HAAGA-HELIA
ammattikorkeakoulu

Web-pohjaisen käyttöliittymän uudistaminen portaalimalliin

Jani Hurme

Opinnäytetyö

Tietotekniikan koulutusohjelma

2013

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

1.5.2013

Tietotekniikan koulutusohjelma

Tekijä Jani Hurme	**Ryhmätunnus tai aloitusvuosi** 2007
Raportin nimi Web-pohjaisen käyttöliittymän uudistaminen portaalimalliin	**Sivujen ja liitteiden määrä** 27
Opettajat tai ohjaajat Marttila Sirpa	

Tässä raportissa kuvataan web-pohjaisen tietojärjestelmäuudistuksen toteuttaminen käyttöliittymätasolla. Toteutusmalli on pitkälti portaalimallin mukainen. Valmista portaalituotetta ei ole kuitenkaan hyödynnetty, vaan toteutus on tehty itsenäiseksi web-sovellukseksi. Tarve uudistukselle syntyi entisen järjestelmän elinkaaren umpeuduttua. Tavoitteena tässä vaiheessa on saada ulkoasultaan entistä vastaava järjestelmä käyttöön siten, että toteutustapa vastaa nykypäivän tekniikkaa ja siten mahdollistaa erilaisia jatkokehityshankkeita tulevaisuudessa. Järjestelmän tulee olla myös skaalautuva, eli käyttäjämäärän lisääntyessä sen tulee kestää enemmän kuormaa hidastumatta.

Tämä uudistus on osa laajempaa hanketta jossa uusitaan toimeksiantajan kaikki järjestelmän osat fyysisistä laitteista lähtien. Hankkeen muita osioita ei toteuteta tämän osaprojektin puitteissa, joskin tämän projektin puitteissa toteutettavan käyttöliittymän tulee integroitua saumattomasti yhteen niin ajoalustan kuin muidenkin osaprojektien kanssa (kuormanjako-ohjelmisto, käyttäjän tunnistus, käyttöoikeudet, rajapinnat taustajärjestelmiin).

Projektin tekemisessä on noudatettu toimeksiantajan projektityömallia ja työ on tehty toimeksiantajan osoittamissa tiloissa. Projektin toteutusvaihe aloitettiin keväällä 2012 ja sitä jatkettiin kesälomien jälkeen aina vuoden loppuun asti. Ensimmäisiä osioita päästiin testaamaan vuodenvaihteessa, jolloin opinnäytetyön puitteissa suoritettava osuus oli valmis. Projektia ei ole ollut mahdollista tehdä täysipäiväisesti vaan sitä on tehty muiden töiden ohessa.

Työn suorittamiseksi valitut toimintatavat, ohjelmistot ja välineet olivat projektiryhmälle ennestään tuttuja, minkä takia työ eteni ilman suurempia esteitä. Lopputulos vastaa tavoitteita, joskin opinnäytetyön puitteissa käytetty työmäärä ei täysin riittänyt valmiin sovelluksen aikaansaamiseksi. Käytännössä testauksen puitteissa löydettyjen puutteiden korjaaminen jäi tämän projektin ulkopuolelle.

Asiasanat
Spring, Java, käyttöliittymä, portaali

HAAGA-HELIA
ammattikorkeakoulu

Abstract

1.5.2013

Degree Program in Information Technology

Author Jani Hurme	**Group or year of entry** 2007
The title of thesis Implementation a web based user interface as a portal	**Number of pages and attachments** 27
Supervisor(s) Marttila Sirpa	

The purpose of this thesis was to demonstrate the implementation of a web-based user interface. The implementation model was based on portal principles. However, no third party software was used as a portal framework.

The client company had used legacy software as the user interface for their web users. The current implementation's lifecycle had come to an end, so the first step was to develop new software to serve active users better. At this point, there was no need for a new layout, but a more important matter was to provide certain functionalities for the system to be used in the future. The new web-based user interface had to be scalable, so it would not slow down should the number of users increase in the future.

This thesis project was a part of a larger project which includes hardware and middleware replacement. Other co-projects were not included in this project but the new user interface, nevertheless, had to be compatible with the other implementations of the project (load-balancing, platform, user identification and authorization, communication to legacy systems).

The guidelines for project were provided by the client. The project team worked in the client's office premises. The project started in the spring of 2012 and continued after summer vacations in the autumn until the end of 2012. A test group started testing in December 2012 when the first phase of the project was done.

There were no significant issues or show-stoppers during the implementation of the project. The environment for middleware had already been set. Spring framework, JBoss EAP 6 as a server and RHEL as a platform had been tested during the Proof of concept phase earlier.

The results of this thesis project met the requirements set for it despite the fact that the project still continues after the first phase. All the bugs reported by the testing group were not fixed within this thesis project due to scheduling issues.

Key words
Spring, java, user interface, portal

Sisällys

1 Johdanto 2
2 Taustatietoa 3
2.1 Java ohjelmointikieli 5
2.2 Java virtuaalikoneen toimintaperiaate 6
2.3 JVM:n muistinkäytön perusteista 7
2.4 Spring framework 8
3 Järjestelmän tekninen kuvaus 10
3.1 JSTL eli tagikirjastot 13
3.2 Tunnistuspalveluiden sekä käyttöoikeuksien integrointi 15
3.3 Kutsurajapinta taustajärjestelmiin 18
3.4 Ulkoasu 19
3.5 Git versionhallinta 20
3.6 Projektin rakenne ja toiminta 21
3.7 Tietoturva 23
4 Yhteenveto ja johtopäätökset 25
4.1. Johtopäätökset 27

1 Johdanto

Opinnäytetyön tavoitteena on uudistaa toimeksiantajan sopimusasiakkaille suunnattu web-käyttöliittymä. Sen käyttäminen edellyttää toimeksiantajan ja asiakkaan välistä kirjallista sopimusta ja käyttäjien tunnistamista joka kerta kun he järjestelmää käyttävät. Sopimuksen allekirjoituksen jälkeen käyttäjät saavat käyttöönsä henkilökohtaiset käyttäjätunnukset joilla he voivat kirjautua järjestelmään. Käyttöliittymän pääasiallisena tarkoituksena on tarjota käyttäjille erilaisia raportteja ja muuta informaatiota luottopäätösten ja riskienhallinnan päätöksenteon tueksi. Uudistettavan järjestelmän käyttäjäjoukko on sopimuskäytännön takia rajattu eikä sinne ole avointa pääsyä kaikilla internetin käyttäjillä. Yhtäaikaisten käyttäjien arvioitu lukumäärä on 1.500 istuntoa vilkkaimpina aikoina vuorokaudesta.

Tämän opinnäytetyön puitteissa suoritettava käyttöliittymän uudistaminen on osa toimeksiantajan laajempaa projektikokonaisuutta. Työ on aloitettu jo vuonna 2010 laite- ja verkkoinfrastruktuurin osalta, jonka jälkeen on edetty käyttöjärjestelmien sekä virtualisoitujen alustojen ja sovelluspalvelinten asennuksiin sekä niiden käyttöönottoihin. Kokonaisarkkitehtuurin valmistuttua on voitu edetä yksittäisiin osa-alueisiin. Käyttöliittymäuudistus on saatu käyntiin keväällä 2012 PoC –vaiheen (Proof of Concept) jälkeen.

Tietojärjestelmien lyhyehköjen elinkaarien takia uudistuksia tehdään jatkuvasti niin laite- kuin ohjelmistopuolellakin säännöllisin väliajoin. Näin toimimalla voidaan vastata loppukäyttäjien odotuksiin korkeasta palvelutasosta ja hyvästä käyttäjäkokemuksesta. Lisäksi laite- ja ohjelmistotoimittajien tuki eri tuotteille loppuu ennalta määrätyn elinkaaren jälkeen eikä kaupallisesti toimivilla yrityksillä ole varaa ottaa sitä riskiä että järjestelmän jonkun osan pettäessä tukea ei olisi saatavilla. Tällä hetkellä toimeksiantajan käytössä olevat järjestelmät ovat osin jo yli kymmenen vuotta vanhoja, joten niiden uudistamiselle on hyvät perusteet. Toimeksiantajan liiketoiminnan kannalta tämä järjestelmäuudistus on kriittinen, sillä liikevaihdosta yli 90% tulee internetin kautta joko koneellisina kyselytapahtumina tai varsinaisten loppukäyttäjien toimesta.

Koska uudistusprojekti on kokonaisuudessaan iso ja laaja hanke, on tämän opinnäytetyön puitteissa tapahtuva työ rajattu siten että siihen sisältyy ainoastaan käyttöliittymä-

