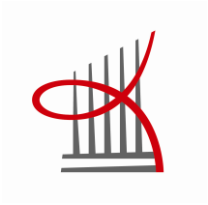


PUOLIAUTOMAATTINEN OHJELMISTON PAIKKAUSTYÖKALU

Lassi Piironen

Opinnäytetyö
Joulukuu 2014
Tietotekniikka
Ohjelmistotekniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Ohjelmistotekniikka

LASSI PIIRONEN:
Puoliautomaattinen ohjelmiston paikkaustyökalu
Opinnäytetyö 46 sivua, joista liitteitä 6 sivua
Joulukuu 2014

Ohjelmiston päivitysprosessissa pitää tehdä tiedostovertailut ohjelman alku- ja lopputilanteiden välillä silloin, kun päivityksen kohteena olevassa ohjelman versiossa on paljon päivityspaketissa huomioimattomia mutta kuitenkin tärkeitä paikallisia muutoksia. Vertailujen eroavaisuudet kopioidaan päivityspakettiin siten, että uudessa lopputulemassa on sekä päivityksen uudet ominaisuudet että tuotantoympäristöön liittyvät paikalliset muutokset. Prosessin toteutus on suhteellisen yksinkertainen, kun päivityskohteita on vain muutamia. Jos päivityksen korjaus on kiireellinen, ja se tulisi asentaa laaja-alaisesti suurelle osaa käytössä olevista tuotantoversioista, on prosessi kuitenkin kokonaisuudessaan raskas, monimutkainen ja siten myös altis virheille.

Tuotannossa voi olla monia eri ohjelmiston versioita, joten päivityspaketti pitää muotoilla kyseisille versioille sopivaan muotoon. Laaja-alaisen päivityksen levittämiseksi ohjelmistosuunnittelijan tulee toteuttaa kyseiset korjaukset yksittäistapauksina kaikkiin käytössä oleviin versioihin. Tuotantoversioiden korjauspäivittäminen vaatii siis jokaiselle päivitettävälle asennukselle oman uniikin versio- ja ympäristökohtaisen päivityspaketin.

Työssä toteutettu puoliautomaattinen paikkaustyökalu on PHP-ohjelmointikielellä toteutettu aputyökalu. Sen tarkoituksena on versiokohtaisten korjauspäivitysten ohjelmallinen luominen sekä niiden jakaminen moneen eri tuotantoversioon siten, että myös asennusten paikalliset muutokset huomioidaan, ja että muutokset on myös mahdollista arvioida ennen käytäntöönottoa. Työkalun on tarkoitus automatisoida tiedostovertailuja vaativasta ohjelmistopäivitysprosessista ne osat, jotka eivät tavallisesti vaadi päivityksen asentajahenkilöltä erityistä harkintaa. Työn ajatuksena on se, että ohjelmistopäivityksessä asentajahenkilö pystyy paremmin keskittymään olennaiseen eli päivityksen riskialttiiden muutosten arvioimiseen.

Työ onnistui tavoitteissaan, mutta sen tuloksena syntynyt työkalu ei ole sellaisenaan ainakaan yksittäisenä työkaluna valmis kokonaisratkaisu suurimpiin päivitysarkkitehtuurisiin ongelmiin. Kehitysympäristöön liittämiseen liittyvällä jatkokehityksellä työkalu kuitenkin toimii työntekoa helpottavana osana kehitysympäristön isoa kokonaisuutta, jossa yhdistyvät versionhallintajärjestelmän ominaisuudet, versiokohtaisten päivitysten luomisen helppous sekä turvallinen tiedonsiirto eri järjestelmien välillä.

Asiasanat: PHP, ohjelmistopäivitys, automaatio, versionhallinta, räätälöity ohjelmisto

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information technology
Software engineering

LASSI PIIRONEN:
Semi-automatic patching tool

Bachelor's thesis 46 pages, appendices 6 pages
December 2014

In a software upgrading process where the destination systems have important locally modified contents that are not included in the new install package, it is required for the process to apply file based comparisons between the starting and the current snapshot point of the software. The differences from the comparison are copied to the update package so that the resulting install package have both the features from the initial update and the local modifications. The implementation of the process is relatively simple when there are only a few production environment instances to upgrade. However, if the patch is urgent and should be installed to a wide range of different production versions the process as a whole becomes heavy, complex and thus prone to errors.

There can be several different software versions in use at the production environments so the initial update package must be modified to suit the specifications of the version. For delivering an wide-range update the software engineer will have to make seperate repairs for each in use version. Patching multiple production versions requires therefore as many unique and environment specific patches as there are production versions.

The semi-automatic patching tool produced for this thesis is a utility tool developed in PHP programming language whose purpouse would be the programmatic creation of version specific patches and the delivery of the created patches in a way that the production systems local changes are taken into account. The tool was designed to automate the parts that do not require user attencion from the software updating processes where the file comparisons are required. The idea was that in a software update the person doing the job could better focus on the essentials of the process.

The work succeeded in its goals but the resulting tool is not a complete answer for the update architectural problems. Developing the tool forward when it is integrated to the some specific development environment the tool could work as a assisting part of the totality that combines the features of the version control systems, the easy creation of version specific patches and secure transfer of information between the systems.

Key words: PHP, software patch, automation, version control, tailored software

SISÄLLYS

LYHENTEET JA TERMIT

AJAX	Asynchronous JavaScript And XML, erilaisia mm. JavaScriptin ja palvelimen välisen taustakommunikaation helpoksi tekeviä työkaluja.
Beyond Compare	Tiedostojen ja kansiorakenteiden vertailuohjelma
CSS	Cascading Style Sheets, muotoillun tekstin tyyliohjemuoto
GET-parametri	Sivulatauksessa ohjelmistolle sivun URL-osoitteen mukana syötetty muuttuja
GIT	Versionhallintaohjelmisto
HTML	Hypertext Markup Language, tekstin merkintäkieli
JavaScript	Tulkattava komentosarjakieli
jQuery	JavaScript-kirjasto
logi-tiedot	Suoritettavan operaation tallentama tapahtumahistoria
Meld	Tiedostojen ja kansiorakenteiden vertailuohjelma
merge-toiminto	Integroidaan kaksi eri versiohallinnassa olevaa versiota uudeksi versioksi
modulaarinen systeemi	Järjestelmätoteutus jossa toiminnallisuudet on järjestelty toisistaan riippumattomiksi yksiköiksi
OpenCart	Avoimeen lähdekoodiin perustuva verkkokauppaohjelmisto
PHP	PHP: Hypertext Preprocessor, ohjelmointikieli
Plugin-systeemi	Modulaarinen tapa tuoda ohjelmistoon lisää ominaisuuksia
Räätälöinti	Ohjelmiston asennusversioon kuulumaton lähdekoodin muokkaus
SCP	Secure copy, tiedonsiirtoprotokolla joka perustuu SSH-yhteyksiin
SSH	Secure Shell, salattuun tietoliikenteeseen tarkoitettu protokolla
TAMK	Tampereen ammattikorkeakoulu
vQmod	Virtual Quick Mod, virtuaalinen tiedostojen muokkausjärjestelmä
XML	Extensible Markup Language, tekstin merkintäkieli

1 JOHDANTO

Tämä dokumentti on kuvaus PHP-ohjelmointikielellä totetutetusta ohjelmiston päivitystyökalusta. Sen avulla on mahdollista luoda yhden ohjelmistoversion päivityspaketista vastaavuudet ohjelmiston aikaisempiin versioihin, ja luoda näistä vastaavuuksista uniikit päivityspaketit, joissa on huomioitu kyseisten versioiden tuotantoympäristöjen paikalliset tiedostomuutokset.

