Opinnäytetyö (AMK)

Tietojenkäsittely

Tietojärjestelmät

2011

Timo Juhala

TYÖNSEURANTA- JA TIEDOTUSJÄRJESTELMÄ KETTERÄSTI

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

Turun ammattikorkeakoulu

Tietojenkäsittely | Tietojärjestelmät

15.11.2011 | 83

Anne Jumppanen

Timo Juhala

TYÖNSEURANTA- JA TIEDOTUSJÄRJESTELMÄ KETTERÄSTI

Opinnäytetyön aiheena oli rakentaa toimeksiantona kohdeyritykselle selaimella toimiva web-järjestelmä työnseurantaa ja tiedottamista varten. Tavoitteena oli suorittaa kyseinen projekti käyttämällä jotain soveltuvaksi katsottua ketterää menetelmää ja tutkia, kuinka se tapahtuu ja mitä hyötyjä ja haittoja siitä seuraa.

Tutkimuksen teoreettisessa osuudessa käsiteltiin ketterien menetelmien yleisiä piirteitä, joitakin yksittäisiä ketteriä menetelmiä ja tarkemmin DSDM Atern –menetelmää. Ketterien menetelmien yleisiä piirteitä ovat muun muassa iteratiivinen kehitys, painottuminen ihmisten väliseen kanssakäymiseen ja tuotoksien nopea toimittaminen asiakkaille. Atern erottuu muista menetelmistä pääasiassa siksi, että sen perusajatuksena on varmistaa projektin valmistuminen aikataulussa, vaikka se vaatisi vähemmän tärkeiden ominaisuuksien karsimista.

Empiirisessä osuudessa Atern-menetelmää sovellettiin tämän ohjelmistokehitysprojektin läpiviemiseen. Joitakin Aternin ominaisuuksia karsittiin projektiryhmän pienen koon johdosta. Kolme kuukautta kestänyt projekti valmistui aikataulussa ja syntynyt ratkaisu sisälsi toimeksiantajan ennen projektia ja sen aikana tärkeimmiksi kirjaamat ominaisuudet. Osa vähemmän tärkeistä ominaisuuksista jäi sallitusti puuttumaan. Empiirisessä osuudessa esitellään myös projektin aikana rakennetun järjestelmän tärkeimpiä osioita.

Tutkimuksen perusteella karsittu DSDM Atern soveltuu hyvin pieniin ohjelmistonkehitysprojekteihin ja johtaa käyttökelpoiseen lopputulokseen.

ASIASANAT:

Ajax-ohjelmointi, ohjelmistokehitys, PHP, tiedonhallintajärjestelmät, verkko-ohjelmointi

BACHELOR'S THESIS | ABSTRACT

Turku University of Applied Sciences

Business Information Technology | Information Systems

15.11.2011 | 83

Anne Jumppanen

Timo Juhala

BUILDING A SYSTEM FOR MONITORING WORK PROGRESS AND NOTIFYING IN AN AGILE WAY

The purpose of this thesis was to construct a web system for monitoring work progress and notifying for the target company. The aim was to carry out the project by using some agile method deemed fit for the purpose and to study how that is done and what are the pros and cons that arise.

Covered in the theoretical part of the thesis are the general characteristics of agile methods, accounts of a few different agile methods and a more specific report on the DSDM Atern method. In general agile methods focus on iterative development, collaboration between people and fast delivery of products to customers. Atern stands out mainly because its basic principle is that a project has to be completed by the given deadline even if some of the less important features have to be stripped to achieve that result.

In the empirical part, the Atern method – stripped of some of its features that were unsuitable for this project – was used to carry out this software development project. The project, which lasted three months, was finished in schedule and the solution born of it included the most important features which were coined by the ordering party before and during the project. Some of the less important features were not delivered, which was allowed by the method. Some of the most important functionalities of the system created are also showcased in this part of the thesis.

The result of this study was that DSDM Atern can quite well be applied at least to small software development projects and using it will lead to a result that is at least somewhat usable.

KEYWORDS:

Ajax programming, data management systems, PHP, software engineering, web programming

SISÄLTÖ

KUVAT

KUVIOT

TAULUKOT

SANASTO

Integraatio	Yhtenäisen kokonaisuuden muodostaminen; jonkin liittäminen osaksi jotakin (MOT 2011a).
Iteraatio	Saman toimituksen peräkkäiseen toistamiseen perustuva menetelmä, jossa tulos tarkentuu jokaisella toistokerralla (MOT 2011b).
Konfiguraatio	Kokoonpano (MOT 2011b).
RUP	IBM Rational Unified Process, ohjelmistonkehitysmenetelmä, jossa järjestelmä kehitetään nollapisteestä valmiiksi neljän peräkkäisen vaiheen aikana. (Aked 2003.)
Skeema	Kaava, malli, suunnitelma (MOT 2011a). Tietokannan rakenne (Roberts 1998).

1 JOHDANTO

Opinnäytetyöni tavoitteena oli rakentaa työnseuranta- ja tiedotusjärjestelmä toimeksiantona eräälle yritykselle, jonka nimeä ja toimialaa ei haluta yleiseen tietoon; sitä kutsutaan jatkossa nimellä Yritys Y, joka toimii toimialalla T. Työ toteutettiin web-järjestelmänä, joka toimii tavallisella verkkoselaimella. Järjestelmän tarkoituksena oli mahdollistaa sekä työn edistymisen ja käytettyjen työtuntien raportointi ja seuranta yrityksen työnjohdolle että täsmällinen tiedottaminen niin yrityksen sisäisesti kuin asiakkaidenkin kanssa.

Tavoitteena oli myös tutustua ketteriin menetelmiin ja valita niistä joku ohjaamaan projektin läpiviemistä sekä tutkia, kuinka menetelmä soveltuu tämänkaltaiseen pienimuotoiseen ohjelmistoprojektiin. Menetelmien yleispiirteitä ja joitakin ketteriä menetelmiä, joita tutkin, esitellään luvussa 2, ja menetelmää, jonka valitsin, eli DSDM Atern –menetelmää, esitellään ripauksen tarkemmin luvussa 3. Menetelmien laajuuden vuoksi vuoksi yritin keskittyä huomionarvoisimpina pitämiini seikkoihin. Lähteinä teoriaosuudelle käytin mahdollisuuksien mukaan aina menetelmien kehittäjien omia tuotoksia, koska koin heidän itse olevan parhaita kertomaan omien menetelmiensä erityispiirteistä; lisäksi AMK:n kirjaston ja Ebraryn tarjonta oli paikoin yllättävän suppeaa aiheesta kirjoitetun kirjallisuuden määrän huomioiden.

Toimialan T piirteet yleisesti sekä toimeksiantajan pyynnöt erityisesti vaikuttivat ratkaisevasti siihen, millaisiin ongelmiin järjestelmän oli tarkoitus toimia ratkaisuna ja miten se teknisesti tulisi toteuttaa. Näitä asioita on esitelty luvussa 4.

Luvussa 5 valotetaan hieman sitä, miten projekti eteni ja sitä, millaisia järjestelmän tärkeimmistä osioista lopulta tuli ja lopuksi luvussa 6 pohditaan työn laatua sekä tutkimuksen yleistä hyödynnettävyyttä.

2 KETTERÄT MENETELMÄT

Ketterillä menetelmillä tarkoitetaan iteratiivisten ja ihmisten väliseen yhteistyöhön mm. vaatimusmäärittelyssä ja ongelmanratkaisussa perustuvia sovelluskehitysmenetelmien joukkoa. Niissä suositaan toisaalta kurinalaista projektinhallintaa usein tapahtuvine tilakatselmuksineen ja toisaalta taas johtamistapaa, joka rohkaisee itseohjautuvuuteen, vastuunottoon ja ihmisten väliseen yhteistyöhön. Päämääränä näillä menetelmillä on korkealaatuisten, asiakkaiden tarpeisiin täsmällisesti vastaavien ohjelmistojen kehittäminen nopeasti. (cPrime 2011.)

Jokainen ketteristä menetelmistä on lähestymistavaltaan yksilöllinen, mutta ne kaikki jakavat ns. ketterän ohjelmistokehityksen julistuksessa, johon palataan myöhemmin, määritellyt visiot ja ydinarvot, ja kaikissa hyödynnetään iteraation tuottamaa jatkuvaa palautetta ohjelmistojen jalostamisessa, toisin sanoen koko ajan jatkuvan suunnittelun, toteutuksen ja testauksen idea on kaikille yhteinen. Lisäksi menetelmissä vaalitaan ihmisten välistä tehokasta yhteistyötä ja nopeaa päätöksentekoa. Ketterät menetelmät ovat myös kevyitä erityisesti perinteisempiin ohjelmistokehitysmalleihin, esim. vesiputousmalliin, verrattuna, eivätkä itse menetelmätkään ole mitään ylhäältä saneltua pyhää tekstiä, vaan niiden luonteeseen kuuluu, että niitä käytetään soveltaen. (VersionOne 2011a.)

Ketterällä kehityksellä tarkoitetaan sellaista kehitysprosessia, joka seurailee edellä mainitun ketterän ohjelmistokehityksen julistuksen perusprinsiippejä:

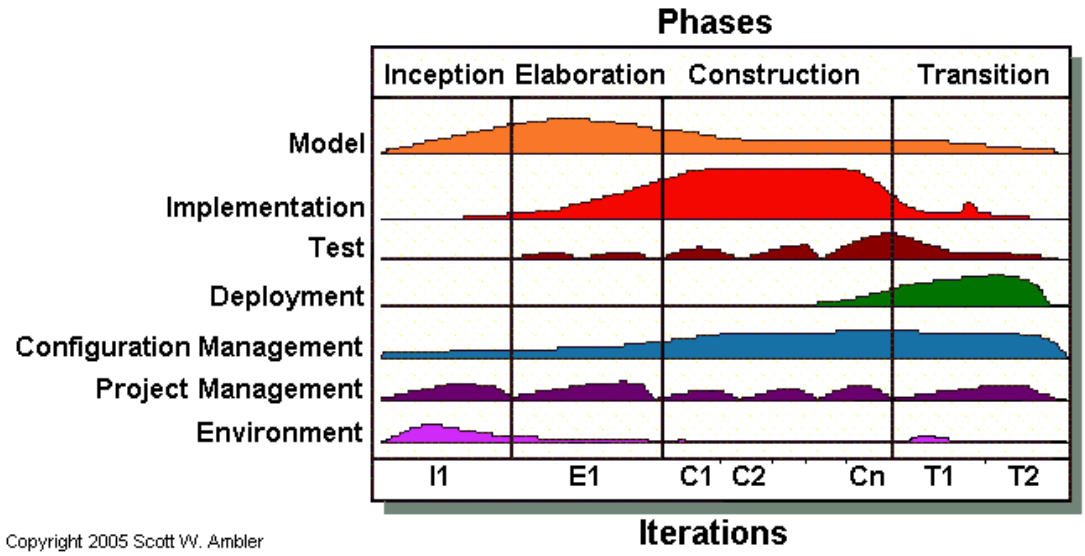
> Löydämme parempia tapoja tehdä ohjelmistokehitystä, kun teemme sitä itse ja autamme muita siinä. Kokemuksemme perusteella arvostamme:
>
> **Yksilöitä ja kanssakäymistä** enemmän kuin menetelmiä ja työkaluja
>
> **Toimivaa ohjelmistoa** enemmän kuin kattavaa dokumentaatiota
>
> **Asiakasyhteistyötä** enemmän kuin sopimusneuvotteluja
>
> **Vastaamista muutokseen** enemmän kuin pitäytymistä suunnitelmassa

> Jälkimmäisilläkin asioilla on arvoa, mutta arvostamme ensiksi mainittuja enemmän.
>
> (Beck ym. 2001.)

Julistuksen on kirjoittanut seitsemäntoistahenkinen ryhmä ohjelmistoalan (ilmeisesti) johtavia hahmoja ja se heijastaa heidän kokemuksiaan siitä, mitkä lähestymistavat toimivat ohjelmistokehityksessä ja mitkä eivät; ketterän ajatusmaailman mukaan asiakkaiden vaatimukset täytetään paremmin ja nopeammin, kun luotetaan työhön sitoutuneiden ihmisten yhteistyökykyyn enemmän kuin mihinkään tiettyyn prosessiin. ”Ihmiset voivat onnistua ilman muodollista prosessia, mutta mikään prosessi ei voi onnistua ilman ihmisiä.” (cPrime 2011.)

2.1 Agile Unified Process

Agile Unified Process (AUP) on yksinkertaistettu versio IBM:n kehittämästä Rational Unified Processista (RUP), jonka ideana on tarjota helppotajuinen lähestymistapa ohjelmistokehittämiseen käyttäen ketteriä tekniikoita ja käsitteitä ja pysyen silti uskollisena RUPin perusajatuksille. (Ambler 2009.)



Kuvio 1. AUPin elämänkaari (Ambler 2009).

AUPin elinkaari (kuvio 1) muistuttaa RUPin vastaavaa, mutta tehtävät (engl. ”disciplines”) ovat muuttuneet. Osa niistä on uusia ja osa syntynyt vanhoja yhdistämällä, esimerkiksi RUPin vaatimusmäärittely- ja analysointi ja määrittelytehtävät on yhdistetty AUPissa yhdeksi mallintamistehtäväksi (model), minkä vuoksi järjestelmän suunnitteluun ja määrittelyyn käytetään vähemmän aikaa; tavoitteena AUPissa onkin tehdä juuri ja juuri riittävän hyvät määrittelyt. Ajassa edetään kuitenkin edelleen vaiheittain alkuvaiheesta (inception) suunnittelun ja rakentamisen kautta palvelun käyttöönottoon (transition). (Ambler 2009.)

AUP on suuressa mittakaavassa sarjamuotoinen, tarkoittaen sitä, että vaiheesta toiseen edetään kuin jatkokertomuksessa ikään; alkuvaiheessa mietitään projektin laajuutta ja rahoitusta, minkä jälkeen ohjelmistoa suunnitellaan kovasti, jotta sitä voidaan lähteä rakentamaan inkrementaalisesti, sidosryhmien tärkeimmät vaatimukset täyttäen, ja lopulta ottaa käyttöön tuotantoympäristössä. (Ambler 2009.)

Pienessä mittakaavassa AUP on iteratiivinen, tarkoittaen sitä, että elinkaaren kuutta tehtävää suoritetaan iteratiivisesti; eri tehtävät vain painottuvat eri lailla eri vaiheissa. Huomionarvoista on, että ketterien menetelmien yleisen tavan mukaisesti suunnittelua ja testausta tehdään läpi koko projektin elinkaaren. RUPista poiketen AUP ottaa kantaa myös projektin tuotosten, esimerkiksi eri ohjelmaversioiden, hallintaan ja seurantaan, yleiseen projektijohtamiseen ja projektitiimin työympäristöön, kuten koulutuksen saatavuuteen, sopiviin työkaluihin. Myös järjestelmän eri versiot julkaistaan iteratiivisesti; kehitysjulkaisuja kehitysympäristöön tulee usein, ja tuotantokäyttöön päästetään aina silloin tällöin versio, joka on osoittautunut toimivaksi laaduntarkkailu- eli QA-ympäristössä. (Ambler 2009.)

AUPilla on myös omat perusperiaatteensa, ja ne ovat hyvin ketterän ohjelmistokehityksen mukaisia: ihmisten osaamiseen tulee luottaa, pidetään dokumentit yksinkertaisina, keskitytään siihen, mikä on tärkeää, työkaluihin ei oteta mallissa kantaa ja mallia saa ja tulee muuttaa omiin tarkoituksiinsa sopivaksi. (Ambler 2009.)

2.2 Crystal

Crystalissa ei ole kyse yhdestä tietystä menetelmästä, vaan kokonaisesta menetelmäperheestä, jonka jäseniä ovat esim. Crystal Clear, Crystal Yellow ja Crystal Orange (VersionOne 2011b). Eri menetelmät on tarkoitettu erilaisiin projekteihin ja sopivan valinta riippuu muun muassa kehitystiimin koosta, järjestelmän kriittisyydestä ja projektin päämääristä (VersionOne 2011b). Kaikista perheenjäsenistä ei kuitenkaan ole saatavilla perinpohjaisia kuvauksia, mutta Crystal Clear on julkaistu kirjamuodossa, minkä lisäksi internetissä on wiki, josta löytyy lisämateriaalia ja keskusteluja aiheesta (Fowler 2005). Kyseinen wiki ei tosin kirjoitushetkellä ollut toiminnassa. Crystal Clearissa annetaan myös ohjeita menetelmän soveltamiseen ja räätälöimiseen omaan organisaatioon sopivaksi, minkä lisäksi siitä voi hyötyä myös ottamatta menetelmää kokonaisuudessaan käyttöön lainaamalla siitä hyödyllisiä tekniikoita niihin menetelmiin, joita käyttää (Rusk 2006).

Eroistaan huolimatta näillä eri menetelmillä on yhteisiä ominaisuuksia ja ne kaikki jakavat samat päämäärät. Päämääriä on kolme, ja ne ovat turvallisuus, tehokkuus ja viihtyisyys. Turvallisuudella tarkoitetaan tässä yhteydessä projektin saattamista onnistuneesti eli turvallisesti päätökseen, kun taas viihtyisyydellä tarkoitetaan menetelmän olevan sellainen tai olevan muokattavissa sellaiseksi, että kehittäjät pystyvät elämään sen kanssa. (Fowler 2005.)

Yhteiset ominaisuudet ovat usein toistuva jakelu, arvioiva kehittyminen, läheinen viestintä, henkilökohtainen turvallisuus, fokus, helppo pääsy asiantuntijoihin ja automatisoiduilla testeillä ja konfiguraationhallinnalla varustettu usein integroitava tekninen ympäristö (Cockburn 2005, 19-37). Näistä tärkeimpinä pidetään kolmea ensimmäistä. (Cockburn 2005, 18). Crystal Clear, joka on suunnattu pienille tiimeille, on tehostanut läheisen viestinnän tehokkaammaksi osmoottiseksi viestinnäksi, mutta muuten Crystal Clearissa kuvattujen ominaisuuksien luvataan toimivan projektissa kuin projektissa (Cockburn 2005, 18). Seuraavassa tarkastellaan hieman näitä tärkeimpiä ominaisuuksia Crystal Clearin näkökulmasta.

2.2.1 Usein toistuva jakelu

Jokaisen ohjelmistoprojektin tärkein ominaisuus Crystal Clearin mukaan on se, että järjestelmän todellisille käyttäjille toimitetaan toimivaa ja testattua koodia aina muutaman kuukauden välein, mielellään vähintään kahdesti puolivuosittain tai vielä useamminkin, web-järjestelmien tapauksessa jopa viikoittain. Tästä seuraa lukuisia hyötyjä: projektin rahoittajat pystyvät seuraamaan kehitystiimin etenemistä, käyttäjät pääsevät arvioimaan, ovatko he pyytäneet oikeasti tarvitsemiansa ominaisuuksia ja antamaan palautetta, kehittäjät pysyvät keskittyneinä ja tiimi pääsee testaamaan kehitys- ja jakeluprosessejaan ja sen jäsenten itsetunto kasvaa onnistumisten myötä. (Cockburn 2005, 19.)

Ovelasti Cockburn varoittaa myös liian usein tapahtuvasta jakelusta ja vihjaa, että loppukäyttäjät saattavat närkästyä jatkuviin päivityksiin. Näissä tapauksissa neuvotaan etsimään ”ystävällinen” käyttäjä, jota ei joko kohteliaisuudesta tai uteliaisuudesta haittaa testailla järjestelmää, ja toimittaa tälle testiversioita, jotta tiimi pääsisi harjoittelemaan järjestelmän jakelua ja saisi palautetta edes yhdeltä käyttäjältä. (Cockburn 2005, 19.)

Kirjassa painotetaan lisäksi jakelun ja iteraation eroa; iteraatiot kehotetaan pitämään ajallisesti lukittuina, esimerkiksi jokainen iteraatio on kaksi viikkoa pitkä, minkä jälkeen tiimi toimittaa aikaansaannoksensa, on se sitten mitä hyvänsä. Näin toimimalla tiimi oppii, mihin se yhden iteraation aikana pystyy, mikä on hyödyllistä projektin suunnittelun kannalta, ja se mahdollistaa myös projektin etenemisvauhdin mittaamisen. Se, laitetaanko iteraation tuotokset jakeluun vai ei, jätetään tiimin ja asiakkaan harkintaan. Toisinaan on käytännöllistä tehdä jakelu joka iteraation jälkeen, toisinaan muutaman iteraation välein ja toisinaan tiettyinä päivämäärinä. (Cockburn 2005, 20-21.)

2.2.2 Arvioiva kehittyminen

Arvoivaksi kehittymiseksi kutsutaan sitä, että projektitiimi aina silloin tällöin – tunti parin viikon välein tai kerran kuukaudessa riittää – kokoontuu yhteen ja keskustelee työstään ja siitä, mikä projektissa toimii ja mikä ei. Tarkoituksena

on kehittää itse työskentelytapoja niin, että projekti kulkisi jouhevammin tulevaisuudessa ja ohjelmisto saataisiin asiakkaan käyttöön aikataulussa. (Cockburn 2005, 22-23.)

Tätä pidetään Crystal Clearissa erityisen tärkeänä siksi, että sen mukaan todella usein ohjelmistoprojektit alkavat surkeasti, jopa katastrofaalisesti – toisaalta juuri sen kaltaiset ongelmatilanteet ovat niitä, joista tiimi voi ottaa opikseen pohtimalla, mikä meni pieleen ja miksi, ja miten tulevaisuudessa vastaavilta tilanteilta vältytään. Ja lähes varmuudella joka tapauksessa projektin aikana joku asia muuttuu, olkoon kyse sitten henkilö-, vaatimus- tai teknologiamuutoksesta, jolloin tiimin työtapoja on jälleen ruuvattava optimaalisen tehokkuuden aikaansaamiseksi. (Cockburn 2005, 22-23.)

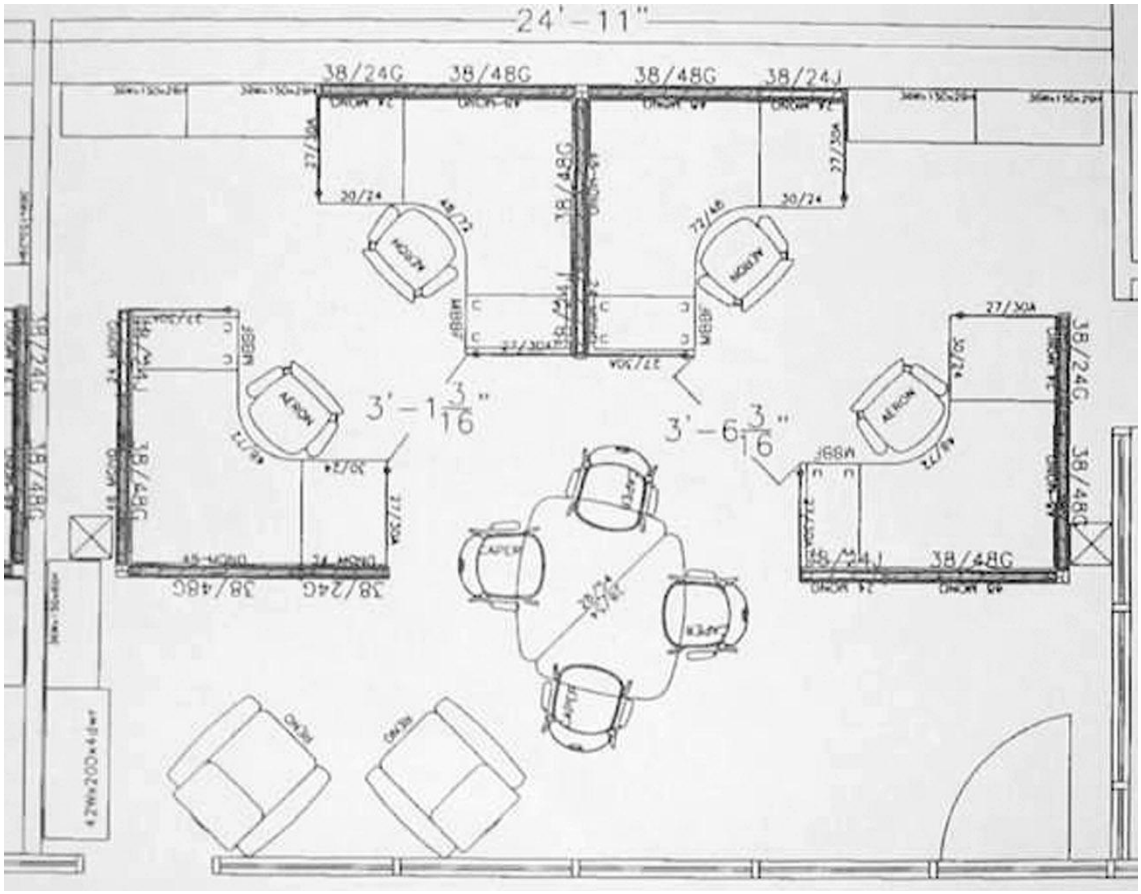
2.2.3 Osmoottinen viestintä

Kun projektitiimi on tarpeeksi pieni, kaikki ihmiset voi istuttaa samaan huoneeseen. Kun joku kysyy kysymyksen, ja joku toinen vastaa, kaikilla muillakin tiimin jäsenillä on mahdollisuus kuunnella, osallistua keskusteluun ja oppia – tai vain jatkaa työntekoa. Siinä pähkinänkuoressa osmoottinen viestintä. (Cockburn 2005, 24.)

Crystal Clear lupaa, että osmoottisen viestinnän ja usein toistuvan jakelun ollessa kunnossa palautetta etenemisestä tulee niin usein ja paljon, että se jo yksistään riittää viemään projektia eteenpäin. Osmoottinen viestintä on myös äärimmäisen halpa viestintätapa, ja samalla tavallaan myös koulutustapa: kysymällä ja kuuntelemalla ihmiset oppivat projektin päämäärät, toisten ihmisten hallitsemasta osaamisesta, sekä vinkkejä ohjelmointiin, testaukseen, suunnitteluun ja työkalujen käyttämiseen. Samalla pienet virheet tulevat korjattua ennen kuin ne ehtivät paisumaan isommiksi ongelmiksi. (Cockburn 2005, 24.)