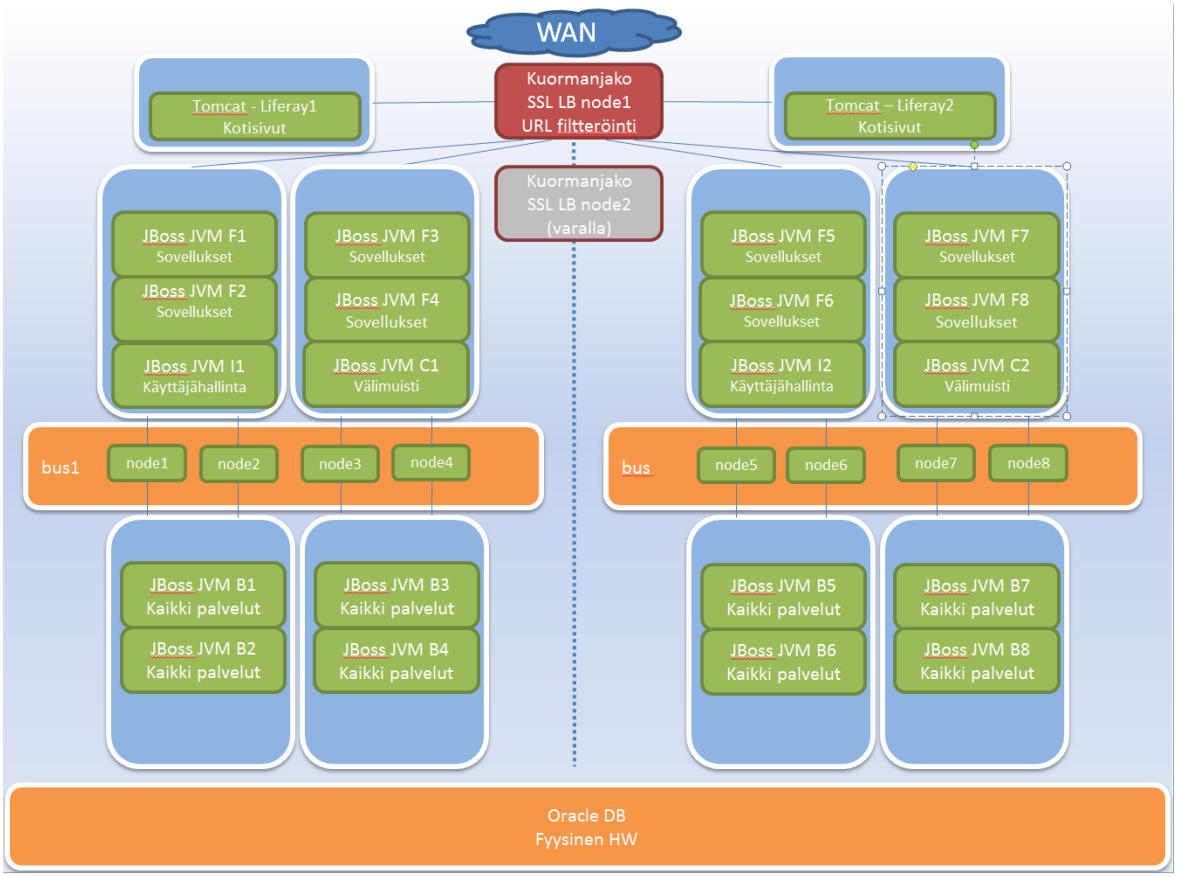
sovelluksen määrittely-, suunnittelu sekä toteutustyö. Ulkoasultaan uudistettava käyttöliittymä toteutetaan tässä vaiheessa nykyisen käyttöliittymän kanssa yhdennäköiseksi. Asiakkaille tarjottava tietosisältö saadaan haettua http-pohjaisen kutsurajapinnan kautta kuten aiemminkin. Määrittelyosuus pohjautuu tässä tapauksessa pitkälti nykyisen käyttöliittymän liiketoimintasääntöihin ja muihin kuvauksiin, mutta suunnittelu ja toteutustyö tehdään uusin ja nykyaikaisin menetelmin. Työn tulokset testataan erillisen testausryhmän toimesta. Myös käyttäjän tunnistaminen ja käyttöoikeuksien tarkistaminen on erillinen kokonaisuus eikä sitä toteuteta tässä osaprojektissa. Tunnistuspalvelut ovat kuitenkin tärkeässä roolissa käyttöliittymän osalta ja ne tullaan integroimaan osaksi käyttöliittymätoimintoja projektin edetessä.

2 Taustatietoa

Käytössä oleva järjestelmä on todettu toimeksiantajan ylimmän johdon toimesta riskitekijäksi yrityksen strategiassa, koska nykyinen järjestelmätoimittaja on pieni yritys ja muutokset siihen vaativat aina ulkopuolisia resursseja avuksi. Lisäksi toteutus ei vastaa kaikilta osin nykypäivän vaatimuksia käytettävyydeltään ja toimintavarmuudeltaan. Uudistuksella tähdätään järjestelmän omatoimiseen hallinnointiin ja kehittämiseen toimittajariskin minimoimiseksi sekä toimintavarmuuden parantamiseen käyttöasteen ennakoidun kasvun takia. Kokonaisprojekti viedään läpi pääosin toimeksiantajan oman IT-osaston muodostaman projektiryhmän toimesta. Ulkopuolisia asiantuntijoita hyödynnetään tarvittaessa.

Toimeksiantajan ohjelmistoarkkitehtuuri perustuu monikerrosarkkitehtuuriin sekä korkeaan skaalautuvuuteen. Tämä osaltaan asettaa tietyt raamit myös tehtäessä käyttöliittymätason uudistusta. Kuormanjako, pyyntöjen reititys sekä staattisten resurssien hallinnointi suoritetaan ennen käyttöliittymäkerrosta ja vastaavasti tietosisällön hakeminen tapahtuu kutsumalla alempaa kerrosta (liiketoimintalogiikka). Alempi kerros on puolestaan yhteydessä toimeksiantajan omiin tietokantoihin tai edelleen muihin järjestelmiin. Virtualisoitu toimintaympäristö on nykypäivän käytäntö helpon skaalautuvuuden takia. Resursseja on näin ollen helppo lisätä käyttötarpeen mukaisesti. Toimeksiantajan järjestelmässä ei kuitenkaan ole tarkoitus ajaa palvelimilla olevia sovelluksia rinnakkain klusteroituna siten että sessiodata replikoituisi eri palvelinten välillä. Sen sijaan eri arkkiteh-

tuuritasojen välille muodostetaan useita identtisiä palveluväyliä, joita pitkin pystytään käyttäjien tekemät palvelupyynnöt suorittamaan. Pyynnön tullessa ylimmälle tasolle järjestelmässä se ohjataan kuormanjakolaitteesta käsin samaa palveluväylää pitkin aina alimmalle tasolle asti. Käyttäjän yhden istunnon aikana suorittamat pyynnöt ohjataan aina samaan palveluväylään sticky-session periaatteella. Näin ollen päivityksiä sekä huoltotoimenpiteitä voidaan tehdä yksittäisiin palveluväyliin ilman erillisiä käyttökatkoja sulkemalla aina yksi tai useampi palveluväylä kerrallaan.



Kuva 1 – Toimintaympäristön arkkitehtuurikuvaus

Siniset komponentit ovat virtualisoituja alustoja, oranssit fyysisiä laitteita

Koska uudistuksen kohteena on yrityksen näkökulmasta kriittinen osa-alue, ratkaisut käytettävistä komponenteista on tehty pitkäjänteisen prosessin kautta. Toimeksiantajan käytäntöjen mukaisesti suuremmista linjauksista päätetään arkkitehtuuriryhmässä. Päätöksenteon pohjaksi esitykset vaihtoehtoisista toteutusmalleista valmistellaan asiantuntijoiden toimesta etukäteen. Käyttöliittymäkerroksen osalta on teetetty kattava selvitys eri vaihtoehtojen sopivuudesta ja ominaisuuksista. Lopullisessa päätöksessä ei päädytty

valmisohjelmistoon vaan päätettiin toteuttaa oma web-pohjainen sovellus, joka tukee portaaliajattelua upotettavien ohjelmistokomponenttien sekä mahdollisimman joustavan sisällönhallinnan osalta.

2.1 Java ohjelmointikieli

Java ohjelmointikieli on alun perin kehitetty 1990-luvun alussa Sun Microsystemsillä (mm. Bill Joy, James Gosling). Kehitystyön tuloksena syntyi versio 1.0, joka julkaistiin 1995 (Wikipedia 2013 Java). Tämän jälkeen Javan suosio on ollut nousujohteista aina tähän päivään asti. Monien mielestä syntaksi on helposti omaksuttavaa ja oliopohjainen ajattelutapa on myös osoittautunut suosituksi kehittäjien keskuudessa. Lisäksi Java tarjoaa laajan ja käyttövalmiin kirjastokokoelman, joka sisältää paljon erilaisia yleiskäyttöisiä valmiita rutiineja ja toimintoja kehittäjien avuksi. Yleiskäyttöiset perusrutiinit nopeuttavat kehitystyötä kun niitä ei tarvitse joka kerta tehdä uudelleen. Useilla apukirjastoilla on kuitenkin riippuvuuksia toisiin kirjastoihin ja monesti haittapuoleksi saattaa muodostua monimutkainen riippuvuussuhdeviidakko eri kirjastojen välillä. Pienetkin sovellukset saattavat paisua monikymmenmegaisiksi paketeiksi, jotka sitten jättävät jälkeensä isomman muistijäljen (memory footprint).

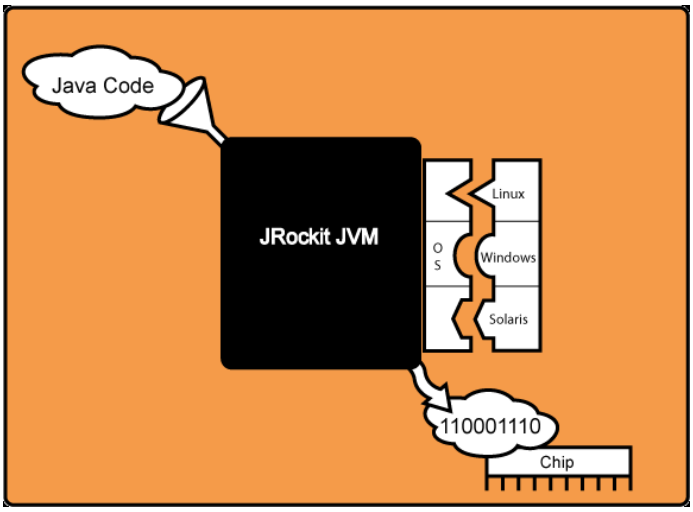
Alun perin Java oli suosituinta web-ohjelmien toteutuksessa (mm. appletit). Sen jälkeen web-toteutustapojen kirjo on laajentunut huomattavasti ja Java puolestaan on laajentanut suosiotaan palvelinohjelmistojen sekä mobiilipuolen tuotekehityksessä sekä erilaisissa sulautetuissa järjestelmissä. Merkittävä etu Javan käytössä on alustariippumattomuus eli sama koodi ajautuu käyttöjärjestelmästä tai esim. sovelluspalvelimesta riippumatta. Monet ohjelmistotalot ovat rakentaneet alkuperäisen Javan päälle omia toteutuksiaan edelleen helpottamaan ja nopeuttamaan sovelluskehitystä omat tarpeensa huomioiden. Kehittäjän kannalta tässä on haittapuolena sitoutuminen tietyn ohjelmistotalon tuotteisiin. Jos tällainen valinta tehdään, niin konvertoiminen takaisin standardin mukaiseen alkuperäiseen Javaan voi jälkikäteen olla varsin työläs homma.

