LAUREA
AMMATTIKORKEAKOULU
Yhdessä enemmän

Huollon mobiilisovelluksen kehittäminen

Peiponen, Aleksi

2015 Kerava

Laurea-ammattikorkeakoulu
Laurea Kerava

Huollon mobiilisovelluksen kehittäminen

Aleksi Peiponen
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Marraskuu, 2015

Laurea-ammattikorkeakoulu **Tiivistelmä**
Kerava
Tietojenkäsittelyn koulutusohjelma

Peiponen, Aleksi

Huollon mobiilisovelluksen kehittäminen

Vuosi 2015 Sivumäärä 32

Tässä opinnäytetyössä tutkittiin ohjelmistokehitystä teorian avulla ja kohdeyritykselle tehdyn huollon mobiilisovelluksen valmistusprosessia tutkien. Tarkoituksena oli tulosten pohjalta löytää hyviä ohjelmistokehityksen käytänteitä ja miettiä, miten kohdeyrityksen ohjelmistokehitystä voidaan kehittää.

Yrityksen ohjelmistokehitystä tarkasteltiin teorian ja käytännön toteutuksen avulla. Teorian avulla tarkasteltiin erilaisia ohjelmistokehityksen malleja ja käytänteitä sekä selvitettiin, miten ohjelmistokehitystä olisi järkevintä toteuttaa kohdeyrityksessä.

Opinnäytetyössä tutkittiin myös kohdeyritykselle tehdyn ohjelmiston valmistusprosessia ja mietittiin miten ohjelmiston kehitys olisi voitu tehdä paremmin ja mitä ongelmakohtia olisi syytä korjata. Kerättyjen tietojen perusteella oli tarkoitus luoda kattava ymmärrys kohdeyrityksen ohjelmistokehityksestä ja sen ongelmista.

Tuloksien avulla luotiin kehitysehdotuksia yrityksen ohjelmistokehityksen ongelmien ratkaisemiseksi. Tutkimuksen avulla oli myös tarkoitus oppia hyviä ohjelmistokehityksen käytänteitä tulevaisuuden ohjelmistokehitysprojekteja varten.

Asiasanat: mobiilisovellus, ohjelmistokehitys, ohjelmistokehitysprosessimalli

Laurea University of Applied Sciences
Laurea Kerava
Business Information Technology

Abstract

Peiponen, Aleksi

Development of Service application

Year 2015 Pages 32

This thesis studied software development using theory and inspecting the creation of target organization's Service application. The purpose of this study was to find meaningful improvements for target organization's software development practices.

Theory of software development was researched in order to get better understanding of software development methods and to find different types of software development models and practices. The purpose was to also find meaningful ways to improve target organization's software development practices.

In the second half, the purpose was to see how software development worked in practice in the organization. This way it was possible to find actual improvable points in the development process. Combining theory and observations from the project, it was possible to get an understanding of the organization's software development practices and their problems.

The goal of this thesis was to find actual problems and offer suggestions for improvements to the target organization. Using these findings and reading theories, it was possible to improve the general understanding of software development.

Keywords: mobile application, software development, software development process model

Sisällys

1 Johdanto

Tutkimuksessa tutkittiin kohdeyritykselle rakennetun huoltotöiden kirjaus- ja hallintajärjestelmän valmistusprosessia sekä mietittiin yritykselle kehitysehdotuksia yrityksen ohjelmistokehityksen parantamiseksi. Tutkimuksessa keskityttiin tarkastelemaan tehdyn huollon mobiilijärjestelmän valmistusprosessia pääosin havainnoinnin ja teorian avulla. Ohjelmiston kehitystä tutkimalla, oli tarkoitus löytää korjattavia kohtia ohjelmistokehityksestä sekä yrityksen omasta toiminnasta projektin vetäjänä.

Tutkittava huollon mobiilijärjestelmä on huoltotöiden kirjausjärjestelmä. Järjestelmän perusajatuksena on tehostaa töiden kirjaamista ja seurantaa. Järjestelmän avulla yritys kykenee standardisoimaan ja suoraviivaistamaan töiden kirjausta ja laskutusta. Järjestelmän mukana haluttiin myös rakentaa hallinnointipuoli, jonka avulla töitä voidaan helpommin ja tehokkaammin jakaa huoltomiehille. Järjestelmä rakennettiin yhtiön Lemonsoft-ohjelmaan linkitetyksi yhteistä tietokantaa ja Web Services -rajapintaa hyödyntäen.

Yritys itsessään ei ole ohjelmistoja tuottava yritys, vaan ohjelmistoja yleensä ostetaan muualta. Jos osaamista riittää, pyrkii yritys hyödyntämään oman osaamisensa. Vuonna 2013 ja 2014 yritys onkin hyödyntänyt työharjoittelijoiden osaamista kehittäen tarvittavia ohjelmistoja itse. Tutkimuksessa haluttiin selvittää, mitä erilaisia ongelmakohtia ohjelmistokehityksestä löytyy ja miettiä, kuinka kyseisiä ongelmia voidaan kehittää. Ongelmakohtien korjaamisen kautta yrityksen omat ohjelmistokehitysprojektit voidaan tulevaisuudessa toteuttaa tehokkaammin.

2 Tutkimuksen taustat ja tavoitteet

2014 keväällä toimin työharjoittelijana kohdeorganisaatiolla ja päädyin ohjelmoimaan huollon mobiilisovellusta. Mobiilisovelluksen rakentaminen osoittautui isoksi projektiksi, joka kesti syksyyn asti. Projektin loppuvaiheilla päätin tehdä opinnäytetyöni kyseisen ohjelmiston kehityksestä. Tutkimus koostuu teoreettisen tiedon tutkimisesta ja ohjelmiston toteutuksen empiirisestä tarkastelusta.

2.1 Tutkimuskysymykset

Tämä tutkimus keskittyy kohdeyritykselle rakennetun ohjelmiston rakennusprosessiin ja ohjelmistokehityksen tutkimiseen. Tutkimuskysymykset on luotu keräämään tietoa projektin kulusta ja sen erilaisista ongelmista. Tietoa on kerätty empiirisesti mobiilisovelluksen kehitystä tutkien sekä teoriaa hyödyntäen.

K1: Minkälainen ohjelmistokehitysprosessimalli sopii parhaiten kohdeyritykselle?
K2: Miten voidaan ohjelmistokehitysprojektin avulla löytää kohdeorganisaatiolle sopivammat tavat kehittää ohjelmistoja?
K3: Mitä hyötyä kohdeorganisaatiolle olisi ohjelmistokehitysmalleista?

Empiirisen osuuden avulla pyritään löytämään vastauksia ensimmäiseen kysymykseen havainnoimalla erilaiset ongelmat projektin ajalta. K1 avulla on tarkoitus löytää teoriaa hyödyntäen kehitettäviä kohtia yrityksen ohjelmistokehitystä varten. Kysymysten avulla saadaan myös hyvä ymmärrys erilaisista ohjelmistokehityksen käytänteistä ja malleista.

K2 ja K3 avulla on tavoitteena löytää ongelmallisia kohtia yrityksen omassa toimimnnassa ja tehdyssä projektissa. Kysymyksiin vastaamalla saadaan kerättyä tietoa ohjelmiston rakentamisessa ilmenneistä ongelmista. Havaittujen ongelmien ja teoriasta löydettyjen mallien avulla voidaan luoda kehitysehdotuksia ja vaihtoehtoja, joita hyödyntämällä yritys kykenee kehittämään toimintaansa.

Tämän tutkimuksen tarkoituksena on saada kattavat vastaukset edellisiin kysymyksiin. Vastauksien avulla kyetään löytämään ratkaisuja sekä antamaan kehitysehdotuksia kohdeyrityksen ohjelmistokehityksen parantamiseksi. Samalla tavoitteena on myös kehittää omia ohjelmistokehitystaitoja tulevaisuuden työtehtäviä varten.

2.2 Keskeiset käsitteet ja lyhenteet

HTML = HyperText Markup Language (HTML) on merkintäkieli, jonka tarkoituksena on kuvata verkkosivun rakenne sekä antaa vihjeitä esitystavasta (w3schools 2014).

CSS = Cascading Style Sheets (CSS) on tyylinmuokkaus kieli, jonka avulla kyetään kuvaamaan ulkonäköä ja muotoilua. CSS:ää käytetään useimmiten HTML:n ja XHTML:n kanssa. (w3schools 2014.)

Javascript = Javascript on yleisesti web-ympäristössä käytetty scriptikieli, jolla voidaan lisätä toiminnallisuutta verkkosivulle (w3schools 2014).

HTML5 Local Storage = HTML5 Local Storage on paikallinen tallennuspaikka selaimessa. Local storage mahdollistaa datan tallentamisen paikallisesti selaimeen. Tämä mahdollistaa laajankin datan tallentamisen paikallisesti vaikuttamatta sivuston toimintaan negatiivisesti. (w3school 2015.)

PHP = PHP ilmainen palvelinpuolen skriptikieli ja Microsoft ASP:n kilpailija. PHP on tehokas työkalu dynaamisten ja interaktiivisten sivustojen teossa. (w3school 2015.)

SQL = Structured Query Language (SQL) on standardi tietokantakieli, jolla päästään käsiksi ja kyetään muokkaamaan relaatiotietokantoja (w3schools 2014).

JQuery = JQuery on ”lightweight” javascript-kirjasto, jonka avulla ohjelmoija voi kirjoittaa vähemmän ja tehdä enemmän. Kirjasto mahdollistaa kehittyneempien toimintojen käytön verkkosivustossa. (w3schools 2014.)

JQM, JQuery Mobile = Jquery Mobile on javascript- ja CSS-kirjasto, joka on rakennettu JQuery kirjaston päälle tuomaan myös visuaalisia ominaisuuksia verkkosivustoon. (Jquery Mobile 2015.)

Lemonsoft = Toiminnanohjausjärjestelmä jolla yrityksen kokonaistoimintaa voidaan hallita helposti yhden järjestelmän avulla (Lemonsoft 2015).

Web Service = Palvelurajapinta, jolla ulkopuoliset ohjelmat voivat kommunnikoida ohjelmiston kanssa (Lemonsoft 2015).

