SAVONIA

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
LUONNONTIETEIDEN ALA

ASIAKASHALLINTA-SOVELLUS ELYSIUM SOLUTIONS OY:LLE

TEKIJÄ/T: Joni Luostarinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Luonnontieteiden ala	
Koulutusohjelma Tietojenkäsittelyn koulutusohjelma	
Työn tekijä(t) Joni Luostarinen	
Työn nimi Asiakashallinta-sovellus Elysium Oy:lle	
Päiväys 12.2.2013	Sivumäärä/Liitteet 41
Ohjaaja(t) Anu Kettunen	
Toimeksiantaja/Yhteistyökumppani(t) Elysium Solutions Oy	

Tiivistelmä

Opinnäytetyön aiheena oli rakentaa asiakashallinta-sovellus Elysium Solutions Oy:lle helpottamaan harjoittelussa tehdyn palvelun käyttöönottoa. Tavoitteena asiakashallinnalla on myös mahdollistaa asiakasrajapinnalle omien tietojen ja sopimusten hallinta. Asiakashallinta toteutettiin web-sovelluksena Apache-ympäristöön ja tietokantana käytettiin MySQL-tietokantaa. Käytettyjä tekniikoita sovelluksessa olivat PHP, HTML, CSS, JavaScript ja jQuery. Työ toteutettiin etätyönä ja lähestymistapa kehityksessä oli ketterä. Pääpaino työskentelyssä oli saada aikaiseksi toimiva sovellus, jolloin dokumentaatio tapahtui lähinnä sovelluksen lähdekoodissa. Asiakashallinnan ulkoasun rakentamisessa käytettiin Twitter Bootstrap-rajapintaa, joka mahdollisti selaimessa käytettävän sovelluksen ulkoasun nopeamman ja tehokkaamman rakentamisen. Versionhallintana työssä käytettiin Git-sovellusta, joka mahdollisti muun muassa historiatietojen ylläpitämisen sovelluksen kehityksen aikana. Sovelluksen kehittäminen aloitettiin hahmottamalla sovellukselta odotettavat toiminnallisuudet, jotka kehityksen aikana hiottiin lopulliseen muotoonsa.

Työn lopputuloksena syntyi sovellus, joka huolehtii asiakkaille rakennettavien sivustojen tietokantoihin ja tiedostoihin liittyvistä toimenpiteistä sekä pitää yllä rekisteriä palvelun käyttäjistä. Työn aikana tärkeiksi seikoiksi havaittiin mm. asiakashallinnan tärkeys aikaisemmin rakennetun palvelun kannalta, Bootstrapin hyödyntäminen sovelluksen ulkoasun rakentamisessa ja versionhallinnan tärkeys sovelluskehityksessä. Asiakashallintaa voidaan hyödyntää jatkossa tukemaan aikaisemmin tehdyn palvelun käyttöönottoa asiakkailla ja asiakkaat pystyvät hyödyntämään asiakashallintaa käyttäessään palvelua. Sovelluksen jatkokehityksessä tulee huomioida varsinkin asiakasrajapinnasta tuleva palaute ja tarpeet, jotta sovellusta voidaan kehittää asiakaslähtöiseksi.

Avainsanat
PHP, asiakashallinta, Twitter Bootsrap, JavaScript, jQuery, Apache, ketterä ohjelmistokehitys

Field of Study Natural Sciences			
Degree Programme Degree Programme in Information Technology			
Author(s) Joni Luostarinen			
Title of Thesis Customer management for Elysium Solutions Oy			
Date	12.2.2013	Pages/Appendices	41
Supervisor(s) Anu Kettunen			
Client Organisation /Partners Elysium Solutions Oy			

Abstract

The subject of the thesis was to build a customer management application for Elysium Solutions Oy to help introduction of a service that was built in during the author's practical training. The objective of the customer management application was also to make it possible for customers to manage their data and contracts. Customer management was implemented as web based software in Apache environment, the database being MySQL. The technologies applied include PHP, HTML, CSS, JavaScript and jQuery. The software was buildt using remote work and the approach was agile. The main point was to build to working software and the documentation took place mainly inside the source code of the application. Twitter Bootstrap was used to build the customer management interface because Bootstrap made it possible to build the interface faster and more efficiently. As for version management, git was chosen.as the application allows creating a log of the stages of software creation among other things. Software development was started by outlining features that were expected from the software. The features were finalized during the development process.

The work resulted in a customer management application that handles databases and files that are used on customers' webpages. Customer management also stores data on application users. Important issues noticed during the development progress were the importance of customer management for a previously developed service, the use of Bootstrap in building the interface and the importance of version control during software development. In the future customer management can be used to support introduction of a previously created service among customers; customers can benefit from customer management when using the service. In the future it's important to listen to customers' feedback and needs. That information can be used to develop more customer-oriented customer management applications.

Keywords
PHP, customer management, Twitter Bootsrap, JavaScript, jQuery, agile

SISÄLTÖ

1 JOHDANTO

Ajatus ja tarve opinnäytetyön tekemiselle syntyivät harjoittelujakson aikana työskennellessäni ohjelmointitehtävissä Elysium Solutions Oy:ssä. Opinnäytetyöstä tuli jatkumo harjoittelulle. Ohjelmointitehtävissä olin rakentamassa sivustoa, jonka asiakkaita varten tarvittiin asiakashallinta ja opinnäytetyö tarjosi mahdollisuuden työkalun rakentamiselle. Toisaalta sain opinnäytetyön kautta mahdollisuuden syventää omaa osaamistani ja tutustua erilaisiin ohjelmointikieliin sekä ratkaisuihin. Koska opinnäytetyö on kuitenkin käytännönläheinen, pyrin kuvaamaan opinnäytetyössä myös käytännön prosessia ja sitä kuinka lopulta päästiin lopulliseen tuotokseen. Ohjelmistokehityksen muotona projektissa käytettiin tapaa, joka otti vaikutteita ketterästä ohjelmistokehityksestä. Tämä osaltaan siksi, että ajatus asiakashallinnan toiminnasta oli jo alkuvaiheessa selvä ja sitä tarkennettiin sovelluksen kehitystyön aikana. Pääpaino pidettiin toimivassa sovelluksessa eikä niinkään kokonaisvaltaisessa dokumentaatiossa. Kun ohjelmistoa kehitettiin pala kerrallaan, pystyttiin suunnitelmia tarvittaessa muuttamaan. Tämä osaltaan vähensi riskiä ominaisuuksille, jotka myöhemmin osoittautuisivat turhiksi kehitystyön aikana.

Asiakashallinta on toteutettu verkkosovelluksena, jonka käyttämiseksi vaaditaan lähinnä moderni Internet-selain ja JavaScriptin salliminen selaimessa. Asiakashallinnalla tarkoitetaan siis opinnäytetyössä projektin lopputuloksena syntynyttä sovellusta. Opinnäytetyössä pystyin hyödyntämään samoja tekniikoita kuin harjoittelujaksolla ja samalla saatoin kerrata sekä syventää osaamistani näiden tekniikoiden osalta. Sovelluksen kannalta suuri merkitys oli PHP-kielellä ja Twitter Bootsrapillä. Bootstrapin avulla olen pystynyt rakentamaan asiakashallinnan ulkoasun nopeammalla tahdilla kuin mitä vastaavan ulkoasun rakentaminen olisi alusta alkaen vaatinut. Toisaalta Bootstrap on mahdollistanut myös ulkoasun muokkaamisen projektin tarpeita vastaavaksi. PHP-kielellä olen vastaavasti pystynyt rakentamaan palvelinpuolella tapahtuvan toiminnallisuuden. Esimerkiksi tallennukset tietokantaan, tunnistautuminen ja sivujen sisältöön vaikuttaminen käyttöoikeuksien mukaan ovat toteutettu käyttämällä PHP-kieltä. Versionhallinnan käyttö osana projektia tuli minulle tutuksi jo harjoittelujaksolla ja vaikka sen käyttö ei muuttunut siirryttäessä projektista toiseen, osoitti se hyödyllisyytensä tämänkin projektin osalta. Ilman kunnollista versionhallintaa olisi versioiden hallinnoimisesta tullut paljon työläämpää.

Opinnäytetyön kirjallisen osion tarkoituksena on sisältää teoriaa projektiin liittyen ja kuvata käytännön toteutusta projektin osalta. Tämän lisäksi kuvataan käytännössä toteutetun sovelluksen toimintaa ja ominaisuuksia. Tämä opinnäytetyö on mahdollistanut itselleni sellaisten tekniikoiden käyttämisen, joita ei oppitunneilla käsitelty, mutta joiden käyttöönottoa koulussa saatu tietämys on helpottanut. Opinnäytetyö on myös vaatinut astumisen pois omalta mukavuusalueelta.

2 ELYSIUM SOLUTIONS OY

2.1 Yritys ja palvelut

Elysium on kuopiolainen it-alan yritys, jossa työskentelee opinnäytetyön kirjoitushetkellä kaksi henkilöä. Yrityksen liikevaihto oli 58 000 euroa vuonna 2012 ja sen toimialaan kuuluvat muun muassa ohjelmistojen suunnittelu ja toteutus (Taloussanomat, 2013). Ennen opinnäytetyön kirjoittamista suoritin opintoihini kuuluvan harjoittelun Elysiumissa ohjelmointitehtävissä.

Elysiumin palveluihin kuuluvat räätälöidyt verkkosovellukset, verkkosivujen rakentaminen, konsultointi, it-tuki ja erilaiset koulutukset. Palvelupaketti voidaan tarjota niin yrityksille kuin yksityishenkilöillekin. Kokemusta yrityksellä on verkkosovellusten osalta sopimusarkistojärjestelmistä ja raportointisovelluksista sekä terveydenhoitoalan arkistointiin ja näytteenseurantaan liittyvistä ratkaisuista. Verkkosovellusten käyttämistä perustellaan sillä, että ne ovat saatavilla siellä missä on Internet-yhteys ja esimerkiksi kotoa työskentely onnistuu helposti. Tärkeänä seikkana verkkosovellusten osalta Elysium näkee tietoturvan, joka huomioidaan varmuuskopioiden sekä palomuuri- ja ohjelmistoratkaisujen muodossa. Palvelintilan tarjoaminen ei kuulu palveluihin, mutta asiakkaalle tarjotaan konsultointia sopivan palveluntarjoajan löytämiseksi. Asiakas voi myös ulkoistaa sivutilan ylläpidon Elysiumille. (Elysium Solutions Oy, 2013)

Verkkosivujen suunnittelussa ja toteutuksessa asiakkaisiin kuuluvat niin yritykset kuin yksityishenkilötkin. Verkkosivuilla ei pelkästään tarkoiteta staattisia sivuja, vaan asiakkaalle voidaan toteuttaa myös keskustelualue, blogi ja sisällönhallintajärjestelmä. Elysiumin tarjoamat koulutuspalvelut kattavat esimerkiksi toimisto-ohjelmistojen, kuvankäsittelyn, tehokkaiden tietoturvamenettelyiden ja Linux-ympäristöjen koulutuksen. Konsultoinnin osalta Elysium tarjoaa asiantuntija-apua liittyen yritysten ja yhteisöjen tietotekniikka- ja tietokonehankintoihin. It-tuki kattaa ohjelmistojen ja laitteiden kanssa ilmenneet ongelmat sekä verkko-ongelmat. Lisäksi tarjotaan tietojenpalautuspalvelua, jolloin asiakkaan vahingossa poistamia tai kadottamia tiedostoja voidaan palauttaa. (Elysium Solutions Oy, 2013)

3 KÄYTETYT MENETELMÄT JA KESKEISET VÄLINEET

3.1 Ketterä ohjelmistokehitys

3.1.1 Keskeiset seikat

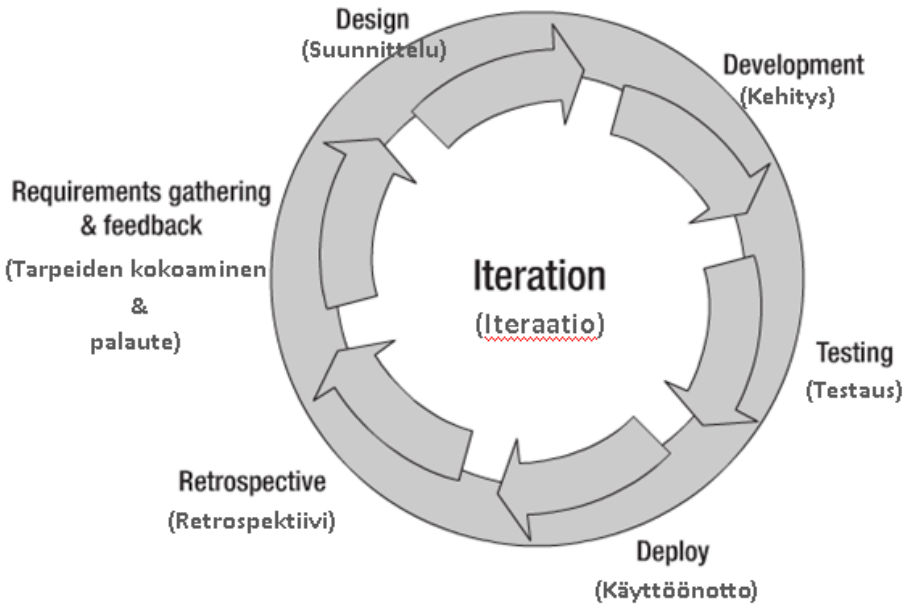
Ketterän ohjelmistokehityksen yksi ydinasioista on tekninen huippuosaaminen. Tämä osaltaan siksi, että onnistunut tuote vaatii keskittymistä laatuun. Kuinka tekninen huippuosaaminen ja laatu saavutetaan? Näiden saavuttamiseksi ketterässä ohjelmistokehityksessä käytetään:

- Testivetoista kehitystä (Testi-Driven Development), jossa testit ohjelmoidaan ennen testattavaa toiminnallisuutta
- Koodin katselmointia, joka voi tapahtua esimerkiksi pariohjelmoinnin kautta
- Tarkastuslistoja
- Iteratiivista kehitystä, jossa koodia mukautetaan uusien ideoiden ja muutosten myötä
- Refaktorointia, jossa koodia parannetaan vaikka ominaisuuksiin ei olisi tullut muutoksia

Ketterä ohjelmistokehitys vaatiikin uudenlaista suunnittelua. Määrittely parhaiden arkkitehtuurien kohdalla ei ole tapahtunut ylhäältä alaspäin. (Appelo, 2010, 23)

Ketterässä ohjelmistokehityksessä ihmisiä ei nähdä pelkkinä korvattavina resursseina, joilla suurin arvo on pään sisällä. Sen sijaan suurimpana arvona nähdään ihmisten välinen vuorovaikutus ja yhteistyö. Ketterässä ohjelmistokehityksessä tiimit ovat pieniä ja koostuvat eri roolin omaavista henkilöistä. Työpisteiden tulisi mielellään olla samassa huoneessa. Metodeja ja prosesseja ei määrätä ryhmälle, jonka vuoksi ryhmän tulee kyetä itsenäiseen organisoitumiseen. Tämän lisäksi luotetaan siihen, että ryhmä saa tehtyä työn valmiiksi parhaaksi katsomallaan tavalla. Tämä kuitenkin vaatii, että ryhmän tulee tietää oikeat tavat ja että ryhmä ottaa vastuun tuloksista. Yhtenä tärkeänä osana ketterää ohjelmistokehitystä nähdään, että tiimi tekee yhteistyötä asiakkaan tai asiakkaan edustajan kanssa. Tällä tavoin kasautuneita ominaisuuksia voidaan jatkuvasti uudelleenpriorisoida ja ylläpitää. (Appelo, 2010, 22-23)

Työkalujen merkitys onnistuneen tuotteen kannalta nähdään ketterässä ohjelmistokehityksessä vähäisenä, vaikka ketterää ohjelmistokehitystä käsittelevissä kirjoissa onkin työkaluja esitelty ja kuvattu. Kehitystyökalut, jotka ovat suosittuja kokeneiden tiimien kesken, liittyvät päivittäisiin ohjelmistoversioihin (daily build), automatisoituun testaukseen ja jatkuvaan integraatioon. Ketterässä ohjelmistokehityksessä tiimien tulee olla motivoituneita, jolloin tylsät ja epämotivoivat tehtävät tulee automatisoida. Tärkeänä nähdään myös kannustava ympäristö, johon kuuluvat esimerkiksi avoimet toimistot ja isot tehtävätaulut. Työkalujen tarkoituksena on ketterässä ohjelmistokehityksessä vahvistaa kommunikaatiota, motivaatiota ja tiimin sisäistä yhteistyötä. (Appelo, 2010, 23-24)