Cockburn puhuu tarkemmin myös siitä, kuinka tiimin työhuone olisi syytä sisustaa, jotta osmoottinen viestintä olisi tehokkainta, mutta periaate on helppo tiivistää: ihmiset laitetaan samaan huoneeseen ja työpisteet sijoitetaan niin, että

muiden ihmisten on mahdollista kerääntyä katsomaan jonkun, esimerkiksi jonkun apua tarvitsevan, näyttöä. Lisäksi suositellaan jonkin erillisen keskustelualueen perustamista, vaikka käytössä olisikin vain yksi huone, niin, että siellä keskustelu ei häiritse työtä tekeviä. Esimerkkipohjapiirros löytyy kuvasta 1. (Cockburn 2005, 25-27.)

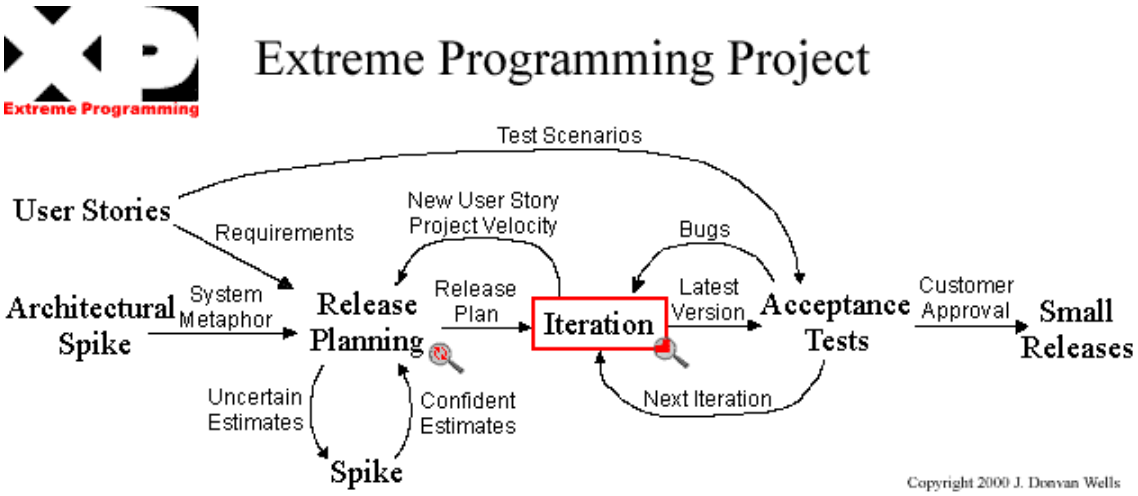


Kuva 1. Esimerkki osmoottisesti viestivän tiimin työhuoneesta (Cockburn 2005, 26).

Tiimissä on yleensä jonkinlainen pääsuunnittelija, joka on kokenein ja paras ohjelmoija. Tämän henkilön piilottaminen omaan huoneeseensa työrauhaan on Crystalin mukaan pääsääntöisesti huono idea. Tämän henkilön tulee olla muun tiimin kanssa samassa tilassa, jotta nämä voivat omaksua osmoottisesti tämän jakamat viisauden muruset ja kehittyä. Lisäksi menetelmässä kehoitetaan tarpeen vaatiessa antamaan tälle gurulle tämän tarvitsemaa työrauhaa jokunen tunti ihan normaaleina työaikoina, mikäli näyttää, että kaikki hänen aikansa

valuu kysymyksiin vastaamiseen; tätä kutsutaan ”hiljaisuuden tötteröksi”, ja pelkästään jo uskomattoman hienon nimensä ansiosta se ansaitsee tulla mainituksi. (Cockburn 2005, 28.)

2.3 Extreme Programming



Kuvio 2. XP:n vuokaavio (Wells 2000a).

Extreme Programming, XP, on kevyt iteratiivinen menetelmä pienille ja keskikokoisille ohjelmistonkehitystiimeille, jotka joutuvat tulemaan toimeen epämääräisten tai nopeasti vaihtuvien vaatimusten kanssa. Se lupaa ohjelmoijille, että nämä saavat toimia päivästä päivään tärkeiden asioiden kanssa ja että heillä on vapaus tehdä parhaansa ja valta tehdä päätöksiä, jotka he parhaiten osaavat tehdä. Asiakkaille ja johtajille XP lupaa, että he saavat suurimman mahdollisen arvon jokaisesta ohjelmointiviikosta, he pääsevät näkemään muutaman viikon välein heille tärkeiden asioiden edistymistä ja he pystyvät tekemään projektin suuntaa muuttavia päätöksiä missä tahansa vaiheessa aiheuttamatta kohtuuttomia kustannuksia. (Wells 2000a; Beck 2003, xv-xvi.)

Erityisen huomion arvoisia asioita XP:ssä ovat seuraavat:

- Ohjelmoijat työskentelevät pareittain.

- Kehitystä ohjaavat testit; ensin testataan, sitten ohjelmoidaan. Kun testit menevät läpi, eikä lisää testejä tule mieleen, toiminnallisuus on valmis.
- Ohjelmointiparit analysoivat, suunnittelevat, kehittävät, implementoivat ja testaavat koko järjestelmää.
- Integraatio ja sen testaaminen seuraa välittömästi kehityksen jälkeen.

(Beck 2003, 9.)

XP on rakennettu viiden arvon ja muutaman perusperiaatteen varaan, ja näistä on johdettu sen toimintaa ohjaavat säännöt (Wells 1999a; Beck 2003, 29-39; Wells 2009a). Seuraavassa tarkastellaan näitä osasia.

2.3.1 Arvot

XP:n arvot ovat yksinkertaisuus, kommunikaatio, palaute, rohkeus ja kunnioitus (Beck 2003, 29-35; Wells 2009a). Arvojen tarkoitus on pitää ihmiset kaidalla polulla, jotta nämä eivät alkaisi sooloilemaan ja ajamaan lyhyen aikavälin omaa etuaan, vaan työskentelisivät yhteisen pitkän aikavälin tavoitteen eteen (Beck 2003, 29).

Yksinkertaisuudella tarkoitetaan, että tehdään vain se, mitä täytyy ja mitä on pyydetty, jotta tehdyn investoinnin luoma arvo pysyisi mahdollisimman korkeana (Wells 2009a). Tavoitteena on pienin askelin ja virheet matkan varrella korjaten edetä maaliin (Wells 2009a). Yksinkertaisuus ei ole helppoa; on vaikeaa olla ajattelematta, mitä laajennuksia tänään tehtyyn työhön pitää tehdä tulevaisuudessa (Beck 2003, 30). XP:ssä kuitenkin tehdään sivistynyt arvaus; veikataan, että on parempi tehdä tänään yksinkertainen ja laajentaa sitä huomenna, mikäli tarvitsee, kuin tehdä tänään monimutkainen ja mahdollisesti huomata, ettei sitä käytetä ollenkaan (Beck 2003, 31).

Kommunikaatioon kuuluu, että kaikki ovat osa tiimiä, joka keskustelee kasvotusten päivittäin. Kaikki asiat vaatimuksista koodiin työstetään yhdessä ja paras mahdollinen ratkaisu kuhunkin ongelmaan kehitetään yhdessä (Wells 2009a). XP:ssä kommunikaatiota pidetään vireillä erinäisillä käytännöillä, kuten

esimerkiksi pariohjelmointi ja tehtävien pituuksien arviointi; ohjelmoijien, asiakkaiden ja johtajien on käytännössä pakko kommunikoida (Beck 2003, 30).

Palautetta kerätään XP:ssä paljon; kehitteillä olevaa järjestelmää demotaan aikaisessa vaiheessa ja usein, kuunnellen huolellisesti kehitystarpeita (Wells 2009a). Ohjelmoijien kirjoittamat ja suorittamat testit antavat välitöntä palautetta koodista ja järjestelmän tilasta (Beck 2003, 31-32). Myös asiakkaille annetaan palautetta heidän järjestelmän suhteen esittämistään toiveista (Beck 2003, 31-32).

Rohkeudella tarkoitetaan esimerkiksi rohkeutta puhua totta; etenemisestä ja arvioista ei valehdella, eikä tekosyitä epäonnistumisille keksitä, vaan tarkoitus on onnistua (Wells 2009a). XP:ssä on myös helppo olla rohkea, koska kukaan ei ole koskaan yksin (Wells 2009a); esimerkkinä pareittain ohjelmointi. Toisinaan rohkeutta vaaditaan myös kehitystyössä; vaikka jonkin koodin kirjoittamiseen olisi käytetty koko päivä, eikä se siltikään toimi tai vaikuta edes etenevän oikeaan suuntaan, se kannattaa heittää surutta roskakoriin (Beck 2003, 33). Joskus joku saattaa saada jonkun villin idean, josta onnistuessaan voisi seurata suuri hyöty, kuten vaikka järjestelmän monimutkaisuuden merkittävä väheneminen – tällöin XP:n kehotus on ”koita pois”; joko se toimii, ja hyöty on suuri, tai ei, jolloin sen voi nakata roskakoriin (Beck 2003, 34).

Kunnioitus tarkoittaa, että jokainen tiimin jäsen tuntee ja antaa arvostetun jäsenen ansaitsemaa kunnioitusta, sillä kaikki tuottavat arvoa (Wells 2009a). Kehittäjät kunnioittavat asiakkaiden ammattitaitoa ja päinvastoin; johtajat kunnioittavat suorittavan osapuolen kykyä ottaa valtaa ja vastuuta omasta työstään (Wells 2009a). Kunnioitus on tavallaan muita arvoja syvempi (Beck 2003, 35). Beckin mukaan (2003, 35) mikäli tiimin jäsenet eivät välitä toisistaan ja työstään, XP on tuhoon tuomittu, kuten tosin luultavasti ovat muutkin sovelluskehitysmenetelmät.

2.3.2 Periaatteet

Beckin mukaan (2003, 37) arvot ovat antavat viitteitä oikeista toimintatavoista, mutta ovat sellaisinaan liian epämääräisiä; siksi tarvitaan konkreettisempia periaatteita. XP:n perimmäisimmät periaatteet ovat nopea palaute, oletus yksinkertaisuudesta, inkrementaalinen muutos, muutoksen hyväksyminen ja laadukas työ (Beck 2003, 37). Lisäksi Beck esittelee (2003, 39) vielä kymmenen vähemmän keskeistä periaatetta, joiden on tarkoitus auttaa tietyissä spesifisissä tilanteissa, mutta seuraavassa keskitytään viiteen keskeisimpään.

Nopean palautteen johtoajatus on siinä, että palaute on, kuten on todistettu, hyödyllisintä, kun se tulee heti eikä sitten joskus; esimerkiksi ohjelmoijat oppivat parhaat tavat järjestelmän suunnittelemiseksi, testaamiseksi ja implementoimiseksi, ja tuo oppi pistetään kiertoon välittömästi eikä päivien, viikkojen tai kuukausien päästä. (Beck 2003, 37-38).

XP kehoittaa lähestymään jokaista ongelmaa aivan kuin sen voisi ratkaista suorastaan naurettavan yksinkertaisesti. Beckin mukaan (2003, 38) 98 tapausta sadasta ovat ratkaistavissa näin, ja näin säästyy paljon aikaa käytettäväksi niiden kahden muun ratkaisemiseksi. Sen sijaan, että murehdittaisiin huomista, vaikka se ihmisluontoon kuuluukin, keskitytään ratkaisemaan tämän päivän ongelma huolellisesti ja luotetaan siihen, että osaaminen riittää laajentamaan ratkaisua huomenna tarpeen vaatiessa. (Beck 2003, 38.)

Kerralla tehdyt isot muutokset eivät Beckin mukaan (2003, 38) vain toimi, vaan kaikki ongelmat tulee ratkaista sarjana pieniä ongelmia, inkrementaalisesti. Monet asiat XP:ssä, kuten järjestelmän suunnitelma ja tiimi, muuttuvat aina vähän kerrallaan. (Beck 2003, 38.)

”Muutokset eivät ole pysyviä, mutta muutos on”, kuten Geddy Lee laulaa (Rush 1981). XP kehoittaa hyväksymään muutoksen (Beck 2003, 38). Beckin mielestä (2003, 38) paras strategia on sellainen, joka jättää eniten valinnanvaraa, mutta ratkaisee silti kiireellisimmän ongelman. Tällä viitattaneen esimerkiksi yksinkertaisuuden periaatteeseen ja kehoitukseen olla huolehtimatta huomisesta; turha murehtia huomista, kun ei tiedä, mitä silloin edes halutaan.

Kukaan ei halua työskennellä leväperäisesti, vaan kaikki haluavat tehdä hyvää työtä, on XP:n perusperiaate työn laadusta. XP:ssä neljästä projektin muuttujasta – laajuudesta, hinnasta, ajasta ja laadusta – laatu ei ole vapaa muuttuja, vaan laatua on vain kahdenlaista, ”erinomaista” ja ”järjettömän erinomaista”. Beckin mukaan (2003, 38) sitä joko tekee hyvää jälkeä tai sitten ei nauti työstään, jolloin projekti on tuomittu kadotukseen. (Beck 2003, 38.)

2.3.3 Säännöt

XP:n säännöt on jaettu viiteen eri aihealueeseen, jotka ovat projektin suunnittelu, johtaminen, järjestelmän suunnittelu, koodaaminen ja testaaminen (Wells 1999a). Ne käydään pintapuolisesti läpi seuraavassa.

Projektin suunnittelun säännöt ovat:

1. **Käyttäjätarinoiden kirjoittaminen**. Nämä muistuttavat käyttötapauksia ja niitä käytetään laajojen vaatimusmäärittelyjen sijasta, aika-arvioiden luomiseen ja testien kirjoittamiseen. Nimensä mukaisesti ne ovat järjestelmän loppukäyttäjien kirjoittamia. (Wells 1999b.)

2. **Julkaisusuunnittelu luo julkaisuaikataulun**. Julkaisusuunnittelu-kokouksessa luodaan julkaisusuunnitelma, johon kehittäjät laittavat ylös jokaiselle käyttäjätarinalle arvioimansa toteutusaika-arviot ja asiakkaat priorisoivat käyttäjätarinat. Julkaisusuunnitelmaa käytetään pohjana iteraatiosuunnitelmille (katso sääntö 5). Julkaisuaikataulussa kerrotaan, mitkä käyttäjätarinat implementoidaan missäkin iteraatiossa. (Wells 1999c; Wells 1999d.)

3. **Tee lukuisia pieniä julkaisuja**, jotta asiakas pääsee mahdollisimman pian antamaan palautetta ja tekemään tarvittaessa muutospyyntöjä (Wells 1999e).

4. **Projekti jaetaan iteraatioihin**. XP suosittelee 1-3 viikon mittaisia iteraatioita, ja iteraatioiden pituus tulee pitää samana läpi projektin; näin syntyy niin sanottu projektin sydämen syke (Wells 1999f).

5. **Iteraation suunnittelu aloittaa jokaisen iteraation**. Joka iteraation alussa pidetään iteraation suunnittelukokous, jossa asiakas valitsee tärkeysjärjestyksessä käyttäjätarinoita, joita haluaa toteutettavan iteraation aikana, ja kehittäjät arvioivat, paljonko kuhunkin kuluu aikaa. Projektin nopeuslaskentaa käytetään perusteena sille, montako tarinaa ryhmä ottaa kontolleen. Käyttäjätarinat ja mahdolliset korjattavat virheet kirjoitetaan korteille ja jaetaan ryhmälle tehtäväksi. (Wells 1999g.)

Johtamisen säännöt ovat:

1. **Anna tiimille oma avonainen työtila**, sillä sellaisessa on helpompi kommunikoida. Eikä saa unohtaa tarpeellisia havainnollistamisen apuvälineitä, kuten piirtotaulua. (Wells 2009b.)

2. **Aseta vakaalla pohjalla oleva etenemisvauhti**. Mikäli näyttää, että tavoitteet eivät täyty, muuta tavoitteita. Älä ylityöllistä ihmisiä. (Wells 2009c.)

3. **Pidä päivittäinen seisomakokous**. Näissä lyhyissä kokouksissa koko tiimi seisoo ringissä ja asialista pidetään lyhyenä ja käytännöllisenä. Jokainen kehittäjä kertoo vähintään kolme asiaa: mitä eilen saavutettiin, mitä tänään yritetään ja mitä ongelmia on. (Wells 1999h.)

4. **Projektin nopeutta mitataan** yksinkertaisesti laskemalla valmiiksi toiminnoiksi saatettujen käyttäjätarinoiden aika-arviot yhteen. Mitä enemmän valmistuu, sitä nopeammin edetään. (Wells 1999i.)

5. **Liikuta ihmisiä ympäriinsä**. Jokaisen pitäisi osata paljon kehitettävästä järjestelmästä eikä olla vain oman kapean alansa asiantuntija (Wells 1999j).

6. **Korjaa XP, kun se hajoaa**, sillä sitä pitää kuitenkin sovittaa projektiin paremmin sopivaksi, ja tuo sovitustyö tulee tehdä yhdessä kehitystiimin kanssa, jotta kaikki pelaavat samoilla säännöillä (Wells 1999k).

Järjestelmän suunnittelun säännöt ovat:

1. **Yksinkertaisuus**. Yksinkertainen asia on aina nopeampi viimeistellä kuin monimutkainen. Mikäli havaitset jonkin asian monimutkaiseksi, yksinkertaista sitä; on nopeampaa ja halvempaa tehdä se nyt kuin myöhemmin. (Wells 2009d.)

2. **Valitse järjestelmälle sopiva vertaus**. Jos asiaa voi verrata johonkin yleisesti tunnettuun, vertaa; se helpottaa myöhemmin järjestelmän pariin tulevia (Wells 1999l).

3. **Käytä LVY-kortteja suunnittelussa**. LVY on lyhenne sanoista luokka, vastuut ja yhteistoiminta. Yksi kortti kuvaa yhtä oliota järjestelmässä, ja kortille kirjataan luokan nimi, vastuut ja luokat, joiden kanssa se on yhteistoiminnassa. Korteilla hahmotellaan, miten oliot järjestelmässä viestivät keskenään. (Wells 1999m.)

4. **Luo piikkiratkaisuja riskin pienentämiseksi**. Piikkiratkaisu on pieni ja vain väliaikaisesti olemassa oleva testiohjelma, jolla testataan jotain ratkaisua johonkin vaikeaan tekniseen tai suunnittelulliseen ongelmaan. (Wells 1999n.)

5. **Yhtäkään toiminnallisuutta ei lisätä etuajassa**. Älä arvaile, mitä saatetaan tarvita myöhemmin; 90 % varmuudella ei kuitenkaan tarvita. Keskity tähän päivään. (Wells 1999o.)

6. **Refaktoroi aina kun mahdollista**. Refaktoroinnilla tarkoitetaan tarpeettoman tai nyttemmin tarpeettomaksi muuttuneen koodin ja päällekkäisyyksien hävittämistä, ja sillä tähdätään koodin siisteyteen ja ymmärrettävyyteen (Wells 1999p).

Koodaamisen säännöt ovat:

1. **Asiakas on aina saatavilla**. XP vaatii, että asiakkaan edustaja on aina läsnä, mielellään kasvotusten ja osana kehitystiimiä, kirjoittamassa käyttäjätarinoita, antamassa palautetta, antamassa lisätietoa kehittäjille ja testaamassa, että kehitettävä järjestelmä tekee, mitä pitääkin. (Wells 1999q.)

2. **Koodin pitää noudattaa sovittuja standardeja**. Koodausstandardit pitävät koodin yhdenmukaisena ja helppona koko tiimille lukea ja refaktoroida. (Wells 1999r.)

3. **Koodaa yksikkötesti ensin**. Näin itse toiminnallisuus on helpompi koodata, kun tietää heti, milloin se toimii oikein. Testejä kirjoitetaan niin monta, että ne todella varmistavat toiminnallisuuden jokaisen aspektin toimivan oikein. (Wells 2000b.)

4. **Kaikki koodi koodataan pareittain**. Menetelmässä väitetään, että kaksi ihmistä yhdellä tietokoneella saa aikaiseksi yhtä paljon kuin kaksi ihmistä omilla tietokoneillaan, mutta syntyvä koodi on laadukkaampaa. Parin molemmat osapuolet osallistuvat tekemiseen. (Wells 1999s.)

5. **Vain yksi pari kerrallaan integroi koodia**. Parit koodaavat samanaikaisesti, mutta vain yksi kerrallaan saa integroida tekemänsä muutokset. Pari julkaisee tuotoksensa muulle tiimille vasta, kun sen on testien avulla osoitettu toimivan saumattomasti muun järjestelmän kanssa hajoittamatta myöskään mitään vanhaa ominaisuutta. (Wells 1999t.)

6. **Integroi usein**. Jotta muutoksien järjestelmään integroimisen aiheuttamat ongelmat tulisivat ilmi mahdollisimman aikaisessa vaiheessa, integrointia tulisi harjoittaa mahdollisimman usein, vähintään kerran päivässä. (Wells 1999u.)

7. **Käytä integroinnille omistettua tietokonetta**. Näin on helpompi pysyä kärryillä siitä, mikä on viimeisin versio, eivätkä parit myöskään pääse

pakottamaan tekemiään muutoksia omalta koneeltaan muille. (Wells 1999v.)

8. **Käytä yhteisomistusta**. Jokainen tiimin jäsen omistaa yhtä lailla koko projektin, joten kaikilla on vapaus muokata mitä tahansa osiota siitä. Jokaisen tulee myös olla ainakin suunnilleen tietoisia järjestelmän yleiskuvasta. Tällä vaimennetaan projektista poistuvan ihmisen aiheuttamaa negatiivista vaikutusta. (Wells 1999w.)

Testaamisen säännöt ovat:

1. **Kaikelle koodille on olemassa yksikkötestit**. Yksikkötestit ovat yksi XP:n kulmakivistä. Niillä varmistetaan kaiken koodin toiminta; kaikkien, vanhojenkin, testien on mentävä aina läpi, tai muuten jokin uusi toiminnallisuus on tehty tai integroitu väärin, eikä sitä voi julkaista. Myöskään koodia, jolle ei ole kirjoitettu yksikkötestiä, ei saa julkaista. (Wells 1999x.)

2. **Kaiken koodin on läpäistävä yksikkötestit ennen julkaisua**. Katso edellinen sääntö.

3. **Kun ohjelmointivirhe löytyy, kirjoitetaan testi**. Näin ihmiset pysyvät perillä siitä, onko virhe korjattu, ja huomaavat, mikäli se tulee takaisin. (Wells 1999y.)

4. **Hyväksymistestejä ajetaan usein ja tulos julkaistaan**. Hyväksymistestit kirjoitetaan käyttäjätarinoiden pohjalta. Niiden on tarkoitus testata, että käyttäjätarinoiden pohjalta kirjoitettu toiminnallisuus, joka saattaa olla laaja ja siksi yksittäisten yksikkötestien testaamattomissa, toimii oikein. Asiakkaiden vastuulla on katsoa, että hyväksymistestien tulokset ovat oikeita, eli järjestelmä tekee, mitä pitääkin, ja priorisoida, mitkä epäonnistuneet hyväksymistestit ovat tärkeimpiä. (Wells 1999z.)

2.4 Feature-Driven Development

Feature-Driven Development, FDD, eroaa muista ketteristä menetelmistä niin, että siinä järjestelmän suunnittelu tehdään pääosin etukäteen. Järjestelmän

oliomalli ja lista toiminnallisuuksista tehdään kerran projektin alussa ja toiminnallisuudet rakennetaan iteratiivisesti. Luonnollisesti toiminnallisuudet ja oliomalli saattavat muuttua, mutta niiden muutokset eivät ole olennainen osa tätä menetelmää. (Koch 2004, 249.)

FDD koostuu kahdeksasta käytännöstä, jotka ovat oliomallinnus, kehittäminen toiminnoittain, luokan omistajuus, toimintotiimit, katselmukset, säännöllinen rakennusaikataulu, konfiguraationhallinta ja raportointi (Koch 2004, 249-251).

Lopullisen järjestelmän oletettuja luokkia kuvaava oliomalli luodaan suhteellisen yksityiskohtaisesti jo projektin alussa, tarkoituksena ottaa kaikkien sidosryhmien oletukset järjestelmän suhteen esiin. Malli toimii myös eräänlaisena karttana projektille, ja sitä täydennetään ja korjaillaan tarvittaessa projektin myötä niin, että se vastaa todellisuutta. (Koch 2004, 249.)

Oliomalli kuvaa järjestelmää luokkakohtaisesti, mutta järjestelmä kehitetään toiminto kerrallaan. FDD:ssä toiminto on määritelty suunnilleen näin: ”toiminto on pieni asiakasta hyödyttävä funktio, jonka muoto on <toiminta> <kohde> <tulos>”, esimerkiksi ”nouda potilaan potilasasiakirjat”. Yleensä toiminnot ovat niin pieniä, että ne voi ohjelmoida muutamassa tunnissa tai päivässä; FDD:n asettama maksimiaika toiminnon toteuttamiselle on kaksi viikkoa. (Koch 2004, 249-250.)

Luokan omistajuus tarkoittaa, että yhdellä luokalla on aina yksi kehittäjä, joka omistaa luokan, ja jonka oletetaan ymmärtävän sen toiminnan ja yksityiskohdat täysin. Ajatuksena on, että tällainen henkilö on pätevämpi tekemään muutoksia luokkaan kuin kukaan muu projektiryhmässä. Tästä luonnollisesti seuraa, että luokan omistajan tulee olla mukana jokaisen luokkaan vaikuttavan toiminnon kehityksessä. (Koch 2004, 250.)

Kuten sanottu, kehitys tapahtuu toiminnoittain, ja jokaisella luokalla on omistajansa. Niinpä toiminnon kehittää toimintotiimi, johon kuuluu toiminnon omistaja ja kaikkien niiden luokkien, joihin toiminto vaikuttaa, omistajat. Tällä tähdätään toiminnon nopeaan ja tehokkaaseen kehittämiseen. Mikäli tiimi huomaa, että toiminnon kehittäminen vaatii kajoamista johonkin muuhunkin

luokkaan, kyseisen luokan omistaja otetaan osaksi tiimiä. Useita toimintotiimejä toimii usein samanaikaisesti rinnakkain, joten yksi kehittäjä saattaa kuulua samanaikaisesti moneen tiimiin. Tiimien koostumus vaihtelee päivästä ja viikosta toiseen dynaamisesti. Tiimi puretaan, kun toiminto on saatu valmiiksi ja se on testattu ja integroitu muuhun projektiin. (Koch 2004, 250-251.)

FDD:ssä käytetään täsmällisiä katselmuksia, joilla pyritään varmistamaan suunnittelun ja koodin laatu pyrkimällä löytämään näistä niiden virheet ja heikkoudet. Tärkeä tavoite on myös se, että luokat tulevat tutuiksi useille projektitiimin jäsenille siltä varalta, että joku luokan omistaja jättää projektin. Katselmukset ovat myös eräänlainen pakkokeino varmistaa, että projektin koodausstandardeja noudatetaan yhdenmukaisesti. (Koch 2004, 251.)