JSON = Javascript Object Notation on kevyt datan välitysformaatti, joka on helposti ymmärrettävissä. JSON on kielestä riippumaton eli JSON-formaatti on tekstimuodossa ja voidaan lukea millä tahansa ohjelmointikielellä. (w3school 2015.)

AJAX = AJAX eli Asynchronous Javascript and XML. AJAX on uusi tapa käyttää vanhoja standardeja. AJAX:n avulla voidaan helposti välittää dataa palvelimelle ja palvelimelta ilman erillisiä sivuston uudelleen latausta. (w3school 2015.)

Google API = Google Application Programming Interface (API) on Googlen ohjelmiston ohjelmointirajapinta, jonka avulla voidaan hyödyntää Googlen tekemiä teknologioita, kuten Google Mapsiä (w3school 2015).

MVC-arkkitehtuuri = Model-View-Controller (MVC) eli Malli-Näkymä-Käsittelijä on ohjelmistokehityksessä käytettävä arkkitehtuurillinen tyyli, jossa ohjelma on jaettu kolmeen eri osaan: malliin, näkymään ja käsittelijään. (TTY ohjelmistotekniikka 2008.)

Versionhallinta = Versionhallinnalla tarkoitetaan ohjelmiston kehityksen eri vaiheiden hallintaa. Versionhallinta mahdollistaa eri versioihin tehtyjen muutosten tarkastelun, vertailun ja tarpeen tullen eri vaiheisiin palaamisen. (Tietojenkäsittelytieteen laitos 2015.)

Git = Git on Linus Torvaldsin kehittämä versionhallintaohjelmisto (Git SCM 2015).

Github = GitHub on web-pohjainen git-kansioiden hostaussivusto ja -järjestelmä, joka mahdollistaa helpon git-kansioiden jakamisen ja hallinnoinnin. (Github 2015.)

3 Ohjelmistokehitys teorian avulla

Ohjelmistokehitystä on tehty jo vuosikymmeniä. Useiden erilaisten kokemusten pohjalta on syntynyt erilaisia käytänteitä ja teorioita hyvästä ohjelmistokehityksestä. Kun lähdetään tutkimaan ohjelmistokehityksen hyviä käytänteitä, on hyvä kerätä tietoa edeltävien osaajien ohjelmistokehityskokemuksista.

3.1 Ohjelmiston suunnittelu ja asiakkaan tarpeet

Ohjelmiston suunnittelu on yksi keskeisimpiä askelia ohjelmistokehityksessä. Suunniteltaessa ohjelmistoa on hyvä miettiä ohjelmiston toimintaa monista eri näkökulmista. Ohjelmisto määritellään yhdessä asiakkaan kanssa, koska asiakas on lopulta se henkilö, joka ohjelmistoa tarvitsee. Suunnitteluvaiheessa pyritään miettimään mahdollisimman selkeä rakenne ja kitkemään pois kaikki isoimmat virheet jo heti kättelyssä. Tällöin kyetään välttämään ongelmien korjaamiselta myöhemmässä vaiheessa, jolloin korjausten tekeminen on selkeästi raskaampaa, kuin alussa. Hyvä ohjelmiston suunnittelu takaa vahvan perustan ohjelmistolle ja helpottaa sovelluksen ohjelmointia.

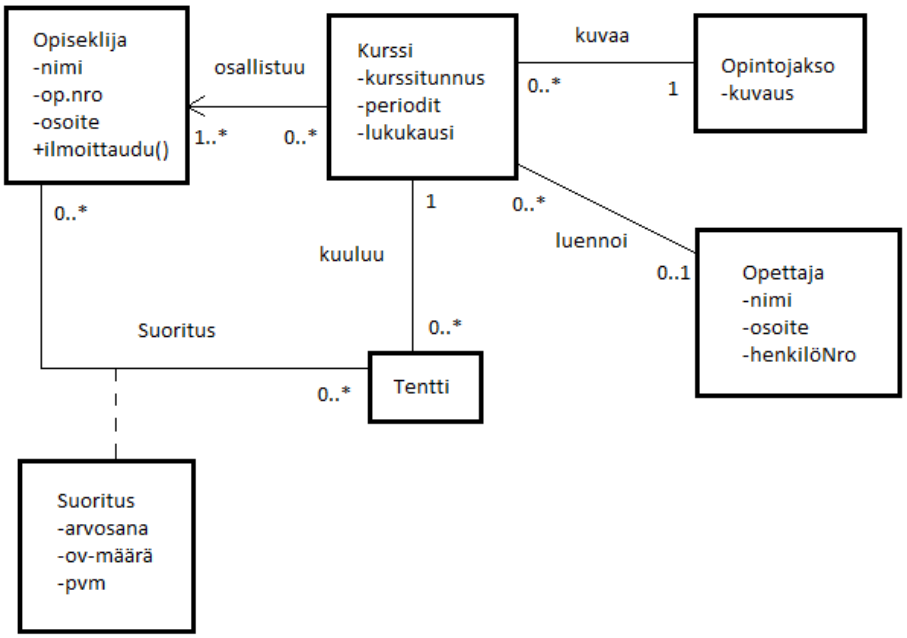
Ohjelmistokehityksessä lähdetään monesti ongelmallisesti rakentamaan ohjelmistoa toiminnallisuuden puolelta. Tämä on ongelmallista, koska ohjelmistoja rakennetaan ratkaisemaan ihmisten ongelmia. Tällöin käyttäjä on henkilö, jolle ohjelmisto pitää suunnitella. Ohjelmistoa suunniteltaessa tärkeäksi kysymykseksi nousevat asiakkaan tarpeet, koska loppukäyttäjänä toimivat kuitenkin asiakasyrityksen työntekijät tai yrityksen omat asiakkaat. (Klimczak 2013, 8.)

Kun tiedetään kohderyhmä, jolle ohjelmisto on tarkoitus kehittää, olisi hyvä tehdä käyttäjätutkimusta keräämään tietoa ohjelmiston halutusta toiminnallisuudesta. Tutkimus voidaan tehdä valmiista ohjelmasta tai yrityksen nykyisestä toimintatavasta. Tutkimusta voidaan tehdä havainnoimalla toimintaa ja haastattelemalla käyttäjiä. Näin päästään käsiksi käyttäjäkohtaiseen tietoon, jonka avulla on helpompi löytää käytettävyysongelmia jo suunnitteluvaiheessa. Tiedoista voidaan rakentaa persoonia käyttäjistä. Persoonien avulla nähdään, minkälaiset tapaukset toistuvat tietyillä ihmisillä. Kerätyn datan avulla voidaan muodostaa skenaarioita, joiden avulla voidaan luoda lista halutuista toiminnallisuuksista. Käyttäjätutkimuksen avulla nähdään ongelmalliset kohdat käyttäjien toiminnassa ja saadaan arvokasta tietoa ohjelmiston kehittämiseen. (Klimczak 2013, 10-20.)

3.2 Luokkakaaviot

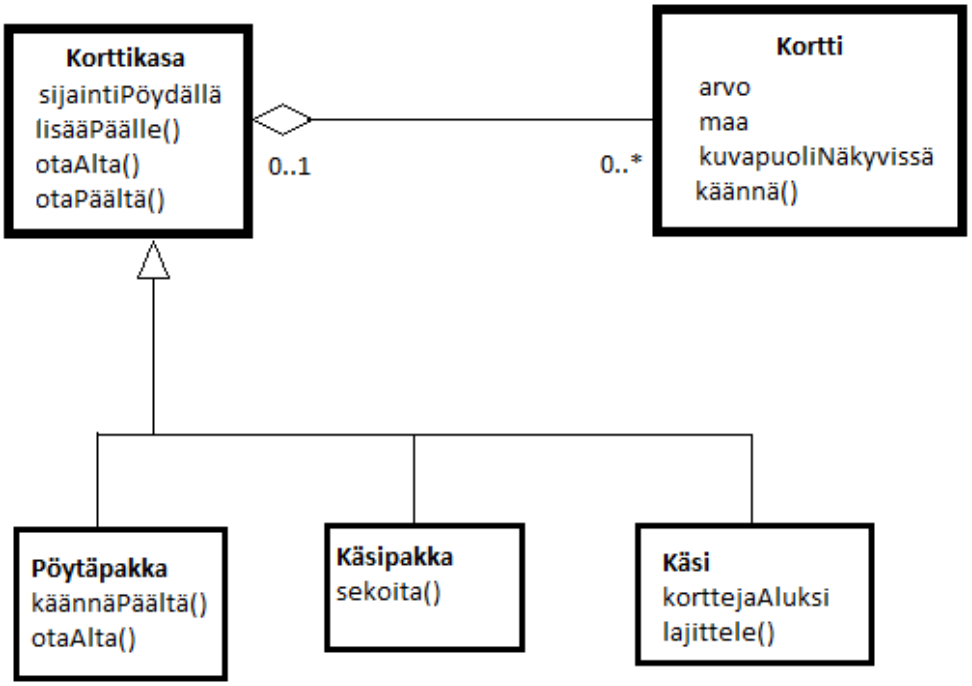
Luokkakaavioilla on tarkoitus kuvata oliokeskeisen ohjelmiston toimintoja ja toimia keskeisenä mallinnusvälineenä. Luokkakaaviot sisältävät luokkien attribuutteja, metodeja eli operaatioita sekä luokkien välisiä yhteyksiä. Attribuutit ovat luokassa olevaa tietoa. Esimerkiksi opiskelija-luokan attribuutteina olisivat nimi, opiskelijanumero, osoite, jne. Metodeilla on taas tarkoitus kuvata toimintoja, joita kyseinen luokka kykenee suorittamaan. Samalla opiskelijalla voisi olla metodeina esimeriksi ’ilmoittaudu()’ ja ’muuta_osoite()’s. Miettimällä suunnitteluvaiheessa ohjelmiston attribuutteja ja metodeja on helppoa havainnolistaa ja nähdä mitä kaikki tietoja ja toimintoja on pakko ohjelmistosta löytyä. (Haikala & Märijärvi 2004, 118-119.)

