 19.) XP poikkeaa useista ketteristä menetelmistä siten, että se tarjoaa joukon käytäntöjä, joita pitää noudattaa yhdessä ja joiden mukaan työskennellään. Vaikka XP:n pääpaino on ohjelmistoteknisessä toteutuksessa, siinä on kuitenkin jonkun verran myös mukana projektinhallintaan liittyviä käytäntöjä, kuten suunnittelupeli. (Ketterät käytännöt 2009c.)

Ensimmäinen Extreme Programming projekti aloitettiin vuonna 1996 ja sen isänä pidetään Kent Beckiä (Abrahamsson–Salo 2006, 13). XP:n ohjenuorana on parantaa ohjelmistoprojektia viidellä keskeisellä tavalla: *kommunikoimalla, yksinkertaisuudella, palautteella, kunnioittamalla ja rohkaisemalla*. XP-tiimi kommunikoi jatkuvasti asiakkaiden ja muiden ohjelmoijien kanssa. Ohjelmisto suunnitellaan selkeäksi ja yksinkertaiseksi, jotta palautetta voidaan saada jo ensimmäisestä päivästä lähtien testaamalla ohjelmistoa. Ohjelmisto toimitetaan asiakkaalle mahdollisimman nopeasti. Jos asiakas haluaa tehdä jotain muutoksia ohjelmistoon, niin ne myös toteutetaan. (Extreme Programming 2009.)

XP korostaa siis erittäin paljon toimivaa tiimityötä. Kaikki tiimiin kuuluvat henkilöt ovat tasa-arvoisia kumppaneita keskenään. Tekemällä yksinkertaista, tehokasta työtä tiimit voivat olla hyvinkin tuottavia ja he pystyvät ratkaisemaan ongelmansa mahdollisimman tehokkaasti. Ohjelmistoprojektissa jokainen pieni onnistuminen syventää kunnioitusta jokaista tiimin jäsentä kohtaan. Tämän onnistumisen ja kunnioituksen avulla XP:n ohjelmoijat pystyvät vastaamaan mahdollisiin muuttuneisiin vaatimuksiin ja teknologiaan. (Extreme Programming 2009.)



Kuvio 9. XP:n elinkaaren prosessimalli (Abrahamsson ym. 2002, 19.)

XP:n elinkaaren prosessimalli koostuu kuudesta eri vaiheesta (kuvio 9), jotka etenevät pääsääntöisesti vaiheesta toiseen, mutta tarvittaessa mahdollistavat paluun takaisin edellisiin vaiheisiin. Beck (1999a, 100–104) määrittelee prosessimallin vaiheet seuraavasti:

*Tutkimusvaiheessa (Exploration phase)* asiakkaat kirjoittavat tarinakorteille toivomuksiaan ensimmäiseltä julkaisulta. Jokainen tarinakortti kuvaa toteutettavaan ohjelmaan lisättävää ominaisuutta. Samaan aikaan projektin työntekijät tutustuvat projektissa käytettäviin työvälineisiin, teknologiaan ja käytäntöihin. Tutkimusvaihe kestää muutamista viikoista muutamiin kuukausiin. *Suunnitteluvaiheessa (Planning phase)* päätetään ensimmäisen julkaisun sisällöstä ja päivämäärästä. Suunnitteluvaihe kestää muutamia päiviä. (Beck 1999a, 100–101.)

*Julkaisun iteraativaiheessa (Iterations to release)* aikataulu pilkotaan pienempiin iteraatioihin, joista jokaisen toteuttamiseen kuluu aikaa yhdestä neljään viikkoa. Ensimmäinen iteraatio luo järjestelmän arkkitehtuurin. Ihannetapauksessa asiakas kirjoittaa valmiin tarinakortin jokaisen iteraation lopussa ja niiden perusteella suoritetaan toiminnalliset testit. Viimeisen iteraation jälkeen järjestelmä on

valmis tuotantoon. *Tuotteistamisvaiheessa (Productionizing phase)* järjestelmälle tehdään lisää testausta ja suoritetaan useita iteraatiovaiheita. (Beck 1999a, 102–103.)

*Loppujulkaisu- ja ylläpitovaiheeseen (Maintaince and Death phase)* siirrytään silloin, kun tuotteen ensimmäinen versio on toimitettu asiakkaalle. Käytössä huomattuja virheitä korjataan ja tehdään edelleen viimeistellympää ohjelmaversiota. Kun järjestelmä tyydyttää asiakkaan tarpeet eikä löydy enää uusia vaatimuksia järjestelmää kohtaan, on aika lopettaa tuotteen kehittäminen. Tuotteesta kirjoitetaan lopullinen dokumentti ja se todetaan valmiiksi. (Beck 1999a, 103–104.)

Ehkä yllättävintä XP:ssä on sen yksinkertaiset säännöt. XP on vähän kuin palapeli: siinä on paljon pieniä osia. Yksin ja erillään niissä ei ole mitään järkeä, mutta kun ne yhdistetään isoksi kokonaisuudeksi, niin kuva selkiytyy huomattavasti. Säännöt voivat vaikuttaa aluksi hankalilta tai jopa naiiveiltakin, mutta ne perustuvat kuitenkin järkeviin arvoihin ja periaatteisiin. (Extreme Programming 2009.)

Beckin (1999a, 47–53) mukaan XP perustuu ketterien menetelmien kaltaisiin käytäntöihin, jossa yhden käytännön heikkouden voi peittää tai korvata jonkun toisen käytännön vahvuuksilla. Nämä käytännöt jakaantuvat kolmeentoista eri ryhmään seuraavasti (Beck 1999b, 71):

- *Suunnittelupeli (Planning game)* on tiivistä vuorovaikutusta asiakkaan ja ohjelmoijien välillä. Ohjelmoijat tekevät arvion työtarpeista, joita tarvitaan käyttäjäkertomusten toteuttamiseksi ja lopulta asiakas päättää julkaisujen laajuuden ja ajankohdan.
- *Pienet julkaisut (Small releases)*  
Järjestelmä valmistetaan jopa muutamassa kuukaudessa, koska kaikkia ongelmia ei tarvitse ratkaista kerralla. Uusia versioita julkaistaan tiheästi, jopa päivittäin tai kuukausittain.

- *Metafora (Metaphor) eli vertauskuva* tarjoaa kokonaiskuvan kehitettävästä järjestelmästä ja sen toiminnasta. Vertauskuvan avulla monimutkainen asia selitetään niin, että kaikki osapuolet ymmärtävät mitä asia tarkoittaa.
- *Yksinkertainen rakenne (Simple design)*  
Tehdään kaikki tarpeelliset asiat kerralla, mahdollisimman yksinkertaisesti ja poistetaan kaikki tarpeeton.
- *Testit (Tests)*  
Tehdään sekä kehittäjien vaatimat yksikkötestit että asiakkaan määrittelemät toiminnalliset testit ennen varsinaisen toiminnon toteuttamista.
- *Refaktoroinnilla (Refactoring) eli rakenteen parantamisella* muokataan koodia paremmin ylläpidettävään, selkeään muotoon muuttamatta kuitenkaan sen toiminnallisuutta.
- *Pariohjelmoinnissa (Pair programming)* kaksi ohjelmoijaa työskentelee yhdellä koneella; toinen kirjoittaa ohjelmakoodia ja toinen taas seuraa vierestä valvoen työn laatua.
- *Jatkuva integrointi (Continuous integration)* on prosessi, jossa koko ohjelmistoa koostetaan ja integroidaan jatkuvasti. Järjestelmän on läpäistävä kaikki testit ennen kuin muutokset voidaan hyväksyä.
- *Koodin yhteisomistus (Collective ownership)* mahdollistaa jokaisen ohjelmoijan muokkaamaan koodia milloin vain.
- *Asiakkaan läsnäolo (One-site customer)* on kokoaikaista ja hän tekee tiivistä yhteistyötä tiimin kanssa läpi projektin.
- *40 tuntiset työviikot (40-hour weeks)*  
Jatkuvaa ylityötä tulisi välttää, koska väsymys heikentää suoritusta. Ohjelmointi on ajatustyötä, jossa parhaimmat oivallukset syntyvät virkeiltä ja levänneiltä aivoilta.

- *Avoin työtila (Open workspace)*

Tiimi työskentelee avokonttorimallisessa tilassa, jossa pariohjelmoijat ovat huoneen keskellä ja muut tiimin jäsenet ovat huoneen laidoilla.

- *Säännöt (Just rules)*

XP- tiimi noudattaa tiettyjä sääntöjä, jotka se voi muuttaa haluamaksensa milloin tahansa. Muutokset tulee kuitenkin hyväksyä ja niiden vaikutuksia on arvioitava.

XP on osoittautunut erittäin onnistuneeksi ohjelmointitavaksi erikokoisissa yrityksissä ympäri maailman. XP on onnistunut, koska se korostaa asiakastytyväisyyttä tuottamalla asiakkaan vaatimusten mukaisen ohjelmiston juuri nyt eikä joskus tulevaisuudessa. Tosin XP vastaa muuttuneisiin asiakasvaatimuksiin vielä myöhemmässäkin elinkaaren vaiheessa. (Extreme Programming 2009.)

### 3.3.2 Scrum

Ketterä prosessimenetelmä Scrum on iteratiivinen ja inkrementaalinen suunnittelu- ja tuotantoprosessimenetelmä, joka tuottaa iteraatioiden päätteeksi valmiin osakokonaisuuden tuotteesta. Menetelmän avulla on helppo seurata ohjelmistotuotantoprojektin edistymistä. Oikean tiedon avulla kyetään tekemään projektia koskevia päätöksiä. Toimintaa pystytään kehittämään jatkuvasti ja projektia hidastaviin seikkoihin voidaan vaikuttaa. Toimintaympäristössä tapahtuviin muutoksiin ja uusiin vaatimuksiin pystytään reagoimaan nopeasti ja hallitusti. Scrum on prosessi, joka kontrolloi ristiriitaisten intressien ja tarpeiden kaaosta, parantaa kommunikaatiota ja maksimoi yhteistyötä sekä tuottavuutta. Menetelmää käytetään niin yksittäisissä projekteissa kuin kokonaisissa organisaatioissa, esimerkiksi Microsoftilla ja Googlella. (Lindström 2006; Controlchaos 2009.)

Scrumissa työskentely on kurinalaista. Kehitystyö tapahtuu yhdestä neljään viikon kestoisissa iteraatioissa, joissa toteutetaan liiketoiminnan kannalta tärkeimmät ohjelmiston ominaisuudet. Scrumissa iteraatiota kutsutaan sprintiksi. Sprintti sisältää määrittelyä, toteutusta, testausta ja toimittamista (Nevalainen 2008, 20). Projektin sidosryhmille esitellään jokaisen sprintin päätteeksi valmis ja toimiva



kokonaisuus. Scrum tiimiin kuuluu kolme roolia. Nämä roolit ovat tuotteen omistaja (product owner), Scrum-mestari (ScrumMaster) ja tiimi. Projektia hallinnoi tuotteen omistaja, jonka tehtävänä on kerätä ja tarkentaa projektin toimitusta koskevat vaatimukset sekä yhdessä tiimin kanssa luoda aikataulu ominaisuuksien toteuttamiselle. Tuotteen omistaja tekee päätökset tuotteen ominaisuuksista sekä toiminnallisuuteen vaikuttavista seikoista. (Lindström 2006; Ketterät käytännöt 2009d.)

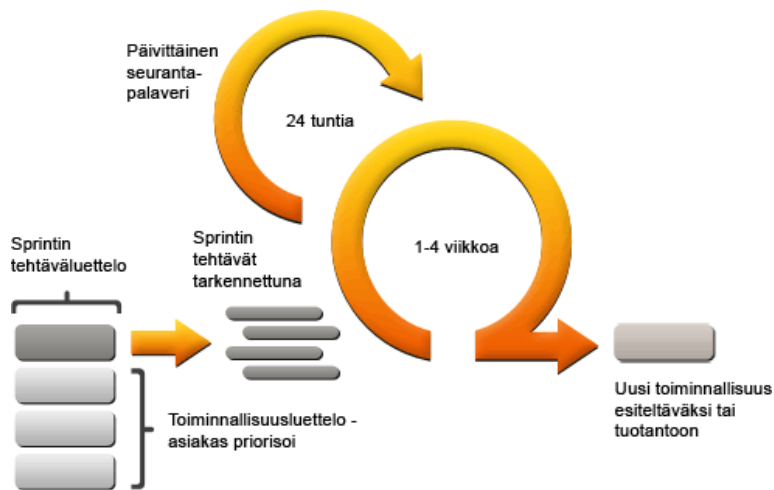
Scrum-mestari huolehtii siitä, että tiimi voi työskennellä optimaalisella tavalla. Tiimin jäsenet raportoivat töitä hidastavista ongelmista mestarille, jonka tehtävänä on ratkaista ja poistaa nämä esteet. Mestarin tehtävänä on myös johtaa aamupalaverit, joita pidetään päivittäin. Scrum-mestari vastaa myös siitä, että Scrumia noudatetaan oikein. (Lindström 2006; Ketterät käytännöt 2009d.)