Säännöllisellä rakennusaikataululla tarkoitetaan sitä, että kehitteillä olevasta järjestelmästä tulee olla aina päivitetty, kaikilla siihen asti kehitetyillä toiminnallisuuksilla varustettu, versio saatavilla, jota käytetään testaamiseen ja dokumentoinnin pohjana sekä asiakkaalle esittelyyn. FDD:ssä ei määritellä mitään tiettyä aikataulua, mutta säännöllisiä rakennuksien tulee olla, ja mielellään ainakin viikottaisia, luonnollisesti projektista ja ympäristöstä riippuen. (Koch 2004, 251.)

FDD:ssä kehotetaan panostamaan konfiguraationhallintaan, ja sitä pidetään yhtenä FDD-projektin onnistumisen elementeistä. Menetelmä ei kuitenkaan kuvaile aihetta tarkasti, vaan sysää vastuun konfiguraationhallinnan yksityiskohdista projektitiimin harteille; heidän tulee päättää, mikä on sopiva ratkaisu riippuen projektin koosta, monimutkaisuudesta ja laajuudesta. (Koch 2004, 251.)

Raportointia ja projektin etenemisen seurantaa varten FDD:hen on kehitetty yksilöllinen mekanismi, jossa jokaiselle toiminnolle määritellään muutama virstanpylväs ja jokaiselle virstanpylväälle oma painokerroin. Virstanpylväitä ovat muun muassa suunnitelma valmiina tarkastukselle, koodi valmiina tarkastukselle ja koodi tarkastettu. Painokertoimia määritellään historian pohjalta; esimerkiksi mikäli johonkin virstanpylvääseen kuluu yleensä 4% ajasta, kyseisen

virstanpylvään painokertoimeksi laitetaan 0,04. Toiminto alkaa nollapisteestä ja valmiin toiminnon arvo on 1. Koko projektin etenemistä voi seurata kaavalla, joka on esitetty kuviossa 3. (Koch 2004, 252.)

Toiminnon arvo = ∑(saavutettujen virstanpylväiden painot)

Aloittamattoman toiminnon arvo = 0,00
Täysin valmiin toiminnon arvo = 1,00
Kehitteillä olevan toiminnon arvo on välillä 0 - 1

$$\text{Projektin tila} = \frac{\sum(\text{toimintojen arvot})}{\text{Toimintojen lukumäärä}}$$

Kuvio 3. Projektin tilayhtälö (Koch 2004, 252).

Kaavaa käyttämällä projektin tila on objektiivisesti ja helposti esitettävissä. Esimerkiksi mikäli 285 toimintoa sisältävästä projektista on suoritettu virstanpylväitä summan 201 edestä, projekti on 70,5-prosenttisesti valmis. (Koch 2004, 252.)

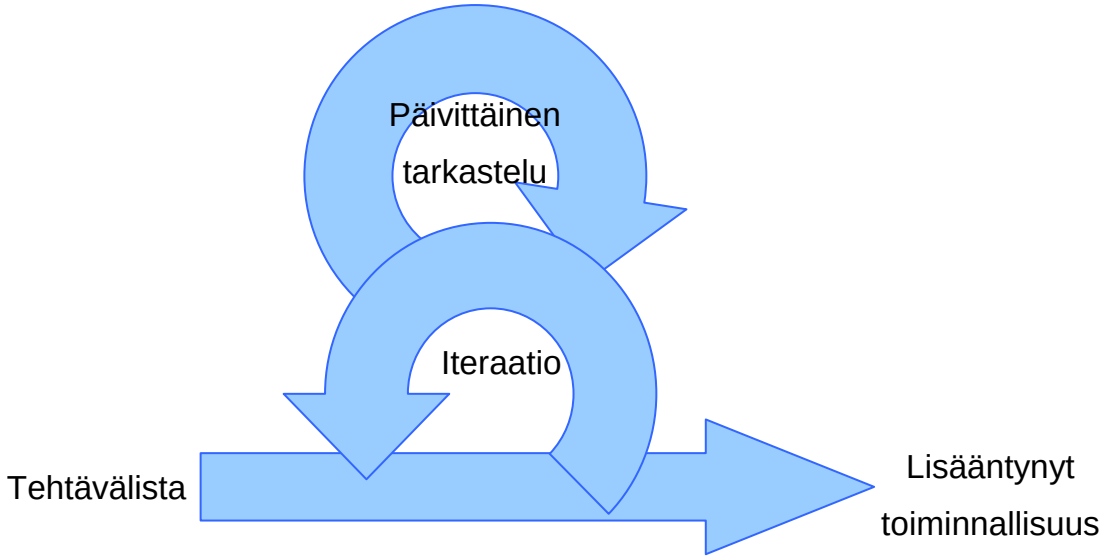
2.5 Scrum

Scrum ei ole pelkkä ohjelmistonkehitysmenetelmä, vaan tuotekehitysmenetelmä, jota voi käyttää monella alalla, mukaan lukien ohjelmistonkehitys. Nimi ”scrum” on peräisin rugbystä ja sillä viitataan strategiaan, jota käytetään pallon palauttamiseksi peliin. (Koch 2004, 257.)

Kaikki Scrumin käytännöt rakentuvat iteratiivisen ja inkrementaalisen prosessirungon päälle. Runko on hahmoteltu kuviossa 4. Ideana on, että jokaisen iteraation, joita Scrumissa kutsutaan sprinteiksi, alussa tiimi tarkastelee, mitä pitäisi tehdä, ja valitsee tehtäväksi ne toiminnallisuudet, jotka se uskoo saavansa iteraation loppuun mennessä valmiiksi. Tämän jälkeen se saa

työrauhan iteraation loppuun asti, jolloin aikaansaannokset esitellään sidos-ryhmille. (Schwaber 2004, 5-6; Schwaber 2004, 8.)

Scrumissa on määritelty kolme roolia ja kolme ”artefaktia”, joita tarkastellaan seuraavassa tarkemmin, kuten myös Scrum-prosessin kulkua.



Kuvio 4. Scrumin runko (Schwaber 2004, 6).

2.5.1 Roolit

Kuten sanottu, on olemassa kolme Scrum-roolia, ja ne ovat tuotteen omistaja, tiimi ja Scrum-mestari; projektin vetovastuu on hajautettu näille kolmelle roolille. (Schwaber 2004, 6.)

Tuotteen omistaja edustaa sitä osapuolta, jolla on panos projektissa ja siitä tuloksena syntyvässä järjestelmässä, eli yleensä asiakasta. Tuotteen omistajan tehtävänä on kerätä rahoitus projektille ja laatia alustavat määrittelyt projektin vaatimuksista, investoinnin tuottoprosentista ja julkaisusuunnitelmista. Projektin vaatimuslistaa kutsutaan tuotteen tehtävälistaksi ja siihen palataan luvussa 2.5.2. (Schwaber 2004, 6-7.)

Itsehallinnoivan, itseorganisoituvan ja monitaitoisen tiimin tehtävä on rakentaa toiminnallisuudet, eli muuttaa tuotteen tehtävälista tai sen osa lisääntyneeksi toiminnallisuudeksi yhden iteraation aikana; tämän – sekä koko projektin – onnistumisesta tiimin jäsenillä on yhteisvastuu. (Schwaber 2004, 7.)

Scrum-mestarin vastuulla on itse Scrum-prosessi: sen opettaminen kaikille osapuolille, sen käyttäminen yrityksen kulttuuriin soveltuvalla tavalla ja sen varmistaminen, että jokainen seuraa Scrumin sääntöjä ja käytäntöjä. (Schwaber 2004, 7.)

Näihin rooleihin valitut ihmiset ovat sitoutuneet projektiin; muut saattavat olla kiinnostuneita, mutta he eivät ole varsinaisesti mukana. Scrum tekee näiden välille tarkan eron pyrkimyksenä varmistaa, että projektista vastuussa olevilla on valta tehdä kaikki tarvittava sen onnistumiseksi, ja että muut eivät pääse häiritsemään. Projektiin sitoutuneita kutsutaan toisinaan sioiksi ja ulkopuolisia kanoiksi perustuen vanhaan vitsiin, jossa kana ja sika ovat perustamassa ravintolaa. (Schwaber 2004, 7.)

2.5.2 Artefaktit

Scrumissa on määritelty kolme artefaktia, joita käytetään läpi prosessin (Schwaber 2004, 9). Ne ovat tuotteen tehtävälista, sprintin tehtävälista ja lista valmiista tehtävistä (Schwaber 2004, 10-12).

Tuotteen tehtävälista (engl. Project Backlog), jota ylläpitää tuotteen omistaja, sisältää listauksen kehitettävälle järjestelmälle asetetuista vaatimuksista. Tuotteen tehtävälista on dynaaminen ja se muuttuu halki projektin sen mukaan, mitä järjestelmän ajatellaan tarvitsevan ollakseen sopiva, kilpailukykyinen ja käyttökelpoinen. (Schwaber 2004, 10.)

Tuotteen tehtävälista on priorisoitu niin, että ominaisuudet, jotka todennäköisimmin tuottavat arvoa, ovat korkean prioriteetin tehtäviä (Schwaber 2004, 8). Tuotteen tehtävälista on myös jaettu julkaisujen mukaan (Schwaber 2004, 8), eli minkä sprintin jälkeen toiminnallisuuden oletetaan olevan valmis.

Sprintin tehtävälistan, jollaisesta on esimerkki kuvassa 2, koostaa tiimi, ja siihen valitaan ne tehtävät, jotka tiimi uskoo saavansa valmiiksi yhden sprintin aikana. Tehtävät jaetaan niin, että jokainen niistä kestää 4-16 tuntia; pidemmät jaetaan täsmällisempiin osiin. Jokaiselle tehtävälle merkataan vastuuhenkilö tai vastuuhenkilöt, ja etenemistä seurataan reaaliaikaisesti. (Schwaber 2004, 12-13.)

				Hours of work remaining unt											
Task Description	Originator	Responsible	Status (Not Started/In Progress/ Completed)	1	2	3	4	5	6	7	8	9	10	11	12
Meet to discuss the goals and features for Sprint 3-6	Danielle	Danielle/Sue	Completed	20	0	0	0	0	0	0	0	0	0	0	0
Move Calculations out of Crystal Reports	Jim	Allen	Not Started	8	8	8	8	8	8	8	8	8	8	8	8
Get KEG Data		Tom	Completed	12	0	0	0	0	0	0	0	0	0	0	0
Analyse KEG Data - Title		George	In Progress	24	24	24	24	12	10	10	10	10	10	10	10
Analyse KEG Data - Parcel		Tim	Completed	12	12	12	12	12	4	4	4	0	0	0	0
Analyse KEG Data - Encumbrance		Josh	In Progress							12	10	10	10	10	10
Analyse KEG Data - Contact		Danielle	In Progress	24	24	24	24	12	10	8	6	6	6	6	6
Analyse KEG Data - Facilities		Allen	In Progress	24	24	24	24	12	10	10	10	10	10	10	10
Define & build Database		Barry/Dave	In Progress	80	80	80	80	80	80	60	60	60	60	60	60
Validate the size of the KEG database		Tim	Not Started												
Look at KEG Data on the G:\		Dave	In Progress	3	3	3	3	3	3	3	3	3	3	3	3
Confirm agreement with KEG		Sue	Not Started												
Confirm KEG Staff Availablity		Tom	Not Started	1	1	1	1	1	1	1	1	1	1	1	1
Switch JDK to 1.3.1. Run all tests.		Allen	Not Started	8	8	8	8	8	8	8	8	8	8	8	8
Store PDF files in a structure		Jacquie	Completed	8	0	0	0	0	0	0	0	0	0	0	0
TopLink. Cannot get rid of netscape parser		Richard	Comleted	4	0	0	0	0	0	0	0	0	0	0	0
Buld test data repository		Barry	In Progress	10	10	10	10	10	10	10	10	8	8	8	8
Move application and database to Qual (incl Crystal)		Richard	Completed	4	4	4	4	4	4	4	0	0	0	0	0
Set up Crystal environment		Josh	Completed	2	2	2	2	1	1	1	0	0	0	0	0
Test App in Qual		Sue	In Progress												20
Defining sprint goal required for solution in 2002		Lynne	In Progress	40	40	40	40	40	40	40	38	38	38	38	38
Reference tables for import process		Josh	In Progress												
Build standard import exception process		Josh	In Progress									12	12	12	10
Handle multiple file imports on same page		Jacquie	Disregarded												
Migrate CruiseControl Servlet to iWS 6.0 (landcc_7101) server		Allen	Not Started	4	4	4	4	4	4	4	4	4	4	4	4
Create web server for Qual on PF1D8		Allen	Completed	1	0	0	0	0	0	0	0	0	0	0	0
LTCS Disk		Danielle/ George	In Progress	12	12	12	12	8	8	8	8	8	8	8	8
Follow thru with questions about KEG data to Sue/Tom, re: Keg, LTO	Jacquie	Danielle	Completed	10	10	10	10	10	8	8	0	0	0	0	0
Map KEG data to Active Tables - see also #14	Jacquie	Jacquie/Allen	In Progress	50	50	50	50	50	50	50	50	50	50	50	50
Preparer SQL to import from KEG tables to Active Tables	Jacquie	George	In Progress	25	25	25	25	25	25	25	25	25	25	25	25

Kuva 2. Esimerkki sprintin tehtävälistasta (Schwaber 2004, 10).

Listaan valmiista tehtävistä lisätään tietenkin sprintin aikana valmiiksi saadut tehtävät. Toiminnallisuuksien tapauksessa niiden pitää olla käyttöohjeella varustettuja, läpeensä testattuja, hyvin jäsenneltyjä ja hyvin kirjoitettuja kooditiedostoja, jotka on käännetty suoritettavaan muotoon. (Schwaber 2004,12-13.)

2.5.3 Prosessin kulku

Scrum-projekti alkaa visiosta. Tuotteen omistaja esittelee vision rahoittajille ja koostaa alustavan tuotteen tehtävälistan, joka tarkentuu ja muuttuu myöhemmin. (Schwaber 2004, 7-8.)

Tuotteen tehtävälista muutetaan valmiiksi tuotteeksi sprintti kerrallaan. Jokainen sprintti on yleensä 2-4 viikon mittainen iteraatio, ja jokainen sprintti alkaa suunnittelukokouksella, jossa tuotteen omistaja ja tiimi yhdessä miettivät, mitä seuraavan sprintin aikana tehdään. Tuotteen omistaja valitsee tehtävälistasta tärkeimmän prioriteetin tehtäviä, joita haluaa tehtävän, ja tiimi kertoo, miten paljon se kuvittelee pystyvänsä tekemään sprintin aikana. Sprintin suunnittelukokous on rajoitettu kestoltaan 8-tuntiseksi, ja siitä ensimmäinen puolikas käytetään tuotteen omistajan ja tiimin yhteissuunnitteluun ja jälkimmäisen neljätuntisen aikana tiimi koostaa sprintin tehtävälistan. (Schwaber 2004, 8; Scrum Alliance 2011.)

Tiimi kokoontuu päivittäin 15-minuuttisen päivittäiseksi scrumiksi kutsutun kokouksen merkeissä. Päivittäisessä scrumissa jokainen tiimin jäsen joutuu vastaamaan seuraaviin kolmeen kysymykseen: (Schwaber 2004, 8)

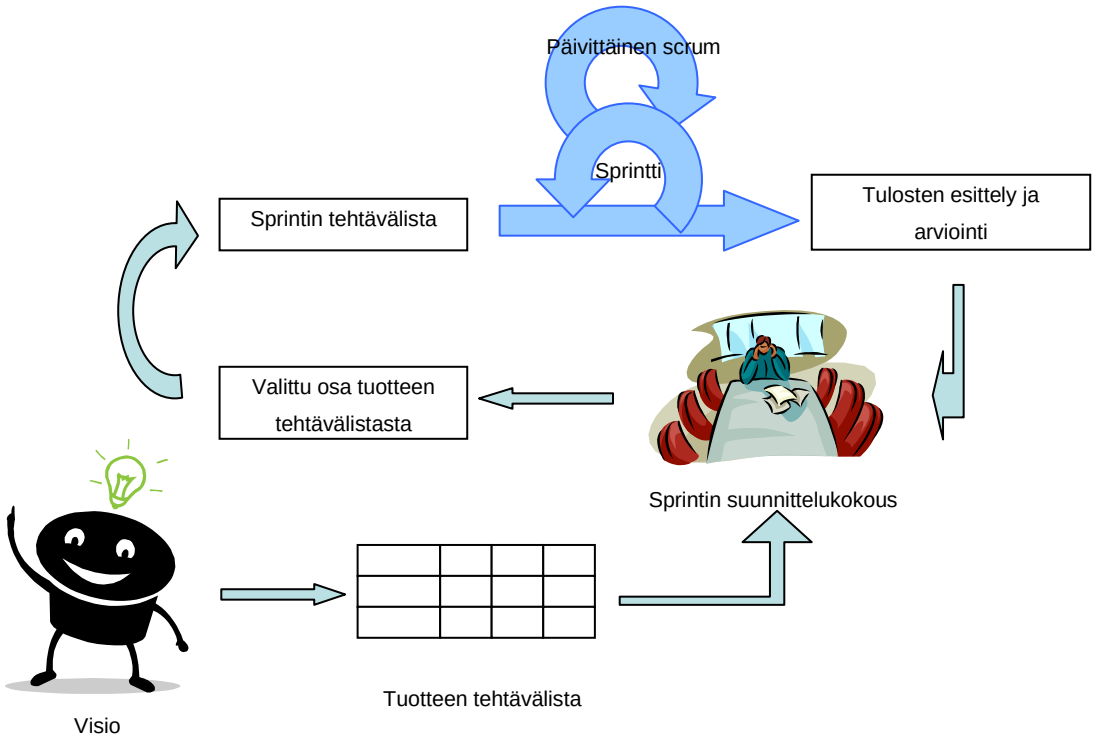
- Mitä olet tehnyt projektin eteen sitten viime kokouksen?
- Mitä aiot tehdä projektin eteen ennen seuraavaa kokousta?
- Mitä esteitä on tielläsi?

(Schwaber 2004, 8.)

Jokaisen sprintin lopuksi pidetään epämuodollinen nelituntinen arviointikokous, jossa tiimi esittelee aikaansaannoksensa tuotteen omistajalle ja muille sidos-

ryhmille, jotka haluavat osallistua. Tarkoituksena on yhteisesti miettiä, mitä tiimin pitäisi seuraavaksi tehdä. Arvioinnin jälkeen ja ennen seuraavaa sprintin suunnittelukokousta Scrum-mestari pitää vielä oman kolmituntisen katselmuksensa menneestä sprintistä tiimin kanssa, jossa tarkoituksena on hioa, Scrumin määrittelemissä rajoissa tietenkin, tiimin kehitysprosessia tehokkaammaksi ja mukavammaksi seuraavaa sprinttiä varten. (Schwaber 2004, 9.)

Scrum-prosessin kulkua on vielä hahmoteltu kuvioon 5.



Kuvio 5. Scrum-prosessi (Schwaber 2004, 9).

3 DSDM ATERN –MENETELMÄ

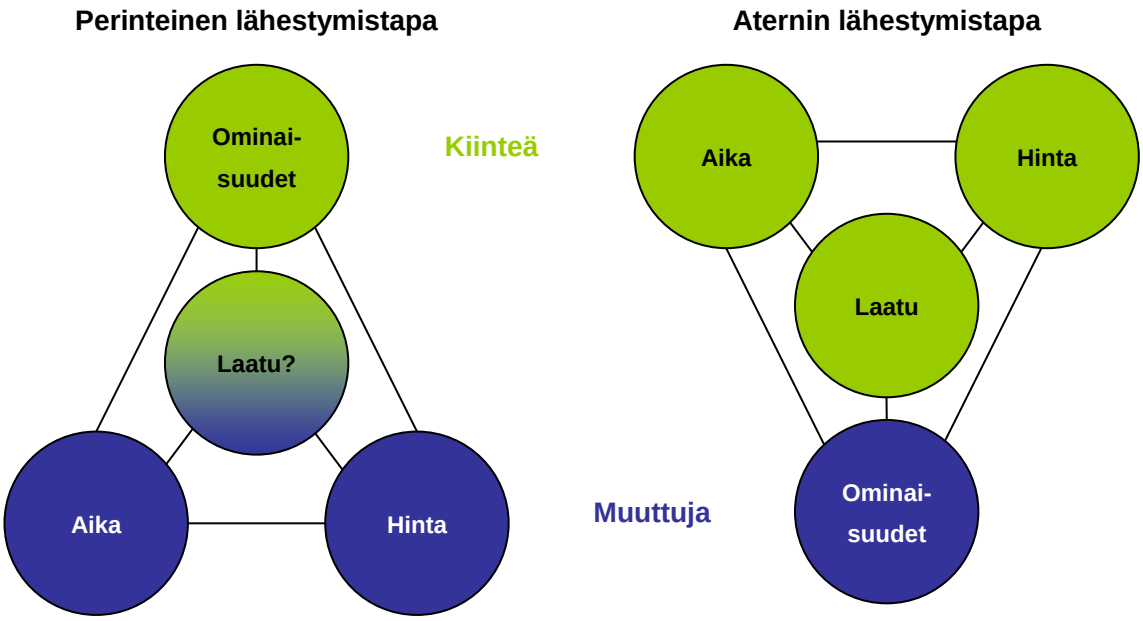
Dynamic Systems Development Method, DSDM, on yksi ketteristä menetelmistä. Vahvoja tunnuspiirteitä sen ketteryydestä ovat muun muassa iteratiivisuus, pelkkien prosessien sijasta ihmisiin kohdistettu huomio ja muuttuvien vaatimusten ottaminen huomioon. Aikaisemmin kyseessä oli puhtaasti IT-alan projekteihin ja ohjelmiin tarkoitettu menetelmä, mutta uusin versio, DSDM Atern, soveltuu myös projekteihin, joilla ei ole minkäänlaista kytköstä teknologiaan. Aternia voi myös käyttää yhdessä jonkun muun ketterän menetelmän kanssa, esimerkkeinä on mainittu ainakin XP ja Scrum. (Clifton & Dunlap 2003; DSDM 2008, 8; DSDM 2008, 14.)

Työkaluista ja teknologioista riippumaton Atern yrittää avustaa ihmisiä työskentelemään tehokkaasti yhdessä tavoitteiden saavuttamiseksi, sillä sen mukaan useampi projekti epäonnistuu ihmisten kuin teknologian takia. Menetelmä perustuu oletukseen, jonka mukaan mitään tuotosta ei koskaan saada täydelliseksi ensimmäisellä yrittämällä, mutta 80 % tuotoksesta pystytään saamaan valmiiksi 20 prosentissa siitä ajasta, joka kuluisi täydellisen ratkaisun tuottamiseen (niin kutsuttu 80-20-sääntö). Menetelmässä otetaan huomioon myös todennäköisesti muuttuvat vaatimukset asiakkaan suunnalta, ja sen tarkoituksena onkin tuottaa ratkaisu, joka täyttää asiakkaan tämänhetkiset välittömät tarpeet sen sijaan, että yritettäisiin rakentaa ratkaisu, joka täyttäisi kaikki kuviteltavissa olevat tarpeet. (DSDM 2008, 8.)

Atern lupaa myös huomioida monia ketteriin menetelmiin kohdistuvia murheita; toisin sanoen, Atern on niin sanotusti suppeneva menetelmä, jossa projektin perustukset, mukaan lukien laajuus ja toimitettavan ratkaisun suurpiirteiset vaatimukset, sovitaan varhaisessa vaiheessa, ja yksityiskohtia lisätään myöhemmin. Menetelmässä vältetään jäykkää tarkkaa etukäteissuunnittelua, mutta myös täydellisen etukäteissuunnittelun puuttumisen aiheuttamia riskejä.

Pyrkimys on löytää näiden väliltä tasapainopiste, jossa tehdään ”juuri tarpeeksi” etukäteissuunnittelua. (DSDM 2008, 9; DSDM 2008, 14-15.)

Aikataulussa pysyminen on Aternin tärkeimpiä tavoitteita. Perinteisestä lähestymistavasta poiketen Atern-projektissa aikataulu ja budjetti lyödään lukkoon, eikä niistä tingitä. Myöskään tuotoksen laatu ei ole muuttuja, koska hyväksymiskriteerit sovitaan jo ennen kehityksen alkamista, joten ainoaksi muuttujaksi jää toimitetun ratkaisun ominaisuudet. Aternin ja perinteisen lähestymistavan eroa projektin muuttujiin on hahmoteltu kuviossa 6. (DSDM 2008, 14-15.)



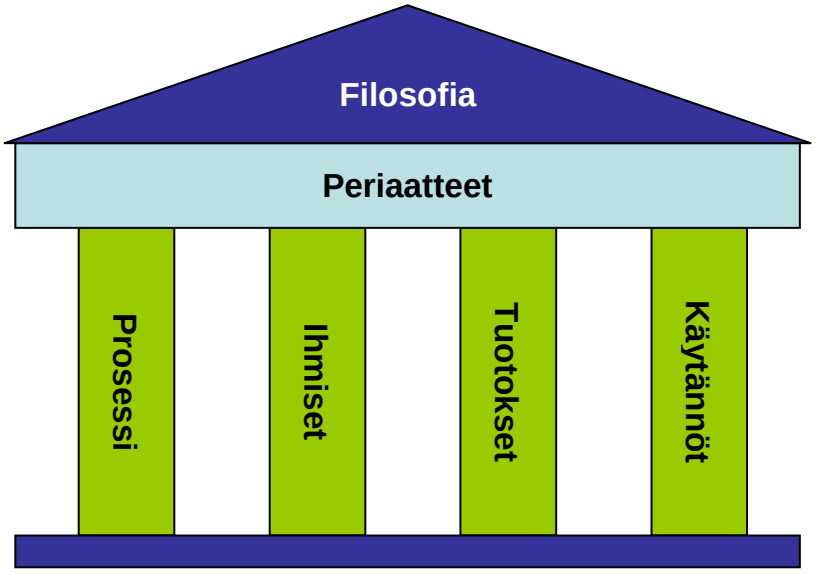
Kuvio 6. Projektin muuttujat: perinteinen vs. Atern (DSDM 2008, 14).