Luokilla on aina myös assosiaatioita toisten luokkien kanssa. Esimerkiksi opiskelija voi osallistua kurssille. Tällöin opiskelija- ja kurssi-luokka ovat suoraan tekemisissä toistensa kanssa. Assosiaatioiden suunnat ovat oletusarvoisesti kaksisuuntaisia, mutta suunta riippuu luokkien suhteesta. Assosiaatioillakin on mahdollista olla attribuutteja, mutta tämä tapahtuu assosiaatio luokan avustuksella. Assosiaatioviivojen päissä näkyvät lukumääräsuhteet luokkien välillä. Nämä ”N..M” yhteydet kertovat, kuinka monta muuta oliota voi tiettyyn olioon liittyä vähintään (N) ja enintään (M). Näin kaavion 1 esimerkissä näkyvä osallistuu assosiaation 0..* tarkoittaa, että opiskelijalla voi olla 0 tai enemmän kursseja ja 1..* taas meinaa, että kurssilla täytyy olla yksi tai enemmän oppilaita. (Haikala & Märijärvi 2004, 119-122.)



Kaavio 1: Luokkakaavio esimerkki (Haikala & Märijärvi 2004, 119.)

Luokkakaavioilla merkitään myös luokkien periytyvyys. Tämä ylä- ja alaluokan suhde kuvataan kaavioon esimerkiksi avoimella nuolella. Tälläisissä tilanteissa tulee selväksi mitkä luokat perivät tiedot toiselta luokalta. Esimerkiksi Kauppa-luokan tiedot voidaan periä kaikille kaupan aliluokille, kuten elektroniikkakauppa. Näin kaupan metodit ja attribuutit tulevat mukaan aliluokkaan, jonka jälkeen voidaan lisätä elektroniikkakaupalle omat attribuutit ja metodit. Koostuminen eli aggregatio voidaan myös ilmaistaan luokkakaaviossa (kaavio 2). Koostumisella tarkoitetaan luokan koostumista monista toisen luokan ilmentymistä. Hyvänä esimerkkinä toimii korttipakka, joka sisältää jokaisesta kortista tehdyn olion. (Haikala & Märijärvi 2004, 122-125.)



Kaavio 2: Periytyminen ja koostuminen (Haikala & Märijärvi 2004, 124.)

Luokkakaavion rakentamisella saadaan suunnitelma ohjelmiston kokonaisuudesta ja toiminnallisuudesta, ennen kuin päästään edes aloittamaan ohjelmiston ohjelmointia. Monet tekniset ongelmat ohjelmiston toiminnan suhteen tulevat myös esiin luokkakaavioita tehdessä ja niihin päästään pureutumaan ennen toteutuksen aloittamista. Itse ohjelmointikin helpottuu merkittävästi, kun tiedetään jo ennalta, mitä erilaisia rakennuspalikoita ohjelmistosta täytyy löytyä. Jos suunnitelma on tehty erittäin hyvin, on ohjelmointikin todella helppoa.

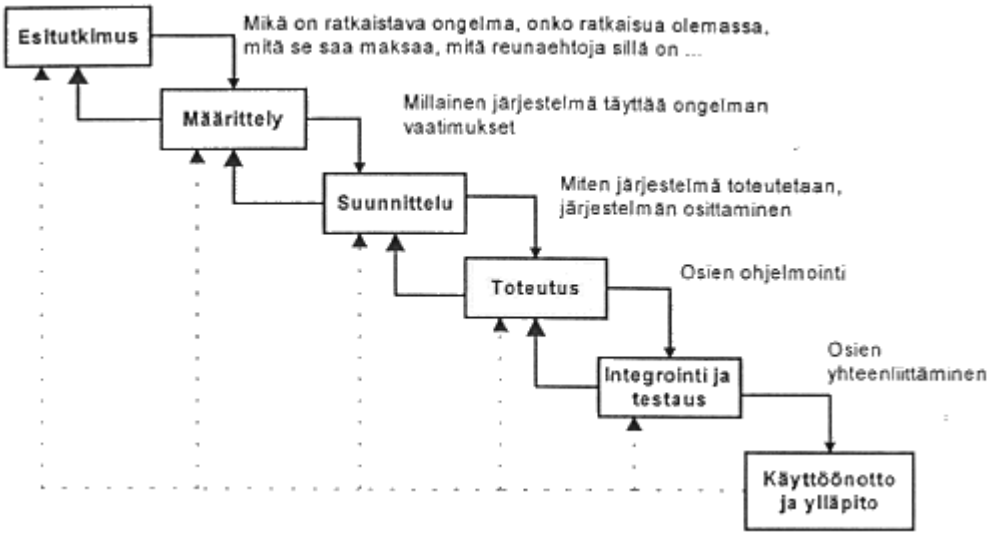
3.3 Ohjelmistokehityksen mallit ja menetelmät

Ohjelmistokehityksen historian aikana on muodostunut erilaisia hyviä malleja ja menetelmiä, joiden mukaan ohjelmistokehitysprojektit yleensä etenevät. Projektin alkuvaiheissa on hyvä tietää, minkä mallin mukaisesti ohjelmistokehitysprojekti halutaan toteuttaa ja miettiä, minkälaista ohjelmistoarkkitehtuuria halutaan käyttää.

3.3.1 Vesiputousmalli

Yksi tavallisimpia vaihejakomalleja on vesiputousmalli (kaavio 3). Vesiputousmallista on erilaisia versioita, mutta yleensä sisältävät ainakin määrittelyn, suunnittelun ja erilaiset toteu-

tusvaiheen osat. Vesiputousmallissa prosessit kulkevat järjestyksessä alusta loppuun, portaalta toiselle vesiputouksen lailla. Tarkoituksena on, että aiemman portaan valmistamat tiedot siirtyvät seuraavalle portaalle käsiteltäväksi. Vaiheiden lopussa on yleensä tarkastuksia ja testausta, joilla pyritään kitkemään pois kaikki virheet. (Haikala & Märijärvi 2004, 36-37.)



Kaavio 3: Vesiputousmalli (Haikala & Märijärvi 2004, 36.)

Tällaisessa lineaarisessa mallissa on erittäin vaikeata suunnitella ohjelmiston toiminnallisuus alusta loppuun valmiiksi projektin alkuvaiheessa. Yleensä ohjelmistokehitysprosessin aikana ilmenee ongelmia ja muutostarpeita. Muutostarpeet voivat tulla monista eri lähteistä niin teknisistä ongelmista kuin myös asiakkaan tai tekijöiden puolelta. Vesiputousmalli ei helposti mukaudu muutoksiin. Väärille urille ajautunut projekti on vaikea saattaa takaisin oikeille raiteille. (Helsingin yliopisto 2009.)

3.3.2 Ketterät menetelmät

Vuonna 1968 NATO Software Engineering konferenssissa käytettiin termiä ”software crisis” eli ohjelmistokriisiä. Tällä termillä kuvailtiin yritysten, armeijan ja hallituksen ongelmia ohjelmistojen suhteen. Ohjelmistoja oli tehty sekalaisin menetelmin, koska teknologiat olivat uusia ja osaaminen sekä menetelmät olivat jatkuvassa muutoksessa. Ohjelmistokehitystä haluttiin yhtenäistää standardisoidummaksi prosessiksi insinöörityön tapaan. Ongelmana oli kuitenkin hyvien periaatteiden puute uudella alallla. Kukaan ei yksinkertaisesti tiennyt mitä pitäisi tehdä ja miten. Hiljalleen alettiin kuitenkin kehittää erilaisia metodeja ja standardeja ohjelmistojen suunnittelemiseen ja rakentamiseen. Myöhemmin 1990-luvulla monet henkilöt ohjelmistokehitysyhteisöstä päätyivät hyvin samankaltaisiin ohjelmistokehityksen malleihin, joita kutsuttiin kevyiksi metodologioiksi. Lopulta monet kevyen metologian päätukijat liittyivät yhteen ja kehittivät ”Agile manifest” -nimellä kulkevan manifestin. Tässä manifestissa kuvat-

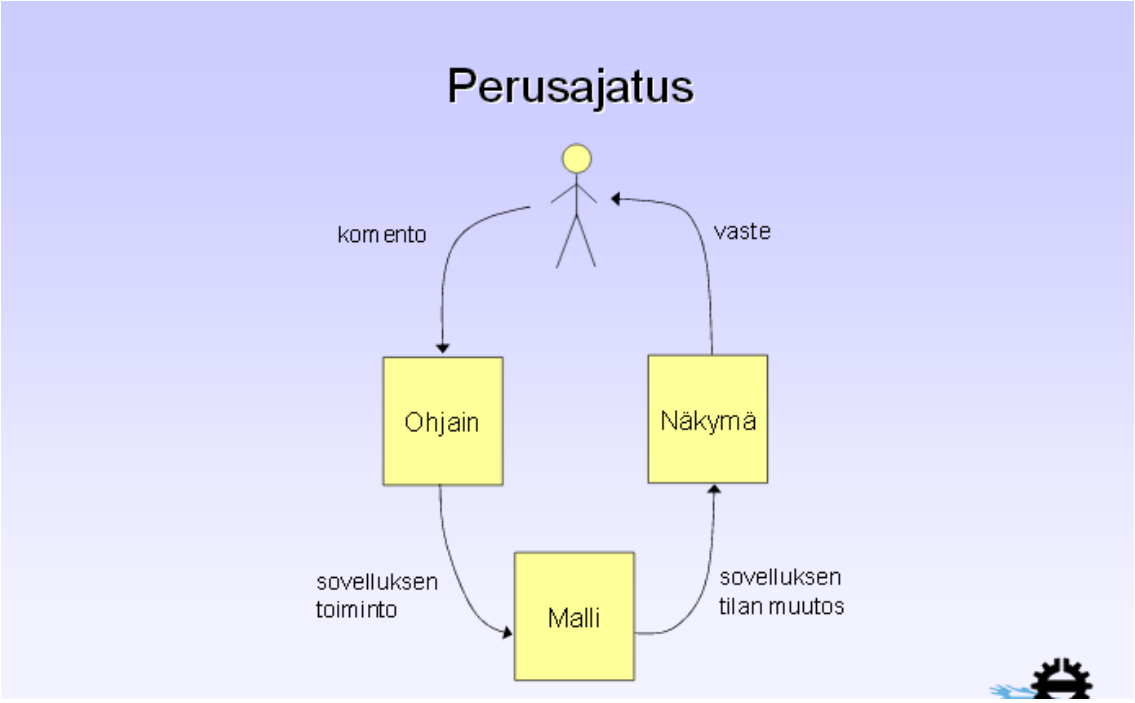
tiin, mitä asioita kirjoittajat olivat löytäneet hyödyllisiksi työtehtävissään. (Kelly 2008, 17-18.)

