Opinnäytetyö (AMK)

Tietojenkäsittely

Tietojärjestelmät

2014

Ville Lehtinen

TIETOJÄRJESTELMÄN RÄÄTÄLÖINTI KETTERIÄ MENETELMIÄ HYÖDYNTÄEN

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely | Tietojärjestelmät

Tammikuu 2014 | 34 sivua

Anne Jumppanen

Ville Lehtinen

TIETOJÄRJESTELMÄN RÄÄTÄLÖINTI KETTERIÄ MENETELMIÄ HYÖDYNTÄEN

Tämän opinnäytetyön tavoite on toteuttaa tietojärjestelmä terveydenhuoltoalan asiakkaalle. Järjestelmä toteutetaan räätälöimällä se saman tuotteen toisesta versiosta, joka on käytössä toisella asiakkaalla. Järjestelmän kehityksessä hyödynnetään ketteriä menetelmiä muun muassa antamalla loppukäyttäjien testata järjestelmää kehityksen aikana ja vaikuttaa sen ominaisuuksiin, ja kehittämällä järjestelmää iteratiivisesti.

Järjestelmän rakenne jakautuu pääpiirteittäin kolmeen osaan, käyttöliittymään, tietokantaan ja kontrollereihin. Käyttöliittymä koostuu näkymistä, jotka ovat pääasiassa lomakkeita ja listauksia. Näkymien tehtävänä on näyttää käyttäjälle dataa ja ottaa käyttäjän syöttämää dataa vastaan. Kontrollerit hoitavat datan käsittelyn ja tallentavat sen tietokantaan, sekä hakevat tarvittavan datan tietokannasta. Järjestelmän kehitystyökalut ovat yleisesti käytössä olevia ohjelmointikieliä. Käyttöliittymän kehitykseen käytetään HTML:ää, CSS:ää ja Javascriptiä. Kontrollerit toteutetaan puolestaan Perlillä ja tietokannan hallinta MySQL:llä. Lisäksi järjestelmän kehityksessä käytetään Template Toolkitia lisäämään joustavuutta kontrollerien ja käyttöliittymän välille.

Järjestelmä räätälöidään lähdejärjestelmästä poistamalla siitä tarpeettomia ominaisuuksia ja lisäämällä uusia ominaisuuksia. Jotkin uudet ominaisuudet otetaan kolmannesta tuotteen versiosta. Käyttöliittymän räätälöinti koostuu lomakkeiden muokkaamisesta haluttuun muotoon, jotta niille voidaan syöttää prosessissa tarvittavaa tietoa, sekä listauksien muokkaamisesta niin, että niistä nähdään tarvittavat tiedot. Tietokannan räätälöinti sisältää suurimmaksi osaksi taulujen ja kenttien lisäämistä, sekä kenttien nimien vaihtamista. Kontrollerit muokataan niin, että ne vastaavat lisättyjä ja muokattuja näkymiä sekä päivitettyä tietokantaa.

ASIASANAT:

Tietojärjestelmä, terveydenhuolto, tietojärjestelmän räätälöinti, Perl, MySQL, SaaS, ketterät menetelmät

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology

January 2014 | 34 pages

Anne Jumppanen

Ville Lehtinen

CUSTOMIZATION OF AN INFORMATION SYSTEM USING AGILE METHODS

The goal for this thesis is to implement an information system for a customer in the field of health care. The system is implemented by customizing it from another version of the product, which is in use by another customer. Agile methods are implemented in the customization process for example by letting the end-users test the system during development, thus involving them in the process, and by developing the system in iterative cycles.

The system consists mainly of three parts, the user interface, the database and the controllers. The user interface consists of views, which are mostly forms and listings. The role of the views is to show data to the user and to receive data input from the user. The controllers handle the data and store it into the database, and get the required data from the database. The tools used in the development process are programming languages widely used in software development. The user interface is implemented with HTML, CSS and Javascript. The controllers are implemented with Perl and the database with MySQL. Additionally, Template Toolkit is used to bring flexibility between the views and the controllers.

The system is customized by removing unnecessary features and by adding new features to the original system. Some of the new features are taken from a third version of the product. The customization of the user interface consists of modifying forms so that the data needed by the user in their process can be stored, and also of modifying the listings so that the necessary data can be viewed. The customization of the database consists mainly of adding tables and columns, and of renaming columns. The controllers are modified so that they match the modified views and database.

KEYWORDS:

Information system, healthcare, software tailoring, Perl, MySQL, SaaS, agile methods,

SISÄLTÖ

LIITTEET

KUVAT

KÄYTETYT LYHENTEET JA SANASTO

CSS	Tyyliohjeiden määrittelyyn tarkoitettu kieli.
Hajautustaulu	Alkioista koostuva lista, jossa jokaista alkiota vastaa jokin avain, joilla ei ole tiettyä järjestystä. Hajautustauluista pystytään helposti hakemaan alkioita avaimen avulla.
HTML	Web-ohjelmointikieli.
Javascript	Komentosarjakieli.
JQuery	Javascript-kirjasto.
Kohdejärjestelmä	Opinnäytetyön kohteena olevan projektin tuloksena syntyvä tietojärjestelmä.
Lähdejärjestelmä	Tietojärjestelmä, joka toimii pohjana uudelle järjestelmälle.
Malli	Järjestelmän osa, joka kuvaa tietokannan hallintaan liittyvät ominaisuudet.
Metodi	Itsenäinen ohjelman osa, joka suorittaa tarkan, etukäteen määritellyn tehtävän.
MySQL	Relaatiotietokantaohjelmisto.
Perl	Ohjelmointikieli.
SaaS	Software as a Service. Palvelu, joka sijaitsee palveluntarjoajan palvelimella ja jota käytetään yleensä internet-selaimen avulla.
Skype	Pikaviestintäohjelma.

1 JOHDANTO

Tietojärjestelmien räätälöinti on yksi tärkeistä tietojärjestelmän kehityksen alalajeista. Lähes kaikilla yrityksillä on käytössään tietojärjestelmiä, ja jokaisella yrityksellä on niille yksilöllisiä vaatimuksia. Valmiina pakettina myytävä ohjelmistotuote ei välttämättä pysty vastaamaan näihin tarpeisiin. Eri alojen sisällä tietojärjestelmiin kohdistuvat tarpeet voivat kuitenkin olla pääpiirteittäin samoja tai vähintäänkin samankaltaisia. Tästä syystä voi taas olla tarpeettoman työlästä ja kallista luoda jokaiselle yritykselle täysin oma järjestelmänsä. Tietojärjestelmien räätälöinti sijoittuu toimintatapana näiden kahden edellä mainitun välimaastoon.

Tietojärjestelmän räätälöinnissä otetaan pohjaksi jo olemassa oleva järjestelmä, jota muutetaan tarvittavilta osin uuden asiakkaan tarpeita vastaavaksi. Tällä tavalla vältytään turhalta työltä suuren osan järjestelmästä ollessa jo valmiina ja vain osa joudutaan tekemään alusta asti. Järjestelmä voidaan myös kehittää paremmin vaatimuksia vastaavaksi.

Tämän opinnäytetyön tavoitteena on toteuttaa terveydenhuollon alan asiakkaalle tietojärjestelmä, joka luodaan toisella asiakkaalla jo käytössä olevan tietojärjestelmän pohjalle. Räätälöinti toteutetaan kopioimalla olemassa oleva järjestelmä ja tekemällä siihen tarvittavat muutokset uutta asiakasta varten. Uusi järjestelmä toteutetaan hyödyntämällä ketteriä menetelmiä. Toinen tämän opinnäytetyön tavoitteista on tutkia ketterien menetelmien soveltuvuutta ohjelmistotuotteen räätälöintiin käyttäen tutkimusmenetelmänä tapaustutkimusta.

