Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikka
Organisaation IT-palvelut

Roni Hirsmäki

Ketterä ohjelmistokehitys pienohjelmistotuotannossa – case: Tuntiraportointijärjestelmä

Opinnäytetyö 2013

Tiivistelmä

Roni Hirsmäki
Ketterä ohjelmistokehitys pienohjelmistotuotannossa – case: Tuntiraportointijärjestelmä, 37 sivua
Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikka
Organisaation IT-palvelut
Opinnäytetyö 2013
Ohjaajat: yliopettaja Päivi Ovaska, Saimaan ammattikorkeakoulu
laboratorioinsinööri Mika Hulkkonen, UPM-Kymmene Oyj

Tämän opinnäytetyön tarkoituksena oli määritellä, suunnitella, toteuttaa ja ottaa käyttöön kevyt tuntiraportointijärjestelmä UPM:n tutkimuskeskuksessa Lappeenrannassa. Tarkoituksena oli myös tarkastella ohjelmistokehitysprojektissa käytettyjä työtapoja ja pohtia niiden sopivuutta kyseiseen kehitysprojektiin. Perehdyin myös ohjelmistotuotannon teoriaan, projektien vaihejakomalleihin ja menetelmiin, joita käytin tehdyssä ohjelmistoprojektissa.

Opinnäytetyössä tehty järjestelmä toteutettiin Microsoft Access -ohjelmistolla. Järjestelmän kehitystyössä sovellettiin opintojen aikana opittuja ohjelmistotuotannon menetelmiä ja käytäntöjä. Järjestelmän kehitysvaiheessa käytettiin apuna Microsoftin ohjeita, kirjallisuutta Access-kehityksestä ja Internetin keskustelupalstoja. Teoriaosuudessa käytin myös tukena artikkeleita ja alan kirjallisuutta.

Toteutettu järjestelmä täytti sille asetetut toiminnalliset vaatimukset ja se otettiin käyttöön ennalta sovitun aikataulun mukaisesti. Järjestelmän kehitysprosessissa käytettyjen menetelmien tutkiminen ja vertaaminen teoriaan, paljasti prosessista niin perinteisiä, kuin ketteriäkin ohjelmistokehityksen piirteitä. Kokonaisuutena opinnäytetyö konkreettisien tuloksien ohella, auttoi ymmärtämään ohjelmistotuotantoa käytännön tasolla.

Asiasanat: Access, Crystal Clear, ketterät menetelmät, ohjelmistotuotanto, SQL, VBA

Abstract

Roni Hirsmäki
Agile methods in small scale software engineering – case: An hour reporting system, 37 pages
Saimaa University of Applied Sciences
Technology Lappeenranta
Bachelor of Engineering in Information Techology
IT-services in Organisation
Bachelor´s Thesis 2013
Instructors: Ms. Päivi Ovaska, Senior Lecturer, Saimaa University of Applied Sciences
Mr. Mika Hulkkonen, Laboratory Engineer, UPM-Kymmene Oyj

The purpose of this thesis was to define, design and deploy a light hour reporting system for Research Center of UPM in Lappeenranta. Another objective was to examine how methods used in the development process applied to the project. I studied also the theory of software engineering, its models and methods that I used in the development process.

The system was implemented with Microsoft Access software and it included a database and a client software for users. In the development process, I used previously learned methods, practices and Internet, professional discussion boards and literature for supporting technical implementation. The same sources with numerous professional articles were supporting the theory part.

The implemented system was taken to production environment at the scheduled time and it met the customer's requirements. When reflecting on the development process, traditional and agile characteristic were found in the methods I used. The thesis helped to understand a domain of software engineering along the implementation system for the customer.

Keywords: Access, agile methods, Crystal Clear, software engineering, SQL, VBA

Sisällys

Käytetyt termit ja lyhenteet

Termi	Selite
.NET	Microsoftin kehittämä ohjelmistokomponenttikirjasto, jota käyttää useat Microsoftin kehittämät ohjelmistointikielet, kuten C# ja ASP.
ASP	Active Server Pages, on Microsoftin kehittämä palvelimella suoritettava komentosarjamoottori dyynamisten Web-sivustojen luomiseen.
Access	Access on Microsoft Office -ohjelmistopakettiin kuuluva tietokantojen ja tietokantaohjelmistojen suunnittelu- ja toteutustyökalu.
C#	Microsoftin kehittämä ohjelmointikieli.
Crystal Clear	Alistair Cockburnin vuonna 2004 kuvaama ketterä ohjelmistokehitysmenetelmä.
Edustatietokanta	Jaetun Access-tietokannan osa, joka sisältää mm. VBA-koodin, makrot, raportit ja lomakkeet.
IEEE	Institute of Electrical and Electronics Engineers on kansainvälinen tekniikan alan järjestö, jonka toimintaa kuuluu mm. julkaisutoiminta ja standardien määrittely.
ISO	International Organization for Standardization on kansainvälinen stardardisoimisjärjestö.
Järjestelmä	Laitteista, ohjelmistoista ja niiden käyttöön liittyvistä vaiheista muodostuva kokonaisuus.
MVC	Model-View-Contoller on ohjelmistoarkkitehtuurimalli.
Ohjelmisto	Ohjelma ja siihen liitettyjen osien muodostama kokonaisuus.
Relaatiotietokanta	Tietovarasto, jossa tieto on ryhmitelty tauluihin, joiden väliset suhteet kuvataan relaatioilla.
SharePoint	Microsoftin kehittämä Web-ohjelmistoalusta.
SQL	Structured Query Language on kyselykieli, jolla voidaan hallita tietokannan tietueita.
Taustatietokanta	Jaetun Access-tietokannan osa, joka sisältää mm. taulut ja indeksit.
UML	Unified Modeling Language on standardoitu kuvauskieli, joka on kehitetty ohjelmistotuotantoon.
VBA	Visual Basic for Applications on ohjelmointikieli Microsoft Office -ohjelmistoille.

1 Johdanto

Tämän opinnäytetyön tarkoituksena on määritellä, suunnitella, toteuttaa ja ottaa käyttöön kevyt tuntiraportointijärjestelmä UPM:n tutkimuskeskuksessa Lappeenrannassa. Tarkoituksena on myös tarkastella ohjelmistokehitysprojektissa käytettyjä työtapoja ja pohtia niiden sopivuutta kyseiseen kehitysprojektiin. Perehdyn myös ohjelmistotuotannon teoriaan, projektien vaihejakomalleihin ja menetelmiin, joita käytän toteutettavassa ohjelmistoprojektissa.

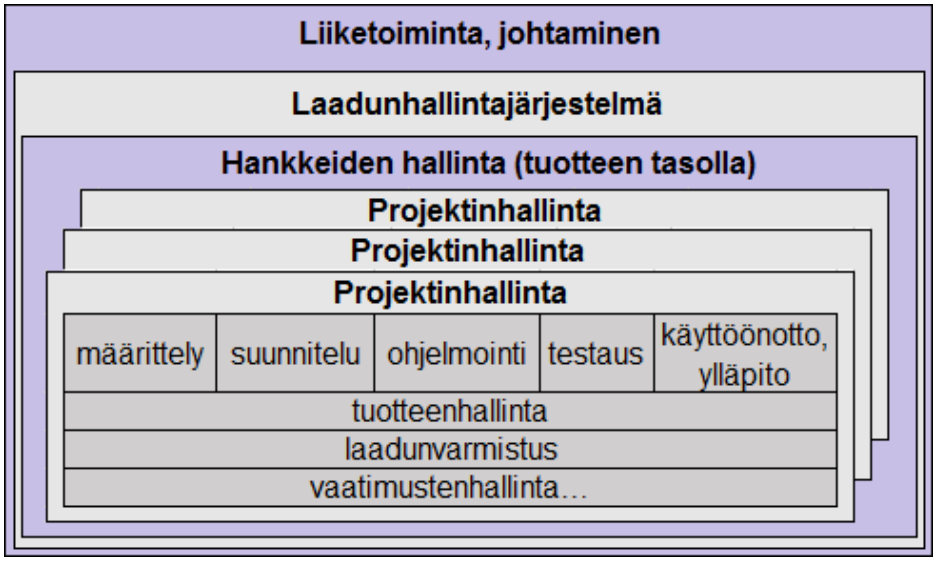
UPM:n tutkimuskeskuksessa sisäisen rahoituksen projektien tuntiseuranta päättyi vuoden 2011 lopussa, jonka jälkeen on raportoitu ainoastaan ulkoisen rahoituksen projektien työtunteja. Tuntiraportointiin on nykyään käytössä järjestelmä, jonka tilalle halutaan uusi, kevyempi järjestelmä, jonka toivotaan helpottavan tuntien kirjaus- ja raportointiprosessia. Uudistuksella haetaan myös kustannussäästöjä, koska nykyisen järjestelmän ylläpito on ulkoistettu. Uuden järjestelmän ohjelmistot tullaan toteuttamaan sellaisilla tekniikoilla ja sellaisille alustoille, joita asiakkaalla on jo käytössään. Ohjelmistosta ei tulla myöskään tekemään yhteyksiä muihin järjestelmiin tai ohjelmistoihin. Järjestelmän kehitys aloitetaan lokakuussa 2012, ja se tullaan ottamaan käyttöön tammikuussa 2013.

Aiempaa kokemusta työskentelystä aidossa ohjelmistokehitysprojektissa minulla ei ole, mikä tekee opinnäytetyöstä mielenkiintoisen. Projektin aikatauluvaatimuksien vuoksi toteutan järjestelmän jo opittujen menetelmien pohjalta, minkä jälkeen pohdin, miten ketterien ohjelmistotuotanto menetelmien hyväksikäyttäminen olisi vaikuttanut tehdyssä projektissa.

Toinen luku on teoriaosuus, jossa kuvailen ohjelmistotuotantoa, sen vaiheita, vaihejakomalleja ja kehitysmenetelmiä. Kolmannessa luvussa esittelen järjestelmän tilaajayhtiön, toimipisteen ja sen liiketoiminta-alueen, johon järjestelmä toimitetaan. Neljännessä luvussa esittelen järjestelmän, joka on tutkimuskeskuksella jo käytössään, ja tulevan järjestelmän, joka tulee korvaamaan kyseisen järjestelmän. Kerron myös uuden järjestelmän ohjelmistokehitysprosessin ja menetelmät, joita käytän ohjelmistokehityksessä. Viimeisessä luvussa pohdin käyttämiäni menetelmiä ja kuinka olisin voinut parantaa ohjelmistokehitysprosessiani.

2 Ohjelmistotuotanto

Ohjelmistotuotanto on tietokoneohjelmien ja niistä koostuvien ohjelmistojen sekä kokonaisten järjestelmien tuotantoa. Tuotanto on systemaattinen, kurinalainen, määritettävissä oleva menettelytapa ohjelmiston kehitykseen, operointiin ja ylläpitoon (1, s. 30). Tuotannon tukena käytetään lukuisia erilaisia menetelmiä, työkaluja, periaatteita ja tekniikoita sekä sen osa-alueet ulottuvat liiketoiminnan strategiasta ohjelman ylläpitoon (Kuva 1).