Ketterät menetelmät valmistautuvat jatkuvasti muuttumaan ohjelmistokehityksen aikana. Ympäristö, henkilöt, ohjelman halutut toiminnot sekä monet ennalta arvaamattomat muutokset jatkuvasti vaikuttavat ohjelmistokehitykseen. Tämän takia onkin tärkeää, että projektiryhmä kykenee kehittämään omia taitojaan ja sopeutumaan uusiin haasteisiin. Jatkuvien muutosten myrskyssä on tärkeätä tiedostaa ohjelmiston etenemisen vaiheet. (Kelly 2008, 17, 75-78, 103.)

Ketterissä menetelmissä jaetaankin ohjelmiston kehitystyö pieniin osiin. Eri osat pyritään jakamaan ajallisesti jaksoihin, joiden aikana yksittäinen osa ohjelmistoa halutaan saada valmiiksi. Näin ohjelmiston valmistuminen etenee selkeiden virstanpylväiden mukaisesti ja eri osia voidaan helposti iteraatioiden avulla kehittää eteenpäin sekä korjata. Iteraatio osuuksien välillä voidaan tarkastella tuotoksia ja suunnitella seuraavan jakson sisältöä. Ohjelmistokehityksessä iteraatiojaksot on yleensä jaettu kalenteriviikkojen mukaan. Ajanjakson pituus voi vaihdella riippuen projektista ja tekijöistä. Scrum-yhteisössä näitä ajanjaksoja kutsutaan sprinteiksi.

3.3.3 Model-View-Controller -arkkitehtuuri

Model-View-Controller (MVC)-arkkitehtuuri on ohjelmistokehityksessä käytettävä arkkitehtuurillinen tyyli, jossa ohjelma on jaettu kolmeen eri osaan: malliin, näkymään ja käsittelijään (kaavio 4). Jokainen osa on kehitetty tekemään sille tarkoitettua tehtävää ja kommunikoimaan tulokset toisille osa-alueille. Näin käyttäjän syöttäessä tietoa ja antaessa komentoja käy ohjelmisto läpi eri osa-alueet ja palauttaa käyttäjälle tiedon toiminnon tuloksesta. Käyttöliittymän pitäisi näin näyttää ohjelmiston tilan muutokset välittömästi käyttäjälle. (TTY ohjelmistotekniikka 2008, 1-4.)



Kaavio 4: MVC-arkkitehtuurin perusajatus (TTY ohjelmistotekniikka 2008, 2.)

Arkkitehtuurissa View eli näkymä toimii tiedonvälittäjänä ohjelmistosta käyttäjälle. Toisin sanoen, se antaa eri näkymiä käyttäjälle. Näkymän tehtävänä on vastaan ottaa tietoa Modelilta ja esittää tieto käyttäjälle hyödyllisessä muodossa. Controller vastaanottaa käyttäjän antamia käskyjä ja välittää tiedon eteenpäin ohjelmiston toimintojen avulla mallille. Modelin eli mallin tehtävänä on hoitaa ohjelmiston toiminnallisuus. Malli toimii ohjelmiston ytimenä ja hoitaa tärkeät toiminnallisuudelle välttämättömät muutokset ja laskelmat. MVC-arkkitehtuurin hyötynä on mahdollisuus useiden näkymien hyödyntämiseen, koska tieto on synkronoitua. Ongelmaksi saattaa koitua useiden näkymien päivityskutsujen määrä. Jos näkymien päivityksiä täytyy tehdä paljon, on myös mahdollista, että mallin tekemät kyselyt aiheuttavat hitautta ja lisäävät suoritusaikaa. (TTY Ohjelmistotekniikka 2008, 2-8.)

4 Tutkimusmenetelmät

Koska tutkimuksen keskeisenä osana oli tutkia kohdeyritykselle tehtyä ohjelmistoa ja ohjelmiston kehitystä, on tutkimusmenetelmien valinnassa vaihtoehdot rajoittuneet laadullisiin tiedonkeruu menetelmiin. Tämän takia tutkimusmenetelmäksi on valittu toimintatutkimus, jossa hyödynnetään laadullisia tutkimusmenetelmiä tiedon keruussa. (Aalto-yliopisto 2014.)

Toimintatutkimus on tutkimusmenetelmä, jossa tutkimusta toteutetaan osana tai tiiviissä yhteistyössä tutkittavan kohteen kanssa. Toimintatutkimuksessa selvitetään nykyinen toiminta ja indentifioidaan mitä kohtia halutaan kehittää. Kun korjattavat kohdat on identifioitu, on tarkoitus löytää ratkaisuja ongelmiin ja pyrkiä korjaamaan ongelmat. Toimintatutkimuksessa

tutkija sekä tutkittava ovat vuorovaikutuksessa keskenään. Tämän takia on mahdollista, että tutkija on osana vaikuttamassa tutkittavan toimintaan. Tutkimusta voidaan suorittaa esimerkiksi työympäristössä yhdessä muiden työntekijöiden kanssa. Tutkimusmenetelmässä keskitytään ymmärtämään ja ratkaisemaan ristiriitoja sekä ongelmia. Toimitatutkimus sopi hyvin tutkimaan yrityksen ohjelmistokehitystä, koska menetelmällä pyrittiin juuri intensiivisesti keskittymään ohjelmistokehityksen ongelmien löytämiseen ja ratkaisemiseen. (McNiff & Whitehead 2000, 198-203.)

Laadullisissa tutkimusmenetelmissä keskeisessä osassa ovat luonnolliset todellisen elämän tilanteet, joita tutkimalla saadaan tutkijan havaintoihin perustuvaa tietoa. Tarkoituksena onkin saada hyvin yksilöllistä tietoa kohteesta, jonka avulla voidaan induktiivista analyysia hyödyntäen havainnoida odottamattomia seikkoja tutkittavasta kohteesta. Lähtökohtana ei ole teorian tai hypoteesin testaaminen vaan aineiston monitahoinen ja yksityiskohtainen tarkastelu. (Hirsjärvi, Remes & Sajavaara 2007, 156-160.)

Tutkimusmenetelmiä mietittäessä otettiin huomioon tutkittava aihe eli kohdeyrityksen ohjemistokehitysprojekti. Yrityksellä ei ole ollut aiempaa ohjelmistokehitys taustaa muuta kuin harjoittelijan tekemä WebCRM- eli asiakkaanhallintajärjestelmä. Tämän takia ei ollut mahdollista hyödyntää tilastollisia tiedonkeruumenetelmiä, koska laajamittaista tietoa yrityksen ohjelmistokehityksestä ei ollut. Myös prosessimalli puuttui yritykseltä. Ohjelmistoja tehtiin laajalti harjoittelijoiden osaamisen mukaan, eikä ohjeistavaa toimintamallia löytynyt.

Reliabiliteettia ja validiteettia koskevia kysymyksiä tarkastellaan eri tavoin kvalitatiivissa ja kvanitatiivisessa tutkimuksessa. Kvantitatiivisissa tutkimuksissa validiteettikysymykset liittyvät usein tunnuslukujen käyttöä koskeviin valintoihin sekä valitun otannan edustavuuteen. Kvalitatiivisissa tutkimuksissa täytyy tarkastella tutkijan rakentamaa tutkimusasetelmaa ja valittua kohderyhmää. Validiteettikysymys on läsnä koko kvalitatiivisen tutkimuksen aikana. Tutkimuksen aikana on tärkeätä jatkuvasit verrata teoreettista viitekehystä ja aineistosta nousevia käsitteitä toisiinsa. Suhteen on tärkeätä olla looginen samoin kuin teoreettisten johtopäätösten ja empiirisen aineiston suhde. (Anttila 1998.)

Reliabiliteettiä eli luotettavuutta käytetään myös yleensä kvantitatiivisissa tutkimuksissa ilmaisemaan tutkimusmenetelmän tai mittausten luotettavuutta ja toistettavuutta ilmiön tutkittaessa. Varsinkin mittauksissa reliabiliteettiä käytetään mittaustulosten toistuvuuden tarkasteluun ja mittausvirheen arviointiin. Reliabiliteetin avulla on tarkoitus varmistaa, että tutkimuksen avulla saatu tieto on hyödyllistä eikä sattumanvaraista. (Virtuaali ammattikorkeakoulu 2015.)

Näiden syiden takia päätin valita laadullisen tiedonkeruumenetelmän tätä tutkimusta varten. Laadullinen tutkimusmenetelmä toimii tässä työssä hyvin, koska olin itse toteuttamassa ohjelmistoa ja osana ohjelmistokehitystä. Näin kykenin havainnoimaan ohjelmistokehityksen aikana erilaisia ongelmia oikeissa työtilanteissa. Tällä tavoin saatu tieto on oikeasti hyödyllistä yrityksen toimintaa ajatellen ja käytettävissä kehitysehdotuksia suunniteltaessa. Laadullisten menetelmien avulla saan tietoa, jonka avulla kykenen antamaan yritykselle kehitysehdotuksia ohjelmistokehitykseen ja ohjelmistokehitysprosessin luomiseen.

Tutkimusmenetelmää valittaessa päädyin hyvin nopeasti laadullisen tutkimuksen puolelle. Yrityksen ohjelmistokehitystä tutkittaessa ja kehitysideoita mietittäessä, on tärkeätä saada mahdollisimman monipuolinen ja yksityiskohtainen kuva yrityksen prosessista ja toiminnasta. Tähän kohtaan laadullinen tutkimusmenetelmä sopi loistavasti tuomaan tietoa prosessista.

5 Huollon mobiilisovellus

Kohdeyritys on ammattikeittiöalalla toimiva suomalainen yritys. Yritys tarjoaa ammattikeittiölaitteita, -tarvikkeita sekä palveluita yksityiselle ja julkiselle sektorille. Yrityksen päätoimipaikka on Helsingissä, josta löytyy myyntinäyttely, keskusvarasto, huolto sekä yhtiön hallinto. Yrityksellä on myös muutamia tytäryhtiöitä Eestissä sekä Suomessa.