2 JÄRJESTELMÄN RÄÄTÄLÖINTI

Nykypäivän työympäristö on kehittynyt joustavammaksi, kun yritykset pyrkivät vastaamaan tehokkaasti ja nopeasti ympäristön vaatimuksiin. Tällaisia muutoksia voi tapahtua esimerkiksi yrityksen rakenteessa tai yrityksen toimialaan vaikuttavissa lakisäädöksissä. Sopeutuessaan tehokkaasti tämänkaltaisiin muutoksiin yritys pystyy kasvattamaan kilpailuetuaan. Yrityksen toimintatavoissa tapahtuvat muutokset vaativat myös käytössä olevilta tietojärjestelmiltä mukautumista muutokseen. Tarjolla olevien tietojärjestelmien laajasta kirjosta huolimatta yrityksen voi olla vaikeaa tai jopa mahdotonta löytää täysin valmista järjestelmää, joka vastaa kaikkiin yrityksen vaatimuksiin.

Tietojärjestelmien rakenne on ajan myötä kehittynyt entistä modulaarisemmaksi, mikä tarkoittaa sitä, että järjestelmä koostuu selkeistä osista, joista jokainen suorittaa jonkin tietyn, tarkkaan määritellyn tehtävän. Modulaarisuus mahdollistaa sen, että jotakin järjestelmän osaa kyetään muuttamaan ilman, että muihin järjestelmän osiin tarvitsee tehdä muutoksia. Vahvasti modulaarisen järjestelmän muokkaaminen ja ylläpitäminen on merkittävästi helpompaa kuin ei-modulaarisen järjestelmän. Modulaarisuudesta on paljon hyötyä tietojärjestelmän räätälöinnissä, koska tarpeettomia osia on helppo poistaa ja uusia lisätä.

Modulaarisuuden lisääntyminen on tehnyt tietojärjestelmistä joustavampia ja uudelleenkäytettävämpiä. Tämä kehitys on mahdollistanut järjestelmien paremman sopeuttamisen nykypäivän työympäristön muuttuviin vaatimuksiin. Tällainen sopeuttamismahdollisuus voidaan ottaa huomioon jo järjestelmän alkuperäisessä kehityksessä. Silloin järjestelmä suunnitellaan alusta alkaen niin, että sitä muokataan jokaiselle asiakkaalle erikseen. Tämä voi olla hyvä lähestymistapa silloin, kun järjestelmän mahdollisten käyttökohteiden ja -ympäristöjen kirjo on laaja. (Dörner 2009, 2.)

Tarvittaessa järjestelmä voidaan räätälöidä eli muokata juuri halutun kaltaiseksi. Räätälöintiä hyödynnetään, mikäli mikään tarjolla olevista valmiista järjestelmistä ei vastaa yrityksen tarpeisiin. Tällöin järjestelmä tukee mahdollisimman hyvin prosessia, jossa se on tarkoitettu käytettäväksi. Monet ohjelmistotuotteet on suunniteltu niin, että niiden päärakenne on vakio, mutta tuotteiden osia voidaan vaihdella tarpeen mukaan. Tällaiset järjestelmät on tarkoitettu räätälöitäviksi jokaiselle asiakkaalle sopivaksi.

Järjestelmän räätälöinti voidaan toteuttaa usealla eri tasolla. Jokaiselle asiakkaalle erikseen räätälöitäviksi suunnitellut järjestelmät räätälöidään usein korkealla tasolla. Korkealla tasolla räätälöinti voi tarkoittaa esimerkiksi sitä, että tuotteeseen valitaan sopivaksi katsotut osat eli moduulit. Asiakkaan prosessi, johon järjestelmää tullaan käyttämään, määrittää, mitkä osat ovat tarpeellisia. Räätälöinti voi tapahtua myös matalalla tasolla, jolloin järjestelmä muokataan yksityiskohtaisesti esimerkiksi lomakkeen kenttä kerrallaan. Matalampi taso varmistaa paremman sopivuuden asiakkaalle, mutta myös kasvattaa räätälöintiin vaadittavaa työmäärää.

Räätälöidyn tietojärjestelmän kehittämisessä erittäin tärkeää on se, että asiakkaan tarpeet ja vaatimukset järjestelmää kohtaan sisäistetään perusteellisesti. Asiakas saa parhaan mahdollisen hyödyn järjestelmästä, jos se vastaa asiakkaan toimintamallia tarkasti. Tämän toteuttaminen ei ole mahdollista ilman kattavaa ymmärrystä asiakkaan toimintamallista ja järjestelmään kohdistuvista vaatimuksista. Koska järjestelmä kehitetään tiettyä asiakasta varten, kehityksestä aiheutuu helposti huomattaviakin lisäkustannuksia asiakkaalle. Kehitysprosessin onnistuessa hyvin räätälöity järjestelmä kuitenkin maksaa itsensä takaisin kasvaneen tehokkuuden ansiosta. (Sybila 2013)

3 KETTERÄT MENETELMÄT

3.1 Määritelmä

Ketterien menetelmien määritelmä on laaja, mutta yleisesti ottaen niillä tarkoitetaan menetelmiä, jotka ovat joustavia, modulaarisia ja yleensä noudattavat lyhyitä iteraatiosyklejä. Iteraatiosyklien avulla kehitysprosessi sopeutuu helposti muutoksiin. Ketterien menetelmien kulmakiviä ovat yksinkertaisuus, nopeus ja joustavuus. Ketterien menetelmien vastakohtana voidaan pitää vesiputousmallia, jossa kehitysprosessi etenee alusta loppuun selkeää reittiä suunnittelusta toteutuksen kautta käyttöönottoon. (Abrahamsson ym. 2002, 7–9.)

Ketterien menetelmien perusperiaatteina voidaan pitää Manifesto for Agile Software Developmentissa (Beck ym. 2001) määriteltyjä arvoja:

- Yksilöt ja kanssakäyminen ennen menetelmiä ja työkaluja

Ketterissä menetelmissä arvostetaan ohjelmistokehittäjien yhteisöllisyyttä ja sopimusten ihmisyyttä jäykkien prosessien ja metodien sijaan. Tämä näkyy käytännössä esimerkiksi läheisissä tiimin sisäisissä suhteissa ja tiimihengen kohottamisen tavoittelussa.

- Toimiva ohjelmisto ennen kattavaa dokumentaatiota

Käytettäessä ketteriä menetelmiä on tärkeää, että työn tuloksena syntyvä ohjelma on koko ajan toimiva. Ohjelmistosta tuotetaan tasaisin väliajoin uusia julkaisuja ja järjestelmän koodi pyritään pitämään mahdollisimman suoraviivaisena ja yksinkertaisena.

- Asiakasyhteistyö ennen sopimusneuvotteluja

Tiivis yhteistyö kehittäjien ja asiakkaan välillä on ketterissä menetelmissä tärkeämpää kuin tiukat sopimukset. Kehitysprojekti pyrkii tuottamaan asiakkaalle liiketoiminnallista arvoa alkuvaiheesta lähtien.

- Muutokseen vastaaminen ennen suunnitelmassa pitäytymistä

Kehittäjien ja asiakkaan edustajien on tarvittaessa kyettävä näkemään muutoksen tarpeellisuus ja tekemään päätöksiä muutoksista. Lisäksi sopimusten pitää olla laadittu siten, että ne mukautuvat kehityksen aikana tehtäviin muutoksiin. (Beck ym. 2001.)

3.2 Käytetyimmät ketterät menetelmät

3.2.1 Extreme Programming (XP)

Extreme Programming -menetelmä perustuu perinteisten ohjelmistokehityksen metodien ongelmiin, joille haluttiin löytää ratkaisu. XP:hen sisältyvät toimintatavat eivät sinällään olleet uusia, mutta XP:ssä ne yhdistettiin toisiinsa uudella tavalla. Siitä syntyi uusi ohjelmistokehityksen menetelmä, joka soveltuu lähinnä pieniin ja keskikokoisiin projekteihin. (Abrahamsson ym. 2002, 18–19.)

XP -prosessi koostuu viidestä vaiheesta:

- Tutkimusvaihe, jossa määritellään järjestelmään halutut ominaisuudet käyttötapausten avulla ja järjestelmän kehittäjät tutustuvat käytössä oleviin työkaluihin.
- Suunnitteluvaiheessa päätetään toteutettavien ominaisuuksien tärkeysjärjestys ja mitkä ominaisuudet toteutetaan ensimmäiseen versioon.
- Iteraatiovaiheessa käydään läpi useita iteraatioita, joiden aikana toteutetaan uusia ominaisuuksia järjestelmään ja jokaisen iteraation lopussa uutta versiota testataan. Viimeisen iteraation jälkeen järjestelmä on valmis tuotantoon.
- Tuotantovaiheessa tehdään lisää testausta ja määritellään jatkossa toteutettavia ominaisuuksia. Lopuksi järjestelmän ensimmäinen versio siirretään asiakkaan käyttöön.