Atern-projektin lopputuloksen luvataan olevan aina elinkelpoinen. Mikäli kaikkien vaatimusten täyttäminen annetussa ajassa alkaa vaikuttamaan mahdottomalta, matalamman prioriteetin ominaisuuksia aletaan pudottamaan tai siirtämään myöhemmäksi kaikkien sidosryhmien hyväksynnän kera. Menetelmässä käytetään niin kutsuttua MoSCoW-priorisointia, johon palataan luvussa 3.5.2, ja mikäli sitä sekä menetelmän aikataulutussääntöjä käytetään oikein, projektin lopputuloksena luvataan syntyvän aina vähintään niin kutsuttu pienin

käytettävä osajoukko, eli tuotos, joka sisältää vähintään tärkeimmät eli korkeimman prioriteetin ominaisuudet. (DSDM 2008, 15.)

Kuten monet ketterät menetelmät, myös Atern vaatii projekti- ja organisaatiokohtaista sovittamista. Liiallinen muodollisuus hidastaa etenemistä ja voi jopa pysäyttää sen, kun taas liian vähäinen muodollisuus johtaa improvisointiin. Riskien arviointi suoritetaan projektin aikaisessa vaiheessa, tarkoituksena määritellä sopiva kurin määrä, jotta aikaa ei valu hukkaan ja jokainen projektin parissa suoritettu toimenpide kasvattaa tuotoksen arvoa. (DSDM 2008, 15.)

Aternin filosofia on, että jokainen projekti pitää kohdistaa selkeästi määriteltyjen strategisten tavoitteiden mukaisesti ja keskittyä todellisten liiketoimintahyötyjen aikaiseen toimittamiseen. Filosofian toteutumiseksi sidosryhmien avainjäsenten tulee ymmärtää liiketoiminnan tavoitteet ja heillä on oltava tarpeeksi valtuuksia sekä kykyä yhteistoimintaan pystyäkseen tuottamaan tarkoituksenmukaisen ratkaisun sovitussa aikataulussa liiketoiminnan asettamien prioriteettien mukaisesti, samalla hyväksyen välttämättömät muutokset, jotka kumpuavat kehitteillä olevan ratkaisun paremmasta ymmärtämisestä. Filosofian tukena on kahdeksan periaatetta, joita taas tukee elinkaari (prosessi), roolit ja vastuut (ihmiset), määritellyt tuotokset ja suositellut käytännöt; rakennetta on hahmoteltu kuvioon 7. Seuraavassa tarkastellaan näitä osasia tarkemmin. (DSDM 2008, 18.)



Kuvio 7. Aternin rakenne (DSDM 2008, 18).

3.1 Periaatteet

Kuten todettu, Atern määrittelee kahdeksan periaatetta, jotka tukevat sen filosofiaa. Periaatteet ohjaavat tiimiä omaksumaan oikean asenteen ja ajatusmaailman; periaatteista tinkiminen horjuttaa filosofiaa, jolloin Aternista ei saada täyttä hyötyä. (DSDM 2008, 20.)

Kahdeksan periaatetta ovat:

1. Keskity liiketoiminnan tarpeeseen.
2. Toimita ajallaan.
3. Tee yhteistyötä.
4. Älä koskaan tingi laadusta.
5. Rakenna inkrementaalisesti kestävien perustuksien päälle.
6. Kehitä iteratiivisesti.
7. Kommunikoi jatkuvasti ja selkeästi.
8. Osoita hallintaa.

(DSDM 2008, 20.)

Keskittymisellä liiketoiminnan tarpeeseen tarkoitetaan, että jokaisen projektissa tehdyn päätöksen tulisi lopulta johtaa siihen, että liiketoiminta saa projektista ajallaan sen, mitä tarvitsee; projekti on matka, ei määränpää. Atern-tiimien tuleekin ymmärtää liiketoiminnan prioriteetit, olla yhteydessä liiketoiminnan kanssa ja taata tulokseksi vähintään pienin käytettävä osajoukko. (DSDM 2008, 20.)

Ajallaan toimittaminen on usein tärkein tekijä onnistumisessa; myöhästyminen saattaa viedä koko projektin järjellisyyden, etenkin, mikäli markkinointitilaisuudet tai laissa säädetyt aikarajat ovat mukana kuvioissa. Siksi Atern-tiimit aikataulluttavat työnsä, keskittyvät liiketoiminnan prioriteetteihin ja osuvat aina määräaikoihin. (DSDM 2008, 21.)

Yhteistyö on tärkeää, sillä sitoutunut ja hyvin keskenään toimeen tuleva tiimi saa enemmän aikaiseksi kuin ryhmä yksilöitä ja on enemmän kuin osiensa summa. Periaatteen mukaisesti Atern-tiimit sisällyttävät oikeat sidosryhmät, mukaan lukien liiketoiminnan edustajia, oikeina aikoina halki projektin, omaavat riittävästi päätösvaltaa ja kehittävät hyvän tiimikulttuurin. (DSDM 2008, 21.)

Tuotoksen laatutaso sovitaan heti projektin alussa, ja kaiken työn tulisi tähdätä noiden kriteerien täyttämiseen, ei enempään eikä vähempään; ratkaisujen tulee olla ”tarpeeksi hyviä”. Mikäli liiketoiminta hyväksyy, että pienimmän käytettävän osajoukon toiminnallisuudet on toimitettu, ratkaisun tulee kelvata. Atern-tiimit pitävät huolta laadusta pitämällä yllä sopivaa suunnittelun, dokumennoinnin ja testauksen tasoa ja tarkastelevat tuotoksiaan jatkuvasti. (DSDM 2008, 21.)

”Rakenna inkrementaalisesti kestävien perustuksien päälle” on, kuten huomataan, melko kankea suomennos, mutta ajatus on selkeä: Aternissa, kuten monissa ketterissä menetelmissä, tähdätään siihen, että ratkaisu toimitetaan pala (inkrementti) kerrallaan liiketoiminnalle, jotta nämä pääsevät sekä antamaan palautetta että mahdollisesti saamaan jotain aikaista hyötyä. Kestävillä perustuksilla tarkoitetaan, että projektin laajuus ja suuren mittakaavan tavoitteet ovat tarpeeksi, mutta ei liian, tarkkaan selvillä ennen rakentamisen alkua. (DSDM 2008, 22.)

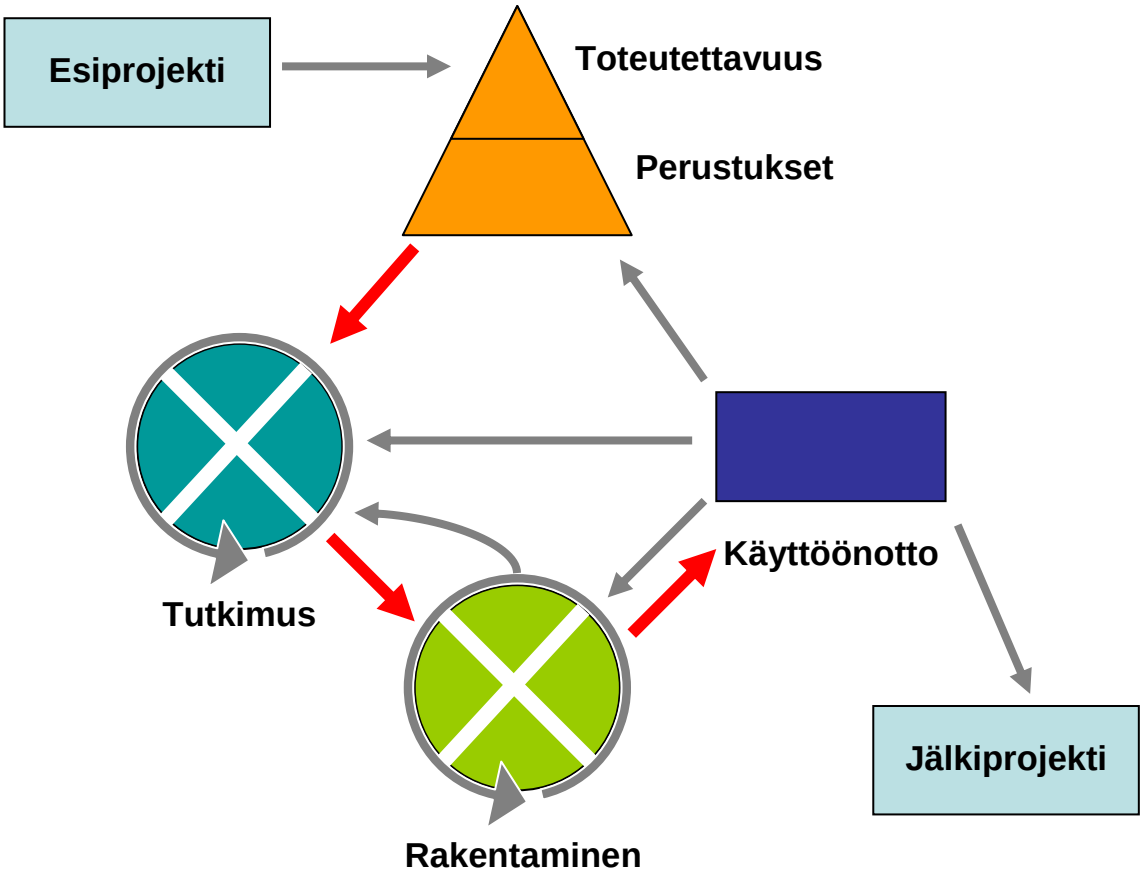
Iteratiivista kehittämistä käytetään, koska Aternissa hyväksytään, että mikään harvemmin tulee tehtyä oikein ensimmäisellä kerralla. Etukäteissuunnittelulla varmistetaan kestävät perustukset projektille, mutta yksityiskohtaisempi suunnittelu ja rakentaminen tapahtuu iteratiivisesti, asiakkaan palautetta kuunnellen. Muutos on väistämätöntä, joten Atern kehottaa XP:n tavoin hyväksymään sen; oikea ratkaisu ei synny ilman sitä. (DSDM 2008, 22.)

Aternin tekniikat on suunniteltu kehittämään kommunikaation tehokkuutta, sillä huono kommunikaatio on usein mainittu suurimmaksi syyksi projektien epäonnistumiselle. Atern-tiimit pitävät, jälleen XP:n tavoin, päivittäisiä seisomakokouksia, minkä lisäksi kehotetaan muun muassa käyttämään mallinnusta ja prototyyppejä asioiden esittämiseksi, pitämään dokumentaatio niukkana ja

ajankohtaisena ja keskustelemaan epämuodollisesti kasvotusten. Nämä tekniikat ovat Aternin mukaan tehokkaampia kuin suuret tekstidokumentit. (DSDM 2008, 23.)

On oleellista, että projektin langat pysyvät käsissä; Atern-tiimin tulee siis osoittaa hallitsevansa projektia – ja vieläpä proaktiivisesti. Siksi tarvitaan sopivaa määrää muodollisuutta etenemisen seuraamiseksi; etenemistä mitataan Aternissa toimitettujen tuotoksien mukaan suoritettujen toimien sijasta. Projektin suunnitelmat ja edistyminen tulee olla kaikkien nähtävillä, ja projektin elinkelpoisuutta arvioidaan jatkuvasti perustuen liiketoiminnan tavoitteisiin. (DSDM 2008, 23.)

3.2 Elinkaari



Kuvio 8. Aternin elinkaari (DSDM 2008, 30).

Atern-projektin elinkaari, jota on hahmoteltu kuvioon 8, on viidennen ja kuudennen periaatteen mukaisesti iteratiivinen ja inkrementaalinen. Ratkaisua ei toimiteta kertarysäyksellä, vaan sarjana lisäyksiä; näin kiireelliset liiketoiminnan tarpeet voidaan täyttää aikaisin ja toimittaa vähemmän tärkeät ominaisuudet myöhemmin. Liiketoiminnalla on myös mahdollisuus nähdä ja kommentoida keskeneräistä työtä ja tehdä muutospyyntöjä kesken projektin. (DSDM 2008, 30.)

Elinkaaren voi joustavasti vääntää moneen muotoon, kuten tämän luvun lopussa tullaan osoittamaan; se ehkä hieman selkeyttää kuvion 8 joka suuntaan osoittavia nuolia. Seuraavassa kuitenkin esitellään palaset, joista elinkaari koostuu.

Lyhyessä ja napakassa esiprojektivaiheessa yksinkertaisesti luodaan muodollinen ehdotus projektista ja asetetaan se asiayhteyteen muiden organisaation projektiportfolion osasten kanssa. Tarkoituksena on kuvata liiketoiminta-ongelma, johon halutaan ratkaisu, varmistaa projektin olevan liiketoiminta-strategian mukainen ja suunnitella ja resursoida toteutettavuus-vaihe. (DSDM 2008, 31.)

Toteutettavuus-vaiheessa tutkitaan, onko ehdotettu projekti toteuttamiskelpoinen liiketoiminnallisesta ja teknisestä näkökulmasta, ja kartoitetaan suurpiirteisesti mahdollisia ratkaisuja, kustannuksia ja aikatauluja. Tarkoituksena on tunnistaa ehdotetun ratkaisun hyödyt, mahdolliset reitit ratkaisuun pääsemiseksi, kuvailla projektin organisaatio- ja hallintomalleja, esittää alustava aikataulu ja budjetti ja suunnitella ja resursoida perustukset-vaihe. (DSDM 2008, 31-32.)

Perustukset-vaiheessa rakennetaan projektin perustukset: yhdistetään liiketoiminnan, ratkaisun ja johdon näkökulmat ja fokusoidaan projekti niin, että siitä tulee sekä jykevä että joustava – tai toisaalta lopetetaan projekti heti alkuunsa, mikäli se vaikuttaa liian riskialttiilta, liian kalliilta tai hyödyttömältä. Vaiheen on tarkoitus tuottaa suurpiirteinen mutta vankka näkymä projektin liiketoiminnallisista ja teknisistä puolista; tuotosten pitää olla vain sen tasoisia, että niiden

pohjalta voidaan siirtyä ensimmäiseen tutkimusvaiheeseen. Vaiheessa määritellään projektille asetettavat suurpiirteiset vaatimukset prioriteetteineen ja merkityksineen, kuvaillaan tarvittaessa ehdotetun ratkaisun tukemia liiketoiminta-prosesseja, identifioidaan ratkaisun käyttämä ja tuottama informaatio, kuvaillaan laadunhallinta-asiat, perustetaan projektille sopiva hallinto ja organisaatio, määritellään aikataulu ratkaisun kehittämiselle ja käyttöönotolle ja vielä kuvaillaan, arvioidaan ja otetaan hallintaan projektiin liittyvät riskit. Vaiheessa tarvitaan suurta liiketoiminnan panosta, joten liiketoiminnan edustajat tulee tunnistaa ajoissa ja sopia, millä tasolla he ovat mukana. (DSDM 2008, 32-33.)

Tutkimusvaiheessa iteratiivisesti ja inkrementaalisesti tutkitaan liiketoiminnan asettamia vaatimuksia ja käännetään ne alustaviksi ratkaisuiksi, joiden ei ole tarkoitus olla tuotantovalmiita, vaan osoittaa, että ratkaisu toimii. Tavoitteena on käsitellä perustuksissa laadittuja suurpiirteisiä vaatimuksia yksityiskohtaisemmin, tutkia liiketoiminnan tarpeet perinpohjaisesti ja toimittaa tarkat vaatimukset kehittyvälle ratkaisulle ja tarpeen vaatiessa luoda ratkaisusta malli, jolla voidaan osoittaa ratkaisun toimivuus; joka tapauksessa liiketoiminnan on hyväksyttävä alustava ratkaisu ennen siirtymistä rakentamisvaiheeseen. Tutkimusvaiheeseen tultaessa ympäristöjen (fyysiset, tekniset) tulee jo olla siinä kunnossa, että niitä voidaan hyödyntää ratkaisun kehittämiseksi, eli esimerkiksi ohjelmistoa kehitettäessä ohjelmointi- ja testausympäristöt. (DSDM 2008, 33-34.)

Rakentamisvaiheessa kehitetään tutkimusvaiheessa luotu alustava ratkaisu iteratiivisesti ja inkrementaalisesti täyteen toimintavalmiuteen, eli täyttämään sovitut hyväksymiskriteerit. Tässä vaiheessa kehitystyötä keskitetään ei-toiminnallisiin vaatimuksiin, kuten esimerkiksi järjestelmän suorituskykyyn ja turvallisuuteen – liiketoiminnan vaatimusten täyttäminenhän hoidettiin jo tutkimusvaiheessa – minkä lisäksi liiketoiminnan edustajien edelleen jatkuva mukanaolo mahdollistaa ratkaisun soveltuvuuden tarkastelun myös toiminnallisuuden näkökulmasta. (DSDM 2008, 34.)

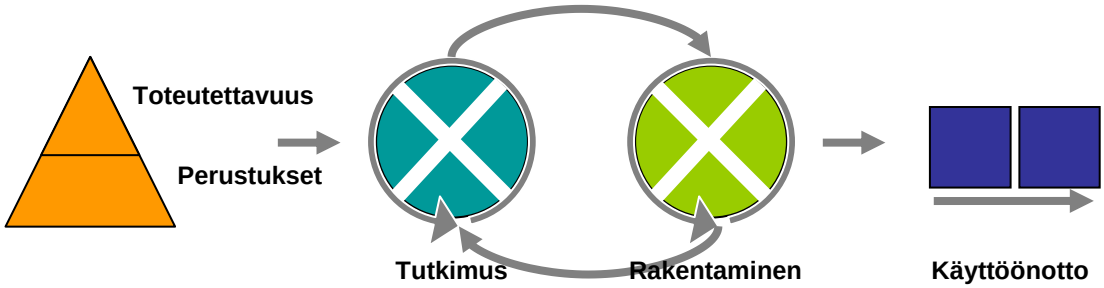
Käyttöönottovaiheen perimmäisin tarkoitus on siirtää ratkaisu todelliseen käyttöön, tai enemmän konkreettisten tuotosten ollessa kyseessä myyntivalmiuteen. Vaiheen toissijainen tarkoitus on toimia arviointipisteenä ennen lopullista käyt-

töönottoa tai tulevaa kehitystyötä. Tavoitteena on myös antaa käyttökoulutus ja/tai tarvittava dokumentaatio sekä ratkaisun loppukäyttäjille että ylläpito- ja tukihenkilöstölle, jotka tulevat huolehtimaan ratkaisun käyttökelpoisena pitämisestä. Se, montako kertaa käyttöönottovaiheeseen tullaan, riippuu siitä, onko liiketoiminnan järkevää ja mahdollista vastaanottaa ratkaisu pala kerrallaan – jos on, on sopivaa miettiä jokaisen toimituksen jälkeen, kannattaako projektia vielä jatkaa, sillä suuri osa ratkaisun hyödyistä saattaa jo olla toimitettu, eikä jatkaminen välttämättä enää kasvata investoinnin tuottoprosenttia; muistetaan 80-20-sääntö. Viimeisen käyttöönoton jälkeen projekti viedään muodollisesti loppuun. (DSDM 2008, 34-35.)

Jälkiprojektivaiheessa pohditaan ratkaisujen avulla todellisuudessa saavutetun liiketoiminta-arvon perusteella, kuinka projekti meni. Tämän arvioinnin tulisi alkaa heti, kun arvoa on mahdollista mitata, yleensä 3-6 kuukautta viimeisen toimituksen jälkeen. Inkrementaalisesti toimitettujen ratkaisun tapauksessa voi olla kohdallaan pohtia asiaa jo ennen viimeistä toimitusta, joka normaalisti on merkkipaalu tähän vaiheeseen siirtymiselle, ja syöttää palaute, eli muutos- ja kehitysehdotukset, takaisin edelleen pyörivälle projektille. (DSDM 2008, 35.)

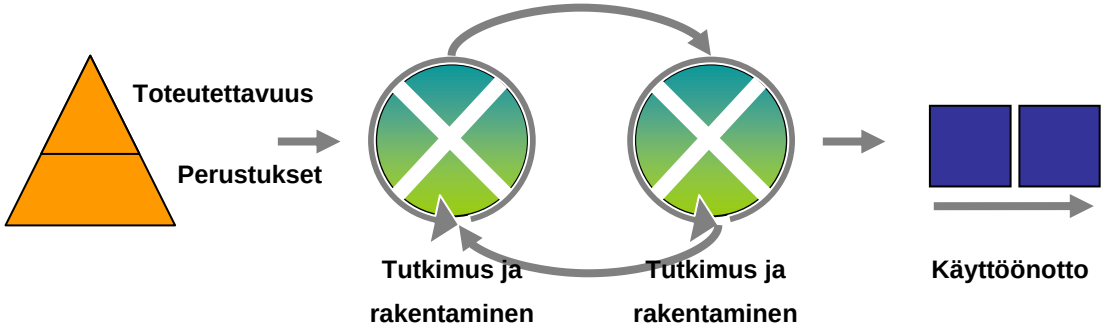
Kuten aikaisemmin mainittiin, Aternin elinkaari on taipuisa. Luvataan, että sen voi sovittaa sopimaan liki kaikenlaisiin projekteihin. Tyypillisesti Atern-projektit, esiprojektivaiheen jälkeen, alkavat toteutettavuus-vaiheesta, jota seuraa perustukset-vaihe; lyhyissä projekteissa nämä kaksi vaihetta voi yhdistää. Joissain tapauksissa myös tutkimus- ja rakentamisvaiheet saatetaan yhdistää. (DSDM 2008, 90; DSDM 2008, 92.)

DSDM-konsortion Atern-käsikirjassa esitellään (2008, 90-93) neljä erilaista esimerkkiä sovelletuista elinkaarista, ja paria niistä esitellään seuraavassa.



Kuvio 9. Esimerkki sovelletusta Aternin elinkaaresta (DSDM 2008, 90).

Kuvion 9 esimerkissä havainnollistetaan iteratiivista kehitystä, jossa ratkaisu syntyy usean tutkimus-rakentaminen –syklin aikana ennen yhden inkrementin käyttöönottoa. (DSDM 2008, 90.)



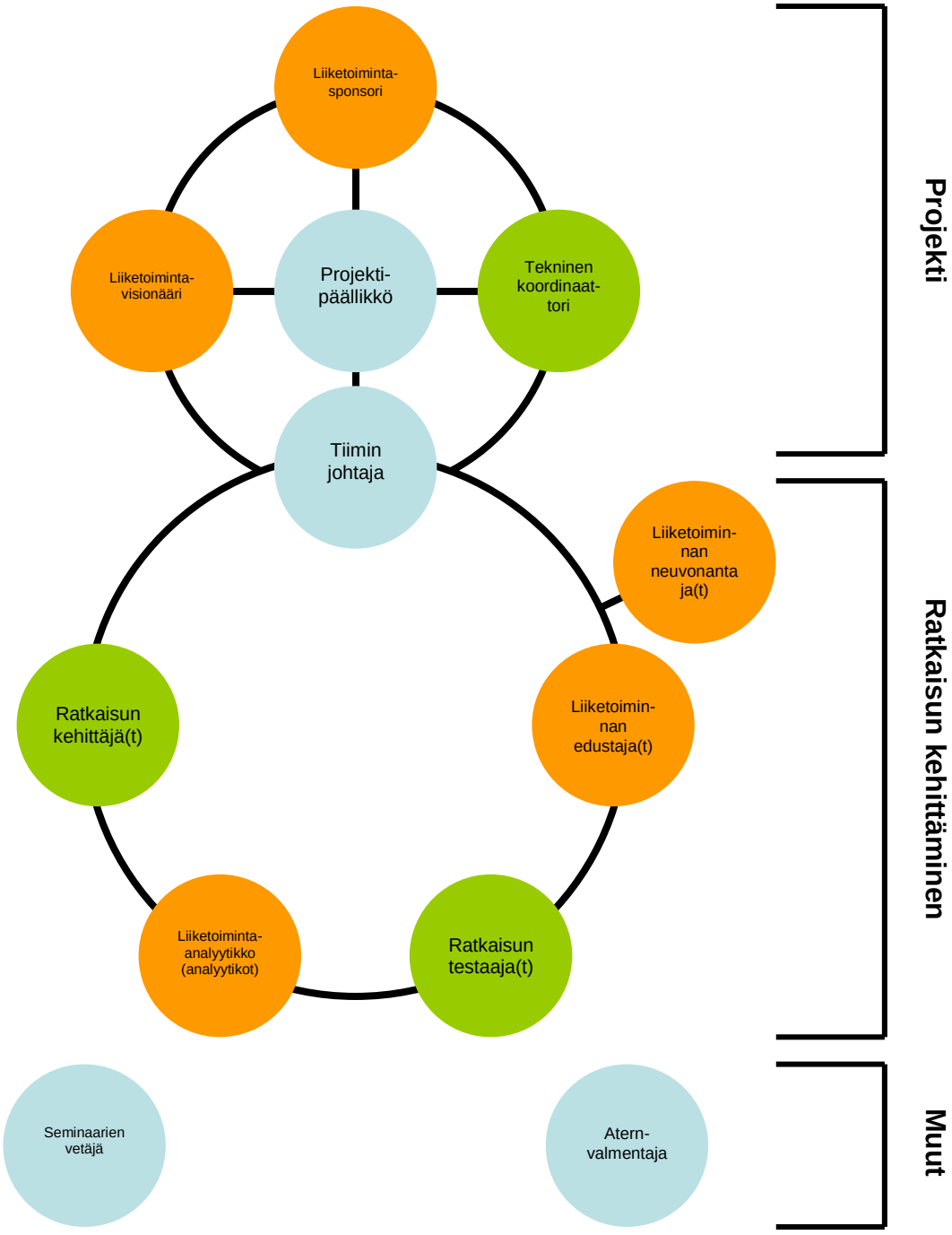
Kuvio 10. Toinen esimerkki sovelletusta Aternin elinkaaresta (DSDM 2008, 92).

Kuvion 10 toisessa esimerkissä tutkimus- ja rakentamisvaiheet on yhdistetty, jolloin yksi kokonainen osa ratkaisua valmistuu aina yhdellä läpikululla; tyypillisesti tällaista mallia käytetään pienissä projekteissa tai kun ratkaisussa vaadittava liiketoimintalogiikka on yksinkertaista (DSDM 2008, 92).

3.3 Roolit ja vastuut

Tehokkaasti yhteistyötä tekevät ihmiset ovat jokaisen onnistuneen projektin perusta. Aternissa tämä on ymmärretty ja siinä asetetaan jokaiselle projektin jäsenelle, asiakkaan ja toimittajan puolelta, selkeät roolit ja vastuut. Nämä kaksi

sidosryhmää työskentelevät hyvin lähekkäin, jotta muureja ei syntyisi. (DSDM 2008, 38.)



Kuvio 11. Aternin tiimimalli (DSDM 2008, 38).