Kuvio 1. Ketterä prosessi. Lisätty suomennokset (Blankenship & Bussa & Millett, 2011 ,4)

Ketterässä ohjelmistokehityksessä käytetään lyhyitä iteraatioita, joiden tarkoituksena on ratkaista tietty liiketoiminnan ongelma. Yleisesti iteraatioiden pituus on kahdesta neljään viikkoa. Iteraatio sisältää itsessään suunnitelmavetoisen prosessin kaikki vaiheet ja mahdollistaa, että palautetta voidaan antaa säännöllisesti ja lyhyessä ajassa toimivan sovelluksen pohjalta. Kun suunnitelmavetoisessa (plan-driven) projektissa keskitytään koko projektiin, iteraatiossa työstetään pientä palaa projektista kerrallaan. Palat koostuvat systeemille asetetuista vaatimuksista, jotka sidosryhmät ovat määrittäneet tärkeimmiksi. Koska asiakas pystyy ottamaan toimivan sovelluksen käyttöönsä lyhyessä ajassa, voi asiakas saada arvoa sovelluksesta. Iteraatioiden avulla voidaan myös parantaa tarkkuutta laskettaessa koko systeemin kehitykselle varattavaa aikaa. Tämä johtuu ketterän ohjelmistokehityksen tavasta pilkkoa suoritteet pienempiin paloihin. Tällöin iteraatioiden parissa työskentelevä tiimi pystyy tyypillisesti kolmen tai neljän iteraation jälkeen antamaan melko tarkkoja arvioita ajankäytön suhteen. Kuviossa 1 kuvataan iteraation eri vaiheita ketterässä prosessissa. (Blankenship, Bussa ja Millett, 2011 ,4)

Suhtautuminen aikaan ei ketterässä ohjelmistokehityksessä ole täysin perinteinen. Osaltaan tämä johtuu mahdollisuudesta projektille asetetun takarajan, toimitusaikojen ja budjetin muuttamiseen lähes mielivaltaisesti. Sovelluksen tuottaminen tapahtuu lyhyissä sykleissä. Yleisesti sykleillä voidaan tarkoittaa aikaruutuja ja sprinttejä. Sovelluksen toimittaminen tapahtuu näin ollen nousevien (incremental) julkaisujen kautta, joista jokaisen tulisi olla toimitettavissa asiakkaalle. Tämä mahdollistaa yritysten omistajille ajankohtien hallinnan. Julkaisupäivämääriä voidaan muuttaa, riippuen siitä mitä ominaisuuksia yritysten omistajat haluavat ja milloin. (Appelo, 2010, 23-24)

Tarve osoittaa vastaus muutokselle oli yksi suurimmista syistä, miksi ketterän ohjelmistokehityksen manifesti aikanaan julkaistiin. Ympäristö muuttuu jatkuvasti eikä kestä muuttumattomana, jolloin tänä päivänä ohjelmistoon kehitetyt ominaisuudet saattavat huomiseen mennessä muuttua turhiksi. Ketterässä ohjelmistokehityksessä tähän haasteeseen pyritään vastaamaan lyhyillä toimitussykleillä ja lyhyellä palautteella. Usein toistuvien julkaisujen ansiosta uudet ja päivitetyt ominaisuudet pystytään toimittamaan asiakkaalle mahdollisimman nopeasti tarpeen ilmaantumisen jälkeen. Lisäksi usein toistuvat julkaisut mahdollistavat palautteen pyytämisen ympäristöltä ja löydöt voidaan sijoittaa takaisin kehitysprosessiin. (Appelo, 2010, 23-24)

Vaikka ketterän ohjelmistokehityksen ajatusmallissa ihmiset menevät prosessien edelle, ovat määrätyt prosessit tärkeitä myös ketterässä ohjelmistokehityksessä. Tärkeitä prosesseja ketterän ohjelmistokehityksen kannalta ovat minimaalinen suunnittelu, kasvotusten tapahtuva kommunikaatio ja toimivan sovelluksen arvioinnin kautta tapahtuva prosessin mittaaminen. Ketterän sovelluskehityksen parissa työskentelevän tulee tiedostaa jatkuva kehittymisen tarve. Tämän vuoksi prosesseja arvioidaan ja säädetään jatkuvasti. Arvioinnissa ja säätämisessä käytetään säännöllistä reflektointia. (Appelo, 2010, 24)

3.1.2 Ketterän ohjelmistokehityksen menetelmät yritysmaailmassa

Euroopassa on pilottien kautta saatu lupaavia tuloksia ketterien menetelmien osalta. Entistä isompi osa yrityksistä muuttaa toimintatapojaan ketteriksi ja tähän ovat osaltaan vaikuttaneet positiiviset kokemukset, joita on saatu kokeilujen, pilottiprojektien ja tutkimustulosten avulla. Tulosten kautta on myös saatu selville, ettei ketteriä menetelmiä käytetä pelkästään pienissä ohjelmistotiimeissä ja ei-kriittisten sovellusten tuotannossa. Myös suuret moniteknologiset yritykset, jotka soveltavat hajautettua alihankintaa ja tuotekehitystä etsivät ratkaisuja siihen, kuinka voisivat soveltaa ja ottaa käyttöön ketteriä menetelmiä. (Abrahamsson ja Salo, 2007)

Ketteryyden sovellutuksia eri sovelluskentissä on tutkittu VTT:n vetämässä AGILE-ITEA tutkimusohjelmassa. Tutkimusohjelmaan kuului 68 pilottiprojektia ja tutkimusohjelmaan osallistui yhteensä 2000 ohjelmistosuunnittelijaa ja insinööriä. 78 % hankkeista päätyi tulokseen, joka oli erinomainen tai hyvä. Suomalaisissa yrityksissä järjestetyt kärkihankkeet johtivat kustannus- ja aikataulusäästöihin, jotka olivat 70 % luokkaa. (Abrahamsson ja Salo, 2007)

Se kuinka kauan menee ketterien menetelmien omaksumiseen organisaatiossa, riippuu tavoiteltujen muutosten laajuudesta ja organisaation koosta. Yleisesti voidaan kuitenkin puhua 2 - 3 vuodesta. Muutoksia kuitenkin tapahtuu välittömästi. Tämä johtuu konkreettisista muutoksista, jotka tapahtuvat sovelluskäytäntöjen tasolla. Kehittäjäyhteisöt ovatkin olleet kiinnostuneita ketterän ohjelmistokehityksen menetelmistään osaltaan siksi, että konkreettiset muutokset työhön ovat välittömiä. (Abrahamsson ja Salo, 2007)

Riittääkö pelkästään se, että ketteröitetään ohjelmistotiimien sisäinen toiminta ja ohjelmistotuotantomenetelmät? Ketterien menetelmien käyttöönottovaiheessa yrityksien tekemät havainnot kuitenkin puhuvat sen puolesta, että tarvitaan myös muita toimenpiteitä jos halutaan saada kaikki ketteryyden tuomat edut esille. Jotta voitaisiin tukea ketterien ohjelmistotiimien työtä ja poistaa esteitä huippusuoritusten tieltä tulee myös ohjelmistotuotannon sidosryhmien muuttaa toimintaansa, jotta se tukisi ketterää ohjelmistokehitystä. Jos tukiorganisaatio toimii eri toiminta-ajatuksella ja perinteisellä rytmillä, hidastaa se ketterämmänkin ohjelmistotiimin toimintaa. (Abrahamsson ja Salo, 2007)

Ajatustapaa on pystyttävä muuttamaan yrityksessä kokonaisvaltaisesti siirryttäessä kohti ketteryyttä. Tämän lisäksi on harkittava uudelleen toimintatapoja kaikilla yrityksen tasoilla. Myöskään kaikki ketteryyden asteet ja ketterät menetelmät eivät sovi kaikkiin yrityksiin. Lisäksi valitut menetelmät ja toimintatavat pitäisi saada toimimaan perinteisempien menetelmien kanssa. Jokaiselle yritykselle tulisikin löytää yrityksen tarpeisiin sopiva ketteryyden aste sekä toimintamalli. Tällöin voidaan käyttää hybridimallia, jossa yhdistetään perinteisen ja ketterän mallin parhaita puolia. Tarvitaankin suunnitelmallisuutta siirryttäessä kohti ketteriä menetelmiä. Ennen yrityksen toimintatapojen ketteröittämistä tulisikin löytää tavoitteet ja lisäedut, joita ketterien menetelmien avulla tavoitetaan. Selkeä ja huolellinen suunnittelu ovat avainasemassa toimintatapoja muutettaessa. Onnistumisen kannalta on myös paljon merkitystä sillä, että eri osapuolet ovat sitoutuneita ja osallistuvat projektiin. Osapuolilla tarkoitetaan esimerkiksi kehittäjiä ja johtoporrasta. Yleinen ja vähiten riskejä sisältävä tapa kokeilla ketteriä menetelmiä on keskittyä sovelluskehitystason toimintoihin. Jos käytetään pilottihankkeita, tulisi pilottihankkeen kattaa toimintamallin muutos koko organisaation laajuisesti. Tämä mahdollistaa kokonaisvaltaisen vaikutusten ja muutostarpeiden näkemisen, jotka johtuvat ketteröitymisestä. (Abrahamsson ja Salo, 2007)

Hartemo (2011) kirjoittaa blogikirjoituksessaan, ettei kukaan kehtaa kertoa vetävänsä projekteja vesiputousmallilla. Ovathan ketterät menetelmät tätä päivää. Enää suunnittelu ja toteutus eivät etene vaiheittain vaan käytetään limittäistä suunnittelua. Tämän lisäksi projekti rakennetaan iteratiivisesti pala palalta. Keskeisiä arvoja tekemisessä ovat muutosalttius, joustavuus ja läheinen yhteistyö asiakkaan sekä toimiston välillä. Ohjelmistokehitykseen ei ole välttämättä pakko uhrata tänä päivänä miljoonia, joka mahdollistaa palvelujen rakentamisen kokeellisesti, nopeasti ja kevyesti. Käyttäjälähtöisyys kulkee edellä mainittujen edellä. Sovelluskaupat ovat mahdollistaneet tuotosten myymisen ympäri maailmaa ja ohjelmoinnin työkalut voi saada lähes ilmaiseksi. Tämän vuoksi suuri pääoma ei ole välttämätöntä ohjelmistotuotannolle.

Kun on päästy ketteriin menetelmiin, tulee myös esiin palvelumuotoilu. Keskeisenä toimintatapana palvelumuotoilussa on yhdessä tekeminen. Tällöin eri osapuolet eivät enää toimi erillään ilman todellista vuorovaikutusta. Palvelukonseptien kautta voidaan rakentaa visualisointeja ja prototyyppejä, joita on mahdollista käydä läpi yhdessä loppukäyttäjien ja projektitiimin kanssa. Tarvitaan siis avoimempia toimintamalleja asiakkaiden ja toimistojen välille. Jos halutaan suunnitella liiketoimintalähtöisiä palveluja, tulee tuntea yrityksen liiketoiminta ja tavoitteet. (Hartemo, 2011)

Taustoittavat ja suunnittelua ohjaavat tutkimukset, prototyypit, workshopit ja pilotit kuuluvat tämän päivän työskentelytapoihin. Piloteista voidaan puhua myös alphoina ja betoina. Ideaalissa tilanteessa pilotti avataan loppukäyttäjille, jolloin loppukäyttäjillä on mahdollisuus antaa arvionsa pilotista ja kehittää sitä eteenpäin. Oma toimipiste saattaa tänä päivänä puuttua kokonaan tai se voi sijaita osan ajasta kumppanin luona. Tärkeintä on kuitenkin, että ollaan läsnä kuluttajien keskuudessa sekä yhteistyössä, jota tapahtuu yritysten välillä. Läsnäolon tulee olla aitoa ja aktiivista. Yhteistyön kannalta on oleellista, että oikeat henkilöt sijoitetaan oikeisiin paikkoihin. Jos osaaminen on omissa joukoissa laajaa, löytyy myös todennäköisemmin tekijä eri tehtäviin. Toisena ratkaisuna voidaan hyödyntää verkostoitumista ja löytää tekijä verkostojen kautta. (Hartemo, 2011)

Kiinteät tiimit ja ansaittu asema eivät välttämättä ole enää avainasemassa. Tekijöiltä vaaditaan liikkuvuutta ja kykyä väistyä sivuun tilanteessa, jossa tekijän osaaminen ei kohtaa tarvitun osaamisen kanssa. Projektien kautta saatua tietoa ei myöskään enää tulisi pantata vaan tieto tulisi olla tulevien tekijöiden käytettävissä. Muutos ei tarkoita, että se tulisi ottaa henkilökohtaisesti. Vaikka tänä päivänä olisi mestari, voi huomenna olla oppipoika. (Hartemo, 2011)

Hartemon (2011) blogikirjoituksesta näkyykin selvästi, että tänä päivänä yhteistyö on saanut entistä suuremman merkityksen. Esimerkiksi betatestauksen avulla voidaan saada sovelluksen tulevilta käyttäjiltä tietoa ja ajatuksia sovelluksesta. Näiden tietojen pohjalta voidaan lähteä rakentamaan sovellusta, joka entistä enemmän vastaa loppukäyttäjien tarpeita. Näen myös hyvänä näkökulmana sen, että asiakkaan liiketoiminta ja tavoitteet tulee tuntea, jotta sovellus voidaan räätälöidä tukemaan näitä. Jos liiketoimintaa ei tunneta, kuinka voidaan tehdä liiketoimintaa tukevia sovelluksia? Tiedon panttaamiseen tulee myös suhtautua niin, ettei se hyödytä liiketoimintaa. Jos vanhojen projektien kautta on saatu tietoa, jota voitaisiin hyödyntää myös tulevissa projekteissa, miksi tietoa tulisi pantata? Tämä voi johtaa siihen, että samaa asiaa joudutaan selvittämään kahdesti, jolloin käytetään aikaa ja resursseja tehottomasti.

Mielipidekirjoituksessa ”Kun ketteryys korvasi kankeuden” Syrjänen (2008) puhuu myös ketteryyden puolesta. Pienet laaja-alaiset toteutustiimit mahdollistavat nopean ja tehokkaan toteutuksen. It-maailmassa on tapahtumassa murros, johon vaikuttaa muutokset asiakkaiden digitaalisessa liiketoimintaympäristössä. Vaatimuksiin kuuluvat Internetin aikakaudella käyttäjäläheisyys ja nopeus. Jotta voitaisiin saavuttaa strateginen ketteryys, tulee uudet ja parannetut palvelut pystyä tuomaan markkinoille nopealla syklillä, joka tarkoittaa 2 - 3 kuukautta. Se, että ennen palveluiden suunnittelun lähtökohdat olivat tekniset, ei toimi enää tänä päivänä. Nyt suunnittelun ohjaavana tekijänä toimii käyttäjä, jotta voidaan toteuttaa käyttäjälähtöisiä palveluita. Ja koska palvelut tulee pystyä toteuttamaan nopealla tahdilla, tulee tekijöiden hallitta uudet teknologiat.