Dokumentissa käsitellään työkalun toteutuksen olennaisimmat logiikat ja pohdiskellaan sen mahdollisia käyttötarkoituksia.

Työ on toteutettu syksyllä 2014 Tampereen ammattikorkakoulun opinnäytetyönä ja sen työelämän edustajana sekä haasteiden tarjoajana on toiminut sähköisen liiketoiminnan palveluja tuottava ProsperCart Oy.

1.1 Työkalun tarpeellisuus

Ohjelmistopäivitysprosessien tulee useimmiten huomioida ohjelmiston paikalliset muutokset. Muutokset voivat olla esimerkiksi ohjelman käyttäjäasetuksia tai suurempia kokonaisuuksia itse ohjelmiston lähdekoodissa. Jos päivityksessä ei huomioida paikallisia muutoksia, saatetaan siinä hävittää esimerkiksi ohjelmistoon räätälöintinä tehdyt ominaisuudet ja/tai pahimmillaan rikkoa koko ohjelmiston toiminta.

Muutosten päivitettävyys on osittain arkkitehtuurikysymys. Ohjelmistoympäristö, jossa erilaisten muutosten lisäämistapa on toteutettu modulaarisesti, on usein myös helppo päivittää. Jos systeemi ei ole paikallisten muutosten osalta modulaarinen tai sen sääntöjen noudattamatta jättäminen on eri muokkauksille tyypillistä, on muutosten huomioiminen tehtävä tekemällä vertailuja alku- ja muutostilanteen välillä.

Tämänkaltaisessa tilanteessa koko ohjelmiston räätälöinnit huomioiva päivitys on aikaa vievä projekti. Jos päivitykset tehdään maksullisina tuntitöinä, ei niiden tilaaminen ole asiakkaan näkökulmasta ainakaan aivan pienten vikojen takia kovin houkuttelevaa. Jos pikkuvika sattuu liittymään esim. tietoturvaan, olisi korjaus kuitenkin saatava mahdollisimman nopeasti asennettua.

Pienten vikojen korjaamisessa ei tarvitse päivittää koko ohjelmistoa, vaan korjaus on mahdollista toteuttaa siten, että vain virheelliset tiedostot päivitetään. Tällä tavalla voidaan säästää yksittäiseen korjaukseen kuluvaa aikaa, sillä käsiteltäviä muutoksia on määrällisesti vähemmän. Myös pienikokoisen korjauspäivityksen tulee huomioida paikalliset muutokset. Epämodulaariselle systeemille korjaus vaatii aina aikaa vieviä tiedostovertailuja.

Korjauspäivitysten jakaminen suurelle määrää asiakkaita on hankalaa, sillä suoraa kopiointia ei versioinnin ja erilaisten räätälöintien takia voi tehdä. Jos päivitettäviä tuotantoympäristöjä on esim. 100 kappaletta, vaatii päivitys ainakin periaatteessa saman verran kullekin instanssille sopivaksi räätälöityjä korjauspäivityksiä. Jokainen päivitettävä instanssi vaatii näin uniikin ja tarkoitukseen erikseen räätälöidyn korjauspäivityksen. Erilaisten päivitysvariaatioiden kokonaismäärä vaihtelee sen mukaan, kuinka paljon eri instansseilla on samoja ohjelmistoversioita ja ovatko yksittäisten instanssien päivitykseen liittyvät tiedostot räätälöityjä.

Tietoturvaan liittyvä virheen korjaus on hyvä esimerkki työn tarpeesta, sillä siinä yhdistyy sekä ohjelmiston toimittajan että sen asiakkaan tarpeet. Kummallakin osapuolella on kiire saada virhe korjattua, sillä tietoturva on osaltaan luottamuskysymys ja sen menettäminen ei ole kenenkään etu. Paikallisten muutosten osalta epämodulaarisessa järjestelmässä korjauspäivitysten nopea jakaminen on kuitenkin toteutuksen osalta melko pulmallista.

1.2 Työn taustat

ProsperCart Oy:n tuottamasta verkkokauppaohjelmisto ProsperCart:sta on tuotantokäytössä useita eri ohjelmiston elinkaaren versioita. Osa tuotannon versioista on räätälöity käyttäjän tarpeisiin sopiviksi. Tuotantoversiot sijaitsevat monen eri www-palveluntarjoajan palvelimilla. Tuotantoversion päivitysprosessi on hyvin dokumentoitu, mutta siinä on vain vähän automatisoituja kohtia. Yrityksen nykyisellä päivitysprosessilla laaja-alaisen korjauspäivityksen toteuttaminen on raskasta.

Idea opinnäytetyöhön on saanut alkunsa mm. OpenCart-ohjelmistossa käytetyn vQmod-kirjaston tiedostojen generointitoiminnallisuuksista. Sen avulla tiedostoista voidaan

generoida erilaisten XML-muodossa olevien haku- ja korvaus-sääntöjen perusteella uusia versioita. Jos kirjasto on integroitu kyseiseen järjestelmään, se ajaa käyttäjille alkuperäisten tiedostojen sijasta generoidut uudet tiedostoversiot. OpenCart-ohjelmistossa kirjastoa käytetään plugin-systeeminä siten, että ohjelmiston toiminnallisuuksia pystyy muokkaamaan muuttamatta itse ohjelmiston lähdekoodia.

Jos tiedoston generoijalle syötetään sopivat säännöt, on suurin osa päivitysprosessistakin automatisoitavissa. Säännöt on mahdollista löytää tiedostovertailuissa havaituista eroavaisuuksista ja ne on myös mahdollista muuntaa tiedoston generoijan ymmärtämään muotoon. Kun tietää etukäteen mitä on päivittämässä, mihin versioon ja missä päivitettävä versio sijaitsee, on kaikki tiedostovertailut ja sisällön siirtelyt ainakin periaatteessa mahdollista tehdä automaattisesti.

Räätälöidyissä ympäristöissä todellisen päivitysautomaation järjestäminen on kuitenkin hyvin hankalaa. Vaikka räätälöinti on dokumentoitu helposti tunnistettavilla merkinnöillä ja on siten ohjelmallisesti tunnistettavissa, ei sen toimivuutta uudemmassa versiossa ole mahdollista varmistaa ilman asiantuntijan tutkimustyötä. Lisäksi päivitettävissä tuotantoversioissa saattaa olla eri toiminnoille tarpeellisia ohjelmiston käyttäjän omia muutoksia, joissa joko tunnisteen syntaksi on väärä tai muutos on vain jäänyt merkitsemättä.