Tämän projektin toteutusympäristön käytössä olevat sovelluspalvelimet (JBoss EAP 6.0) tukevat Java6 EE –pohjaisia sovelluksia (JBoss Community. JBoss Application Server 2012). Lisäksi toimeksiantajan aiemmissa projekteissa java on ollut selvästi käytetyin oh-

jelmointikieli joten valinta tässä kohdin oli helppo. Javasta olisi ollut saatavilla uudempikin versio (7), joka on julkaistu heinäkuussa 2012 (Oracle docs 2013. The Java EE 6 Tutorial). Se ei ole vielä tuettuna useimmissa sovelluspalvelinympäristöissä. Lisäksi uusimpien julkaisuiden nopeaan käyttöönottoon sisältyy aina riski niiden mahdollisten puutteiden osalta. Kun jokin ohjelmisto on ollut maailmanlaajuisesti yleisessä käytössä jonkin aikaa, siitä on yleisimmät viat ja puutteet jo havaittu sekä mahdollisesti jo korjattukin.

2.2 Java virtuaalikoneen toimintaperiaate

Virtuaalikone on keskeinen osa Java-ympäristöä. Sen vastuulla on ohjelmien suorittaminen käytetystä lähdekoodikielestä riippumatta. Lähdekoodi käännetään ensin niin sanotuksi byte-koodiksi jota virtuaalikone ymmärtää. Näin ollen lähdekoodin ei tarvitse välttämättä olla kirjoitettu java ohjelmointikielellä, vaan riittää että lähdekoodi on käännetty virtuaalikoneen ymmärtämään bytecode-muotoon ennen ohjelman suoritusta.



Kuva 2 Jrockit JVM toimintaperiaate (Oracle 2009. JRockit JVM Diagnostics guide)

Lähdekoodi kirjoitetaan javassa .java –tiedostoihin ja käännetään kääntäjän avulla sen jälkeen .class –tiedostoiksi, joita JVM osaa suorittaa. Ajettavat tiedostot voidaan myös pakata .jar –tiedostoiksi (java archive) eli paketeiksi (Oracle 2009. JRockit JVM Diagnostics guide).

Yksi tärkeimmistä perusperiaatteista virtuaalikoneessa on sen toimintavarmuus siten, että ajettavat sovellukset eivät pysty sen toimintaa keskeyttämään esimerkiksi ohjelmointivirheitä tekemällä. Toinen tässä yhteydessä mainitsemisen arvoinen asia on toimintaperiaate, jonka mukaan kerran kirjoitettu ja käännetty lähdekoodi on ajettavissa

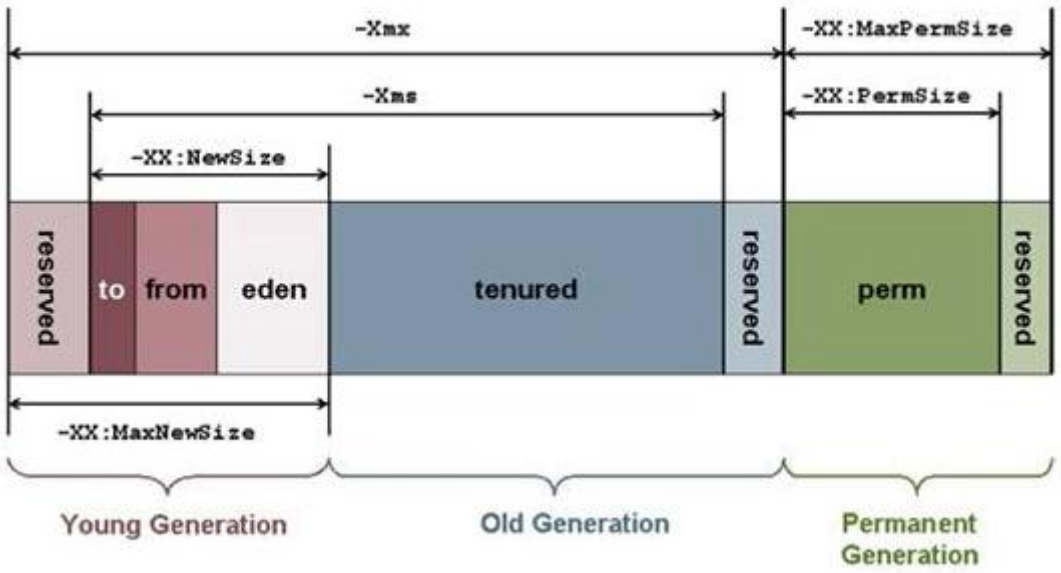
eri virtuaalikoneissa käyttöjärjestelmästä riippumatta. Myös Java ohjelmointikielestä julkaistavat uudet versiot ovat alaspäin yhteensopivia eli esim. 1.4 versiolla tehdyt ohjelmat toimivat 1.6 ympäristössä.

2.3 JVM:n muistinkäytön perusteista

Javan peruskehittäjäpaketti (JDK eli Java Development Kit) on ilmainen ja ajettavia ohjelmia voi tehdä pelkällä tekstieditorilla ja niitä voi ajaa suoraan komentorivikomennolla. Tähänkin on haettu nopeuttavia ja ohjelmointityötä helpottavia ratkaisua niin kehittäjäyhteisöjen kuin ohjelmistotalojenkin osalta; IDE:t eli kehittimet osaavat nykypäivänä ilmoittaa jo lähdekoodin kirjoittamisen aikana esiintyvät virheet eikä koodin kääntämistä enää tarvitse ensin yrittää virheiden havaitsemiseksi.

Yksi valmis ominaisuus Javan peruspaketissa on muistinhallinnan mahdollistamiseksi tehty GarbageCollector eli roskienkerääjä. Tällä on pyritty siihen, ettei kehittäjän tarvitse erikseen huolehtia muistin vapauttamisesta vaan ohjelmointialusta huolehtii asiasta automaattisesti. Eri JVM:ien välillä roskienkeruumekanismeissakin on merkittäviä eroja, joskin niiden perusperiaate on sama. Mekanismien osalta valittavissa on useita strategioita muistinhallinnan kontrolloimiseksi ja lisäksi eri strategioille on määritettävissä kymmeniä eri optioita.

Virtuaalikoneen muistialue jaetaan niin sanottuihin sukupolvialueisiin (Nyberg, Gregory & Patrick, Robert. 2003). Uudet vastaluodut objektit sijoitetaan nuorten objektien muistialueelle, josta ne joko tuhotaan tai säilytetään myöhempää käyttöä varten. Objekti voidaan tuhota muistista, mikäli siihen ei ole olemassa enää viittauksia mistään ajettavan ohjelman osasta. Mikäli tällainen viittaus jää ohjelmoijalta poistamatta, objekti saattaa turhaan roikkua muistissa ja varata tilaa sieltä. Näin ollen objektien elinkaaren määrittäminen on varsin oleellista ohjelmoijan kannalta. Hyvin suunnitellut ja toteutetut järjestelmät osaavat ottaa huomioon objektien vaaditun elinkaaren jolloin ne tuhotaan välittömästi kun niitä ei enää tarvita.



Kuva 3 – JVM muistialueen jakautuminen (Visualising Garbage Collection in the JVM 2012).

2.4 Spring framework

Spring framework on open souce –pohjainen Java-laajennos joka koostuu useista apukirjastoista. Sen tarkoituksena on helpottaa ja nopeuttaa varsinaista ohjelmointityötä.

Käytännössä tämä tapahtuu XML-pohjaisten konfiguraatiotiedostojen sekä ohjelmakoodiin upotettavien annotaatioiden avulla. Apukirjastot pakataan mukaan sovellukseen tai luetaan sovelluspalvelimen luokkapolusta JVM:n käynnistysvaiheessa.
Vain osaa Spring-apukirjastoista voidaan käyttää tarpeen mukaan, joskin kirjastojen kesken on joitakin riippuvuuksia. Tämän projektin osalta Spring frameworkin käyttö rajoittui Web- ja Security-kirjastoihin. Koko Spring framework –laajennoksen käytetyimmät osat ovat:

- Spring Security
- Spring Integration
- Spring Batch
- Spring Roo
- Spring Data
- Spring Web Flow
- Spring Web Services

- Spring Mobile
- Spring Social

Sovelluspalvelimien osalta Spring framework integroituu yleisimpiin käytössä oleviin, jotka tukevat Java EE:tä. Yleisimmin käytetty on Apache Tomcat.
(Spring Community 2013)

Ensimmäisen version Spring frameworkista teki Rod Johnson ja se julkaistiin vuonna 2003. Kehitystyö on siitä lähtien jatkunut aktiivisesti ja viimeisin versio (3.2.2) julkaistiin maaliskuussa 2013 ja version 4 odotetaan valmistuvan loppuvuonna 2013. Spring framework on vuosien saatossa saavuttanut laajan suosion Java-ohjelmistokäyttäjien keskuudessa (Wikipedia 2013 Spring Framework 2013).