Muutaman hengen osaavat tiimit ovat syrjäyttäneet suuret tiimit, jotka koostuvat kapean roolin omaavista henkilöistä. Tekninen toteutus ja ongelmien ratkaisu ovat yksi kokonaisuus eivätkä irrallaan toisistaan. Laaja-alainen osaaminen ja hyvä ymmärrys tehtävästä asiasta ovat vaatimuksia tällaisessa tiimissä työskentelylle. Tällä tavoin voidaan vähentää koordinointia, joka on tuottamatonta. Pieni tiimi tuo mukanaan positiivisia haasteita ja vaikutusmahdollisuuden projektin onnistumiseen. Myös asiakkaat osaavat vaatia ketteryyttä, koska myös asiakkaat ovat joutuneet vaihtamaan kankean ja kiveen hakatun strategiansa ketteryyteen menestymisen vuoksi. (Syrjänen, 2008)

3.2 Versionhallintajärjestelmä

3.2.1 Tietoa versionhallintajärjestelmästä

Tämän projektin osalta voidaan puhua versionhallinnasta koodinhallinnan näkökulmasta. Tällöin versionhallinnalla tarkoitetaan joukkoa erilaisia käytäntöjä ja tekniikoita, joiden avulla on mahdollista seurata projektiin kuuluvien tiedostojen muutoksia. Erityisesti seuranta kohdistuu lähdekoodiin, Internet-sivuihin ja dokumentaatioon. Miksi sitten versionhallinta on niin yleispätevä? Versionhallinta auttaa sisäisessä kommunikaatiossa ja virheiden sekä julkaisujen hallinnassa. Lisäksi versionhallinnan avulla voidaan vakauttaa koodia sekä mahdollistaa se, että tietyt kehittäjät nimeävät ja valtuuttavat muutokset. Versionhallinnan avulla voidaan nämä eri osa-alueet sovittaa yhteen. (Tonu, 2012)

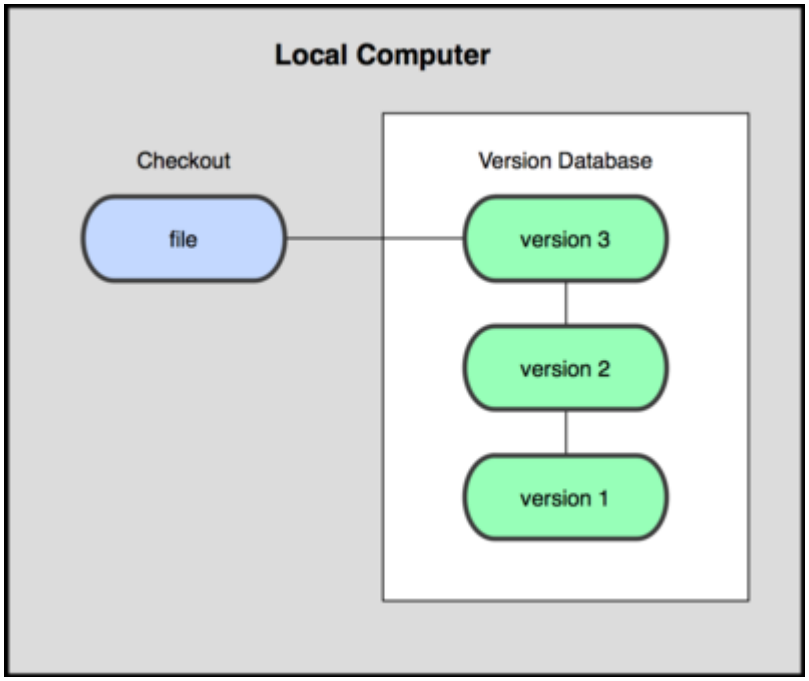
Muutoksenhallinta on yksi versionhallinnan päätoiminnallisuuksista. Muutoksenhallinnan avulla voidaan tunnistaa ja kommentoida jokainen muutos projektitiedostoissa. Tämän lisäksi muutoksesta jää metatiedot, jotka sisältävät esimerkiksi aikaleiman ja muutoksen tekijän. Tämän jälkeen tiedot ovat saatavilla kysyttäessä. Tällöin muutoksenhallinta on kommunikaation muoto, jossa muutos on tiedon yksikkö. Muuttuneita tiedostoja voidaan verrata keskenään ja tarvittaessa palauttaa. (Tonu, 2012)

Versionhallintajärjestelmän yhtenä vaatimuksena onkin, että se pystyy pitämään yllä historiaa tiedostojen muutoksista. Historiatietojen lisäksi tiedostojen vanhojen versioiden tulee tarvittaessa olla palautettavissa, jotta ominaisuudesta olisi hyötyä. Vaikka tiedostoa ei tällä hetkellä tarvittaisikaan, tulee muistaa, että siihen voi joutua myöhemmin viittaamaan. (Belcham ja Baley, 2010, 35)

Toisena vaatimuksena voidaan versionhallintajärjestelmälle asettaa liitettävyys. Tämä saavutetaan käyttämällä versionhallintajärjestelmää koodin lähteenä, johon kehittäjät ottavat yhteyden. Versionhallintajärjestelmä toimii tällöin yhteyspisteenä ja koodin viimeisimmän version tulee löytyä versionhallintajärjestelmän kautta. (Belcham ja Baley, 2010, 35)

Kolmantena vaatimuksena voidaan versionhallintajärjestelmälle asettaa mahdollisuus päivittää joukko erillisiä tiedostoja samalla kertaa ja päivityksen koskea kaikkia tiedostoja. Muussa tapauksessa päivitystä ei suoriteta mihinkään joukossa olevaan tiedostoon. Tällä tavoin pyritään välttämään tilanne, jossa koodi jäisi epävakaaseen tilaan päivityksen jälkeen. (Belcham ja Baley, 2010, 35)

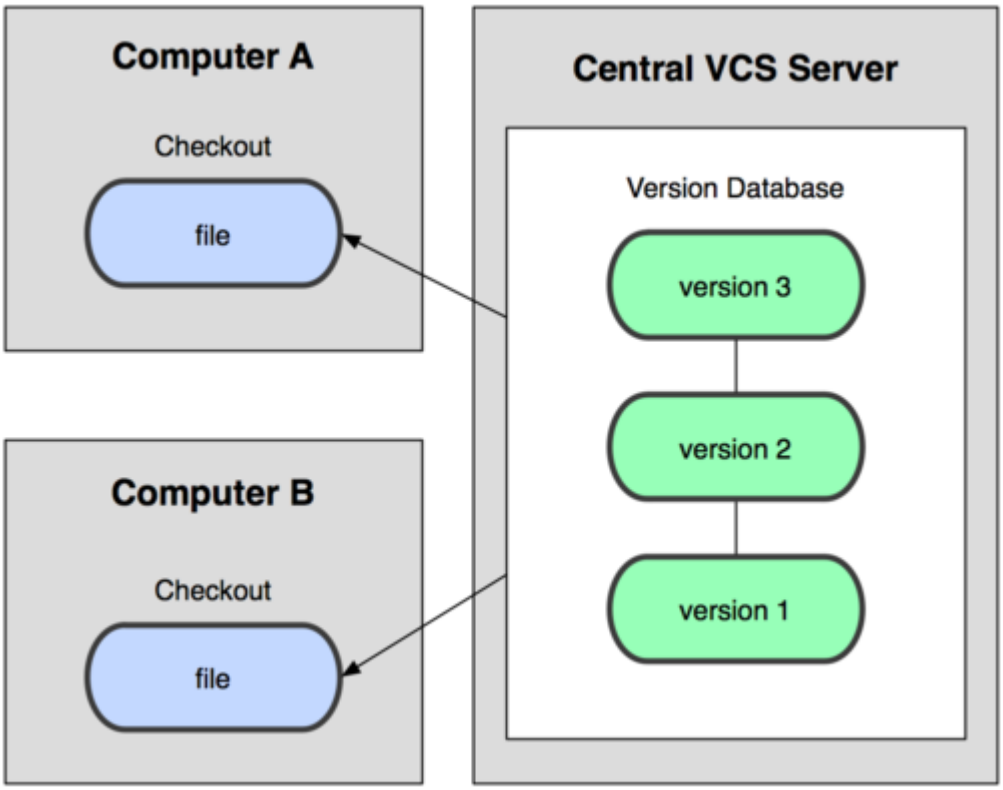
Puhuttaessa versionhallintajärjestelmistä voidaan versionhallintajärjestelmät jakaa ainakin paikallisiin, keskitettyihin ja jaettuihin versionhallintajärjestelmiin. Tässä luvussa kerrotaan paikallisesta ja keskitetystä versionhallintajärjestelmästä. Jaetusta versionhallintajärjestelmästä kerrotaan Gitistä kertovan luvun yhteydessä. Paikallisessa versionhallintajärjestelmässä (kuvio 2) tiedostoihin tapahtuvat muutokset tallennetaan yksinkertaiseen tietokantaan. Tämä tapahtuu paikallisesti käyttäjän tietokoneella, jolloin esimerkiksi tietokoneen hajotessa on riski tietojen katoamiselle. Varsinkin jos varmuuskopioista ei ole huolehdittu. Paikallisiin versionhallintajärjestelmiin kuuluu suosittu työkalu rcs, joka toimitetaan muun muassa Mac OS X-käyttöjärjestelmän mukana. Rcs pitää yllä tietoja tiedostojen välisistä muutoksista muutostiedostojen (patch) avulla. Jos tiedosto halutaan palauttaa tiettyyn ajankohtaan, voidaan palautus tehdä käyttämällä muutostiedostoja. (Chacon, 2009a)



Kuvio 2. Paikallinen versionhallintajärjestelmä (Chacon, 2009a).

Projektin kannalta paikallinen versionhallinta ei olisi täyttänyt projektin vaatimuksia. Yhtenä suurena puutteena paikallisessa versionhallinnassa on, että tiedostoihin pääsee käsiksi vain siltä tietokoneelta, jolla projektia työstetään. Projektin aikana toimeksiantajalla tuli olla pääsy tiedostoihin ja tarvittaessa tehdä muutoksia, jolloin paikallinen versionhallintajärjestelmä ei olisi sopinut projektin tarpeisiin.

Keskitetty versionhallintajärjestelmä (kuvio 3) mahdollistaa yhteistyön samassa projektissa eri kehittäjien kesken. Keskitetyssä versionhallintajärjestelmässä käytetään palvelinta, joka sisältää keskitetysti versioidut tiedostot ja johon kehittäjät ottavat yhteyttä tarkastaakseen tiedostot. Etuna voidaan mainita, että kaikki tietävät tietyllä tasolla mitä kukakin projektissa tekee. Lisäksi ylläpitäjillä on mahdollisuus kontrolloida mitä kukakin pystyy tekemään. Jos kuitenkin esimerkiksi palvelin kaatuu, johtaa tämä tilanteeseen, jossa yhteistyö katkeaa eikä muutoksia pystytä tallentamaan. Laiterikon, esimerkiksi kovalevyn hajoamisen yhteydessä on myös riski projektitiedostojen ja historiatietojen katoamiselle jos varmuuskopioista ei ole huolehdittu. Tällöin ainoa palautettavissa oleva tieto on käyttäjien koneilla olevat yksittäiset tilannevedokset (snapshots). (Chacon, 2009a)



Kuvio 3. Keskitetty versionhallintajärjestelmä (Chacon, 2009a).

Keskitetty versionhallinta olisi jo tarjonnut niitä ominaisuuksia, joita projektin läpivieminen vaatii. Kuitenkin yhdessä vaiheessa projektia palvelimeen ei saanut yhteyttä. Tämä seikka oli jo tiedossa, mutta jos projektissa olisi käytetty keskitettyä versionhallintaa, en olisi pystynyt tallentamaan muutoksia versionhallintaan. Jaettua versionhallintaa käytettäessä pystyin jatkamaan projektin työstämistä normaalisti ja tallentamaan tekemäni muutokset versionhallintaan. Kun palvelimeen taas saatiin yhteyttä, vein vain tekemäni muutokset palvelimelle. Keskitetty versionhallinta ei olisi mahdollistanut tätä. (Chacon, 2009a)

3.2.2 Versionhallintajärjestelmä projektin näkökulmasta

Versionhallintaa ei tämän projektin aikana käytetty kuin lähinnä sovelluksen koodiin tapahtuvien muutosten seurantaan, joten en tässä opinnäytetyössä lähde arvioimaan versionhallinnan soveltuvuutta muihin tarkoituksiin. Mielestäni versionhallintaa voidaan kuitenkin hyödyntää esimerkiksi dokumentaation muutosten seurantaan. Tässä projektissa versionhallinta kuitenkin toimi myös osaltaan sisäisen kommunikaation välineenä. Esimerkiksi toimeksiantaja pystyi seuraamaan sovellukseen tapahtuvia muutoksia versionhallinnan kautta. Tämä kuitenkin vaatii tilaajalta tietoteknistä osaamista, jotta muutosten seuraaminen onnistuu. On myös hyvä huomioida, että jos muutoksia haluaa arvioida kooditasolla, tulee tilaajalla olla sovelluskehityspuolen osaamista. Ovatko nämä sitten välttämättömiä taitoja, jotta versionhallintaa voitaisiin hyödyntää tilaajan puolelta? Versionhallinnan kautta tilaaja voi myös seurata milloin muutoksia projektiin on tehty ja mitä muutokset koskevat. Tämä vaatii kuitenkin, että muutoksien yhteyteen laaditaan selkeä viesti siitä, mitä muutokset koskevat.

Metatiedot mahdollistivat projektissa sen, että muutoksen tehneen henkilön tiedot ja muutoksen ajankohta jäivät versionhallinnan tietoon. Opinnäytetyön aikana toimin itse ainoana henkilönä, joka teki muutoksia projektiin. Kuitenkin tulevaisuudessa on mahdollista, että sovelluksen parissa työskentelee useampi henkilö. Tällöin näen tarpeellisena, että muutoksen tehneen henkilön tiedot jäävät versionhallinnan historiaan. Tällöin on myös mahdollista, että versionhallinta toimii osana kehittäjien välistä kommunikaatiota. Jos kehittäjien sekä tätä kautta muutosten määrä kasvaa, voi kasvaa tarve muutosten vertailulle ja esimerkiksi tietojen palauttamiselle. Versionhallinnan hyötyjä voidaan siis saavuttaa pienemmässäkin projektissa ja toisaalta versionhallinnan käytöstä ei omasta mielestäni ole haittaakaan. Jos versionhallinta otetaan käyttöön jo projektin alussa, voi projektin pariin liittyä myös uusia henkilöitä, jolloin keskitetty tietolähde projektille on olemassa.

Projektin aikana versionhallintajärjestelmä toimi myös lähteenä sovelluksen lähdekoodeille. Tämä mahdollisti sen, että projektin aikana kehittäjä- ja tilaajapuoli käyttivät samaa lähdettä sovelluksen lähdekoodien sekä oheistiedostojen hakemiseen. Tämä mahdollisti, että kummallakin osapuolella oli käytössään samat tiedot. Esimerkiksi testauksen kannalta saavutettiin se hyöty, että kummallakin osapuolella oli käytössään sama lähdekoodi. Tämä ei kuitenkaan sulje pois testeissä ilmenneitä eroavaisuuksia, jotka johtuivat testiympäristöön liittyvistä seikoista. Testauksen ja sovelluksen toimivuuden kannalta oli tärkeää, että muutoksia viedessä päivitys koskee kaikkia niitä tiedostoja, joihin on tehty muutoksia. Koska esimerkiksi luokkien välillä voi olla riippuvaisuuksia, on tärkeää, että muutokset viedään kumpaankin luokkaan. Muussa tapauksessa esimerkiksi toiseen luokkaan tehdyt muutokset rampauttavat toisen luokan toiminnallisuuden.

Projektin näkökulmasta katsottuna en näe syytä sille, miksei versionhallintaa tulisi käyttää varsinkin osana sovelluskehitystä. Omalta osaltaan tämä johtuu siitä, ettei projektin aikana ilmennyt sellaisia negatiivisia seikkoja, jonka vuoksi versionhallintaa ei tulisi käyttää. Ainoa ongelma, joka projektin aikana ilmeni, oli tilanne, jossa palvelimeen ei saatu yhteyttä. Tällöin muutoksien vieminen ulkoiselle palvelimelle kaikkien nähtäväksi ei onnistunut. Tämä ei kuitenkaan estänyt projektin työstämistä eteenpäin ja muutokset pystyi viemään myöhemmin palvelimelle, kun yhteys palvelimeen saatiin taas toimimaan. Tämä arvio on kuitenkin tehty sellaisen ympäristön perusteella, jossa muutokset sovellukseen teki yksi henkilö. Jos muutoksia tekee useampi henkilö, on mahdollista, että tilanne on toisenlainen.