Tiimi muodostuu kaikista, jotka ovat projektia tekemässä. Tiimin sisältä ei erikseen nimetä arkkitehteja, ohjelmoija, käyttöliittymäsuunnittelijoita eikä testaajia. Scrum-tiimi koostuu henkilöistä, joilla on tarvittava osaaminen ja jotka osaavat yhdessä rakentaa tuotteen. Edellä mainitulla käytännöllä halutaan korostaa sitä, että kaikki tiimin jäsenet ovat projektin kannalta yhtä tärkeitä. Koko tiimi vastaa tuotteen kaikista puolista, ei koskaan yksittäinen henkilö. (Ketterät käytännöt 2009d.)

Scrum pohjautuu sitoutumiseen, keskittymiseen, avoimuuteen, kunnioitukseen ja rohkeuteen. Tiimin tehtävä on sitoutua yhteiseen päämäärään. Jotta tavoitteeseen voi sitoutua, on tavoitteen oltava niin selkeä, että iteraation jälkeen voidaan todeta, onko tavoite saavutettu vai ei. Scrum-tiimi on oikeutettu saamaan kaiken tuen, jotta sitoumukset saavutetaan. Tiimi sitoutuu toimittamaan seuraavan sprintin aikana tietyt toiminnallisuudet ja sillä on vapaus toimia parhaaksi katsomallaan tavalla. Scrum-mestari ja muu organisaatio sitoutuvat tiimin tukemiseen poistamalla tiimiä häiritsevät tekijät. Scrum-tiimin jäsenet keskittyvät vain siihen, minkä ovat luvanneet tehdä. Avoimuudella tarkoitetaan sitä, kuinka tärkeää on se, että kaikki projektiin liittyvät tiedot ovat kaikille näkyviä. Jokaisen sprintin tulokset esitellään julkisesti ja päivittäiskokous on avoin kaikille. Päivit-

täiskokouksessa kuitenkin vain tiimin jäsenet saavat puhua. Kutakin tiimin jäsentä kunnioitetaan juuri sellaisena kuin hän on. Myös tiimin itseorganisoituvuutta ja työrauhaa kunnioitetaan. Tiimin jäsenten on myös kunnioitettava toisiaan tekeillä koko ajan parhaansa. Rohkeus on rohkeutta sitoutua, toimia, olla avoin. Omat virheet on myönnettävä, jotta niistä voi oppia. (Ketterät käytännöt 2009e.)

Scrumissa on erilaisia sääntöjä. Sprintin suunnittelukokous on päivän mittainen työpaja, jonka ensimmäisellä puoliskolla valitaan työlistasta seuraavan sprintin vaatimukset ja toinen puoli päivästä käytetään sprintin valmisteluun. Päiväpala-  
verin kesto on 5-15 minuuttia riippumatta tiimin koosta. Se pidetään joka työpäivä samaan aikaan samassa paikassa. Sprintti on 1-4 viikon mittainen iteraatio. Kun iteraatio on päättynyt, esitellään katselmoinnissa sprintin aikana toteutetut toiminnot. Sprintin katselmointi kestää enintään neljä tuntia. Katselmoinnin tarkoituksena on esittää sprintin tuloksia (kuvio 10). (Lindström 2006; Ketterät käytännöt 2009f.)



Kuvio 10. Scrum-malli (Lindström 2006.)

### 3.4 Dokumentointi ketterissä menetelmissä

Haikalan (2009b) mukaan ketterät menetelmät määrittelevät vaatimukset hyvin yleisellä tasolla. Esimerkiksi XP:ssä käytetään käyttäjätarinoita (user stories) ja scrumissa tuotteen toiminnallisuusluetteloa (product backlog). Toiminnallisuusluettelossa tuotteen ominaisuudet on priorisoitu listassa, jota päivitetään ja tarkennetaan jatkuvasti projektin edetessä (Lindström 2006).

Ketterissä menetelmissäkin liikkeelle lähdetään siis vaatimusmäärittelystä. Vaatimusmäärittelyn yksityiskohtien taso riippuu kehitettävästä systeemistä ja siitä, millainen prosessimalli on käytössä. Toisinaan täytyy kriittisen järjestelmäspesifikaation olla tarkka ja hyvin yksityiskohtainen. Silloin taas, kun vaatimukset ovat joustavammat ja käytössä on iteratiivinen kehitysprosessi, vaatimukset voivat olla vähemmän yksityiskohtaisia ja epäselvyydet ratkaistaan systeemin kehityksen aikana. (Sommerville 2007, 136).

Koska ketterät menetelmät ovat iteratiivisia (Eriksson–Penker 2000, 240), on niissä tuotettava dokumentointi kevyempää kuin perinteisissä prosessimalleissa. Ketterien menetelmien dokumentoinnin tulisi olla kevyttä, mutta riittävää (Chau–Maurer–Melnik 2003). Van Vlietin (2008, 44, 55) mukaan ketterät menetelmät eivät käytä paljon energiaa dokumentointiin. Varsinkaan kehityksen aikana dokumentointiin ei kiinnitetä paljoa huomiota. Ei kannata tuhlaa aikaa sellaiseen, joka kohta kuitenkin on jo vanhentunut. Dokumentit on kuitenkin toimitettava siinä vaiheessa, kun järjestelmä on valmis ja luovutettu asiakkaalle. Ketterissä menetelmissä luotetaan hiljaiseen tietoon. Van Vliet kehottaakin, että jos sinulla on kysyttävää, niin kysy ystävältäsi, äläkä kerää laajaa paperikasaa, joka ei kuitenkaan tarjoa vastausta.

Dokumentin arvo on sen pysyvyydessä. Jos ei ole muita kenen kanssa kommunikoida, niin dokumentissa tieto on pysyvästi luettavissa. Tämä tietenkin edellyttää, että dokumentti on olemassa. Ennen dokumentin tuottamista kannattaa miettiä muutamia kysymyksiä. Mikä on dokumentin tarkoitus? Keitä varten dokumentti tehdään? Voiko yhden dokumentin tehdä vastaamaan useamman eri

käyttäjäryhmän tarpeisiin? Parantaako dokumentti ihmisten välistä kommunikointia? Tarvitaanko dokumenttia silti? Kannattaako dokumentin ylläpito pidemmällä aikavälillä? Jos ei, on se jätettävä. (Koch 2004, 101–104.)

Ketterät menetelmät suosivat yksinkertaisia ja monikäyttöisiä dokumentteja, kuten itsedokumentoivaa koodia. Itsedokumentoiva koodi tarkoittaa ohjelmakoodia, joka on kirjoitettu siten, että se kuvaa myös vaatimukset, suunnittelun ja toteutuksen. (Koch 2004, 263–264.)

Eri ketterät menetelmät vaativat eri määrät dokumentointia. Ketterät menetelmät käyttävät kehitystiimin tuottamia eri dokumentointitasoja. Jotkut näistä eroista on löydettävissä menetelmästä itsestään. Melkein jokaisen ketterän ohjelmoijan mielestä kuitenkin jonkinlainen suunnitteludokumentti on tarpeen. Mielipiteet vaihtelevat siitä, mikä menetelmä on tarkoituksenmukaisin ja täyttää suunnitteludokumentointia koskevat minimivaatimukset. (Schuh 2004, 287, 293.)

Puuttuvat dokumentit voivat aiheuttaa pitkäaikaisia ongelmia tiimeille. Dokumentointia käytetäänkin jakamaan tietoa ihmisten välillä. Esimerkiksi uudella tiimin jäsenellä voi olla paljon kysyttävää projektista juuri silloin kun se, joka tietää asiasta, on vaihtanut työpaikkaa. Kysymyksiin voi saada vastauksen muilta tiimin jäseniltä tai lukemalla ne hyvästä dokumentista. Hyvän dokumentoinnin etuna on, että aikaa ei kulu muilta tiimin jäseniltä siihen kun yritetään selittää monimutkaista hanketta uudelle tiimin jäsenelle. Dokumentointi myös vähentää tietojen menetystä, jos työntekijä siirtyy toiseen projektiin tai vaihtaa työpaikkaa. Jos ohjelman elinkaari on pitkä, on dokumentoinnista hyötyä. (Paetsch–Eberlein–Maurer 2003, 5.) Ketterissä menetelmissä pitkän aikavälin ongelmia voisi rajoittaa kirjoittamalla mahdollisimman puhdasta ja kompaktia koodia. Ohjelmistoyrityksen olisi hyvä luoda ohjeistus koodauksesta, sillä esimerkiksi Perl-ohjelmointikielellä on mahdollista tuottaa sekä yksinkertaista että hyvin monimutkaista koodia.

Ketterissä menetelmissäkin dokumentointi kannattaa tehdä, etenkin jos tiimi on iso. Kunnollisella dokumentoinnilla säästetään paljon aikaa ja kustannuksia,

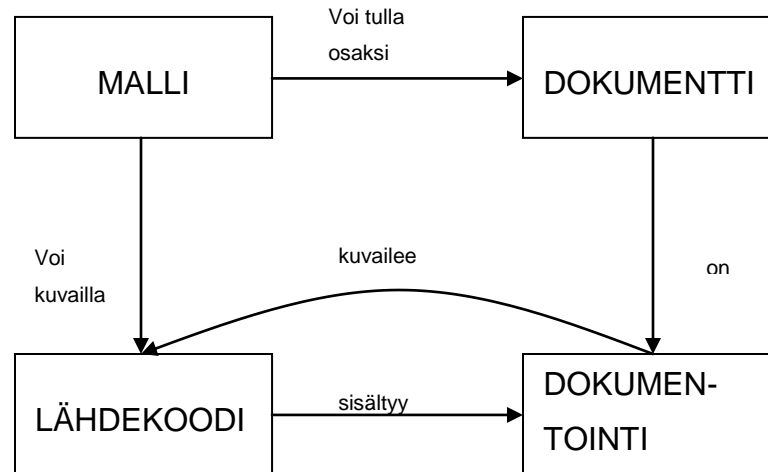
koska samoja asioita ei tarvitse selittää erikseen jokaiselle työntekijälle (Paetsch ym. 2003, 5). Ketterissä menetelmissä korostetaan sitä, että suurin osa dokumentoidusta tiedosta on jaettavissa tiimin jäsenten kesken. Ketterissä menetelmissä ei olla täysin ilman dokumentointia, mutta niissä käytetään mieluummin epämuodollista kuin muodollista dokumentointia. Kaikki ovat kuitenkin yhtä mieltä siitä, että dokumentoinnin on oltava kevyttä ja riittävää. (Chau ym. 2003.)

Nevalaisen (2008) mukaan se, että tehdään vain tarpeelliseksi koettuja kuvauksia ja muut jätetään tekemättä, johtaa siihen, että käytännössä dokumentointi jää tekemättä. Tai sitten asiakasdokumentointi on jotain sinne päin. Dokumentteja ei kuitenkaan tehdä vain suunnittelijoita varten, vaan niitä tarvitaan myöhemmin esimerkiksi ylläpidon ja jatkokehityksen tarpeisiin. (Nevalainen 2008, 18–20.)

Yleinen väärinkäsitys on, että ketteryys rinnastetaan dokumentoimattomuuteen. Ketterissä projekteissa dokumentoidaan, mutta eri tavalla kuin perinteisissä menetelmissä. Lähdekoodin laatuun kiinnitetään paljon huomiota. Käyttäjille tehdään käyttöohjeet. Koska testaus kulkee koko ajan mukana ketterässä menetelmässä, myös testitapaukset kertovat esimerkein kuinka ohjelmiston pitäisi toimia. (Raussi–Virtanen 2009, 16–19.) Ketterät menetelmät näkevät ensisijaisena dokumentointina koodin ja siihen liittyvät testitapaukset (Virtanen 2003, 22).

### 3.4.1 Parhaita käytäntöjä ketterän dokumentin kirjoittamiseen

Tämä luku esittelee Amblerin (2009a) parhaita käytäntöjä ketterän dokumentin kirjoittamiseen.

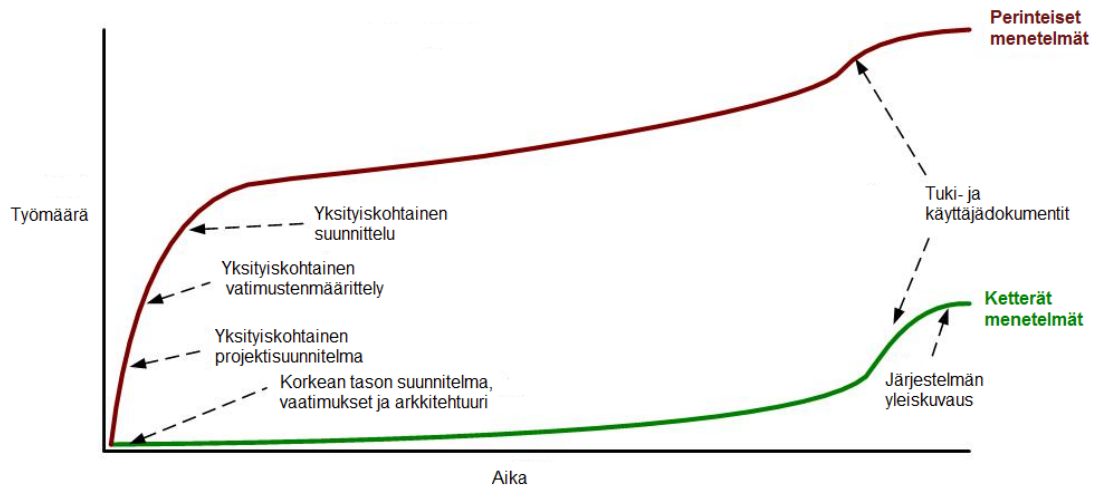


Kuvio 11. Mallien, dokumenttien, lähdekoodin ja dokumentoinnin suhteet Amblerin (2009b) mukaan.