1.3 Työn tavoitteet

Ohjelmistopäivityksessä on olennaista että päivitettävä toiminto ei hajoa ainakaan päivityksen sisältöön liittymättömistä syistä. Inhimillisten virheiden minimoimiseksi asentajan päivitysprosessista on hyvä karsia kaikki sellaiset toiminnot jotka eivät suoranaisesti vaadi harkintaa. Tiedonsiirrot, yksittäisen päivitystilanteen järjestäminen ja räätälöimättömien tiedostojen vertailut eivät ole tässä suhteessa ainakaan asentajan näkökulmasta olennaisia asioita. Päivityksen asentajalle on tärkeää arvioida ja korjata ne tiedostomuutokset, joissa päivityksellä on mahdollista rikkoa vanhan version jokin toiminto.

Työn tavoitteena on luoda järjestelmä millä pienten korjausten laajamittainen levittäminen olisi asentajan näkökulmasta edes suhteellisen helppoa. Työkalun tarkoitus

on vähentää asentajan työtaakkaa automatisoimalla pohja- ja siirtotöitä siten että asentajan pääasiallinen tehtävä on luotujen korjausten arvioiminen sekä hyväksyminen.

Ideaali päivitysprosessi on tavoitteiden toteutuessa seuraavanlainen:

- ohjelmoija luo korjauksen ohjelmiston uusimpaan kantaversioon
- asentaja asettaa työkalun asetukset ja lisää korjauksen työkaluun
- työkalu luo korjauksesta kantaversiokohtaiset korjaukset
- asentaja tarkastaa että korjaukset muodostuivat oikein
- työkalu luo kantaversikohtaisista korjauksista tuotantoversiokohtaiset korjaukset
- asentaja tarkastaa että korjaukset muodostuivat oikein
- asentaja hyväksyy päivityksen
- työkalu asentaa päivityksen asiakkaan tuotantoversioon.

Asentajan tehtäväksi jää vain olennaisien muutoksien arvioiminen ja virhetilanteiden korjaaminen.

Työn tarkoitus on vastata kiireellisten korjauspäivitysten jakamisen ongelmaan ja tutkia olisiko vastaavasta toteutuksesta mahdollista kehittää työkalua joka voi myöhemmin olla osana koko järjestelmän päivitysproseduuria.

2 TYÖN TOTEUTUS

2.1 Yleiskuvaus työn toiminnallisuuksien toteutuksesta

Työn olennaisimmat toiminnallisuudet on totetutettu PHP-ohjelmointikielellä hyödyntäen olio-ohjelmoinnin periaatteita. Ohjelman rakenteessa päivityksen versionumerot käydään läpi listamaisesti, ja jokaiselle versiolle luodaan toiminnallinen olio. Olion tyyppi ja siten toiminnot ovat käyttöliittymän eri näkymissä erilaisia. Työkalun käyttötarkoituksen kannalta olennaisimmat toiminnallisuudet ovat versioiden välinen tiedostojen vertailu, vertailussa havaittujen eroavaisuuksien muuntaminen vQmod-kirjaston syntaksin mukaisiksi haku- ja korvaus-säännöiksi sekä näiden sääntöjen avulla muutosten lisääminen päivityspakettiin.

2.1.1 Toimintojen käynnistys-sivu

Työkalussa kaikki toiminnot suoritetaan työkalun asennuskansion indeksitiedoston (index.php) kautta. Toteutuksen ajatuksena on se, että työkalun yksittäisen näkymän sivusisältö ladataan näkyville vain ensimmäisellä latauskerralla. Loput näkymän sivusisällöin päivitykset sekä toimintojen tulokset tuodaan esille erilaisten AJAX-kutsujen vastauksien sisällöistä.

Etusivun tehtävä on lisätä sivulataukseen toimintojen tarvitsemat kirjastoviitteet ja näyttää käyttäjälle sivusisältöä sen mukaan, mikä näkymä on käytössä ja minkälainen tilanne kyseisessä näkymässä on. Sivulla ajetaan eri näkymien mukaan erilaiset toiminnallisuusfunktiot yksinkertaisella switch-case –rakenteella, jossa muuttuvana tekijänä on näkymän tunnistemuuttuja. Kun eri näkymien välillä navigoidaan, syötetään tunnistemuuttuja uuteen sivulataukseen GET-parametrinä, mutta latauksen sivusisällöstä päivitetään vain lataus-scriptissä määriltety HTML-säiliön sisältö. Esimerkin 1 JavaScript-koodissa on sivunäkymän vaihtajalinkin kuuntelija. Linkkiä klikatessa kuuntelija lataa sivun uusiksi jQuery-kirjaston load()-metodilla, mutta päivittää sivusisällöstä vain parametrinä annetut HTML-säiliöt.

```
$("body").delegate("#generate_view", "click", function(e) {
        e.preventDefault();
        var location = window.location.href.substring(0,
             window.location.href.lastIndexOf("/"));
        $("#tool_container").load(location + "?view=generate #tools");
```

```
    });
```

Esimerkki 1 Sivunäkymän vaihtajalinkin kuuntelija

Uuden näkymän tunniste toimitetaan sivulatauksen osoitteen mukana annetun ”view”-nimisen GET-parametrin mukana. Uudessa sivulatauksessa GET-parametri tallennetaan PHP:n istuntomuuttujaan ja se on siten voimassa myös näkymän ensimmäisen sivulatauksen jälkeen tehtävissä toiminnoissa. Sivunäkymien toiminnallisuusfunktiot ajetaan kaikille työkalun asetuksissa määritetyille tuotantoversioille.

Esimerkissä 2 työkalun asetuksiin määritellyt päivitykseen kuuluvat tuotantoversiot järjestetään ensiksi versionumerojärjestykseen PHP:n uasort()-funktiolla, joka käyttää vertailuun kyseiseen tarkoitukseen tehtyä CompareVersionNumbers() -funktiota. Funktio vertaa sille syötetyt versionumerot käyttämällä PHP:n version_compare()-funktiota ja palauttaa kutsujafunktiolle uuden järjestyksen suurimmasta pienimpään.

```
uasort(Config::$shops, 'CompareVersionNumbers');
$shop_folders = array_keys(Config::$shops);
$generated_patch_version_views = array();

foreach ($shop_folders as $key => $shop_folder)
{
    try
    {
        switch ($view)
        {
            case 'generate':
                $current_version = Config::$shops[$shop_folder]['shop_version'];
                if (!in_array($current_version, $generated_patch_version_views))
                {
                    $generated_patch_version_views[] = $current_version;
                    GenratePatch($current_version);
                }
                break;
            case 'upgrade':
                StartUpgrade($shop_folder);
                break;
            default:
                # code...
                break;
        }
    }
    catch (Exception $e)
    {
        echo "Exception: ".$e->getMessage()."<br>";
        exit;
    }
}
```

Esimerkki 2 Päivityksen tuotantoversioiden lajittelu ja toimittaminen eri näkymien toiminnoille

Esimerkin 2 GenratePatch() ja StartUpgrade() –funktiot luovat kyseiselle versiolle funktion suorituksen ajaksi näkymän toimintoihin liittyvän olioluokan joka toteuttaa kaikki näkymän vertailu-, generointi- ja tulostustoiminnot kyseiselle versiolle. ”generate”-näkymässä toiminnot ajetaan eri versiolle aina vain yhden kerran.