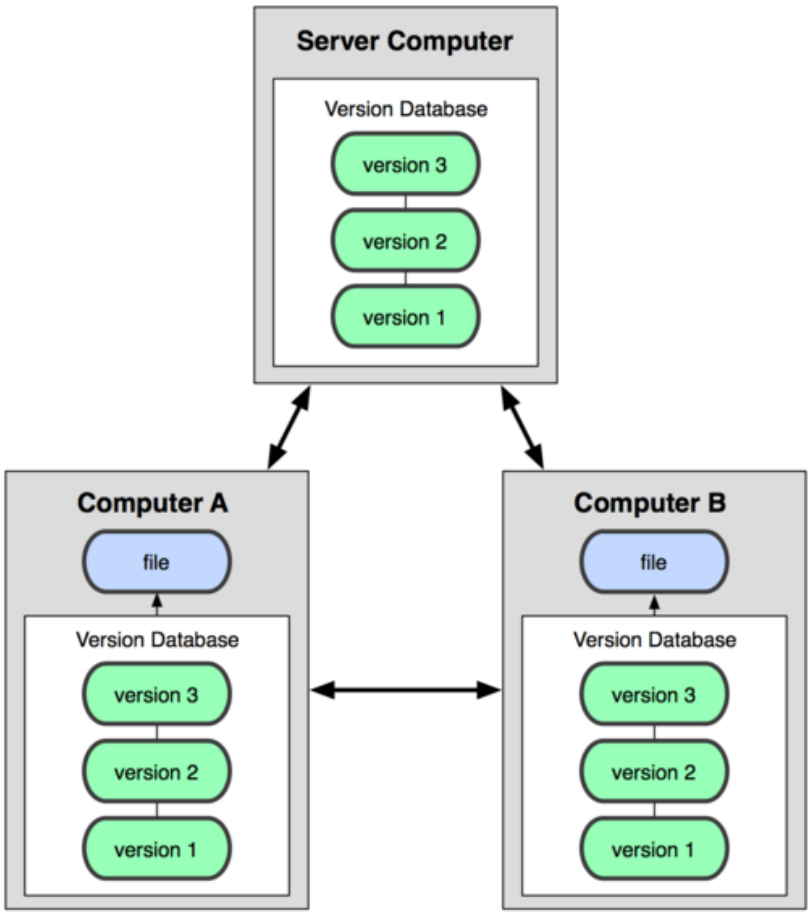
3.2.3 Git

Gitin kehitys alkoi tarpeesta rakentaa versionhallintajärjestelmä Linuxin ytimen ylläpitämistä varten. Vuosina 1991 - 2002 muutokset tehtiin sovellukseen käyttämällä korjaustiedostoja (patch) ja arkistoituja tiedostoja (archived files). Suljetun lähdekoodin versionhallintajärjestelmä BitKeeper otettiin käyttöön vuonna 2002. Kuitenkin vuonna 2005 välit Bitkeepiriä ylläpitäneen yrityksen ja Linuxin ydintä ylläpitäneen yhteisön välillä hajosivat. Samalla myös versionhallintajärjestelmän tila muuttui niin, ettei se ollut enää ilmainen. Tämä toimi myös kannustimena sille, että yhteisö ja Linux Torvalds lähtivät kehittämään omaa työkalua. Uuden työkalun pohjana käytettiin niitä asioita, joita he olivat oppineet BitKeeperin käytön kautta. Tavoitteita, joita uudelle työkalulle asetettiin, olivat muun muassa yksinkertainen suunnittelu, kyky käsitellä suuria projekteja, nopeus, täysi jaettavuus ja tuki ei-lineaariselle kehitykselle. Ei-lineaarisella kehityksellä haettiin mahdollisuutta useille rinnakkaisille haaroille projektissa. (Chacon, 2009b)

Merkittävin ero puhuttaessa Gitin ja muiden versionhallintajärjestelmien eroista on se kuinka Git näkee datan. Yleisesti muut versionhallintajärjestelmät käyttävät listoja, jotka perustuvat tiedostojen muutoksiin. Näitä listoja käytetään tällöin tietojen taltioimiseen. Tällöin tieto nähdään joukkona tiedostoja ja niiden muutoksia. Gitissä data nähdään joukkona pienen tiedostojärjestelmän tilannevedoksia. Kun projektin tilanne tallennetaan, Git ottaa kuvan tiedostojen sisällöstä ja tallentaa samalla viittauksen tähän tilannevedokseen. Yksi tehokkuuden kannalta oleellinen seikka on se, ettei tiedostoa tallenneta uudestaan jos siihen ei ole tehty muutoksia. Sen sijaan Git luo linkin edelliseen identtiseen tiedostoon, joka löytyy järjestelmästä. (Chacon, 2009c)

Monissa toiminnoissa ei Gittiä käytettäessä tarvita tietoa toiselta verkon tietokoneelta, vaan yleensä tieto löytyy oman tietokoneen kovalevyn tiedostoista ja lähteistä. Kun esimerkiksi koko projektin historia on saatavilla suoraan omalta tietokoneelta, voidaan toimintoja suorittaa nopeasti. Tällöin projektin historiaa voidaan selata käyttämällä paikallista tietokantaa eikä ulkopuoliseen tietokantaan tarvitse ottaa yhteyttä tai jos halutaan verrata tiedoston nykyisen ja vanhemmat version eroja, suorittaa Git laskelman eroavaisuuksista paikallisesti. Milloin tästä toiminnallisuudesta on hyötyä? On olemassa tilanteita, joissa esimerkiksi Internet-yhteys tai yrityksen suojattu yhteys ei toimi. Tällöin pystytään projektia muokkaamaan ja kun Internet-yhteys tai suojattu yhteys on saatavilla, voidaan muutokset siirtää palvelimelle. (Chacon, 2009c)

Git voidaan määritellä jaetuksi versionhallintajärjestelmäksi (distributed version control system). Tällöin Git ei pelkästään tarkasta tietolähteessä (esimerkiksi palvelin) olevien tiedostojen viimeisintä tilannevedosta vaan tietolähde peilataan myös sille koneelle, jolla tarkastus on tehty. Jos esimerkiksi palvelin rikkoutuu, voidaan tiedostot palauttaa tarkastuksen tehneen käyttäjän koneelta. Kuvio 4 kuvaa jaetun versionhallintajärjestelmän toimintaa. (Chacon, 2009a)



Kuvio 4. Jaettu versionhallintajärjestelmä (Chacon, 2009).

Git on ollut jatkuvasti läsnä projektin edetessä ja mahdollistanut työskentelyn etänä niin, että kaikki projektin osapuolet ovat pystyneet seuraamaan asiakashallinnan edistymistä. Käytännössä Gitin käyttö tapahtui niin, että muutokset tallennettiin ensin omalle koneelle ja tuotiin paikallisesti Gitin tietoon. Lopuksi muutokset vielä vietiin ulkopuoliselle palvelimelle, jolloin kahdessa paikassa oli ajantasainen versio projektista. Samalla myös pystyttiin kirjaamaan tehdyt muutokset, jotka liittyivät uuteen versioon. Tietoturvan kannalta tällä saavutettiin ainakin se, että projekti ei ollut pelkästään tallennettuna yhteen paikkaan. Jos projekti olisi esimerkiksi kadonnut omalta koneelta, olisi se ollut haettavissa palvelimelta. Tietenkin tämä vaatii, että muutokset viedään aina myös palvelimelle. Koska Git tunnistaa automaattisesti projektiin tehdyt muutokset, viedään palvelimelle vain ne tiedostot, joihin on tehty muutoksia. Jos projektin olisi joka kerta vienyt käsin palvelimelle, olisi jokainen päivitys pitänyt laittaa uuteen kansioon siinä tapauksessa, että halutaan säilyttää projektin historia. Tällöin joutuu tekemään ylimääräistä työtä, varsinkin kun päivityksiä voi olla paljon. Jos taas entinen projekti aina tuhottaisiin uuden tieltä, menetettäisiin mahdollisuus palata virheen sattuessa vanhempaan toimivaan versioon. Gittiä käytettäessä näitä seikkoja ei tarvitse pohtia vaan historia tallentuu automaattisesti ja tämä mahdollistaa esimerkiksi edelliseen pisteeseen palaamisen projektissa.

Käytettäessä Gittiä kaikelle sisällölle luodaan tarkistussumma ennen varastointia. Käytännössä tämä tarkoittaa, että kun tiedostoihin tai kansioihin tehdään muutoksia, on Git tietoinen asiasta. Jos esimerkiksi tiedosto korruptoituu tai tietoa katoaa siirron aikana, pystyy Git tunnistamaan tämän. Tarkistussummien luomisessa käytetään SHA-1-tiivistettä. 40-merkkisen merkkijonon laskemisessa käytetään joko tiedoston sisältöä tai hakemiston rakennetta. Merkkijonon sisältö koostuu numeroista ja kirjaimista väliltä a-f. (Chacon, 2009c)

Gitissä tehdyt toimenpiteet pääsääntöisesti vain lisäävät tietoa Gitin tietokantaan. Tämän vuoksi peruuttamattomien muutoksien tekeminen tai tiedon poistaminen on vaikeaa. Tiedon menettäminen on kuitenkin mahdollista, jos muutosta ei ole vahvistettu (commit). Tämä ei kuitenkaan ole yksin Gitin ongelma vaan koskee myös muitakin versionhallintajärjestelmiä. Tietojen menettämisen riskiä voidaan vielä vähentää säännöllisillä projektin viemisillä toiseen säilytyspaikkaan (repository). (Chacon, 2009c)

3.3 Web-palvelin, ohjelmointikielet ja tietokannan hallintajärjestelmä

3.3.1 Apache

Apache on vakaa avoimen lähdekoodin web-palvelin, joka on yhteensopiva HTTP:n kanssa. Sivustoista yli 60 % käyttää Apachea ja Apache-pohjaisten ratkaisujen käyttäjiin lukeutuvat esimerkiksi Yahoo, IBM ORACLE, Amazon ja Red Hat. Apachen on kehittänyt Apache Software Foundation. Apachen rakentamisessa on käytetty moduuliarkkitehtuuria, jolloin olemassa olevia moduuleita voidaan käyttää uudestaan. Tällöin vanhaa moduulia voidaan käyttää toisten moduulien, protokollien ja ohjelmien kehittämisessä. Moduulin kehittämisessä ei myöskään tarvitse käyttää yhtä tiettyä kieltä, vaan riittää, että ohjelmointikieli kuuluu Apachen tukemiin kieliin. Moduuliarkkitehtuurin ansiosta pystytään parantamaan skaalautuvuutta, joustavuutta ja sopeutumiskykyä. Virtuaalisen hostauksen ansiosta Apache-palvelimella voidaan ajaa yhden sivuston sijasta monia eri sivustoja yhtä aikaa. (Allen, Qian, Tao ja Fu, 2010, 436-437)

Tietoturva kuuluu tärkeänä osana Apachen ominaisuuksiin. Apache sisältää moduulit, joiden avulla saadaan tuki SSL- (Secure Socket Layer) ja TLS- (Transport Layer Security) protokollille. SSL- ja TLS-protokollien ansiosta voidaan asiakkaan ja web-palvelimen välinen liikenne suojata. Näiden protokollien lisäksi Apache sisältää moduulit tunnistautumista ja kulunvalvontaa (access control) varten. Käyttäjän tunnistamisessa käytetään yleisesti käyttäjätunnusta ja salasanaa, jolloin käyttäjätunnusta ja salasanaa verrataan tietokannassa oleviin. Kuormantasauksen (Load Balancing) ja pyyntöjen uudelleenohjauksella (Request Redirection) avulla tuetaan Apachen skaalautuvuutta. Web-palvelimen hallintatyökalujen avulla voidaan erilaisia palvelimia rakentaa, valvoa ja konfiguroida. (Allen ym. 2010, 436-437)

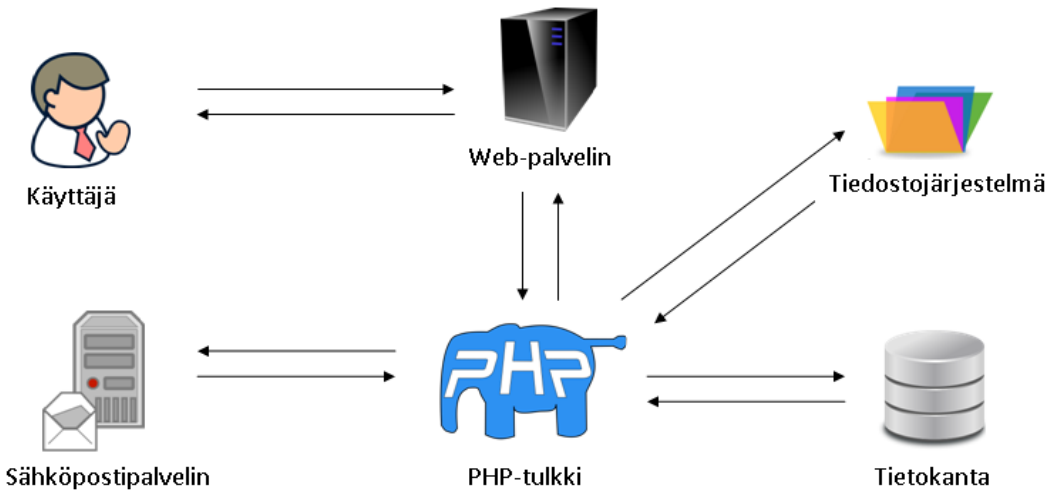
Apachea voidaan käyttää monilla eri alustoilla ja näihin alustoihin kuuluvat esimerkiksi Windows, Mac OS X, UNIX ja Linux. Moniajon ja säikeistystekniikan avulla on voitu saavuttaa korkea suorituskyky. Apache sisältää integroidun tuen PHP, HTTP, FTP, XML, SNMP, COBRA ja LDAP kehitysympäristöille. Tämän lisäksi Apache tukee PDF-tiedostojen rakentamista. Apachen tarjoamat rajapintamoduulit (SNMP, FTP) mahdollistavat, että näitä moduuleja voidaan käyttää apuna uusia protokollia rakennettaessa. (Allen ym. 2010, 436-437)

Monon .NET ja APS .NET rajapintojen sekä Apachen .Net rajapinnalle tehdyn ohjelmistorajapinnan ansiosta voidaan Apachen moduuleja kirjoittaa myös C# ja VB .NET kielillä. Tämän lisäksi .NET:illä tehtyjä web-ohjelmia on mahdollista ajaa Apache-palvelimella. Javan palvelinpuolen sovelluksia voidaan ajaa Tomcatin avulla. Apacheen on lisäksi integroituna tuki erilaisille skriptikielille, joihin kuuluvat Python, JavaScript, PHP ja Ruby. Tämän lisäksi Apache tukee tietokannoista MySQL-tietokantaa. (Allen ym. 2010, 436-437)

Apachen rooli projektissa oli mahdollistaa testiympäristö, jonka testituloksia voitiin pitää luotettavana. Koska asiakashallinta tulisi olemaan Apache web-palvelimella, oli myös testiympäristön oltava kyseisessä ympäristössä. Käytettäessä Apachea Ubuntu-käyttöjärjestelmässä onnistui asennus suoraan paketinhallinnan kautta.

3.3.2 PHP

PHP on palvelinpuolella suoritettava ohjelmointikieli, jolla rakennettu toiminnallisuus suoritetaan palvelimella ennen web-sivun lähettämistä käyttäjän selaimeen. Tämä eroaa esimerkiksi JavaScript-kielestä sillä, että Javascript suoritetaan selaimessa sivun lataamisen jälkeen. Yhteensopivuusongelmaa selaimen suhteen ei synny, sillä selaimelle lähetetään standardia HMTL-koodia, jonka selain tulkitsee. PHP mahdollistaa sivun sisältöön vaikuttamisen ennen kuin se lähetetään käyttäjän selaimelle. Tällöin esimerkiksi sivun sisältö voi näyttää erilaiselta ylläpitäjälle kuin käyttäjälle. Tätä voidaan pitää rajoituksena interaktiivisten sivujen suhteen. Koska PHP-koodi suoritetaan sivun latautumisen yhteydessä, tulee käyttäjän esimerkiksi lähettää lomake, jonka jälkeen latautuneella sivulla näytetään tulokset lomakkeen suhteen. Vastaavasti JavaScriptillä voidaan vaikuttaa sivun käyttäytymiseen sivun latautumisen jälkeen, jolloin esimerkiksi sisältöä ja ulkoasua voidaan muuttaa ilman sivun uudelleenlataamista. (Yank, 2012)



Kuvio 5 PHP:n toiminta.