**Dokumenttien kirjoittamisessa** kannattaa suosia suorittavia spesifikaatioita muuttumattomien dokumenttien sijaan. Yleensä suurin osa tiedosta on kirjattu perinteisiin määrittelydokumentteihin, kuten vaatimusmäärittelyyn, arkkitehtuuri-suunnitteludokumentteihin tai suunnitteludokumentteihin, mutta ne voidaan kirjata myös testauksen aikana. Testausta tehdään muun muassa tarkentamaan vaatimuksia ja vähentämään virheitä. (Ambler 2009a.)

Ambler (2009a) kehottaa dokumentoimaan vain pysyviä ajatuksia, ei spekulatiivisia ideoita. Kuten kuviossa 12 näkyy, strategiana ketterissä menetelmissä on siirtää kaikkien dokumenttien luominen niin myöhäiseksi kuin mahdollista. Dokumentit tehdään vasta sitten, kun niitä tarvitaan. Esimerkiksi järjestelmän kuvaus on parasta kirjoittaa kehityksen loppuvaiheessa, koska silloin tietää mitä on tehty. Samoin tuki- ja ohjedokumentit kannattaa kirjoittaa vasta kehitystyön elinkaaren lopussa. Muistiinpanoja voi tehdä läpi järjestelmän kehityksen, jolloin ei menetetä kriittistä tietoa. Muistiinpanojen ei tarvitse olla kuin pieniä kohdemuis-

tiinpanoja, koska viimeistelty dokumentti tehdään vasta juuri ennen lopullista toimitusta.



Kuvio 12. Dokumentoinnin vaiheita ohjelmistokehityksen elinkaaren aikana Amblerin (2009b) mukaan.

Se, että dokumentointi tehdään vasta viimeiseksi, säästää Amblerin (2009a) mukaan kuluja ja pienentää riskejä, koska silloin vähemmän dokumentteja on päivittämättä kun sitä ei ole. Jos kirjoittaa dokumentoinnin, joka sisältää epävaakaata tietoa, on olemassa riski, että joutuu työstämään dokumentin uudelleen, koska tieto on muuttunut. Projektin alussa ei siis kannata käyttää aikaa eri vaihtoehtojen dokumentoimiseen. Kannattaa odottaa kehitystyön elinkaaren loppupuolelle ennen kuin kirjoittaa dokumentoinnin. Silloin tieto on vakiintunut ja tiedetään, mikä tieto itse asiassa on tarpeellista. Näin ollen dokumentointi ei myöskään kulje aina muutamia iteraatioita jäljessä kehitystyöstä.

Ambler (2009a) kehottaa luomaan järjestelmädokumentoinnin. Nykyiset ohjelmistopohjaiset suunnittelutyökalut mahdollistavat olemassa olevan koodin tutkimisen monelta eri kantilta.

**Pidä dokumentointi yksinkertaisena**, mutta älä liian yksinkertaisena. Paras dokumentointi on yksinkertaisimmillaan sellainen, millä saa työn tehdyksi. Ei kannata tehdä 50-sivuista dokumenttia, jos viisi sivua riittää. Eikä viisisivuista,

jos viisi kohtaa riittää esittämään asian. Ei kannata luoda yksityiskohtaista ja monimutkaista dokumenttia, jos luonnos ajaa saman asian. Älä toista informaatiota muualla kuin mihin se kuuluu. Aloita minimaalisella asiakasvaatimuksilla ja täydennä tarvittaessa. Minimidokumentointi määräytyy sen mukaan, miten ja miksi asiakas aikoo käyttää sitä. Lyhyempi dokumentti ei sisällä yksityiskohtaista tietoa, mutta tarjoaa kartan, joka neuvoo mistä löytyy koodi tai muut dokumentit yksityiskohdille. Ambler (2009a) korostaa, että ei pidä laajaa dokumenttia automaattisesti heikompilaatuisena kuin suppeaa, mutta pitää sitä sellaisena kunnes toisin todistetaan.

Harvat dokumentit kannattaa kirjoittaa siten, ettei niissä ole päällekkäisyyttä. Kuttakin dokumenttia kirjoitetaan vain sen verran, että ne ovat juuri ja juuri tarpeeksi hyvä täyttääkseen tarkoituksen. Laajemmat dokumentit voi rakentaa pienemmistä dokumenteista. (Ambler 2009a.)

Informaatio laitetaan tarkoituksenmukaisimpaan paikkaan. Kannattaa miettiä, milloin dokumentointia tarvitaan? Myös työn laatu ohjaa dokumentoinnin sijoittamista. Ambler (2009a) kehottaa myös laittamaan informaation yleisesti nähtävälle. Kun mallit ovat nähtävillä julkisesti, esimerkiksi sisäisillä web-sivuilla, se edistää tiedonkulkua ja kommunikaatiota. Mitä enemmän kommunikaatiota projektissa on, sitä vähemmän tarvitaan yksityiskohtaista dokumentointia, koska ihmiset jo tietävät mitä ovat tekemässä. (Ambler 2009a.)

**Mitä dokumentoidaan?** Seuraavat käytännöt auttavat Amblerin (2009a) mukaan määrittämään, mitä dokumentoida:

- Dokumentin tarkoitus
- Keskitä todellisten asiakkaiden dokumentointitarpeet
- Asiakas päättää dokumentin riittävydestä.

Dokumentti pitäisi tehdä ainoastaan silloin, jos sille on selkeä tarve ja sillä on välitön päämäärä projektin tavoitteiden kannalta. Dokumentin tarve voi olla lyhyt- tai pitkäaikainen. Se saattaa välittömästi tukea ohjelmistokehityksen tavoitteita tai sitten ei. Pitää muistaa myös, että joka järjestelmällä on omat dokumentointi-



tarpeensa, yksi malli ei sovi kaikille. Tästä voidaan tehdä johtopäätös, että mitään tiettyä mallia ei ole olemassa, joka toimisi kaikissa projekteissa. (Ambler 2009a.)

Tiivis työskentely asiakkaiden kanssa selvittää, mitä vaatimuksia heillä on dokumentoinnin suhteen. Ensin täytyy tunnistaa potentiaaliset asiakkaat ja heidän tarpeensa. Sen jälkeen neuvotella heidän kanssaan siitä, mitä he todella tarvitsevat. Jotta saadaan selville asiakkaiden vaatimukset, kysytään mitä he tekevät, miten he sen tekevät ja kuinka he haluavat työskennellä tuotetun dokumentoinnin kanssa. Ymmärtämällä asiakkaiden tarpeet, voidaan toimittaa ytimekäs ja riittävä dokumentointi juuri sinne, missä sitä oikeasti tarvitaan ja mistä se myös löydetään. Ei ole väliä sillä, kuinka hyvin dokumentti on kirjoitettu, jos kukaan ei tiedä sen olemassaolosta. Dokumentin kirjoittajan on varmistettava, että dokumentilla on todellista merkitystä, ja että se tarjoaa käyttäjilleen vastinetta. Asiakkaan tehtävänä on vahvistaa, että tämä myös toteutuu. (Ambler 2009a.)

**Milloin dokumentoidaan?** Seuraavat käytännöt auttavat Amblerin (2009a) mukaan päättämään milloin dokumentoida:

- Toista, toista ja toista
- Löydä parempi tapa kommunikoida
- Aloita malleista, joita tosiaan pidät yllä
- Päivitä ainoastaan, kun on tarve.

Ihanteellisesti dokumentointia pitäisi luoda läpi ohjelmiston suunnittelun elinkaaren, silloin kun se on mielekästä. Yleensä mielekkäin hetki on elinkaaren loppupuolella. Kirjoittamisessa pitää ottaa inkrementaalinen ja iteratiivinen lähestymistapa, jolloin saa palautetta dokumentin todellisesta arvosta. Toisin sanoen, kannattaa kirjoittaa pieninä paloina, näyttää jollekin, saada palautetta, työstää palautetta ja sitten toistaa. Parhaat dokumentit on kirjoitettu iteratiivisesti, eli kaikkea ei ole kirjoitettu kerralla. Iteratiivinen lähestymistapa auttaa tiedostamaan, millaista dokumentointia itse asiassa tarvitaan. (Ambler 2009a.)

Kommunikaation ensisijainen päämäärä on ymmärtäminen, ei suinkaan dokumentointi. Siksi dokumentointia ei tulisi yliarvostaa. Hyvin kirjoitettu dokumentti toki tukee organisaation muistia tehokkaasti, mutta on keuhko tapa kommunikoida projektin aikana. Dokumentointi tukee tiedon siirtymistä, mutta se on vain yksi mahdollinen tapa. (Ambler 2009a.)

Mallit joita ei ole päivitetty, eivät todennäköisesti ole tärkeitä. Ne joita on ylläpidetty läpi kehityksen, ovat tärkeitä. Dokumentoinnin voi rakentaa niiden ympärille. Ambler (2009a) suosittaa päivittämään dokumentin ainoastaan silloin, kun on pakko. Ketterien dokumenttien, aivan kuten ketterien mallienkin, pitäisi olla juuri riittävän hyviä. Usein dokumentit voivat olla päivittämättä, mutta silloin se ei haittaa kovin paljon. (Ambler 2009a.)

**Dokumentoinnin pyrkimykset?** Seuraavat käytännöt auttavat parantamaan dokumentoinnin pyrkimyksiä:

- Käsittele dokumentteja kuten vaatimuksia
- Tunnista dokumentoinnin tarpeet
- Hanki joku, jolla on kirjoittajakokemusta

Vaatimuksen pitäisi olla arvioitua, priorisoitua ja sen tulisi sovitaa työn pienet osat kokoon muiden osien kanssa. Dokumentin kirjoittaminen selkeästi on vaatimus, kuten on toiminnon kirjoittamisenkin tarve. Käsittelemällä dokumenttia kuten vaatimusta, tehdään selkeä päätös sidosryhmien huomioimiseksi. Pohjimmiltaan dokumentoinnin investoinnit ovat liiketoimintaan liittyviä päätöksiä, eivät teknisiä sellaisia; dokumenttia ei pidä luoda, koska prosessi sanoo niin – se pitää luoda silloin, kun sidosryhmät niin sanovat. (Ambler 2009a.)

Dokumentin tulisi tarjota parasta vastinetta omistajilleen. Silloin on mietittävä, kuinka paljon siihen halutaan investoida. On hyvä vaatia ihmisiä myös perustelemaan dokumentoinnin vaatimukset. Todella hyviä kysymyksiä ovat, mihin he aikovat käyttää dokumentointia ja miten? Tekemällä näin huomataan, että kaikkia dokumentteja ei käytetä. Sen sijaan vain halutaan, että dokumentit ovat olemassa varmuuden vuoksi. Kannattaakin yksinkertaisesti kysyä asiakkaalta, mitä

dokumentteja tämä tarvitsee, eikä haluaako niitä. Dokumentoinnin hyödyn täytyy olla suurempi kuin kustannukset sen luonnista ja ylläpidosta. (Ambler 2009a.)

Toimivan ohjelmiston lisäksi tarvitaan myös käyttöohjeet, tuki- ja ylläpitodokumentit sekä järjestelmän yleiset dokumentit. Tiimin ensisijaisena tavoitteena on kehittää ohjelmaa, toissijaisena tavoitteena on mahdollistaa seuraava vaihe. On tärkeää rakentaa sidosryhmien tarpeita vastaava korkealaatuinen toimiva ohjelmisto, mutta tärkeää on myös varmistaa ohjelmiston ylläpidettävyys, kehitys sekä käyttö- ja tukitoiminnot. Dokumentoinnin tulisi palvella jatkuvasti tarkoitusta. Ylimmän tason tiedon hallitseminen on tärkeää – ei yksityiskohtien. Tärkeän tiedon jatkuva hallitseminen ja joskus säännöt edellyttävät tietyn tason dokumenttia. (Ambler 2009a.)

Tekniset kirjoittajat ovat arvokaita silloin, kun on dokumentin kirjoittamisen aika. He tietävät kuinka järjestää ja esittää tietoa tehokkaasti. Jollei ole mahdollisuutta tekniseen kirjoittajaan, on seuraavassa muutamia strategioita kirjoittamiseen: kannattaa tutustua oppaisiin, joissa opetetaan kirjoittamaan teknisistä asioista niin, että kaikki ymmärtävät. Eli opettele kirjoittamisen perusteet. Kokeile kirjoittaa dokumentointi parin kanssa. Jaettu omistus kaikille dokumentoinnille, jotta useat voivat työskennellä dokumentoinnin parissa. Hanki ohjelma, joka muuttaa tekstin puheeksi. Se sallii kuunnella mitä on kirjoitettu ja on hieno tapa löytää huonosti kirjoitetut kohdat. (Ambler 2009a.)