Spring frameworkin taustalle on muodustunut varsin laaja käyttäjäyhteisö. Yhteisö ylläpitää nettisivustoa osoitteessa http://www.springsource.org

3 Järjestelmän tekninen kuvaus

Käyttöliittymätason uudistusprojektin alkaessa toimintaympäristö oli valmiiksi rakennettu ja se koostuu seuraavista komponenteista ja ohjelmistoista:

- käyttöjärjestelmä Red Hat Enterprise Linux 2.6.32-220.2.1.el6.x86_64 (64-bit)
- sovelluspalvelin JBoss Enterprise Application Server 6.0
- ohjelmointialustana Java Development Kit 6 (jdk 1.6.0_29)
- käyttöliittymäframework Spring 3.1.0
- web-kirjastokokoelma Java Standard Tag Library (JSTL 2.0)

Toteutustyössä käytettävä kehitysympäristö puolestaan koostuu seuraavista ohjelmistoista ja työvälineistä:

- käyttöjärjestelmä Windows7 työasema
- kehitin Eclipse Java EE IDE for Web Developers Indigo Service Release 2
- sovelluspalvelin JBoss Enterprise Application Server 6.0
- paketointiohjelmisto Apache Maven 3.0.3 (r1075438; 2011-02-28 19:31:09+0200)
- versionhallinta GIT version 1.7.7.1.msysgit.0
- projektinhallinta ja dokumentointi JIRA, Confluence
- testauksessa käytettävä raportointi Bugzilla

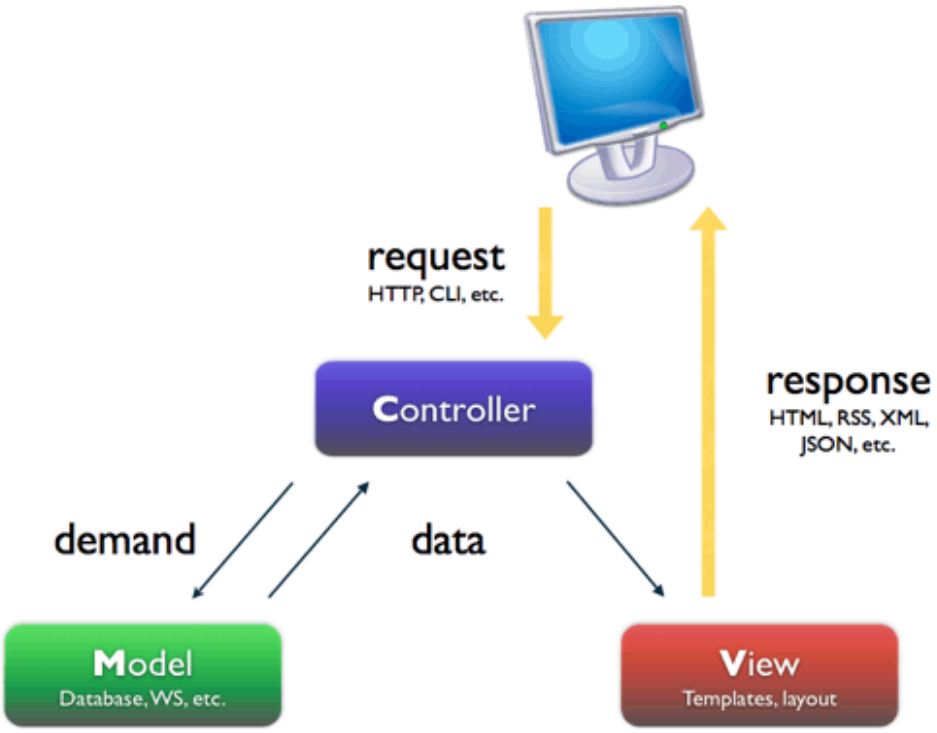
Kehittäjät työstävät järjestelmän osia omilla henkilökohtaisilla työasemillaan josta ne voidaan siirtää testiympäristöön. Testiympäristössä sovellukset ovat muiden kehittäjien ja testaajien saatavilla. Testiympäristö on suljettu ympäristö eikä sinne ole pääsyä ulkopuolisista tietoverkoista. Vasta valmiit ja hyväksytyt versiot ohjelmistoista voidaan siirtää tuotantoympäristöön, josta ne ovat loppukäyttäjien saatavilla.

Toimeksiantajan arkkitehtuuriryhmä ei ottanut kantaa tämän opinnäytetyön puitteissa tehtävän käyttöliittymäuudistuksen valintoihin eri toteutustapojen osalta. Ratkaisut käytettävien ohjelmakomponenttien, laajennosten sekä apukirjastojen suhteen voitiin tehdä projektiryhmän toimesta. Valinnoissa kiinnitettiin huomiota seuraaviin seikkoihin:

- yhteensopivuus JBoss EAP 6.0 -alustalle
- yhdenmukaisuus yrityksessä tehtyjen teknisten ratkaisujen osalta
- valittujen ratkaisuiden oltava riittävän yksinkertaisia ja helposti ylläpidettäviä
- liian laajoja ja paljon muistia vieviä ratkaisuita vältettävä
- käytettävien komponenttien lisensiointi oltava open source -mallinen

Käyttöliittymän uudistusprojektissa pohdittiin laajemmalla ryhmällä valmiin portaalituotteen käyttämistä. Pohdintaan osallistuneen ryhmän toimesta teetettiin selvitys portaalituotteista talon ulkopuolisella toimijalla. Mukaan selvitykseen valittiin Oraclen Web Center, Liferay (open source) ja JBoss Seam. Näistä parhaimman arvion sai Liferay, mutta huolimatta kattavasta raportista ryhmän näkemys tuki kuitenkin enemmän omatoimisesti rakennettua ratkaisua kuin valmistuotetta. Tämä näkemys sai myös arkkitehtuuriryhmän hyväksynnän, joten suunnitelmat portaalituotteen käytöstä voitiin hylätä.

Näiden päälinjausten jälkeen tein valinnat muiden hyödynnettävien komponenttien osalta omien kokemusten ja selvitysten perusteella. Spring web-frameworkin osalta toteutustapa oli hyvin samankaltainen kuin yrityksessä aiemmin käytetty Oracle-pohjainen ratkaisu, joka osaltaan vaikutti sen valitsemisen puolesta. Molemmat perustuvat yleisesti käytössä olevaan MVC (Model – View – Controller) –malliin.
Tutustuin myös muihin vaihtoehtoihin kuten Apache Wicket, Vaadin, Google Web Toolkit, Tapestry
(Raible, Matt - Comparing JVM Web frameworks, 2010)
(Apache Software foundation. Apache Wicket)
(Apache Software foundation. Apache Tapestry)



Kuva 4 – Model-View-Controller malli (Developpez.com 2013)

MVC-pohjaisessa Spring Framework ratkaisussa palvelinpuolen koodi käsittää olennaisena osana Controller-luokat, jotka voidaan määritellä sovellusten web.xml.ssä latautumaan automaattisesti.

Esimerkki määrityksestä controller luokan latautumisesta:
<context:component-scan base-package="fi.*yrityksen_nimi*.sovellus.controllers"/>

Controller –luokat ottavat käyttäjien palvelupyynnöt (GET ja POST) vastaan ja käsittelevät ne ja ohjaavat pyynnöt eteenpäin. Vastaanotettavat pyynnöt määritellään controller-luokkiin annotaatioiden avulla REST-pohjaisesti. Esimerkiksi toiminto, jolla halutaan hakea tietosisältöä taustajärjestelmistä, voidaan määritellä näin:
@RequestMethod.GET ”/haku”

return ”etusivu”

Tämä suorittaa haun ja palauttaa käyttäjän määritellylle jsp-sivulle (etusivu.jsp).

Annotaatiopohjaisella määrityksellä kerrotaan myös attribuutit, jotka halutaan pitää saatavilla koko käyttäjän istunnon ajan:

```
@Controller
@SessionAttributes({"xxx","yyy"})
```

3.1 JSTL eli tagikirjastot

Kun käyttöliittymätasolla on tarve tuottaa dynaamista sisältöä web-pohjaisissa sovelluksissa, pelkkä HTML harvoin tarjoaa riittävät toimintaedellytykset jotta kaikki halutut piirteet voitaisiin toteuttaa. Java Server Pages (JSP) on yleisesti käytössä oleva tekniikka, joka tarjoaa mahdollisuuden hyödyntää monipuolisia toimintoja esityskerroksessa. Modernimpi ratkaisu tältä pohjalta on Java Server Faces (JSF). Kolmantena vaihtoehtona nousi esiin XSLT eli xml-pohjainen HTML-laajennos.

Oman haasteensa tietojen esittämiselle toi taustajärjestelmien tuottama tietorakenne, joka oli monelta osin varsin monimutkainen ja vaikeasti tulkittava. Pyrkimyksenä oli kuitenkin datan mahdollisimman minimaalinen käsittely, jotta vasteajat loppukäyttäjien pyynnöille jäisivät kohtuullisiksi eikä muistia tarvitsisi turhaan kuormittaa uusien objektien generoinnilla. Päädyin ratkaisussa JSP-vaihtoehtoon hyödyntäen JSTL (Java Standard Tag library) kirjastokokoelmaa, joka tarjoaa JSP-sivuille kattavat lisätoiminnot sisällön monimuotoiselle esittämiselle. JSTL-kirjastoja voidaan määritellä jokaisen jsp-sivun alkuun tarpeen mukaan seuraavasti:

```
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@taglib prefix="c"   uri="http://java.sun.com/jsp/jstl/core" %>
<%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

Tässä yhteydessä toteutetun sovelluksen osalta tärkeimmät JSTL:n tarjoamat toiminnot olivat <choose-when-otherwise> -rakenne, <if> -ehtolause, fn-funktiokirjaston eri toiminnot kuten fn:contains, fn:trim, fn:length jne.
Esimerkiksi <choose-when-otherwise> -rakenne on käyttökelpoinen jos halutaan kertoa vaikka nimitieto tai vaihtoehtoisesti ilmoittaa että sellaista ei ole:

```
<c:choose>
  <c:when test="${perustiedot != null}">
    <p>Nimi: ${perustiedot.nimi}</p>
  </c:when>
  <c:otherwise>
    <p>Nimeä ei ole saatavilla!</p>
  </c:otherwise>
</c:choose>
```

Kaikki käyttöliittymän tekstit sijoitettiin erilliseen tekstitiedostoon, jotta ne olisivat yhdessä paikassa helposti ylläpidettävissä. Lisäksi tämä toimintatapa mahdollistaa kieliversioiden lisäämisen kätevästi. Jotta tällaista tekstitiedostoa (resource bundle) voidaan käyttää, tulee se määritellä JSP-sivulle seuraavasti:

```
<fmt:bundle basename="tiedoston_nimi_resource_bundle">
```

Tämän jälkeen tiedoston yksittäiseen tekstiin voidaan viitata <fmt:message> tagin avulla seuraavasti (key on tunniste jonka avulla oikea tekstinpätkä saadaan haettua):