Kuviosta 5 voidaan tarkastella PHP:n toimintaa käyttäjän saapuessa sivulle, jonka päätteenä on .php. Tällöin web-palvelin ohjaa pyynnön PHP-tulkille, joka suorittaa sivun prosessoinnin. PHP-tulkki ottaa tarvittaessa yhteyttä tiedostojärjestelmään, sähköpostipalvelimeen tai tietokantaan jos näistä tarvitaan tietoa näytettävälle sivulle. Kun PHP-tulkki on suorittanut prosessoinni loppuun, palauttaa tulkki sivun web-palvelimelle, josta se ohjataan käyttäjän selaimeen. (w3schools, 2012)

Käytännön esimerkkinä asiasta voidaan ottaa lomakkeen tietojen tarkastaminen. JavaScriptin avulla voidaan mahdollistaa virheilmoituksen ilmestyminen sivulle, jos lomakkeeseen syötetyn tiedon muoto on vääränlainen. Tällöin voitaisiin välttyä lomakkeen lähettämiseltä, jos muotovaatimukset eivät täyty. Tällä tavoin ei voida kuitenkaan huolehtia koko tarkastuksesta, sillä koska JavaScript suoritetaan sivun lataamisen jälkeen, voi käyttäjä esimerkiksi estää JavaScriptin suorittamisen selaimessa. Tällöin myös tietojen tarkastaminen menee pois päältä. Lopullinen tarkastus tulee suorittaa PHP:n avulla ja koska PHP-koodi suoritetaan palvelimen puolella, ei käyttäjä voi vaikuttaa sen suorittamiseen. Tarkastuksen jälkeen voitaisiin PHP:n avulla tulostaa virheilmoitus, jos syötteiden tarkastuksessa ilmenee ongelmakohtia.

Nykyinen PHP on seuraaja PHP/FI-nimiselle tuoteelle. PHP:n historia alkaa vuodesta 1994, jolloin Rasmus Lerdorf loi ensimmäisen version PHP:sta. Tätä versiota käytettiin seuraamaan Lendorfin Internetissä olevan ansioluettelon vierailuja. Versio sisälsi yksinkertaisen valikoiman CGI (Common Gateway Interface) ohjelmakoodia ja versiosta Lendorf käytti nimeä ”Personal Home Page Tools”. Tarve uusille toiminnallisuuksille johti PHP Toolsin uudelleenkirjoittamiseen. Uusi malli mm. mahdollisti integraation tietokannan kanssa ja yksinkertaisten dynaamisten web-sovellusten luomisen. Kesäkuussa vuonna 1995 PHP Toolsin lähdekoodi julkaistiin, mikä mahdollisti työkalun vapaan käytön kehittäjille ja samalla kannusti käyttäjiä parantamaan työkalua esimerkiksi korjausten muodossa. (The PHP Group, 2013)

Syyskuussa 1995 työkalun nimeksi vaihtui FI (Form Interpreter) samalla kun työkalua laajennettiin. Tämä toteutus sisälsi perustoiminnallisuuksia, joita löytyy nykyisestä PHP:sta. Näihin toiminnallisuuksiin lukeutuvat esimerkiksi HTML-kieleen sulautettu syntaksi, Perl-tyyliset muuttujat ja lomakkeen muuttujien automaattinen tulkkaus. Käytetty syntaksi muistutti Perliä, mutta oli yksinkertainen ja rajoittenuumpi. Jos koodia haluttiin kirjoittaa HTML-tiedostoon, oli kommentoinnissa käytettävä HTML-kommentteja. Lokakuussa 1995 julkaistiin täysin uudelleen kirjoitettu koodi, jonka myötä työkalu alkoi muuttua kohti ohjelmointikieltä. Kielen rakenne muistutti C-ohjelmointikieltä ja tämä helpotti kielen omaksumista kehittäjillä, jotka hallitsivat C:tä ja Perliä. (The PHP Group, 2013)

Seuraava suurempi askel kehityksessä nähtiin huhtikuussa vuonna 1996. Julkaisusta käytettiin nimeä PHP/FI ja julkaisun myötä PHP alkoi kehittyä omaksi ohjelmointikielekseen. Tällöin julkaisu sisälsi tuen muun muassa kekseille, käyttäjän omille funktioille ja tietokannoille. Tuettuihin tietokantoihin kuuluivat DBM, mSQL ja Postgres95. (The PHP Group, 2013)

Vuosina 1997 ja 1998 PHP/FI:n käyttäjäkunta koostui muutamasta tuhannesta käyttäjästä. Toukokuussa vuonna 1998 PHP/FI 60 000 domainin header sisälsi merkinnän PHP:sta. Tämä selvisi Netcraftin kyselystä. Tuohon aikaan 60 000 domainia vastasi yhtä prosenttia kaikista Internetin domaineista. Yhtenä PHP/FI:n kasvun esteenä oli tuolloin kehitystyö, josta huolehtivat yksittäiset käyttäjät. (The PHP Group, 2013)

PHP:n versio 3 julkaistiin helmikuussa vuonna 1999. Version 3 myötä myös FI-liite poistui nimestä ja nimeksi vakiintui PHP. Suurimpana vahvuutena PHP 3:n osalta oli laajennettavuus. Lisäksi käyttäjille tarjottiin kunnollinen rajapinta moniin tietokantoihin, protokolliin ja ohjelmistorajapintoihin. PHP 3:n ominaisuuksiin kuuluivat lisäksi tuki oliopohjaiseen ohjelmointiin ja tehokas sekä johdonmukainen syntaksi. PHP 4 julkaistiin toukokuussa vuonna 2000 ja sen ydin perustui Zend Engine-moottoriin. Php 4 paransi monimutkaisten sovellusten suorituskykyä ja koodipohjan (code base) modulaarisuutta. Lisäksi PHP 4 tuki HTTP-istuntoja, tulosteiden puskurointia (output buffering), turvallisia tapoja käyttäjän syötteiden käsittelyyn, uusia kielen rakenteita ja laajempaa kirjoa erilaisia palvelimia. Opinnäytetyön kirjoitushetkellä uusin versio PHP:sta oli PHP 5, joka julkaistiin heinäkuussa vuonna 2004. PHP 5 sisältää Zend Enginen version 2.0, joka sisältää muun muassa uuden objektimallin (object model) ja uusia ominaisuuksia. (The PHP Group, 2013)

3.3.3 HMTL

HTML muodostuu sanoista Hypertext Markup Language. HTML-kieltä käytetään kuvailemaan dokumentin sisällön yleistä rakennetta, ei niinkään sivuston ulkoasua. HTML mahdollistaa dokumenteissa olevien yleisten elementtien määrittämisen. Yleisiä dokumenteissa olevia elementtejä ovat esimerkiksi otsikot, listat ja kappaleet. HTML:n määritykset eivät yleisesti ottaen ota kantaa sivun ulkoasuun vaan tätä varten käytetään CSS-kieltä. HTML:n avulla osoitetaan eri elementtien käyttötarkoitus eli onko elementti esimerkiksi lista vai otsikko. Elementtien merkitsemisessä käytetään tageja, joiden sisään elementin sisältö laitetaan. Kun dokumenttia selataan esimerkiksi Internet-selaimella, muuntaa selain tagit ja niiden sisällön näytöllä esitettävään muotoon. Tällöin esimerkiksi otsikko voidaan erottaa muusta tekstistä isommalla fonttikoolla. (Colburn ja Lemay, 2010, 50-51)

HTML ei kuitenkaan C:n, Perlin ja Rubyn tavoin ole ohjelmointikieli vaan semanttinen kieli, jota käytetään tekstin merkitsemiseen. Lisäksi HTML:n avulla merkitään linkit, joiden avulla viitataan muihin dokumentteihin, esimerkiksi toiseen sivuun. Linkit mahdollistavat viittauksen muihin dokumentteihin riippumatta siitä, missä dokumentit sijaitsevat fyysisesti. (Aronson, 2010, 3-4)

Alkuperäisen HTML 2.0-standardin kehitti ja sen ylläpidosta vastasi W3C. HTML 2.0 oli standardin lisäksi joukko erilaisia tageja, joita jokaisen selaimen tuli tukea. Iso osa näistä tageista on käytössä vielä tänäkin päivänä. HTML:n versio 3.2 kehitettiin vuonna 1996 yhteistyössä W3C:n, Microsoftin, IBM:n, Novellin, Sun Microsystemsin, Netscape Communications Corporationin, SoftQuadin ja Spyglassin kanssa, HTML 3.2 toi mukanaan mm. Taulut, sovelmat (applets) ja mahdollisuuden kuvaa kiertävään tekstiin. Vuonna 1997 esiteltiin HTML 4.0, joka mahdollisti suunnittelijoille suuremman vallan ulkoasuun kuin HTML:n aikaisemmat versiot. Tätä voi pitää HTML 4.0:n suurimpana uudistuksena, vaikka HTML 4 sisälsi myös esimerkiksi kehykset (frame). (Colburn ja Lemay, 2010, 53-55)

Tänä päivänä Internet-selainten määrä on kasvussa. Suosituista selaimista voidaan mainita Microsoft Internet Explorel ja Mozilla Firefox. Lisäksi voidaan mainita Apple Safari, Google Chrome ja Opera. Uusia versioita selaimista kehitetään jatkuvasti, mutta sen sijaan, että selaimiin kehitettäisiin omia epästandardeja ominaisuuksia, kehitetään selaimia tukemaan Internetin standardeja. Sen sijaan Microsoftin ja Netscapen aikakaudella kävivät molemmat kamppailua siitä, kuka julkaisee hienompia ominaisuuksia HTML:ään, jolloin standardointi ei kestänyt perässä. (Colburn ja Lemay, 2010, 54)

3.3.4 CSS

CSS-kieltä käytetään merkkauskielellä kirjoitettujen dokumenttien ulkoasun määrittämiseen ja kuvaamiseen. Näihin merkkauskieliin lukeutuu muun muassa Web-sivujen rakentamisessa käytetty HMTL-kieli. CSS on lyhenne sanoille "Cascading Style Sheets". CSS mahdollistaa sivuston visuaalisen ilmeeseen vaikuttamisen. CSS:n avulla voidaan esimerkiksi määrittää sivun taustakuva, tekstin väri ja fontti sekä sarakkeiden koko. Nämä ovat vain pieni osa isosta joukosta erilaisia määrityksiä. CSS mahdollistaa samojen määritysten käyttämisen usealla sivulla, jolloin kaikille sivuille voidaan luoda yhtenäinen tyyli. Samalla myös tehdyt muutokset periytyvät kaikille sivuille, joilla CSS-määritykset ovat käytössä. (Pouncey ja York, 2010, 3)

Yleisin käyttötarkoitus CSS-kielelle löytyykin web-sivujen tyylien määrittämisessä. Web-sivuja kehitettäessä voidaan CSS-kieltä käyttää yhdessä HTML- tai XHTML- ja JavaScript-kielten kanssa. Yhdistettynä sisällön kuvaamiseen tarkoitettuihin HTML- ja XHTML-kieliin sekä interaktiivisuutta lisäävään JavaScript-kieleen, muodostuu CSS tehokkaaksi työkaluksi. (Pouncey ja York, 2010, 3)

Internetin alkuaikoina tehtiin yhdeksän ehdotusta kielestä, jolla voitaisiin määrittää dokumentin tyyli. Tällä tavoin voitaisiin erotella dokumentin tyyli ja sisältö toisistaan. Ehdotukset tehtiin World Wide Web Consortlum -nimiselle standardointiorganisaatiolle. Yritys tunnetaan yleisemmin nimestä W3C. Håkon Wium Lie jätti ehdotuksen Cascading HTML Style Sheet -kielestä vuonna 1994. Tällöin hän työskenteli CERN:issä Internetin keksineiden Tim Berners Leen ja Robert Cailliaun kanssa. Nykyisin Håkon Wium työskentelee Opera Softwaren teknologiajohtajana. CHSS muutettiin kuitenkin CSS-nimeen, sillä CSS-kieltä voidaan käyttää muuallakin kuin HTML-dokumenteissa. Ensimmäinen CSS:n versio julkaistiin vuonna 1996. (Pouncey ja York, 2010, 3)

CSS:n tavoitteena oli olla joustava ja tarjota kehittäjillä tapa sivujen tyylin määrittämiseen. Tämän lisäksi yhtäläinen mahdollisuus haluttiin tarjota käyttäjille. Priorisoinnin ansiosta käyttäjälle jätettiin ylin valta sivun ulkoasun tyylien suhteen. Vaikka kehittäjä tarjoaa sivulleen tyylin, voi käyttäjä vaikuttaa sivun ulkoasuun. Ennen CSS-tyylimäärittelyjä kehittäjät eivät kuitenkaan olleet pystyneet vaikuttamaan mitenkään sivujen tyyliin, vaan valta tyylien suhteen oli jätetty käyttäjille. Kehittäjä pystyi ainoastaan merkitsemään esimerkiksi otsikot ja tekstin. Käyttäjä määritteli miltä ne näyttivät. (Meyer, 2012)

Ensimmäisen version jälkeen on julkaistu kolme CSS-määrittelyä. Julkaisuista on huolehtinut W3C-organisaatio. CSS versiosta 2 tuli suositus vuonna 1998. CSS:n versio 2.1 julkaistiin myöhemmin ja se korjasi joitakin virheitä, joita versiossa 2 oli ilmennyt. CSS 3 on tällä hetkellä kehitystyön alla ja ehdokkaana seuraavaksi suositukseksi. (Pouncey ja York, 2010, 4)

Niin kuin aikaisemmin historiasta selviää, on CSS syntynyt tarpeesta erotella tyyli ja sisältö toisistaan. CSS-määreet voidaan kirjoittaa erillään HTML-sisällöstä, joka mahdollistaa samojen määritysten käyttämisen usealla sivulla. Tällä tavoin voidaan tehdä suunnittelusta, ylläpidosta ja tuotannosta helpompaa. Jos tyylimääritykset tehtäisiin suoraan HTML:n sekaan, jouduttaisiin tyylimääritykset päivittämään aina kaikille sivuille erikseen. Vastaavasti tehtäessä uutta sivua, jouduttaisiin määritykset lisäämään siihen erikseen. (Pouncey ja York, 2010, 4)

CSS mahdollistaa saman dokumentin optimoimisen erilaisille laitteille. Esimerkiksi sivustolle voidaan tehdä oma tyyli kännyköitä ja taulutietokoneita varten. Selaimien hyödyntämän väimuistin ansiosta ladataan CSS-tiedosto tai web-sivu selaimen välimuistiin, jolloin niiden käyttö nopeutuu huomattavasti. Uusi versio tiedostosta ladataan ainoastaan jos siihen on tehty muutoksia. Tällöin vähennetään esimerkiksi sivustolle syntyvää kuormaa. (Pouncey ja York, 2010, 4)

Tämän lisäksi websivujen käyttäjillä on mahdollisuus laatia omia tyylitiedostoja. Tällöin voi käyttäjä esimerkiksi laatia tyylitiedoston, joka muuttaa sivuston kontrastin korkeaksi. Tällöin parannetaan sivuston luettavuutta. (Pouncey ja York, 2010, 4)

3.3.5 MySQL

MySQL relaatiotietokannan historia ulottuu 15 vuoden päähän, jolloin se syntyi TcX DataKonsult AB nimisen yrityksen sisäisessä projektissa. Ensimmäinen julkinen julkaisu projektin osalta saatiin vuoden 1996 lopulla. Ohjelmiston suosion vuoksi vuonna 2001 sen ympärille perustettiin oma yritys MySQL AB. Uusi yritys tarjosi MySQL-tietokantaan räätälöityä palvelua sekä MySQL-tuotteita. Alun jälkeen yritys kasvoi suurin harppauksin ja perusti konttoreita useisiin maihin. Tämän lisäksi yritys onnistui houkuttelemaan pääomasijoittajia ja saamaan itselleen korkean tason yhteistyökumppaneita. Näihin yhteistyökumppaihin lukeutuivat mm. Veritas, Novell, Red Hat ja Rackspace. Vuonna 2008 Sun Microsystems osti MySQL AB:n ja vuonna 2009 Oracle osti Sun Microsystemsin. (Gillmore, 2010, 477)