- Ylläpitovaiheessa järjestelmää ylläpidetään ja asiakkaalle tarjotaan tukea järjestelmän käyttämisessä sekä kehitetään järjestelmää edelleen uusissa iteraatioissa. Lopuksi tarvittava dokumentointi kirjoitetaan, kun järjestelmään ei enää kehitetä uusia ominaisuuksia.

Yksi XP:n soveltamisen periaatteista on se, että mikään prosessi ei sovi kaikkiin projekteihin sellaisenaan, vaan prosessista voidaan ottaa tarvittavia osia käyttöön tarpeen mukaan. XP on ketteristä menetelmistä tutkituin ja sen käytöstä on suurelta osin myönteisiä kokemuksia. (Abrahamsson ym. 2002, 19–27.)

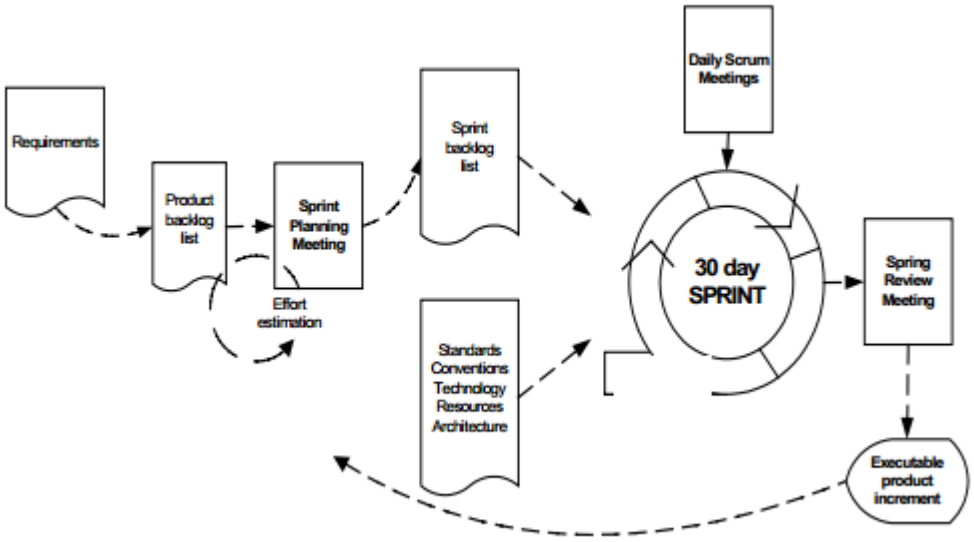
3.2.2 Scrum

Scrum pohjautuu ajatukseen, että ohjelmiston kehitysprojektissa on monia tekijöitä, kuten vaatimukset, aikataulu ja resurssit, jotka muuttuvat projektin kuluessa. Tästä johtuen kehitysprosessilta vaaditaan joustavuutta, jotta se pystyy vastaamaan näihin muutoksiin. Scrum-projekti jakautuu kolmeen päävaiheeseen:

- Valmisteluvaihe, joka jakautuu puolestaan kahteen alavaiheeseen:
    - Suunnitteluvaihe, jossa järjestelmän vaatimukset määritellään ja laitetaan tärkeysjärjestykseen
    - Arkkitehtuurivaihe, jossa tehdään korkean tason suunnittelu ja suunnitellaan järjestelmän arkkitehtuuri (Abrahamsson ym. 2002, 29.)
- Kehitysvaihe, joka on Scrumin varsinainen ketterä vaihe. Kehitysvaihe toteutetaan niin sanotuissa sprinteissä (Kuva 1), jotka kestävät viikosta kuukauteen ja joista jokainen sisältää perinteisen kehityksen vaiheet: vaatimusmäärittelyn, analyysin, suunnittelun, kehityksen ja toimituksen. Näiden sprinttien tarkoituksena on mahdollisimman hyvä järjestelmän mukautuminen muutoksiin. (Abrahamsson ym. 2002, 29-30.)

- Loppuvaihe, jossa järjestelmä valmistellaan tuotantoa varten ja luodaan muun muassa dokumentaatio ja mahdolliset integraatiot. (Abrahamsson ym. 2002, 30.)

Keskeinen osa Scrumia on niin sanottu Backlog, joka sisältää kussakin vaiheessa tiedossa olevat vaatimukset lopulliselle järjestelmälle. Backlog voi sisältää esimerkiksi toiminnallisuuksia, tiedossa olevia bugeja ja korjauksia, sekä pyydettyjä muutoksia. Backlogia päivitetään jatkuvasti projektin edetessä, jotta se on ajan tasalla projektin vaatimuksista. Toinen tärkeä ominaisuus Scrumissa ovat päivittäiset Scrum-kokoukset, joissa tiimin jäsenet kertovat, mitä he ovat edellisen kokouksen jälkeen tehneet ja mitä aikovat tehdä seuraavaksi. (Abrahamsson ym. 2002, 32-33.)



Kuvio 1. Scrumin sprint-kaavio.

Kuviossa 1 on havainnollistettu Scrumin sprinttien kulku. Ensimmäisenä ovat vaatimukset (*requirements*), jotka kohdistuvat järjestelmään. Vaatimukset kootaan tuotteen kehitysjonoon (*Product backlog list)*, joka määrittelee, mitä valmiin tuotteen tulee sisältää. Product backlog listin tehtävistä arvioidaan niihin kuluvat resurssit (*Effort estimation)*. Sprintin suunnittelupalaveri (*Sprint planning meeting)* on varsinaisen sprintin ensimmäinen vaihe. Siinä määritellään kunkin sprintin tavoitteet. Sprintissä toteutettavaksi valitut

toiminnallisuudet kootaan sprintin tehtävälistaan (*Sprint backlog list)*, joka on kuin product backlog list, mutta pienemmässä, yhden sprintin mittakaavassa.

Tästä käynnistyy sprintti, jonka kesto on yleensä noin kuukausi. Sprintin aikana pidetään päivittäisiä Scrum-kokouksia (*Daily scrum meetings),* joissa kukin tiimin jäsen kertoo, mitä on tehnyt viimeisen päivän aikana ja mitä aikoo seuraavaksi tehdä. Scrum-kokoukset toimivat myös pienimuotoisina suunnittelupalavereina. Sprintin lopuksi pidetään sprintin loppupalaveri (*Sprint review meeting)*, jossa sprintin tulokset esitellään projektin johdolle ja asiakkaille. Palaverissa päätetään seuraavat toimenpiteet ja palataan sprintin suunnittelupalaveriin, josta alkaa jälleen seuraava sprintti. (Abrahamsson ym. 2002, 32-34.)

3.3 Ketterien menetelmien edut

Yksi ketterien menetelmien selkeimpiä vahvuuksia on muutoksiin reagointi hyvin nopealla aikataululla. Erilaiset kehitysprosessin aikana ilmaantuvat muutokset ovat yleisiä ja siksi perinteiset, tiukan rakenteelliset kehitysmallit ovat joutuneet antamaan sijaa ketterille menetelmille. Yksi muutosherkimpiä tekijöitä järjestelmän kehityksessä on järjestelmään kohdistuvat vaatimukset, jotka voivat muuttua useista eri syistä.

Jos vaatimusmäärittelyä ei ole tehty kunnolla, loppukäyttäjien tarpeet järjestelmältä voivat jäädä alun perinkin epäselviksi tai virheellisiksi. Tällaisessa tapauksessa käy lähes poikkeuksetta niin, että jossakin vaiheessa joudutaan toteamaan, että järjestelmä ei vastaa alkuperäisiä odotuksia. Ketteriä menetelmiä käytettäessä tällaiset ristiriidat huomataan aikaisessa vaiheessa käyttäjien aktiivisen osallistumisen johdosta ja ne pystytään korjaamaan helpommin. (Abrahamsson ym. 2002, 9-17.)