### **3.4.2 Amblerin ketterän asiakirjan kriteerit**

Amblerin (2009b) mukaan asiakirja on ketterä, kun se täyttää seuraavat kriteerit:

1. Ketterät asiakirjat maksimoivat sidosryhmien sidotun pääoman tuottoprosentin.
2. Ketterien dokumenttien hyödyt ovat suuremmat kuin investoinnit niiden luomiseen ja ylläpitoon. Sidosryhmien on siis ymmärrettävä dokumenttien kokonaiskustannukset ja oltava valmiita investoimaan niiden luomiseen ja ylläpitoon.

3. Kun kirjoitat ketterää dokumenttia, muista periaatteena yksinkertaisuuden oletus: yksinkertaisin dokumentointi on riittävää. Luo siis mahdollisimman yksinkertaista sisältöä.
4. Ketterät dokumentit täyttävät tarkoituksensa, kun asiakirjat ovat yhtenäisiä. Ja jos et tiedä miksi olet luomassa dokumenttia, lopeta sen tekeminen ja ajattele uudelleen mitä teet.
5. Dokumenteissa kuvataan asiat, jotka on hyvä tietää. Ei siis itsestään selviä asioita, vaan ainoastaan tärkeät tiedot.
6. Dokumentti on kirjoitettu tietyn kohderyhmän tarpeisiin. Järjestelmädokumentit on tyypillisesti kirjoitettu kehittäjille, tarjoten heille katsauksen arkkitehtuuriin sekä yhteenvedon kriittisistä vaatimuksista ja suunnittelu päätöksistä. Käyttäjädokumentit sisältävät usein käyttöohjeet järjestelmän käyttämiseen, jolloin ne on kirjoitettava sellaisella kielellä, että käyttäjä ymmärtää sitä. Eli erilaiset asiakkaat, erilaiset dokumentit ja erilaiset kirjoitustyyli. On siis työskenneltävä lähellä asiakasta (tai mahdollisia asiakkaita), jotta dokumentoinnista voi luoda sellaisen, että se välittömästi kohtaa asiakkaan tarpeet.
7. Dokumentit ovat riittävän tarkkoja, johdonmukaisia ja yksityiskohtaisia. Niiden ei tarvitse olla täydellisiä, vaan riittävän hyviä.

## 4 TOIMINNALLISEN MÄÄRITTELYDOKUMENTIN LUOMINEN

### 4.1 Sisällönhallintajärjestelmä

Ensin on paikallaan valottaa hieman sitä, mikä on sisällönhallintajärjestelmä. Sisällönhallintajärjestelmä on yleisnimitys tietojärjestelmälle, joka palvelee koko organisaation sisällönhallintaa. Erilaisia sisällönhallintajärjestelmiä ovat esimerkiksi dokumenttienhallintajärjestelmät, julkaisujärjestelmät eli www-sisällönhallintajärjestelmät, verkkokauppajärjestelmät sekä aineistonhallintajärjestelmistä muun muassa kuva-aineistot, videot ja multimedia. Sisällönhallinnalla tarkoitetaan toimintaa, jonka tarkoituksena on hallita digitaalista informaatioisisältöä mahdollisimman tarkoituksenmukaisesti. Sisältö voi olla dokumentteja, www-sivuja, audio-, video- ja kuvatiedostoja, sähköpostiviestejä ynnä muuta tallennettavissa olevaa informaatiota. (Wikipedia 2009c.)

Www-sisällönhallinta tarkoittaa toimintaa, jossa mahdollisimman tarkoituksenmukaisesti hallinnoidaan verkkopalvelun sisältöjä. Luonteeltaan www-sisällönhallinta on julkaisupainotteista sisällönhallintaa. Keskeisessä asemassa ovat sivupohjat, jotka koostavat pienistä sisältöyksiköistä www-sivut ja pitävät verkkopalvelukokonaisuuden kasassa. Www-sisällönhallintajärjestelmät liittyvät olennaisesti www-sisällönhallinnan käytännön toteutukseen. Www-sisällönhallinnalle tyypillistä on sisältöjen, rakenteiden ja ulkoasun erottaminen toisistaan. Sivupohjien avulla toteutettava kokonaisuus mahdollistaa esitysmuotojen yhtenäisyyden ja keskitetyn ylläpidon. Sivupohjien sisältö koostuu esimerkiksi navigaatio-elementeistä sekä kaikilla www-sivuilla toistuvasta grafiikasta, kuten logoista, taustaväreistä, muotoiluista, otsikoista ja pudotusvalikoista. Sivupohjiin perustuva julkaisu mahdollistaakin useiden erilaisten päätelaitteiden ja jakelukanavien huomioimisen. (Wikipedia 2009c.)

Syventävän perusharjoittelun aikana tehtäväksi saatu toiminnallisen määrittelyn päivittäminen eteni siten, että ensin oli perehdyttävä kirjallisuuden kautta ohjelmistotuotantoon. Sitä kautta ymmärrys ohjelmistotuotannosta hahmottui ja ymmärrettiin, että dokumentointi on myös osa ohjelmistotuotantoa. Toiminnallinen

dokumentti jouduttiin luomaan lähes alusta, sillä vanhaa versiota ei saatu käyttöön kuin muutaman kommentin verran.

Tehtävä osoittautui melko haasteelliseksi, koska tekijöillä ei ollut aikaisempaa kokemusta dokumentoinnin tuottamisesta. Koska tilaajayritys soveltaa ketterää menetelmää, oli myös ketterään ohjelmistokehitykseen perehdyttävä. Ketterät menetelmät poikkeavat perinteisistä ohjelmistotuotantoprosesseista siten, että siinä tuotetaan mieluummin puhdasta koodia kuin paljon dokumentteja. Joissain ketterissä malleissa dokumenttien tuottaminen sijoittuu ohjelmiston kehityskaaren loppupuolelle, toisissa malleissa dokumentteja tehdään rinnan ohjelmiston kehityksen kanssa.

Perinteisissä menetelmissä toiminnallinen määrittely syntyy vaatimusmäärittelyn pohjalta. Toiminnallinen määrittely pyrkii kuvaamaan sen, miten järjestelmän halutaan toimivan, millaisia toimintoja järjestelmässä pitäisi olla. Suunnittelun tehtävänä on sitten muuttaa määrittely tekniselle kielelle arkkitehtuurin ja moduulien suunnittelun kautta tekniseksi määrittelyksi. (Haikala–Märijärvi 2004, 39, 81.) Tässä raportoitava dokumentti tehtiin, kun itse ohjelma oli melkein valmis. Toiminnallinen määrittely tehtiin jo lähes valmiista ohjelmasta, jolloin samalla tapahtui myös ohjelman testaamista. Toiminnallinen määrittely tehtiin siis ikään kuin käänteisessä järjestyksessä.

## **4.2 Dokumentin työstäminen**

Toiminnallista määrittelyä lähdettiin työstämään tutustumalla ensin ohjelmistotuotantoon ja ketteriin menetelmiin. Suurimmaksi osaksi tekemämme työ olikin ajatustyötä ja uuden oppimista, koska meillä ei ollut kovin kattavaa aikaisempaa tietämystä ohjelmistotuotannosta. Ketterät menetelmät olivat kokonaan uusi tuttavuus. Termin määrittelydokumentit olimme kuulleet, mutta dokumenttien sisältö ei ollut tuttu. Lähdimme siis lähes nollatietämyksellä rakentamaan toiminnallista määrittelyä. Saimme seuraavalla tapaamisella avuksi oheismateriaalia, joka koostui kampanja-, palvelu- ja esittelykuvauksista sekä ohjeista. Saimme käyt-

töömme myös edelliset versiot toiminnallisesta ja teknisestä -määrittelystä sekä koe-version uudesta ohjelmasta.

Toimeksiantaja halusi kokonaan uuden toiminnallisen määrittelyn, joten edellinen versio toiminnallisesta määrittelystä oli melkein puhtaaksi pyyhitty sisältäen vain määrittelyrunгон ja muutamia tietoja. Meidän tehtäväksemme jäi prosessin ja niihin liittyvien käytötapausten kuvaaminen sekä kaikkien järjestelmään kuuluvien toiminnallisten vaatimuksien kuvaaminen ja niiden hyväksymiskriteerit.

<b>Sisällysluettelo</b>	
1.	Johdanto.....5
1.1.	xxx projektin tarkoitus .....5
1.2.	Lähtökohta.....5
1.3.	Lopputulos .....5
2.	Liiketoimintamallit .....5
2.1.	xxxx nykyiset liiketoimintamallit .....5
2.2.	xxx ohjelmiston Prosessikaavio.....5
3.	Käsitteet .....5
4.	Käyttäjät ja roolit .....5
4.1.	Esim. Ylläpitäjä .....5
4.2.	xxxx.....5
5.	Toiminnot ja käytötapaukset.....5
5.1.1.	UC0001, xxxx .....5
6.	Yleiset ei-toiminnalliset vaatimukset.....5
6.1.	Formaattivaatimukset.....5
6.2.	Asentamiseen liittyvät vaatimukset.....5
6.3.	Dokumentointivaatimukset.....5
6.4.	Ylläpitovaatimukset.....6
6.5.	Siirrettävyysvaatimukset.....6
6.6.	Tietoturva vaatimukset.....6
6.7.	Lokalisointivaatimukset .....6
7.	Käyttöliittymä (asiakasrajapinta).....6
7.1.	WI01 xxxx .....6
7.1.1.	xxxx.....6
8.	Rajapinnat .....7
8.1.	Rajapinnat muihin järjestelmiin .....7
8.2.	Raportit.....7
8.3.	Sovellukset .....7
8.4.	Kommunikointi.....7
9.	Tietovarastot.....7
9.1.	Tietokannat.....7
9.2.	Muut tietovarastot.....7
10.	Lyhenteet.....7
11.	Käytettyjen menetelmien kuvaus.....7

Kuvio 13. Määrittelydokumentin sisällysluettelo (Mielikäinen 2006.)

Etsimme toiminnallisen määrittelyn malleja niin kirjoista kuin Internetistäkin. Internetistä löysimmeikin valmiita määrittelypohjia, joista yksi oli samanlainen (kuvio 13) kuin saamamme dokumentti, mutta sisältöä niissäkään ei ollut sen enempää kuin aikaisemmassa versiossa. Teknisestä määrittelystä oli siten suuri apu toiminnallista määritelmää luodessamme. Pystyimme sieltä tarkistamaan

suuntaa sille, millaisia kuvailuja kulloinkin tekeillä olevaan kohtaan haettiin. Jos näytti, että emme päässeet jossakin kohdassa eteenpäin, jätimme kohdan sikseen ja siirryimme seuraavaan, kuten toimeksiantaja oli meitä ohjeistanut. Niin etenimme määrittelyn tekemisessä viimeiseen toimintoon ja aloitimme alusta uudestaan. Teimme toiminnallista määrittelyä iteratiivisesti, siis aivan kuten ketterissä menetelmissä on tapana toimia. Tekemämme toiminnallinen määrittely poikkeaa Mielikäisen määrittelydokumenttipohjan sisällöstä jonkin verran, kuten kuviosta 14 voidaan havaita.

Sisällysluettelo	
1.	Johdanto ..... 4
1.1.	xxx projektin tarkoitus ..... 4
1.2.	Lähtökohta ..... 4
1.3.	Lopputulos ..... 4
2.	Käsitteet ..... 4
3.	Käyttäjät ja roolit ..... 4
4.	Toiminnot ja käyttötapaukset ..... 4
4.1.	Kirjautuminen ..... 13
4.1.1.	UC0101, Sisäänkirjautuminen ..... 13
4.2.	Sivustonhallinta ..... 15
4.2.1.	UC0201, Sivustorakenne ja editori ..... 15
4.2.2.	UC0202, Lisää sivu ..... 20
4.2.3.	UC0203, Poista sivu ..... 21
4.2.4.	UC0204, Muokkaa sivun ominaisuuksia ..... 22
4.3.	Tiedostopankki ..... 23
4.3.1.	UC0301, Lisää tiedosto ..... 23
4.3.2.	UC0302, Tiedostohaku ..... 24
4.3.3.	UC0303, Poista tiedosto ..... 25
4.4.	Kuvapankki ..... 26
4.4.1.	UC0401, Lisää kuva ..... 26
4.4.2.	UC0402, Kuvahaku ..... 27
4.4.3.	UC0403, Poista kuva ..... 27
4.5.	Uutisryhmä ..... 28
4.5.1.	UC0501, Lisää uutinen ..... 28
4.5.2.	UC0502, Muokkaa uutista ..... 29
4.5.3.	UC0503, Poista uutinen ..... 30
4.6.	Käyttäjähallinta ..... 31
4.6.1.	UC0601, Lisää käyttäjä ..... 31
4.6.2.	UC0602, Poista henkilö ..... 33
4.6.3.	UC0603, Nimeä ja poista ryhmiä ..... 34
5.6.4.	UC0604, Lisää ryhmä ..... 35
5.	Yleiset ei-toiminnalliset vaatimukset ..... 35
5.1.	Formaattivaatimukset ..... 35
5.2.	Asentamiseen liittyvät vaatimukset ..... 35
5.3.	Dokumentointivaatimukset ..... 35
6.	Ohjelmistoarkkitehtuuri ..... 36
6.1.	Tekninen arkkitehtuuri ..... 36
6.2.	Ohjelmistoarkkitehtuuri ..... 37
6.3.	Moduulit ..... 37
6.3.1.	Kirjautuminen ..... 37
6.3.2.	Sivustonhallinta ..... 37
6.3.3.	Tiedostopankki ..... 37
6.3.4.	Kuvapankki ..... 38
6.3.5.	Uutisryhmä ..... 38
6.3.6.	Käyttäjähallinta ..... 38
6.3.7.	Ohjeet ..... 38
7.	Lyhenteet ..... 38
8.	Käytettyjen menetelmien kuvaus ..... 38

Kuvio 14. Sisällysluettelo valmiista toiminnallisesta määrittelydokumentista

Määrittelydokumentin **ensimmäisessä luvussa** eli johdannossa kerrotaan projektin tarkoitus, lähtökohta ja lopputulos. Tässä projektissa asiakas halusi saada



Internet-sivustonsa ylläpitämiseen mahdollisimman helppokäyttöisen ja toimivan järjestelmän, jonka avulla sivustoa päivitetäisiin. Lopputuloksena syntyi asiakkaalle räätälöity sivustonhallintajärjestelmä, jota käytetään Internet-selaimella ja johon kirjaututaan URL-osoiterivin kautta.