Itse päädyin yritykselle harjoittelijaksi it-osastolle ohjelmoimaan uusia ohjelmistoja ja korjaamaan sekä kehittämään eteenpäin aiempia ohjelmistoja. Harjoittelun ohessa päädyin tekemään opinnäytetyöni ohjelmoimastani huollon mobiiliohjelmistosta ja sen ohjelmistokehitysprosessista.

5.1 Yrityksen tarve

Yrityksellä oli useita toimintoja, joita haluttiin kehittää sekä omia ideoita toimintojen kehittämiseen. Yksi näistä ongelmista, johon myös tämä opinnäytetyö keskittyy, oli huoltotöiden hallinta- ja kirjausjärjestelmä. Yrityksellä oli suuri tarve helpottaa huoltotöiden kirjausta ja laskuttamista. Yrityksen huoltotyöt laskutettiin tekemällä paperiset tositteet ja tuomalla ne takaisin toimistolle prosessoitavaksi. Tätä prosessia haluttiin kehittää tehokkaammaksi. Tehokkuuden saavuttamiseksi haluttiin siirtää töiden kirjaus sähköiseen muotoon, jolloin työt voidaan prosessoida ja laskuttaa nopeasti ja suoraviivaisesti. Huoltotöiden hallinnoinnissakaan ei ollut ennalta minkäänlaista ohjelmistoa, jolla töitä voitaisi helposti seurata tai jakaa reaaliaikaisesi huoltomiehille.

Harjoittelua aloittaessani kohdeyrityksellä oli jo tehty sopimus ohjelmistoja valmistavan yrityksen kanssa järjestelmän rakentamisesta. Jatkuvien myöhästymisien takia sopimus purettiin

ja ohjelmisto aiottiin hankkia mahdollisesti toiselta yritykseltä. Samoihin aikoihin yrityksen tietohallintopäällikkö tiedusteli osaamistani asian suhteen ja muutamien päivien miettimisen jälkeen uskoin kykeneväni rakentamaan ohjelmiston.

Jokainen huoltomies käytti jo ennaltaan työssään iPadia, joten alusta alkaen oli hyvin selvää, mille laitteistolle uusi järjestelmä haluttiin suunnitella. Toiseksi oli hallinnointipuoli, jonka haluttiin toimivan konttorin tietokoneilla. Uskottiin, että web-pohjainen järjestelmä olisi paras vaihtoehto ja mahdollistaisi toiminnan useiden laitteiden välillä. Tällöin ohjelmistoa voisi käyttää myös muilla mobiililaitteilla. Lopulta päädyttiin tekemään järjestelmä mobiililaitteille web-pohjaisia teknologioita käyttäen.

5.2 Määrittely

Ohjelmistokehityksen alkuvaiheessa yrityksellä oli jo selvänä aiemmin mainittu tarve, johon haluttiin ratkaisu. Tämän takia määrittelyä oli helppo lähteä tekemään. Halutun ohjelmiston oli siis tarkoitus pystyä kirjaamaan töitä helposti ja yhteisiä standardeja käyttäen. Näin työt saadaan tallennettua aina samaan paikkaan organisoidusti, mikä helpottaa töiden hallintaa. Yhtenäisen järjestelmän avulla laitteiden- ja varaosien tiedot tuodaan kaikille laitteille samasta paikasta ja pysyvät näin yhtäläisinä kirjatessa töitä.

Tärkeänä ominaisuutena painotettiin ohjelmiston toimimista offline-tilassa. Offline-tila nähtiin tarpeelliseksi, koska töitä tehtiin välillä maanalaisissa tai kaukaisissa paikoissa. Näissä paikoissa ei voida olla varmoja internetyhteyden laadusta. Offline-toimintojen takia oli selkeää, että ohjelmisto tulisi painottumaan vahvasti javascriptin varaan. Javascript toimii Clientin eli tässä tapauksessa selaimen puolella. Tämä tarkoittaa sitä, että javascript-koodi pyörittää selaimen puolella tarvittavia toimintoja, vaikka laitteella ei olisi internet-yhteyttä.

Javascriptin muuttujat eivät itsessään tallenna tietoa pysyvästi ja selaimen cookies-ominaisuus, joka tallentaa tietoa ja on mukana jokaisessa palvelin kutsussa, ei soveltunut hyvin laajemman tiedon tallentamiseen. Tämän takia offline-tilan tiedon tallennusta varten tarvittiin pysyvämpi ratkaisu. Alun määrittelyä tehtäessä päätettiin, että tietojen pysyvämpää tallennusta varten käytettäisiin selaimen HTML5 Local Storagea. Local storage mahdollistaa tiedon tallentamisen tekstimuodossa itse selaimeen, jolloin tiedot pysyvät tallessa selaimessa, vaikka nettiyhteyttä ei olisi saatavana. Näin pystytään hoitamaan tiedon eheys myös offline-tilassa.

Lopuksi ajan salliessa haluttiin ohjelmistoon lisätä hallintanäkymä, jossa työt olisi mahdollista nähdä selkeästi kartalta. Näin töitä olisi mahdollisuus myös allokoida oikeille henkilöille tämän näkymän avulla. Ohjelman avulla helpotetaan töiden hallinntapuolen tehtäviä ja reaali-

aikaisen tilanteen hahmottamiskykyä. Hallintaosuutta pidettiin jo määrittelyä tehtäessä toissijaisena ja haluttiin keskittyä ensin itse töiden kirjausohjelman rakentamiseen.

5.3 Suunnittelu

Suunnittelua aloitettaessa oli tiedossa jo pääpiirteittäin miten ohjelmiston haluttiin toimivan. Yrityksellä oli jo kuvalliset rautalankamallit ohjelmistosta, joiden pohjalta lähdettiin suunnittelemaan eteenpäin ohjelmiston toiminnallisuutta ja ulkonäköä.

Suunnittelun ensimmäinen askel oli palaveri, jossa mukana oli huollon työntekijöitä, huollon kehityspäällikkö, minä ja tietohallintopäällikkö IT-tiimistä. Tietohallintopäällikkö toimi projektin vastaavana henkilönä ja ohjasi toiminnan etenemistä. Palaverissa käytiin läpi ohjelmiston mallikuvia ja kirjattiin ylös kaikki ongelmakohdat ja muutostarpeet mitä mallikuvien perusteella saatiin. Samalla pyrittiin saamaan mahdollisimman paljon tietoa huoltotöiden kirjaamisesta huoltomiehiltä ja kysyttiin heidän mielipiteitä sekä ehdotuksia ohjelmiston toiminnan suhteen. Näin kykenin paremmin hahmottamaan huoltotyöprosessia ja töiden kirjaamista. Yrityksen toiminnasta tietämättömänä henkilönä minun oli tärkeätä päästä paremmin sisälle yrityksen toimintaan ja huoltotyöprosessiin.

Tietohallintopäällikön kanssa päätimme rakentaa ohjelmiston JQueryä ja JQuery Mobilea hyödyntäen. Tähän ratkaisuun päädyttiin, koska yrityksen aiemmatkin ohjelmistot oli valmistettu kyseisiä teknologioita hyödyntäen. Näin projektin aikana oli valmista koodipohjaa käytettävissä. Tämä helpotti merkittävästi ohjelmiston valmistusprosessia, kun kykenin monessa kohdassa hyödyntämään aiemmin kirjoitettua koodia ja valmiita JQuery Mobilen tarjoamia muotoiluominaisuuksia. Varsinkin itselleni melko vieraan javascriptin ja JQuery kirjastojen käytössä oli merkittävä apu, kun kykenin katsomaan mallia toisen ohjelmiston koodista.

Mitään tarkempia suunnitelmia ei tehty suunniteltaessa ohjelmiston rakentamista. Suunnittelupalaverissa keskityttiin täysin miettimään ja määrittelemään huoltomiesten tarpeiden pohjalta ohjelmiston toimnnallisuutta. Yrityksellä ei myöskään ollut ennalta valmiiksi suunniteltuna ohjelmistokehitysprosessin vaiheita tai työtapoja. Tämän takia ohjelmiston tarkempi tekninen suunnittelu ja toteutus jäivät suurimmaksi osaksi minun harteilleni. Heti suunnittelupalaverin jälkee minulle annettiin vapaus lähteä toteuttamaan ohjelmistoa itse parhaaksi näkemälläni tavalla.

5.4 Ohjelmiston rakenteen suunnittelu

Päätin alkuun suunnitella itsenäisesti ohjelmiston toiminnallisuutta tarkemmin. Mietin, miten ohjelmiston toiminnot jaettaisiin erillisiin osiin, jotka suorittaisivat tiettyjä toimintoja. Käytin

hyödyksi koulussa oppimiani luokkakaavioita ja ohjelmistokehityksen käytänteitä. Pyrin suunnittelussa käyttämään MVC-arkkitehtuuria. Jaoin toiminnot eri tiedostoihin, jotka suorittivat omat roolinsa. Nämä javascript-tiedostot olivat päätiedosto (main_controller.js), datan hallintatiedosto (data_controller.js), kirjoittajatiedosto (service_writer.js), aputiedosto (functions.js) ja palvelinpuolen toiminnallisuus PHP-tiedostojen avulla.

Päätiedosto (main_controller.js) toimi ohjelmiston ytimenä. Tämän scriptin ajo kerää tarvittavat toiminnallisuudet muista osista ja asettaa kaikki tarvittavat toimminnot HTML-sivun elementeille. Datan hallintatiedosto (data_controller.js) tarkoituksena on toimia ohjelmiston tiedon käsittelijänä. Tämän tiedoston koodi hoitaisi siis MVC-mallin controllerin eli käsittelijän tavoin lähettäen dataa PHP-tiedostoihin, jotka hoitavat tiedon lopullisen käsittelyn, tietokanta yhteydet ja tiedon tallentamisen sekä hakemisen SQL:ää hyödyntäen.