MySQL:n kehityksessä on keskitytty alusta alkaen ohjelmiston skaalautuvuuteen ja suorituskykyyn. Vaikka tällä tavalla saatiin aikaiseksi hyvin optimoitu tuote, puuttui siitä monia ominaisuuksia, jotka ovat standardeja yritystason tuotteita. Näihin ominaisuuksiin lukeutuvat esimerkiksi talletettu proseduuri (stored procedure), laukaisimet (triggers) ja transaktiot. MySQL kiinnosti kuitenkin käyttäjiä, jotka arvostivat enemmän nopeutta ja skaalautuvuutta, kuin ominaisuuksista, jotka voisivat kaikesta huolimatta jäädä hyödyntämättä. Myöhemmissä versioissa ominaisuudet on lisätty, jolloin saatiin entistä enemmän kiinnostuneita käyttäjiä. (Gillmore, 2010, 477)

Tänä päivä MySQL:n käyttäjiin lukeutuu monia tunnettuja yrityksiä kuten CNET Networks, NASA, Yahoo!, Google, The Weather Channel ja Cisco Systems. (Gillmore, 2010, 477)

MySQL toimii monissa eri ympäristöissä. Näihin ympäristöihin kuuluvat muun muassa Windows, Linux, FreeBSD ja Solaris. Tämän lisäksi lähdekoodi on kaikkien saatavilla, jolloin ohjelmakoodi voidaan itse kääntää käytettävään ympäristöön. MySQL sisältää laajan kirjon ohjelmistorajapintoja eri ohjelmointikielille. Näihin kieliin lukeutuvat mm. C++, Java, C, PHP, Tcl ja Ruby. Tietokantamoottoreiden kirjo MySQL:ssä on laaja. Näihin kuuluvat muun muassa MyISAM, MEMORY, InnoDB ja ARCHIVE. Jokaisella tietokantamoottorilla ovat omat vahvuutensa ja heikkoutensa, jonka vuoksi tietokantamoottori tulisi valita käyttötarkoituksen perusteella, jotta tapa käsitellä tietoa palvelisi käyttötarkoitusta. Myös tietokannan tauluille voidaan valita tilanteen mukaan eri tietokantamoottorit, sillä taulujen käyttötarkoitus voi erota toisistaan. (Gillmore, 2010, 477-478)

MySQL tukee tekstien indeksoimista ja hakemista, jonka vuoksi suorituskyky on parantunut huomattavasti haettaessa tietoa taulun tekstipohjaisista sarakkeista. Nopeutetaan liittyvänä etuna voidaan myös mainita MySQL:n tuki kyselyiden tallentamisesta välimuistiin Tällöin SELECT-kysely ja sen avulla saadut tulokset voidaan tallentaa muistiin. Jos sama kysely tehdään uudestaan, osaa MySQL palauttaa tulokset muistista, jolloin tietoa ei tarvitse hakea tietokannasta. Tällöin nopeus paranee. Vanhentuneiden tulosten välttämiseksi, mekanismi poistaa väärät tulokset ja lataa uudet tulokset muistiin seuraavan kyselyn aikana jos muutoksia tuloksiin on tapahtunut. (Gillmore, 2010, 479)

Replikoinnin avulla voidaan yhden MySQL-palvelimen sisältämät tiedot kopioida toiselle palvelimelle, jolloin kumpikin palvelin sisältää samat tiedot. Jos esimerkiksi yhteys palvelussa käytettyyn pääpalvelimeen katoaa, voi toinen palvelin ottaa pääpalvelimen tehtävät itselleen. Replikointi mahdollistaa myös kyselyiden suorittamisen useiden palvelinten kautta, jolloin yhteen palvelimeen kohdistuva kuorma alenee. Lisäksi varmuuskopioita otettaessa voidaan palvelu pitää ylhäällä, sillä samalla kun toinen palvelin vastaa palvelun kyselyihin voi toinen palvelin suorittaa varmuuskopioiden ottamisen. (Gillmore, 2010, 479)

3.3.6 phpMyAdmin (MySQL-tietokannan hallintajärjestelmä)

PhpMyAdmin on MySQL-tietokannan hallintaan tarkoitettu web-sovellus, joka kirjoitettu PHP-kielellä. Se kuitenkin sisältää myös JavaScript-, XHTML- ja CSS-koodia, jotka ovat yleisiä myös muissakin web-sovelluksissa. PhpMyAdminia ovat tukeneet useat kehittäjät ja kääntäjät maailmanlaajuisesti. Tätä on edesauttanut se, että phpMyAdmin on ollut alusta alkaen avointa lähdekoodia. Tavoitteena phpMyAdminilla on valmis web-pohjainen MySQL-palvelimien ja tiedon hallintatyökalu. Lisäksi phpMyAdminin tavoitteena on kestää mukana MySQL:n ja web-standardien kehityksessä. Ilmoitettujen bugien ja toivottujen ominaisuuksien pohjalta kehittäjät hienosäätävät phpMyAdminia jatkuvasti. Tämän vuoksi uusi versio phpMyAdminista julkaistaan säännöllisesti. Ominaisuuksien puolesta phpMyAdmin kattaa MySQL-tietokantaa ja tauluja koskevat perustoiminnot. Kehittyneempiin ominaisuuksiin kuuluu metatietoja ylläpitävä sisäinen relaatiojärjestelmä (internal relational system). PhpMyAdmin mahdollistaa lisäksi käyttäjäjien ja käyttöoikeuksien hallinnan. Käyttöoikeuksien hallinnan avulla pystytään vaikuttamaan siihen, millaisia toimintoja käyttäjä voi suorittaa MySQL palvelimella. (Delisle, 2010, 9)

Projektin osalta phpMyAdmin on ollut tärkeä työkalu asiakashallinnan kehittämisessä. Projektin alkuvaiheessa mahdollisti phpMyAdmin tietokannan ja tietokannan taulujen rakentamisen käyttämällä graafista työkalua. Samalla pystyy myös hahmottamaan mitä eri tauluja tietokanta sisältää. Tietokannan taulujen rakentamisessa phpMyAdmin mahdollisti taulujen ja niiden sarakkeiden toteuttamisen helposti. Jos myöhemmin tauluun joutui lisäämään uuden sarakkeen tai uusia sarakkeita, oli taulujen sisältöön mahdollista vaikuttaa. Projektin edetessä tietokantaan tuli muutoksia, yleisesti voidaan puhua lisätarpeista, joita olemassa oleva tietokanta ei palvellut. Tällöin phpMyAdmin helpotti muutosten toteuttamista. Koska testausta suoritettiin myös sovelluksen asiakaspuolella, tuli ajantasainen testitietokanta pystyä toimittamaan toimeksiantajalle. Tällöin phpMyAdmin mahdollisti tietokannan viemisen tiedostoon, jonka avulla tietokanta saatiin toimeksiantajan testiympäristöön.

PhpMyAdmin soveltui myös tietokantaan tapahtuvien muutosten seuraamiseen. Esimerkiksi lisätessä uusi käyttäjä tulee pystyä katsomaan, että käyttäjä lisätään todella tietokantaan. Samoin kuin muokkauksessa voidaan varmistaa, että muutokset tietoihin ovat todella tapahtuneet. Poistettaessa tietoa saattaa olla, että tiedolla voi olla yhteyksiä taulusta toiseen. Tällöin tulee pystyä varmistamaan, että tietokantaan ei jää haamutietoa, joita ei voida yhdistää alkuperäiseen yhteyteen. PhpMyAdminin asentaminen tapahtuu Ubuntussa paketinhallinnan kautta, jolloin käyttöönotto tapahtuu vaivattomasti eikä riippuvuuksista tarvitse huolehtia.

3.3.7 JavaScript

Yhtenä JavaScriptin määritteenä voidaan käyttää olioperustaista, heikosti tyypitettyä (weakly typed) skriptikieltä. Näitä voidaan pitää kriittisimpinä näkökulmina JavaScriptin osalta. Olioperusteisuus näkyy JavaScriptsissa muuttujina, jotka ovat oikeastaan objekteja. Muuttujatyyppinä objektit sisältävät mahdollisuuden omiin alimuuttujiin eli ominaisuuksiin (properties) ja funktioihin eli metodeihin. Yhteisenä nimityksenä ominaisuuksista ja metodeista voidaan käyttää jäseniä (members). (Ullman, 2012, 4)

Seuraavaksi voidaan käyttää esimerkkinä merkkijonoa, jonka avulla käydään läpi muutama esimerkin ominaisuuksista ja funktioista. Merkkijono muodostuu merkeistä, jotka ovat lainausmerkkien välissä. Tällöin esimerkissä käytetty muuttuja teksti on oikeastaan objekti, jonka tyyppinä on merkkijono. Tällöin saadaan käyttöön merkkijonoon liittyvät ominaisuudet ja metodit. (Ullman, 2012, 4)

Esimerkkinä käytettävä tekstijono:
var teksti = "ESIMERKKI";
Esimerkki metodista, joka muuttaa tekstin kirjaimet pieniksi kirjaimiksi:
teksti = teksti.toLowerCase();
Esimerkki ominaisuudesta, jonka avulla selvitetään merkkijonon pituus:
pituus = teksti.length();

Heikko tyypitys näkyy JavaScriptissa siinä, että muuttujien ja datan tyyppejä pystyy muuntamaan helposti tyypistä toiseen. Myöskään uutta muuttujaa luotaessa sen tyyppiä ei tarvitse määrittää, jolloin riittää, että muuttujalle annetaan arvo. (Ullman, 2012, 5)

Esimerkki numerotyyppisestä muuttujasta:
var kappaleita = 5;
Esimerkki muunnoksesta, jossa numero muutetaan tekstiksi:
kappaleita += 'kappaletta'; // Tuloksena syntyy teksti "5 kappaletta"

Heikko tyypittely tarjoaa mahdollisuuden joustavuuteen ja toiset ohjelmoijat arvostavat tätä ominaisuutta. Toiset ohjelmoijat vastaavasti näkevät, että joustavuus mahdollistaa huolimattoman ohjelmoimisen. Koska muunnokset tyypistä toiseen voivat tapahtua automaattisesti, kutsutaan JavaScriptiä myös dynaamisesti tyypitetyksi. Tällöin bugeja voi syntyä epäsuorien muunnosten vuoksi. (Ullman, 2012, 5-6)

Skriptauskielen JavaScriptista tekee se, että JavaScript ajetaan sen ohjelman kautta, joka suorittaa koodin. Esimerkiksi C täytyy kielenä kääntää ensin ohjelmaksi, jonka jälkeen käännetty ohjelma voidaan ajaa. JavaScript ajetaan vastaavasti usein Internet-selaimen kautta, jolloin koodin suorittamiseen käytetään selaimessa olevaa JavaScript-moottoria. (Ullman, 2012, 6)

3.3.8 jQuery

JQuery itsessään ei ole erillinen ohjelmointikieli vaan JavaScript-kirjasto, joka sisältää kokoelman erilaisia toiminnallisuuksia, jotka mahdollistavat yleisesti tehtyjen tehtävien suorittamisen. Tällöin web-kehittäjien ei tarvitse rakentaa näitä toiminnallisuuksia joka kerta alusta vaan he voivat lisätä jQueryn omiin projekteihinsa. JQuery mahdollistaa tällöin yleisten ja tylsien JavaScriptilla tehtävien käsittelemisen helpommin. Lisäksi jQueryn kehityksessä on huomioita eri selainten erot JavaScriptin käsittelyssä, jolloin jQuerylla rakennetut toiminnallisuudet toimivat monilla eri selaimilla. Muutamalla rivillä koodia voidaan suorittaa tehtäviä, jotka puhdasta JavaScriptiä käyttämällä voivat vaatia monta riviä koodia. (MacLees, 2012, 7)

jQueryn käyttäjät koostuvat kehittäjistä ja suunnittelijoista ympäri maailman, mutta myös yritykset hyödyntävät jQueryä. Näihin yrityksiin lukeutuvat mm. Google, Mozilla, Wordpress, Twitter ja Microsoft. Pelkän JavaScriptin käyttäminen vaatisi mm. prototyyppien, funktioiden ja luokkien opettelua, kun taas jQuery mahdollistaa ideoiden tuottamisen nopeammassa ajassa. (Castleline ja Sharkie, 2010, 1)

3.3.9 Twitter Bootstrap

Twitter Bootsrap on avoimeen lähdekoodiin perustuva joukko työkaluja, joiden avulla suunnittelijat ja kehittäjät voivat helpommin rakentaa sivustoja. Tavoitteena on saada aikaiseksi laaja ja hyvin dokumentoitu kirjasto, joka sisältää joustavia suunnitteluun tarkoitettuja HTML, CSS ja JavaScript-komponentteja. Twitterin sisäisessä käytössä olleet aikaisemmat työkalut eivät olleet hiottuja ja helposti lähestyttäviä, joka vaikeutti nopeaa kehitystä ja iterointia. Tässä nähtiin kuitenkin mahdollisuus tulevien projektien kannalta. Kun tämä oli tiedostettu, aloitettiin prosessi, jossa suunnittelu ja tekniikka yhdistettiin jo varhaisessa vaiheessa. (Otto, 2012)

Artikkelissa prosessi kuvataan nelivaiheisena. Ensimmäisessä vaiheessa kehittäjät työskentelivät tuotepäälliköiden ja työkalujen potentiaalisten käyttäjien kanssa, jotta tärkeät toiminnallisuudet ja ominaisuudet voitaisiin tunnistaa. Toisessa vaiheessa yksi Bootsrapin kehittäjistä, Mark Otto teki työtä kehittäjien kanssa yhteisten tarpeiden tunnistamiseksi. Tämän jälkeen tarpeet suunniteltiin selaimessa, jotta saataisiin luotua visuaalinen kieli ja, että vuorovaikutusta voitaisiin tutkia. Kun ensimmäinen toteutus oli saatu tehtyä, käytiin jokaisesta komponentista keskustelu sekä punnittiin myös muut toteutukset ja vaihtoehdot. Kolmannessa vaiheessa suunniteltiin ja ohjelmoitiin yksittäiset komponentit. Kolmannen vaiheen aikana uudet ominaisuudet nopeasti toteutettiin, testattiin ja iteroitiin. Neljännessä vaiheessa Mark Otto otti komponentit, jotka olivat tehty sisäisiin työkaluihin ja lisäsi ne jaettuun koodikantaan. Koodikannalla tarkoitetaan tässä yhteydessä Bootstrappia. Lisäys tehtiin, jotta Otto voisi tehdä komponenteista yhteenvedon ja dokumentoida komponentit tulevia projekteja ajatellen. (Otto, 2012)

Koska Bootstrapin rakentamisessa käytettiin tällaista lähestymistapaa, oli kommunikaatio avainasemassa. Lisäksi suurin osa suunnittelutyöstä tapahtui koodissa. Ideoista kommunikoiminen kooditasolla aina kun se oli mahdollista, tuntui järkevimmältä, sillä tuotteena Bootstrap on myös koodia. Suurin osa komponenteista ja niiden ominaisuuksista on tehty yhteistyössä kehittäjien ja suunnittelijoiden kanssa. (Otto, 2012) Prosessi uusien ominaisuuksien kehittämiseksi ja uusien komponenttien suunnittelemiseksi kehittyi muotoon:

1. Ideointi
2. Ominaisuuden arvioiminen ja keskustelu
3. Toteutus
4. Abstraktio ja dokumentaatio

Näitä samaa neljää vaihetta käytetään myös silloin, kun olemassa olevia ominaisuuksia halutaan muokata tai poistaa. Bootstrapin dokumentaatiossa komponenteista on olemassa toimivat esimerkit ja tätä kautta on voitu jakaa suunnittelijoiden opastusta. Interaktiivisen dokumentaation vuoksi Bootsrap ei ole pelkästään joukko työkaluja. Tällä tavoin on myös pystytty huomioimaan käyttäjien erilaiset tasot. Käyttäjällä on mahdollisuus mm. ladata lähdekoodi tai selata koodia selaimen avulla, jolloin käyttäjän osaamistaso ei ole esteenä Bootstrapin käytölle. (Otto, 2012)

Twitter Bootsrapin kehityksen käynnisti pieni ryhmä, joka koostui Twitterin työntekijöistä. Päämääränä oli luoda työkalut, joita voitaisiin käyttää Twitterin sisäisesti ja muuallakin. Tämän vuoksi lähdettiin rakentamaan järjestelmää, joka auttaisi rakentamaan uusia projekteja sen päälle. Bootstrapin kehittäjät eivät halunneet Bootstrapin olevan pelkästään Twitterin sisäinen työkalu vaan sen käyttäminen haluttiin mahdollistaa kaikille suunnittelijoille ja kehittäjille. (Otto, 2012)