Kuva 1. Ohjelmistotuotannon osa-alueet (2, s. 29)

Ohjelmistotuotannossa kaiken toiminnan lähtökohtana on liiketoiminta. Laadunhallintajärjestelmän tarkoituksena on toteuttaa liiketoiminnassa ja sen johtamisessa järjestelmällisesti laadunhallintaa ja -varmistusta, millä vaikutetaan laadun syntymiseen. Laadunhallintajärjestelmä ulottuu yrityksen liiketoimintatasolta tekniseen toteutukseen asti. (2, s. 30.)

Yrityksen projektit liittyvät usein tiettyyn hankkeeseen, jossa on tietty strateginen tavoite, esimerkiksi tuotantoyksikön tehostaminen (2, s. 30). Projektit tähtäävät määrättyjen kokonaisuuksien ratkaisemiseen, johon liittyy useita vaiheita ja menetelmiä, joita esittelen seuraavissa luvuissa 2.1 ja 2.2.

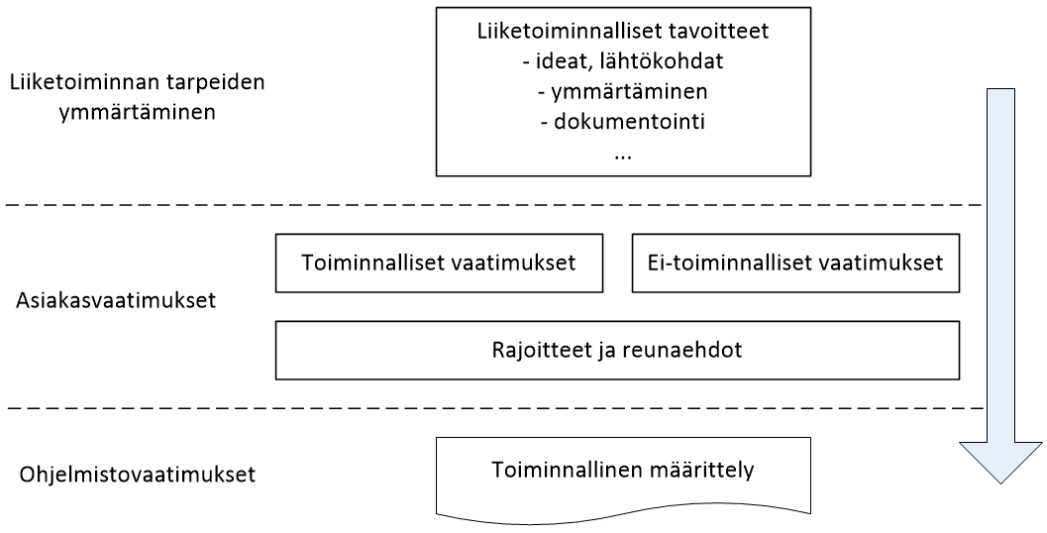
2.1 Ohjelmistotuotantoprojektin vaiheet

Ohjelmistoja tuotetaan usein projekteissa, joissa toteutetaan asiakkaan vaatimuksiin perustuvia ohjelmistoja. Asiakkaan vaatimukset perustuvat liiketoiminnallisiin tavoitteisiin, joiden saavuttamista pyritään helpottamaan niitä tukevilla ohjelmistoilla.

Projekteissa pyritään täyttämään asiakkaan ohjelmistolle asettamat kohtuulliset vaatimukset, siten että ennalta arvioidut kehityskustannukset ja aikataulu eivät ylity (2, s.12). Usein projektiaikataulut tai kustannukset eivät kuitenkaan pysy ennalta arvioiduissa rajoissa vaan monesti ylittyvät reilustikin. Viimeaikaisimpia esimerkkejä epäonnistuneista projekteista on VR:n lippujärjestelmä, terveydenhuollon yhteinen potilastietojärjestelmä ja Sampo Pankin tietojen siirto Danske Bankille (3).

Määrittely

Ohjelmistoprojektin määrittelyvaiheessa määritellään ja dokumentoidaan ne toiminnallisuudet, joita asiakas tuotettavalta ohjelmistolta haluaa ja vaatii. Vaatimustenmäärittely on erittäin tärkeä osa ohjelmistoprojektia, koska se määrittelee sen, minkälaista ohjelmistoa projektissa ollaan tekemässä. Useat asiantuntijat ja tutkimukset ovat osoittaneet, että ohjelmistoprojektien epäonnistumisen syyt johtuvat yli 60-prosenttisesti huonosta vaatimustenkäsittelystä (2, s.61).



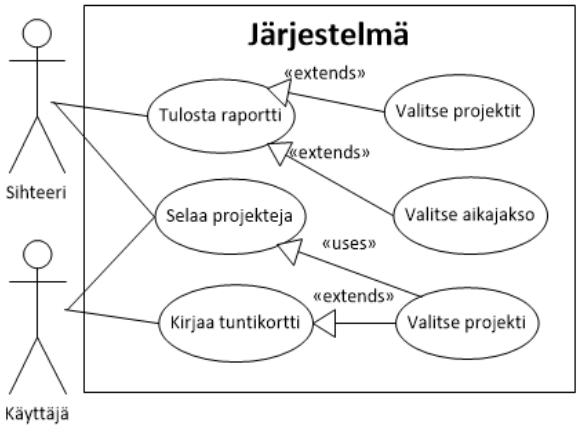
Kuva 2. Asiakas- ja ohjelmistovaatimukset (2, s. 62)

Määrittelyvaiheessa määritellään ne ohjelmiston toiminnallisuudet, jotka täytyy ohjelmistossa olla. Määrittelyn tarkoituksena on vastata asiakkaalle ja toimittajalle kysymykseen: Mitä projektissa ollaan tekemässä ja miksi sitä tehdään?

Määrittelyyn kuuluu asiakasvaatimuksina toiminnallisia ja ei-toiminnallisia vaatimuksia, rajoitteita ja reunaehtoja (Kuva 2). Toiminnalliset vaatimukset ovat asiakkaan määrittelemiä vaatimuksia toiminnoista, jotka ohjelmistossa täytyy olla, esimerkiksi: ohjelmassa on mahdollisuus nähdä kunkin työntekijän tunnit projekteittain. Ei-toiminnalliset vaatimukset ovat vaatimuksia, jotka liittyvät esimerkiksi ohjelman suorituskykyyn, vasteaikaan tai käyttöliittymän värimaailmaan. Reunaehdot ovat asiakkaan asettamia ehtoja, joiden puitteissa ohjelmisto täytyy toteuttaa, esimerkiksi käytettävästä tekniikasta, kuten että ohjelmisto toteutetaan Microsoft Access 2010 -ohjelmistolla. (2, s. 61.)

Asiakasvaatimuksia analysoidaan ja niistä johdettavat ohjelmistovaatimukset määrittelevät toteuttavan järjestelmän vaatimukset. Esimerkiksi asiakasvaatimus: mahdollisuus suodattaa raporttiin sisällettyjä tuntikortteja voisi sisältää seuraavat ohjelmistovaatimukset: kysy suodatus parametrit, suodata raportti. (2, s. 62.)

Vaatimusten määrittelyssä voidaan käyttää hyväksi erilaisia työkaluja ja tekniikoita. Yleisesti käytössä oleva menetelmä käyttää UML-notaatioita, joka on ISO-standardoitu kuvauskieli, jossa käytetään erilaisia kaavioita kuvaamaan muun muassa ohjelmiston vaatimuksia, toimintoja (Kuva 3), data virtaa ja luokkia. Kuvauskieli on hyvä työkalu visualisoimaan ohjelmiston toimintaa jo määrittelyvaiheessa siten, että se on jokaisen ymmärrettävissä.



Kuva 3. Käyttötapauskaavio

Määrittelyvaiheen asiakasvaatimukset ja ohjelmistovaatimukset on kirjattava selkeästi toiminnallisen määrittelydokumenttiin. Huolellisesti tehty määrittely dokumentointeineen on projektin perusta, joka auttaa ymmärtämään asiakkaan tarpeet, joita voidaan tarkastella projektin dokumentaatiosta projektin myöhäisimmissä vaiheissa. Määrittelydokumentti on tarpeellinen myös, koska se on sopimus siitä, mitä on sovittu tehtäväksi, mikä on hyödyllistä esimerkiksi riitatilanteissa.

Suunnittelu

Suunnitteluvaiheessa vastataan siihen, miten ohjelmisto toteuttaa määrittelyvaiheessa määritellyt toiminnot. Suunnitteluvaiheeseen voi kuulua muun muassa arkkitehtuurisuunnittelu, jonka tarkoituksena on kuvata ohjelmisto korkealla teknisellä tasolla. Arkkitehtuurisuunnittelua seuraa usein moduulisuunnittelu, jossa suunnitellaan ohjelmiston moduulien eli osien suunnittelu. Suunnittelun tukena käytetään laajalti muun muassa UML-kaavioita, kuten luokkakaavioita. (4, s. 40.)

Toteutus

Toteutus- eli itse ohjelmointivaiheessa ohjelmisto kirjoitetaan ohjelmointikielellä suunnitteluvaiheen suunnitelmien mukaisesti. Toteutusvaiheen tuloksena syntyy käännetty ohjelma, jota voidaan suorittaa ja testata. Suunnittelu-, toteutus- ja testausvaihe on usein yhdistetty yhdeksi toistuvaksi vaiheeksi, jota toistetaan ohjelmistokehityksen aikana. Ohjelmiston toiminnallisuus voidaan esimerkiksi jakaa moduuleihin, joista kukin suorittaa tietyn toiminnon. Ohjelmistokehittäjät suunnittelevat ja toteuttavat moduulit, joissa he suorittavat yksikkötestauksen, minkä jälkeen suoritetaan integraatiotestaus, joka testaa moduuleiden yhteistoiminnan. (4, s. 40.)

Testaus

Testausvaiheen tarkoitus on löytää ohjelmistosta virheitä ja tunnistaa, täyttyvätkö asiakkaan määrittelyvaiheessa ohjelmistolle asettamat vaatimukset (2, s. 30). Testausvaiheen aikana korjataan ohjelmistossa ilmenneet viat, jonka jälkeen ohjelmisto testataan uudelleen. Testausta jatketaan, kunnes voidaan todeta, että ohjelmisto täyttää sille asetetut laadulliset ja toiminnalliset vaatimukset.

Yleisimpiä testaustekniikoita on muun muassa yksikkötestaus, integraatiotestaus, regressiotestaus, käyttäjähyväksyntätestaus sekä kuormitus- ja suorituskykytestaus. Yksikkötestauksessa testataan ohjelmiston osia, joita olen edellä kuvannut moduuleiksi. Integraatiotestauksessa testataan moduuleiden muodostamien kokonaisuuksien yhteistoimintaa. Regressiotestauksessa pyritään havaitsemaan, ilmaantuuko ohjelmiston aiemmin toimiviin toimintoihin virheitä ohjelmistopäivityksen myötä (5). Käyttäjähyväksyntätestauksessa ohjelmisto hyväksytetään asiakkaalla, jonka hyväksyttyä ohjelmisto voidaan ottaa käyttöön (6). Kuormitus- ja suorituskykytestauksessa ohjelmistoa kuormitetaan, jonka aikana ohjelmiston toimintaa ja suorituskykyä testataan (7).