```
<fmt:message key="perustiedot_nimiteksti" />
```

Javabean on luokka, johon voidaan tallentaa jokin looginen kokonaisuus attribuutteineen eli ominaisuuksineen. Yleisesti JavaBeaneistä käytetään myös nimitystä POJO (Plain Old Java Object) –luokka (Wikipedia 2013). POJO -luokkiin tallennettujen muuttujien vertailu ja näiden perusteella esitettävien näkymien päätteleminen muodostui paljon käytetyksi toiminnoksi varsinkin <c:if> -rakenteiden kanssa (equals ja equals not). Myös erilaisten taulukoiden läpikäynti ja tiedon esittäminen niiden avulla on yleinen toiminto. Tämän mahdollisti <forEach> -rakenne. Esimerkiksi lista toimihenkiloista voidaan tulostaa sivulle toimihenkilon tyypin perusteella seuraavasti:

```
<c:forEach var="toimihenkilo" items="${perustiedot.toimihenkilot}">
  <c:if test="${toimihenkilo.tyyppi eq '1'}">
    <fmt:message key="perustiedot_nimiteksti" />
    <c:out value="${toimihenkilo.nimi}"></c:out>
  </c:if>
```

</c:forEach>

equals –määreen syntaksi on 'eq' ja vastaavasti not equals –määreen syntaksi on 'ne'.

Päivämäärien ja numeeristen arvojen esittäminen onnistui <fmt:parseDate >, <fmt:formatDate> ja <fmt:formatNumber> tagien avulla seuraavasti:

<fmt:parseDate var="pvm" value="${perustiedot.muutospvm}" type="DATE" pattern="ddMMyyyy"/>
<fmt:formatDate value="${pvm}" pattern="dd.MM.yyyy"/>

Päivämäärän konvertointi ja esittäminen muodosta 'päivätkuukaudetvuodet' muotoon 'päivät.kuukaudet.vuodet'. Muita esimerkkejä tärkeimmistä JSTL-tageista:

Muuttujan asettaminen:

<c:set var="osakkuustyyppi" value="${kaikki_osakkaat.tyyppi}" />

<fn:length ja fn:trim> käyttö:

<c:if test="${fn:length(fn:trim(toimihenkilot_lkm)) > 0}">

<fmt:formatNumber> käyttö numeeristen arvojen esittämisessä:

<fmt:formatNumber value="${ toimihenkilot_lkm}"/>

3.2 Tunnistuspalveluiden sekä käyttöoikeuksien integrointi

Rinnakkaisena osaprojektina toteutettu tunnistuspalveluiden ja käyttöoikeuksien hallinnointiprojekti toteutettiin pääasiallisesti ulkopuolisten konsulttien toimesta. Käyttäjät ja heidän käyttöoikeutensa olivat jo valmiina tietokannassa ja tätä ei lähdetty muuttamaan. Sen sijaan siitä ylöspäin käyttäjähallinta päätettiin uudistaa. Projektiryhmä päätti edetä käyttäjien provisioinnilla erilliseen LDAP-tietokantaan, josta OpenAM –niminen open source-ohjelmisto voisi hyödyntää käyttäjätietoja. Käyttöliittymän ja OpenAM:n yhteistoiminnan mahdollistamiseksi sovittiin rajapinnaksi Spring-laajennoksen security-rajapinta, jonka kautta käyttöliittymä voi rajata yksittäiselle käyttäjälle näkyviä toimintoja. Tämä tietysti edellyttää ensin käyttäjän sisäänkirjautumista järjestelmään.

Web-pohjaisissa java –käyttöliittymäsovelluksissa voidaan tehdä sovelluskohtaisia määrityksiä web.xml –nimisessä konfiguraatiotiedostossa. Nimensä mukaisesti se on xml-pohjainen kuvaustiedosto, jossa tyypillisesti määritellään ainakin seuraavat asiat sovelluksen toiminnan osalta:

```
<servlet>
  <servlet-name>sovelluksen_nimi</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/context.xml</param-value>
  </init-param>
  <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>sovelluksen_nimi</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Yksi web.xml –määritystiedoston perustoiminnoista on erilaisten filtterien määrittäminen. Tämä tarkoittaa sitä, että käyttäjien lähettäessä pyyntöjä järjestelmälle niille voidaan tehdä erilaisia tarkistuksia tai edelleenohjauksia. Tässä projektissa järjestelmään pääsyn kontrolloimiseksi sovellukselle määritettiin filtteri, jonka tehtävänä on aina tarkistaa että käyttäjä on kirjautunut ennen kuin haluttu pyyntö voidaan toteuttaa.

Filtteri voidaan määrittää kaikille tuleville pyynnöille tai kohdistaa vain tiettyyn osaan järjestelmää tuleviin pyyntöihin. Tässä tapauksessa halutaan varmistaa käyttöoikeus jokaisen pyynnön osalta, joten url-pohjainen filtteri määritettiin seuraavasti:

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
```

```
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Filter-name edustaa tässä filtterin nimeä ja filter-class kertoo suorittavan filtterin sijainnin java-luokkahierarkian sisällä. Url-pattern määritys puolestaan kertoo mille osalle järjestelmään kohdistuvista pyynnöistä toimenpiteet suoritetaan (/* -määritys tarkoittaa kaikkia pyyntöjä).

OpenAM –integroinnin osalta tämä ei vielä riitä, vaan yksityiskohtaisemmat määrittelyt pitää suorittaa security-context-OpenAM.xml –tiedostossa. Siinä otetaan kantaa mm. seuraaviin asioihin:

- mihin käyttäjä ohjataan tunnistautumista varten (url)
- mihin käyttäjä ohjataan jos tunnistautuminen epäonnistuu
- mihin käyttäjä ohjataan onnistuneen sisäänkirjautumisen seurauksena

Käyttöoikeuksien välittäminen web-sovellukselle toteutetaan Spring security-kirjastokokoelman avulla. Käyttöliittymän näkymät (view) on toteutettu JSP-tekniikalla (Java Server Pages). Viittaamalla jsp-sivun alussa Spring security –tagikokoelmaan voidaan rajoittaa käyttäjälle näkyviä osioita. Viittaus on seuraavanlainen:

```
<%@ taglib prefix="security" uri="http://www.springframework.org/security/tags" %>
```

Jos näkymästä halutaan määritellä joku osio vain tietyn käyttöoikeuden omaaville käyttäjille, voidaan se tehdä määrittelemällä security-tagi:

```
<security:authorize access="hasRole('ROLE_PERUSTIEDOT')" >
        tähän näytettävä osuus…
```

</security:authorize>

Spring-laajennoksen näkökulmasta käyttöoikeuksista käytetään nimitystä roolit (role). Tällä toimintamallilla pystytään liittämään toimeksiantajan käytössä oleva käyttäjätunnusten ja käyttöoikeuksien hallintajärjestelmä osaksi uutta järjestelmää.

3.3 Kutsurajapinta taustajärjestelmiin

Uudistuksen yhteydessä kokonaisarkkitehtuurin kannalta on tehty linjaus, jonka mukaan tietokannasta haettava data tarjotaan tietysti pisteestä eteenpäin aina samanlaisessa formaatissa kutsurajapinnasta riippumatta. Kutsut taustajärjestelmiin tehdään http-pyyntöinä ja ne osoitetaan Service Bus –nimiselle ohjelmistolle, joka puolestaan ohjaa ne eteenpäin tehden tarvittavat konversiot tiedon esitystavasta lähtien.

http-rajapinnan muodostamisessa käytetään Apachen open source HttpClient –toteutusta. Kutsuvan pään (client) alustaminen tehdään näin:

```
HttpClient client = new DefaultHttpClient();
```

jonka jälkeen kutsu voidaan suorittaa:

```
HttpResponse response = client.execute(request);
```

Kutsu palauttaa xml-pohjaisen vastauksen, joka parsitaan käyttöliittymän ymmärtämään muotoon javax.xml.parsers.DocumentBuilder –toteutuksen avulla. XML –rakenteesta parsitaan ne elementit, jotka sisältävät käyttöliittymän haluaman tietosisällön. Elementit ovat objekteja, jotka sisältävät itsessään hyvinkin monimuotoisia ja hankalasti käsiteltäviä rakenteita. Näiden soveltaminen käyttöliittymän tarpeisiin osoittautui projektin edetessä varsin haasteelliseksi tehtäväksi ja se edellytti monessa kohtaa erikoiskäsittelyä useamman tietorakenteen osalta.

Objektipohjaisten tietosisältöjen lisäksi omat kutsurajapinnat piti muodostaa niin kuvien kuin pdf-muotoisten raporttien binääridataa varten. Kuvien ja pdf-raporttien binääridata välitetään käyttäjien pyynnöille byte[] –taulukkomuodossa.

3.4 Ulkoasu

Käyttöliittymän toteutustavasta ja –tekniikasta riippumatta käyttäjän internet-selain saa aina purekseltavakseen HTML-koodia. Tässä projektissa ulkoasuasioihin ei tarvinnut kovin paljoa paneutua koska ulkoasullisesti järjestelmän haluttiin vastaavan tässä vaiheessa vanhaa järjestelmää. Sopivat HTML-pohjat oli otettavissa suoraan sieltä. Vaatimuksina oli tosin käyttää HTML5 –standardia sekä siirtää mahdollisimman paljon muotoilua css-tiedostoihin (Cascading Style Sheet).

Työnjako tämän osalta tehtiin siten että graafikko toimitti HTML-pohjat tyylitiedostoineen. Oma osuuteni ulkoasun osalta rajautui HTML-pohjien muuttamiseksi JSP-muotoisiksi. JSP-sivuille piti toki tämän jälkeen lisätä toiminnallisuudet siten, että palvelimella oleva sovellus ymmärsi käyttäjien pyynnöt.

Lähes kaikki nykyaikaiset web-sovellukset hyödyntävät JavaScriptiä erilaisten toimintojen suorittamiseksi ilman että pyyntöä täytyy välittää palvelinpuolen servletille. Toteutettu uudistus hyödyntää myös JavaScriptiä lähinnä palvelinpuolelle välitettävien syötteiden generoimisessa sekä erilaisten tarkistusten muodossa. Tähän tarkoitukseen on tarjolla paljon erilaista valmista toteutusta, mainittakoon JQuery ehkä yleisimpänä.