**Toisessa luvussa** kävimme läpi dokumentissa käytettyjä merkintätapoja eli yhtenä esimerkkinä oli painikkeiden esittäminen lihavoidulla fontilla.

**Kolmannessa luvussa** kuvasimme käyttäjät ja roolit. Käyttäjryhmiä olivat pääkäyttäjät ja käyttäjät, joilla ovat eri oikeudet järjestelmään. Pääkäyttäjät-ryhmään kuuluvat käyttäjät omaavat automaattisesti kaikki oikeudet palveluun ja ainoastaan he voivat antaa käyttäjät-ryhmille rajattuja oikeuksia.

Käyttäjät-ryhmään kuuluvilla käyttäjillä on rajatut oikeudet palvelussa, esimerkiksi jos käyttäjällä ei ole oikeuksia moduuleihin, näkyvät kuvakkeet harmaina. Käyttäjä pääsee vain niihin moduuleihin, joihin hänelle on annettu oikeudet. Moduuli, johon on oikeus, näkyy oranssina.

**Neljännessä luvussa** kuvasimme kaikki järjestelmään tulevat prosessit sekä niihin liittyvät käyttötapaukset. Lisäksi kuvasimme kaikki järjestelmän toiminnalliset vaatimukset sekä niiden hyväksymiskriteerit.

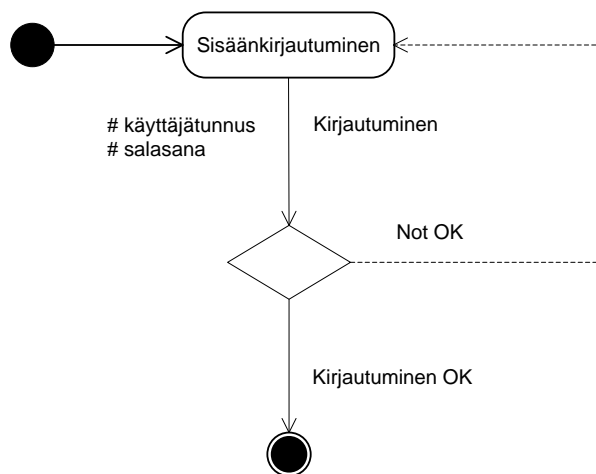
Kuvasimme kaikki ohjelmaan kuuluvat toiminnot sanallisesti, kaavioiden ja käyttötapauksen avulla. Aloitimme ohjelmiston toimintojen kuvaamisen järjestelmään kirjautumisella eli ensin kirjoitimme toiminnon ja sen jälkeen kuvasimme lyhyesti, mitä käyttäjän pitää tehdä päästäkseen järjestelmään sisään. Lopuksi piirsimme toiminnoista kaaviokuvan sekä kirjasimme toiminnon tarkaksi käyttötapaukseksi.

*Kirjautumisvaiheessa* järjestelmä näyttää käyttäjälle kaksi tekstikenttää, joihin käyttäjä kirjaa oman käyttäjätunnuksen ja salasanan. Painamalla Login-painiketta käyttäjä kirjautuu sisään järjestelmään (kuvio 15).



Kuvio 15. Järjestelmän sisäänkirjautumisikkuna

Sama tapahtuma on esitetty järjestelmän kannalta katsottuna kuviossa 16. Tässä järjestelmä tarkistaa käyttäjätunnuksen ja salasanan oikeellisuuden ja päästää käyttäjän sisään, jos tunnukset ovat oikeita. Jos tunnukset ovat virheellisiä, sisäänkirjautuminen epäonnistuu.



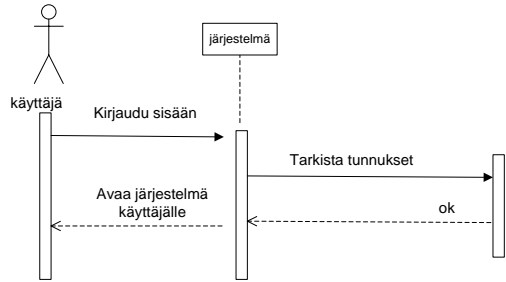


Kuvio 16. Järjestelmän sisäänkirjautumisvaihe

Tämän jälkeen kävimme valmista ohjelmaa läpi toiminto toiminnolta ja kirjasimme kaikki toiminnot sanatarkoiksi käyttötapauksiksi (use case). Käyttötapaukset ovat oliomallinnukseen yhteydessä yleinen käytetty tapa kuvata järjestelmälle asetetut vaatimukset. Perinteisessä ohjelmistosuunnittelussa luodaan osana vaatimusmäärittelyn dokumentointia kaikki järjestelmään liittyvät käyttötapaukset kokoava use case -malli. Sen tarkoituksena on kuvata järjestelmän toiminta niin

kuin se näkyy käyttäjälle. (Pohjonen 2002, 152.) Käyttötapauksessa selostetaan yksityiskohtaisesti, mitä tapahtuu kun painaa jotain painiketta.

Esimerkkinä käyttötapauksesta sisäänkirjautuminen.

Tunniste	Kirjautuminen – Sisäänkirjautuminen
Kuvaus	Käyttäjä kirjautuu onnistuneesti sisään järjestelmään.
Alkuehto	Käyttäjän tiedot löytyvät tietokannasta, hänelle on valmis tunnus ja salasana.
Normaali tapahtumien kulku	Käyttäjä kirjoittaa oma käyttäjätunnuksen <b>Login name</b> -ikkunaan ja salasanan <b>Password</b> -ikkunaan. Painamalla  -painiketta käyttäjä kirjautuu sisään järjestelmään.
Vaihtoehtoinen tapahtumien kulku	Jos käyttäjätunnus tai salasana tai molemmat ovat väärä, järjestelmään ei pääse kirjautumaan sisään. Ikkunat tyhjentyvät.
Loppuehto	Käyttäjä on onnistuneesti kirjautunut sisään järjestelmään.
Erikoisvaatimukset	Käyttäjätunnus on oikea ja salasana on oikea.
Käyttäjät	Pääkäyttäjät ja käyttäjät
Järjestelmäversio	3.0
Näyttömalli	
Tila UML	 <pre> sequenceDiagram     actor käyttäjä     participant järjestelmä     käyttäjä-&gt;&gt;järjestelmä: Kirjautu sisään     activate järjestelmä     järjestelmä-&gt;&gt;järjestelmä: Tarkista tunnukset     activate järjestelmä     järjestelmä--&gt;&gt;järjestelmä: ok     deactivate järjestelmä     järjestelmä--&gt;&gt;käyttäjä: Avaa järjestelmä käyttäjälle     deactivate järjestelmä     </pre>

Kuvio 17. Käyttötapaus sisäänkirjautumisesta

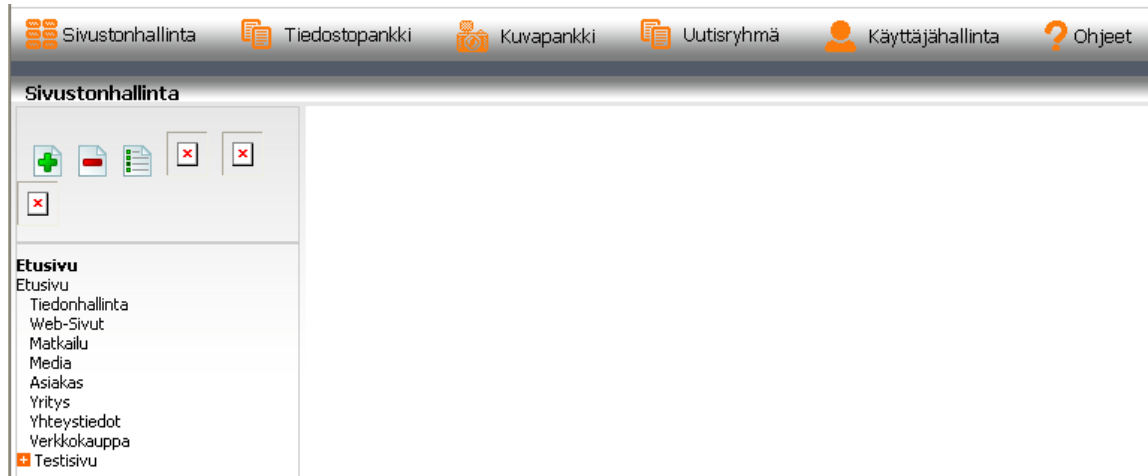
Käyttötapauksia on elävöitetty ja selkiytetty painikkeiden kuvilla. Käyttötapauskuvaukset koostuvat seuraavista osista: kuvaus, alkuehto, normaali tapahtumien kulku, vaihtoehtoinen tapahtumien kulku, loppuehto, erikoisvaatimukset, käyttäjät, järjestelmäversio, näyttömalli ja tila. Kuvaus-kohdassa kerrotaan mitä toiminnossa tapahtuu, eli kyseessä on toiminto, jossa käyttäjä kirjautuu sisään järjestelmään. Normaali tapahtumien kulku -kohdassa selitetään kuinka sisäänkirjautuminen tapahtuu (kuvio 17).

Vaihtoehtoinen tapahtumien kulku kertoo mitä tapahtuu, jos sisäänkirjautuminen epäonnistuu. Loppuehto kuvaa tapahtumaa, mihin toiminnon suorittaminen johtaa, eli esimerkkitapauksessa käyttäjä on onnistuneesti kirjautunut sisään järjestelmään. Erikoisvaatimuksissa kerrotaan ne vaatimukset, joiden pitää toteutua, että käyttäjä voi kirjautua sisään järjestelmään. Tässä tapauksessa täytyy siis salasanan ja käyttäjätunnuksen olla oikeita. Käyttäjä-kohtaan on kirjattu ne käyttäjäryhmät, jotka voivat olla käyttäjinä tässä toiminnossa. Järjestelmäversio ilmoittaa järjestelmäversion numeron.

Näyttömallikohdassa on otettu kuvakaappaus käyttöliittymästä silloin kun ollaan sisäänkirjautumistilassa. Tila-kohdassa on UML-mallinnuksen mukainen kaavio järjestelmän tilasta.

Sisäänkirjautumisvaiheen jälkeen avautuu näkymä järjestelmän etusivulle, joka on myös nimeltään *sivustonhallinta* (kuvio 18). Näkymän yläreunassa näkyy järjestelmän viisi päätoimintoa. Toiminnot vasemmalta oikealle ovat: sivustonhallinta, tiedostopankki, kuvapankki, uutisryhmä ja käyttäjähallinta. Oikeassa yläreunassa oleva ohjeet-toiminto avaa järjestelmän käyttöohjeet erilliseen ikkunaan pdf-tiedostomuodossa.

*Sivustonhallinnassa* hallitaan itse sivustoja ja niiden sisältöä. Sivuja voidaan lisätä, poistaa tai muokata.