Käyttöönotto ja ylläpito

Ohjelmiston käyttöönotto voidaan tehdä, kun on todettu, että ohjelmistossa ei ole virheitä tai ne on kontrolloitu niin, että ohjelmiston normaali toiminta voi jatkua. Käyttöönoton jälkeen siirrytään ohjelmiston ylläpidolliseen vaiheeseen, jossa korjataan, huolletaan ja parannetaan sen toimintaa asiakkaan pyyntöjen mukaisesti.

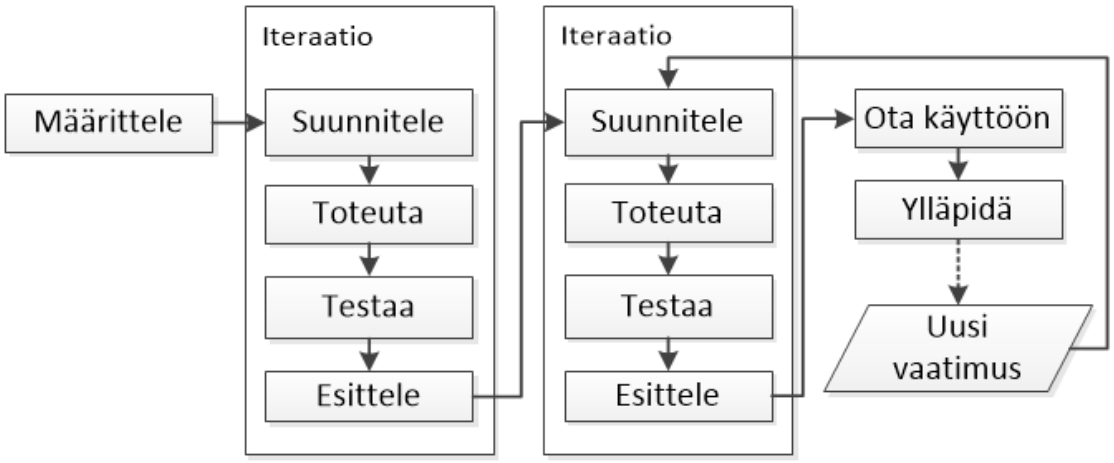
2.2 Vaihejakomallit

Ohjelmistokehitystyö ja koko sen elinkaari jaetaan usein vaiheisiin, joita on esitelty lukuisissa erilaisissa vaihejakomalleissa. Ohjelmiston elinkaari on se aika, joka alkaa ohjelmiston kehitysprosessin alkamisesta ja päättyy ohjelmiston poistamiseen käytöstä. Esittelen tässä luvussa kaksi vaihejakomallia, jotka ovat mielestäni yksinkertaisimpia malleja ja tavalla tai toisella esiintyvät useimmissa julkaistuissa vaihejakomalleissa.

Kuva 4. Esimerkki lineaarisesta ohjelmistokehitysmallista

Vaihejakomalleissa voi olla lineaarisia (

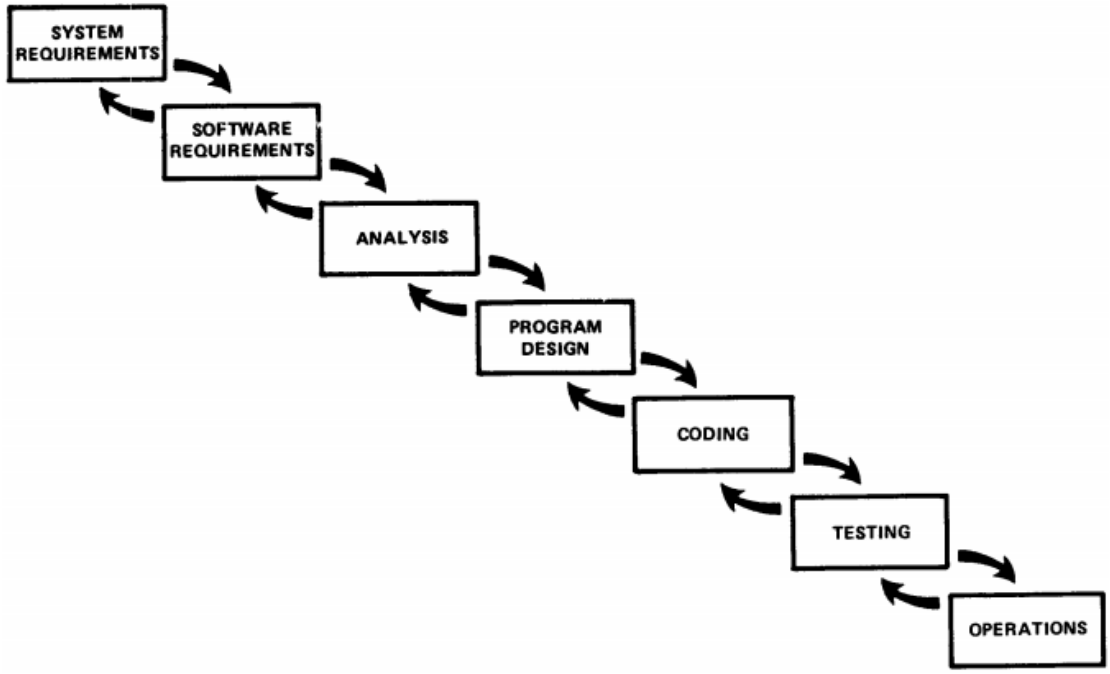
Kuva 4), iteratiivisia (Kuva 5) ja inkrementtaalisia piirteitä. Lineaarinen malli etenee vaiheesta toiseen, kun kaikki, käynnissä olevan vaiheen tehtävät on suoritettu loppuun. Iteratiivisessa mallissa ohjelmistoa kehitetään lineaarisesti, mutta tietyt toistuvat vaiheet muodostavat iteraatioita, joita toistetaan kehitysprosessin aikana. Inkrementtaalisessa mallissa ohjelmistojulkaisuja tehdään iteraatioiden päätteeksi ja jokainen julkaistu ohjelmisto sisältää enemmän toiminnallisuuksia, minkä vuoksi sitä kutsutaan inkrementaaliseksi eli kasvavaksi kehitykseksi.



Kuva 5. Esimerkki lineaarisesta mallista, jossa on toistuva iteraatiovaihe

2.2.1 Vesiputousmalli

Vesiputousmalli on yksi tunnetuimmista ja käytetyimmistä vaihejakomalleista ohjelmistokehityksessä. Mallin kuvaili nykyisessä muodossaan tiedettävästi ensimmäisenä amerikkalainen Winston W. Royce julkaisussa vuonna 1970. Julkaisussaan Royce ei käytä mallista (Kuva 6) nimeä vesiputousmalli, mutta se kuvaa hyvin nykyisin tunnettua vesiputousmallia, josta on nykyisin olemassa useita eri muunnelmia. (8.)



Kuva 6. Roycen esittämä vesiputousmalli (8)

Vesiputousmalli kuvataan usein lineaariseksi (

Kuva 4) vaihejakomalliksi, jossa jokainen vaihe on saatettava loppuun ennen seuraavan vaiheen aloittamista. Roycen esittämässä mallissa on kuitenkin iteratiivisuutta vaiheiden välillä, jota ei ole yksinkertaisimmassa vesiputousmallissa.

Vesiputousmallia, jossa ei iteratiivisuutta vaiheiden välillä ole lainkaan, on kritisoitu sen huonosta kyvystä vastata muuttuviin vaatimuksiin, koska kaikki ohjelmiston toiminnot on määriteltävä ja suunniteltava tarkasti ennen toteutusta. Kaikkien määritelmien tekeminen harvoin onnistuu, koska asiakkaiden tarpeet muuttuvat eikä kaikkia mahdollisia teknisiä rajoituksia välttämättä tiedetä ohjelmiston määrittelyvaiheessa.

2.2.2 Protoilumalli

Protoilussa toteutetaan esikatseltavia ja tutkittavia prototyyppejä ohjelmistosta, jotka konkretisoivat ohjelmistokehityksen tulokset asiakkaalle ja luovat mahdollisuuden ongelmien ja uusien vaatimuksien tunnistamiseen. Malli etenee vesiputousmallin mukaisesti, mutta toteutusvaiheessa on useita iteraatioita, joista kukin päättyy ohjelmiston protoiluun.

Protoilussa tuotetaan ohjelmistosta tai sen osista prototyyppejä, jotka voivat olla kertakäyttöisiä tai evoluutioprototyyppejä. Kertakäyttöiset prototyypit ovat luonnoksia, joiden kehittämistä ei jatketa, vaan niillä pyritään demonstroimaan esimerkiksi ohjelmiston käyttöliittymää ja toimintoja. Evoluutioprototyyppien ohjelmakoodia voidaan käyttää edelleen tulevissa prototyypeissä ja näin julkaistavan prototyypin toiminnallisuudet lisääntyvät projektin edistyessä.

Kuten muissakin malleissa, myös protoilussa on omat ongelmansa. Evoluutioprotoilussa ongelmana on, että prototyypin huonosti toteutetut toiminnot jäävät osaksi ohjelmistoa, joko tarkoituksella tai inhimillisen unohduksen myötä. Protoilussa asiakas voi saada myös mielikuvan miltei valmiista ohjelmistosta jo projektin varhaisessa vaiheessa, mikä on estettävissä esimerkiksi tekemällä käyttöliittymästä tarkoituksella keskeneräisen näköinen. (2, s.39.)

2.3 Kehitysmenetelmät

Ohjelmistokehitysmenetelmät ovat kokoelmia hyväksi todettuja menetelmiä, tekniikoita ja periaatteita, joita käytetään ohjelmistotuotannossa. Menetelmiä ja niitä määritteleviä oppaita on julkaistu lukemattomia. Monissa menetelmissä esiintyy samoja piirteitä, mutta niitä on muokattu sopivimmiksi ohjelmiston monimutkaisuuden, kriittisyyden tai projektiryhmän koon mukaan. Esittelen tässä luvussa lyhyesti kaksi menetelmien suuntausta ja tarkemmin yhden ketterän menetelmän, johon tutustuin opinnäytetyöni aikana.

Suunnitelmaohjautuvat menetelmät

Suunnitelmaohjautuvien menetelmien painopiste on prosessin toistettavuudella ja ennustettavuudella, joiden tukena on tarkka dokumentointi ja standardoidut prosessit (9). Menetelmiä käytetään etenkin tuotantoteollisuudessa, kuten autoteollisuudessa, jossa jokainen prosessi on tarkkaan määritelty ja jota pitää noudattaa, jotta tuotteen laadulliset vaatimukset täyttyvät.