Aternin tiimimalli on hahmoteltu kuvioon 11. Oranssit roolit ovat liiketoiminta-henkilöille, siniset Aternin projektinhallintahenkilöille ja vihreät roolit hoitavat

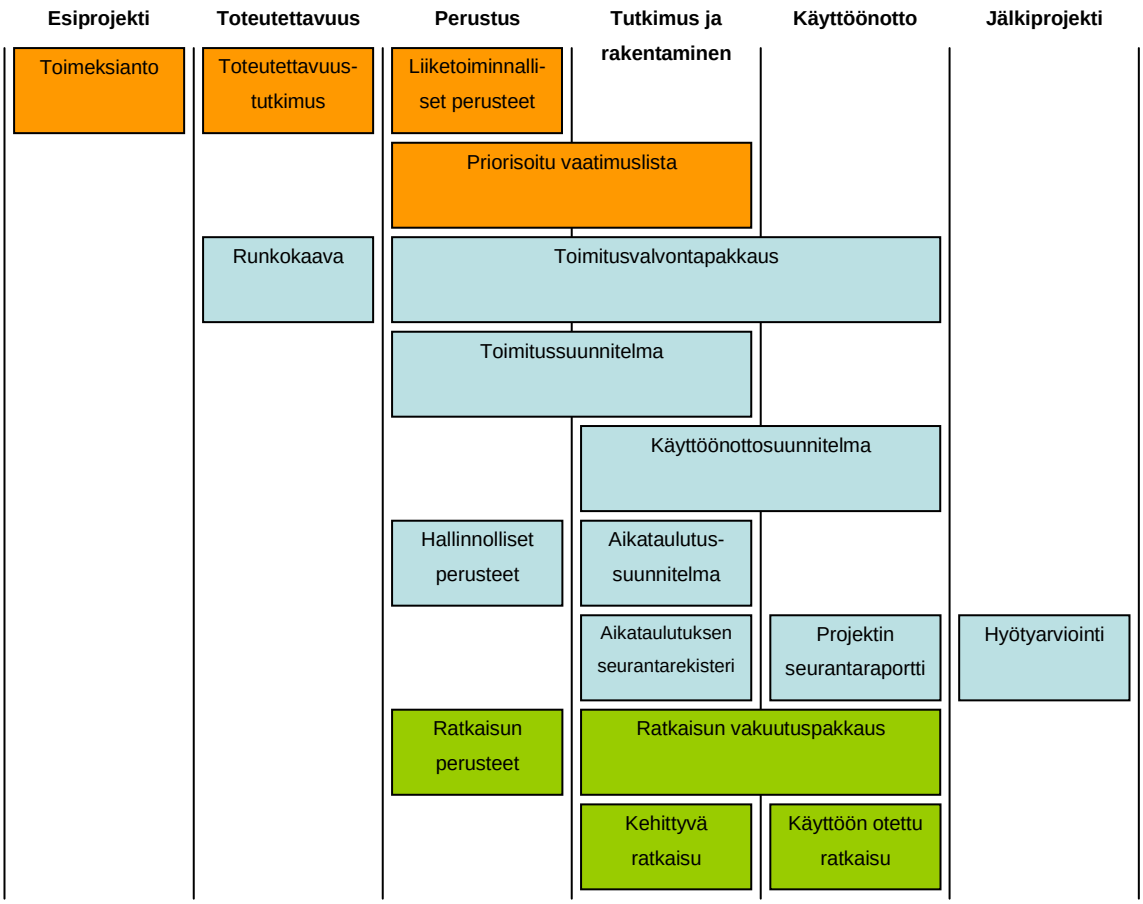
ratkaisun teknisen kehittämisen. Projektitason roolit hoitavat projektin ohjaamisen ja johtamisen; he joko kuuluvat projektin ohjausryhmään tai raportoivat suoraan sille. Kehitystason roolit ovat projektin ”konehuone”; he muotoilevat ja rakentavat ratkaisun ja ovat vastuussa siitä ja sen soveltuvuudesta liiketoiminnan käyttöön. Kehitysryhmiä voi olla useampiakin. Yksittäiset tiimin jäsenet raportoivat normaaleja kanavia pitkin ja myös oman etupiirinsä vanhemmille edustajille; esimerkiksi normaali raportointi kulkee tiimin johtajalle, mutta liiketoiminnan edustaja raportoi myös visionäärille ratkaisun soveltuvuudesta liiketoiminnan käyttöön ja ratkaisun testaaja raportoi tekniselle koordinaattorille ratkaisun teknisestä laadusta. Muut roolit, joihin lukeutuu myös liiketoiminnan neuvonantaja, ovat projektin tukihenkilöitä, jotka avustavat projektin etenemistä asiakohtaisesti ja tilapäisesti halki elinkaaren. (DSDM 2008, 38.)

Yksi rooli ei välttämättä tarkoita yhtä ihmistä, vaan projektin koosta riippuen yksi henkilö saattaa pitää useampaa hattua, tai toisaalta yksi rooli saattaa jakautua useamman henkilön kesken. Maantieteelliset seikat ja henkilökunnan määrä saattavat vaikeuttaa ihanteellisen tiimin luomista, mutta Atern suosittelee, että kaikkia rooleja tarkastellaan ja niiden vastuut jaetaan soveltuvasti. (DSDM 2008, 38-39.)

3.4 Tuotokset

Atern-projektissa syntyy jokaisessa elinkaaren vaiheessa joitain tuotoksia. Jotkut kuuluvat vain yhteen vaiheeseen, kun taas osa kehittyy peräkkäisten vaiheiden halki. Kaikkia tuotoksia ei tarvita kaikissa projekteissa, minkä lisäksi tuotoksien tarkka sisältö vaihtelee projektin ja organisaation mukaan; Atern ei suoranaisesti määrää mitään niiden sisällön suhteen, mutta antaa kyllä suosituksia. (DSDM 2008, 46.)

Kuviossa 12 on esitelty, mitä tuotoksia missäkin elinkaaren vaiheessa tuotetaan. Tässäkin kuviossa oranssi edustaa liiketoimintaa, sininen projektin hallinta-asioita ja vihreät tuotteet liittyvät kehitettävään ratkaisuun. (DSDM 2008, 46.)



Kuvio 12. Aternin tuotokset halki elinkaaren (DSDM 2008, 46).

Suuri osa tuotoksien nimistä kuvaa nähdäkseni riittävällä tarkkuudella, mistä niissä on kyse, ja kun niiden tarkkaa muotoa ei ole määrätty, en pidä tarpeellisena käydä kaikkia läpi. Osaa kuitenkin lienee järkevää hieman avata.

Liiketoiminnalliset perusteet –tuotos sisältää projektin onnistumisen kannalta elintärkeää tietoa liiketoiminnasta ja/tai liiketoiminnalle, minkä kaikkien sidosryhmien oletetaan ymmärtävän ennen ratkaisun kehittämisen aloittamisesta. Sisältöön voi kuulua esimerkiksi kuvaus siitä, miten liiketoiminnan tulisi toimia, mikä on tämänhetkinen toimintatapa, ja miten projekti auttaa saavuttamaan tavoitteen. (DSDM 2008, 48.)

Hallinnollisissa perusteissa kerrotaan, millainen on projektiorganisaatio ja miten sitä hallitaan, ja miten Aternin käytäntöjä käytetään projektissa. Siihen voi kuulua myös projektin tavoitteet ja onnistumiskriteerit (DSDM 2008, 51.)

Ratkaisun perusteissa on tietoa ratkaisusta, joka on tärkeää projektin onnistumisen kannalta; sisältöön voi kuulua esimerkiksi kuvailu järjestelmän arkkitehtuurista ja käytettävästä kehitystavasta, esimerkkinä ohjelmisto-tuotannossa noudatettavat ohjelmointistandardit ja –tyylit. (DSDM 2008, 49-50.)

Priorisoitu vaatimuslista kuvailee vaatimukset, jotka projektin tuotoksen on täytettävä. Se laaditaan suurpiirteisesti perustukset-vaiheessa ja se muuttuu yksityiskohtaisemmaksi tutkimus- ja rakentamisvaiheissa. **Aikataulutus-suunnitelmassa** kuvataan tarkemmin, mitä vaatimuksia missäkin vaiheessa yritetään toteuttaa. (DSDM 2008, 49; DSDM 2008, 54.)

Toimitusvalvontapakkaus (engl. ”delivery control pack”) sisältää projektin tilaan liittyvät raportit, dokumentit ja lokit. Sisältöön voisi kuulua esimerkiksi jaksottaiset raportit, joissa kuvaillaan ratkaisuun pääsyä, lasketaan tähänastista kulutusta ja tuodaan esille mahdollisia riskejä ja ongelmia, jotka saattavat vaikuttaa projektin laajuuteen tai aikatauluihin. (DSDM 2008, 51-52.)

Ratkaisun vakuutuspakkaus (engl. ”solution assurance pack”) sisältää elementtejä, joilla todennetaan kehittyvän ratkaisun olevan aukoton ja tarkoitukseen sopiva. Sisältöön voi kuulua esimerkiksi testiraportit sekä liiketoiminnan että tekniikan näkökulmista; esimerkiksi todisteet siitä, että ratkaisu toimii oikein ja on tietoturvallinen. (DSDM 2008, 52-53.)

3.5 Käytännöt

Tässä luvussa perehdytään hieman Aternin käytäntöihin, tuohon viimeiseen kuvioon 7 piirretyn kreikkalaisen temppelin tukipylvääseen.

3.5.1 Seminaarit

Organisaatioiden ja tiedon monimutkaistuessa ei ole enää mahdollista tai järkevää luottaa yhteen ihmiseen, joka tekee kaikki päätökset; siksi Atern tarjoaa omanlaisensa seminaarikäytännön, jota pidetään yhtenä koko menetelmän avainkäytäntönä. Seminaarien tavoitteena on saattaa ihmiset yhteen, jotta nämä yhteistyön avulla – tehokkuutta unohtamatta – saapuisivat sovittuun

maaliin, olkoon se sitten esimerkiksi ratkaisu johonkin ongelmaan, jonkin suunnitelman laatiminen tai vaatimusten kerääminen. Seminaareja voidaan pitää missä tahansa elinkaaren vaiheessa, kun olosuhteet ovat sellaiset, että asioiden tarkastelu useammasta näkökulmasta on hyödyllistä, esimerkiksi juuri suunnitelmia laatiessa. (DSDM 2008, 58.)

Seminaareista saatavia välillisiä ja välittömiä hyötyjä ovat muun muassa nopea ja korkealaatuinen, kaikkien olennaisten sidosryhmien sisällyttäminen päätöksentekoon, tiimihengen kohentuminen, konsensuksen saavuttaminen ja ongelmien selvittäminen. (DSDM 2008, 58-59.)

Seminaarin vetäjän, jonka tulisi olla puolueeton, tehtävänä on suunnitella ja muunnella prosessia, jonka avulla ryhmä saavuttaa maalinsa. Tähän voi käyttää lukuisia työkaluja, kuten vaikkapa fläppitaulua tai tarralappujen avulla hahmottelua, ja lukuisia tekniikoita, kuten aivoriihiä tai SWOT-analyysejä. Vetäjän tulee myös huolehtia ryhmädynamiikasta; minkään yhden henkilön tai ryhmän ei tulisi dominoida tilaisuutta, vaan mielipiteitä tulisi kysyä myös hiljaisemmilta ja ujommilta osallistujilta, eikä keskustelua tulisi päästää pois raiteilta, eikä varsinkaan antaa sen mennä henkilökohtaisuuksiin. (DSDM 2008, 59.)

Seminaarin vetäjän, jolla saattaa olla myös avustaja, lisäksi Aternissa on määritelty muutama muukin seminaareihin liittyvä rooli. Seminaarin omistaja on se henkilö, joka omistaa seminaarissa synnytettävän lopputuotteen; esim. aikataulujen suunnitteluseminaarin omistaja saattaa olla projektipäällikkö tai tiimin-vetäjä. Seminaariin osallistuja on henkilö, joka on kutsuttu seminaariin, koska hänen tietämystään, taitojaan ja kokemustaan tarvitaan maaliin pääsemiseksi, ja hänen Atern-tiimimallin roolinsa voi olla mikä tahansa, kuten vaikkapa liiketoiminnan edustaja tai ratkaisun kehittäjä. Tarkkailija on henkilö, joka ei osallistu keskusteluun tai päätöksentekoon, vaan on mukana joko auditoimassa seminaarin vetäjää tai seminaarityöskentelyä tai keräämässä itselleen oleellista taustatietoa; esimerkkinä sellainen ratkaisun kehittäjä, jonka kohtalona on vain noudattaa annettuja ohjeita pääsemättä itse vaikuttamaan päätöksiin. Kirjurin tehtävänä on kirjata ja julkaista seminaarin tuotokset ja päätökset olennaisen lisätiedon kera; kirjuri saattaa myös olla vastuullinen seuraamaan, että sovitut

asiat tulevat hoidettua. Seminaariin osallistujia, tarkkailijoita ja kirjureita voi olla useampiakin. (DSDM 2008, 59-61.)

3.5.2 MoSCoW-priorisointi

Atern-projekteissa, joissa projektin kesto on kiinteä, on elintärkeää ymmärtää asioiden suhteellinen tärkeys, jotta projekti etenisi ja pysyisi aikataulussa. Priorisoida voi vaatimuksia, tehtäviä, tuotoksia, käyttötapauksia, käyttäjätarinoita, hyväksymiskriteereitä, ohjelmistovirheitä, testejä ja oikeastaan ihan mitä vain. MoSCoW-priorisointitekniikka auttaa ymmärtämään, mikä on oleellista. Tekniikan nimi tulee kirjaimista MSCW, ja ne taas ovat lyhenteitä sanoista **M**ust have (pakko olla), **S**hould have (pitäisi olla), **C**ould have (voisi olla) ja **W**on't have this time (ei tule olemaan tällä kerralla). MoSCoWia on päädytty käyttämään Aternissa, koska näin prioriteetit ovat täsmällisempiä kuin vaikkapa epämääräiset ”korkea prioriteetti” tai ”matala prioriteetti”. (DSDM 2008, 64; DSDM 2008, 68.)

Siitä, mitä tarkalleen ottaen eri prioriteetit tarkoittavat, tulee sopia priorisointia tekevien kesken ennen MoSCoWin käyttöönottoa (DSDM 2008, 64). Seuraavassa tutkitaan hieman, mitä eri prioriteetit voisivat tarkoittaa.

”Must have”, pakko olla, on melko vahva ilmaus. Sitä suositellaan käytettäväksi vain tapauksissa, joissa kysymykseen ”mitä tapahtuu, jos tätä vaatimusta ei täytetä?” ainoa vastaus on ”projekti peruutetaan, sillä ratkaisussa ei ole mitään järkeä ilman tätä”. Tällaisia seikkoja voivat olla esim. se, että ratkaisu ei toimi ollenkaan ilman vaadittua ominaisuutta, se, että ratkaisu ei ole lainmukainen ilman ominaisuutta tai se, että ratkaisu ei ole turvallinen ilman ominaisuutta. Jos ominaisuuden puutteen pystyy kiertämään, olkoon se sitten miten työlästä tahansa, vaatimuksen prioriteetiksi tulee asettaa S tai C. (DSDM 2008, 64.)

Jo aiemmin mainittu pienin käytettävä osajoukko koostuu siis kaikista M-prioriteetin vaatimuksista, ja kaikki nämä vaatimukset kehitettävän ratkaisun on aivan välttämätöntä toteuttaa. (DSDM 2008, 64.)

”Should have”, pitäisi olla. Tällä tarkoitetaan jotain, joka on tärkeä, mutta ei elintärkeä; jotain, minkä pois jättäminen tekee kipeää, mutta ei tapa ratkaisua. Eroa Could have –prioriteettiin voi tehdä arvioimalla, kuinka paljon kipeää tekee eli millainen on vaikutus liiketoiminnan saamaan arvoon tai kuinka moni ihminen kärsii, jos vaatimusta ei täytetä. (DSDM 2008, 64.)

”Could have” eli voisi olla – nimensä mukaisesti jotain, jota haluttaisiin, mutta joka ei kuitenkaan ole niin tärkeää, eikä pois jättäminen satu niin paljon kuin S-prioriteetin tapauksessa. (DSDM 2008, 64.)

”Won’t have this time” –prioriteetin vaatimuksia ei edes pyritä täyttämään kyseessä olevan projektin aikana, mutta niistä pidetään kirjaa, jotta sama vaatimus ei hiipisi niin sanotusti takaoven kautta vaatimuslistalle. W:tä käytetään siis hillitsemään odotuksia. (DSDM 2008, 65.)

Kuten todettua, priorisoida voi myös hyväksymiskriteereitä. Esimerkkinä on käytetty varmuuskopiointia; jonkin järjestelmän pitäisi pystyä (S) tekemään varmuuskopiot alle neljässä tunnissa, mutta sen on aivan välttämättömästi tehtävä (M) ne alle 24 tunnissa. (DSDM 2008, 67.)

MoSCoW-priorisointi on tarkoituksenmukaista vain määritellyssä aikahaarukassa; vaikka jollekin ratkaisun osalle olisikin annettu prioriteetiksi Must have, sen prioriteetti ensimmäisessä toimitettavassa inkrementissä ei välttämättä ole M. Esimerkkinä mainitaan ohjelmisto, johon tarvitaan prioriteetilla M datan arkistointimahdollisuus; tämä ominaisuus pitää toimittaa projektin loppuun mennessä, mutta sen prioriteetti projektin alussa saattaa olla S tai C, koska ohjelmistoa pystyy todennäköisesti hyödyntämään muutaman kuukauden ilman tällaista ominaisuuttakin. Sekaantumisen välttämiseksi menetelmässä kehotetaankin käyttämään erillisiä prioriteettilistoja projektille ja iteraatioille. (DSDM 2008, 66.)

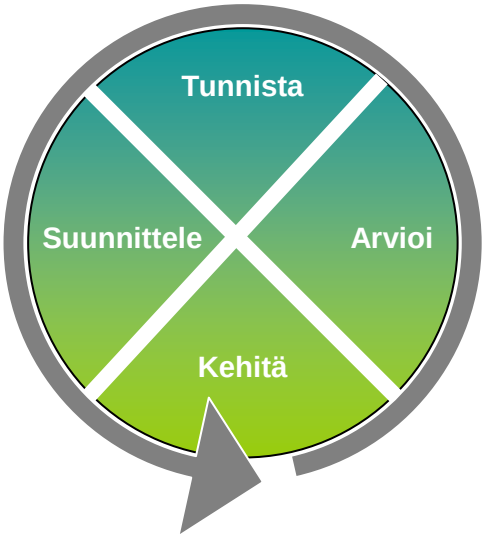
Peukalosääntönä mainitaan, että korkeintaan 60 % ratkaisun vaatimuksista saisivat olla prioriteettia M. M- ja S-prioriteetin vaatimusten yhteenlasketun summan tulisi olla korkeintaan 80 % kaikista vaatimuksista. Tarkoituksena ei tietenkään ole toimittaa pelkästään pienintä käytettävää osajoukkoa, mutta sen

toimittaminen on pystyttävä takaamaan, minkä lisäksi liiketoiminta voi perustellusti odottaa saavansa muutakin, jolleivät projektin olosuhteet ole jostakin syystä äärimmäisen haastavia. (DSDM 2008, 65-66.)

MoSCoW-priorisointi ja samalla koko Aternin perusajatus kiinteästä aikataulusta ja budjetista menettää kaiken järkevyytensä, mikäli kaikkien vaatimusten prioriteetti on M; menetelmän mukaan tällainen tilanne kieliikin enemmän siitä, että ratkaisulle asetettuja suurpiirteisiä vaatimuksia ei ole osattu jakaa osiin ja priorisoida näitä osasia niin, että ratkaisu on toimitettavissa annetussa aikataulussa ja budjetissa. (DSDM 2008, 67.)

3.5.3 Iteratiivinen kehitys

Aternin avaintekniikka suurpiirteisten ajatusten muuttamiseksi toimiviksi ratkaisuiksi on iteratiivinen kehitys. Kehittyvä ratkaisu on oleellisin tuotos, joka kehitetään tekniikkaa käyttäen, mutta tekniikan käsitteitä oletetaan sovellettavan suurimpaan osaan projektin tuotoksista. (DSDM 2008, 70.)



Kuvio 13. Iteratiivisen kehityksen vaiheet (DSDM 2008, 70).

Iteratiivinen kehitys tapahtuu sykleittäin, ja syklin neljä vaihetta ovat tunnista, suunnittele, kehitä ja arvioi, kuten on esitetty kuviossa 13. Tämä prosessi on olennainen osa aikataulutusta, jossa sillä varmistetaan sekä aikarajojen – jotka

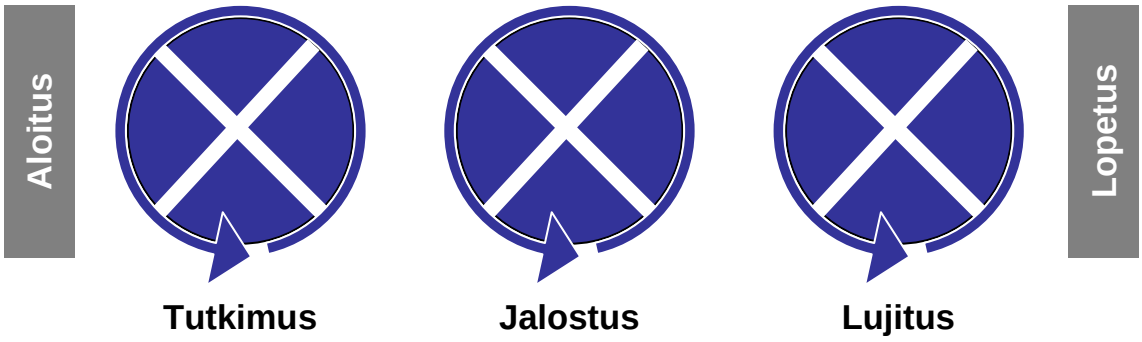
kehitysvaiheen aikataulujen tapauksessa ovat lyhyitä, tyypillisesti päivien tai jopa tuntien mittaisia – olevan hallittuja että palautesilmukan muodostuminen osaksi ratkaisun kehittämistä. Syklin mahdollistama palaute varmistaa oikean ratkaisun kehittymisen hallittavalla tavalla ajan myötä, ja sitä voi käyttää myös muiden Atern-tuotosten, esimerkiksi dokumenttien, tekemiseksi, joskin näissä tapauksissa syklit ovat yleensä pidempikestoisia. (DSDM 2008, 70.)

Tunnista-vaiheessa tiimi sopii, mikä on seuraava maali, johon tähdätään ja suunnittele-vaiheessa tiimi selvittää, mitä kenenkin pitää tehdä, jotta maaliin päästään. Kehitä-vaiheessa sovitut tehtävät tehdään sovitussa ajassa, ja arvioi-vaiheessa tehtävien tuloksia tutkitaan ja katsotaan, päästiinkö maaliin. (DSDM 2008, 70.)

Mikäli sykli johti tiimin maaliin, sen aikana tehdyt muutokset hyväksytään ja siitä tulee uusi pohjaviiva kehitettävälle asialle. Mikäli tarve on, sykli alkaa uudelleen uuden päämäärän kera. Mikäli maaliin ei päästy, tiimi voi joko hylätä tehdyt muutokset palaten takaisin viime pohjaviivalle (miettien mahdollisesti uuden lähestymistavan seuraavalle yritykselle) tai tunnistaa, mitä korjausliikkeitä on tehtävä, jotta maali saavutettaisiin seuraavan iteraation jälkeen. (DSDM 2008, 70.)

3.5.4 Aikataulutus

Atern suosittelee voimakkaasti kehitystyön aikataulujen (aikataululla, engl. ”timebox”, tarkoitetaan tässä yhteydessä siis jotain molemmista reunoista rajattua ajanjaksoa, joita voi olla ja onkin useampia peräkkäin – yksi aikataulu on siis tavallaan pieni osa koko projektin keston aikajanasta) jakamista kolmeen iteraatioon, jotka on esitetty kuviossa 14. (DSDM 208, 72.)



Kuvio 14. Aikataulun sisältämät iteraatiot (DSDM 2008, 72).

Aikataulu alkaa aina aloitustilaisuudella, jossa ratkaisun kehitystiimi tarkastelee aikataulun päämääriä ja varmistetaan, että aikataulussa on mahdollisuus pysyä, vetäen suunnitelmia uusiksi, jos ei ole; samalla selvitetään tarkkaan, miten paljon kehitystiimin jäsenet ovat saatavilla aikataulutettuihin töihin. Jokaiselle aikataulun aikana tehtävälle tuotokselle sovitaan myös hyväksymiskriteerit ainakin suurpiirteisesti. Aloitustilaisuuden tulisi kestää noin 1-3 tuntia. (DSDM 2008, 84.)

Tutkimusiteraatiossa, jossa siis pyörähdetään normaalisti iteraatiosyklin (tunnista, suunnittele, kehitä, arvioi) läpi, tarkastellaan kaikkia tuotoksia, jotka tämän aikataulun aikana pitäisi tuottaa ja määritellään tarkat vaatimukset – joko priorisoitua vaatimuslistaa muokkaamalla tai kirjoittamalla muodollisia tuotoksia – ja ratkaisuehdotukset niille, minkä lisäksi hyväksymiskriteerit määritellään, jos ne jäivät aloitustilaisuudessa epäselviksi. Ideaalitilanteessa tässä iteraatiossa valmistellaan myös jonkinlainen alustava prototyyppi ratkaisusta, jolla voidaan osoittaa vaatimusten oikeanlainen ymmärtäminen ja luoda ensivaikutelma ratkaisusta arvioitavaksi. Tutkimusiteraation tulisi peukalosääntönä viedä 10-20 % aikataulun kokonaisajasta. (DSDM 2008, 71; DSDM 2008, 85.)

Jalostusiteraatiossa tehdään aikataulun enin työ; sen tulisi viedä 60-80 % aikataulusta. Tavoitteena on tehdä iteratiivisesti niin paljon kehitystyöstä kuin mahdollista, testaaminen mukaan lukien. Jalostuksen tulisi alkaa nopealla epämuodollisella suunnittelukokouksella, jossa selvitetään, ketkä tiimin jäsenistä keskittyvät mihinkin tehtäviin ja missä järjestyksessä. Tehtävien

suoritusjärjestyksen tulisi noudattaa MoSCoW-priorisoitua aikataulun vaatimuslistaa, mutta järkevistä syistä, esimerkiksi tekniset syyt tai tarvitun henkilön poissaolo, siitä voidaan poiketa. Jalostuksen tulisi päättyä liiketoiminnan edustajien ja tarvittaessa muiden sidosryhmien kanssa suoritettuun arviointiin, minkä tarkoituksena on selvittää, mitkä toimet ovat tarpeellisia, jotta työ saataisiin loppuun hyväksymiskriteerien mukaiseksi; tämän pisteen jälkeen uutta työtä ei enää saa aloittaa tässä aikataulussa, ja myöskin suurien muutoksien pyytämisen kanssa tulisi olla hyvin tarkkana. (DSDM 2008, 72; DSDM 2008, 85.)