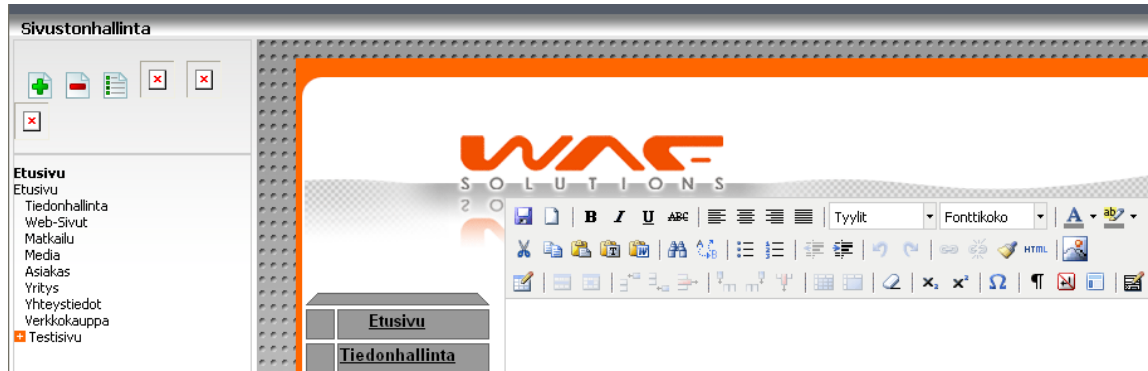


Kuvio 18. Näkymä etusivulta




*Tiedostopankki* ja *kuvapankki* ovat tallennuspaikkoja kaikille sivuilla näkyville liitetiedostoille ja kuville. Tiedostopankissa ja kuvapankissa voidaan lisätä, hakea ja poistaa sekä tiedostoja että kuvia. Kuvapankki muotoilee kaikki kuvat automaattisesti sivustolle sopiviksi. Käyttäjällä on mahdollisuus lisätä Kuvapankkiin kuva tai Tiedostopankkiin tiedosto, antaa niille esimerkiksi kuvaukset tai laittaa hakusanat (liite 4).

*Uutisryhmässä* julkaistaan ajankohtaisia tiedotteita ja uutisia. Uutisia voidaan lisätä, muokata ja poistaa. Liite 5 esittää uutisryhmän näyttömallikuvaa aloitusnäkymässä. Kuviossa näkyy uutisen luomispäivä, otsikko, onko uutinen julkinen vai ei sekä uutisen käyttäjä-ryhmä.

*Käyttäjähallinta* toimii palvelun työkaluna, joka mahdollistaa keskitetyn käyttäjien yksilöllisen ja dynaamisen käyttöoikeuksien hallinnan. Käyttäjähallinta antaa myös palvelun käyttäjille mahdollisuuden muuttaa omia henkilökohtaisia tietojaan, kuten salasanaan palveluun. Käyttäjähallinnassa lisätään ja poistetaan käyttäjäryhmiä ja käyttäjiä sekä hallitaan oikeuksia moduuleihin.

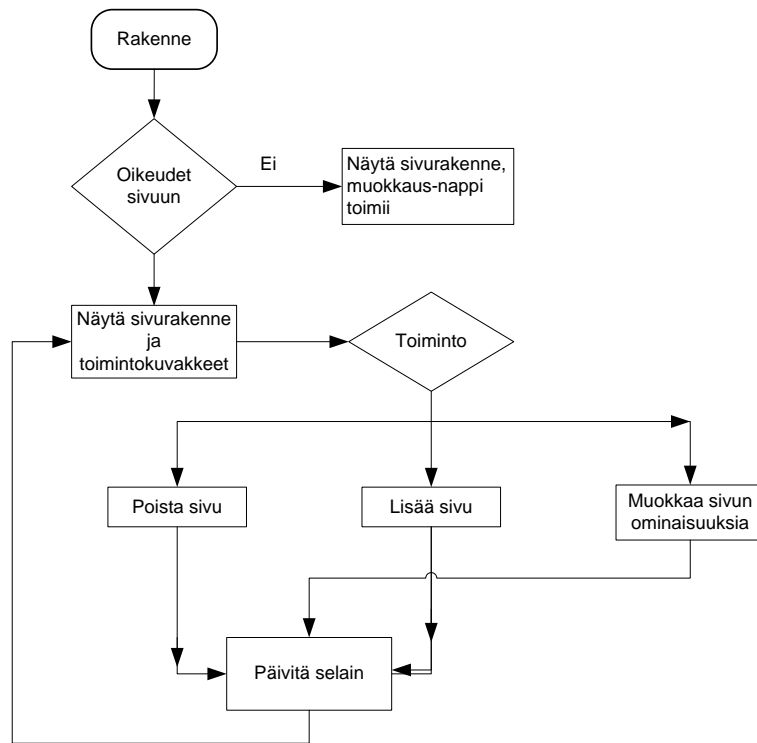


Kuvio 19. Editorinäkymä

Sivustonhallinnan vasemmassa laidassa näkyy puurakenteena sivustorakenne, jossa sivuja voidaan lisätä, poistaa tai muokata (kuvio 19). Sivustorakenteesta valitaan sivu, johon halutaan tehdä muutoksia. -kuvakkeesta lisätään uusi sivu valitun sivun alle, -kuvakkeesta valittu sivu poistetaan ja -kuvakkeesta voidaan muokata valitun sivun ominaisuuksia.

Olemassa olevia sivuja voidaan muokata valitsemalla sivustorakenteesta se sivu, jota halutaan muokata. Valittu sivu avautuu työtilänäkymään ja klikkaamalla hiirellä jotain kohtaa sivusta avautuu näkymään editori (kuvio 19). Editorissa voidaan muokata olemassa olevia sivuja ja myös luoda uusia sivuja. Editorin muokkaustoiminnot näkyvät Windows-maailmasta tuttuina toimintakuvakkeina.

Sivustorakenteen toiminnoista tehtiin kaaviokuva kuvion 19 mukaisesti. Kuvio 20 esittää sivustorakenteen toimintokuvauksen, jossa järjestelmä tarkastaa onko käyttäjällä oikeudet sivuun. Jos riittävät oikeudet ovat olemassa, niin käyttäjä pääsee tekemään haluamansa toiminnot. Jos riittäviä oikeuksia ei ole, niin sivut näkyvät harmaana eikä niitä pääse muokkaamaan.



Kuvio 20. Sivurakenteen toiminnot

Tämän jälkeen teimme sivustorakenteesta ja editorista samanlaisen käyttötapauskuvauksen (liite 2 ja 3) kuin sisäänkirjautumisestakin. Kuvasimme jokaisen järjestelmän toiminnon samoin kuten edellä eli kuvasimme sanallisesti *kirjautumisen* ja *sivustonhallinnan* lisäksi *tiedostopankin*, *kuvapankin*, *uutisryhmän* ja *käyttäjähallinnan*. Piirsimme myös kaavioita ja teimme tarkat käyttötapauskuvaukset järjestelmään *sisäänkirjautumisen* lisäksi *uloskirjautumisesta*, jo edellä mainituista sivustonhallinnan *sivurakenteesta* ja *-editorista*, tiedosto- ja kuvapankin *lisää*, *hae* ja *poista* -toiminnoista, uutisryhmän *lisää*, *muokkaa* ja *poista* -toiminnoista sekä käyttäjähallinnan *lisää käyttäjä*, *poista henkilö*, *nimeä* ja *poista ryhmä* sekä *lisää ryhmä* -toiminnoista.

Kuvasimme **viidennessä luvussa** yleisiä ei-toiminnollisia vaatimuksia, kuten formaattivaatimuksia, asentamiseen liittyviä vaatimuksia sekä dokumentointivaatimuksia. Tiedostopankin tiedostomuotoihin sopivat kaikki formaattityypit, mutta

kuvapankissa käytettiin ainoastaan gif ja jpg -formaatteja. WAFnet 3.0 vaati toimivan Internet-yhteyden. Järjestelmää suositeltiin käytettäväksi joko Internet Explorer 7+ tai Mozilla Firefox 2 -selaimella. Dokumentointivaatimuksina olivat toiminnallinen ja tekninen määrittely sekä käyttöohjeet.

**Kuudennessa luvussa** kuvasimme teknistä arkkitehtuuria ja ohjelmistoarkkitehtuuria piirrosten avulla sekä luettelimme moduulien sisältämät toiminnot (kirjautuminen, sivustonhallinta, tiedostopankki, kuvapankki, uutisryhmä, käyttäjähallinta ja ohjeet). **Seitsemännessä luvussa** kävimme läpi lyhenteitä, esimerkiksi mitä tarkoittavat CGI ja HTTP. **Kahdeksannessa luvussa** kuvasimme yleisesti käytettyjä menetelmiä.

### 4.3 Vertailua ja analysointia

Kuten luvussa 2.3 sanotaan, testaamista käytetään virheiden eliminointikeinona, Haikalan (2004, 287) mukaan testaus ilman spesifikaatiota on mahdotonta. Perinteisissä menetelmissä tehdään, tai ainakin pitäisi tehdä, toiminnallinen ja tekninen määrittely ensin ja sitten vasta testataan niiden pojalta.

Ketterät dokumentit pitäisi tehdä vain, kun niitä tarvitaan varmasti. Ei ole olemassa mitään tiettyä mallia, mitkä ovat ne dokumentit, jotka pitäisi olla olemassa. Vähimmäisdokumentointi pitäisi kuitenkin olla. Yritys käyttää ketteriä menetelmiä, mutta ei varsinaisesti jotain tiettyä menetelmää, vaan yrityksen menetelmä sisältää XP:tä ja Scrumia. XP:ssä lopullinen dokumentointi kirjoitetaan tuotetestamisvaiheessa. Kuten jo aiemmin todettiin, teimme toiminnallisen määrittelyn, kun järjestelmän kehitys oli loppuvaiheessa. Joten siltä osin dokumentin teossa noudatettiin XP:n käytäntöä.



Seuraavaksi vertaamme dokumenttia ketterän dokumentoinnin parhaisiin käytäntöihin (Ambler 2009a), jotka on esitetty luvussa 3.4.1.

AMBLERIN PARHAAT KÄYTÄNNÖT		KYLLÄ	EI
<b>1</b>	<b>KIRJOITTAMINEN</b>		
	Suosi suorittavia määrittelyjä staattisten dokumenttien sijaan	x	
	Dokumentoi pysyvää ei spekulatioita	x	
	Luo järjestelmädokumentointi		x
<b>2</b>	<b>PELKISTÄMINEN</b>		
	Pidä dokumentointi mahdollisimman yksinkertaisena		x
	Kirjoita harvat dokumentit välttämällä päällekkäisyyttä	x	
	Laita informaatio tarkoituksenmukaisimpaan paikkaan	x	
	Laita informaatio nähtäville julkisesti	x	
<b>3</b>	<b>RATKAISE MITÄ DOKUMENTOIDA</b>		
	Dokumentin tarkoitus	x	
	Keskity asiakkaan dokumentointitarpeet	x	
	Asiakas päättää dokumentin riittävydestä	x	
<b>4</b>	<b>RATKAISE MILLOIN DOKUMENTOIDA</b>		
	Iteroi, iteroi ja iteroi	x	
	Löydä parempi kommunikointitapa	x	
	Aloita mallista jota todella ylläpidät	x	
	Päivitä ainoastaan kun on pakko	x	
<b>5</b>	<b>YLEISTÄ</b>		
	Käsittele dokumentointia kuin vaatimuksia	x	
	Vaadi perustelut dokumentin vaatimuksille	x	
	Tunnista että tarvitset jotain dokumentointia	x	
	Hanki joku jolla on kirjoittajakokemusta	x	

Kuvio 21. Vertailun tuloksia

Ketterissä menetelmissä ei suositella pysyvien dokumenttien tekoa kehitystyön alkuvaiheessa, vaan suositellaan tehtäväksi kehityksen aikana muistiinpanoja. Eli kun jokaiseen iteraatioon kuuluu testaaminen, voidaan vaiheista tehdä muistiinpanoja esimerkiksi muistilapuille. Lopullinen dokumentointi kirjoitetaan vasta järjestelmän valmistumisvaiheessa, jolloin dokumentteihin ei tule vanhentunutta tietoa vaan **ainoastaan pysyvää tietoa**. Juuri näin toimittiin toiminnallisen dokumentin kanssa. Kirjoitimme dokumentin, kun järjestelmä oli jo miltei valmis.

Se, kuinka yksinkertaisena dokumentointi pidetään, määräytyy asiakkaan tarpeiden mukaan. Dokumentin olisi hyvä olla mahdollisimman lyhyt, ei liian yksityiskohtainen eikä tietoa tulisi toistaa. Koska tekemämme toiminnallinen määrit-

tely on melkein 40-sivuinen sekä yksityiskohtainen ja tarkka, voidaan todeta, että se ei aivan vastaa edellä mainittuja kriteereitä laajuutensa osalta. Toisaalta Ambler sanoo, että **asiakas päättää dokumentin riittävyys**. Koska toimeksiantaja halusi, että määrittely on mahdollisimman yksityiskohtainen ja tarkka, voidaan todeta, että dokumentti noudattaa ketterän dokumentin parhaita käytäntöjä.

**Kirjoittaminen siten, että niissä on vain vähän päällekkäisyyttä.** Koska tässä tarkastellaan vain toiminnallista määrittelydokumenttia eikä teknistä määrittelyä luotu, ei päällekkäisyyttä tältä osin voida sanoa olevan. Eli parhaita käytäntöjä noudatettiin.

**Informaatio on laitettu tarkoituksenmukaisimpaan paikkaan.** Informaatio on esitetty esimerkiksi käyttötapauksissa mahdollisimman tarkasti, koska toimeksiantaja niin halusi. Toisaalta toiminnallisesta määrittelypohjasta poistettiin muutamia kohtia, mitkä voidaan dokumentoida tekniseen määrittelyyn. Parhaita käytäntöjä siis noudatettiin. **Informaation julkistaminen.** Tarkoitus oli, että toiminnallinen määrittely on kaikkien työntekijöiden nähtävillä yrityksen sisäisessä verkossa. Tältä osin ketterä käytäntö toteutui.