Aina epämääräisiksi jäävät vaatimukset eivät johdu huonosti tehdystä vaatimusmäärittelystä. On myös tilanteita, joissa kehitettävän järjestelmän vaatimuksia ei kyetä selvittämään tarkasti alussa. Käyttäjillä on myös järjestelmää kohtaan *hiljaisia* odotuksia, joita he eivät osaa pukea sanoiksi ja

tuoda ilmi, tai välttämättä edes itse tiedosta. Osittain tästä syystä alkuperäisessä vaatimusmäärittelyssä ei koskaan kyetä ottamaan huomioon kaikkia vaatimuksia, joita järjestelmälle kohdistetaan. (Coplien & Björnvig 2010, 43.)

Vaikka järjestelmän vaatimukset kartoitettaisiin hyvinkin kattavasti heti projektin alussa, ne voivat silti muuttua projektin edetessä. Varsinkin nykypäivän muutosherkässä liikemaailmassa järjestelmään kohdistuvat odotukset ja vaatimukset saattavat muuttua kehitysprosessin myöhäisessä vaiheessa.

Tämä tulee esille erityisesti pitkissä projekteissa, jolloin on suurempi todennäköisyys sille, että projektin aikana esimerkiksi asiakasyrityksen liiketoimintamalliin tulee muutoksia, jotka myös muuttavat järjestelmältä vaadittavia ominaisuuksia.

Nopeasta muutoksiin reagointivalmiudesta johtuen ketterät menetelmät soveltuvat hyvin myös sellaisiin projekteihin, joissa itse kehitettävältä järjestelmältä vaaditaan jatkuvaa muutosta ja sopeutumista, kuten web-sovellukset, jotka ovat asiakkaiden käytössä ympäri vuorokauden. Tämänkaltaisten järjestelmien toimintaa ei voida etukäteen määritellä tietyn aikajanan mukaiseksi, koska niiden käyttötapaukset ovat hyvin moninaisia. Järjestelmien toiminta riippuu hyvin vahvasti käyttäjän tekemistä päätöksistä, joihin järjestelmän tulee vastata hyvin nopeasti. (Abrahamsson ym. 2002, 9-17.)

3.4 Haasteet ketterien menetelmien soveltamisessa

Ketterät menetelmät eivät ole parhaimmillaan sellaisissa ympäristöissä, joissa järjestelmän toiminta ei riipu vahvasti käyttäjästä, vaan järjestelmä toimii tietyn protokollan mukaan riippumatta tilanteesta. Tällaisia voivat olla esimerkiksi järjestelmät, jotka on sisällytetty syvälle suurempaan ohjelmistokokonaisuuteen ja joiden vuorovaikutus käyttäjän kanssa on vähäistä tai olematonta. Tämänkaltaisessa tilanteessa, jossa ympäristö rajoittaa järjestelmän toimintaa voimakkaasti ja mahdollisuudet muutokseen ovat siten pienet, ketteristä

menetelmistä ei välttämättä ole merkittävää hyötyä. (Coplien & Björnvig 2010, 8–9.)

Ketterät menetelmät varmistavat järjestelmän etenemisen oikeaan suuntaan muun muassa tuomalla loppukäyttäjät aktiiviseksi osaksi kehitysprosessia. Näin he pysyvät ajan tasalla järjestelmän tilasta ja pääsevät vaikuttamaan sen kehityksen kulkuun. Tämä näkyy muun muassa siten, että asiakkaat joutuvat usein kehitysprosessin aikana testaamaan järjestelmää ja mahdollisesti hyväksymään sen hetkisen version ennen kuin projekti voi jatkua eteenpäin. Tämä vaatii kuitenkin asiakkaalta paljon aikaa ja merkittävän sitoutumisen järjestelmän kehitykseen. (Waters 2007)

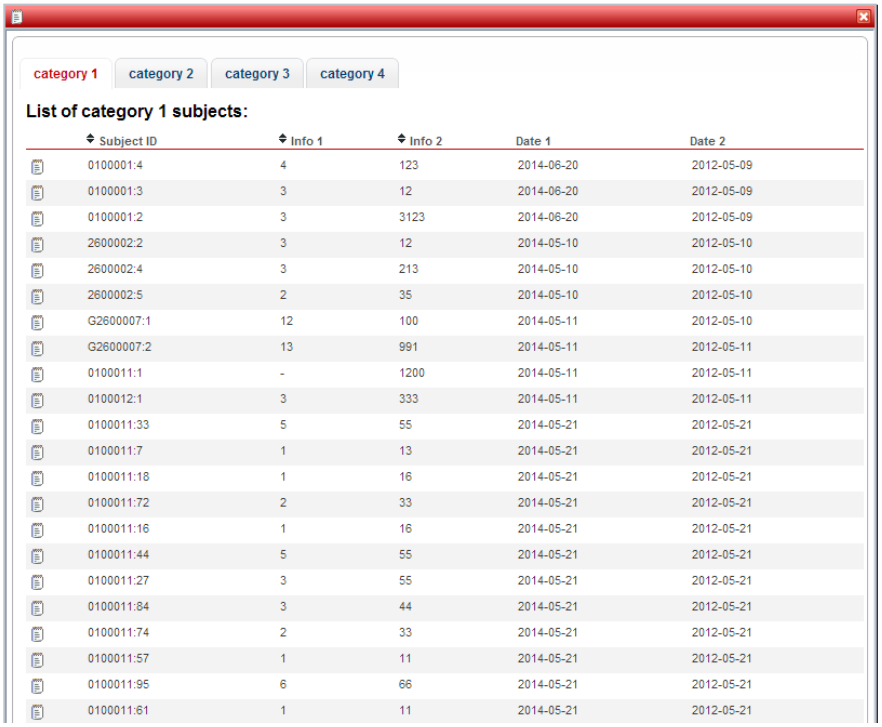
4 LÄHDEJÄRJESTELMÄ

4.1 Järjestelmän rakenne

Lähdejärjestelmä, jonka pohjalta räätälöinti toteutetaan, voidaan jakaa rakenteeltaan kolmeen osaan:

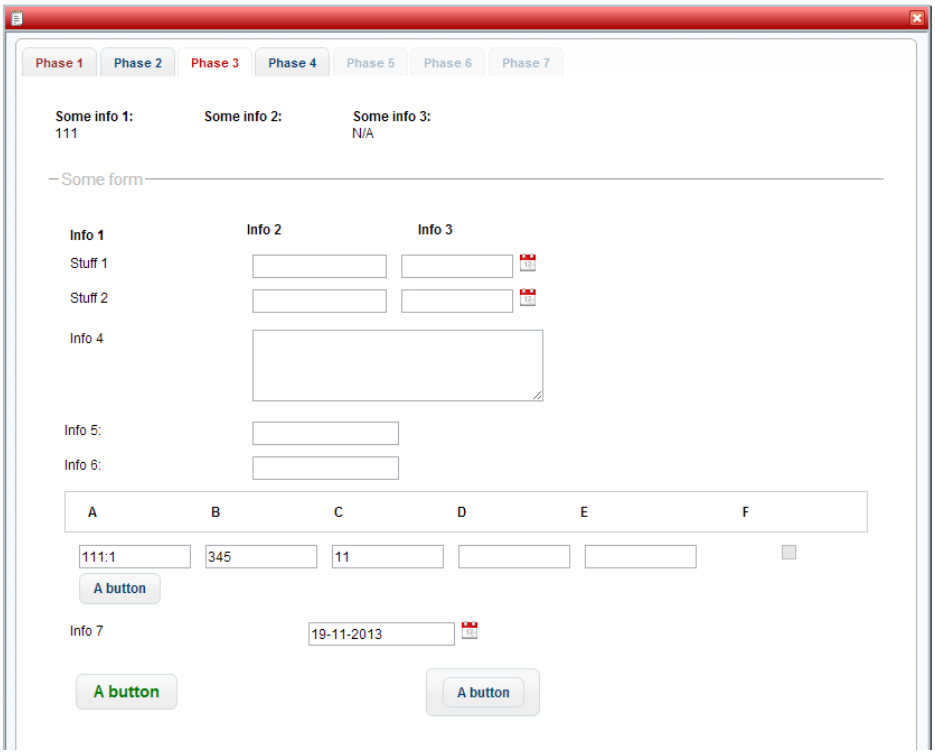
- näkymät
- kontrollerit
- tietokanta.