2.1.2 Työkalun asetukset

Työkalu käyttää tarvittavien asetuksien määrittämiseen tekstitiedostoa config.php. Esimerkissä 3 näytetään asetusten määrittäminen tiedoston muuttujiin. Eri arvot tallennetaan ohjelman sisällä kaikkialle näkyvälle staattiselle asetus-luokalle Config. Luokalla on neljä muuttujaa jotka ovat assosiatiivisiä listoja. Asetuksen nimi tallennetaan listaan indeksiarvona ja asetuksen arvo tietueena kyseiselle indeksille. Luokan muuttujat ryhmittelevät eri asetukset käyttötarkoituksen mukaan. Esimerkiksi päivityspaketin generoimiseen liittyvät asetukset ovat tallennettu eri muuttujaan kuin päivitettävien kohteiden tunnistetiedot.

```
/*
 * Upgrading process paths
 */
Config::$general_operation_info["base_versions_directory"] = "base_versions";
Config::$upgrade["working_directory"] = "kaupat";
Config::$patches["working_directory"] = "generated_patches";
```

Esimerkki 3 Asetusten määrittämistapa ohjelmiston asetustiedostossa config.php

Ohjelmisto käyttää asetuksia joko suorana viitteenä staattiseen luokkaan tai tallentamalla asetuksen kutsuvan luokan sisäisiin muuttujiin. Esimerkissä 4 näytetään esimerkki asetuksien tallentamisesta olion muuttujiksi rakentajafunktiossa.

```
function __construct($current_shop = null)
{
    foreach (Config::$general_operation_info as $variable => $value)
    {
        if (property_exists(__CLASS__, $variable))
        {
            $this->$variable = $value;
        }
    }
}
```

Esimerkki 4 Asetusten tallentaminen olioluokan sisäisiksi muuttujiksi

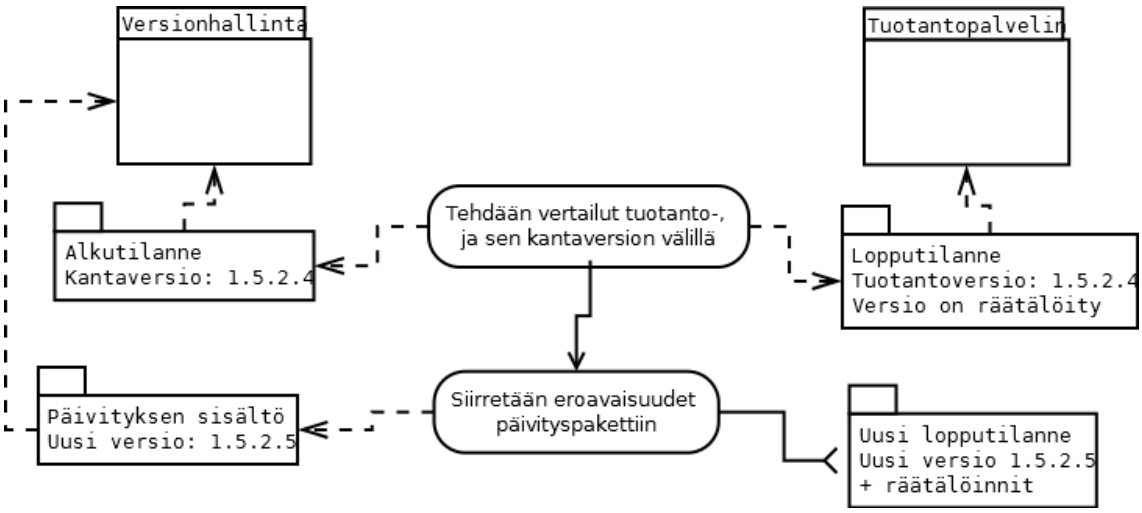
Eri päivitysten kohdeversiotiedot määritellään esimerkin 5 mukaisesti asetustiedoston $shops-muuttujaan. Lähdeversio eli päivityspaketin versiotieto tallennetaan $patches-muuttujaan.

```
/*
 * Patch versions
 */
Config::$patches["patch_base_version"] = "1.6.1.0";
Config::$patches["lowest_base_version"] = "1.5.0.0";
Config::$patches["skip_versions"] = array("1.5.7.0");

/*
 * Specific shop information
 */
$kauppa = "kauppa1";
Config::$shops[$kauppa]['shop_version'] = "1.6.0.3";
Config::$shops[$kauppa]['hostname'] = "localhost";
Config::$shops[$kauppa]['username'] = "tunnus";
Config::$shops[$kauppa]['password'] = "salasana";
//Config::$shops[$kauppa]['public_key'] = "";
//Config::$shops[$kauppa]['private_key'] = "";
Config::$shops[$kauppa]['public_html_folder'] = "public_html/shop_folder";

$kauppa = "kauppa2";
Config::$shops[$kauppa]['shop_version'] = "1.5.2.4";
...
```

Esimerkki 5 Päivityksen ja tuotantoversioiden tietojen määrittäminen ohjelmiston asetustiedostossa config.php

2.1.3 Vertailuympäristön kansiorakenne

Päivityksen räätälöinnit huomioon ottavassa versiovertailussa tarvitaan tieto eri versioiden tiedostojen sijainnista. Päivitysprosessissa ensimmäinen vertailu tehdään kuvan 1 havainnollistamalla tavalla räätälöidyn version alku ja lopputilanteen väliltä. Alkutilanne tarkoittaa käytännössä version asennuspakettia, josta tiedetään ettei siinä ole ulkopuolisen tahon tekemiä muutoksia ja lopputilanne on mahdollisesti muunneltu versio samasta asennuspaketista. Alku- ja lopputilanteiden sijaintien lisäksi työkalu tarvitsee sijainnit itse päivityksen sisällölle sekä sijainnin johon lopputulokset tallennetaan ennen tuotantoon siirtämistä.



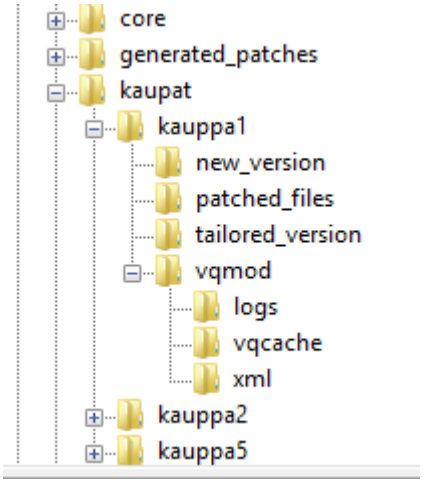
Kuva 1 Versiovertailun tilannekansiot

Vertailujen onnistuminen vaatii sitä, että eri versioiden alkutilanteet pysyvät ensimmäisen asennuksen jälkeen muuttumattomina. Tästä syystä alku- tai kantaversioiden sijainti kehitysympäristössä on myös usein staattinen. Lopputilanteet sijaitsevat usein ulkopuolisilla palvelimilla ja versioiden kansiosijainti on eri instansseilla erilainen. Lopputilanteet ovat myös käytössä olevia tuotantoversioita joten turvallisuuden nimissä päivitystyökalun pitää tehdä jokaisen päivitettävän instanssin alkuperäisistä tiedostoista varmuuskopiot.