Ketterän dokumentin yksi kriteeri on se, että **dokumentin tekemisen pitää olla perusteltua**. Miksi dokumentti tehdään? Mihin sitä tarvitaan? **Dokumentilla pitää olla selkeä tarve.** Tässä työssä käsiteltävä toiminnallinen määrittely tehtiin toimeksiantajan tarpeesta. Järjestelmä oli valmistumassa, eikä toiminnallista määrittelyä ollut uudesta versiosta olemassa. Koska kyseessä on yrityksen oma tuote, palvelu jota myydään, niin siltä osin on aivan **perusteltua, että tehdystä määrittelystä tuli yksityiskohtainen**, vaikka se ei siltä osin ehkä olekaan aivan ketterän dokumentin kriteerien mukainen. Mutta kuten jo useasti edellä on mainittu, dokumentti on **tehty toimeksiantajan tarpeesta ja toimeksiantaja nimellinen toive** oli, että dokumentti on mahdollisimman tarkka. Yrityksen työntekijäthän voivat esimerkiksi joskus vaihtua. Tässä kohdassa voidaan sanoa, että käytäntöjä on noudatettu.

Amblerin ketterän dokumentin yksi kriteereistä on, että **dokumentti on kirjoitettu sellaisella kielellä, jota dokumentin kohderyhmä ymmärtää**. Kirjoitimme määrittelyn asiakkaan toiveesta sellaisella kielellä, että kuka tahansa lukija sitä ymmärtää. Koska näin on, voidaan sanoa, että ketterän dokumentin kriteeri täyttyy.

**Dokumentit ovat riittävän tarkkoja, johdonmukaisia ja yksityiskohtaisia.** Niiden ei tarvitse olla täydellisiä, vaan riittävän hyviä. Dokumentista tuli tarkka, johdonmukainen ja yksityiskohtainen. Koska se oli asiakkaan toive, voidaan kriteerin todeta täyttyneen tältä osin. Dokumentti tuskin on täydellinen, mutta on kuitenkin riittävän hyvä.

Toista eli **iteroi**. Parhaat dokumentit on kirjoitettu pienissä paloissa. Se tarkoittaa, että kirjoitettavaa dokumenttia näytetään jollekin, saadaan siitä palautetta ja sitten työstetään lisää palautteen pohjalta. Tämän jälkeen jatketaan toistamalla, kunnes dokumentti on valmis. Dokumentointi kannattaa siis kirjoittaa iteratiivisesti, eli noudattaen ketterien menetelmien käytäntöä. Toiminnallinen dokumentti kirjoitettiin kuvatulla tavalla. Tältä osin dokumentti vastaa hyvin ketterän dokumentoinnin parhaita käytäntöjä. Amblerin parhaiden käytänteiden mukaisesti, kirjoitimme dokumentin iteratiivisesti ohjelmistokehityksen elinkaaren loppupuolella. Dokumenttia näytettiin palaverissa ohjelmoijalle ja toimitusjohtajalle, jolloin työstä saatiin palautetta ja työtä muokattiin palautteen mukaan. **Kommunkaation ensisijainen päämäärä on ymmärtäminen.** Dokumentti antaa mahdollisuuden esim. uudelle työntekijälle **perehtyä järjestelmän toimintaan** lukemalla dokumentti, jolloin ei tarvita toista työntekijää sitä erikseen **selittämään**. **Dokumentti luotiin järjestelmästä jota on kehitetty ja ylläpidetty.** Dokumentti on luotu sellaisessa elinkaaren vaiheessa ja niin tarkasti, että **jos se jää päivittämättä pienien järjestelmään mahdollisesti tulevien muutosten jälkeen, se ei haittaa paljon**. Korkealaatuisen toimivan ohjelmiston kehittäminen on tärkeää, mutta tärkeää on myös **varmistaa ohjelmiston ylläpidettävyyys, kehitys sekä käyttö- ja tukitoiminnot**.

**Pohjimmiltaan dokumentoinnin investoinnit ovat liiketoimintaan liittyviä päätöksiä.** Dokumentointia ei pidä luoda, koska prosessimallit vaativat niin, vaan se pitää luoda silloin, kun sidosryhmät niin sanovat. Luotu toiminnallinen määrittely on toimeksiantajalle perusteltua liiketoiminnan kannalta. Sille oli tarve eikä siihen jouduttu investoimaan paljon, koska dokumentti tehtiin opiskelijatyöharjoitteluna.

**Työskentelimme ikään kuin teknisinä kirjoittajina.** Kirjoitimme dokumenttia **parityönä**, mikä mahdollisti keskustelun ja dokumentin ääneen lukemisen ja kuulemisen. Se taas mahdollisti sen, että dokumentti tuli kirjoitettua mahdollisimman selkeällä kielellä.

Edellä on käyty läpi toiminnallinen määrittelydokumentti etsien siitä vastaavuuksia Amblerin listaamiin ketterän dokumentin parhaisiin käytäntöihin. Voidaan sanoa, että toiminnallinen määrittelydokumentti on tehty noudattaen ketterän dokumentin parhaita käytäntöjä. Kuuteentoista kohtaan kahdeksastatoista voidaan vastata kyllä, kun kysytään noudatettiinko käytäntöä.

Vertaamme vielä kirjoittamaamme dokumenttia Amblerin kriteereihin ketterästä dokumentista, jotka on esitetty luvussa 3.4.2. Jos verrataan toiminnallista määrittelydokumenttia ja sen tekemistä Amblerin ketterän asiakirjan kriteereihin, voidaan sanoa seuraavaa (kommentit dokumentista **lihavoidulla** fontilla):

1. Ketterät asiakirjat maksimoivat sidosryhmien sidotun pääoman tuottoprosentin. **Asiakirjan luominen maksoi toimeksiantajalle muutaman palaverin ajan.**
2. Ketterien dokumenttien hyödyt ovat suuremmat kuin investoinnit niiden luomiseen ja ylläpitoon. **Dokumentin hyöty on suurempi kuin investointi sen luomiseen ja ylläpitämiseen.**
3. Kun kirjoitat ketterää dokumenttia, muista periaatteena yksinkertaisuuden oletus: yksinkertaisin dokumentointi on riittävää. Luo siis mahdollisimman

yksinkertaista sisältöä. **Dokumentti on kirjoitettu sellaisella kielellä, että sitä ymmärtää maallikkokin.**

4. Ketterät dokumentit täyttävät tarkoituksensa, kun asiakirjat ovat yhtenäisiä. Ja jos et tiedä miksi olet luomassa dokumenttia, lopeta sen tekeminen ja ajattele uudelleen mitä teet. **Asiakirja luotiin toimeksiantajan tarpeesta.**
5. Dokumenteissa kuvataan asiat, jotka on hyvä tietää. Ei siis itsestään selviä asioita, vaan ainoastaan tärkeät tiedot. **Dokumentissa on kuvattu kaikki tiedot mitkä toimeksiantaja katsoi tarpeelliseksi. Ne kohdat, mitä ei toimeksiantajan mielestä tarvittu, jätettiin pois.**
6. Dokumentti on kirjoitettu tietyn kohderyhmän tarpeisiin. **Koska dokumentti kirjoitettiin sellaisella kielellä, että jokainen voi sitä ymmärtää, on sillä mahdollisuus palvella useampaa kohderyhmää.**
7. Dokumentit ovat riittävän tarkkoja, johdonmukaisia ja yksityiskohtaisia. Niiden ei tarvitse olla täydellisiä, vaan riittävän hyviä. **Dokumentti on tehty vastaamaan toimeksiantajan toiveita, jolloin voidaan sanoa, että dokumentti on kaikkea tätä.**

niin voidaan todeta, että kaikki edellä esitetyt seitsemän ketterän asiakirjan kriteeriä täyttyvät.

## 5 YHTEENVETO

Opinnäytetyöprosessi opetti paljon ohjelmistotuotannosta ja ketteristä menetelmistä, joista jälkimmäisestä emme tiedäneet ennestään mitään ennen syventävää työharjoittelua. Opinnäytetyöprosessi oli hidas, koska aihealueeseen ja materiaaliin perehtyminen vei valtavasti aikaa. Ketteriä menetelmiä käsittelevää kirjallisuutta oli hankala löytää suomeksi. Englanninkielistä materiaalia löytyi, kun sitä osasi etsiä oikeasta paikasta. Suomenkielisen materiaalin saatavuutta voi rajoittaa tulevaisuudessakin se, että ohjelmistotuotannon kieli on englanti.

Opinnäytetyön tekeminen saadusta aiheesta oli todella haastavaa, koska aihe oli ennestään tuntematon tekijöilleen. Olemme oppineet paljon ohjelmistotuotannosta, ohjelmistotuotannon prosessimalleista, ketteristä ohjelmistotuotannon menetelmistä ja dokumentoinnista. Prosessi kokonaisuudessaan kehitti asiantuntijuutta erityisesti ohjelmistotuotannon ja dokumentoinnin saralla.

Toimeksiantajan tavoitteena oli saada sisällönhallintajärjestelmän toiminnallinen määrittelydokumentti, joka olisi kirjoitettu mahdollisimman selkeästi ja yksityiskohtaisesti. Sen takia epäilimme, voiko luomamme toiminnallinen määrittelydokumentti olla ketterä.

Voimme kuitenkin ilolla todeta, että tämän opinnäytetyön tutkimuksen kohteena oleva toiminnallinen määrittelydokumentti vastaa Scott W. Amblerin listaamia parhaita käytäntöjä ketterän dokumentin kirjoittamisesta. Dokumentti täyttää myös Amblerin ketterän dokumentin kriteerit.

Toisaalta dokumentista tuli suhteellisen laaja ja siltä osin olisimme voineet päätyä myös sellaiseen johtopäätökseen, että dokumentti ei täysin vastaa ketterän dokumentin kriteereitä laajuutensa osalta. Tästä syntyykin jatkotutkimusaihe: kuinka dokumentin saisi tiiviimmäksi ja siten vastaamaan täydellisesti ketterän dokumentin käytäntöjä myös laajuutensa osalta?

## LÄHTEET

- Abrahamsson, Pekka – Salo, Outi – Ronkainen, Jussi – Warsta, Juhani 2002. Agile software development methods: review and analysis. VTT:n julkaisu 478. Osoitteessa <http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf>. 15.10.2009.
- Abrahamsson, Pekka – Salo, Outi 2006. Ketteriin menetelmiin keskittyvä XP2006-konferenssi oli menestys! Systeemityö 3 / 2006, 12–13. Osoitteessa <http://www.pcuf.fi/sytyke/lehti/kirj/st20063/ST063-12A.pdf>. 10.11.2009.
- 2007. Ketterät tuulet ohjelmistotuotannossa. Systeemityö 4 / 2007, 4–6. Osoitteessa <http://www.pcuf.fi/sytyke/lehti/kirj/st20074/ST074-04A.pdf>. 12.10.2009.
- Agile Alliance 2009a. Manifesto for Agile Software Development. About the Manifesto. Osoitteessa <http://www.agilemanifesto.org/history.html>. 13.10.2009.
- 2009b. Manifesto for Agile Software Development. Osoitteessa <http://www.agilemanifesto.org/>. 13.10.2009.
- 2009c. Manifesto for Agile Software Development. Twelve Principles of Agile Software. Osoitteessa <http://www.agilemanifesto.org/principles.html>. 13.10.2009.
- Ambler, Scott W. 2009a. Best Practices for Agile/Lean Documentation. Osoitteessa <http://www.agilemodeling.com/essays/agileDocumentationBestPractices.htm>. 16.12.2009.
- 2009b. Agile/Lean Documentation: Strategies for Agile Software Development. Osoitteessa <http://www.agilemodeling.com/essays/agileDocumentation.htm#WhenIsADocumentAgile>. 7.11.2009.
- Beck, Kent 1999a. Extreme Programming Explained: embrace change. Addison-Wesley. Osoitteessa <http://www.inf.unioeste.br/~victor/processoIII/Methodologias/Extreme%20Programming%20Explained%20-%20Kent%20Beck.pdf>. 9.11.2009.
- 1999b. Embracing Change with Extreme Programming. Computer 10 / 1999, 70–77.