Koska eri selainmoottorit tukevat JavaScriptiä varsin kirjavasti, koitin pitää JavaScriptin käytön mahdollisimman yksinkertaisena ja käyttää sitä vain niissä kohdin joissa ei tuntunut järkevää vaihtoehtoa toteutustavalle olevan. Varsinaisten standardien puuttuessa eri selainversiot aiheuttavat web-ohjelmoinnissa todella paljon ylimääräistä työtä niin Javascriptin kuin HTML:nkin osalta. Lähes poikkeuksetta Internet Explorerin eri versiot toimivat aina muista selaimista poikkeavalla tavalla. Kaikkien käytössä olevien selainten tukeminen olisi käytännössä ollut mahdoton tehtävä. Nykyjärjestelmästä otettu otanta antoi kuitenkin hyvää osviittaa siitä mitä selaimia järjestelmän ainakin pitäisi tukea. Tämän analyysin lopputulemana projektin johtoryhmä päätyi ratkaisuun, jonka perusteella järjestelmän pitää tukea seuraavia selaimia:

- Internet Explorer, versiot 7-9 (82% käyttäjistä)
- Mozilla Firefox, versiot 5-20 (12% käyttäjistä)

- Google Chrome, uusimmat versiot (2% käyttäjistä)
- Safari, uusimmat versiot (1% käyttäjistä)
- Muut päätelaitteet (3% käyttäjistä)

3.5 Git versionhallinta

Toimeksiantajan kehitysympäristössä käytetään Git-versionhallintaohjelmistoa. Koko projektirakenne piti siirtää sinne jotta lähdekoodi on tallella ja kaikkien saatavilla. Erillistä graafista käyttöliittymää Gitin käyttöön ei ollut saatavilla, joten sen käyttö tapahtui joko suoraan Eclipse-pluginin kautta tai suoraan komentoriviltä (Git Bash). Koska käytössäni ollut Eclipse-versio ei toiminut saumattomasti yhteen Gitin kanssa käytin komentorivipohjaista työkalua Gitin hallinnointiin.

Kun projektin hakemistorakenne oli vakiintunut, lisäsin projektin ensimmäistä kertaa Gitiin:

- git add *

Projektin perustamisen jälkeen tiedostojen päivittäminen Gitiin tapahtui komennolla

- git commit –a –m ”uusi paivitys”
- git push

Ensimmäinen komento siirsi päivitykset lokaaliin varastoon (repository) ja jälkimmäinen vasta päivittää muutokset kaikkien käyttäjien yhteiseen repositoryyn. Yhteisestä varastosta voi hakea muiden tekemiä päivityksiä komennolla:

- git pull

3.6 Projektin rakenne ja toiminta

Projektin hakemistorakenne määräytyi osittain Maven –ohjelmiston asettamien vaatimusten pohjalta. Lähdekoodin juurihakemiston tulee olla src/main/java –polussa. Tämän polun perään tuli toimeksiantajaan viittaava polku, jonka perään sijoitin projektin alihakemistot. Päädyin seuraavaan rakenteeseen:

adapters -> tänne tuli kaikki ulkoisiin rajapintoihin liittyvät apuluokat

beans -> tänne tuli JavaBeanit, joiden avulla tietoa voidaan välittää esityskerroksen ja palvelinpuolen välillä

controllers -> tänne tuli käyttäjien pyyntöihin reagoiva luokka. Käsittelee kaikki GET- ja POST –pyynnöt. Näitä voisi olla useampia eri asiakokonaisuuksiin liittyen, mutta tässä yhteydessä päädyin ratkaisuun yhden luokan riittävyydestä

utils -> tänne tuli kaikenlaiset apuluokat (mm.url-syötteiden tarkistukset, käytetyt vakio-määrittelyt, päivämäärien parsinta jne)

JSP-sivujen osalta hakemistorakenne on erillään varsinaisesta lähdekoodista ja se sijoitettiin src/main/webapp –hakemistoon. Varsinaisten sivujen lisäksi tänne tuli xml-määritystiedostoja liittyen JBoss-sovelluspalvelimeen sekä itse sovellukseen (web.xml jne). JSP-sivut tulivat src/main/webapp/WEB-INF/jsp –hakemiston alle. Toteutettu sovellus on jsp-sivujen määrän suhteen laajahko, sillä sivuja tuli kaiken kaikkiaan 53

kappaletta. Pääosin sivut ovat erilaisten näkymien esittämistä varten, mutta joukossa on myös yleisiä osioita kuten menuvalikko, ylä- ja alareunan esityspalkit (header & footer), virhesivu jne.

Toteutetun projektin luonteesta johtuen tietyt vakiomenut ja muut osiot oli järkevää eriyttää omiksi kokonaisuuksikseen, jotta niitä ei joka sivulla tarvitsisi erikseen määritellä. Vain sisältöosa vaihtuu sivustolla sen mukaan millaisia valintoja käyttäjä on tehnyt ennen raportin hakua. Sisältöosa voidaan määritellä dynaamisesti vaihtuvaksi <include> -tagin avulla:

<%@ include file="/WEB-INF/jsp/toimihenkilot.jsp" %>

Hakuehtojen valitsemismahdollisuudet täyttävät koko vasemman reunan käyttäjälle näkyvästä näkymästä eli valintamahdollisuuksia on verrattain runsaasti. Menu –osioon piti lisäksi sijoittaa avautuvia ja sulkeutuvia valintaosioita. Kaikki valinnat eivät tietysti näy kaikille käyttäjille, vaan niiden näkyvyyttä rajoitetaan käyttöoikeuksilla jolloin yksittäisen käyttäjän näkymä pysyy järkevänä. Lisäksi erityyppiset raportit sijoitettiin niiden tyypin mukaan omiin alihakemistoihinsa yllämainittuun rakenteeseen.

Sovelluksen kehitystyö tapahtui Windows 7 työasemalla Eclipse –nimistä editoria käyttäen. JBoss EAP 6 sovelluspalvelin oli integroitu Eclipseen siten, että sen käynnistäminen ja sammuttaminen onnistui editorista käsin. Lisäksi sovellus oli helppo ”julkaista” ko. sovelluspalvelimelle napin painalluksella joten tehdyt muutokset pystyi aika vaivattomasti näkemään heti tekemisen jälkeen. Tosin sovelluksen koon paisuessa julkaiseminen selvästä hidastui mikä osaltaan viivästytti päivittäistä tekemistä. Käytännössä vaativimpien kohtien ratkaiseminen edellytti sovelluksen ajamista palvelimella debug-moodissa, jolloin muuttujien tilaa ja sovelluksen käyttäytymistä pystyi seuraamaan rivi kerrallaan. Lisäksi pyrin lisäämään tarpeeksi kommentteja lähdekoodin yhteyteen sekä tulostamaan kriittiset tiedot (mahdolliset virhetilanteet, käyttäjän suorittamat haut parametreineen jne) sovelluslokille.

Aina kun olin saanut yksittäisen toiminnon tai raportin valmiiksi, siirsin sen testiympäristöön varsinaisten testaajien testattavaksi. Valmiin paketin kasaaminen tapahtui Maven –nimisen työkalun avulla komentorivipohjaisesti:

- mvn clean install

Komennon seurauksena Maven paketoi kaikki tarvittavat luokat kirjastoineen yhteen war-tiedostoon edelleen testiympäristön palvelimelle siirtoa varten. JBoss-sovelluspalvelimen osalta sovelluksen julkaisemiseksi riitti tämän paketin siirtäminen tiettyyn hakemistoon, josta sen sisäinen tiedostonhallintajärjestelmä automaattisesti osasi huomioida sinne siirretyn paketin.

3.7 Tietoturva

Projektin esikartoitusvaiheessa suunniteltua ratkaisumallia vasten oli tehty tietoturvatestaus ulkopuolisen asiantuntijaorganisaation toimesta. Testauksen tuloksena ratkaisumallista löytyi joitakin kohtia jotka piti ottaa huomioon sovellusta tehtäessä. Yleisesti ottaen kaikki syötteet tulee validoida riittävän tarkasti jottei vahingollisia syötteitä pääse ohjelmakoodin suoritusosioihin asti. Tietoturvatestauksessa ilmeni seuraavanlaisia haavoittuvuuksia:

- XSS (Cross-site scripting)
- Turvaton HttpOnly eväste asetuksen puuttuminen
- Pääsy JBoss EAP 6.0 hallintakonsoleihin
- Stack trace –tietojen näyttäminen

Mainittujen haavoittuvuuksien korjaamiseksi löytyi sopivat apukirjastot esimerkkeineen 'The Open Web Application Security Project' –sivustolta (owasp). Niiden avulla erityyppiset syötteet oli helppo tarkistaa (validoida).

```
import org.owasp.esapi.ESAPI;
public static boolean isValidHTTPParameterValue(String param) {
  try {
    param = ESAPI.validator().getValidInput("HTTPParameterValue check",
    param, "HTTPParameterValue", 50, true);
  }
  catch (ValidationException e) {
        log.error("Virhe: isValidHTTPParameterValue(String param),
        param="+param+" Syy: "+e.getMessage());
    return false;
  }
  catch (IntrusionException ex) {
     log.error("Virhe: isValidHTTPParameterValue(String param),
     param="+param+" Syy: "+e.getMessage());
    return false;
  }
  log.info("isValidHTTPParameterValue(String param), param="+param+",
  PASSED!");
  return true;
}
```

Tarkistusten toimivuuden osalta tarvittiin vielä konfiguraatiotiedostot, joissa määriteltiin minkälaisia syötteitä missäkin kohdin voidaan hyväksyä. Esimerkiksi url-parametrin syötteessä hyväksytään vain tällaisen määrityksen mukaiset merkit:
Validator.HTTPParameterValue=^[a-zA-Z0-9.\\-\\/+=@_åöäÅÖÄ]*$

Huomionarvoista oli, että perusasetuksilla skandinaaviset aakkoset eivät menneet läpi vaan kirjaimet å, ä ja ö (isot ja pienet kirjaimet) piti erikseen lisätä sallittujen merkkien listalle.