Järjestelmässä on pääasiassa kahdenlaisia näkymiä, eli käyttäjälle näkyviä osia: listauksia ja lomakkeita. Listauksesta näkee listattavien kohteiden perustiedot, ja siitä voi avata kunkin yksittäisen kohteen lomakkeen tarkastelua ja muokkausta varten (Kuva 1). Listauksesta voi myös avata tyhjän lomakkeen uuden kohteen luomiseksi.



Kuva 1. Esimerkki listausnäkymästä.

Lomakkeet on järjestetty loogiseen järjestykseen, joka mukailee järjestelmän käyttöprosessin etenemistä kronologisesti (Kuva 2). Jotkin lomakkeet on lukittu, kunnes jokin tietty ehto on toteutunut. Tällainen ehto voi olla vaikkapa kohteen tila, jonka tulee olla hyväksytty, ennen kuin prosessissa pääsee eteenpäin. Pääasiallisesti käyttäjä voi halutessaan muuttaa kaikkea tietoa prosessin jokaisessa vaiheessa. Näkymät on toteutettu käyttäen HTML:ää, CSS:ää, Javascriptiä, JQuerya sekä Template Toolkitia.



Kuva 2. Esimerkki lomakenäkymästä.

Kontrollereilla on kaksi tärkeää toimintaa: tietokannasta haettujen tietojen välittäminen näkymille sekä lomakkeiden kautta syötettyjen tietojen tallentaminen tietokantaan. Tietokannan jokaiselle taululle on olemassa malli, josta voidaan luoda taulua vastaava objekti. Mallit luovat automaattisesti haku- ja asetusfunktiot objektin ominaisuuksille, jotka vastaavat tietokannan kenttiä. Näitä funktioita kutsumalla kontrollerit hakevat ja tallentavat tietoja tietokantaan. Kontrollerit toimivat kahdella eri tavalla sen mukaan, millä parametreillä niitä

kutsutaan. Tietyillä parametreillä kontrolleri kutsuu joko listaus- tai lomakenäkymää välittämällä tiedot Template Toolkitille yleensä hajautustaulumuodossa. Jos kontrolleria kutsutaan lomakkeelta tallennettaessa tietoja, kontrolleri ottaa tiedot vastaan ja tallentaa ne kantaan. Kontrollerit on toteutettu Perl –ohjelmointikielellä.

4.2 Järjestelmän toiminta

Järjestelmä toimii SaaS-periaatteella eli se sijaitsee palveluntarjoajan palvelimella, johon asiakas ottaa yhteyden ja käyttää järjestelmää internet-selaimen avulla. Käyttäjä kirjautuu järjestelmään käyttäjätunnuksella ja salasanalla, ja hänelle näytetään eri ominaisuuksia järjestelmästä käyttäjätasosta riippuen. Käyttäjä voi katsoa listauksista kohteiden tärkeimpiä tietoja. Listauksissa kohteita voidaan järjestää eri ominaisuuksien perusteella. Jokaiselle kohteelle voidaan avata lomake, josta voidaan katsella ja muokata kohteen tietoja. Listauksista voidaan myös avata tyhjä lomake uuden kohteen luomiseksi. Lomakkeilla käytetään validointia varmistamaan, että esimerkiksi tiedot syötetään oikeassa muodossa ja kaikki tarpeelliset tiedot tulevat täytetyiksi.

Joitakin kohtia järjestelmän käyttöprosessissa ohjaillaan niin, että esimerkiksi tietty prosessin vaihe tulee kuitata hyväksytyksi, jotta prosessia voidaan jatkaa. Näin varmistetaan, että järjestelmässä ei voi vahingossa tehdä toimintoja, joita oikeassa prosessissa ei saa tapahtua. Järjestelmästä voi myös tulostaa raportteja, joista käyttäjä näkee tilastoja halutuista kohteista. Useissa raporteissa käyttäjä voi itse valita ajanjakson, jolta data kerätään raporttiin.

5 JÄRJESTELMÄN TOTEUTUS

5.1 Järjestelmän toteutustekniikat

HTML

HTML on kieli, jota käytetään internet-sivujen sisällön ja rakenteen määrittelyyn. HTML:n avulla voidaan julkaista muun muassa dokumentteja, jotka sisältävät tekstiä, listoja ja taulukoita. HTML:ää täydennetään usein CSS:llä ja Javascriptillä.

CSS

CSS (Cascading Style Sheet) on kieli, jota käytetään internet-sivujen ulkoasun kuten värien, asettelun ja fonttien määrittelyyn. CSS on itsenäinen HTML:stä, ja sitä voidaan käyttää minkä tahansa XML-pohjaisen ohjelmointikielen kanssa. Tämä erottelu mahdollistaa vaivattomamman ylläpidettävyyden, tyylien yhtenäistämisen sivujen välillä ja sivujen ulkoasun vaivattomamman kustomoinnin. (www.w3.org)

Javascript / JQuery

Javascript on skriptikieli, jolla lisätään selainpuolen toiminnallisuutta internet-sivuihin. Javascript on olennainen osa melkein mitä tahansa web-sivua. (Flanagan 2006.) JQuery puolestaan on avoimen lähdekoodin Javascript-kirjasto, joka tarjoaa valmiita Javascript-ratkaisuja kuten animaatioiden luonti, tapahtumien käsittely ja Ajax-applikaatioiden luonti (JQuery.com)

Perl

Perl on 1980-luvulla kehitetty yleiskäyttöinen skriptikieli, jota käytetään esimerkiksi web- ja käyttöliittymien kehittämiseen. Perlin vahvuuksia ovat helppokäyttöisyys, tuki sekä proseduraaliselle että oliopohjaiselle ohjelmoinnille sekä kattava kokoelma kolmannen osapuolen tekemiä moduuleja.

Template Toolkit

Template Toolkit on ilmainen, avoimen lähdekoodin template-ohjelma, joka on kirjoitettu Perl-ohjelmointikielellä, lukuun ottamatta joitakin avainelementtejä, jotka on kirjoitettu C-ohjelmointikielellä. Template Toolkit myös kääntää templatet Perl-kielelle ajoa varten. Template Toolkit tukee kaikkia tavanomaisia templaten käsittelyominaisuuksia kuten ehtolauseita ja silmukoita sekä monimutkaisia datatyyppejä kuten hajautustauluja ja objekteja. (Template Toolkit Home Page)

5.2 Räätälöintiprosessi

Järjestelmä, jonka pohjalta räätälöinti aloitetaan, on huomattavasti laajempi kuin uusi järjestelmä. Prosessi on kuitenkin hyvin samankaltainen. Tästä syystä toteutusratkaisuksi on valittu räätälöinti sen sijaan, että järjestelmä toteutettaisiin alusta asti uudestaan. Tässä projektissa käytetyt räätälöintimenetelmät voidaan jakaa kolmeen tyyppiin, jotka ovat lisääminen, poistaminen ja muokkaaminen.

Koska lähdejärjestelmä on laajempi kuin kohdejärjestelmä, suuri osa räätälöintiprosessista koostuu tarpeettomien osien ja moduulien poistamisesta. Poistettavien osien koko vaihtelee yksittäisestä lomakkeen kentästä kokonaisiin laajoihin toiminnallisuuksiin. Osioiden poistaminen on räätälöintimenetelmistä kaikkein suoraviivaisin ja helpoin. Suurin osa poistamisesta toteutetaan yksinkertaisesti poistamalla koodista osiot, joita ei tarvita uudessa järjestelmässä. Jotkin osat voidaan piilottaa jättämällä kyseinen osa koodiin, mutta asettamalla se kommenttiosuuden sisään. Näin toimitaan, jos kyseessä olevan osion ei katsota olevan tarpeellinen tällä hetkellä, mutta se saattaa olla tarpeellinen tulevaisuudessa (Kuva 3). Käsiteltävänä olevasta tuotteesta on useita eri versioita käytössä eri asiakkailla, ja jotkin osiot ovat yhteisiä useissa versioissa. Tällaisten osioiden tapauksessa voidaan pitää todennäköisenä, että ne tullaan lisäämään myös tässä projektissa käsiteltävään järjestelmään myöhemmissä vaiheissa.