Vertailuympäristön kansiorakenne on toteutettu siten, että kantaversioille, räätälöidyille versioille ja generoiduille versiolle on määritelty omat työkansionsa. Työkansioissa eri versioiden tiedostot luetaan alikansiosta jotka on nimetty versioiden nimien mukaan. Räätälöidyissä ja generoiduissa versiokansioissa on lisäksi jaottelu päivityksen tai korjauksen generoinnin eri tiloille. Päivityksen eri tilakansiot ovat:

- kansio jossa säilytetään alkuperäistä versiota
- kansio jossa on uusi versio
- kansio johon tallennetaan kyseisen version hyväksytty päivitys.

Lisäksi tilallisissa päivityskansioissa on aina vQmod:n generoimien tiedostojen ja logitietojen säilytyskansio. Kuvassa 2 näkyy työkansiorakenne puurakenteessa esitettynä. Kuvan rakenteessa päivityksen tuotantoinstanssit on järjestetty omiin työkansioihin, joihin työkalu on luonut tarvittavat tilakansiot työkansion alihakemistoiksi.



Kuva 2 Avattu puunäkymä työkalun päivitysnäkymän työkansioista

Ohjelmisto määrittää suoritettavan operaation perusteella sisäisen työkansion, johon sen hetkisen ajon tehtävät suoritetaan. Esimerkissä 6. näytetään ensimmäisellä rivillä työkansion määritetteleminen operaatiosta vastaavan luokan sisällä. Päivitysoperaatiossa käytettävä työkansio luetaan ohjelmiston asetuksista muuttujalla Config::$upgrade, ja kyseisen versioinstanssin polku määritellään muuttujalla $current_shop.

```
$this->current_working_directory =
Config::$upgrade['working_directory'].$current_shop;

$this->upgrade_folder = $this->current_working_directory.$this->upgrade_folder;
```

Esimerkki 6 Työkansion määrittäminen operaatioluokan sisällä

Vastaavasti esimerkin 6 toisella rivillä näytetään kuinka versioinstanssin sisäiset kansiot määritellään lisäämällä työkansio etuliitteeksi kansiomuuttujaan.

2.1.4 Kansiorakenteen tutkiminen

Työkalun asetuksien avulla voidaan rajoittaa ja tarkkaan määrittää mitä tiedostoja tai kansiorakenteita päivitys koskee ja etsitäänkö tiedostomuutoksia myös määriteltyjen kansioiden alihakemistoista. Järjestelmän lähdekoodissa asetusten seuraaminen on totetutettu muuttujalogiikalla ja PHP:n iteraattoriluokilla. Esimerkissä 7 käynnistetään

kansiorakenteen tutkimisen funktio TraverseFolder() joko päivityskansioon tai tiettyihin päivityskansion alihakemistoihin.

```
if (!$this->version_control_folders)
{
            $this->TraverseFolder($this->new_version_folder);
}
else
{
            foreach ($this->version_control_folders as $folder)
            {
                        // Path in the current patching workspace
                        $real_folder = $this->new_version_folder.$folder;
                        $this->TraverseFolder($real_folder);
            }
}
```

Esimerkki 7 Kansiorakenteen tutkiminen määriteltyjen kansiotietojen perusteella

Esimerkissä 8 näytetään kuinka TraverseFolder()-funktiossa luodaan sille parametrinä annetusta kansionimestä PHP:n rekursiivinen kansioiteraattori. Iteraattorin tietueista suodatetaan pois päivitykseen kuulumattomat tiedostonimet, ja jäljelle jääneet tiedostonimet toimitetaan eteenpäin tiedoston käsittelijäfunktiolle HandleFileInfo().

```
private function TraverseFolder($folder)
{
            $directory = new RecursiveDirectoryIterator($folder);
            $filter = new UpgradeFileFilter($directory, $this->recursive_diff,
                        $this->filename_override);
            $src_files = new RecursiveIteratorIterator($filter,
                        RecursiveIteratorIterator::SELF_FIRST);

            foreach ($src_files as $fileinfo)
            {
                        $this->HandleFileinfo($fileinfo);
            }
}
```

Esimerkki 8 Kansion käsittelyfunktio TraverseFolder()

Kansioiteraattorin tietueet suodatetaan UpgradeFileFilter –luokalla (ks. Liite 1.), joka on PHP:n RecursiveFilterIterator –luokan perillinen. Suodatus- tai filtteröintiluokalla rajataan luettavat tiedostot tiedostopäätteiden mukaan. Luokan rakentajalle annetaan tiedot siitä, jatketaanko lukemista hakemistojen alihakemistoihin ja rajataanko haku vain tiettyihin tiedostonimiin. Lopuksi suodatetusta kansio-iteraattorista luodaan PHP:n rekursiivisen iteraattorin iteraattori RecursiveIteratorIterator. Suodatetun

iteraattoriluokan kaikille jäsenille toteutetaan tiedoston käsittelykutsu HandleFileinfo(), jossa suoritetaan tiedoston kaikki tiedostovertailuoperaatiot.

2.1.5 Tiedostojen vertaileminen

Työkalu käsittelee version kansiorakenteen ja siihen kuuluvat tiedostot HandleFileinfo()-funktiolla. Funktiossa määritellään yksittäisen tiedoston eri tiedostopolut päivitykseen tai generointiin liittyvissä kansioissa, ja tarkastetaan onko kyseisen tiedoston käsittelyyn annettu ohjeita työkalun asetuksissa, tai että onko tiedosto jo käsitelty aikaisemmissa ajoissa. Jos kyseinen tiedostopolku on tyypiltään ”kansio” varmistetaan vastaavien kansioiden olemassa olo päivityksen lopputuloksen kansiorakenteessa. Jos vertailua ja tiedostogenerointia ei olla vielä tehty, eikä kyseiselle tiedostolle ole eritys-sääntöjä, suoritetaan vertailu muunnetun- ja alkutilannetiedoston välillä.

Tiedostojen vertailu tehdään Stephen Morleyn toteuttamalla PHP Diff-luokkatoteutuksella. Luokan compareFiles() –metodille syötetään vanhan ja uuden tiedoston tiedostopolut ja se antaa paluuarvona tiedostojen rivien erot ja yhteneväisyydet kaksiulotteisessa listamuodossa. Esimerkissä 9 metodille annetaan ensimmäisenä parametrinä tiedoston kantaversion ja toisena tuotantoversion polku.

```
$diff = Diff::compareFiles($filepath_base, $filepath_tailorings);
```

Esimerkki 9 Kahden tiedoston eroavaisuuksien tallentaminen listamuuttujaan

Muodostetun listan alkioissa on tallessa tarkasteltavana ollut tiedoston rivi ja sen Diff-kirjastomuotoinen tilavakio, joka ilmoittaa onko kyseessä muokkaamaton, poistettu vai lisätty rivi.