4 Yhteenveto ja johtopäätökset

Varsinkin laajat sekä monitahoiset IT-projektit vaativat hyvin organisoitua projektinhallintaa sekä projektiryhmän tiivistä yhteistyötä ja sitoutuneisuutta. Annetussa aikataulussa ja kustannuksissa saattaa olla vaikeuksia pysyä. Yllättäviä ongelmatilanteita ja viivästyksiä saattaa esiintyä sitä enemmän, mitä enemmän toteutettavassa hankkeessa on eri tahoja mukana. Vaikka tämän opinnäytetyön puitteissa tehtävä työ oli tiukasti rajattu, liittyi siihen monia riippuvuuksia laajemman kokonaisprojektin myötä. Tämän takia työ ei aina edennyt toivotun aikataulun mukaisesti. Lisäksi muiden päivittäisten työtehtävien hoitaminen häiritsi jonkin verran omistautumista projektityölle.

Projektiryhmän keskinäistä kommunikointia helpotti yhteinen työskentelytila, joka oli varattu projektiryhmän käyttöön. Toteutusvaiheessa kehittäjät saivat myös erilliset työasemat, joihin asennettiin kehitysympäristöt vain tätä projektia varten.

Projekti poikkesi tavanomaisesta toteutuksesta siltä osin että uudistetun käyttöliittymän piti toimia kuten edeltäjänsä. Uudella järjestelmällä on oltava tarpeeksi selkeät määritykset ja säännöt miten sen tulee toimia. Tämän projektin yhteydessä joutui monesti turvautumaan käytössä olevaan käyttöliittymään tarkistamalla miten siellä tietyt asiat toimivat. Kaikkia toimintoja ei kuitenkaan ole mahdollista toteuttaa tällä periaatteella, vaan määrittelydokumentaatiosta pitäisi olla yksiselitteisesti todennettavissa kaikki käyttötapaukset.

Projektinhallinnan osalta tähän projektiin kohdistui paljon muutoksia. Liikkeelle lähdettiin pelkän toteutustiimin kanssa ilman varsinaista projektipäällikköä tai vastaavaa. Tilanne ajautui kuitenkin nopeasti siihen, että kenelläkään ei ollut kokonaisvastuuta, vaan kukin edisti sitä osa-aluetta jonka parhaiten osasi. Myöhemmässä vaiheessa projektiin saatiin hallinnollinen projektipäällikkö, jonka myötä työnjako ja vastuut hieman selkenivät. Siitä huolimatta projektinhallinnan osalta jäi varsin sekava kuva kokonaisuudesta, joka mielestäni haittasi projektin etenemistä jossain määrin.

Käytössä olevien työvälineiden ja ohjelmistojen osalta tilanne oli parempi. Uudet työasemat ja niihin vain tätä projektia varten asennettu kehitysympäristö oli hyvä ratkaisu.

Koneet olivat myös tarpeeksi tehokkaita ja prosessien pyörittämiseen riitti tarpeeksi konetehoja.

Projektiryhmälle järjestetty yhteinen työtila helpotti keskinäistä kommunikointia ja edesauttoi päivittäisten ongelmien ratkaisua. Sen sijaan projektiryhmäläisten epäsäännöllinen paikallaolo heikensi hieman tehokkuutta, joskin tiukkoja sidoksia eri kokonaisuuksien välillä ei tässä projektissa ollut. Projektiryhmäläisten ammattitaito oli riittävällä tasolla onnistuneen lopputuloksen saavuttamiseksi. Tarpeen tullen ongelmatilanteisiin haettiin ratkaisuja spontaaneilla keskusteluilla ja palavereilla projektiryhmäläisten kesken.

Opinnäytetyön tekemiseen suunniteltu työmäärä on 400 tuntia. Projektisuunnitelmaa ja sen rajauksia etukäteen määrittäessä oli hankala arvioida mihin kaikkeen mainittu tuntimäärä riittää. Projektin toteutusvaiheen aikana kävi ilmi että käyttöliittymässä on lukuisia sellaisia toimintoja, jotka eivät olleet aloitusvaiheessa tiedossa. Loppujen lopuksi opinnäytetyön puitteissa toteutettavat toiminnot piti aikataulullisesti rajata vuoden 2012 loppuun ja siirtää osa toiminnoista tehtäväksi myöhemmin. Tämän takia koko projektin kustannusten toteumaa on hankala analysoida, mutta yleisellä tasolla voitaneen todeta että koko projektin budjetti on ylittynyt roimasti.

Oppimisen ja tekemisen kannalta opinnäytetyön suorittaminen oikean työelämän puitteissa motivoi tehokkaasti ja säästi aikaa. Lisämotivaationa toimi tieto siitä että järjestelmä tulee aikanaan oikeaan tuotantokäyttöön todellisen toimeksiannon tuloksena.

Laskennallisesti tämän projektin toteutus kesti siis noin 10 viikkoa (10 * 37,5h) ja loput tunnit menivät projektin hallinnointiin sekä dokumentointiin kokonaistyömäärän ollessa reilusti yli 400 tuntia. Koska projektia ei ollut käytännön syistä mahdollista tehdä 10 viikkoa peräkkäin niin kalenteriaikana mitattuna projekti jakautui noin kuudelle eri kuukaudelle (4/2012 – 12/2012 väliselle ajalle).

Vaikka tärkeimmät pullonkaulat projektin etenemisen osalta olivat jo ratkottu ennen projektin aloittamista, pienempiä haasteita riitti jatkuvasti. Ongelmanratkaisutilanteita tuli esille sopivasti ja ennalta valitut ratkaisumallit taipuivat riittävästi vaatimusten mu-

kaan. Eniten luovuutta vaati taustajärjestelmistä tulevan tiedon käsittely sellaiseksi, että sen pystyi järjellisellä tavalla esittämään käyttöliittymässä (JSP & JSTL). Vain muutamissa kohdissa tietorakenteet vaativat jonkinlaista esikäsittelyä palvelinpuolella, muutoin JSTL taipui yllättävänkin monimutkaisiin toimintoihin.

Muilta osin työ oli välillä jopa hieman yksitoikkoista. Toki pyrkimyksenä oli pitää toiminta mahdollisimman yksinkertaisena ja jatkossa helposti ylläpidettävänä. Oppimisen osalta projektia voisi luonnehtia syvälliseksi Spring web-frameworkiin tutustumisena JSTL-kirjastojen avulla. Java-ohjelmointikielen osalta uutta asiaa ei oikeastaan tullut esille, mutta JavaScript -osaaminen sen sijaan karttui jonkin verran. Toimeksiantajan osalta työ herätti suurta kiinnostusta, joka tietenkin toimi hyvänä motivaationa projektin edetessä. Kokonaisuutena voidaan todeta, että tämä projekti opinnäytetyönä oli sopivan haastava ja käytännönläheinen vahvan työelämäsidonnaisuutensa takia. Aiemmista ammattikorkeakoulun opintojaksoista ei ollut suoranaista hyötyä projektin toteuttamisessa, joskin yhtäläisyyksiä löytyi jonkin verran. Jos aiempaa työkokemusta tämän tyyppisestä työstä ei olisi ollut, niin opintojaksoilla opitut asiat olisivat varmasti olleet suuremmassa roolissa.

4.1. Johtopäätökset

Esikartoitusvaiheessa oli vaihtoehtona valmiin portaalituotteen käyttöönotto ja sovelluksen rakentaminen sen päälle. Yhtä aikaa tämän projektin kanssa toteutettiin toinen projekti jossa tällainen ratkaisu tehtiin. Yleisesti ottaen valmiiden ohjelmistojen käyttöönotolla pyritään säästämään aikaa ja kustannuksia. Haittapuolena on monesti se, että niitä tarvitsee kustomoida voimakkaasti käyttötarpeiden mukaisesti. Tämä ei aina onnistu, koska se on joko estetty tai toteutettavissa vain alkuperäisen ohjelmistotoimittajan toimesta.

Koska tässä tehty käyttöliittymäuudistus on täysin toimeksiantajalle räätälöity ratkaisu, on perusteltua että valmisohjelmistoa ei käytetty. Toisen osaprojektin kautta saadut kokemukset ja ongelmat sen käytössä vahvistavat tätä näkemystä.

Uudistettavan sovelluksen osalta toiminnot näyttivät aluksi helposti toteutettavilta ja yksinkertaisilta. Toteutusvaiheessa paljastui sovelluksen kompleksisuus sekä lukuisat

sellaiset toiminnot jotka eivät olleet aloitusvaiheessa tiedossa. Tämä toi osaltaan lisähaasteita projektille. Jälkeenpäin voidaan todeta, että tarkempi perehtyminen olemassa olevaan sovellukseen ja sen toimintaan olisi ollut tarpeellista.

Kriittisten ohjelmistokomponenttien uusiminen on aina iso muutos ja sisältää paljon riskejä. Tämän projektin osalta ensiarvoisen tärkeää oli se, että järjestelmän eri kokonaisuudet pystyttiin todentamaan toimiviksi erillisen POC (Proof of concept) –vaiheen tuloksena. Samalla toteutukseen liittyvät asiat tulivat tekemisen kautta tutuiksi, joka nopeutti ja helpotti varsinaista tekemistä. Kunhan lukuisat lisätoiminnot ja erikoistapaukset saadaan toteutettua varsinaisen ohjelmarungon lisäksi, niin projektin tavoitteet voidaan todeta saavutetuiksi. Lisäksi virtualisoitu ajoalusta takaa skaalautuvuutta käyttötarpeen mukaan, joten sovelluksella on kaikki edellytykset saavuttaa asemansa luotettavana ja toimintakykyisenä osana toimeksiantajan järjestelmäkokonaisuutta.

Suurimpana kehityskohteena nousee esiin sekava projektinhallinta. Projektiin olisi ehdottomasti tarvittu heti alusta alkaen projektipäällikön tehtäviä päätoimisesti hoitava henkilö, jolla olisi ollut myös vahva osaaminen projektin teknisistä asioista. Tehtävät tulisi selkeästi vastuuttaa, aikatauluttaa ja seurata että ne tulevat tehtyä annetun aikataulun puitteissa. Tältä osin toiminta oli mielestäni liian leväperäistä ja asiat jäivät liiaksi roikkumaan.