Kirjoittajatiedoston (service_writer.js) tarkoituksena oli toimia MVC-arkkitehtuurin näkymän tavoin ja hoitaa tiedon välitys käyttäjälle. Tämän tiedoston tehtävänä on hoitaa HTML-dokumentin kirjoittamiseen liittyvät toiminnnot. Functions javascript-tiedostoon on kerätty apufunktioita, jotka tekivät yksittäisi toimintoja ja eivät sopineet mihinkään aiempaan kategoriaan. Lisäksi ohjelmistossa käytetään PHP-skriptejä siirtämään SQL-kyselyjen avulla dataa palvelimen tietokannoista käyttäjälle ja käyttäjältä palvelimen tietokantoihin.

5.5 Toteutus

Projektin alusta lähtien oli tiedossa, että ohjelmoisin ohjelmiston itse ja vastaisin myös rakenteellisesta suunnittelusta. Koska ohjelmiston tuli toimia offline-tilassa oli selkeää, että visuaalinen rakenne tultaisiin rakentamaan yksittäiseen HTML-tiedostoon. Sivujen vaihdot voitiin toteuttaa helposti Jquery mobilen omien toimintojen avulla. Jquery mobilen avulla kyettiin hyödyntämään div-elementtien ID-attribuutteja luomaan erillisiä näkymiä ohjelmistoon. Sivujen vaihdot toimivat yksinkertaisesti vaihtamalla linkkien href-osoitteen #idnimi-muotoon. Jquery mobilen tyylimuotoilut mahdollistivat sivustolle tyylikkään ulkonäön yksinkertaisten luokka nimien avulla. Näin visuaalinen ilme saatiin helposti ja nopeasti rakennettua mallikuvien laisiksi.

Visuaalisen ilmeen ja sivujen valmistuttua lisättiin ohjelmistoon toiminnallisuus. Alkuun olin rakentanut sekä data_controllerin että main_controllerin toiminnot main_controlleriin. Ajan myötä tuli kuitenkin selväksi, että jos koodista haluaa saada mitään selvää, toiminnot oli jaettava omiin tiedostoihinsa. Kuvan 1 main_controller-tiedostoon jätettiin init()-funktio, events()-funktio sekä muutama muu funktio, jotka javascriptin toiminnallisuuden takia oli helpointa sisällyttää main_controlleriin. Main_controller-tiedostossa events()-funktioon, kerättiin kaikki elementtien tapahtumat kuten painallukset. Tämä funktio sisältää ryhmän funk-

tioita, jotka vastaavat käyttäjän syöttämiin painalluksiin ja tapahtumiin. Init()-funktion tarkoituksena oli valmistella ja ajaa kaikki tarvittavat toiminnot dokumentin valmistuessa.

main_controller.js

```
(function(data_controller, service_writer, functions, jQuery){
    .....
    data_controller.function1(); //Kutsu funktiota data_controller objektista

    //Tapahtumien kutsut sisältää erilaiset klikkauksien tapahtumat yms.
    function events(){
        $('.button').click(function(event){
            //Tee jotain kun nappulaa klikataan
        });
        ... //Lisää tapahtumia
    }

    //Alusta ohjelmisto
    function init(){
        events(); //Aja events funktio, jolloin tapahtumat lisätään dokumenttiin
        ... //Käy läpi kaikki tarvittavat funktiot
    }

    ..... //Lisää funktioita

    init(); // Aja alustus funktio jolloin ohjelmisto tulee käyttöön
})(data_controller, service_writer, functions, jQuery) //Kutsuu itsensä lopusaa
                                                       //käyttäen annettuja parametrejä
```

Kuva 1: Main_controllerin rakenne.

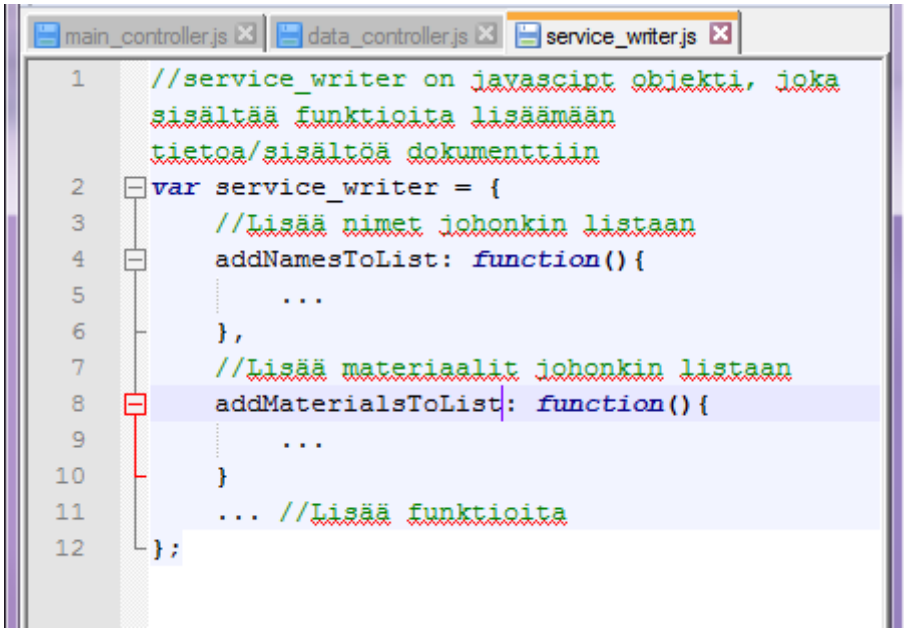
Kuvan 2 data_controller-tiedosto sisälsi kaiken tiedonkulkuun tarvittavan koodin. Data_controlleri lähetti AJAX-kutsuja palvelimen puolelle hakien tarvittavaa tietoa ohjelmiston käyttöön. Tiedot saatiin palvelimelta JSON-muotoisena javascript-oliona. Tiedot voitiin sitten lähettää service_writerin käsiteltäväksi tai tallentaa selaimen muistiin. Data_controller toimi myös tiedon tallentajana ja hakijana selaimen puolella. Offline-toiminnallisuuden saamiseksi täytyi tietoa tallentaa selaimen muistiin string- eli merkkijonomuotoisena. JSON-oliot muutettiin javascriptin JSON-komentojen avulla string-muotoon tallennusta varten. JSON.stringify()- ja JSON.parse()-funktiot mahdollistivat tiedon muuntamisen merkkijonosta objektiksi sekä toisinpäin.

main_controller.js data_controller.js

```
1  //Data_controller hallitsee erilaiset dataan liittyvät toiminnot kuten
   selaimen muistiin tallennus ja palvelimelle kutsujen lähettäminen
2  var data_controller = {
3      getData: function(indicator){
4          //Hyödyntäen jQuery kirjaston AJAX kutsu toimintoa
5          // $ = jQuery,  $.get = käytä jQueryn get() funktiota joka tekee AJAX
           kutsun annettuun osotteeseen
6          $.get("palvelin_tiedosto.php").success(function(response){
7              //Tee jotain onnistuneen vastauksen saavuttua
8          });
9      },
10     //Lähetä dataa palvelimelle
11     sendData: function(data){
12         //Hyödyntäen jQuery kirjaston AJAX kutsu toimintoa
13         //Lähetä POST kutsu palvelimelle mukana tietona nimi
14         $.post("palvelin_tiedosto.php", {nimi: "Aleksi"}).success(function(
           response){
15             //Tee jotain onnistuneen vastauksen saavuttua
16         });
17     },
18     //Tallenna dataa selaimen muistiin
19     saveData: function(data){
20         ...//Tee jotain
21     }
22
23 };
```

Kuva 2: Data_controllerin rakenne

Kuvan 3 service_writer-tiedosto toimi dokumentin kirjoittajana ohjelmistossa eli tämän olion avulla muutettiin palvelimelta haettu data käyttäjälle hyödylliseen muotoon. Tämä tiedosto keräsi service_writer olioon kaikki funktiot, joiden tehtävänä oli lisätä tietoa dokumenttiin. Esimerkiksi töihin kuuluvan tiedon lisääminen käyttäjälle näkyvään listaan tai virheilmoitusten näyttäminen käyttäjälle hoidettiin service_writerin avulla. Kirjoitettavat tiedot tuotiin funktioihin parametreinä main_controllerin sisällä. Tiedot siirrettiin JSON-objektien muodossa parametreinä service_writer-tiedoston funktioille. JSON-objektien avulla data pysyi selkeästi jäsenneltynä ja se pystyttiin helposti siirtämään järkevään muotoon dokumenttiin nähtäväksi.



Kuva 3: Service_writerin rakenne

Ohjelmiston kehityksen yhteydessä oli jatkuvasti mukana huoltotöiden puolelta kehityspäällikkö ja IT-osastolta projektin vetäjänä toiminut tietohallintopäällikkö. Heidän kauttaan tuli jatkuvasti tietoa huoltomiehiltä ohjelmiston ongelmista ja tarvittavista ominaisuuksista. Tietohallintopäällikkö auttoi kokemuksensa ja yrityksen tietokanta rakenteen ymmärtämisen takia monissa tietokantaan liittyvissä ongelmissa. Yhteydenpitoa hoidettiin sähköpostin avulla, paikan päällä keskustellen ja muutaman viikon välein pidetyissä kokouksissa.

Ohjelmiston isoin ongelma oli offline-toiminnallisuuden toteuttaminen. Jos yhteyttä palvelimelle ei jostain syystä saatu, syötetyt tiedot piti tallentaa selaimen muistiin. Valmiit muistiin tallennetut työt täytyi myöhemmin lähettää palvelimelle automaattisesti, jos toimiva internetyhteys havaitaan. Tietokantaan tallennettaessa täytyi tarkastaa, mitkä tiedot oli jo tallennettu palvelimelle ja poistaa onnistuneesti tallentuneet tiedot selaimen muistista. Yhdeksi isoimmaksi ongelmaksi nousi osittaisen tiedon tallentuminen tietokantaan. Tällöin tietokantaan tallentamattomat tiedot poistettiin selaimen muistista yhdessä tallennettujen tietojen kanssa. Näin osa tiedoista katosi kokonaan tallentumatta mihinkään. Ongelman ratkaisuksi rakennettiin PHP-tiedostoon tietokantayhteyksiä hoitava lista onnistuneista ja epäonnistuneista tietokanta kirjoituksista. Tämä lista palautettiin takaisin JSON-muodossa javascriptin AJAX-kutsun vastauksena. Listan avulla kyettiin poistamaan selaimen muistista vain ne tiedot, jotka oli jo tallennettu ja merkitä muut tiedot tallentamattomiksi. Merkintöjen avulla tallentamattomat tiedot voitiin myöhemmin lähettää uudelleen automatisoidun tarkistusfunktion avulla.