- Boehm, Barry 2002. Get Ready for Agile Methods, with Care. Computer 2 / 2002, 64–69. Osoitteessa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=976920&isnumber=21069?tag=1>. 7.11.2009.
- Chau, Thomas – Maurer, Frank – Melnik, Grigori 2003. Knowledge Sharing: Agile Methods vs. Tayloristic Methods. Osoitteessa <http://ase.cpsc.ucalgary.ca/uploads/Publications/ChauMaurerMelnik.pdf>. 24.11.2009.
- Controlchaos 2009. Scrum. Osoitteessa <http://www.controlchaos.com/about/>. 10.11.2009.
- Eriksson, Hans-Erik – Penker, Magnus 2000. UML. Edita, Jyväskylä.
- Extreme Programming 2009. Osoitteessa <http://www.extremeprogramming.org/values.html>. 28.9.2009.
- Fowler, Martin – Scott, Kendall 2004. UML. 2. painos. Docendo, Jyväskylä.
- Haikala, Ilkka – Märijärvi, Jukka 2004. Ohjelmistotuotanto. 10. painos. Talentum, Helsinki.
- 2009a. Toiminnallinen määrittely. Osoitteessa [http://www.cs.tut.fi/ohj/dokumenttipohjat/pohjat/maarittely/sovellusohje\\_hytt\\_drmaarittely.pdf](http://www.cs.tut.fi/ohj/dokumenttipohjat/pohjat/maarittely/sovellusohje_hytt_drmaarittely.pdf). 22.9.2009.
- 2009b. Luentokalvo. Osoitteessa <http://www.cs.tut.fi/~otupk/luennot/kalvot/speksaus.pdf>. 13.11.2009.
- Huttunen, Janne 2006. Ketterän ohjelmistokehitysmenetelmän määrittely, vertailu ja käyttäjäkysely. Diplomityö, teknillinen korkeakoulu. Osoitteessa <http://lib.tkk.fi/Dipl/2007/urn007665.pdf>. 5.11.2009.
- Ketterät käytännöt 2009a. Ketteryys. Vaatimuksia vai lisäarvoa? Onko projekti vaatimusten täyttämistä vai lisäarvon tuottamista? Osoitteessa <http://www.ketteratkaytannot.fi/fi-FI/Ketteryys/VaatimuksiaVaiLisaaarvoa/>. 22.9.2009.
- 2009b. Menetelmät. Osoitteessa <http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/>. 4.11.2009.
- 2009c. Menetelmät. XP. Osoitteessa <http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/XP/>. 9.11.2009.
- 2009d. Menetelmät. Scrum. Osoitteessa <http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/Scrum/>. 10.11.2009.



- 2009e. Menetelmät. Scrum. Arvot. Osoitteessa  
<http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/Scrum/Arvot/>  
 10.11.2009.
- 2009f. Menetelmät. Scrum. Säännöt. Osoitteessa  
<http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/Scrum/Säännöt/>  
 10.11.2009.
- Klemetti, Pasi 2007. VTT uudistaa ohjelmistotutkimusta. Osoitteessa  
<http://www.hightechforum.fi/index.cfm?j=695473>. 24.9.2009.
- Koch, Alan 2004. Agile Software Development: Evaluating the Methods for Your Organization. Artech House.
- Lindström, Jukka 2006. Scrumia käytettäessä kaikki tietävät, miten projekti etenee. Osoitteessa <http://www.reaktor.fi/web/fi/teknologia-ja-tutkimus/scrum>. 25.10.2006.
- McConnell, Steve 2002. Ohjelmistotuotannon hallinta. Edita, Helsinki.
- Nevalainen, Risto 2008. Ketterästi tehtyä ja laadukasta – onnistuuko? Systeemityö 4 / 2008, 18–20.
- Nykänen, Pirkko 2009. Ketterä (agile) tietojärjestelmien suunnittelu / ohjelmistotuotanto. Osoitteessa  
[http://www.cs.uta.fi/tjsum/TJSUM\\_08042009\\_PirkkoNykanen.pdf](http://www.cs.uta.fi/tjsum/TJSUM_08042009_PirkkoNykanen.pdf).  
 22.9.2009.
- Nykopp, Sebastian – Koskela, Lasse 2006. Projektin laadun parantaminen ketterillä menetelmillä. Systeemityö 3 / 2006, 6–8. Osoitteessa  
<http://www.pcuf.fi/sytyke/lehti/kirj/st20063/ST063-06A.pdf>.  
 21.9.2009.
- Paetsch, Frauke – Eberlein, Armin – Maurer, Frank 2003. Requirements Engineering and Agile Software Development. Osoitteessa  
<http://ase.cpsc.ucalgary.ca/uploads/Publications/PaetschEberleinMaurer.pdf>. 9.10.2009.
- Pohjonen, Risto 2002. Tietojärjestelmien kehittäminen. Docendo, Jyväskylä.
- Raussi, Tarja – Virtanen, Pentti 2009. Vaatimusmäärittelyä ketterästi vai perinteisesti? Systeemityö 2 / 2009, 16–19.
- Schuh, Peter 2004. Integrating Agile Development in the Real World. Charles River Media. Osoitteessa  
<http://site.ebrary.com/lib/ramklibrary/docDetail.action?docID=10078511&p00=extreme%20programming>. 7.11.2009.

Sommerville, Ian 2007. Software engineering. Eighth edition. Addison-Wesley.

Tepsa, Tauno 2006. Prosessimallit. Osoitteessa  
<http://ta.ramk.fi/~tauno.tepsa/TJS/Luennot%20wk%2047.pdf>.  
24.11.2006.

van Vliet, Hans 2008. Software Engineering. Principles and Practice. John Wiley & Sons.

Virtanen, Pentti 2003. Ketteryydellä tuottavuutta ja laatua. *Systeemityö* 1 / 2006, 22–23.

Wikipedia 2009a. Ohjelmistotuotanto. Vesiputousmalli. Osoitteessa  
<http://fi.wikipedia.org/wiki/Vesiputousmalli>. 13.10 2009.

– 2009b. Dokumentti. Osoitteessa <http://fi.wikipedia.org/wiki/Dokumentti>.  
25.9.2009.

– 2009c. Sisällönhallintajärjestelmä. Osoitteessa  
<http://fi.wikipedia.org/wiki/Sis%C3%A4ll%C3%B6nhallintaj%C3%A4rjestelm%C3%A4>. 25.9.2009.

## LIITTEET

Agile manifesti	Liite 1
Sivurakenteen käyttötapaus	Liite 2
Osa Editorin käyttötapauksista	Liite 3
Kuvapankin ja Tiedostopankin Lisää-toiminnot	Liite 4
Uutisryhmän etusivu	Liite 5

## Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.  
Through this work we have come to value:

Individuals and interactions over processes and tools  
Working software over comprehensive documentation  
Customer collaboration over contract negotiation  
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.



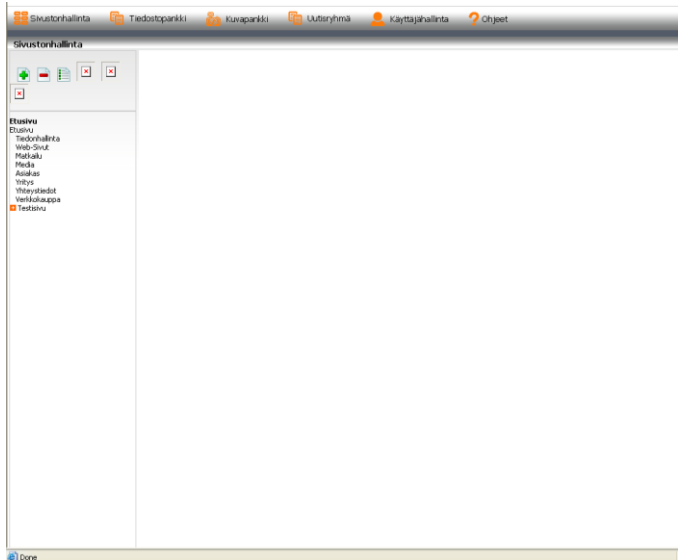
Kent Beck  
Mike Beedle  
Arie van Bennekum  
Alistair Cockburn  
Ward Cunningham  
Martin Fowler








James Grenning  
Jim Highsmith  
Andrew Hunt  
Ron Jeffries  
Jon Kern  
Brian Marick

Robert C. Martin  
Steve Mellor  
Ken Schwaber  
Jeff Sutherland  
Dave Thomas

## Sivurakenteen käyttötapaus

## Liite 2

Tunniste	Sivustonhallinta – Sivurakenne
Kuvaus	Käyttäjä voi havainnollisesti tarkastella omien sivujensa sijoittumista sivustonhallintamoduulin vasemmassa laidassa näkyvästä sivurakenteesta.
Alkuehto	Käyttäjä on onnistuneesti kirjautunut sisään järjestelmään. Käyttäjällä on riittävät oikeudet Sivustonhallinta-moduuliin ja hänelle on luotuna tietokantaan vähintään yksi sivu, muuten sivurakenne on tyhjä.
Normaali tapahtumien kulku	Sivustonhallinnan aloitusnäkymässä sivurakenne näkyy vasemmassa laidassa puurakenteena. Käyttäjä painaa  -painiketta sen sivun edessä jota haluaa tarkastella, jolloin näkyviin avautuu halutun sivun alasisivut. Painamalla  -painiketta käyttäjä sulkee kyseisen sivun alasisivut näkyvistä.
Vaihtoehtoinen tapahtumien kulku	Rajatut käyttöoikeudet: → Sivut, joihin ei ole käyttöoikeuksia näkyvät harmaina.
Loppuehto	Käyttäjä pystyy selailemaan sivurakennetta onnistuneesti.
Erikoisvaatimukset	Istuntotunnus on voimassa, käyttäjätunnus on oikea ja salasana on oikea sekä käyttöoikeudet moduuliin.
Käyttäjät	Pääkäyttäjät ja käyttäjät
Järjestelmäversio	3.0
Näyttömalli	<p>Sivurakenne:</p> 

Tunniste	Sivustonhallinta – Editori
Kuvaus	Editorilla luodaan sisältö www-sivulle. Editorilla muokataan olemassa olevia sivuja ja luodaan uusia sivuja.
Alkuehto	Käyttäjä on onnistuneesti kirjautunut sisään järjestelmään. Käyttäjällä on riittävät oikeudet Sivustonhallinta-moduuliin.
Normaali tapahtumien kulku	<p>Sivurakenteesta valitaan sivu, jota halutaan muokata. Valittu sivu avautuu keskelle työtilanäkymään ja klikkaamalla hiirellä jotain kohtaa sivusta avautuu näkymään editori. Editorissa voidaan muokata olemassa olevia sivuja ja luoda uusia sivuja. Editorin muokkaustoiminnot näkyvät Windows-maailmasta tuttuina toimintakuvakkeina.</p> <p>Kun käyttäjä esimerkiksi painaa  -kuvaketta, avautuu pop-up -ikkuna, joka kysyy: ”Haluatko varmasti tyhjentää kaiken sisällön?” Valitsemalla <b>Ok</b> sivun sisältö poistetaan ja sivulle voidaan kirjoittaa uutta sisältöä tyhjässä työtilassa. <b>Cancel</b>-painiketta painamalla tapahtuma peruuntuu.</p> <p>Kun sivulle on luotu uutta sisältöä tai muokattu vanhaa, niin  -kuvaketta painamalla uusi sisältö tallentuu tiedostoon ja vanha poistuu. Jos taas uutta sisältöä ei luoda jo olemassa olevaan sivuun vaan jätetään se tyhjäksi, niin vanha sisältö ei poistu tiedostosta vaikka painettaisiin  -kuvaketta.</p> <p>Editorin toimintakuvakkeet:</p> <p><b>Tallenna</b>  Tallentaa muokatun sivun tietokantaan.</p> <p><b>Uusi tiedosto</b>  Poistaa sivun sisällön.</p> <p><b>Lihavoitu</b> <b>B</b> Lihavoi valitun tekstin.</p> <p><b>Kursivoitu</b> <b>I</b> Kursivoi valitun tekstin.</p> <p><b>Alleviivattu</b> <b>U</b> Alleviivaa valitun tekstin.</p> <p><b>Yliviivattu</b> <b>ABC</b> Yliviivaa valitun tekstin.</p> <p><b>Tasaa vasemmat reunat</b>  Tasaa tekstin vasemmat reunat.</p> <p><b>Keskitä</b>  Keskitää tekstin.</p>

## Kuvapankin ja Tiedostopankin Lisää-toiminnot

## Liite 4

Sivustonhallinta Tiedostopankki Kuvapankki Uutisryhmä Käyttäjähallinta Ohjeet

### Kuvapankki

**Lisää kuva**




Kuvakoon pienennys  
160 pikseliä - oikealle kuva-alueeseen

Kuvaus

Hakusanat

Tekijänoikeus

**Esikatselu**

Nimi	Koko	Lisätty
IMG_0351.JPG	159 KB	2009-04-03 13:37:01
Untitled-1.gif	31 KB	2009-02-13 16:06:26
luuri.gif	23 KB	2009-02-13 15:40:40

Sivustonhallinta Tiedostopankki Kuvapankki Uutisryhmä Käyttäjähallinta Ohjeet

### Tiedostopankki

**Tiedostopankki**

**Lisää tiedosto**

Kuvaus

Hakusanat

**Tiedosto**

Banneri-pallari...

agreement.txt

testiäö.doc

DSC\_0172[1].jpg

Muoto	Koko	Lisätty
	55 KB	2009-03-20 10:22:19
	3 KB	2009-03-20 10:22:11
	23 KB	2009-03-09 13:37:49
	3959 KB	2009-02-05 11:35:11

## Uutisryhmän etusivu

## Liite 5

Uutisryhmät			
			
Luotu	Otsikko	Tila	Ryhmä
2009-03-20 11:15	otsikko	Näkyvillä	Pääkäyttäjät
2009-03-19 13:30	testi	Näkyvillä	Pääkäyttäjät
2009-02-12 15:39	safddsa	Näkyvillä	Pääkäyttäjät
2009-02-05 12:08	otsikkouutinen	Näkyvillä	Pääkäyttäjät
2009-02-04 14:24	Juhan testi	Näkyvillä	Pääkäyttäjät