Suunitelmaohjautuvien menetelmien etuna on toistettavuus, jonka myötä prosessia voidaan tehostaa hallitusti kun tiedetään kaikki prosessin toimintavaiheet, jotka on tarkasti dokumentoitu. Ohjelmistotuotannossa kuitenkin lopputuote on aina erilainen ja se voidaan toteuttaa useilla eri tavoilla, minkä vuoksi tietty ohjelmistotuotantoprosessi ei ole toistettavissa kokonaisuudessaan samanlaisena.

Ketterät menetelmät

Suunitelmaohjattujen, myös raskaina kehitysprosesseina tunnettujen menetelmien vastapainoksi kehiteltiin 1990-luvulla useita, kevyiksikin luonnehdittuja, ketteriä menetelmiä. Nykyisin tunnettuja menetelmiä julkaistiin vuosikymmenen aikana kymmenkunta, joiden useat kehittäjät kokoontuivat helmikuussa 2001 Yhdysvalloissa keskustellakseen kevyistä ohjelmistokehitysmenetelmistä. Osa tapaamiseen osallistuneista henkilöistä oli perustamassa myöhemmin voittoa tavoittelemattoman Agile Allicance -järjestön, joka pyrkii edistämään ketterän ohjelmistokehityksen menetelmiä (2, s.43-44). Järjestön perusteoksena julkaistiin ketterän ohjelmistokehityksen julistus (10), jonka ydin on sen arvoissa, jotka esitettiin julistuksessa seuraavasti:

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.

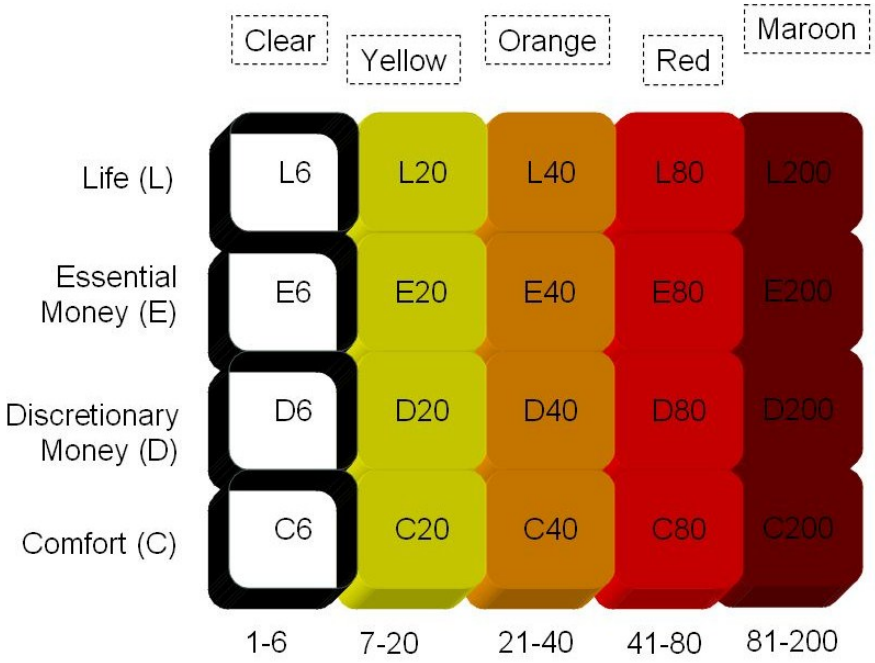
Julistuksessa on esitelty tarkemmin ketterän kehityksen menetelmien 12 periaatetta, jotka ovat ennalta mainittujen arvojen tarkennuksia. Lyhyesti sanottuna ketterien menetelmien perusideana on asiakaskeskeisyys, yhteistyöhön ja ihmisiin luottaminen tarkkojen prosessien ja menetelmien sijaan.

Ketteriä ohjelmistokehitysmenetelmiä on kehitetty lukemattomia, mutta tiedettävästi laajimmin käyttöön otettu menetelmä on Scrum, josta ensimmäinen kuvaus julkaistiin vuonna 1986 (2, s.46). Scrum sisältää vaihejakomallin, joukon ohjelmistokehityksen sekä projektihallinnan menetelmiä ja tekniikoita. Toinen suosittu menetelmä on XP eli Extreme Programming, jossa on paljon yhteneväisyyksiä seuraavassa luvussa kuvatun Crystal Clear -menetelmien kanssa.

Crystal Clear -menetelmät

Alistair Cockburn, yksi ketterän ohjelmistokehityksen julistuksen tekijöistä, julkaisi vuonna 2004 kirjan Crystal Clear -menetelmistä, joka kuuluu Crystal-menetelmien sarjaan (Kuva 7) (11). Sarjaan kuuluu yhteensä ainakin seitsemän vaihtoehtoista menetelmää, jotka on suunnattu erikokoisille ja -kriittisyystason projekteille. Kaksi menetelmistä, joita ei kuvassa näy, ovat Crystal Sapphire ja Crystal Diamond, joita käytetään erittäin tehtävä-kriittisten sovellusten kehityksessä. Crystal Clear on suunniteltu projekteihin, joissa työskentelee alle kymmenen työntekijää eikä niissä toteuteta liiketoimintakriittisiä järjestelmiä. Crystal Clearissa noudatetaan ketterän kehityksen periaatteita, toimintatapoja, strategioita ja työskentelytekniikoita.

Crystal Clear -menetelmien kuvauksessa on tiettyjä strategioita, ominaisuuksia ja tekniikoita, joiden on todettu toimivan pienissä projektiryhmissä. Kuvauksessa painotetaan, että strategiat, ominaisuudet tai tekniikat eivät itsessään takaa ohjelmistokehityksen onnistumista, vaan että menetelmiä on sovellettava omaan projektiympäristöön sopivaksi. Esittelen seuraavissa luvuissa Alistairin kirjassa mainittuja hyväksi todettuja käytäntöjä, tekniikoita ja strategioita.



Kuva 7. Crystal-menetelmien sarja (12)

Käytännöt

Crystal Clear -menetelmien noudattavan ryhmän odotetaan noudattavan kolmea käytäntöä, säännöllistä toimitusta, heijastavaa kehitystä ja niin sanottua osmoottista kommunikointia. Kirjassa on esitelty myös neljä muuta käytäntöä, jotka on tarkoitettu metodologian kehittyneimmille soveltajille ja joita en tässä luvussa esittele.

Tärkein yksittäinen käytäntö, missä tahansa ohjelmistoprojektissa, on säännöllinen toimitus. Säännöllisen toimituksen etuina ovat, että projektin sidosryhmät saavat tietoa projektin etenemisestä, järjestelmän käyttäjät voivat antaa määritelmiin tarkennuksia, ohjelmistonkehittäjät motivoituvat toimituksien lähentyessä ja projektiryhmä voi arvioida iteraatiossa suoriutumistaan ja muokata toimintatapojaan tarpeen vaatiessa. Toimituksia ei kannata tehdä kuitenkaan liian usein, koska se voi kuormittaa asiakasta, eikä myöskään liian harvoin, koska silloin tarvittavia muutoksia ei havaita tarpeeksi aikaisin. Toimituksien välien eli iteraation pituuksien tulisi olla pysyviä, jotta projektin nopeus voidaan mitata. Nopeudella tarkoitetaan sitä, kuinka paljon projektiryhmä tekee asioita yhden iteraation aikana.

Toinen tärkeä käytäntö on reflektiivinen parantaminen, joka tarkoittaa sitä, että projektiryhmän jäsenet tutkivat säännöllisesti työskentelytapojaan ja heijastavat niitä muiden käyttämiin tapoihin. Tämän tarkoituksena projektiryhmä pystyy tunnistamaan heille kulloinkin sopivat työmenetelmät, asiat, jotka hidastavat heidän työtään tai toimintatavat, jotka voisivat tehostaa heidän työskentelyä. Projektiryhmän tulisi kokoontua säännöllisin väliajoin ja pyrkiä tunnistamaan näitä seikkoja. Säännöllisellä reflektiolla parannetaan nopeutta reagoida haasteisiin tunnistamalla ne aikaisin ja sillä voidaan kehittää uusia, projektiryhmälle sopivia työskentelymenetelmiä.

Kolmantena käytäntönä on osmoottinen kommunikointi, joka tarkoittaa, että yhteisissä tiloissa, lähekkäin työskentelevät ohjelmistokehittäjät voivat reagoida heille tärkeään tietoon. Tieto on kaikkea keskustelua, jota käydään kehittäjien välillä suullisesti. Projektin jäsenet voivat korjata toisten vastauksia, antaa arvokasta lisätietoa tai antaa nopeaa apua ongelmissa. Työskentely yhteisissä tiloissa lähekkäin voi tuottaa ongelmiakin, kun informaatiota tai häiritsevää ääntä on liikaa, mikä voidaan ratkaista sopimalla hiljaisia työaikoja.

Tekniikat ja strategiat

Crystal Clear ei määrittele tarkkaan mitään käytettäviä tekniikoita tai strategioita käytettäväksi ohjelmistoprojektissa, mutta sen mukaan on hyvä olemassa vaihtoehtoja, joita voidaan tarvittaessa käyttää ja soveltaa omaan projektiin. Alla on kuvattu tekniikoita, joita käytetään modernimmissa ketterissä kehitysryhmissä.

360°-tutkimukseksi kuvattu vaihe on ohjelmiston määrittelyvaihe, jossa perinteisien toimenpiteiden lisäksi, kehittäjät aloittavat tutustumisen teknologioihin, joita projektissa mahdollisesti käytetään. Tutustuminen voi pitää sisällään esimerkiksi pienen prototyypin tekemistä, joka tutustuttaa kehittäjät uusiin teknologioihin tai ympäristöön, mikä voi auttaa tunnistamaan teknisiä rajoituksia jo hyvin varhaisessa vaiheessa projektia.

Ennenaikaiset voitot ovat projektiryhmää yhdistävä voima, joka parantaa jäsenien itseluottamusta ja auttaa jäseniä tuntemaan toisensa paremmin. Voitot on hyvä pyrkiä saavuttaa projektin alkaessa, esimerkiksi aloittamalla ohjelmiston toteutus helpoimmasta asiasta, jonka toteutettuaan projektiryhmä saa itseluottamusta ja tarvittavaa tuntemusta, jotta voidaan siirtyä haastavimpiin haasteisiin.

Ohjelmiston esiversio, jota Alistair nimittää käveleväksi luurangoksi. Luuranko on kuten protoilu-vaihejakomallin evoluutiopromalli, joka kehittyy lopulliseksi järjestelmäksi. Luurangossa tulisi yhdistyä arkkitehtuurin pääkomponentit, kuten tietokerros, logiikkakerros ja käyttöliittymä. Toimintoja ei luurangossa tarvitse olla, kunhan kokonaisuuden pystyy testaamaan. Järjestelmän ”luurangon” luominen projektin varhaisessa vaiheessa ja arkkitehtuurin kehittäminen nousevasti mahdollistavat arkkitehtuurin ja toiminnallisuuksien kehittämisen rinnakkain. Luurangon kehittäminen mahdollistaa tunnistamaan mahdollisia teknisiä rajoituksia ja se toimii projektiryhmälle ennenaikaisena voittona.