Bootsrapin rakensivat Mark Otto ja Jacob Thornton. Tänä päivänä Bootsrap koostuu tuhansista komponenteista ja GitHubissa sillä on 13 000 seuraajaa sekä 2 000 haaraa. Tämän lisäksi Bootstrap on suosituin projekti GitHubissa. Uusia ominaisuuksia on suunnitteilla, mutta niiden päätyminen Bootsrappiin vaatii ominaisuuksien tarpeellisuuden ja toiminnallisuuden tunnistamisen. Otton ja Thornotin tehtäviin kuuluu jäljittää bugeja ja ongelmia sekä hallinnoida pyyntöjä uusien ominaisuuksien osalta. Uusia ominaisuuksien lisäämisessä kiinnitetään huomiota siihen, etteivät ne hämmennä käyttäjiä eivätkä tarpeettomasti paisuta rajapintaa. Projektin kannalta tärkeänä ja tarpeellisena nähdään, että ihmiset pystyvät tasavertaisesti arvostelemaan uusia ominaisuuksia. (Otto, 2012)

3.3.9.1 Twitter Bootstrapin rooli projektissa

Bootstrapin käyttäminen projektissa toi mukanaan monia etuja, jotka omalta osaltaan nopeuttivat työkalun rakentamista ulkoasun osalta. Bootstrap toi mukanaan laajan kirjon erilaisia elementtejä, joiden rakentaminen alusta alkaen projektia varten olisi vaatinut paljon työtä ja aikaa. Koska Bootsrapissa on eri selainten eroavaisuudet pyritty ottamaan huomioon, vähentää tämä työtä sivuston viilaamisessa kaikille selaimille sopivaksi. Tämä ei kuitenkaan poista tarvetta testaukselle eri selainten avulla. Voisi myös helposti luulla kaikkien Bootsrapilla tehtyjen sivujen näyttävän samalta. Vaikka oletuksena Bootsrap sisältääkin valmiiksi määritetyt tyylit elementeille esimerkiksi napeille, ei tämä tarkoita, ettei näihin asetuksiin voisi itse vaikuttaa. Internetistä löytyy jopa valmiita työkaluja, joiden avulla voi muokata tyyliä ja muutoksia pystyy esikatselemaan. En näe Bootsrapin tarkoituksena tehdä kaikista sivuista samannäköisiä vaan tarjota hyvä pohja, jolla pääsee alkuun.

Bootsrapin sisältämän grid systemsin ansiosta sivuston sisältöä pystyi helposti jakamaan sarakkeisiin tarpeen mukaan. Grid systems mahdollistaa sarakkeiden yhteisleveyden määrittämisen tarkaksi arvoksi, jolloin oletuksena sarakkeet ovat yhteensä 940 pixeliä leveitä. Tätä asettelua käytettiin projektissa. Grid system mahdollistaa myös asettelun, jossa sarakkeiden kokoon vaikuttaa selaimen ikkunan koko, mutta tämä malli tuotti ongelmia projektin alussa, jolloin siirryttiin käyttämään 940 kuvapisteen yhteisleveyttä. Bootstrap sisälsi valmiit määritykset navigointipalkin ja siinä käytettyjen linkkien sekä valikkojen rakentamiseen. Tällöin suurempi työ tehtiin ohjelmoimalla PHP:lla ehdot, joiden perusteella navigointipalkki mukautui henkilön käyttöoikeuksien perusteella. Tämä siksi, ettei ole järkevää näyttää linkkiä sivulle, jonne henkilöllä ei ole oikeuksia.

Tämän projektin kannalta oleellisimmiksi elementeiksi muodostuivat lomakkeet, taulut sekä painikkeet. Käytettävyyden kannalta voidaan mainita Bootsrapissa määriteltyjen painikkeiden värimaailma. Esimerkiksi vihreällä painikkeella voidaan kertoa siitä, että toiminto, jonka painike suorittaa on positiivinen. Esimerkiksi kun lisätään asiakasta, voidaan tämä mieltää positiivisena asiana. Jos vastaavasti asiakkaan tietoja ollaan muokkaamassa, voidaan käyttää keltaista painiketta. Tällöin väri viestii siitä, että muokkaukseen tulee suhtautua varauksella, sillä se muokkaa olemassa olevia tietoja. Lisäämisen vastakohtana voidaan vastaavasti pitää poistamista. Punaisen värinen poistopainike kertoo värillään negatiivisesta toimenpiteestä.

4 ASIAKASHALLINTA

4.1 Tausta

Johdanto-osassa käy jo ilmi, että olen ollut opintoihin liittyvässä harjoittelussa yrityksessä, jolle tämä opinnäytetyö on tehty. Harjoittelussa olin ohjelmoimassa palvelua, jolla tulisi olemaan enemmän kuin yksi asiakas, minkä vuoksi harjoittelun aikana ilmeni tarve asiakashallinnalle. Asiakashallinnan tulisi palvella niin ylläpitäjiä kuin asiakkaitakin, sillä määrättyjen toiminnallisuuksia tulisi olla käytettävissä myös asiakkailla. Harjoittelussa toteutettua palvelua varten tuli saada tehtyä työkalu, joka mahdollisti sivujen automaattisen luomisen asiakkaalle, jolloin vältytään toistuvien toimenpiteiden suorittamiselta. Koska sivuston runko on kaikilla asiakkailla sama, tuli työkalun pystyä luomaan palvelun käyttäjien sivustoille henkilökohtaiset asetustiedostot ja tietokannat. Asiakkaan tietokanta sisältää esimerkiksi keskustelualueet ja keskustelualueilla käydyt keskustelut sekä tiedotteet. Dynaamisten sivustojen vuoksi tietokanta on siis välttämätön.

Tämän luvun tarkoituksena on kuvata itse asiakashallintaa ja kuinka siinä on eroteltu ylläpitäjän sekä asiakkaan puoli asiakashallinnan käyttäjän näkökulmasta. Asiakashallinnan tarkoituksena on toimia työkaluna lisättäessä uusia käyttäjiä järjestelmään ja suorittaa sellaisia toimenpiteitä, jotka jouduttaisiin muussa tapauksessa tekemään käsin. Tällöin voidaan vähentää sellaisen työn määrää, joka on automatisoitavissa. Samalla asiakashallinta tukee työharjoittelussa rakennettua palvelua ja mahdollistaa sen käyttöönottamisen eri asiakkailla nopeammin ja helpommin. Asiakashallinnan asiakaspuolen ansiosta asiakkaiden ei tarvitse teetättää kaikkia toimintoja palveluntuottajan kautta, jolloin säästyy asiakkaan ja palvelua tuottavan yrityksen aikaa sekä resursseja. Esimerkiksi asiakas voi päivittää omat tietonsa omien tunnuksien avulla. Vastaavasti palveluntuottaja voi käyttää säästyneen ajan ydintoimintoihinsa, esimerkiksi palvelun parantamiseen. Tämä mahdollistaa sen, että asiakas voi tehdä esimerkiksi jatkosopimuksen itselleen sopivana ajankohtana.

4.2 Toteutus

Asiakashallintaprojektin toteutuksessa käytettiin ketterän ohjelmistokehityksen periaatteita ja tämä näkyi esimerkiksi siinä, ettei käytettäviä työkaluja tarkasti määritetty. Tällöin pystyin valitsemaan ne työkalut, joiden käyttö tuntui itsestä luontevalta, mikä lisäsi motivaatiota työskentelyn osalta. Tietenkin esimerkiksi versionhallintana käytettiin Gittiä, mutta se käytinkö Gittiä komentorivin vai editorin kautta jäi itseni valittavaksi. Lopulta huomasinkin, että komentorivin kautta tapahtuva Gitin käyttäminen oli itselleni luontevinta. Koska tietokannan ja palvelinsovelluksen osalta käytettiin avoimia ratkaisuja, pystyin käyttöjärjestelmän ja työkalujen osalta valitsemaan juuri itselle sopivat. Toisaalta tällainen toimintamalli vaatii itsenäistä paneutumista hyviin ja itselle sopiviin työkaluihin, jotta päästään haluttuun lopputulokseen. Tätä osaltaan tukee sellaisten tekniikoiden ja standardien käyttö, joille on kehitetty erilaisia työkaluja.

Toinen seikka, joka ketterän ohjelmistokehityksen osalta ilmeni projektissa, oli vuorovaikutus. Ilman vuorovaikutusta koko asiakashallintaprojekti ei todennäköisesti olisi onnistunut. Kun projektia päivitetään versionhallinnan kautta lyhyissä sykleissä, on tärkeää saada palautetta päivityksen toimivuudesta, jotta esimerkiksi virheet voidaan korjata mahdollisimman pian. Toisaalta myös yhteistyö korostui kehitystyössä. Tarvittaessa pystyin kysymään neuvoa esimerkiksi sovelluksessa käytettyjen ratkaisujen osalta.

Aikakäsitys erosi tässä projektissa ketterän ohjelmistokehityksen periaatteista, sillä osaltaan opinnäytetyölle varattu aika muodosti takarajan projektin valmistumiselle. Tällöin projektin takarajan ja toimitusaikojen muuttaminen mielivaltaisesti on mahdotonta tai tällöin ei projektia saada valmistumaan haluttuun aikarajaan mennessä. Tämän aikakäsityksen näen kuitenkin hyvänä kun esimerkiksi sovelluksesta on olemassa toimiva versio ja sitä kehitetään eteenpäin. Tällöin jos esimerkiksi haluttujen toiminnallisuuksien määrä vähenee, voidaan siirtää takarajaa aikaisemmaksi. Tällöin esimerkiksi projektia varten varatut resurssit voidaan ohjata muihin projekteihin, jolloin voidaan vähentää joutokäyntiä. Näen kuitenkin nopean reagoinnin tärkeänä lyhyemmässäkin projektissa, jotta tiukempi aikataulu pystytään pitämään.

Varsinainen sovelluksen kehitystyö käynnistyi sillä, että toimeksiantaja määritti ne toiminnallisuudet, joita sovellukselta odotetaan. Toiminnallisuuksia ei kuitenkaan määritetty hyvin tarkalla tasolla, vaan projektin edetessä yksityiskohdat hiottiin lopulliseen kuosiinsa. Tämä osaltaan mahdollisti nopeamman reagoinnin muutostarpeisiin, joita projektin aikana saattoi ilmetä. Toisaalta kun suunnitelmia ei tehty hyvin tarkalla tasolla, pystyttiin vähentämään sellaisia ominaisuuksia, jotka myöhemmin osoittautuisivat turhiksi. Lyhyempien kehityssyklien ansiosta pystyi palautetta pyytämään nopeammin uuden ominaisuuden osalta ja jos testauksessa ilmeni ongelmia, pystyttiin palaamaan ominaisuuden kehittämiseen. Jos kokonainen sovellus olisi viety keralla testaukseen, olisi tämä todennäköisesti tarkoittanut työläämpää virheiden korjaamista. Tämä kuitenkin vaatii, että kaikki osapuolet ovat sitoutuneita projektiin ja sen läpiviemiseen.

Asiakashallinnan työstäminen tapahtui pääsääntöisesti etätyöskentelyn avulla, joka sulki pois kasvokkain tapahtuvan kommunikaation. Projektin aikana kommunikaatio tapahtuikin pääsääntöisesti pikaviestien ja sähköpostien avulla. Kommunikaatiota tapahtui toimeksiantajan suuntaan myös versionhallinnan kautta. Versionhallinnassa muutoksista kirjoitetaan viesti, jossa kerrotaan tehdyistä muutoksista. Tällöin versionhallinnan kautta ovat nähtävissä ohjelmoijan jättämät viestit, jotka ovat esimerkiksi toimeksiantajan luettavissa. Tämän lisäksi tapanani oli päivitysten osalta myös ilmoittaa muutoksista sähköpostin ja pikaviestien välityksellä. Näin halusin varmistaa, että tieto muutoksista saadaan toimeksiantajan tietoon esimerkiksi testausta ajatellen.

Toimeksiantajan osalta kommunikaatio tapahtui sähköpostin ja pikaviestien välityksellä. Esimerkiksi isomman testauksen jälkeen sain sähköpostilla tiedot tarvittavista muutoksista, kehitysideoista ja ilmenneistä ongelmista sovelluksen osalta. Sähköpostiviestistä ideat ja ilmenneet ongelmat oli helppo poimia sekä suorittaa tämän jälkeen sovellukseen tehtävät muutokset. Toisaalta pikaviestien avulla oli koko projektin ajan mahdollisuus kysyä toimeksiantajalta tarkentavia kysymyksiä ja tarkentaa ilmenneitä ongelmakohtia sekä ideoita. Myös kustannustehokkuuden osalta voidaan sähköpostia ja pikaviestejä pitää hyvänä ratkaisuna, sillä niiden avulla voidaan esimerkiksi vähentää yhteydenottoja puhelimen välityksellä.

Kuinka sitten kommunikaatiota voitaisiin parantaa, jotta se tukisi paremmin etätyöskententelyä? Yhtenä vaihtoehtona voisi olla lisätä esimerkiksi etänä tapahtuvat videoneuvottelut, joissa käytäisiin läpi projektin etenemistä. Lisäksi voitaisiin käyttää projektityöskentelyyn tarkoitettuja työkaluja esimerkiksi uusien ideoiden ilmetessä. Varsinkin projektin kasvaessa suuremmaksi kasvaisi tarve uusille työkaluille varmasti merkittävämpään asemaan.

Kommunikaation merkitystä ei voi vähätellä projektin osalta ja kun sovellusta lähdettiin rakentamaan kohti lopullista versiota, olisi kommunikaation puuttuminen voinut helposti johtaa projektin pysähtymiseen. Jos projektia halutaan kehittää vähemmällä kommunikaatiolla, vaatisi tämä mielestäni tarkemmat suunnitelmat, joiden mukaan projekti toteutettaisiin. Tällöin kuitenkin esimerkiksi huononisi mahdollisuus nopeampaan reagointiin esimerkiksi muutosten osalta, jolloin syntyisi riski myöhemmin turhiksi osoittautuvien ominaisuuksien syntymiselle.

4.3 Asiakastietojen hallinta

4.3.1 Uuden asiakasympäristön luominen

Asiakkaan lisääminen ei asiakashallinnassa tarkoita tässä tapauksessa pelkästään asiakaan tietojen lisäämistä tietokantaan. Koska asiakkaalla voi olla useita sivustoja palvelussa, tulee jokaista varten luoda oma sopimus ja suorittaa tarvittavat toimenpiteet sivuston toimintaan saattamiseksi. Nämä toiminnot sisältävät tietokannan luomisen asiakkaan sivustolle ja tietokannan rungon rakentamisen. Tietokannan lisäksi tulee sivustoa varten rakentaa asetustiedosto, jotta esimerkiksi osataan yhdistää oikeaan tietokantaan. Asiakkaan lisäämisestä huolehtii asiakashallinnan ylläpitäjä, eli palveluun ei ole mahdollista rekisteröityä itse käyttäjäksi. Tulevaisuudessa käyttäjälle voitaisiin antaa mahdollisuus rekisteröityä palveluun itse, jolloin ylläpidon ei tarvitsisi huolehtia kaikista rekisteröitymisistä.

4.3.2 Asiakkaan tietojen muokkaaminen

Asiakkaalla on mahdollisuus muokata tietojaan, jolloin tietojen muuttuessa ei tarvitse ottaa yhteyttä ylläpitoon tietojen päivittämiseksi. Tällä tavoin pyritään vähentämään ylläpidon tämän toiminnallisuuden osalta. Kuitenkin tarvittaessa ylläpitäjillä on mahdollisuus tietojen päivittämiseen. Tämä osaltaan säästää niin asiakkaan kuin ylläpidonkin aikaa tietojen muokkaamisen osalta.