Lähdeluettelo

Apache Software foundation. Apache Wicket.
Luettavissa:
http://wicket.apache.org/
Luettu: 13.4.2012

Apache Software foundation. Apache Tapestry
Luettavissa:
http://tapestry.apache.org/
Luettu 14.4.2012

Bloch, Joshua. 2008. Effective Java, Second Edition. Addison-Wesley. USA

Developpez.com 2013. Club des devepoppeurs et IT pro.
Luettavissa:
http://bpesquet.developpez.com/tutoriels/php/evoluer-architecture-mvc/
Luettu 14.4.2012

JBoss Community. JBoss Application Server
Luettavissa:
http://www.jboss.org/overview/
Luettu: 14.4.2012

Liferay Inc. 2012. Liferay portaali
Luettavissa:
http://www.liferay.com/
Luettu 3.4.2012

Mak, Gary. 2008. Spring Recipes: A Problem-Solution Approach. Apress. USA

Nyberg, Gregory & Patrick, Robert. 2003. Mastering BEA Weblogic Server. Wiley

Oracle docs 2013. The Java EE 6 Tutorial
Luettavissa:
http://docs.oracle.com/javaee/6/tutorial/doc/
Luettu 22.3.2013

Oracle 2009. JRockit JVM Diagnostics guide R27.6.
Luettavissa:
http://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/pdf/diagnos.pdf
Luettu 14.4.2012

Oracle Technology Network 2012. Java.
Luettavissa:
http://www.oracle.com/technetwork/java/javase/overview/index.html
Luettu 13.4.2012

Powers, Shelley. 2010. JavaScript Cookbook, First Edition. O'Reilly Media. USA

Raible, Matt 2012 Comparing JVM Web frameworks 2010
Luettavissa
http://raibledesigns.com/rd/entry/my_comparing_jvm_web_frameworks
Luettu 13.4.2012

Red Hat Inc 2013. JBoss Enterprise Middleware
Luettavissa:
http://fi.redhat.com/products/jbossenterprisemiddleware/
Luettu 12.1.2013

Spring Community 2013 Spring Framework, Spring projects
Luettavissa:
http://www.springsource.org
http://www.springsource.org/projects
Luettu 12.1.2013

Visualising Garbage Collection in the JVM
Luettavissa: http://redstack.wordpress.com/2011/01/06/visualising-garbage-collection-in-the-jvm/
Luettu 29.4.2012

Wikipedia 2013 Java
Luettavissa:
http://fi.wikipedia.org/wiki/Java
Luettu 1.4.2013

Wikipedia 2013 Plain Old Java Object
Luettavissa:
http://en.wikipedia.org/wiki/Plain_Old_Java_Object
Luettu 19.6.2012

Wikipedia 2013 Spring Framework
Luettavissa:
http://en.wikipedia.org/wiki/Spring_Framework
Luettu 19.6.2012

LIITTEET

Loppuraportti

HAAGA-HELIA
ammattikorkeakoulu

Web-pohjaisen käyttöliittymän uudistaminen portaalimalliin

Opinnäytetyön loppuraportti

Jani Hurme

Tietotekniikan koulutusohjelma

2013

HAAGA-HELIA
ammattikorkeakoulu

Loppuraportti

10.5.2013

Tietotekniikan koulutusohjelma

Tausta

Opinnäytetyöprojekti suoritetaan opiskeluiden loppuvaiheessa ja se on laajuudeltaan yhteensä 15 opintopistettä. AMK-tutkinnon kokonaislaajuus yhteensä 210 opintopistettä. Tämän projektin aloitusvaiheessa olin suorittanut 190 opintopistettä ja syksyn 2012 aikana oli tarkoitus suorittaa puuttuvat 5 opintopistettä opinnäytetyön lisäksi.

Olin suunnitellut tekeväni opinnäytetyön työnantajan palveluksessa jonkin todellisen projektin puitteissa. Useimmiten opinnäytetyöt suoritetaankin jonkun yrityksen toimeksiannosta todellisen tarpeen mukaisesti. Sopiva projekti löytyi isommasta projektikokonaisuudesta, josta voitiin rajata sopiva osuus opinnäytetyöprojektiksi (n. 400 tuntia). Ehdotettu kokonaisuus hyväksyttiin, ja projekti voitiin aloittaa keväällä 2012.

Suoritetut opinnot tukivat opinnäytetyöprojektin aihealuetta, joskin pitkähkö työura saman työnantajan palveluksessa antoi paremmat valmiudet työn suorittamista varten.

Saavutetut tulokset

Projektin lopputulos vastaa pääosin projektisuunnitelmassa asetettuja tavoitteita. Toimeksiantajan käyttöön on toteutettu käyttöliittymäsovellus määrittelyiden mukaisesti. Projektin aikana on pidetty aloituskokous, seurantakokous sekä päättökokous. Edistymisestä on raportoitu kokousten yhteydessä.

Oppimisen osalta tavoitteet eivät sen sijaan toteutuneet siinä mittakaavassa mitä tavoitteisiin oli kirjattu. Koska projekti tehtiin toimeksiantajan käytäntöjen ja välineiden avulla, ei opintojen yhteydessä hankittua osaamista päästy hyödyntämään siinä laajuudessa mitä alun perin oli suunniteltu. Tärkein toteutunut tavoite oppimisen kannalta oli omaaloitteisuus ja yksittäisten asioiden eteenpäinvienti organisaation sisällä sekä ulkopuolis-

ten tahojen kautta. Projektiryhmälle ei ollut määritelty tarkkoja vastuualueita joten vastuunottoa ja oma-aloitteisuutta tarvittiin enemmän kuin tyypillisessä ohjelmistoprojektissa.

Laadullisesti projektin lopputulos olisi voinut olla parempi. Toimeksiantajan käytössä on oma laatujärjestelmä, mutta kaikkia sen edellyttämiä dokumentteja ei ole ollut käytettävissä tai niitä ei tehty koskien määrittely-, suunnittelu- ja toteutusvaihetta. Lisäksi puutteita esiintyi projektinhallinnassa, joka yleisesti ottaen ei ollut tarpeeksi järjestäytynyttä.

Työn eteneminen ja kustannusten toteutuminen

Työtilat, -välineet ja toimintatavat olivat valmiina toimeksiantajan käytäntöjen mukaisesti eli niihin ei juurikaan ollut päätäntävaltaa. Mainitut asiat olivat tuttuja, koska olen toiminut toimeksiantajan palveluksessa reilut seitsemän vuotta. Välineet ja menetelmät olivat nykyaikaisen ohjelmistokehityksen mukaiset sekä projektiryhmän jäsenet kaikki kokeneita ohjelmistokehittäjiä.

Uutta opeteltavaa tuli toimintaympäristön myötä käyttöjärjestelmien sekä sovelluspalvelinympäristöjen myötä. Projektia varten luotiin uusi täysin virtualisoitu ympäristö joka edellytti uuden omaksumista. Työ eteni ajoittain varsin tiiviisti yhteisen projektityötilan ansiosta. Etenemistä hidastutti projektiryhmäläisten epäsäännöllinen paikallaolo sekä päivittäisten työtehtävien hoitaminen projektityön ohessa.

Kustannusten osalta alkuperäinen arvio ylitettiin reippaasti. Opinnäytetyöprojektin osalta tuntimäärä ylittyi noin viidennesosalla.

Resurssien käyttö

Opinnäytetyöprojektin näkökulmasta tärkein resurssi on opiskelija itse. Ajankäytön osalta toteutustyö onnistui aika hyvin koska se oli mahdollista tehdä työajalla. Sen sijaan

kirjalliset tuotokset vaativat aikamoisia ponnisteluita ja ohjeistus tuntui monessa kohdin puutteelliselta. Jotta johdonmukainen eteneminen olisi ollut mahdollista, tulisi ohjeistuksen olla yksiselitteistä eikä jättää toteuttajalle mitään arvailujen varaan. Opinnäytetyön ohjaaja ja muut sidosryhmät reagoivat ongelmatilanteissa ripeästi.

Kokemukset ja suositukset

Positiivisina kokemuksina opinnäytetyöprojektin osalta mainittakoon haastava ja mielenkiintoinen ja haastava toteutustyö. Työn aikana esille tulleet ongelmatilanteet selvisivät monesti oma-aloitteisesti ratkaisua hakemalla tai turvautumalla muiden asiantuntijoiden apuun. Myös usean tahon yhteistyö sujui yleisesti ottaen joustavasti johtuen lähinnä toimeksiantajan sisäisestä tilaaja-toimittaja asetelmasta.

Toimeksiantajan käytäntöihin liittyen suosittelen paremmin organisoitua projektinhallintaa. Vaikka ohjelmistokehitys on pitkälti tiimityötä, tulisi yksittäiset asiat vastuuttaa selkeämmin ja seurata että ne tulevat toteutetuksi ajallaan. Toki projektin luonne vaikutti tähän asiaan, koska kyseessä oli pitkälti toimeksiantajan IT-osaston sisäinen osaprojekti. Lisäksi projektinhallinnan välineiden osalta jäi toivomisen varaa, ne eivät täyttäneet toiveita odotusten mukaisesti.

Ohjelmistoarkkitehtuurin osalta pidän tehtyjä ratkaisuja hyvin harkittuina ja perusteltuina. Niiden pohjatyö oli riittävän laajaa ja asiantuntemusta haettiin monelta taholta. Ennen lopullisia teknologiavalintoja tehtiin pienempi toteutustyö sekä useampia raporttipohjaisia selvityksiä riittävän tiedon saamiseksi.

Kokonaisuutena opinnäytetyöprojektiksi valittu kokonaisuus sopi mielestäni kohtalaisen hyvin oppilaitoksen määrittelyn mukaiseksi systeemityöhankkeeksi. Toimeksiantajan organisaatio vastaa hyvin tyypillistä ympäristöä jossa ohjelmistoprojekteja tuotetaan.