Listan rivien käsittelyssä on käytetty samaa logiikkaa kuin Morleyn kirjaston toTable()-funktiossa. Listan rivit käydään läpi while-silmukassa, jossa kaapataan eri yhteneväiset muutostilat esimerkin 10. mukaisella switch-case rakenteella sekä Morleyn getDiffLinesForType() –apufunktiolla omiin listatietueisiin. Apufunktion avulla listaa luetaan eteenpäin niin kauan kuin muutostila pysyy samana.

```
switch ($diff[$index][1])
{
```

```
case Diff::UNMODIFIED:
            $last_unmodifieds = $this->getDiffLinesForType($diff,
                        $index, Diff::UNMODIFIED);
            break;
case Diff::DELETED:
            $modified = true;
            $last_deletes = $this->getDiffLinesForType($diff, $index,
                        Diff::DELETED);
            $last_inserts = $this->getDiffLinesForType($diff, $index,
            Diff::INSERTED);
            break;
case Diff::INSERTED:
            $modified = true;
            $last_inserts = $this->getDiffLinesForType($diff, $index,
                        Diff::INSERTED);
            break;
}
```

Esimerkki 10 Tiedoston muutostilojen käsittely

Apufunktiolle syötetty indeksi-parametri on referenssitieto jota kasvatetaan kunnes muutostila vaihtuu. Näin saadaan tallennettua tiedoston riveistä muutostilojen alueita joissa viereisillä riveillä tilat ovat samoja.

Vertailun merkittävistä osista on tämän dokumentin liitteenä koodinäyte (ks. Liite2). Liitteessä näytetään miten vertailun muutostilat toimitetaan vQmod XML-tietueiden generoinnista vastaavalle oliolle käsiteltäväksi. Jos yksittäisessä tiedostossa huomataan muutoksia, tallennetaan tiedoston tunniste ylös työkalun myöhempää käyttöä varten.

2.1.6 Vertailujen muuntaminen vQmod-säännöiksi

Työkalu syöttää tiedostojen vertailussa muodostetut tiedoston muutostilatiedot VQModXMLGenerator –tyyppiselle oliolle käyttämällä sen AppendRules() –metodia. Muutostiedot syötetään aina silloin, kun luettavassa vertailutiedossa tapahtuu tilojen osalta muutoksia. Esimerkiksi oliolle syötetään tiedot edellisistä ”muokkaamattomista” riveistä silloin, kun tiedoston seuraava rivi on ”muokattu” rivi, ja vastaavasti olio saa parametrikseen tiedot edellisistä ”muokatuista” riveistä silloin, kun seuraava tiedoston rivi on taas ”muokkaamaton”.

Näiden muokattujen ja muokkaamattomien rivien perusteella olio joko kerää tietoa edellisistä muokkaamattomista riveistä tai luo vQmod-syntaksin mukaisia lisäämis- tai korvaamis-sääntöjä. Muokkamattomia rivejä käytetään sääntöjen hakuehtoina niissä tilanteissa, missä muodostettu sääntö ei sisältänyt itsessään hakulausetta. Eri

muokkaamattomien ja muokattujen rivien muutostilat ja niihin liittyvät olion käyttäytymissäännöt ovat seuraavat:

- oliolle on syötetty vain muokkaamattomia rivejä
    - nollataan edellisen muokkaamattoman rivin etäisyysindeksi
    - käydään rivit läpi ja lisätään hakusanaksi kelpaava rivi käytettyjen hakusanojen tauluun jos se ei vielä ollut siellä
    - nostetaan kyseisen hakusanan tiedostoilmentymän indeksiä
    - jos muokkaamattomissa riveissä ei ollut sopivia hakusanoja, kasvatetaan edellisen sopivan hakusanan etäisyysindeksiä
- oliolle on syötetty poistettuja rivejä
    - käytetään vQmod:n replace-operaatiota
    - pidetään replace-operaation lisäysmuuttuja ”add” tyhjänä
    - jos poistetujen rivien ensimmäisessä alkiossa ei ollut sopivaa hakusanaa, etsitään sopiva hakusana edellisistä muokkaamattomista riveistä
    - jos poistettujen rivien ensimmäisestä alkiosta löytyi sopiva hakuehto ja sitä ei ole vielä käytetty aikaisemmilla riveillä, lisätään se tiedostossa käytettyihin hakuehtoihin
    - kasvatetaan hakuehdon ilmentymän indeksiä
- oliolle on syötetty sekä poistettuja että lisättyjä rivejä
    - korvataan poistettujen rivien replace-operaatioon lisäysmuuttuja ”add” lisätyillä riveillä.
- oliolle on syötetty vain lisättyjä rivejä
    - käytetään vQmod:n after-operaatiota
    - etsitään edellinen hakusana viimeiseksi listatuista muokkaamattomista riveistä ja lisätään hakuehtoihin etäisyys- ja ilmentymäindeksit.

Jos aikaisempia hakuehtoja ei ole, käytetään hakuoperaationa vQmod:n ”top” -käskyä, jolloin muutoksen sijainti generoidussa tiedostossa lasketaan tiedoston alkupäästä. Käyttäytymis-sääntöjen toteutukset ovat karsittuna versiona tämän dokumentin liitteenä (ks. Liite 4).

Esimerkissä 11 näytetään XML-muotoinen esimerkki tiedoston rivin korvaamisoperaatiosta. Operaatiossa korvataan tiedostosta moi.php hakusanan ’echo

”kauppahan se”;’ ensimmäinen ilmentymä uudella rivillä ’ echo ”kauppahan se, kauppa mikä kauppa.”;

```
<operation info="Upgrade: 1.6.0.3_1.6.0.3-patch1-1 :: moi.php @LassiTesti">
        <search position="replace" trim="false" index="1" >
        <![CDATA[    echo "kauppahan se";]]>
        </search>
        <add trim="false">
        <![CDATA[    echo "kauppahan se, kauppa mikä kauppa.";]]>
        </add>
</operation>
```

Esimerkki 11 vQmod-syntaksi tiedoston rivin korvausoperaatiosäännöstä

vQmod-syntaksin mukaan operaation tyyppi annetaan search-tietueen arvoon ”position”. Työkalu käyttää search-tietueen tyyppiä ”replace” kun sääntötyyppi on rivien korvaaminen tyhjällä tai uusilla riveillä, ja tyyppiä ”after” kun uusia rivejä lisätään vanhojen rivien jatkeeksi. Search-tietueen arvo ”index” on hakuehto sille kuinka mones ilmentymä hakulauseesta osuman tulisi olla. Lisäksi työkalu käyttää tietueen arvoa ”offset” määrittämään kuinka kaukana hakuehdosta lisäyksen sijainti on.

VQModXMLGenerator-olio pitää generoidut haku- ja korvausoperaatiot muistissa kunnes kyseisen tiedoston vertaileminen on päättynyt siten, että olion käyttäjäluokka kutsuu olion Commit()-metodia. Metodissa muodostetuista säännöistä luodaan ja tallennetaan yksi yhtenäinen tiedoston muutoksiin liittyvä XML-tiedosto kyseisen version työkansion alikansioon vqmod/xml.