3 Case: Asiakkaan esittely

Tässä luvussa esittelen järjestelmän tilaajan eli asiakkaan. Esittelen yhtiön, sen historiaa, tutkimus- ja kehitystoimintaa sekä toimipisteen, jossa järjestelmä otettiin käyttöön.

3.1 UPM-Kymmene Oyj

UPM-Kymmene on yksi maailman suurimmista biometsäteollisuusyhtiöistä. Yhtiöllä on tuotantolaitoksia 17 maassa ja henkilökuntaa 45 maassa noin 22 000, joista noin 8600 työskentelee Suomessa. Yhtiön liikevaihto vuonna 2012 oli yli 10 miljardia euroa, ja sen osakkeet on listattu NASDAQ OMX Helsingin pörssissä. (13; 14, s.39.)

Yhtiön ydinliiketoimintaa ovat kuituihin perustuvat liiketoiminnot. Liiketoimintoihin kuuluu paperi, energia, biopolttoaineet, sellu, metsät ja puuhankinta, UPM timber eli mänty- ja kuusisahatavaran valmistus ja jatkojalostus, tarramateriaalit, vaneri ja puumuovikomposiitti. (15.)

UPM syntyi syksyllä 1995, kun Kymmene Oy ja Repola Oy sekä sen tytäryhtiö Yhtyneet Paperitehtaat Oy (United Paper Mills) ilmoittivat yhdistymisestään. Uusi yhtiö, UPM-Kymmene, aloitti toimintansa 1.5.1996. Nykyinen UPM-konserni muodostuu kaikkiaan noin sadasta aikoinaan itsenäisenä yrityksenä toimineesta yhtiöstä. Yritykseen ovat sulautuneet muun muassa seuraavat metsäteollisuusyritykset: Kymi, Yhtyneet Paperitehtaat, Kaukas, Kajaani, Schauman, Rosenlew, Raf. Haarla ja Rauma-Repolan metsäteollisuus. (16.)

UPM:llä on Suomessa pitkät perinteet metsäteollisuudessa. Konsernin ensimmäiset puuhiomot ja paperitehtaat sekä sahalaitokset käynnistyivät 1870-luvun alkupuolella. Sellunvalmistus aloitettiin 1880-luvulla ja paperinjalostus 1920-luvulla. Vanerin valmistukseen konsernissa ryhdyttiin 1930-luvulla. (16.)

UPM:n ulkomaisista tehtaista ylivoimaisesti vanhin on Koillis-Ranskassa sijaitseva hienopaperitehdas Papeteries de Docelles, joka aloitti toimintansa perinteisenä käsipaperimyllynä jo 1400-luvun lopulla. Ensimmäinen paperikone tehtaalle tuli 1830-luvulla. (16.)

3.2 Tutkimus ja kehitys

UPM:n tutkimus- ja kehitysohjelmien sekä liiketoiminnan kehittämisen tavoitteena on tukea uusien teknologioiden ja tuotteiden kehittämistä UPM:n uusissa liiketoiminnoissa sekä varmistaa nykyisten tuotteiden kilpailukykyä. Yhtiön käyttämät varat tutkimukseen ja kehitykseen ovat kasvaneet tasaisesti varsinkin kasvuliiketoiminnoissa. Yhteensä UPM käytti kehittyvien ja nykyisten liiketoimintojen tutkimukseen ja kehitykseen noin 81 miljoonaa euroa vuonna 2012. (14, s.34.)

UPM:n tutkimus- ja kehitystyössä työskentelee noin 300 ammattilaista viidessä tutkimuskeskuksessa (Kuva 8). Lappeenrannan yksikkö keskittyy kuitujen ja papereiden tutkimukseen. Samassa kaupungissa toimii myös UPM:n biojalostamokehityskeskus. Tampereella kehitetään tarralaminaatteja, ja Lahdessa tehdään vaneri- ja komposiittitutkimusta. UPM:n uusiokuidun tutkimus on Augsburgissa Saksassa. Aasian T&K-keskus Kiinassa vastaa paikallisten kuituraaka-aineiden tutkimuksesta. Uruguayn Fray Bentosiin vuoden 2012 alkupuoliskolla perustettava osaamiskeskus tutkii eukalyptuspuulajeja ja niiden vaikutusta sellun lopputuotteiden ominaisuuksiin.

Kuva 8. UPM-Kymmene Oyj tutkimus- ja kehitysyksiköt (17)

3.3 Lappeenrannan tutkimus- ja kehityskeskus

Opinnäytetyön case-osio eli tuntiraportointijärjestelmä toimitettiin Lappeenrannassa sijaitsevan tutkimuskeskuksen (Kuva 9) ja biojalostamokehityskeskuksen käyttöön. UPM:n Lappeenrannan tutkimus- ja kehitysyksikkö keskittyy kuitujen ja papereiden tutkimukseen ja se on suurin viidestä yhtiön tutkimuskeskuksesta. Henkilöstöä Lappeenrannan tutkimuskeskuksessa on noin 160. Tutkimuskeskuksen työntekijöihin kuuluu päällikköjä, laborantteja, laboratorioinsinöörejä, tutkijoita, laboratorioiden esimiehiä sekä toimistotyöntekijöitä. (18.)

Kuva 9. UPM Tutkimuskeskus, Lappeenranta (19)

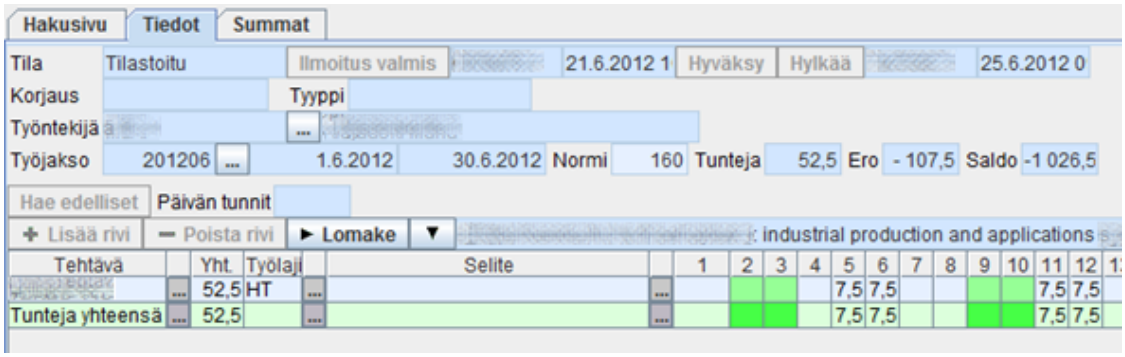
4 Case: Tuntiraportointijärjestelmä

Tässä luvussa esittelen ohjelmiston, joka toteutettiin osana opinnäytetyötäni. Esittelen myös lyhyesti ohjelmiston, joka korvattiin kehitetyllä järjestelmällä. Lopuksi esittelen ohjelmistokehitysprosessin, jota käytettiin kehitettäessä uutta ohjelmistoa.

4.1 Vanha ohjelmisto

Tutkimuskeskuksen käytössä oli ollut Prosla-projektilaskutusjärjestelmä vuosituhannen vaihteesta saakka. Ohjelmistoa käytettiin projektilaskutukseen vuoden 2010 loppuun asti ja ulkoisen rahoituksen projektien tuntiseurantaan vuoden 2012 loppuun asti. Pelkään tuntiseurantaan järjestelmä oli liian laaja ja sen käyttöliittymä oli monimutkainen, eikä sen käyttöön ollut tarjolla ohjeita tai koulutusta lainkaan. Järjestelmän käyttäjät joutuivat opettelemaan itse sen käytön, jonka seurauksena kaikkia ohjelmiston toimintoja ei osattu hyödyntää tehokkaasti.

Järjestelmä oli toteutettu Java-teknologialla, joka on ollut viime vuosina runsaasti uutisten otsikoissa siitä löytyneiden tietoturvahaavoittuvuuksien vuoksi. Proslan oli toteutettu asiakaspalvelintyyppisesti niin, että käyttäjän koneelta suoritettava ohjelma yhdisti palvelimella sijaitsevaan tietokantaan, jonne data siirrettiin.

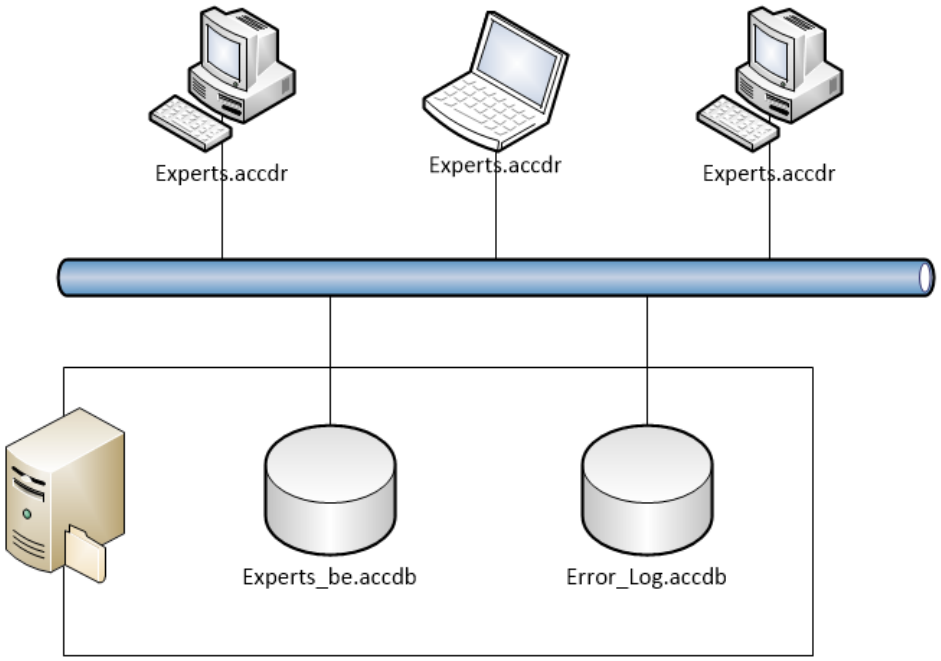


Kuva 10. Prosla-ohjelmiston tuntikorttiensyöttölomake

4.2 Uusi ohjelmisto

Uudeksi tuntiraportointi- ja seurantajärjestelmäksi kehitettiin UPM RC Experts eli UPM Research Center External Projects Reporting System. Järjestelmässä voidaan kirjata tuntikortteja ennalta määritettyihin ulkoisen rahoituksen projekteihin ja tulostaa ja viedä tietoa kirjatuista tuntikorteista. Ulkoisen rahoituksen projekteja ovat esimerkiksi EU- ja TEKES-rahoitteiset tutkimusprojektit. Järjestelmä toteutettiin kokonaisuudessaan Microsoft Access 2010 -ohjelmistolla, joka on osa Office 2010 Professional -ohjelmistopakettia (20).