Lujitusiteraatiossa jalostuksen loppuarviossa sovitut tehtävät suoritetaan, samoin kuin kaikki organisaation tai projektin standardien täyttämiseksi tarvittavat työt. Kaikki tuotokset läpikäyvät viimeiset laaduntarkastukset. Ne tuotokset, jotka eivät täytä hyväksymiskriteerejä aikataulun loppuun mennessä, jätetään toistaiseksi toimittamatta. Lujituksen tulisi kestää tutkimuksen tavoin 10-20 % aikataulun kokonaiskestosta. (DSDM 2008, 72; DSDM 2008, 85.)

Lopetustilaisuudessa merkitään muodollisesti ylös kaikki aikataulun loppuun mennessä valmistuneet, hyväksymiskriteerit täyttävät, tuotokset. Tärkeää on myös selvittää, mitä tapahtuu tehtäville, jotka sovittiin osaksi aikataulua, mutta joita ei tehty valmiiksi; niitä saatetaan harkita seuraavaan aikatauluun, ne saatetaan ajoittaa johonkin vielä myöhemmäs tai ne saatetaan pudottaa rakenteilla olevasta inkrementistä tai koko projektista. Kaikkea ei-valmista ei ole aina suoralta kädeltä tarkoitus lykätä seuraavaan aikatauluun, mikäli koko projektin halutaan valmistuvan ajallaan; kuten todettua, priorisointi on oleellista. Lopetustilaisuudessa on myös tarkoitus tarkastella, miten sen lopettama aikataulu sujui, ja voiko siitä oppia jotain, joka helpottaisi kehitys- tai hallinnoimistyötä tulevaisuudessa. (DSDM 2008, 85.)

Vielä yksi asia aikataulutuksesta ja sitä myöten koko Aternista: XP:n tavoin Aternissakin tiimi kokoontuu joka päivä seisomakokoukseen. Idea on täsmälleen sama, ja toteutuskin melkein. Kokoukseen osallistuvat kaikki ratkaisun kehitystiimin jäsenet, joista jokainen vastaa kolmeen kysymykseen (mitä olet tehnyt sitten viime kokouksen, mitä aiot tehdä ennen seuraavaa kokousta ja

onko jotain ongelmia), kokouksen kesto on rajattu noin 15 minuuttiin ja se pidetään ringissä seisoen. (DSDM 2008, 86.)

4 PROJEKTIN KUVAUS

4.1 Vaatimukset järjestelmälle

Tässä luvussa kerrotaan rakennetun järjestelmän vaatimuksista, ja toteutusmenetelmistä – miksi ja miten se on tehty. Alkuun voi olla kuitenkin tarpeen kertoa hieman toimialasta T, jolla toimivalle yritykselle Y järjestelmä siis rakennettiin, jotta eräät järjestelmän piirteet olisi helpompi ymmärtää.

Toimialalle T ominaista on tehtävien töiden projektiluonteisuus ja se, että toimialan yritysten kiinteissä toimipaikoissa hoidetaan projektien suunnittelu ja hallinta, kun taas projektien suoritus tapahtuu asiakkaiden tiloissa vuorovaikutuksessa asiakkaiden kanssa. Tästedes tulen jaottelemaan Yritys Y:n toiminnot suunnittelu- ja tuotantopisteisiin, joista jälkimmäinen on selvästi merkittävämpi tämän järjestelmän kannalta, sillä järjestelmä on tarkoitettu apuvälineeksi juuri tuotantopisteisiin sekä Yritys Y:n työntekijöille että asiakkaille, ja vielä niin, että työnseurantaa tehdään toki vain Yritys Y:n työntekijöistä, ja asiakkaat pääsevät tarkastelemaan vain heille tarkoitettuja tiedotteita.

Tällaista työnseuranta- ja tiedotusjärjestelmää tarvitaan siksi, että tällä hetkellä monet toimialalla T toimivat yritykset ovat henkilöstömäärältään suppeita organisaatioita – johdon saama tieto voi olla vanhentunutta tai sävyttynyttä, minkä lisäksi osaavista työnjohtajista on pula. Toiminnoista tarvitaan kuitenkin ajantasaista ja oikeaa tietoa, tilastoja, ennusteita ja hälytyksiä, jotta tilanteisiin pystyttäisiin paremmin reagoimaan ja niitä pystyttäisiin ohjailemaan. Tätä tietoa välittämään tämä järjestelmä juuri rakennettiin.

Taulukossa 1 on pureuduttu vielä tarkemmin muutamaan toimialan yrityksissä vallitsevaan ongelmaan, joihin järjestelmästä halutaan ratkaisu. Lähteenä on käytetty Yritys Y:n edustajaa, eli siis tämän opinnäytetyön toimeksiantajaa.

Taulukko 1. Toimialan ongelmat ja tavoitteet järjestelmälle.

Ongelma	**Tavoite**
Tiedostoversiot: tikulla tai sähköpostilla välitettyä tiedostoa muokattu ennen seuraavaa päivitystä sekä tuotanto- että suunnittelupisteissä, jolloin tietojen yhdistäminen tai yhtäaikainen lukeminen edellisen tiedoston kanssa tuottaa ylimääräistä vaivaa ja ajanhukkaa.	Tuotantopisteistä kiinteät yhteydet verkkokansioon; järjestelmän on mahdollistettava tiedostojen muokkaaminen / uusien versioiden lataaminen verkkolevylle.
Reaaliaikaisuus: kun jokin asiakirja välitetään tikulla tai sähköpostilla, se pitää avata ja analysoida erikseen. Mikäli näin ei toimita, ongelmatilanteet saattavat jäädä huomaamatta.	Ylemmän johdon mahdollisuus seurata tilannetta ja saada raportteja/hälytyksiä automaattisesti reaaliaikaisesta tilanteesta tuotantopisteellä.
Tiedon kirjaaminen ja hyödyntäminen: tietoa kirjataan eri tuotantopisteillä eri tavoin ja eri paikkoihin johtuen sekä työnjohtajien kirjavista ATK-kyvyistä että ei-yhdenmukaisista syöttölomakepohjista. Näin kerättyä tietoa ei pystytä hyödyntämään ilman työläitä välilaskelmia sekä muuta tiedonkeruuta, joka on hidasta ja ei-mielekästä, jolloin se helposti unohtuu. Lisäksi manuaalisessa laskemisessa saattaa olla virheitä.	Luodaan helppokäyttöiset, nopeasti täytettävät ja yhdenmukaiset nettilomakkeet, joiden tiedot ajetaan tietokantaan. Luodaan valmiita ja itsemuokattavia hakuehtolauseita, joiden avulla voidaan hakea välittömästi haluttu tilasto.

Taulukko 1 (jatkoa)

Ongelma	**Tavoite**
Tiedottaminen: tuotantopisteet tuottavat erinäisiä asiakirjoja, joita pitää päivittää ja saattaa osapuolten tietoon. Päivityksiä tehdään, mutta jakelu ontuu. Valtaosa tuotantopisteiden tiedottamisesta tapahtuu suoraan asiakkaille. Johdon on vaikea olla ajan tasalla siitä, mitä on tiedotettu. Epäselvyyksiä viimeisestä versiosta on vähän väliä; päivitetyn tiedon lähettäjästä ei aina ole varmuutta, eikä siitä, kenelle se on lähetetty.	Osapuolille, mukaan lukien asiakkaat, annetaan projektikohtaiset tasoihin jaetut oikeudet lukea, päivittää ja poistaa eri asiakirjoja sekä tiedotuksia, toki niin, että asiakkaat saavat normaalisti vain lukuoikeudet, eikä niitäkään kaikkeen mahdolliseen. Asiakirjojen viimeisimmät versiot ovat aina saatavilla järjestelmän kautta. Kaikesta tiedottamisesta jää myös merkintä järjestelmään, jotta tiedotteiden jako- ja lukuajankohdat pysyvät tallessa.
Täsmätiedottaminen: tulee tapauksia, joissa tuotantopisteiden pitää tehdä nopeasti poikkeavia järjestelyitä, joista voi olla väliaikaista haittaa osalle asiakkaista. Aina tietoa ei saada asianosaisille ajoissa, jolloin vastineeksi tulee ikävää palautetta.	Asiakasrekisteriin luodaan täsmätiedottamista varten tietueita, joiden avulla asianosaiset saadaan seulottua nopeasti, ja joille voidaan näin tiedottaa poikkeusjärjestelyistä välittömästi esim. sähköpostilla tai tekstiviestillä.
Aikataulut: osa johdon antamista ohjeista ja deadlineista unohtuu työn tuoksinassa tai ne jäävät puolitiehen.	Järjestelmän tulee mahdollistaa deadlinejen asettamiset työtehtäville ja muistuttaa näistä projektikohtaisesti, minkä lisäksi sen tulee sisältää mahdollisuus antaa tiedotteita yrityksen johdolta tuotantopisteiden henkilökunnalle.

Taulukko 1 (jatkoa)

Ongelma	**Tavoite**
Reklamaatiot ja yhteydenotot: näistä ei välttämättä tehdä kirjauksia, vaan ne ovat täysin työnjohtajan vastuulla. Hän on myös ainoa, joka tietää niiden yksityiskohdat.	Reklamaatiot ja kaikki niihin liittyvät toimenpiteet (aikataulut, mitä sovittu, kuka vastuussa, seuranta, asiakkaan kuittaus toimenpiteestä) kirjataan järjestelmään, joka hoitaa tiedon välityksen ja muistuttaa unohduksista. Johto voi tarvittaessa puuttua asiattomiin vaateisiin.

4.2 Toteutustapa

Projektin läpiviemiseksi on käytetty hyväksi soveltuvaa ketterää menetelmää, tässä tapauksessa yksinkertaistettua versiota Aternista. Tämä menetelmä valittiin juuri siksi, että sen perusideana on tuottaa ratkaisu tietyssä aikataulussa niin täydellisesti kuin mahdollista; minulla oli rajallisesti aikaa käytettävissä ennen muihin haasteisiin siirtymistä.

Järjestelmän eri osioille asetettiin eri tärkeysasteita MoSCoW-menetelmällä, minkä jälkeen se suunniteltiin, toteutettiin ja testattiin iteratiivisesti ja inkrementaalisesti. Jokaiselle iteraatiolle ei asetettu tarkkaa aikarajaa tai vaatimuslistaa, vaan käytimme koko projektin kattavaa yhtä to do –listaa, mutta suunnilleen toiminnallisuus viikossa –tahdissa mentiin, hyvinä viikkoina syntyi kaksi toiminnallisuutta. Myöskin monista tuotoksista, joita Aternissa pitäisi tuottaa, luovuttiin, koska aikaa oli rajallisesti ja tekijöitä vähän, joten resurssit suunnattiin mieluummin itse ratkaisun kehittämiseen kuin vaikkapa yksityiskohtaisiin testiraportteihin – tämä ei toki tarkoita, etteikö järjestelmää silti olisi testattu. Monia tuotoksia syntyi silti itse järjestelmän lisäksi, esim. toimeksiantajan tekemä yleiskuvaus järjestelmästä sekä ketterästi laaditut yksityiskohtaisemmat suunnitelmat kustakin järjestelmän osasta, jotka toteutettiin

milloin milläkin tavalla, esimerkkinä Excelillä laadittuja käyttöliittymähahmotelmia ja tekstimuotoon kirjoitettuja toimintakuvauksia.

4.3 Käytetyt teknologiat

Järjestelmä on toimeksiantajan toiveesta ohjelmoitu PHP-kielellä – toimeksiantaja mainitsi jo alussa osaavansa kyseistä kieltä jonkin verran, ja toivoi sitä käytettävän, jotta hän itse pystyisi ylläpitämään ja jatkokehittämään järjestelmää. PHP:n lisäksi on käytetty JavaScriptiä eräiden selainpuolen ominaisuuksien luomiseksi; näihin sisältyy myös jokunen dynaaminen AJAXia hyödyntävä ominaisuus. Tavallisen JavaScriptin sekaan on otettu myös ripaus jQuery-kirjaston ominaisuuksia. Näiden lisäksi järjestelmässä hyödynnetään Flashia, mutta toimeksiantaja otti vastuun näistä ominaisuuksista, enkä näin ollen päässyt tai joutunut opettelemaan kyseistä teknologiaa, joskin moni Flash-olioille dataa syöttävistä PHP-koodeista on vähintään osittain omaa käsialaani.

Toimiakseen järjestelmä vaatii myös SQL-tietokannan, ja toteutuksen yhteydessä loimme yhdessä toimeksiantajan kanssa MySQL-tietokannan, jonka skeemaa rakensimme ja muokkasimme yhdessä muun toteutuksen tahdissa.

5 JÄRJESTELMÄN TOTEUTUS

”Arvostamme toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota”, todetaan ketterän ohjelmistokehityksen julistuksessa (Beck ym. 2001). Tätä periaatetta tuli ainakin noudatettua huolella, ja niinpä nyt jälkiprojektivaiheessa tätä kirjoittaessani huomaan unohtaneeni melko paljon siitä, mitä hektisten ohjelmointikuukausien aikana sainkaan aikaiseksi. Muistia on myös vaikea virkistää, sillä järjestelmä on otettu todelliseen käyttöön, enkä enää pääse sitä katsomaan, koska se sisältää nykyään ihan oikeaa dataa. Onneksi käytettävissäni oli kuitenkin joku sattumalta arkistosta löytynyt vanha versio sekä sen kanssa eri ajalta peräisin oleva, koodin kanssa osittain ristiriidassa oleva, tietokantavedos, joten sain edes jonkinlaisen ympäristön pystyyn. Tässä luvussa koitan näistä olosuhteista huolimatta parhaani mukaan hieman selittää, millainen käyttöliittymä järjestelmässä on ja miten järjestelmän tärkeimmät ominaisuudet, eli työnseuranta- ja tiedotusosiot, toimivat. Mitään täydellisiä tietokantamalleja tai lainauksia PHP-koodista ei ole luvassa. Aloitetaan kuitenkin hieman tarkemmalla kuvauksella siitä, miten Atern-menetelmää tuli sovellettua.

5.1 Atern-menetelmän soveltaminen

Atern on laaja menetelmä, vielä laajempi kuin luvussa 3 kuvattiin, joten projektissa, jossa on kaksi ihmistä, suuri osa sen ominaisuuksista oli käytännössä pakko jättää pois. Leikkuriin joutuivat kokonaisuudessaan roolit – seminaarien vetäjää ja Atern-valmentajaa ei tietenkään edes harkittu, ja jos loppuja haluaa jälkikäteen jakaa, niin itselläni taisivat olla kaikki ”vihreät roolit” (tekninen koordinaattori, ratkaisun kehittäjä, ratkaisun testaaja) ja toimeksiantajalla kaikki loput, minkä lisäksi myös hän kehitti ja testasi ratkaisua – kunnioitettavat yhdeksän hattua!

Tuotokset, kuten on jo aiemmin todettu, jäivät nekin aika vähiin. Kehittyvä ja käyttöön otettu ratkaisu tietenkin löytyivät, mutta muuten kirjallisia tuotoksia olivat lähinnä to do –lista (priorisoitu vaatimuslista) sekä kaikenlaiset käyttöliit-

tymähahmotelmat ja toimintakuvaukset. Keskustelimme päivittäin, tai oikeastaan lähes koko ajan, toimeksiantajan kanssa Skypessä, ja tavallaan tuo Skypeloki olikin jo itsessään melkoinen dokumentti, joka sisälsi toimintakuvauksia, testituloksia, projektin seurantaa ja ties mitä kaikkea – tosin onhan se näin jälkikäteen aika lukukelvotonta materiaalia.

Muodollinen seminaarikäytäntökin jätettiin pois, joskin joitakin tapaamisia pidimme toimeksiantajan kanssa kasvotusten erinäisissä ravitsemusliikkeissä, ja nuo tapaamiset olivat sangen hyödyllisiä etenkin siinä suhteessa, että niiden avulla opin paljon liiketoiminnan tarpeista ja käytännöistä – toimiala T kun ei ennestään ollut tuttu.

Mitä sitten käytettiin? Aika paljon kuitenkin. Projekti eteni elinkaaren mukaisesti, tietenkin; mikähän olisi sellainen ohjelmointiprojekti, jota esittämään Aternin elinkaarta ei saisi? Eihän tämä kaikessa rehellisyydessä oikeasti alkanut Atern-projektina, mutta voi ajatella, että tulin mukaan yhdistetyssä toteutettavuus- ja perustukset –vaiheessa, jossa toimeksiantaja esitteli tekemiään käyttöliittymähahmotelmia ja toimintakuvauksia ja minä vakuuttelin, että kyllä tuollaisen pystyy rakentamaan. Atern-mausteet keksin heittää mukaan vasta ripauksen myöhemmin, varmaan jossain perustuksista tutkimukseen johtavan nuolen kieppeillä, jos sen haluaa kaavioon väkisin sijoittaa. Siitä eteenpäin meno olikin sitten kolme kuukautta yhtä tutkimus-rakentaminen-käyttöönotto –sykliä, samaan tapaan kuin kuviossa 9 joskus kauan sitten esitettiin. Kolmen kuukauden työnteon jälkeen tuli viimeinen käyttöönotto, jolloin jätin projektille hyvästit ja lähdin hyvin ansaitulle hermolomalle. Ja nyt, kuukausia myöhemmin, koen olevani jälkiprojektivaiheessa kriittisesti arvioimassa tehtyä työtä.

Aikataulutuksesta tulikin jo aiemmin mainittua, että sitä ei kovin jäykästi noudatettu, mutta jotenkin luonnostaan päädyimme rytmiin, jossa toimeksiantaja kertoi maanantaiaamuna, mitkä toiminnallisuudet haluaa nähdä perjantaina valmiina, ja sitten ne olivat perjantaina valmiina, kun olin niitä ensin vähän miettinyt ja sitten koodannut – toisinaan vasta iltamyöhään, mutta perjantaina kuitenkin. Aloitus-tutkimus-jalostus-lujitus-lopetus. Hyvinä viikkoina tekemisen tahti oli aikataulutettua nopeampaa, ja silloin tietysti tehtiin muuten

tulevaisuuteen varattuja asioita; huonoina viikkoina taas aikataulu uhkasi venyä, mutta jotenkin sitä sai useimmiten kirittyä.

MoSCoW-priorisointia käytettiin myös hyödyksi, ja hyötyä siitä tosiaan olikin. Oli mukavaa, kun oli koko ajan perillä siitä, mikä on oikeasti tärkeää, ja myös siitä, mitä kaikkea on vielä tekemättä; eipähän ainakaan liiallinen hyvänolontunne päässyt hiipimään puseroon – ainakaan kovin usein.

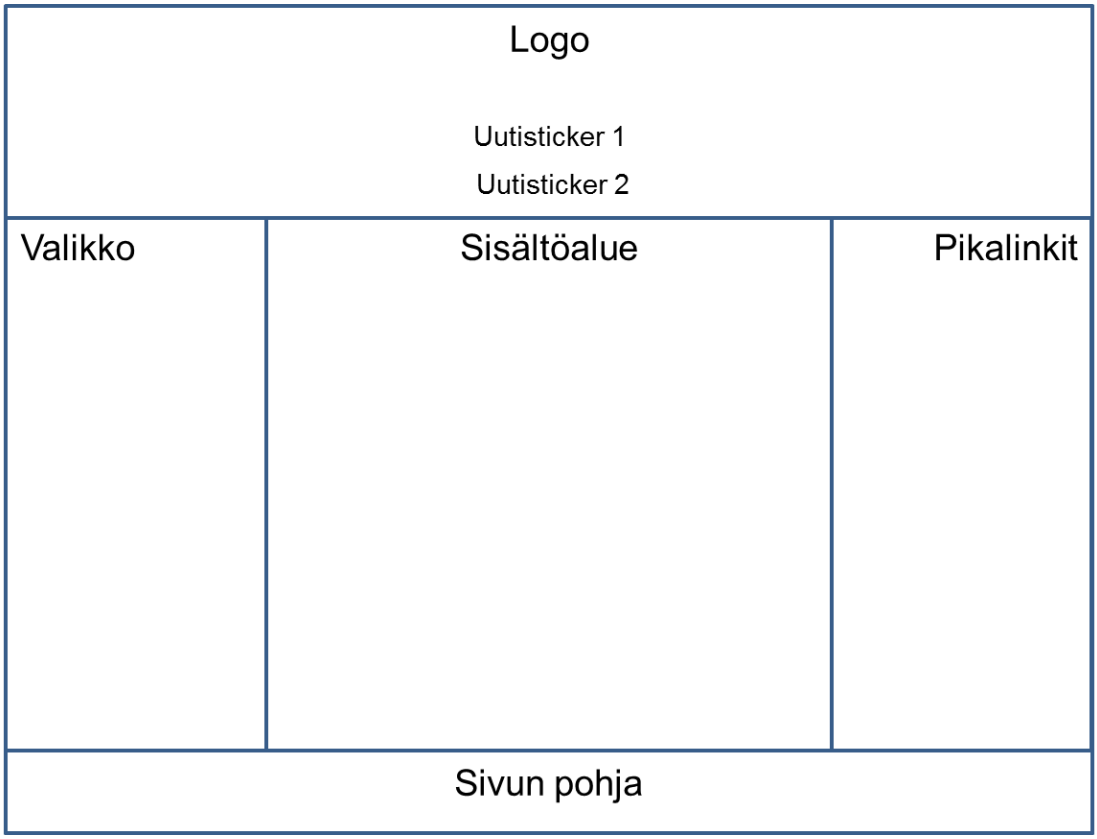
Iteratiivinen kehitys on se menetelmän osa, jota kaikkein eniten tuli käytettyä. Hyvä on, en missään vaiheessa julistanut suurellisesti olevani syklin ”tunnista”- tai ”kehitä”-vaiheessa, mutta kyllä nuo kaikki neljä syklin vaihetta – tunnista, suunnittele, kehitä, arvioi – tuli aina käytyä läpi, kuten luonnollista onkin. Syklit olivat välillä todella nopeita, kuten kehitysvaiheessa kuuluukin; katsoin, millainen toiminnallisuus pitää tehdä ja jaoin sen palasiin, jotka sitten yksi kerrallaan suunnittelin, kehitin ja arvioin. Yksi toiminnallisuus siis syntyi aina monen iteraation jälkeen. Käyttöönottovaiheessakin käytiin taajaan, sillä julkaisupolitiikkani oli ladata keskeneräinen kooditiedosto aina jokaisen erillään testattavaksi aikomani muutoksen jälkeen palvelimelle. Lopulta sinne päätyi versio, joka oli toimiva ja täytti vaatimukset.

Minkäänlaisia rajoittavia koodausstandardeja ei projektissa otettu käyttöön. Syitä on lukuisia; ei ollut aikaa omaksua sellaisia, ei ollut aikaa tutkia vaihtoehtoja ja sitten se todellinen syy, eli ei vain yksinkertaisesti käynyt mielessäkään. Niinpä koodista tuli paikoitellen melkoista spagettia, mutta ainakin se toimii. Ja kun PHP:sta ja JavaScriptistä on kyse, niin ehkä se annetaan anteeksi; ainakin oma näppituntumani on, että niillä helposti syntyy melko epämääräistä koodia vähän jonkun antiikkisen Basic-koodin hengessä, eikä niinkään mitään selkeämmin jäsenneltyä ja tyypitettyä tavaraa, jota vaikka Javalla saa (on pakko saada) aikaiseksi.

5.2 Käyttöliittymä

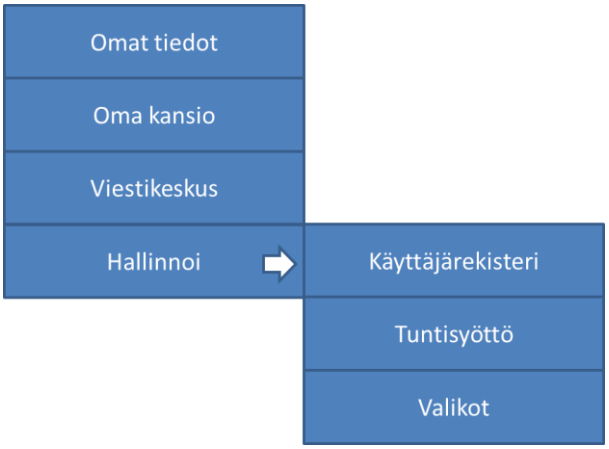
Järjestelmä toimii niin, että sisäänkirjautumisen jälkeen käyttöliittymä – valikko, linkit, sen sellaiset – pysyy vakiona, mutta kulloinkin tarvittava alasivu ladataan

sivun keskiosaan omalle sisältöalueelleen. Käyttöliittymää on hahmoteltu kuvioon 15.



Kuvio 15. Järjestelmän käyttöliittymä.

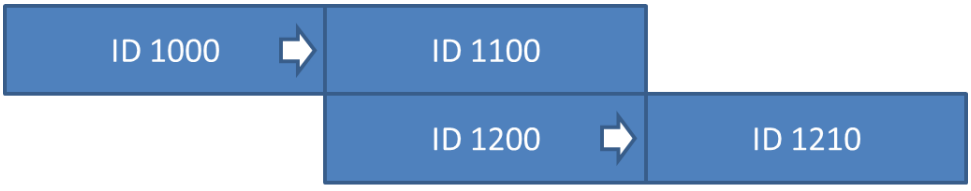
Pikalinkkeihin kuuluvat muun muassa linkki postilaatikkoon, johon palataan, kuten uutistickereihinkin, luvussa 5.4, linkki projektin vaihtamiseen (järjestelmässä voi olla kirjautuneena sisään vain yhteen projektiin kerrallaan) sekä uloskirjautumislinkki.



Kuvio 16. Valikon rakenne.

Vasemman laidan valikko, josta osa on hahmoteltu kuvioon 16, on toteutettu käyttämällä valmiina netistä löydettyä javascript-koodia (Dynamic Drive 2011), ja siinä voi olla loputtomasti sisäkkäisiä valikoita. Valikon sisältö luetaan tietokannasta, joka mahdollistaa sen, että yrityksen Y eri projekteilla voi olla jokaisella omanlaisensa valikko; valikon kohteet saa myös näkymään jokaiselle projektille. Samoin tietyt kohteet saa näkymään vain vähintään tietyn tasoisille käyttäjille, jolloin yritys Y:n työntekijät voivat säilyttää järjestelmässä esimerkiksi sisäisiä asiakirjojaan ilman, että asiakkaat pääsevät niitä katsomaan; oletuksena valikon kohteet toimivatkin linkkeinä erinäisiin tiedostolistauksiin, mutta niihin saa myös upotettua minkä tahansa linkin. Valikkojen luomista ja muokkaamista varten rakensin oman alasivunsa, jotta kaikkea ei tarvitsisi tehdä suoraan tietokantaan.