Muokkaaminen on myös suoraviivaista, koska se koostuu kenttien nimien vaihtamisesta tai siirtämisestä lomakkeelta toiselle. Jotkin sellaiset osiot, joita lähdejärjestelmässä ei ole, on otettu kolmannesta saman tuotteen versiosta. Koska lähde- ja kohdejärjestelmät ovat englanninkielisiä ja edellä mainittu kolmas järjestelmä on suomenkielinen, näiden osioiden tapauksessa lomakkeet on käännetty suomesta englanniksi.

Vaativin ja aikaavievin käytetty räätälöintimenetelmä on osioiden ja toiminnallisuuksien lisääminen. Uudet ominaisuudet vaativat vaihtelevan määrän suunnittelua, joka lisää työmäärää merkittävästi. Lisäksi se on kolmesta menetelmästä ainoa, jossa tuotetaan uutta koodia.

```
<legend>[% Term("Lomakkeen väliotsikko") %]</legend>
        <!-- Tästä kommentoitu pois osio, joka saatetaan myöhemmin ottaa käyttöön
        <div style="float:right;">
            <button type="button" id="button1" name="button1" disabled>[% Term("Toiminto 1") %]</button><br/><br/>
            <button type="button" id="button2" name="button2" disabled>[% Term("Toiminto 2") %]</button>
        </div>
        -->
        <h5>[% Term("Kohde numero ") %][% subject_id %] </h5>

        <div id="some_div" class="properties">

            <label style="width:250px;">[% Term("Lomakkeen osion otsikko") %]:</label><br/><br/>

            <!-- Template Toolkit mahdollistaa silmukat, joiden avulla voidaan esimerkiksi käydä läpi
                 hajautustauluja, joihin kontrollerien tietokannasta hakemat kohteet on tallennettu -->
            [% FOREACH item IN subjectHash.keys.sort %]
                [% FOREACH count IN subjectHash.$item.keys.nsort %]
                    <!-- Jos tämänhetkisen kohteen ominaisuus "attribute" on 1, luodaan sen tiedoista lista-alkio -->
                    [% IF subjectHash.$item.$count.attribute != '1' %]
                        <li id="li_[% subjectHash.$item.$count.id %]">
                            <!--<input type=checkbox alt="Checkbox" title="Checkbox" align=left id="check_[% subjectHash.$item.$count.id %]"
                            name="check_[% subjectHash.$item.$count.id %]" onClick="Function();" value="[% subjectHash.$item.$count.id %]"/>-->
                            [% Term(' #') %][% subjectHash.$item.$count.id %]
                            <img src="image.gif" onclick="function('[% subjectHash.$item.$count.id %]');"
                            alt="Button" title="Button" style="height:13px;vertical-aligment:top;">
                        </li>
                    [% END %]
                [% END %]
            [% END %]
```

Kuva 3. Esimerkki koodin kommentoinnista.

5.2.1 Käyttöliittymä

Käyttöliittymän räätälöinti koostuu suurelta osin lomakkeiden kenttien tai kokonaisten lomakkeiden poistamisesta, lisäämisestä tai uudelleen nimeämisestä. Käyttöliittymän toteutuksessa lisähaasteita asettaa asiakkaan käytössä oleva internetselain, Internet Explorer 7. Siitä puuttuu tuki useille ominaisuuksille, joita uudet selaimet tukevat. Tämä tulee huomioida käyttöliittymän ominaisuuksien suunnittelussa. IE7:n rajoitukset vaikuttavat

räätälöintiprosessissa ainoastaan uusien ominaisuuksien lisäämiseen, koska olemassa olevissa ominaisuuksissa nämä seikat on jo huomioitu.

Lomakkeiden kenttiin liittyvät toimenpiteet ovat suoraviivaisimmat, koska ne vaativat yleensä hyvin vähän suunnittelua. Kentän lisäämisen yhteydessä uudelle kentälle päätetään paikka, nimi ja tyyppi, minkä jälkeen se lisätään lomakkeelle.

Kun käyttöliittymään lisätään uusia toiminnallisuuksia, ne voidaan suurimmaksi osaksi ottaa kolmannesta järjestelmästä, joskus lähes sellaisenaan. Usein ne vaativat kuitenkin joitain pieniä muokkauksia, jotta ne saadaan sopimaan uuteen järjestelmään. Esimerkiksi toiminnallisuus, jonka voi muuten ottaa suoraan kolmannesta järjestelmästä, mutta ero kohteiden tunnusten formaatissa vaatii pientä muokkausta lomakkeella käytettyyn javascriptiin.

5.2.2 Tietosisältö

Ytimeltään järjestelmän tietosisältö pysyy lähes samana. Prosessin ollessa periaatteeltaan sama lomakkeilta tallennettava datakaan ei missään järjestelmän osiossa muutu merkittävästi. Tästä syystä suuri osa tietosisällöstäkin voidaan säilyttää samana.

Tietoa käsitellään järjestelmän sisällä pääasiallisesti kahdella tavalla, hajautustauluina ja objekteina. Lähdejärjestelmässä lomakkeelta lähetettävät tiedot otetaan vastaan hajautustauluihin, joista ne tallennetaan tietokantaan. Tietojen tallennuslogiikan ollessa valmiina lomakkeelta lähetettävän datasisällön muuttuessa riittää, kun uusi data lisätään valmiisiin hajautustauluihin. Kokonaan uudet lomakkeet otetaan kolmannesta järjestelmästä, jolloin käytetään kyseisessä järjestelmässä hyödynnettävää uudempaa tallennustapaa. Tämä toteutetaan siten, että lomaketta tallennettaessa kontrolleri hakee ensin mallin get-metodeita hyväksi käyttäen tietokannan oikean rivin tiedot tietokannan taulusta luotuun objektiin. Tämän jälkeen lomakkeelta tulleet tiedot tallennetaan objektiin vanhojen tietojen päälle ja lopuksi mallin set-metodien avulla tietokantaan. Jos tietokannasta ei löydy

vastaavaa riviä, luodaan uusi objekti, johon tiedot tallennetaan (Liite 1). Uudempaa menetelmää käytettäessä tallennukseen käytettävä osio koodista ja malli voidaan usein tuoda sellaisenaan uuteen järjestelmään.

Lomakkeita ladattaessa tiedot haetaan tietokannasta ja tallennetaan hajautustauluihin (Kuva 4). Kaikki lomakkeen näyttämiseen tarvittava data kootaan lopuksi yhteen hajautustauluun, joka välitetään Template Toolkitille (Kuva 5). Koska malli luo automaattisesti get-metodit kaikille tietokannan taulusta luodun objektin ominaisuuksille, usein uusien tietojen hakemiseen lomakkeelle riittää se, että ne on päivitetty malliin ja tietokantaan.

```
my @array = ();

    # Tähän lohkoon mennään, jos kontrolleria on kutsuttu tietyillä parametreilla,
    # jotka on tallennettu muuttujiin.
    if($param1 && $param2 && $param3){

        # Param3 on String-tyyppinen muuttuja, joka sisältää tiettyjen kohteiden ID:t eroteltuina pilkuilla.
        # Param3 pilkotaan osiin ja ID:t asetetaan uuteen array-listaan.
        my @newArray = split(',', $param3);

        # Käydään läpi uusi array-lista ja jokaisen alkion perusteella
        # Haetaan dataa tietokannasta
        foreach my $tmp (@newArray) {
            my $sql="SELECT x, y FROM z";
            my $where_clause="WHERE z.id=?;";
            $sql.=$where_clause;

            # Tietokannasta haetut kohteet tallennetaan uuteen array-listaan.
            my $query= $dbh->prepare($sql);
            $query->execute($tmp);
            while(my $ref = $query->fetchrow_hashref()){
                push(@array,$ref);
            }

            $query->finish();
        }

        our %hash_table = ();

        # Tietokannasta haetut kohteet tallennetaan sisäkkäisiin hajautustauluihin.
        foreach my $subject (@array) {
            $hash_table{$subject->{'attribute1'}}{$subject->{'id'}}{'id'} = $subject->{'id'};
            $hash_table{$subject->{'attribute1'}}{$subject->{'id'}}{'other_attribute'} = $subject->{'other_attribute'};
            $hash_table{$subject->{'attribute1'}}{$subject->{'id'}}{'third_attribute'} = $subject->{'third_attribute'};
        }

        $subject_amount = @newArray;

    }
```

Kuva 4. Datan haku tietokannasta ja tallennus hajautustauluihin.