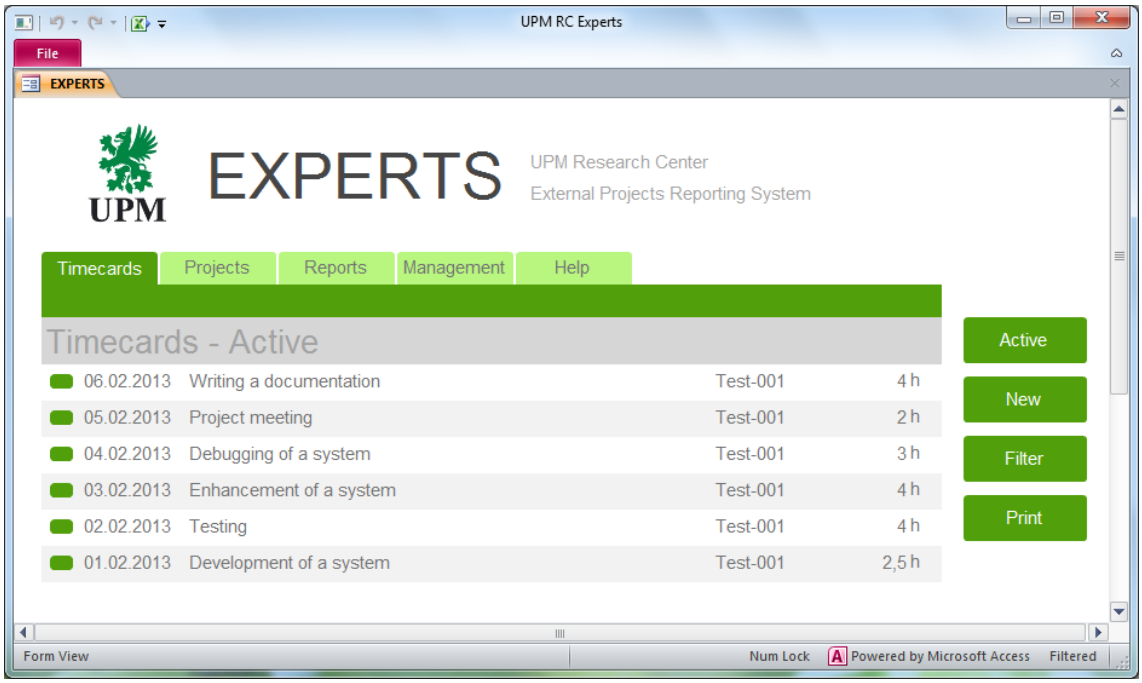
Järjestelmä koostuu käyttäjien käyttämistä, asiakaskoneilla paikallisesti suoritettavista edustatietokannoista (accdr-tiedostopääte), jotka sisältävät ohjelman toiminnallisuudet raporttipohjineen ja lomakkeineen. Informaatio edustatietokannalle haetaan taustatietokannasta (accdb-tiedostopääte). Käyttöliittymä toteutettiin käyttämällä Access-ohjelmiston lomakkeita ja valmiita kontrolliobjekteja. Ohjelmiston toimintalogiikka toteutettiin hyödyntämällä valmiita Access-makroja ja VBA-ohjelmointia.



Kuva 11. UPM RC Experts -ohjelmiston infrastruktuuri

4.2.1 Käyttöliittymä

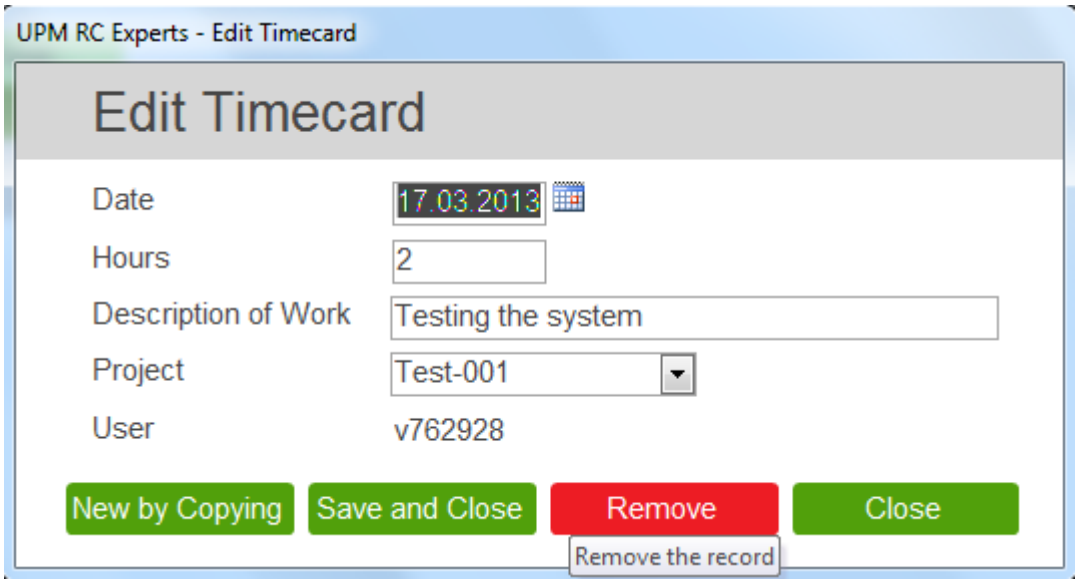
Järjestelmän käyttöliittymä on käyttäjän tietokoneella paikallisesti suoritettavassa edustatietokannassa, joka sisältää myös kaiken ohjelmistologiikan. Käyttäjille jaettava edustatietokanta käännettiin Access Runtime -tiedostoksi, jonka avulla saavutetaan useita etuja monikäyttäjä ympäristöissä: käännetyn edustatietokannan lähdekoodia ei voi muokata, sen käyttöliittymä on pelkistetty poistamalla ohjelman toimintoja, se rajaa käyttäjän näkemään vain hänelle tarkoitettua informaatiota, suorituskyky paranee ja se on myös käytettävissä ilmaisella Access Runtime -ohjelmalla (21).



Kuva 12. UPM RC Experts, ylläpitäjän aloitusnäkymä

Käyttöliittymän suunnittelussa huomioitiin asiakkaan omat ulkoasuvaatimukset, jotka koskivat UPM-logon käyttöä, värimaailmaa ja käytettävää kirjaisintyyppiä. Suunnittelussa pyrittiin mahdollisimman pelkistettyyn näkymään ja helppokäyttöisyyteen. Näkymän pelkistäminen toteutettiin toteuttamalla dynaaminen näkymä, joka mukautuu käyttäjän oikeuksien ja käytetyn välilehden mukaan. Dynaamisella näkymällä parannetaan käyttäjän käyttökokemusta niin, että hän näkee vain toiminnot, jotka hän itse pystyy suorittamaan.

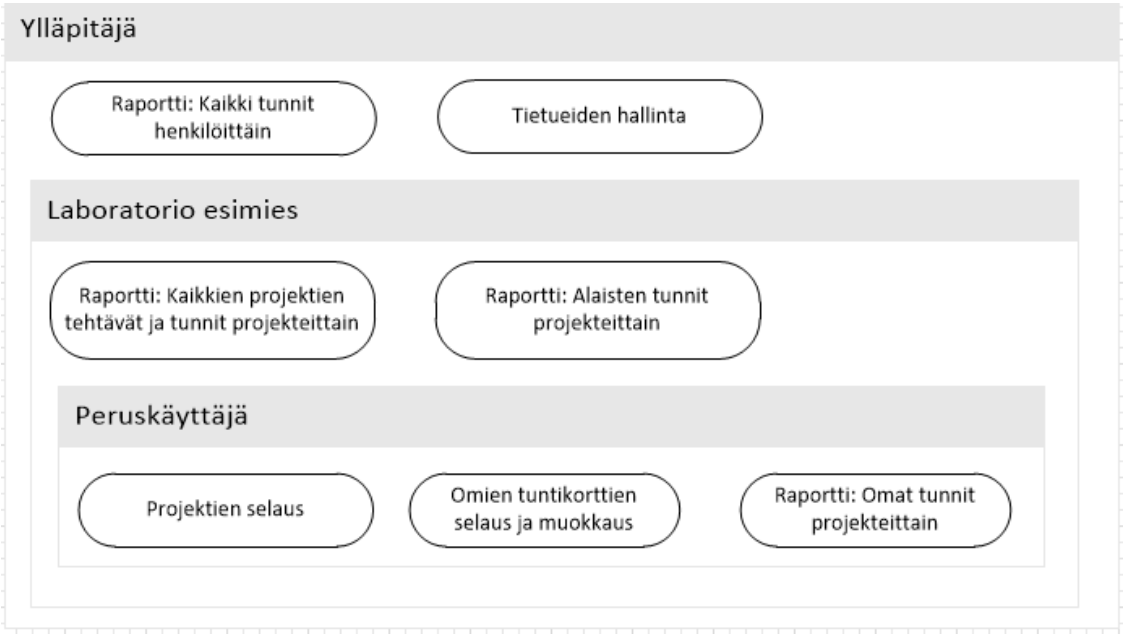
Helppokäyttöisyyttä pyrittiin lisäämään staattisilla toimintonäppäimillä ikkunan oikeassa reunassa. Käyttöliittymän värimaailma valittiin asiakkaan vaatimusten mukaiseksi, mutta poisto-toimintoja korostettiin sitä kuvaavalla värillä (Kuva 13).



Kuva 13. UPM RC Experts, tuntikortin muokkausikkuna

4.2.2 Käyttäjäroolit ja -tunnistus

Ohjelmalla on kolme eri käyttäjäryhmää: peruskäyttäjät, laboratorioesimiehet ja ylläpitäjät. Käyttöliittymän näkymä ja käytettävissä olevat toiminnot määräytyvät käyttäjäryhmän. Toiminnot periytyvät ylemmille tasoille niin, että kaikki perus-käyttäjän toiminnot ovat myös laboratorioesimiehillä ja esimiesten toiminnot yl-läpitäjillä.