Tietokannassa valikot näkyvät niin, että jokaista valikkokohdetta varten on oma rivinsä, ja kullakin valikolla on ID-numero (esim. 1000, 2000…) sekä ”ylävalikon ID”; mikäli ylävalikon ID on 0, valikkokohde näkyy aina, eli se on ylimmän tason kohde, mutta mikäli se on jotain muuta, niin valikkokohde näkyy määritellyn valikkokohteen alavalikkona. Näin esim. valikko 1100, jonka ylävalikko-ID on 1000, näkyy valikon 1000 alavalikkona. Asiaa on hahmoteltu kuvioon 17. ID-kenttien lisäksi valikolla on tietenkin myös muita ominaisuuksia, kuten vaikkapa käyttäjälle näkyvä teksti.



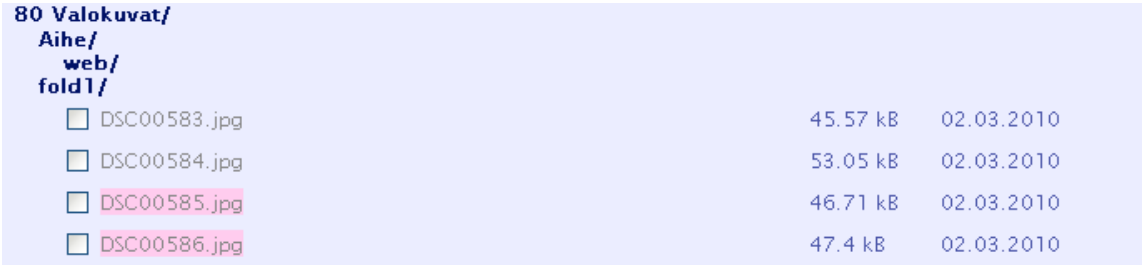
Valikon ID	Ylävalikon ID
1000	0
1100	1000
1200	1000
1210	1200

Kuvio 17. Valikon rakenne tietokannassa.

Jokaiselle valikkokohteelle on määritelty, minkä tasoiset käyttäjät näkevät sen, mille projektille se näkyy sekä linkki, mikäli niin halutaan; kuten sanottu, muuten valikon kohde toimii linkkinä tiedostolistaukseen.

Järjestelmään on siis mahdollista tuoda mitä tahansa tiedostoja, kuten kuvia tai asiakirjoja. Ne voi tuoda joko liitetiedostoina kirjoittaessaan viestiä (tähänkin palataan luvussa 5.4), tuoda omaan kansioonsa tai sitten viedä FTP:llä palvelimelle jonkin projektin kansioon. Se, että nämä ovat kolme ainoaa tapaa, oli toimeksiantajan asettama vaatimus; yrityksen Y puolella järjestelmää on tarkoitus käyttää yhdessä jonkin sovelluksen kanssa, joka mahdollistaa FTP-palvelimen käyttämisen ikään kuin Windowsin levyasemana, esimerkkinä NetDrive, ja asiakkaiden taas ei ole juuri tarvis tiedostoja järjestelmään viedä.

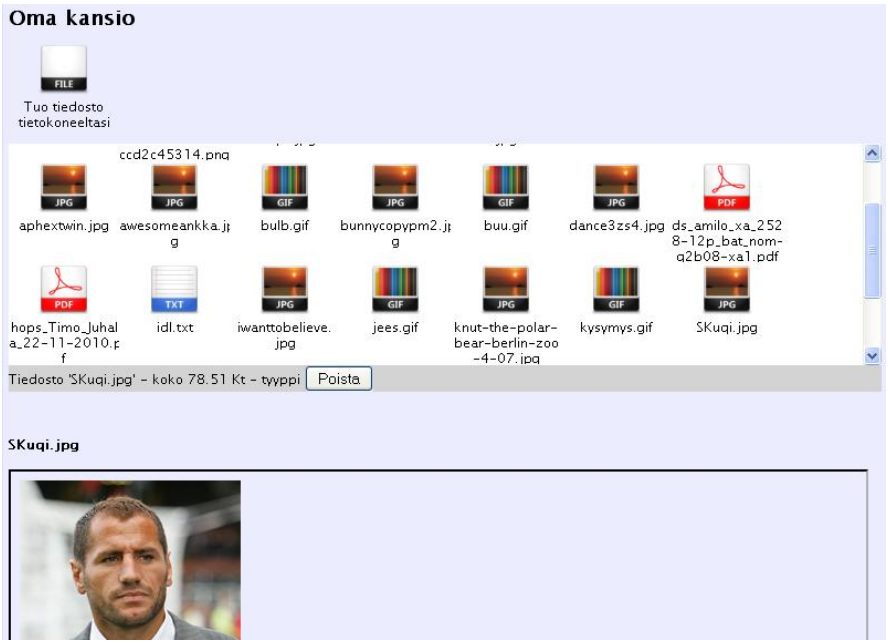
FTP:llä tuodut tiedostot täytyy ensin rekisteröidä järjestelmään, jota varten on oma alasivunsa, joka näyttää kaikki valitun projektin tiedostot ja sen, mitkä niistä ovat jo rekisteröityjä ja mitkä rekisteröimättä; ks. kuva 3, jossa punaisella merkatut tiedostot ovat rekisteröimättä. Rekisteröinnillä tarkoitetaan sitä, että tiedoston nimi ja vaaditut käyttäjätasot sen avaamiselle ja muokkaamiselle tallennetaan tietokantaan, kuten myös se, minkä valikkokohteen alle tiedosto kuuluu.



Kuva 3. Tiedostojen hallintaa järjestelmässä.

Kun tiedostoille on määritelty käyttäjäoikeudet ja valikkokohde, niitä voi tarkastella valikosta käsin. Kuten mainittu, valikkokohde, jota ei ole erikseen määritelty linkiksi, aukeaa tiedostolistaukseen, jossa kaikki valikkokohteeseen yhdistetyt

tiedostot näytetään nimineen pieninä kuvakkeina, joita klikkaamalla tiedostot aukeavat. Osa tiedostoista, kuten kuvat, PDF-tiedostot ja tavalliset tekstitiedostot, aukeavat suoraan järjestelmään tiedostolistauksen alapuolelle, ja lopuista aukeaa selaimen tavanomainen ”lataa tiedosto” –ikkuna. Kuvassa 4 on kuvaruutukaappaus omasta kansiosta, jonka tiedostolistaus toimii täsmälleen samalla periaatteella, tosin kuvassa näkyvä ”Tuo tiedosto tietokoneeltasi” –toiminnallisuus on siis vain omassa kansiossa.

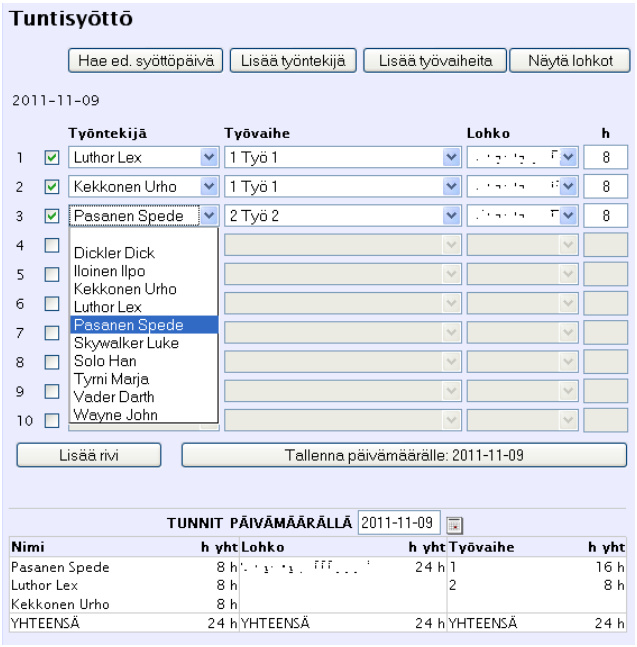


Kuva 4. Tiedostojen listaus omassa kansiossa.

Luonnollisesti järjestelmässä on muitakin alasivuja kuin pelkkiä tiedostolistauksia, mutta valtaosa niistä on varsin tavanomaisia lomakesivuja, eikä niissä siis ole mitään teknisesti tai ulkonäöllisesti kovin erikoista. Kaikkiaan erilaisia alasivuja on 17 ja erinäisiä minun tekemiäni PHP-skriptejä yhteensä 89 kappaletta – monet alasivut koostuvat useammasta PHP-tiedostosta, minkä lisäksi AJAX-toiminnallisuuksia on hajautettu omiin skripteihinsä. Kaikkiaan tuotin yhteensä hiukan yli 740 000 merkkiä eli monia tuhansia rivejä PHP:ta, HTML:ää ja Javascriptiä. Tärkeimpiin asioihin, eli työnseurantaan ja tiedottamiseen, liittyviä alasivuja tarkastellaan seuraavissa alaluvuissa 5.3 ja 5.4.

5.3 Työnseurantaosio

Työnseuranta on järjestelmässä jaettu kahteen osioon, tuntisyöttöön ja toteutumansyöttöön. Tuntisyöttö tehdään erikseen jokaiselle työntekijälle työnjohdon toimesta, kun taas toteutuma syötetään yleisesti, eli kirjataan, mitä koko projektiryhmä on saanut aikaiseksi. Kummassakaan näistä ei ole sinänsä teknisesti mitään kovin ihmeellistä, vaan molemmat ovat omia alasivujaan, jotka on toteutettu tavallisina HTML-lomakkeina.



Kuva 5. Tuntisyöttölomake.

Tuntisyöttölomake on esitetty kuvassa 5, ja se lienee melko selkeä. Laittamalla ruksin ruutuun tuntisyöttörivi aktivoituu, jolloin pudotuslistoista voi valita työntekijän, tämän suorittaman työvaiheen ja lohkon eli osaprojektin, jonka parissa työ on suoritettu, minkä lisäksi syötetään työn parissa menneet tunnit. Lomakkeen alapuolella on yhteenveto valitun päivämäärän tunneista sekä kalenteri, josta päivämäärää voi vaihtaa. Erilaisia internetistä löytyviä valmiita kalentereita tuli seulottua läpi useita, mutta lopulta löytyi todella hyvä (Ortega-Hernández 2009), jonka otin käyttöön tällä ja jokaisella muulla päivämäärien syöttöä vaativalla alasivulla. Aiemmille päivämäärille syötettyjä tunteja voi muokata, mikäli uutta toteutumaa ei ole syötetty kyseisten tuntien syöttämisen jälkeen, minkä lisäksi ”Hae ed. syöttöpäivä” –nappia painamalla järjestelmä hakee viimeksi syötetyt tunnit valmiiksi tallennettavaksi nykyiselle päivälle – kätevää, mikäli samat työntekijät tekevät samoja töitä monta päivää putkeen. ”Lisää rivi” –nappi luonnollisesti lisää taulukkoon uuden tuntisyöttörivin käyttäen javascriptin insertRow–, insertCell– ja appendChild–funktioita.

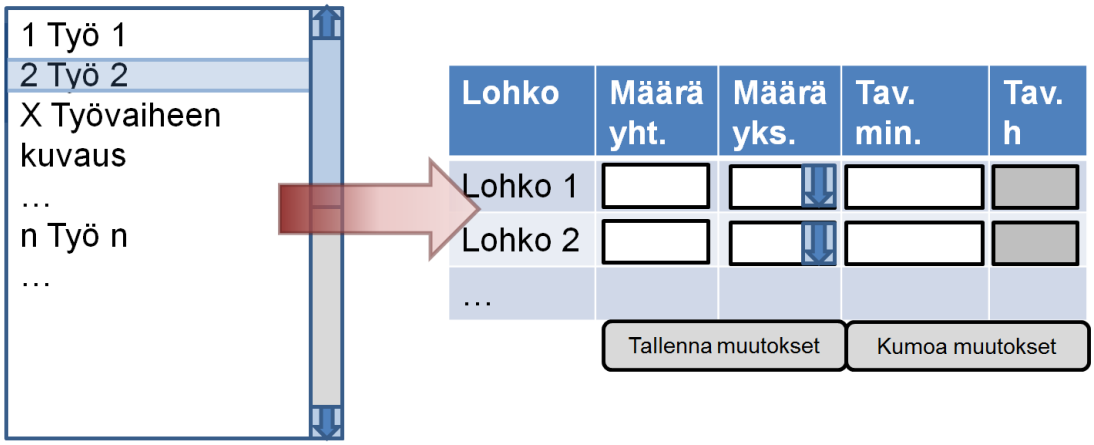
Lohko	Työvaihe	Edellinen toteutuma				h	yks	%
(lohko)	Työ 1	2011-11-08	3/6	(yksikkö)	50 %	16		
(lohko)	Työ 2	2011-11-08	29/76	(yksikkö)	38 %	8		

Kuvio 18. Toteutumansyöttölomake.

Toteutumansyöttölomaketta on hahmoteltu kuvioon 18. Siinä taulukkoon listataan lohkoittain kaikki työvaiheet, joille on syötetty tunteja edellisen toteutumansyötön jälkeen ja näytetään myös oleelliset tiedot viime toteutumansyötöstä, eli kuinka paljon mitäkin työtä silloin oli saatu aikaiseksi, sekä kuinka monta tuntia kullekin työlle on sittemmin syötetty. Jokaisen rivin kahteen viimeiseen ruutuun tulevat syöttöruudut, joihin nykyisen toteutuman voi syöttää joko yksikköperustaisesti tai prosenttilukuna, mikä nyt kulloinkin on kätevämpää. Mikäli jollekin lohkolle tai työvaiheelle on asetettu jokin hälytystaso, kyseinen rivi näkyy eri värisenä tässä näkymässä, mikäli hälytystaso on saavutettu.

Lohkojen muokkaamiselle ja työvaiheiden yhdistämiselle lohkoihin, eli kuinka monta tuntia varataan millekin työlle per lohko ja kuinka monta yksikköä kutakin työtä on, on omat alasivunsa, joiden yhteydessä edellä mainittuja hälytystasoja voidaan myös asettaa. Käytännössä hälytykset toimivat niin, että jollekin työvaiheelle valitaan joku tuntiraja, esimerkiksi 16 tuntia; mikäli tämä tuntimäärä täyttyy tai ylittyy tuntisyötön yhteydessä, toteutumansyöttölomakkeessa kyseisen työvaiheen rivit näkyvät eri värisinä. Samoin lohkoille voi asettaa hälytysrajoja; jokaiselle työlle määritellään tietty tuntimäärä per lohko, jonka työn suorittamiseen lasketaan kuluvan, joten voidaan määritellä tietty tuntimäärä, jonka täytyttyä tuntisyötössä kyseisen lohkon rivit näkyvät jälleen eri värisinä huomiota herättäen toteutumansyötössä. Tunteja on toki mahdollista syöttää lisää vaikka hälytysrajat tai ideaalit maksimituntimäärät olisivatkin jo ylittyneet; muutoinhan ylimenevistä tunneista joutuisi pitämään kirjaa jossain muualla, mikä taas sotisi koko järjestelmän hyötyä vastaan.

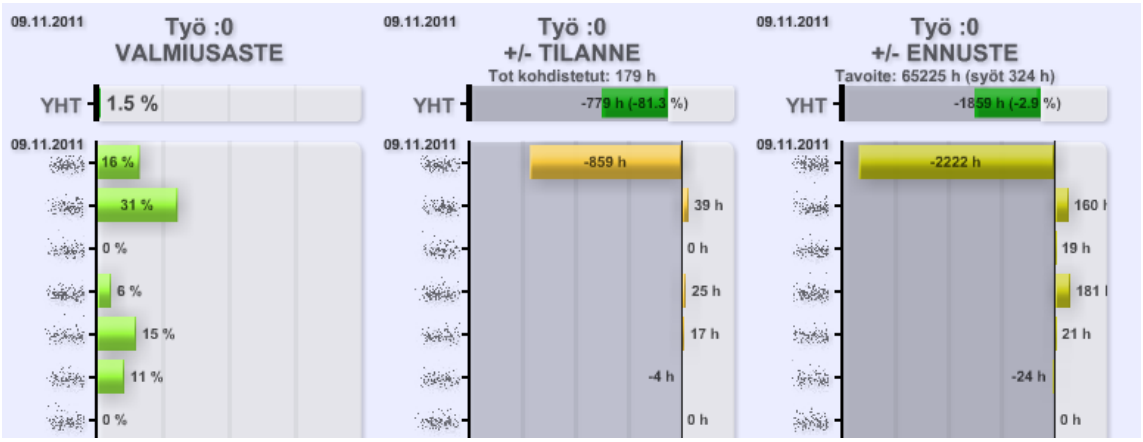
Lohkojen muokkaussivussa ei ole kerta kaikkiaan mitään ihmeellistä; siinä listataan allekkain olemassa olevat lohkot, minkä lisäksi on mahdollisuus lisätä uusia lohkoja ja nimetä uudelleen tai poistaa sellaisia lohkoja, joille ei ole syötetty tunteja. Työvaiheiden yhdistämisessä lohkoihin sentään pääsin käyttämään AJAXia. Toiminnallisuutta on hahmoteltu kuvioon 19.



Kuvio 19. Työvaiheiden yhdistäminen lohkoihin.

Kun vasemmalla olevasta listasta, johon on listattu kaikki mahdolliset tietokannasta noudetut työvaiheet, valitaan jokin työvaihe, oikealle ajaxoidaan sisään taulukko, johon listataan kaikki lohkot, joille voidaan sitten syöttää, paljonko tätä valittua työtä niihin sisältyy, ja mikä on tavoiteaika. Tavoiteaika syötetään minuutteina, ja tuntimäärä lasketaan automaattisesti; se on määrä yhteensä kertaa tavoiteminuutit jaettuna kuudellakymmenellä. Kun käyttäjä syöttää taulukon johonkin ruutuun jotain tai valitsee pudotuslistasta jonkin yksikön, vasemmalla oleva lista muuttuu inaktiiviseksi; tällä varmistetaan se, etteivät aiotut muutokset tuhoudu vahingossa. Listan voi palauttaa aktiiviseksi joko tallentamalla tai perumalla tekemänsä muutokset. Lohkorivin voi tietenkin jättää myös tyhjäksi, mikäli kyseiselle lohkolle ei valittua työtä aiota tehdä, ja luonnollisesti taulukkoon ladataan aina myös aikaisemmin syötetyt arvot, mikäli niitä on.

Työn etenemisen seurannan helpottamiseksi järjestelmässä on vielä tilasto-osio, jossa voi tutkia kaikenlaisia diagrammeja ja kaavioita siitä, miten projekti etenee ja paljonko ollaan aikataulusta jäljessä, mutta siitä en osaa paljon kertoa. Tein kyllä toimeksiantajalta saamieni matemaattisten kaavojen pohjalta erinäisiä arvoja laskevat PHP-funktiot, joiden tulokset syötetään erinäisille toimeksiantajan hankkimille Flash-palikoille, jotka sitten piirtävät noita kaavioita, mutta en minä oikeastaan niitä ymmärrä, eikä ole tarviskaan. Ilmeisesti ne kuitenkin toimivat kuten oli tarkoituskin. Esimerkki kuvassa 6.



Kuva 6. Esimerkki tilasto-osiosta.

5.4 Tiedotusosio

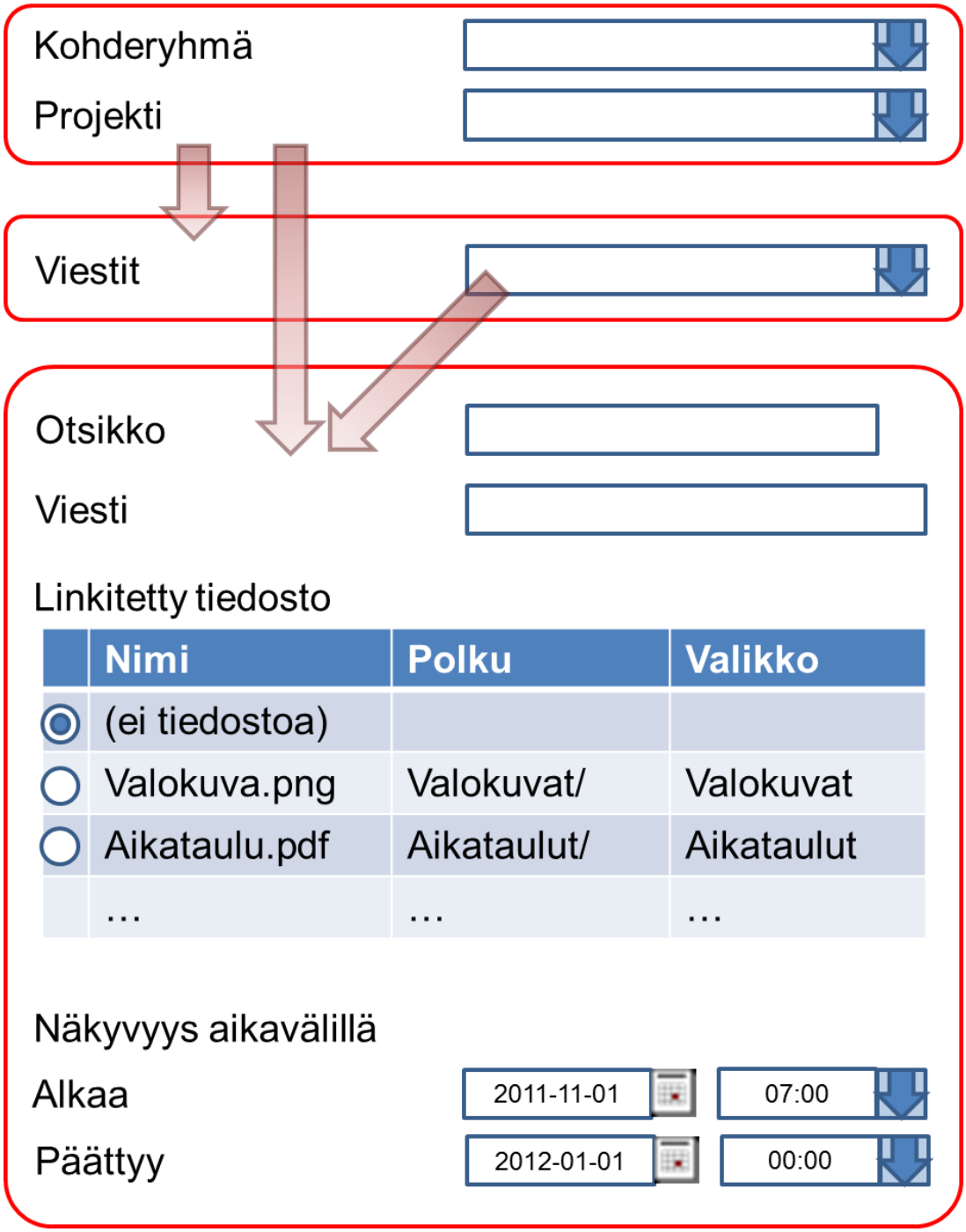
Järjestelmä tarjoaa kolme selvästi erillistä tapaa tiedottaa käyttäjiään, ja ne ovat tiedostojen lukumahdollisuus, tickerit ja järjestelmän sisäinen viestikeskus.

Tiedostojen tuomista järjestelmään esiteltiinkin jo luvussa 5.2; järjestelmä tarjoaa siis mahdollisuuden tuoda kaikenlaisia tiedostoja ja suodattaa käyttäjätason mukaan, kenellä on mahdollisuus tiedostoja tarkastella. Tätä voi ja on tarkoituskin hyödyntää myös tiedottamisessa; yrityksen edustajat voivat tuoda järjestelmään esimerkiksi aikatauluja ja valokuvia asiakkaiden ja/tai henkilöstön ihmeteltäväksi.

Tickerit ovat pieniä Flash-palikoita, joissa rullaa tekstiä vaakasuuntaisesti. Ne, tai ainakin toinen niistä, ovat käyttöliittymässä aina näkyvissä, kuten esitettiin kuviossa 15. Alempi ticker on tarkoitettu kaikkia järjestelmän käyttäjiä, mukaan lukien asiakkaat, koskevien viestien välittämiseen, kun taas ylempi näkyy vain yrityksen henkilökunnalle, ja sitä voi käyttää sisäisten asioiden tiedotukseen, esimerkiksi muistuttamaan, että toteutumaa olisi taas hyvä syöttää, koska tunteja on kertynyt niin ja niin paljon.