Tietokannan taulut pitää myös päivittää uutta järjestelmää vastaaviksi. Vanhasta järjestelmästä tarpeettomiksi jääviä tauluja poistetaan harvoin, jotta ne voidaan ottaa tarvittaessa helposti käyttöön. Tietokannan muokkaaminen koostuu lähinnä uusien taulujen ja sarakkeiden lisäämisestä. Kokonaan uudet toiminnallisuudet vaativat usein uuden taulun tai taulujen lisäämisen.

Lomakkeiden muokkaaminen puolestaan vaatii sen, että vastaavaan tauluun lisätään uusia lomakkeen kenttiä vastaavat sarakkeet. Jos lomaketta muokataan vaihtamalla lomakkeen kentän nimeä, uutta saraketta ei tarvitse luoda, vaan sama nimenvaihdos voidaan tehdä myös tietokantaan.

```
3
4       my %data = (
5           save_rights         => \%user_rights,
6           debug               => $debug,
7           messages            => Common::get_messages(),
8           error_message       => $error,
9           errors              => \@validation_errors,
10          Term                => sub {Common::Term($_[0])},
11          Title               => sub {Common::Title($_[0])},
12          DateFormat          => sub{ Common::DateFormat($_[0], $_[1], $_[2]) },
13          user_accounts       => \@names,
14          user_level          =>$usrlvl,
15          attribute           =>\%attribute,
16          subject             =>$object,
17          id                  => $id,
18          attribute2          =>$attribute2,
19          subjectHash         => \%subject_hash,
20          amount              => $amount
21      );
22
23      my $template = Template->new();
24
25      $template->process("views/subjects/subject.tt", \%data) || do {
26          Common::templateError($template->error())
27      } && die $template->error();
```

Kuva 5. Datan kokoaminen yhteen hajautustauluun.

5.2.3 Raportointi

Raportoinnin muuttamiseen pätevät pääosin samat säännöt kuin lomakkeisiin ja listauksiin, sillä raportit ovat myös eräänlaisia näkymiä. Lähdejärjestelmästä voidaan tulostaa suuri määrä erilaisia raportteja. Kaikkia lähdejärjestelmän raportteja ei tulla hyödyntämään, joten ne voidaan yksinkertaisesti poistaa. Jäljelle jätettävistä raporteista määritellään tarkemmin, mitä tietoja niistä halutaan. Raporttien ulkoasu on pelkistetty ja ne koostuvat lähinnä valmiiksi määritetyn datan listauksista. Tästä syystä pääosa raporttien muutostyöstä kohdistuukin tietosisältöön eikä ulkoasuun.

5.2.4 Määrittely

Järjestelmän määrittely toteutetaan asiakkaan kanssa yhteistyössä. Aluksi asiakkaan toimintamalliin ja prosessiin tutustutaan, jotta saadaan yleinen käsitys siitä, miten se eroaa lähdejärjestelmän käyttöprosessista. Kehitysprosessin aikana asiakkaan kanssa pidetään palavereita, joissa on paikalla ainakin osa tulevista järjestelmän käyttäjistä. Näissä palavereissa senhetkinen versio järjestelmästä esitellään asiakkaalle, jolloin he pääsevät vaikuttamaan siihen.

Palavereissa järjestelmää käydään läpi yksityiskohtaisesti käyttöprosessia mukaillen. Näin asiakkaat näkevät heti, jos järjestelmässä on jokin osio tai toiminto, joka ei sovi heidän prosessiinsa. Näistä epäkohdista keskustellaan siihen asti, että löydetään vaihtoehtoinen ratkaisu. Asiakkaalle voidaan myös ehdottaa ratkaisumalleja, jotka on havaittu toimiviksi muissa tuotteen versioissa.

5.2.5 Ketterien menetelmien hyödyntäminen

Ketteryys järjestelmän kehityksessä on selkeimmin nähtävissä projektin iteratiivisuudessa. (Abrahamsson ym. 2002, 7–9). Järjestelmä toteutetaan sykleissä, jotka kestävät yleensä viikon. Kehitystiimi pitää palaverin järjestelmän etenemisestä viikon välein. Palaverissa arvioidaan järjestelmän tilanne ja asetetaan tavoitteet seuraavalle syklille. Palaverissa käydään myös läpi sellaiset toiminnot, jotka ovat osoittautuneet haasteellisiksi. Tällaisia ovat yleensä epäselvyyksiä joko siinä, miten jokin ominaisuus kannattaisi toteuttaa, tai toiminnon määrittelyssä.

Viikoittaisten palaverien lisäksi kehitystiimi pitää säännöllisesti yhteyttä toisiinsa järjestelmän kehityksen kulusta. Tiimin jakautuessa kahteen eri kaupunkiin pääsääntöisenä yhteydenpitovälineenä toimii Skype (http://www.skype.com/). Skypen avulla vastaan tulevat epäselvät asiat saadaan selvitettyä nopeasti ilman, että ongelman ratkaisua pitäisi odottaa seuraavaan viikkopalaveriin asti. Satunnaisesti viikkopalaveri voidaan myös korvata Skype-palaverilla.

Asiakkaan aktiivinen osallistuminen järjestelmän määrittelyyn myös sen kehityksen aikana lisää osaltaan järjestelmän kehityksen ketteryyttä. Projektin aikana järjestelmän uusin versio siirretään useampaan kertaan testiympäristöön, jossa loppukäyttäjät pääsevät kokeilemaan järjestelmää. Tällä tavoin järjestelmää saadaan ohjattua mahdollisimman tarkasti kohti toivottua lopputulosta koko kehityskaaren ajan.

6 YHTEENVETO

6.1 Projektin yhteenveto

Projekti saatiin valmiiksi ja järjestelmä käyttöön aikataulussa. Muutenkin koko projekti eteni hyvin aikataulun mukaisesti. Ainoastaan yhden version siirto testiympäristöön myöhästyi muutamalla päivällä. Olen tyytyväinen projektin onnistumiseen ja lopputuloksena syntyneeseen järjestelmään. Haasteet, joita kohtasin projektin aikana, ratkesivat aina kohtuullisen nopeasti. Tällaisia haasteita olivat esimerkiksi jonkin yksinkertaisen tehtävän paljastuminen yllättävän hankalaksi tai järjestelmän määrittelyssä olevan asian vaikea hahmottaminen.

Hieman lisätyötä aiheutti määrittelyiden muuttuvuus. Olin teoriatasolla tutustunut asiaan, mutta oli mielenkiintoista seurata ilmiötä käytännössä. Loppukäyttäjillä oli Coplienin ja Björnvigin (2010, 43) mainitsemia *hiljaisia* odotuksia. Usein heille tulikin vasta järjestelmää läpikäydessä mieleen asioita, joita he halusivat sen sisältävän. Melkein joka palaverissa asiakkaan kanssa jotkin toiminnot päätettiin muuttaa. Suurimmaksi osaksi muutokset olivat pieniä, mutta jotkin niistä vaativat jokseenkin paljon lisätyötä. Tästä syystä mielestäni oli tärkeää, että järjestelmää käytiin läpi asiakkaan kanssa myös kehitysprosessin aikana.

Räätälöinti järjestelmän kehitysmenetelmänä osoittautui luonnollisesti helpommaksi kuin kehittäminen alusta lähtien. Kun järjestelmän runko ja suurin osa toiminnallisuuksista olivat valmiit, uuteen järjestelmään tuli rakentaa hyvin vähän uutta alusta lähtien. Prosessi, johon järjestelmä tulisi käyttöön, oli hyvin samankaltainen alkuperäisen järjestelmän prosessin kanssa. Sen vuoksi olisi ollut turhaa rakentaa uusi järjestelmä alusta lähtien.