4.3.3 Asiakasympäristön poistaminen

Vain järjestelmän ylläpitäjä voi poistaa asiakkaan ja poistamiseen liittyy myös muita toimenpiteitä kuin pelkästään asiakkaan tietojen poistaminen tietokannasta. Sen lisäksi, että huolehditaan asiakkaan tietojen poistamisesta, tulee huolehtia, että asiakkaan sivustot ja sivustoihin liittyvät tietokannat poistetaan. Tällä tavoin vältetään tilannetta, jossa palvelimelta löytyy sivustoja ja tietokantoja, joiden alkuperää ja omistajaa ei tiedetä. Tämä osaltaan voisi johtaa tilanteeseen, jossa tietokantoja ja sivustoja jouduttaisiin poistamaan käsin. Koska kuitenkin asiakashallinnan tarkoituksena on helpottaa ylläpidon työtä, tulee asiakashallinnan suoriutua poistoon liittyvistä toimenpiteistä.

4.4 Sopimustietojen hallinta

4.4.1 Sopimuksen lisääminen

Vaikka asiakkaan lisäämisessä tehdäänkin asiakkaalle sopimus yhtä sivustoa varten, ei tämä tarkoita, että asiakkaalla voisi olla vain yksi sopimus ja tätä kautta vain yksi sivusto. Koska asiakkaalla voi olla monta sivustoa käytössään, on asiakkaalla mahdollisuus tehdä itselleen uusia sopimuksia. Sopimuksen lisääminen myös lisää sopimusta koskevaa sivustoa varten tietokannan ja muodostaa tietokannan rungon sekä huolehtii asetustiedoston rakentamisesta sivustoa varten. Asiakas pystyy lisäämään kategoriaan vain yhden sopimuksen. Jos kategoriaan halutaan lisätä enemmän kuin yksi sopimus, huolehtii ylläpito lisäämisestä.

4.4.2 Sopimuksen jatkaminen

Sopimukset tehdään järjestelmässä aina määräaikaisina, eli toistaiseksi voimassa olevia sopimuksia ei ole olemassa. Tämän vuoksi on tärkeää, että asiakas voi jatkaa entistä sopimustaan jatkosopimuksella esimerkiksi sopimuksen vanhetessa tai ennen kuin sopimus ehtii vanhentua. Kun asiakkaalla on itsellään mahdollisuus sopimuksen jatkamiseen, ei sopimuksen jatkamiseksi ole välttämätöntä ottaa yhteyttä ylläpitoon vaan voi asiakas tehdä jatkosopimuksen itselleen sopivana ajankohtana.

4.5 Laskutustietojen hallinta

Asiakashallinnan tehtäväksi ei nähty opinnäytetyön tekohetkellä laskutuksen hallintaa ja toisaalta laskutuksen rakentamisesta olisi pystynyt muodostamaan oman kokonaisuutensa. Myöskään olemassa olevaan laskutukseen ei ollut saatavilla rajapintaa, jonka vuoksi päädyttiin ratkaisuun, jossa uudet sopimukset merkitään laskuttamattomiksi tietokantaan. Asiakashallinnan kautta on siis mahdollisuus selata laskuttamattomia ja laskutettuja sopimuksia. Kun esimerkiksi on katsonut sopimuksen tiedot laskutusta varten, voi sopimuksen merkata laskutetuksi. Vastaavasti laskutettuja sopimuksia voi merkitä laskuttamattomaksi esimerkiksi tilanteessa, jossa sopimuksen on vahingossa merkinnyt laskutetuksi.

Jatkokehitysideana tämän toiminnallisuuden osalta olisikin käytännöllistä rakentaa ratkaisu, jossa uudet sopimukset menevät automaattisesti laskutusjärjestelmään. Tämä vaatii laskutusjärjestelmältä toimivan rajapinnan tietojen viemiseksi ja toisaalta laskutusjärjestelmältä toiminnallisuuden, joka huolehtii tietojen viemisestä laskutukseen. Tällä tavoin vähennettäisiin työtä, joka syntyy laskujen luomisesta asiakashallinnassa olevien sopimustietojen pohjalta.

5 POHDINTA

Mitä sitten voi ajatella projektista lopulta jääneen käteen? Ainakin ensimmäisenä voisi todeta, ettei opinnäytetyön aikana tehty projekti mennyt perinteisen ohjelmistokehitysmallin mukaisesti. Esimerkiksi vesiputousmallia ei projektissa käytetty ja sen sijaan, että alussa olisi keskitytty tekemään tarkat suunnitelmat ja määritykset, aloitettiin asiakashallinnan ohjelmoiminen jo heti alussa. Tämä ei tarkoita, ettei alussa olisi ollut kuvaa siitä, millaisia ominaisuuksia työkalulta odotetaan vaan yksityiskohdat hiottiin sovelluksen kehittyessä eteenpäin. Voisikin todeta pääpainon olleen toimivassa sovelluksessa, eikä niinkään tarkoissa dokumentaatioissa. Koska opinnäytetyölle on varattu rajallisesti aikaa, olisi määritysten tarkka tekeminen saattanut olla koko opinnäytetyö. Tällöin varsinaisen sovelluksen ohjelmoiminen olisi jäänyt omaksi projektikseen. Kun sovellusta kehitettiin eteenpäin pala kerrallaan, pystyttiin uusiin tarpeisiin reagoimaan nopeammalla tahdilla. Jos sovellus olisi tehty ensin määritysten mukaan valmiiksi ja muutostarpeita olisi tämän jälkeen ilmennyt, olisi sovellus saattanut esimerkiksi sisältää sellaisia ominaisuuksia, joita ei enää tarvitakaan.

Koska lähestymismalli, jossa sovellusta kehitetään pala kerrallaan eteenpäin, vaatii kommunikaatiota, tulee myös tilaajan olla sitoutunut kommunikaatioon projektin edetessä. Tämän opinnäytetyön aikana kommunikaatiota tapahtui useaan otteeseen. Myös tilaajan puolelta tapahtuneesta testauksesta oli tärkeää saada tietoa, jotta näiden tietojen pohjalta voitiin kehittää sovellusta. On selvää, ettei kaikissa tapauksissa tällainen lähestymistapa ole mahdollista esimerkiksi yrityksellä olevien muiden projektien vuoksi. Tällöin näen tärkeänä, että voidaan pohtia myös toisenlaisia lähestymistapoja. Vaikka sovelluksen työstäminen tässä projektissa tapahtuikin pääsääntöisesti etänä, ei tämä ollut este kommunikaatiolle. Sähköpostit ja pikaviestit olivat tärkeässä osassa projektin kannalta. Jos jatkon kannalta haluttaisiin jotain kehittää, voitaisiin harkita ääni- ja videopuheluita. Mahdollisesti voitaisiin myös kartoittaa tarvetta projektityöskentelyä tukevien työkalujen osalta. Ohjelmointiin liittyvien työkalujen suhteen näen itse hyödyllisenä, että tekijällä on valtaa päättää käyttämänsä työkalut. Internet mahdollistaa tiedon hakemisen ja erilaisten työkalujen vertailun, jolloin itselleen sopiva työkalu on helpompi löytää. Tätä tukevat myös avointen tekniikoiden käyttö, joita voidaan kehittää monilla eri työkaluilla.

Twitter Bootsrap mahdollisti sivuston ulkoasun nopeamman kehittämisen, jolloin säästynyttä aikaa pystyttiin käyttämään esimerkiksi toiminnallisuuden kehittämiseen. Tämän opinnäytetyön aikana Bootsrappia käytettyäni on helppo suositella sitä varsinkin silloin, kun projektin pääasiallisena tehtävänä ei ole keskittyä rakentamaan esimerkiksi sivuston ulkoasua. Myös sivuston toimivuus erilaisilla selaimilla on helpompaa varmistaa, kun käytetään rajapintaa jossa selaimien erot on pyritty huomioimaan rajapinnan käyttäjän puolesta. Huonona puolena tällöin voidaan kuitenkin nähdä, että valmiita työkaluja käytettäessä voivat ne pyrkiä ohjaamaan tekemään asioita tietyllä tavalla. Toisaalta jos asiat tehdään tietyllä tavalla, tekee se sivustosta yhtenäisen.

Projektin aikana pystyin kertaamaan ja laajentamaan omaa osaamistani niiden osa-alueiden osalta, joita olin harjoittelun kautta ennen opinnäytetyön aloittamista oppinut. Opinnäytetyössä hyödynsin PHP:n nimiavaruuksia ja luokkia. Nämä olivat PHP:n osalta sellaisia ominaisuuksia, joita en ollut aiemmin käyttänyt. Luokkien avulla saatiin sovelluksessa käytetyt funktiot parempaan järjestykseen ja parannettiin funktioiden nimeämistä. Työkalun tulevaisuuden kannalta luokkien avulla voidaan helpottaa uusien toiminnallisuuksien jaottelua. Myös monien muiden tekniikoiden käyttö, joita ei ollut koulussa käytetty, laajensi omaa osaamistani. Voikin sanoa, että on tärkeää pystyä ylittämään tekniikoiden osalta oma mukavuuskynnyksensä. Opintojen kautta olin kuitenkin saanut pohjan ohjelmoinnin opiskelulle, joten ainakaan PHP ei tuntunut ylitsepääsemättömän vaikealta. Esimerkiksi hakukonetta on hyvä osata hyödyntää uuden kielen opiskelussa, kunhan muistaa suhtautua lähteisiin aina kriittisesti. Onnistumisen tunne syntyikin osaltaan siitä, että pystyi ylittämään oman mukavuuskynnyksensä ja saamaan tällä tavoin aikaiseksi toimivan sovelluksen, jonka tekemiseen on käytetty erilaisia tekniikoita.

Se mitä opinnäytetyön kautta viimeistään opin, oli versionhallintajärjestelmän tärkeys sovelluskehityksessä. Opinnäytetyötä kirjoittaessani aloinkin ihmettelemään miksei sovelluskehityksen tunneilla käsitelty tätä aihealuetta? Ilman versionhallintajärjestelmää voidaan pärjätä yhden ihmisen projekteissa, joissa muilla ei ole tarvetta tiedostojen seuraamiseen tai tarkasteluun. Toisaalta tällöin menetetään mahdollisuus esimerkiksi automaattisen historian ylläpitämiseen ja mahdollisuuteen palata versiosta toiseen. Onko versionhallinnan käytöstä haittaa sovelluskehityksen kannalta? Tämän projektin aikana versionhallinnan käyttö osoittautui tarpeelliseksi eikä negatiivisia seikkoja ilmennyt.

Harjoittelussa toteutetun palvelun kannalta asiakashallinnan kehittäminen oli välttämätöntä. Ilman asiakashallintaa palvelun käyttöönottaminen eri asiakkailla muuttuisi helposti työlääksi ja vaikka asiakashallinnan suorittamat toimenpiteet eivät olisikaan työläitä, voidaan sen avulla välttää samojen toimenpiteiden tekemistä monia kertoja. Ja koska asiakkaalla on mahdollisuus suorittaa tiettyjä toimintoja itse, säästyy näiden osalta ylläpidon aikaa. Tämän opinnäytetyön tehtävänä ei ole ollut huolehtia asiakashankinnasta tai olla mukana lanseeraamassa palvelua. Opinnäytetyön tehtävänä on ollut tehdä työkalu, joka tukee palvelun käyttöönottoa. Tulevaisuuden kannalta näen tarpeellisena, että asiakashallintaa kehitetään sen käyttäjien tarpeiden mukaan. Tällöin kehitysideoita voi olla odotettavissa muun muassa palvelun asiakasrajapinnasta. Tällöin tulee pystyä vastaamaan tarpeisiin, joita asiakashallinnan käyttäjillä ilmenee. Muutoksiin, tarpeisiin ja palautteeseen on hyvä pystyä reagoimaan mahdollisimman lyhyessä ajassa, jotta palvelua voidaan kehittää tehokkaasti palvelemaan sen käyttäjien tarpeita. Tällöin ketteryys voi toimia helpottavana tekijänä.

LÄHTEET

ABRAHAMSSON, Pekka ja SALO, Outi 2007. Ketterät tuulet ohjelmistotuotannossa. Systeemityö [viitattu 2013-01-06]. Saatavissa: http://www.pcuf.fi/sytyke/lehti/kirj/st20074/ST074-04A.pdf

ALLEN, Richard, QIAN, Kaj ja TAO, Lixin ja FU, Xiang. 2010. Web Development with JavaScript and Ajax Illuminated. Mississauga: Jones & Barlett Learning.

APPELO, Jurgen. 2010. Management 3.0: Leading Agile Developers, Developing Agile Leaders. Crawfordsville: Addison-Wesley Professional.

ARONSON, Larry. 2010. HTML Manual of Style: A Clear, Concise Reference for Hypertext Markup Language (Including HTML5), Fourth Edition. Crawfordsville: Addison-Wesley Professional.

BELCHAM, Donald ja BALEY, Kyle. 2010. Brownfield Application Development in .NET. Greenwich: Manning Publications.

BLANKENSHIP, Jerrel, BUSSA, Matthew ja MILLETT, Scott. 2011. Pro Agile .NET Development with Scrum. New York: Apress.

CASTLELINE, Earle ja SHARKIE, Craig. 2010. jQuery: Novice to Ninja. Collingwood: SitePoint.

CHACON, Scott. 2009a. Getting Started – About Version Control. Git [viitattu 2013-01-15]. http://git-scm.com/book/en/Getting-Started-About-Version-Control.

CHACON, Scott. 2009b. Getting Started – Git Basics. Git [viitattu 2013-01-14]. Saatavissa: http://git-scm.com/book/en/Getting-Started-A-Short-History-of-Git.

CHACON, Scott. 2009c. Getting Started - A Short History of Git. Git [viitattu 2013-01-14]. Saatavissa: http://git-scm.com/book/en/Getting-Started-Git-Basics.

COLBURN, Rafe ja LEMAY, Laura. 2010. Sams Teach Yourself Web Publishing with HTML and CSS in One Hour a Day: Includes New HTML5 Coverage, Sixth Edition. Indianapolis: Sams Publishing .

DELISLE, March. 2010. Mastering phpMyAdmin 3.3.x for Effective MySQL Management. Birmingham: Packt Publishing Ltd.

GILLMORE, W. Jason. 2010. Beginning PHP and MySQL: From Novice to Professional, Fourth Edition. New York: Apress.

ELYSIUM SOLUTIONS OY KOTISIVU, 2013. Elysium Solutions Oy [viitattu 2013-02-28]. Saatavissa: http://elysium.fi/.

HARTEMO, Mari 2011-10-31. Ketteryys löytyy korvien välistä [verkkoaineisto]. Kauppalehti [viitattu 2013-01-06]. Saatavissa: http://www.kauppalehti.fi/sponsoroidutblogit/sek/ketteryys-loytyy-korvien-valista.

MACLEES, Natalie. 2012. jQuery for Designers. Birmingham: Packt Publishing.

MEYER, Eric A. 2012. CSS and Documents. Sebastopol: O'Reilly Media, Inc.

OTTO, Mark. 2012. Building Twitter Bootsrap. A List Apart [viitattu 2013-01-06]. Saatavissa: http://www.alistapart.com/articles/building-twitter-bootstrap/.

THE PHP GROUP, 2013. History of PHP. PHP Manual [viitattu 2013-01-08]. Saatavissa: http://php.net/manual/en/history.php.php.

POUNCEY, Ian ja YORK, Richard. 2010. Beginning CSS: Cascading Style Sheets for Web Design, Third Edition. Indianapolis: Wiley Publishing.

SYRJÄNEN, Tuomas. 2008. Kun ketteryys korvasi kankeuden [verkkoaineisto]. digitoday [viitattu 2013-01-10]. Saatavissa: http://www.digitoday.fi/mielipide/2008/11/21/kun-ketteryys-korvasi-kankeuden/200830208/66.

TALOUSSANOMAT, 2013. Elysium Solutions Oy. Taloussanomat [viitattu 2013-01-28]. Saatavissa: http://yritys.taloussanomat.fi/y/elysium-solutions-oy/kuopio/2411665-7/.

TONU, M A Hossain. 2012. PHP Application Development with NetBeans Beginner's. Birmingham: Packt Publishing.

ULLMAN, Larry. 2012. Modern JavaScript: Develop and Design. Berkley: Peachpit Press.

W3SCHOOLS, 2012. PHP Tutorial – Introduction. W3schools [viitattu 2013-01-15]. Saatavissa: http://www.w3schools.in/php-tutorial/intro/.

YANK, Kevin. 2012. PHP & MySQL: Novice to Ninja, Fifth Edition. Collingwood: SitePoint.