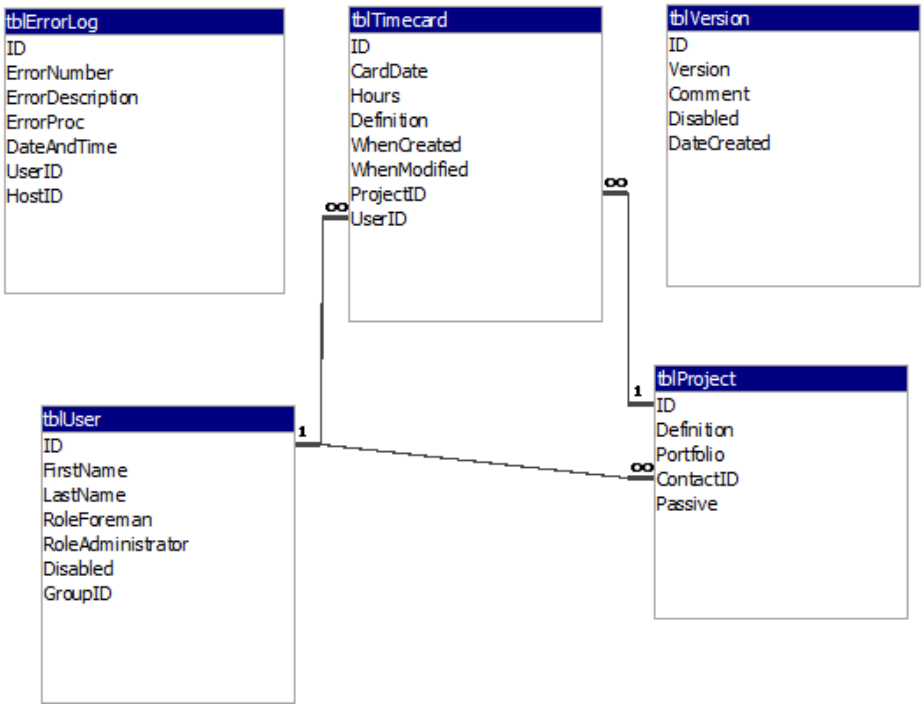


Kuva 14. UPM RC Experts, järjestelmän käyttäjäroolit

Käyttäjän tunnistautuminen on toteutettu automaattiseksi niin, että ohjelma hakee tietokoneelle kirjautuneen käyttäjän käyttäjätunnuksen käyttöjärjestelmän käyttäjänimi-ympäristömuuttujasta, jota se hakee taustatietokannasta taulusta, jossa tietueet käyttäjistä on. Mikäli käyttäjätunnusta ei löydy taulusta tai se on estetty, ohjelman käyttö estetään ja käyttäjä saa virheilmoituksen ennen ohjelman sulkeutumista.

4.2.3 Tietokanta

Edustatietokantojen käyttämä informaatio on tallennettu taustatietokantoihin, jotka ovat jaetussa kansiossa, johon kaikilla järjestelmän käyttäjillä on luku- ja kirjoitusoikeudet. Muut tauluista ovat samassa tietokantatiedostossa paitsi tblErrorLog-taulu, joka sisältää informaatiota edustatietokannassa tapahtuneista virheistä. Tallentamalla virhetilanteet eri tiedostossa olevaan tietokantaan, saadaan tallennettua myös tietoa silloin, kun yhteys katkeaa siihen tietokantaan, jossa muut taulut ovat.

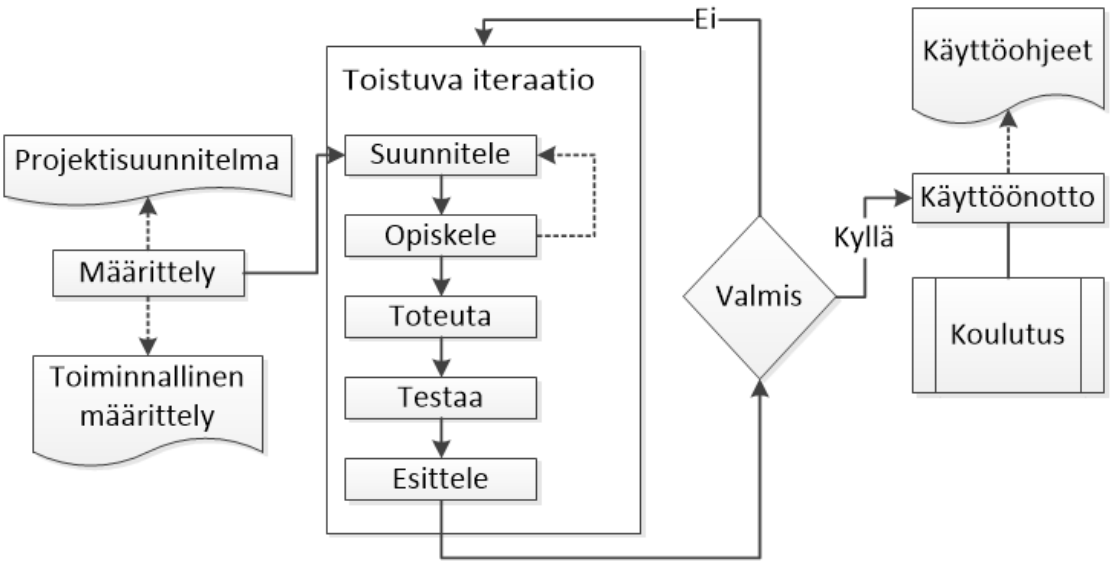


Kuva 15. UPM RC Experts, tietokannan Entity Relationship -kaavio

Tietokantaan määriteltiin johdannaismuutoksia ja rajoitettuja muutoksia. Johdannaismuutokset määriteltiin käyttäjän ja projektin tunnisteeseen, mikä tarkoittaa sitä, että muutettaessa tunnistetta, päivittyy muutos automaattisesti kaikkiin viite-avaimiin muissa tauluissa. Rajoitettuja muutoksia määriteltiin projektin poistamiseen, minkä vuoksi ei tuntikortillisia projekteja voi poistaa, ennen kuin sen kaikki tuntikortit on poistettu. Järjestelmän toimintaa suorituskykyä parannettiin määrittelemällä indeksejä kenttiin, jotka ovat usein käytetty edustatietokannan SQL-kyselyissä. Indeksejä määriteltiin etenkin viite-avaimille ja perusavaimille, jotka ovat uniikkeja ja niitä käytetään usean taulun kyselyissä.

4.3 Ohjelmistokehitysprosessi

Tässä luvussa kuvaan toteutetun tuntiraportointijärjestelmän ohjelmistokehitysprojektin vaiheet (Kuva 16) ja käyttämäni kehitysmenetelmät.



Kuva 16. Projektissa käytetty vaihejakomalli

4.3.1 Määrittely

Projektin määrittelyvaiheessa asiakas esitteli järjestelmän toiminnalliset, ei-toiminnalliset vaatimukset rajoitteineen ja reunaehtoineen. Annettujen tietojen pohjalta tutkin mahdollisia toteutusvaihtoehtoja ja esitin asiakkaalle teknisiä toteutusvaihtoehtoja, joilla järjestelmä voidaan toteuttaa asiakkaan ympäristöön.

Toteutusvaihtoehdot:

1. Web-käyttöliittymä ASP.NET -teknologialla, johon toiminnallisuus toteutetaan C#-ohjelmointikielellä ohjelmoitavalla logiikkakerroksella, joista luodaan yhteys Oracle-tietokantaan. Toteutusmenetelmä olisi noudattanut nykyisin suosittua MVC-mallia, jolla verkkopalveluita toteutetaan. Vaihtoehto hylättiin, koska sen toteuttamiseksi olisi jouduttu hankkimaan erillinen alusta, joka olisi lisännyt työmäärää ja kustannuksia huomattavasti.

2. Käyttöliittymä toteutetaan Access Web Services -tekniikalla SharePoint-sivustolle, minkä tietovarastona toimii Access-tietokanta. Käyttämällä SharePoint-sivustoa, etuna olisi ollut valmiiksi toimiva käyttäjätunnistus. Toteutustavan etuna olisi myös ollut jakelun yksinkertaistuminen, koska käyttöliittymän jakelu olisi tehty vain yhteen pisteeseen eli SharePoint-järjestelmään. Vaihtoehto valittiin toteutustavaksi, mutta esitestausvaiheessa ilmeni teknisiä esteitä, joiden vuoksi vaihtoehto hylättiin.
3. Käyttöliittymän ja tietokannan toteutus Access-ohjelmistolla niin, että tietokanta jaetaan edustatietokantaan ja taustatietokantaan. Vaihtoehtoa puolsi se, että asiakkaalla oli käytössään myös toinen järjestelmä, joka on toteutettu kyseisellä tavalla ja se on todettu toimivaksi ja käyttäjäystävälliseksi. Vaihtoehto valittiin lopuksi, kun huomattiin tekniset rajoitteet SharePoint-järjestelmässä.

4.3.2 Suunnittelu, toteutus ja testaus

Suunnittelu-, toteutus- ja testausvaihe tehtiin iteroinneissa, jossa kussakin toteutettiin tietty kokonaisuus järjestelmästä (Kuva 17). Iteraation pituus oli kaksi viikkoa, jonka päätteeksi tehty työ esiteltiin asiakkaalle. Esittelyissä asetettuja vaatimuksia tarkennettiin ja tarvittaessa lisättiin sekä toteutettuja toimintoja esikatseltiin.

Kuva 17. Iteraatiot

Arkkitehtuuri-iteraatiossa suunnittelin järjestelmän perusarkkitehtuurin ja infrastruktuurin ja loin Crystal Clear -menetelmissäkin mainitun ”kävelevän luurangon”, eli tein järjestelmän, jossa oli jo käytössä kaikki sen komponentit, mutta toiminnallisuudet olivat hyvin rajattuja.

Käyttöliittymä-iteraatiossa suunnittelin ja toteutin järjestelmän käyttöliittymän, johon en tehnyt mitään toiminnallisuuksia, vaan loin protoilu-vaihejakomallista tutun kertakäyttöisen prototyypin. Esikatselmuksessa ilmeni lomakkeiden toimintaan uusia asiakasvaatimuksia, joita emme huomioineet järjestelmän määrittelyvaiheessa. Uudet asiakasvaatimukset eivät kuitenkaan lisänneet uudelleen tehtävän työn määrää, koska ohjelmisto oli vasta pois heitettävän prototyypin asteella ja vaatimukset voitiin huomioida seuraavaan iteraatioon toteutettavaksi.

Lomakkeet-iteraatiossa toteutin käyttöliittymän perustuen esittelemääni evoluutioprototyyppiin, johon toteutin toimivat lomakkeet, jotka oli tarkasti määritelty. Loin myös useita kertakäyttöisiä lomakkeita, jotta sain yhdessä asiakkaan kanssa hyvän kokonaiskuvan käyttöliittymän toiminnoista ja sen käytettävyydestä. Toiminallisuutta en kertakäyttöisiin lomakkeisiin lisännyt lainkaan.

Raportit-iteraatiossa toteutin raportit asiakkaan vaatimusten mukaisesti ja suunnittelin niihin liittyviä toimintoja, joita ei määrittely vaiheessa tarkasti määritelty, esimerkiksi raportin suodatus- ja vientimahdollisuudet.

Lisäominaisuudet-iteraatiossa toteutin asiakasvaatimukset, jotka ilmenivät aiemmissa iteraatioissa ja viimeistelin aiemmin toteutettuja toiminnallisuuksia ja käyttöliittymää.