6.2 Ketterien menetelmien soveltuvuus projektiin

Koin järjestelmän kehityksen iteratiivisuuden erittäin tärkeäksi projektin onnistumisen kannalta. Erityisesti kehittäessäni järjestelmää suurimmaksi osaksi itsenäisesti, oli hyvin tärkeää, että kävimme projektin johtajan kanssa viikoittain järjestelmän senhetkisen tilanteen läpi. Usein tuli vastaan tilanteita, joissa en ollut aivan varma, mitä määrittelyssä tarkoitettiin. Näissä tilanteissa oli hyödyllistä, että pystyimme käsittelemään asian, jolloin sain taas tarkan käsityksen siitä, mitä minun tulisi tehdä. Järjestelmän säännöllinen vertaaminen määrittelyyn varmisti, että kehitys menee koko ajan oikeaan suuntaan.

Loppukäyttäjien osallistaminen järjestelmän arviointiin ja määrittelyyn projektin aikana oli myös selkeästi eduksi. Kaikkea mahdollista ei pystytty ottamaan huomioon alkuperäisessä määrittelyssä ja jotkin määritellyt asiat muuttuivat kehityksen aikana. Joissakin tapauksissa tämä johtui väärinkäsityksestä edellisessä määrittelyssä, joskus taas loppukäyttäjät huomasivat, että johonkin asiaan jokin toinen ratkaisu voisi sittenkin olla parempi. Vielä koulutuksessakin tuli esille asioita, joita haluttiin muuttaa järjestelmässä. Jos siis järjestelmä olisi toteutettu yhden ainoan määrittelyn pohjalta, lopputulos olisi todennäköisesti ollut kovin erilainen kuin mitä asiakas loppujen lopuksi halusi tai tarvitsi.

Ketterien menetelmien etuja tässä projektissa ei voi kiistää. Kuitenkin ketterien menetelmien käyttämisestä olisi saatu varmasti samat hyödyt siinäkin tapauksessa, että järjestelmä olisi kehitetty tyhjältä pöydältä. Mainitsemani hyödyt koskevat siis tietojärjestelmän kehittämistä kokonaisuutena. Uskon, että ketterien menetelmien käytöstä ei saada tietojärjestelmää räätälöitäessä sellaisia hyötyjä, joita ei normaalissa kehitysprojektissa saataisi.

6.3 Jatkosuunnitelmat

Työn palautushetkellä järjestelmä on ollut käytössä jo vajaat kaksi kuukautta ja toiminut hyvin. Jatkokehityssuunnitelmia on tehty ja joitakin laajennusehdotuksia on jo tarjottu asiakkaalle. Ehdotukset laajennuksista

perustuvat muualla käytössä oleviin tuotteen versioihin, joista toiminnallisuudet voidaan suoraan ottaa. Uutta vastaavaa räätälöintiin perustuvaa kehitysprojektia ei ole tällä hetkellä suunnitteilla, vaikka jossain vaiheessa sellainen saattaa olla hyvinkin mahdollinen. Tämä projekti kuitenkin osoitti, että räätälöinti on paras vaihtoehto järjestelmän kehitykselle, jos kehitettävästä järjestelmästä halutaan hyvin samankaltainen jonkin valmiin järjestelmän kanssa. Pääasiallinen kriteeri on prosessin, johon järjestelmää tullaan käyttämään, samankaltaisuus. Jos asiakkaat tilaavat tulevaisuudessa järjestelmän, joka pystytään järkevästi räätälöimään aikaisemmasta järjestelmästä, kannattaa räätälöinti valita toteutusmenetelmäksi.

LÄHTEET

Abrahamsson P., Salo O., Ronkainen J. & Warsta J. 2002. Agile Software Development Methods. Review and Analysis. Espoo: VTT Publications Viitattu 12.2.2014

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. Manifesto for Agile Software Development. Viitattu 12.2.2014 http://agilemanifesto.org/

Cohn M. 2010. Succeeding With Agile Software Development Using Scrum. Boston: Addison-Wesley. Viitattu 12.2.2014

Coplien J. & Björnvig G. 2010. Lean Architecture for Agile Software Development. UK: John Wiley & Sons Ltd. Viitattu 12.2.2014

Dörner C. 2009. Tailoring Software Infastructures. Integration of End-User Development and Service-Oriented Architectures. Köln: Josef Eul Verlag GmbH . Viitattu 12.2.2014

Flanagan D. 2006. Javascript: The Definitive Guide. O'Reilly Media, Inc. Viitattu 12.2.2014

Agile Manifesto 2001. Manifesto for Agile Software Development. Viitattu 12.2.2014 http://agilemanifesto.org/

JQuery 2014. What is JQuery? Viitattu 12.2.2014 www.jquery.com

Sybila C. 2013. How Tailored Software Development can do good to your company. Viitattu 12.2.2014 https://exploreb2b.com/articles/how-tailored-software-development-can-do-good-to-your-company

Template Toolkit Home Page 2013. About the Template Toolkit. Viitattu 12.2.2014 http://www.template-toolkit.org/

Waters K. 2007 Disadvantages of Agile Development http://www.allaboutagile.com/disadvantages-of-agile-development/

W3C 2014. What is CSS? Viitattu 12.2.2014 http://www.w3.org/Style/CSS/

Tietojen tallennus tietokantaan

```
1
2 #Kontrolleria kutsutaan tietyllä parametrilla,
3 #jolloin tietyltä lomakkeelta tuleva data tallennetaan tietokantaan
4 elsif(param("this_param")){
5
6   ########################
7   # Aloitetaan tallennus #
8   ########################
9
10  my $phase="this";
11
12  my $object;
13
14  # Haetaan objekti kutsumalla hakufunktiota
15  my @objects = @{Function->get_object(Model,{
16      where => {'this' => $that
17                }
18  })};
19
20  Objekti tallennetaan muuttujaan tai jos objektia ei saatu, luodaan uusi
21  $object = scalar @objects == 1 ? $objects[0] : Model->new();
22
23  #Käydään objektin ominaisuudet läpi
24  foreach my $property (@{$object->get_properties()}){
25
26     #Tarkistetaan, vastaako ominaisuus tiettyä ehtoa
27     if($property =~ m/match/){
28
29        #Toinen tarkistus
30        if($property !~ m/other_match/){
31
32        # Suoritetaan objektin ominaisuudelle operaatio x lähettämällä se
33        # parametrina funktiolle y ja tallettamalla se muuttujaan
34        my $variable=Function(param($phase."_".$property));
35
36        # Jos objektin ominaisuus ei vastaa lomakkeelta tullutta uutta arvoa,
37        # objekti pitää tallentaa
38        $needs_save = $object->$property() ne $variable ? 1 : $needs_save;
39
40        # Asetetaan uusi arvo objektiin
41        $object->$property($variable);
42
43     }
44
45     #Jos ei, tarkistetaan toista ehtoa vasten
46     }elsif($property =~ m/match$/){
47
48      # Suoritetaan objektin ominaisuudelle operaatio x lähettämällä se
49      # parametrina funktiolle y ja tallettamalla se muuttujaan
50      my $varibale=Function(param($some_variable));
51
52      # Jos objektin ominaisuus ei vastaa lomakkeelta tullutta, uutta arvoa,
53      # objekti pitää tallentaa
54      $needs_save = $object->$property() ne $datetime_value ? 1 : $needs_save;
55
56      # Asetetaan uusi arvo objektiin
57      $object->$property($variable);
58
59      #Jos ominaisuus ei vastaa kumpaakaan ehtoa, mennään tähän lohkoon
```

```
60      }else{
61        $variable_x = $phase."_".$property;
62
63        # Jos objektin ominaisuus ei vastaa lomakkeelta tullutta, uutta arvoa,
64        # objekti pitää tallentaa
65    $needs_save = $object->$property() ne param($variable_x) ? 1 : $needs_save;
66
67        # Asetetaan uusi arvo objektiin
68        $object->$property(param($variable_x));
69      }
70
71  }
72
73  # Tallennetaan objekti
74  if ($needs_save){
75     my ($var1, $var2, $var3) = $object->save();
76  }
77
78}# End This save
```