Ticker-Flashit ovat valmiita palikoita, jotka toimenantaja toimitti, enkä niiden sielunelämästä näin ollen tiedä sen enempää. Niissä pyöriville teksteille, jotka haetaan tietokannasta, tein kuitenkin hallintasivun, jossa käytetään AJAXia vahvasti, ja jonka toimintaa on hahmoteltu kuvioon 20. Hallintasivulla on näkyvissä kaksi pudotusvalikkoa, joista toisesta valitaan kohderyhmä, eli käytännössä kumpaa tickeriä hallinnoidaan, ja toisesta jokin projekti, jolle viestin näkyvyys rajoitetaan, tai vaihtoehtoisesti kaikki projektit. Kun nämä kaksi valintaa on tehty, sivulle ladataan lomake, jolla on mahdollista joko tehdä uusi ticker-viesti tai muokata vanhoja; käytännössä näkyviin ilmestyy uusi pudotusvalikko, josta voi halutessaan valita jo olemassa olevan ticker-viestin, syöttöruudut viestin otsikolle ja viestille, lista järjestelmään rekisteröidyistä tiedostoista, joista yhden voi halutessaan linkittää viestiin sekä päivämääräsäätimet, joilla voidaan päättää, mistä ajankohdasta mihin viesti on näkyvissä. Valittaessa pudotuslistasta joku olemassa jo oleva viesti nämä kaikki kentät toki päivittyvät

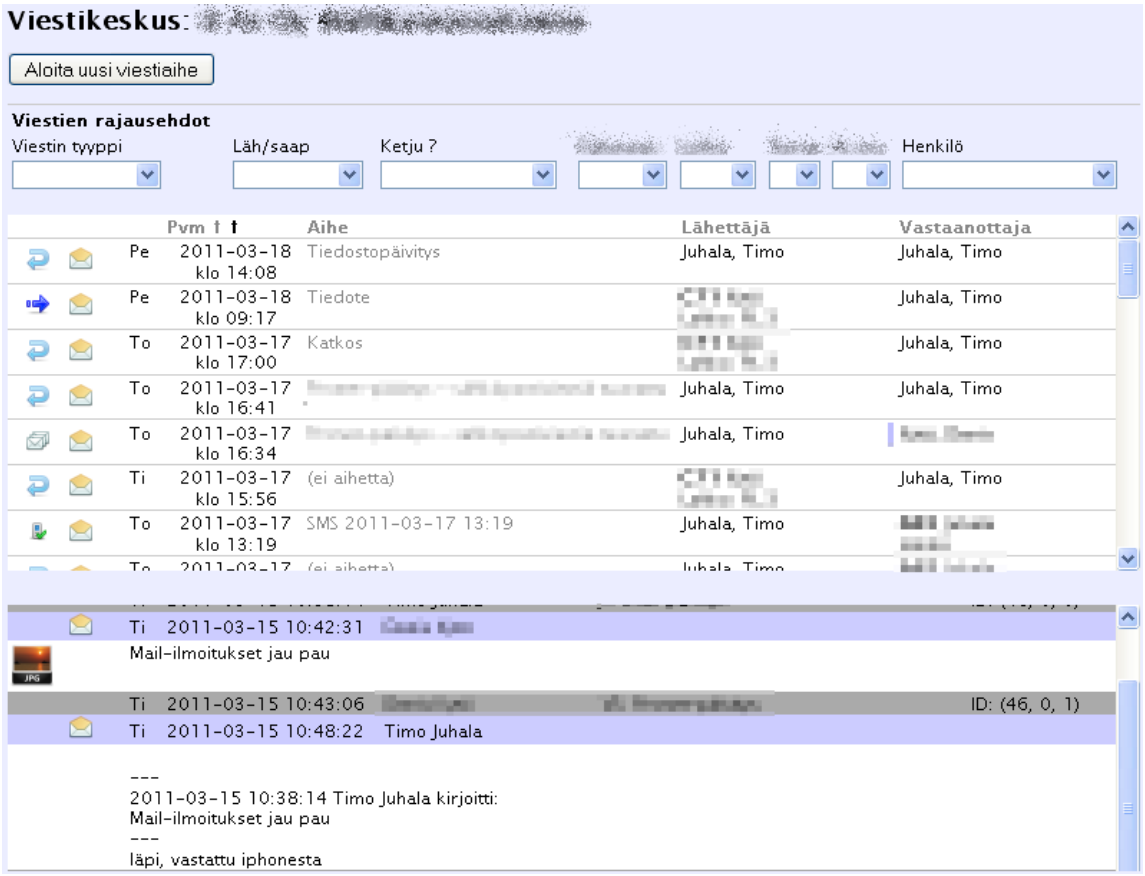
lennossa vastaamaan valittua viestiä, minkä lisäksi tarjotaan mahdollisuus tallentaa muutokset, tallentaa muutokset uutena viestinä (näin voi tehdä vaikka kopioita samasta viestistä, jos niin haluaa) tai poistaa viesti kokonaan.



Kuvio 20. Tickereiden hallintasivun toiminta.

Kuten todettua, järjestelmään on rakennettu myös viestikeskus, joka mahdollistaa järjestelmän sisäisten viestien lähettämisen. Viestikeskuksen etusivulla näkee hieman sähköpostiohjelman tai Facebookin tyyliin listan kaikista viesteistä, joissa on itse osallisena joko lähettäjänä tai vastaanottajana, minkä lisäksi kerrotaan viestin otsikko, lähettäjä, vastaanottaja ja lähetysaika. Listassa näkyviä viestejä voi suodattaa muun muassa sen mukaan, onko itse niiden lähettäjä vai vastaanottaja, onko viestiin vastattu, viestin tyypin mukaan ja henkilön mukaan. Suodatus tapahtuu lennosta, eli lista haetaan AJAXilla aina uudelleen jonkin suotimen vaihtaessa asentoa.

Klikkaamalla viestin otsikkoa itse viesti avautuu viestilistan alapuolelle, jolloin siihen on myös mahdollista vastata. Nykyisen Facebookin tyylisesti vastaamalla saa-maansa viestiin muodostaa ketjun; kaikki tietyn viestin aloittamat keskustelut näkyvät viestilistassa aina vain yhden otsikon alla. Kuvassa 7 näkyy, miltä viestilista ja viestin lukeminen suunnilleen näyttävät.



Kuva 7. Viestikeskus.

Viestilistassa käytetään myös havainnollisia kuvakkeita kertomaan, minkä tyyppinen viesti kulloinkin on kyseessä; esim. u-käännöksen tekevä nuoli kertoo, että kyseessä on ketju, eli alkuperäiseen viestiin on vastattu. Luetut viestit merkataan avonaisella kirjekuorella ja lukemattomat suljetulla – siis nimenomaan vastaanottajan lukemat viestit; lähettäjä itsekin voi näin tarkkailla, onko vastaanottaja ollut hereillä. Sisäänkirjautunut käyttäjä näkee myös koko ajan pikalinkeistä, onko hän saanut viestejä, ”Postilaatikko”-linkki korvautuu ”Sinulla on n lukematonta viestiä” –tyyppisellä ilmoituksella, mikäli näin on käynyt.

”Aloita uusi viestiaihe” –napin painallus ohjaa käyttäjän viestinlähetyssivulle. Tällä sivulla käyttäjä voi valita haluamansa vastaanottajat; valittavissa ovat projektin asiakkaat sekä valittu osa projektihenkilöstöstä, esimerkiksi projektipäällikkö tai laskuttaja – tosin mikäli järjestelmään sisäänkirjautunut henkilö on itse asiakas, valittavissa on vain projektihenkilöstö; asiakkaiden ei siis ole tarkoitus viestitellä toisilleen. Vastaanottajia voi siis valita myös useampia kerrallaan. Näiden lisäksi löytyy pudotusvalikko, josta voi valita viestin aiheen (vaihtoehdot riippuvat sisäänkirjautuneen käyttäjän tasosta; projektihenkilöstöllä näitä ovat mm. ”Tiedote”, ”Tiedostopäivitys” ja ”Järjestelmäpäivitys”), tekstikenttä viestin otsikolle, tekstiruutu itse viestille sekä kaksi tiedostovalitsinta, joiden avulla viestiin voi halutessaan liittää liitetiedoston. Näin ladatut tiedostot menevät viestin lähettäjän omaan kansioon. Kaikkia tiedostoja järjestelmä ei anna liittää, mutta sallittuja ovat esimerkiksi MS Officen tuottamat tiedostot, PDF-tiedostot, kuvat ja tekstitiedostot. Mikäli liitetiedosto sattuu olemaan suuri kuvatiedosto, se pienennetään PHP:n GD-kuvankäsittelykirjastoa hyödyntäen niin, että se on korkeintaan 640 pikseliä leveä tai korkea.

Yrityksen edustaja voi lähettää samalta sivulta myös SMS-viestejä eli tekstiviestejä vastaanottajien matkapuhelimiin. Näihin viesteihin ei voi liittää liitetiedostoja, valita viestin tyyppiä tai kirjoittaa otsikkoa ja niiden maksimipituus on 160 merkkiä. Vastaanottajalistasta ei voi valita henkilöitä, joiden matkapuhelinnumeroa ei ole tallennettu järjestelmään. SMS-viestien lähetys tapahtuu PHP:n Client URL –eli cURL-kirjastoa käyttäen; viestit lähetetään eräälle SMS-viestilähetyspalveluntarjoajalle ISO-8859-15-koodattuina (tähän tarvittiin lukuisia

yrityksiä, erehdyksiä ja PHP:n iconv-funktiota) ja niihin liitetään viestin lisäksi tietenkin muun muassa vastaanottajan puhelinnumero. Kaikki SMS-viestit lähtevät samalla myös normaaleina järjestelmän sisäisinä viesteinä, eli vastaanottaja voi lukea ne myös kirjautumalla järjestelmään, joka on myös ainoa keino vastata viesteihin.

Järjestelmän sisäisiin viesteihin vastaaminen tapahtuu täysin samalla sivulla kuin originellien viestien lähettäminenkin, mutta vastaanottajalista on rajattu tasan yhteen käyttäjään – viestin lähettäjään – eikä sitä voi muuttaa.

Kaikista järjestelmän sisäisistä viesteistä lähtee myös ilmoitus viestin vastaanottajille sähköpostitse; tämä viesti on standardimuotoinen, ja sen sisältönä on vain ilmoitus siitä, että järjestelmässä on lukematon viesti odottamassa ja linkki sisäänkirjautumissivulle.

6 JOHTOPÄÄTÖKSET

”Mitä hyötyä on ihmiselle kaikesta vaivannäöstä, jolla hän itseään rasittaa auringon alla?” (Saarn. 1:3). Niin, suunnaton vaivannäkö on nyt ohi; jäikö siitä mitään käteen? Pätevin vastaaja kysymykseen on tietysti toimeksiantaja, jota lähestyinkin asian tiimoilta sähköpostitse. Seuraavassa esittämiäni kysymyksiä, lainauksia hänen vastauksestaan sekä omaa pohdintaani asioista, minkä jälkeen koitan vielä hieman miettiä tutkimuksen yleisempää hyödynnettävyyttä sekä sitä, mitä tekisin toisin, jos nyt aloittaisin alusta.

6.1 Palautetta toimeksiantajalta

Kysymys: Onko lopputulos hyvä?

Vastaus: ”Tällaiset sovellukset eivät taida koskaan tulla ihan valmiiksi, mutta nyt tekemääni työmäärää sen päälle johon se huhtikuussa jälkeesi jäi arvioisin ohjelman olleen n. 60 % valmis, eikä vielä käyttökelpoinen.

”On tietysti selvää että nämä vievät aikaa, ja testaamisen kautta osaa palikoista hienosäädetään osa muutetaan kokonaan. Niin kävi tässäkin.”

”Kuten aloitettaessa mainitsinkin, web-ohjelmointini oli lapsen kengissä, ja sain oppia monia asioita koodatessasi pohjaa tälle järjestelmälle.

”Ilman sitä oppia ja tekemääsi koodia, olisi tämä projekti jäänyt sudeksi. Lopputulos tällä hetkellä arvosanalla 9+?”

Pohdintaa: Tuo luku 60 % on sattumaa eikä ole suoraan johdettu Aternin pienin käytettävä osajoukko –peukalosäännöstä. Vaatimuslistan vaatimuksista kaikki M-priorisoidut vaatimukset tuli kyllä tehtyä, kuten myös suuri osa S- ja C-vaatimuksistakin; se, että järjestelmä ei toimeksiantajan mielestä ollut käyttökelpoinen projektin päätyttyä, kertoo mielestäni siitä, että kaikkia tärkeimpiä vaatimuksia ei oltu huomioitu ajoissa, ja näin se itse asiassa taitaa ollakin; nykyisellään järjestelmässä on mm. tuki monelle eri tietokannalle ja tiedostojen käsittelyä on käsittääkseni myös uudistettu rajulla kädellä.

Alkuperäisiin vaatimuksiin peilaten väittäisin järjestelmän kuitenkin olleen käyttökelpoinen, mistä kielii myös vastaus seuraavaan kysymykseen. Joka tapauksessa hyvä pohja laajentamiselle ainakin saatiin aikaiseksi.

Kysymys: Ylittyivätkö tai alittuivatko odotukset?

Vastaus: ”Miten hankalaa onkaan selvittää toiselle tämän tasoisessa ohjelmassa kaikkia pikku yksityskohtia joilla on kuitenkin merkittävä osuus ohjelman toiminnan kannalta... No sehän oli jo alussa kaikilla osapuolilla tiedossa. Olisin kuvitellut sen olevan vähän helpompaa, nyt kuitenkin itsenäisesti sitä tehneenä tiedän, että mahdollisuuksia on miljoonia. Koodaajan tehtävä ei ole helppoa...

”Toteutunut on varmaan lähellä sitä tasoa jossa odotuksetkin olivat.”

Pohdintaa: Se, että on kohtalainen koodaaja, ei vielä pätevöitä ymmärtämään liiketoiminnan tarpeita. Ongelmia monessa kohdassa projektin aikana aiheutti se, että toimiala T oli minulle likimain täysin tuntematon, ja siksi toimeksiantaja joutuikin harrastamaan aika paljon kädestäpitelyä. Kukaan tuskin myöskään osaa AMK-pohjalta ulkoa koko PHP:n funktiokavalkadia, joten ihan kielen opetteluunkin meni aikaa, ja niin on ilmeisesti mennyt toimeksiantajaltakin.

Kysymys: Onko kommentteja jostain järjestelmän tietystä osiosta, jota pidät tärkeänä?

Vastaus: ”Yksi joka nyt tulee heti mieleen on valikon muunneltavuus ja monikäyttöisyys.”

Pohdintaa: Tuo valikko on kyllä melko joustava, ja olen itsekin tyytyväinen tapaan, jolla lopulta keksin sen toteuttaa. Kyllä sitä monta kertaa tuli muokattuakin suuntaan jos toiseen; siinä sitä iteratiivista kehitystä kerrakseen.

Kysymys: Mitä mieltä olet käytetyistä työkaluista ja menetelmistä ja niiden sopivuudesta projektiin (Skype, Excelillä tehdyt käyttöliittymäluonnostelmat, tehtävien MoSCoW-priorisointi...)?

Vastaus: ”Skype ihan ehdoton, iman sitä tämä olisi jäänyt alkutekijöihinsä. Vie tosin opastajalta lähes saman ajan päätteen edessä, mutta mikäli ajan haluaa hyödyntää, en näe muutakaan keinoa. Excel-mallit auttoivat varmistamaan informaation oikein ymmärtämisen. Hyvä työkalu. Priorisointi myös ok. Hyvät menetelmät kaiken kaikkiaan.”

Pohdintaa: Kysymys oli huonosti aseteltu, mutta ei lisättävää noita asioita koskevaan vastaukseen. Omasta puolestani voin kehua myös ketterien menetelmien ja Aternin perusajatusta iteratiivisuudesta ja inkrementaalisuudesta; oli ensinnäkin motivoivaa nähdä itse nopeasti omia aikaansaannoksiaan ja toisekseen oli hyvä saada palautettakin heti, eikä sitten, kun muutosten tekeminen olisi jo käynyt turhan työlääksi.

6.2 Tutkimuksen hyödynnettävyys

Johdantoluvussa tutkimusongelmaksi asetettiin suunnilleen ”kuinka ketterät menetelmät yleisesti ja DSDM Atern erityisesti soveltuvat pienimuotoiseen ohjelmistoprojektiin”. Nähdäkseni vastaus tähän on, että sopivasti menetelmää soveltamalla hyvin.

Aternista jätettiin tässä paljon asioita pois, mutta ketterien menetelmien yleispiirteiksi laskettavat asiat, kuten iteratiivinen kehitys ja läheinen kommunikaatio, kuitenkin säilytettiin, ja mielestäni nuo piirteet ovat oikein soveltuvia ohjelmistonkehitykseen. Erityisen toimivaa oli mielestäni näiden kahden perusasian luoma nopea palautesykli, joka helpotti muutokseen vastaamista merkittävästi; olisi ollut todella turhauttavaa puurtaa päivä ja sitten kuulla tehneensä asiat väärin, puhumattakaan pidemmästä ajanjaksosta.

Ne Aternille ominaiset asiat, jotka säilytettiin, jättivät myös itsestään hyvän maun. MoSCoW-priorisoitu vaatimuslista erityisesti on mielestäni loistelias keksintö, joka todella tarkoituksensa mukaisesti auttoi ymmärtämään, mikä milloinkin on oleellista ja mikä ei.

Se, miten karsimaton Atern sopii minkä tahansa kokoisen ohjelmistoprojektin läpiviemiseksi, jäi tietenkin ratkaisemattomaksi. Musta tuntuu –metodia käyttä-

mällä uskaltaisin kuitenkin veikata, että hyvin. Pidän Aternissa siitä, että se tavallaan pakottaa liiketoiminnan edustajatkin ajattelemaan asioita ja sitoutumaan projektiin; esimerkiksi tuo oleellinen vaatimuslista oli oranssilla värillä merkattu, jolloin se on juurikin liiketoiminnan vastuulla päättää, mitä he oikeasti haluavat ja kuinka palavasti. Myös perusfilosofia lukkoon lyödystä aikataulusta miellyttää; en ole vielä kovinkaan kokenut ohjelmistokehittäjä, mutta lyhyen, tähän opinnäytetyöhön siis millään tavalla liittymättömän, työurani aikana olen jo ehtinyt huomaamaan, miten yli aikataulujen ja budjetin venyvät projektit ovat omiaan aiheuttamaan närkästystä sekä asiakkaassa että toimittajassa. Tämä menetelmä tarjoaa ainakin yhdenlaisen ratkaisun tuohon ongelmaan.

6.3 ”Jos nyt saisin aloittaa alusta…”

…en tekisi paljoakaan toisin. Näin jälkikäteen ajateltuna on kuitenkin joitakin asioita, jotka osaisin nykyään tehdä fiksummin.

Selainpuolen ohjelmointityössä käyttäisin jQuery-kirjastoa niin paljon kuin mahdollista. Myöhempi kokemus on osoittanut sen nopeuttavan ja helpottavan javascript-koodien tekemistä merkittävästi, minkä lisäksi se vaikuttaisi olevan maagisesti enemmän yhteensopiva eri selainten kanssa kuin tavallinen javascript paikoitellen on! Myös AJAX-tuki on jQueryssä erittäin hyvä, ja olisin säästänyt ainakin kaksi kokonaista työpäivää, mikäli en olisi joutunut hakkaamaan aikanaan päätäni seinään tihrustaessani kyyneltulvan läpi W3Schoolsin AJAX-tutoriaalia. Oman opettelunsa tuokin kirjasto tietysti vaatii, mutta uskoisin siihen käytetyn ajan kuitenkin maksaneen tässäkin projektissa itsensä takaisin.

AJAX on ylipäätään niin hieno teknologia, että sitä olisi voinut käyttää enemmänkin. Järjestelmään jäi jokunen alasivu, jotka toimivat oikein perinteisellä tavalla, eli lomake pitää lähettää palvelimelle ja ladata koko sivu uudelleen, kun on vaikka tehnyt jonkun valinnan jostain valikosta ja sen pohjalta pitää tarjota lisää vaihtoehtoja käyttäjälle. (Onneksi jätin nämä sivut esittelemättä.) Näistäkin olisi voinut tehdä vaikuttavampia ja vähemmän turhaa verkkoliikennettä synnyttäviä käyttämällä AJAXia.

Kansainvälistyminen on yksi päivän sanoista, ja sen olisi voinut huomioida paremmin, vaikka eri kieliversiot eivät vaatimuslistassa olleetkaan. Nythän järjestelmä on pelkästään suomenkielinen ja kaikki käyttäjälle näkyvät tekstit on kirjoitettu suoraan koodin sekaan; järkevämpää olisi varmasti ollut sijoittaa ne vaikka tietokantaan tai omiin resurssitiedostoihinsa, jotta aivan jokaista kooditiedostoa ei tarvitsisi käydä muokkaamassa, mikäli järjestelmästä haluaisi tehdä monikielisen.

Koodin kommentointia olisi voinut harrastaa enemmän, mutta niinhän se aina on ollut ja tulee varmaan aina olemaan.

Järjestelmän käyttöliittymä ei ole aivan yhtenäinen, eikä aina täysin looginenkaan, vaan se saattaa eri alasivuilta toiselle vaihdella paljonkin sekä ulkonäöltään että toimintamalleiltaan. Kaikessa rehellisyydessä on kyllä mainittava, että sellainen siitä varmaan tulisi nytkin, sillä käyttöliittymä-suunnittelu ei missään nimessä kuulu vahvuuksiini. Joka tapauksessa sen voisi tehdä paremmin.

Jos ajatellaan vielä järjestelmän ulkopuolelta itse työtapoja, niin korjattavaa löytyisi niistäkin. Heräilin joka aamu ennen kahdeksaa, vaikka en ole aamuihminen, ja se olikin ihan hyvä juttu, koska siinä neljän kieppeillä pystyi hyvällä omatunnolla lopettelemaan – paitsi toisinaan perjantaisin, koska aina vasta silloin heräsi todellisuuteen, jossa viikon asioista oli vielä vaikka kuinka paljon tekemättä. Yrittäisinkin tehdä mieluummin alkuviikosta niitä pitkiä päiviä, koska havaitsin perjantai-iltojen kiirekoodaamisen olevan huonoksi mielialalle.

Ja sinulle, joka et vielä opinnäytetyötäsi tehnyt, yksi vinkki: **kirjoita sitä opinnäyteraporttia samalla kun suoritat itse tutkimusta** tai pidä hyvä ihminen edes päiväkirjaa! Oli nimittäin todella tuskallista havahtua yhtäkkiä, kuukausia itse koodiorjuuden jälkeen, siihen, että tämä työkin tosiaan pitäisi kirjoittaa, ja sitten huomata, ettei enää oikein edes muista mitään tekosistaan…

LÄHTEET

Aked, M. 2003. Risk reduction with the RUP phase plan. Viitattu 7.11.2011 http://www.ibm.com/developerworks/rational/library/1826.html#N100E4

Ambler, S. 2009. The Agile Unified Process (AUP). Viitattu 12.10.2011 http://www.ambysoft.com/unifiedprocess/agileUP.html

Beck, K.; Beedle, M.; van Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; Kern, J.; Marick, B.; Martin, R.; Mellor, S.; Schwaber, K.; Sutherland, J. & Thomas, D. 2001. Ketterän ohjelmistokehityksen julistus. Suom. Koskela, L. & Agile Finland. Viitattu 12.10.2011 http://agilemanifesto.org/iso/fi/

Beck, K. 2003. Extreme Programming Explained. 10th Printing. Boston: Addison-Wesley.

Clifton, M. & Dunlap, J. 2003. What Is DSDM? Viitattu 23.10.2011 http://www.codeproject.com/KB/architecture/dsdm.aspx

Cockburn, A. 2005. Crystal Clear: A Human-Powered Methodology for Small Teams. Boston: Addison-Wesley. Luku 2 saatavissa myös http://www.pearsonhighered.com/assets/hip/us/hip_us_pearsonhighered/samplechapter/0201699478.pdf

cPrime 2011. Scrum & Agile FAQ:The everything you need to know guide! Viitattu 12.10.2011 http://www.cprime.com/about/scrum_faq.html

Dynamic Drive 2011. All Levels Navigational Menu. Viitattu 9.11.2011 http://www.dynamicdrive.com/dynamicindex1/ddlevelsmenu/

DSDM Consortium 2008. DSDM Atern – The Handbook. V2. Kent: DSDM Consortium. Saatavissa myös http://www.dsdm.org/atern-handbook/flash.html

Fowler, M. 2005. The New Methodology. Viitattu 13.10.2011 http://martinfowler.com/articles/newMethodology.html

Koch, A. 2004. Agile Software Development: Evaluating the Methods for Your Organization. Norwood: Artech House.

MOT 2011a. Kielitoimiston sanakirja 2.0. Viitattu 31.10.2011 www.nelliportaali.fi > Oikotiet > MOT.

MOT 2011b. Gummerus Uusi suomen kielen sanakirja 1.0. Viitattu 31.10.2011 www.nelliportaali.fi > Oikotiet > MOT.

Ortega-Hernández, H. 2009. DatePickerControl v1.1.7. Viitattu 9.11.2011 http://dali.mty.itesm.mx/~hugo/js/datepickercontrol/

Roberts, M. 1998. Schema. Definition. Viitattu 31.10.2011 http://searchsqlserver.techtarget.com/definition/schema

Rusk, J. 2006. Crystal Clear Methodology. Viitattu 13.10.2011 http://www.agilekiwi.com/other/agile/crystal-clear-methodology/

Schwaber, K. 2004. Agile Project Management With Scrum. Redmond: Microsoft Press.

Scrum Alliance 2011. The Scrum Framework in 30 Seconds. Viitattu 18.11.2011. http://www.scrumalliance.org/pages/what_is_scrum

VersionOne 2011a. Agile Development. Viitattu 12.10.2011 http://www.versionone.com/Agile101/Agile_Development.asp

VersionOne 2011b. Agile Methodologies. Viitattu 13.10.2011 http://www.versionone.com/Agile101/Methodologies.asp

Wells, D. 1999a. The Rules of Extreme Programming. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules.html

Wells, D. 1999b. User Stories. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/userstories.html

Wells, D. 1999c. Release Planning. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/planninggame.html

Wells, D. 1999d. Release Plan. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/commit.html

Wells, D. 1999e. Make frequent small releases. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/releaseoften.html

Wells, D. 1999f. Iterative Development. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/iterative.html

Wells, D. 1999g. Iteration Planning. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/iterationplanning.html

Wells, D. 1999h. Daily Stand Up Meeting. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/standupmeeting.html

Wells, D. 1999i. Project Velocity. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/velocity.html

Wells, D. 1999j. Move People Around. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/movepeople.html

Wells, D. 1999k. Fix XP When It Breaks. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/fixit.html

Wells, D. 1999l. Choose a System Metaphor. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/metaphor.html

Wells, D. 1999m. CRC Cards. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/crccards.html

Wells, D. 1999n. Create a Spike Solution. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/spike.html

Wells, D. 1999o. Never Add Functionality Early. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/early.html

Wells, D. 1999p. Refactor Mercilessly. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/refactor.html

Wells, D. 1999q. The Customer is Always Available. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/customer.html

Wells, D. 1999r. Coding Standards. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/standards.html

Wells, D. 1999s. Pair Programming. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/pair.html

Wells, D. 1999t. Sequential Integration. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/sequential.html

Wells, D. 1999u. Integrate Often. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/integrateoften.html

Wells, D. 1999v. Dedicated Integration Computer. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/dedicated.html

Wells, D. 1999w. Collective Ownership. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/collective.html

Wells, D. 1999x. Unit Tests. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/unittests.html

Wells, D. 1999y. When a Bug is Found. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/bugs.html

Wells, D. 1999z. Acceptance Tests. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/functionaltests.html

Wells, D. 2000a. Extreme Programming Project. Viitattu 18.10.2011 http://www.extremeprogramming.org/map/project.html

Wells, D. 2000b. Code the Unit Test First. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/testfirst.html

Wells, D. 2009a. The Values of Extreme Programming. Viitattu 18.10.2011 http://www.extremeprogramming.org/values.html

Wells, D. 2009b. Give the Team a Dedicated Open Work Space. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/space.html

Wells, D. 2009c. Set a Sustainable, Measurable, Predictable Pace. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/overtime.html

Wells, D. 2009d. Simplicity is the Key. Viitattu 18.10.2011 http://www.extremeprogramming.org/rules/simple.html