4.3.3 Testaus ja optimointi

Testausvaiheen alussa testasin järjestelmää manuaalisella menetelmällä eli käytin järjestelmää, kuten käyttäjät sitä käyttäisivät. Kirjasin löydetyt ongelmat ja korjasin ne kriittisyysjärjestyksessä. Kun en havainnut järjestelmässä enää virheitä, siirryttiin testaamaan järjestelmää loppukäyttäjillä. Laadin järjestelmän käyttäjille testitapauksia, eli ohjelmiston käyttötapauksia, joissa tapahtuvia virheitä he raportoivat. Ensimmäisessä käyttäjien suorittamassa testauksessa ilmeni yllättävän monta virhettä, koska käyttäjien käyttämä testausympäristö poikkesi kehitysympäristöstä, jossa olin testannut ohjelmistoa.

Käytättäjien suorittamien testitapausten tulosten perusteella korjasin järjestelmää. Korjausten jälkeen aloitin järjestelmän kapasiteetti- ja suorituskykytestauksen, jossa pyrin havaitsemaan järjestelmän skaalautuvuuden rajat. Kapasiteettitestauksessa generoin järjestelmän tietokantaan dataa, jonka yhteydessä testasin ohjelmiston toimintaa. Generoin järjestelmään dataa, kunnes järjestelmän käytettävyys laski sellaisella tasolle, jonka katsoin olevan liian huono järjestelmän suhteellisen miellyttävän käytön kannalta.

Testausvaiheen lopuksi järjestettiin toinen käyttäjätestaus, jossa käyttäjien suorittamista testeissä ei ilmennyt virheitä ja järjestelmä todettiin olevan valmis siirrettäväksi tuotantoon. Testausvaiheen lopuksi testaukseen osallistuneet loppukäyttäjät ehdottivat kehitysehdotuksia järjestelmään, jotka kirjasin työlistalle ja priorisoin toiminnallisuudet niiden tarpeellisuuden mukaan.

4.3.4 Käyttöönotto ja koulutukset

Järjestelmän testausvaiheen päätyttyä, edustatietokannasta luotiin asennuspaketti asiakkaan käyttämällä ohjelmistojakelujärjestelmällä, jonka kautta ohjelmisto jaettiin käyttäjien käyttämille työasemille.

Ennen järjestelmän varsinaista käyttöönottoa, loin sen käyttöohjeet ja loppukäyttäjille järjestettiin koulutustilaisuuksia käyttäjäryhmittäin. Koulutustilaisuuksien jälkeen annettiin käyttäjille käyttää järjestelmää tutustumistarkoituksessa vielä viikon ajan, jonka jälkeen tietokannat tyhjennettiin ja järjestelmä siirrettiin kokonaisuudessaan tuotannolliseen tilaan.

4.3.5 Ylläpito ja jatkokehitys

Käyttöönoton jälkeen järjestelmän kehitystä jatkettiin, koska koulutustilaisuuksissa ilmeni useita kehitysehdotuksia, jotka valittiin toteutettavaksi järjestelmään. Järjestelmässä ilmeni myös lieviä toiminnallisia puutteita, jotka huomattiin myös koulutustilaisuuksien yhteydessä.

5 Yhteenveto ja pohdinta

Opinnäytetyön ensisijaisena tavoitteena oli toteuttaa asiakkaalle tuntiraportointijärjestelmä, joka täyttää sille asetetut vaatimukset. Tavoite onnistui mielestäni hyvin, koska järjestelmä toimitettiin asiakkaalle alkuperäisen aikataulun mukaisesti ja se sisälsi asiakkaan vaatimat toiminnallisuudet. Asiakkaan antaman palautteen mukaan ohjelmisto on täyttänyt heidän asettamat vaatimukset ja he ovat olleet tyytyväisiä uudistuneeseen ja ennen kaikkea helppokäyttöisempään ohjelmistoon.

Jälkikäteen ohjelmistotuotantoprosessiani arvioidessani löydän useita puutteita käyttämissäni prosesseissani ja menetelmissäni, mutta kuitenkin onnistuin mielestäni mukautumaan haasteiden edessä niin, että sain aikaan itseäni ja asiakasta tyydyttävän lopputuloksen. Asiakkaan mielestä projektin määrittelyyn, olisi voinut ottaa mukaan useampia loppukäyttäjiä, jotka olisivat auttaneet tunnistamaan useampia vaatimuksia heti projektin alkuvaiheessa. Olen asiakkaan kanssa samaa mieltä, koska ohjelmiston määrittelyssä ei ollut mukana kaikkia käyttäjäryhmiä, mikä aiheutti sen, että tietyt vaatimukset tunnistettiin vasta ohjelmiston testausvaiheessa. Huomioiden ohjelmiston kokoluokan, ei tarvittavien muutosten tekeminen ohjelmistoon kuitenkaan ollut hankalaa projektin loppuvaiheessa.

Projektin määrittely ja toteutus toteutettiin pääosin työskentelemällä osa-aikaisesti ja projektipalaverit pidettiin käyttämällä etäkommunikointityökaluja. Mielestäni osa-aikanen työskentely projektin parissa rasitti prosessin etenemistä hieman ja etätyöskentely asetti omat motivoitumishaasteensa. Testaus, käyttöönotto ja koulutusvaiheessa työskentelin asiakkaan toimipisteessä kokopäiväisesti, joka helpotti huomattavasti ohjelmiston viimeistelemistä.

Ohjelmistotuotantoprojektissa käytettiin niin lineaarista, iteratiivista kuin inkrementaalista ohjelmistotuotantotapaa monen ketterän kehityksen periaatteen mukaisesti. Näin pienessä ohjelmistoprojektissa dokumentointia ja projektin hallintaa ei juuri tarvittu, koska toteutin ohjelmiston yksin. Mikäli ohjelmistonkehittäjiin olisi kuulunut useampi henkilö, olisi se pitänyt mielestäni toteuttaa työskennellen samassa toimipisteessä, jossa olisi mahdollistettu myös Crystal Clearissa mainittu osmoottinen kommunikointi. Käyttämäni iteratiivinen ja inkrementaalinen menetelmä oli mielestäni sopiva ratkaisu projektin toteutusvaiheeseen. Erityisesti protoilutyyppinen ohjelmistokehitys oli järkevää, koska käytettävät teknologiat eivät olleet minulle kovin tuttuja ja protoilu mahdollisti kokeilutyyppisen kehityksen. Protoilussa oli myös omat ongelmansa, jotka havainnollistuvat projektin loppuvaiheessa. Ongelmana olivat keskeneräiset toiminnot, jotka olivat unohtuneet ohjelmistoon ja ne huomattiin vasta ohjelmiston testausvaiheessa.

Ohjelmistotuotannon teoria, eivätkä käytännöt olleet minulle erityisen tuttuja aihealueita ennen opinnäytetyön aloittamista, koska en suuntautunut opinnoissani tietojärjestelmien kehitykseen. Opin prosessin aikana huomattavan määrän ohjelmistotuotannon käytännöistä ja erityispiirteistä, joiden syvällisempi ymmärtäminen auttaa ymmärtämään paremmin organisaation IT-palveluiden kokonaisuuksia ja hallinnoimista. Opin runsaasti myös VBA-ohjelmoinnista, joka oli minulle täysin uusi alue opinnäytetyötä aloitettaessa. Aikaisempi kokemus ohjelmoinnista kuitenkin nopeutti uuden ohjelmointikielen käytäntöjen oppimista.

Ohjelmistoprojektissa toteutettu ohjelmisto oli pieni työkalu tietyn liiketoiminnan osa-alueen toiminnan tukemiseen. Tällaisien pienten ohjelmistojen lisääntyminen organisaatiossa alkaa rasittaa jossain vaiheessa ylläpitoa ja käyttäjiä. UPM tutkimuskeskuskeskuksen hankkiessa samankaltaisia ohjelmistoja Access-alustalle, kannattaa tutkia mahdollisuutta integroida järjestelmiä yhteen toiminnallisuuksiltaan ja käyttöliittymältään. Integrointi voisi vähentää ylläpidollista työtä ja hyvin tehtynä, se mahdollistaisi uusien pienohjelmistojen integroinnin pääohjelmaan, joka toimisi ainoana käyttöliittymänä, josta kaikki aliohjelmat olisivat käytettävissä.

Kuvat

Lähteet

1. IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf. Luettu 9.3.2013.

2. Haikala, I & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Talentum Media Oy.

3. Nuortio, J. 2012. Pahimmin pieleen menneet it-projektit. 3T. http://www.3t.fi/artikkeli/uutiset/teknologia/pahimmin_pieleen_menneet_it_projektit. Luettu 9.3.2013.

4. Haikala, I & Märijärvi J. 2004. Talentum Media Oy.

5. Microsoft. Regression Testing. http://msdn.microsoft.com/en-us/library/aa292167%28v=vs.71%29.aspx. Luettu 16.3.2013.

6. Princeton University. User Acceptance Testing. http://web.princeton.edu/dms/public/methodology/dev/testbase.html. Luettu 9.3.2013.

7. Gheorhiu, G. 2005. Performance vs. load vs. stress testing. http://agiletesting.blogspot.fi/2005/02/performance-vs-load-vs-stress-testing.html. Luettu 9.3.2013.

8. Rouce, W. 1970. Managing the development of large software systems. http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf. Luettu 10.3.2013.

9. Williams, L. 2004. A Survey of Plan-Driven Development Methodologies. http://agile.csc.ncsu.edu/SEMaterials/PlanDriven.pdf. Luettu 9.3.2013.

10. Agile Finland. Ketterän ohjelmistokehityksen julistus. http://agilemanifesto.org/iso/fi. Luettu 9.3.2013.

11. Cockburn, A. 2005. Pearson Education, Inc. Crystal Clear, A Human-Powered Methodology for Small Teams.

12. Crystal Family. http://assets.devx.com/articlefigs/17424.jpg. Luettu 21.3.2013.

13. UPM. Yhtiön esittely. http://www.upm.com/FI/UPM/Pages/default.aspx. Luettu 16.2.2013.

14. UPM Vuosikertomus 2012. http://www.upm.com/FI/SIJOITTAJAT/Documents/UPMVuosikertomus2012.pdf. Luettu 26.2.2013.

15. UPM:n liiketoiminnot. http://www.upm.com/FI/UPM/Liiketoiminnot/Pages/default.aspx, Luettu 26.2.2013.

16. UPM:n historia. http://www.upm.com/FI/UPM/UPM-Lyhyesti/Historia/Pages/default.aspx. Luettu 26.2.2013.

17. UPM: tutkimus- ja kehitystyö - uusia tutkimusaloja. http://www.upm.com/FI/MEDIA/Pressikansiot/Liiketoiminta/Tutkimus-ja-kehitys/Documents/UPM_RD_brochure_update_2011_fi.pdf. Luettu 21.3.2013.

18. UPM:n tutkimus- ja kehitys. http://www.upm.com/FI/UPM/Tutkimus/Pages/default.aspx. Luettu 26.2.2013.

19. UPM Intranet. UPM Reseach Center Lappeenranta. Luettu 21.3.2013.

20. Microsoft. Microsoft Office suites. http://office.microsoft.com/en-us/suites/. Luettu 17.2.2013.

21. Basics for Building Access 2007 Runtime-Based Solutions. http://msdn.microsoft.com/en-us/library/office/cc136539%28v=office.12%29.aspx. Luettu 17.2.2013.