2.1.7 Tiedostojen generointi vQmod-säännöistä

Kun yhden versio-instanssin kaikki vertailut on tehty, työkalu alustaa staattisen VQMod-kirjaston käyttämään kyseistä työkansiota. Esimerkissä 12 näytetään kuinka tiedostomuutoksista löydetyt säännöt ajetaan uuden version vastaaviin tiedostoihin. Esimerkin ensimmäisellä rivillä kirjaston bootup()-metodille annetaan parametreiksi kyseinen työkansio, boolean-muuttuja jolla määritetään tallennetaanko ajoista logi-tiedot sekä kansionimi johon muodostetut säännöt ajetaan.

```
VQMod::bootup($this->current_working_directory, true, $this->new_version_folder);
foreach ($this->filestocheck as $key => $filename)
{
            // Generate file template
            VQMod::modCheck($filename);
```

```
}
$this->vqmod_errors = VQMod::GetErrors();
$this->filestocheck = array();
```

Esimerkki 12 vQmod-kirjaston alustaminen ja hakusääntöjen ajaminen tiedostoille

Sääntöjen ajaminen tehdään vQmod-kirjaston modCheck()-metodilla. Jos luotujen sääntöjen ajaminen onnistuu kyseiseen tiedostoon, generoi kirjasto cache-, eli välimuistikansioon tiedoston, jossa on sekä uuden että räätälöidyn version muutokset. Generoinnin jälkeen uusi tiedosto löytyy kyseisen version kansion alihakemistosta vqmod/cache.

Virhetilanteet kaapataan kirjastolta esitettäväksi työkalun käyttäjälle, ja ne myös tallennetaan version kansion alihakemistoon vqmod/logs.

2.1.8 Tiedon siirto tuotantoympäristöön

Työkalun siirtoprotokollaksi on asetettu SCP, ja sen PHP-kielinen toteutus on tehty käyttäen Jarmie Munron aiheeseen liittyvän artikkelin mallia (Munron 2013) sekä käyttämällä phpseclib-kirjastoa (Wigginton, Monnerat, Fischer & Petrich).

Työkalussa siirretään tiedostoja käyttämällä tarkoitukseen luotua Transfer-luokkaa (ks Liite 3).

Tiedonsiirtoinstanssi ladataan esimerkin 13 mukaan Transfer -luokan GetInstance() –metodilla. Metodi luo uuden SCP-yhteyden kyseisen tuotantoversion palvelimeen silloin kun yhteyttä ei ollut jo olemassa, tai se oli yhdistettynä väärälle palvelimelle. Yhteyden kirjautumistunnukset asetetaan työkalun asetuksiin, ja ne voivat perustua joko tunnus-salasana tai tunnus-julkinen avain-yksityinen avain –yhdistelmiin.

```
$scp = Transfer::GetInstance($current_shop);
$scp->send($upgraded_filename, $destination_filename);
```

Esimerkki 13 Tiedoston siirtäminen työkansiosta tuotantoympäristöön

Transfer -luokka perii Munron tiedonsiirtoluokat ja sen ominaisuudet. Luokkien avulla tiedostojen lähetys onnistuu Transfer-luokan send()-metodilla ja vastaanottaminen recv()-metodilla.

Työssä tiedonsiirtoluokkaa on käytetty siten, että tuotantoympäristön tiedostoja on mahdollista ladata päivityksen työkansioihin ja takaisin tuotantoympäristöön.

Jos päivitettäviin tiedoistoihin liittyvä kyseinen tuotantoversion tiedosto ei löydy päivityksen työkansiosta, ilmoittaa työkalu asian kuvan 3 mukaisesti ”Not found”-viestillä.

Kuva 3 Päivitettävän instanssin tiedostoa ei olla ladattu päivityksen työkansioon

Kuvan 3 ”Fetch from the server” –linkki käynnistää AJAX-kutsun jolla yritetään noutaa puuttuva tiedosto Transfer-luokan metodeilla. Jos tiedoston noutaminen ei onnistunut, siitä annetaan asiaan liittyvä virheviesti. Tiedostojen lähettäminen takaisin palvelimelle on toteutettu samalla logiikalla.

2.1.9 Käyttöliittymän klikattavien nappien sitominen toiminnallisuuksiin

Työnkalun käyttöliittymään sidotut toiminnallisuudet on toteutettu siten, että eri toiminnot käynnistäviin HTML-nappeihin tai linkkeihin on asetettu kuuntelijat jQuery –JavaScript-kirjaston delegate()-metodilla. Metodi kuuntelee sille parametriksi syötetyn tunnisteen perusteella esimerkiksi käyttäjän klikkauksia. Kun käyttäjä klikkaa tunnistetta vastaavaa linkkiä, niin kuuntelija toteuttaa sille parametriksi annetun funktiokutsun.

Käyttöliittymään liitettyjen nappien tunnisteet on määritelty HTML:n ”class” eli luokka-tietueella. Samassa näkymässä on useampi eri kontekstissa saman toiminnon toteuttava nappi, joten tunnisteet on asetettu kuuluvaksi tiettyyn tunnisteryhmään eli luokkaan. Eri kontekstit tarkoittavat tässä tapauksessa versiolistauksen eri versioita ja niiden eri listattuja tiedostoja. Jokaisella käyttöliittymässä listatulla tiedostolla on omat tunnistetiedot jotka on tallennettu listaukseen käyttäjältä piilotettuina elementteinä. Klikattavat toimintonapit ja piilotetut tunnistetiedot ovat tulostettu yhteisen säiliö-

elementin alle, jonka avulla napin painalluksen kuuntelija tietää minkä version ja sen tiedoston rivistä nappia on kutsuttu ja mitä tietoja kuuntelijan tulisi kerätä lähetettäväksi eteenpäin toiminnon toteuttajalle.

Esimerkissä 14 kuunnellaan jQuery:n delegate()-metodilla näkymän kaikkien ”Commit” –linkkien klikkaustapahtumaa. Kun elementtiä klikataan, kuuntelija toteuttaa sille parametriksi annetun funktion.

```
$('body').delegate('.diff_container .commit', 'click',function(e)
{
    e.preventDefault();
    var trigger = $(this);
    var params = {
        action: "CommitDiff",
        caller: trigger.parent().find(".caller").text(),
        current_shop: trigger.parent().find(".current_shop").text(),
        upgraded_filename: trigger.parent().find(".upgraded_filename").text(),
        upgrade_cache_new_file:
trigger.parent().find(".upgrade_cache_new_file").text(),
        plain_filename: trigger.parent().find(".plain_filename").text()
    };
    $.post(window.location, params, function(data) {
        var message = "Something failed..";
        if (data.new_diff_row_html)
        {
            trigger.parent().after(data.new_diff_row_html).remove();
        }
        if (data.message)
        {
            message = data.message;
        }
        trigger.parent().find(".notice").html(message);

    }, "json");
});
```

Esimerkki 14 Toiminnallisuuslinkkien kuuntelija

Esimerkin 14 tapauksessa kuuntelijalle annettu funktio kerää kyseisen rivin tiedoista parametrit, lähettää ne jQuery:n AJAX-työkaluilla palvelimelle jossa pyyntö käsitellään ja toimii lopuksi palvelimen JSON-muotoisen vastauksen perusteella sekä päivittämällä kyseisen rivin tiedot että näyttämällä mahdollinen palvelimen antama virhe- tai onnistumisviesti. Jokaisella tiedostorivillä on samat piilotetut tietueet. Eri klikkaustoiminnot keräävät näistä tietueista vain kyseiseen toimintoon liittyvät tiedot. Jokaiselle AJAX-kutsulle täytyy syöttää kutsuvan näkymäluokan tunnistetieto ”caller” sekä luokan metodin nimi ”action”.