Tietokantatoimintojen testausta toteutettiin jonkin verran testitaulukoihin, mutta alun jälkeen päädyttiin käyttämään aktiivisia tietokantatauluja. Lopullisessa versiossa tiedot kulkivat

ennalta rakennettujen trigerien avulla väliaikaisen taulun kautta lopullisiin kohteisiin. Testitietokannan käytöstäkin puhuttiin, mutta päädyttiin pääasiassa käyttämään reaaliaikaista tietokantaa. Tämä oli melko riskialtista ja mahdollisti vakavienkin virheiden tapahtumisen. Tietokannan tai taulun poistaminen olisi voinut aiheuttaa isojakin ongelmia. Tämän takia täytyi pitää erityistä varovaisuutta työskennellessäni aktiivisen tietokannan kanssa.

Kesä-heinäkuussa toteutettiin pilottitestin yhdessä valittujen huoltomiesten kanssa. Pilottitestin aikana testiin osallistuneet huoltomiehet merkitsivät työt vanhan kaavan mukaan paperille sekä uuteen huollon mobiilisovellukseen. Viestintä kulki pääasiassa huollon kehityspäällikön kautta, mutta välillä sain myös suoraa viestiä huoltomiehiltä. Pilottitesti tehtiin melko huonoon aikaan kesälomien keskellä. Tämä vähensi testin tehokkuutta ja ongelmia ilmentyi testin aikana melko vähän. Huoltomiesten löytämät ongelmat keskittyivät pääasiassa käytettävyyteen ja ulkonäköön. Hallinnointipuolen testaus ei näyttänyt olevan mukana tässä osassa, koska mitään virheitä PHP- ja tietokantapuolen osalta ei tullut.

Ohjelmiston käyttöönotto tapahtui elokuussa. Alkuun huoltotyöt kirjattiin pilottitestin tavoin paperille ja ohjelmistoon, jolloin ongelmia oli helpompi nähdä ja korjata. Heti alkuun ilmeni vakavia ongelmia tiedon kulkemisessa ja tallentumisessa palvelimelle, jota ei ollut huomattu pilottitestissä. Tiedot eivät tallentuneet tietokantaan ja töitä sekä varaosia tallentui välillä tuplana. Tuplaantumisongelma liittyi tietokannan omiin trigger-ominaisuuksiin. Triggerit ovat tietokantaohjelmaan kirjoitettuja toimintoja, jotka ajetaan kun tietty toiminto tapahtuu tietokannassa. Tarkoituksena oli siirtää väliaikaisesta taulusta tiedot oikeisiin yrityksen käyttämiin tauluihin. Tiggereitä oli jäänyt kaksi kappaletta, jolloin data siirtyi kahteen kertaan eteenpäin. Tietohallintopäällikkö ratkaisi tuplaantumisongelman nopeasti poistamalla ylimääräisen trigger-toiminnon tietokannasta.

5.6 Ongelmia ja muutostarpeita

Yhdeksi isoksi ongelmaksi ajan myötä nousi koodin rakenteen solmuuntuminen. Projektin aikana pyrin tekemään muutoksia ja korjauksia sitä mukaan, kun niitä huomattiin. Projektin lopun lähestyessä aloin huomata kuinka koodi oli solmuuntunut ja oli vaikeata enää hahmottaa, missä mikäkin toiminto sijaitsi. Uuden ongelman havaittua piti monesti käyttää reilusti aikaa selvittämään, missä kyseinen toiminto oli ja miten siihen päädyttiin.

Kuten toteutusosiosta jo huomattiin, yrityksellä ei ole oikeastaan minkäänlaista prosessia ohjelmistokehitystä varten. Ohjelmiston rakentaminen eteni melko spontaanisti oman osaamiseni mukaisesti. Yritys on tehnyt ohjelmistoja nyt kahden vuoden ajan harjoittelijoiden avulla. Tämän takia yrityksellä ei ole minkäänlaista kaavaa tai prosessia, jota seurata ohjelmistoja kehitettäessä. Ohjelmiston kehitys ja eteneminen on jätetty suurimmaksi osaksi har-

joittelijoiden keksittäväksi. Tämä aiheuttaa ongelmia eri tavoin ja työn tulokset jäävät vahvasti harjoittelijan oman osaamisen varaan. Harjoittelijoita auttaisi merkittävästi jos olisi suunniteltu kaava, joka auttaisi hahmottamaan hyvät ja tehokkaat käytänteet jo alusta alkaen.

Ensinnäkin projektin alussa olisi hyvä rakentaa aikataulu ja jaksottaa ohjelmistokehityksen eri vaiheet omiin lohkoihinsa. Näin on heti alusta alkaen jotenkuten selvää, missä mennään kunakin viikkona. Lohkot kannattaa jakaa viikkojen pituisiin väleihin. Esimerkiksi kunnolliseen suunnitteluun olisi hyvä varata ainakin muutama viikko aikaa ja pyrkiä tekemään mahdollisimman yksityiskohtaiset suunnitelmat, luokkakaaviot ja muut valmistelut. Ennen toteutuksen aloittamista on vielä hyvä pitää palaveri ja käydä läpi suunnitelma kohta kerrallaan. Tarkisaa, että kaikki ohjelmiston toiminnot on otettu huomioon ja miettiä mitä tietokannan tauluja ja tietoja käytetään. Hyvä suunnittelu takaa selkeämmän ohjelmiston rakenteen, helpottaa merkittävästi myöhempää työtä ja vähentää aiemman koodin korjailua.

Ohjelmiston testaamisen vakavuutta olisi hyvä painottaa kohdeyrityksessä. Pilottitestauksessa on tärkeätä olla kaikki ohjelmistoon liittyvät osa-alueet aktiivisesti mukana. Pitää tarkistaa, että käyttöliittymä toimii ja kaikki tarpeelliset toiminnot ovat mukana. On tärkeätä tarkistaa, että tiedot tulevat oikein perille tietokantaan. Aktiivinen tarkastaminen nopeuttaa ongelmien löytämistä ja helpottaa niiden ratkaisemista. Kyseisen ohjelman kanssa tuntui, että testausta oli tehty vain käyttöliittymäpuolella ja tietojen tallentamista ei ollut tarkastettu kunnolla, jos ollenkaan. Testauksen puutteellisuuteen vaikutti varmasti myös kesälomat, jotka osuivat osittain päällekkäin testin kanssa. Testausta varten olisikin ollut hyvä valita kunnollinen aikaväli lomien ulkopuolelta, jolloin töitä olisi ollut enemmän ja työntekijät paikalla.

Ohjelmiston jakaminen modulaarisesti erilaisia toimintoja tekeviin osiin mahdollistaisi yksittäisten osien manuaalisen testaamisen, jolloin virheiden korjaaminen voidaan keskittää ja virheet löytää helpommin. Testausta voitaisiin toteuttaa aina osioiden valmistuttua, jolloin testaus olisi kohdistetumpaa. Jaetut toiminnot helpottaisivat ohjelmiston rakenteen hahmottamista ja virheiden löytämistä sekä korjaamista myöhemmissä vaiheissa. Yrityksen ohjelmistokehitysprojekteihin kannattaa alkuun lisätä kannustusta olio-pohjaiseen ohjelmointiin teoreettisen tiedon ja erilaisten kirjojen avulla. Näin opastaa harjoittelija olio-ohjelmoinnin käytännön menetelmiin.

Ohjelmistokehityksen aikana kannattaisi pitää kunnollista lokia tehdyistä muutoksista. Nykyisessä toiminnassa korjauksia ei aluksi merkitty minnekkään ja vasta käyttöönotetussa versiossa aloin pitämään lokia eri versioista ja niihin tehdyistä muutoksista. Kannattaisikin heti alusta asti pitää jonkinlaista lokia tapahtumistta ja ohjelmiston etenemisestä. Lokien avulla voi-

daan myöhemmin tarkastella, milloin on mitäkin osa-aluetta muokattu ja hyödyntää tietoja korjauksissa.

Versionhallinnan puuttuminen toi myös omat ongelmansa. Versionhallinnan mahdollistamat paluut vanhoihin versioihin ja eri versioiden välinen helppo vertailtavuus helpottaisivat ohjelmointia suuresti. Versionhallintatyökaluissa on yleensä hyödyllisiä hallinnointiominaisuuksia. Esimerkiksi Gitin ja Githubin avulla on helppo tarkastella eri versioihin tehtyjä muutoksia aina yksittäisille riveille tehdyistä muutoksista saakka. Github tarjoaa mahdollisuuden myös kommunikoimiselle sekä tilastollista tietoa muutosten määrästä. Githubin ”Issues” toiminnolla on mahdollista listata erilaisia ongelmia, jotka vaativat muutoksia ja merkata ne tehdyiksi, kun tehtävä on saatu suoritettua. Tämä toimii hyvänä työkaluna töiden merkkaamiseen ja antaa mahdollisuuden muillekin kiinnostuneille nähdä ohjelmiston reaaliaikainen tilanne. Kohdeyrityksellä tätä olisi voitu käyttää esimerkiksi pitämään huollon kehityspäällikkö ajan tasalla uusimmista muutoksista.

6 Yhteenveto ja loppupäätelmät

Projektin aikana ja jälkeenpäin tutkiessa kävi selväksi, että kunnollista ohjelmistokehitysprosessia yrityksellä ei ollut. Projektin aikana tuli monia ongelmia. Vaikka osa ongelmista olikin oman osaamiseni puutetta, löytyi myös monia ongelmia, jotka hyvällä ohjelmistokehitysprosessilla olisi voitu välttää.

Tutkimuksessa on tutkittu vain yrityksen yhden projektin ohjelmistokehityksen kulkua ja yhden tekijän näkökulmasta. Tämä takia tutkimusmateriaalin kattavuus ei välttämättä ole paras mahdollinen. Useamman projektin tarkastelu olisi mahdollisesti tuonut monia muita näkökulmia ja löytöjä yrityksen ohjelmistokehityksestä. Ohjelmistokehitysprosessin puutteen takia olisi mielenkiintoista nähdä, mitkä kohdat ohjelmistokehityksessä toistuvat tekijöistä riippumatta.

Koska projektin tekijänä on ollut yksi henkilö, nousee ongelmaksi tekijän oma kyky niin projektin tekijänä ja tekemänsä projektin tarkastelijana. Yrityksessä ohjelmistokehityksen kulku seuraa vahvasti tekijän osaamista. Tämän takia eri projekteilla on erilaisia toimintatapoja ja ongelmia. Muuttujien vaihtelevuuden vuoksi, tutkimuksessa on pyritty löytämään projektin kautta yhtäläisyyksiä teoriasta löydettyihin ohjelmistokehityksen toimintamalleihin, jolloin tuloksia on voitu vertailla tiedettyihin toimiviin käytänteisiin. Hyödyntämällä teorian antamaa viitekehystä, kyetään löytämään oikeasti hyödyllisiä ohjelmistokehityksen toimintamalleja ja -tapoja kohdeyrityksen tarpeisiin.

Kohdeyrityksellä on useita erilaisia kehitettäviä asioita ohjelmistokehitykseen liittyen. Yksi tärkeimpiä kehittämisen aiheita on ohjelmistokehityksen prosessimallin luominen. Kunnollinen ohjelmistokehityksen prosessimalli helpottaisi suunnattomasti uusien harjoittelijoiden työtä ja toisi yleisesti selkeyden ohjelmistokehityprojektien etenemiseen. Ohjelmistokehityksessä seurattaisiin aina samanlaista kaavaa, jolloin esimiehen olisi helpompi ohjata uusia harjoittelijoita projektin alkuvaiheissa. Näin yritys itse pysyisi myös selkeämmin kartalla, missä vaiheessa kehitystä mennään. Projektin aikana jatkuvasti muutettiin ja lisättiin ominaisuuksia, jolloin täytyi uudelleen kehitellä ohjelmiston toimintaa vastaamaan vaatimuksia. Tämän takia yrityksen kannattaisi hyödyntää ketterien menetelmien antamia esimerkkejä ohjelmistokehityksessään.

Prosessimallin lisäämisen lisäksi kannattaa ottaa käyttöön konkreettisia ohjelmistoja, joiden avulla voidaan helpottaa työskentelyä projektin parissa. Versionhallintatyökalun lisääminen ja omaksuminen osaksi ohjelmistokehitysprosessia olisi hyvä ja selkeä aloituskohta. Versionhallintatyökallulla voitaisiin helposti seurata projektin etenemistä konkreettisella tavalla. Versionhallinnalla kyetään sekä näkemään että palaamaan aiempiin versioihin, jolloin työn seuraamisesta tulee helppokäyttöistä ja visuaalisesti hahmotettavaa. Projektin eri vaiheet ja ohjelmiston eteneminen kyettäisiin helposti dokumentoimaan hyödyntäen versionhallintatyökalua. Näin projektin hahmottaminen tulisi selkeämmäksi kaikille projektin osallisille.

Modulaarisuutta kannattaa hyödyntää ja painottaa, koska sillä voitaisiin helpottaa ohjelmistokehitystä. Ohjelmiston jakaminen osiin tekee työstä helpompaa ja kevyempää, kun on selkeät ohjelmiston osat, joiden valmistuminen voidaan jakaa ajallisesti virstanpylväisiin. Ohjelmiston osien testaaminen helpottuu, kun voidaan aina yhden osion valmistuttua suorittaa testausta ja käydä läpi muutostarpeet palavereissa. Näin yritys pysyy selkeämmin mukana missä mennään ohjelmiston suhteen ja mitkä osat ovat jo toimivia. Näin yrityksen työntekijät, jotka toimivat ohjelmiston loppukäyttäjinä, pääsevät paremmin kiinni ohjelmiston kehittämiseen. Käyttäjät pystyvät tarkemmin kommentoimaan osia ja niiden toiminnallisuutta, kun osia testataan erikseen ja niitä tarkastellaan tarkemmin sitä mukaan, kun osat valmistuvat. Tehokkaan modulaarisuuden toteuttaminen ei aina ole mahdollista harjoittelijan osaamisen takia, mutta painotus modulaarisuuten ja sen omaksumiseen olisi hyvä aloitus.

Yksi suurimpia puutteita projektin aikana oli kattavan testauksen puuttuminen. Testaukseen kannattaakin tulevaisuudessa panostaa enemmän ja pyrkiä tekemään testausta kunnolla, jolloin mahdollisimman monilta ongelmilta voitaisiin välttyä ohjelmiston käyttöönoton aikana. Testaus kannattaa ajoittaa tarkasti ja saada testikäyttäjät ymmärtämään testauksen tärkeys. Testausta ei ole hyvä suorittaa pelkästään kesälomien aikana, jolloin testausta ei voida suorittaa yhtä mittavasti töiden vähäisyyden ja testihenkilöiden lomien takia. Testaukselle olisi hyvä varata aikaa ajanjaksolta, jolloin testaukseen voidaan keskittyä ja toteuttaa tehokkaasti.

Työskentely yrityksessä projektin parissa oli mukavaa ja hyvin opettavaista. Opin töissä uskomattoman paljon uusia asioita javascriptistä, PHP:sta ja SQL:stä. En aluksi ollut täysin varma omasta osaamisestani projektin laajuuden ja teknisten ongelmien takia. Lopputulokseen olin kuitenkin tyytyväinen ohjelmiston näkökulmasta, mutta koodin puolella lopussa huomasin merkittäviä ongelmia ohjelmiston rakenteessa. Rakenteelliset ongelmat vaativat paljon aikaa ja aloin ymmärtää hyvän suunnittelun sekä erilaisten frameworkien arvon ohjelmistoa rakennettaessa. Tutkimusta tehdessä opin vielä lisää hyvistä käytänteistä ja niiden hyödyntämisestä tulevissa ohjelmistokehitysprojekteissa. Toivon tästä opinnäytetyöstä olevan myös hyötyä yrityksen tulevissa projekteissa.

Lähteet

Kirjallisuus

Haikala, I. & Märijärvi, J. 2004. Ohjelmistotuotanto. Hämeenlinna: Talentum Media Oy

Hirsjärvi, S., Remes, P. & Sajavaara, P. 2007. Tutki ja kirjoita. 13., osin uudistettu painos. Helsinki: Tammi

Kelly, A. 2008. Changing software development – Learning to become agile.

Klimczak, E. 2013. Design for Software: A Playbook for Developers. Chichester: John Wiley & Sons Ltd.

McNiff, J. & Whitehead, J. 2000. Action Research in Organisations. Lontoo: Routledge

Mishra, R.C & Sandilya, A. 2009. Reliability and Quality Management. New Delhi: New Age International

Sähköiset lähteet

Aalto-yliopisto. 2014. Tutkimusongelman muodostaminen. Viitattu 4.9.2014.
http://viestinnantietoaines.aalto.fi/Tieteellinen/ongelma.htm

Agile Alliance. Itreration. Viitattu 3.9.2015.
guide.agilealliance.org/uide/iteration.html

Anttila, P. 1998. Tutkimisen taito ja tiedonhankinta. Viitattu 19.11.2015.
http://www.metodix.com/fi/sisallys/01_menetelmat/01_tutkimusprosessi/02_tutkimisen_taito_ja_tiedon_hankinta/10_tutkimuksen_luotettavuus/10_2_2laadullisen_tutkimuksen_validiteetti

Github. Build software better, together. Viitattu 25.3.2015.
https://github.com

Git SCM. Git –distributed-is-the-new-centralized. Viitattu 25.3.2015.
www.git.scm.com

Helsingin yliopisto. 2009. Ohjelmistoprosessit ja ohjelmistojen laatu. Viitattu 21.10.2014.
http://www.cs.helsinki.fi/u/taina/opol/k-2009/pdf/luku-6_2.pdf

Helsingin yliopisto. 2015. Versionhallinta. Viitattu 20.11.2015.
https://www.cs.helsinki.fi/node/61379

Hiltunen, L. 2009. Validiteetti- ja reliabiliteetti. Jyväskylän yliopisto. Viitattu 19.11.2015.
www.mit.jyu.fi/ope/kurssit/Graduryhma/PDFt/validius_ja_reliabiliteetti.pdf

jQuery Mobile. 2015. A Touch-Optimized Web Framework. Viitattu 25.3.2015.
https://jquerymobile.com

Jyväskylän yliopisto. Laadullinen tutkimus. Viitattu 14.1.2015.
https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/tutkimusstrategiat/laadullinen-tutkimus

Lemonsoft. Lemonsoft. Viitattu 3.9.2015.
www.lemonsoft.fi

Lemonsoft. Lemonsoft palvelurajapinta. Viitattu 3.9.2015.
in-
fo.lemonsoft.eu/LemonNethelp/default.htm#!Documents/palvelurajapinnatwebservices.htm

Nurkse, A. 2013. Anna Nurkse: Lean-filosofia, SCRUM vai vesiputousmalli? Kirjoitus Tietotyömaan blogista 11.12.2013. Viitattu 15.10.2014.
http://tietotyomaa.meteoriitti.com/2013/12/11/anna-nurkse-lean-filosofia-scrum-vai-vesiputousmalli/

Saukkonen, P. Tutkimusmenetelmät. Viitattu 4.9.2014.
http://www.mv.helsinki.fi/home/psaukkon/tutkielma/Tutkimusmentelmat.html

TTY Ohjelmistotekniikka. 2008. Malli-näkymä-ohjain arkkitehtuuri. Viitattu 16.3.2015.
www.cs.tut.fi/~ohar/luennot/luennot2008/Ohar6%20Arkkitehtuurityylit3.pdf

Virtuaali Ammattikorkeakoulu. 2015. Tutkimuksen reliabiliteetti. Viitattu 19.11.2015.
http://www2.amk.fi/digma.fi/www.amk.fi/opintojaksot/0709019/1193463890749/1193464185783/1194413792643/1194415307356.html

w3schools. 2014. Learn Web Building. Viitattu 2.9.2014.
http://www.w3schools.com/default.asp

Kuvat

Kaaviot