Klikkauksen lähettämän AJAX-kutsun vastaanottaa palvelimen päässä POST-tyyppisten kutsujen vastaanottajaluokka AjaxHandler (ks. Liite 5). Luokka lähettää kyseiselle näkymäluokalle kyseisen klikkauksen toimintokutsun ja siihen liitetyt parametrit sekä palauttaa toiminnon vastauksen JSON-muodossa.

Esimerkin 14 JavaScript-koodissa kutsuttiin toimintoa ”CommitDiff”. Esimerkissä 15 näytetään yhden näkymäluokan vastaus kyseiselle toiminnolle. Esimerkissä kopioidaan generoitu tiedosto päivityskansioon ja palautetaan kutsujalle uuden tilanteen mukainen rivisisältö.

```
public function CommitDiff($post)
{
    $success = false;
    $data = array();

    $upgrade_cache_new_file = $post['upgrade_cache_new_file'];
    $plain_filename = $post['plain_filename'];
    $upgrade_file = $post['upgraded_filename'];

    $shop = $post['current_shop'];

    $file_contents = file_get_contents($upgrade_cache_new_file);
    if (($file_contents) === false)
    {
        throw new Exception(__CLASS__."::CommitDiff() Could not read file contents:
            $upgrade_cache_new_file");
    }
    $file_contents = self::RemoveTmpVariables($file_contents);

    if ((file_put_contents($upgrade_file, $file_contents) !== false))
    {
        $success = true;
    }

    $data['new_diff_row_html'] = self::GetNewDiffRow($shop, $plain_filename,
            $upgrade_cache_new_file);
    $data['message'] = ($success) ? "Wrote $upgrade_cache_new_file to $upgrade_file"
            : "Failed to write $upgrade_file";
    $data['success'] = $success;
    echo json_encode($data);
}
```

Esimerkki 15 Palvelimen vastaus AJAX-kutsuun

2.2 Käyttöliittymä

Työkalun käyttöliittymä on tehty toimintojen esittelyä varten. Kun työkalun toimintalogiikka integroidaan kehitysympäristöön, tulee käyttöliittymäratkaisut miettiä

kyseiseen ympäristöön parhaiten sopivaksi. Työkalun käyttöliittymän toteutus on tehty www-selaimella käytettävään muotoon.

Käyttöliittymässä on kaksi näkymää joiden välillä navigoidaan selain-näkymän ylävasemmalla olevien graafisesti laatikoitujen linkkien kautta (Kuva 4).

Kuva 4 Käyttöliittymän näkymien navigointilinkit "Generate patch" ja "Upgrade view"

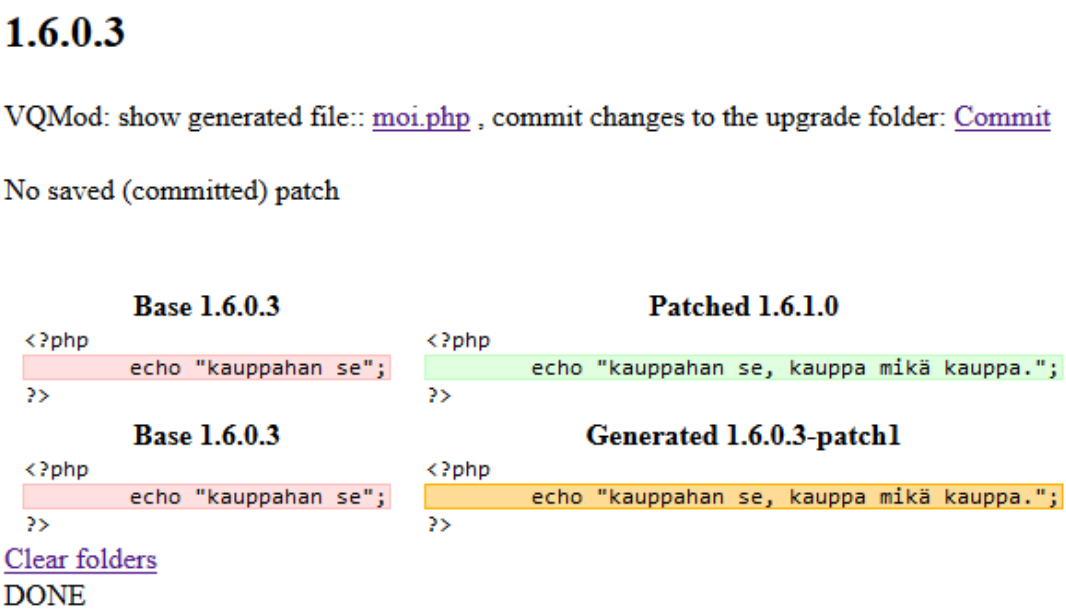
Näkymien nimet sitoutuvat niihin liittyviin toiminnallisuuksiin, ja tässä dokumentissa näkymistä kirjoittaessa nimet ovat paikkauksen generointinäkymälle joko Generate patch- tai paikkauksen generointinäkymä ja päivityksen asennusnäkymälle joko Upgrade view- tai päivitysnäkymä.

Molemmat näkymät listaavat kaikki kyseiseen päivitykseen liittyvät versionumerot ja kyseisten versioiden ne tiedostot, jotka liittyvät kyseiseen päivitykseen.

Listattujen tiedostonimien yhteydessä näytetään tiedoston tila, tiedoston nimi sekä sen hetkiseen tilaan liittyvät työkalulinkit.

Kuvassa 4 näytetään ohjelmistoversiolle 1.6.0.3 tiedosto nimeltä ”moi.php”, joka on ainut tiedosto, josta esimerkin päivitysversioiden välinen tiedostovertailu on löytänyt eroavaisuuksia.

Klikkaamalla tiedostonimeä avautuu kyseisen tiedoston vertailunäkymä (Kuva 5).



Kuva 5 Tiedoston vertailunäkymä

Vertailunäkymässä näytetään eri sisältö eri käyttöliittymän päänäkymissä ja kyseisen tiedoston sen hetkisessä tilassa. Kuvassa 5 näytetään paikkauksen generointi- tai ”Generate patch” –näkymän tiedoston vertailunäkymä silloin, kun työkalun käyttäjä ei ole vielä hyväksynyt generoitua uutta versiota sopivaksi ohjelmiston paikkaukseksi.

Vertailunäkymässä esitetään kahden eri tiedoston sisällöt vierekkäin ja eroavaisuudet merkitään värikoodeilla:

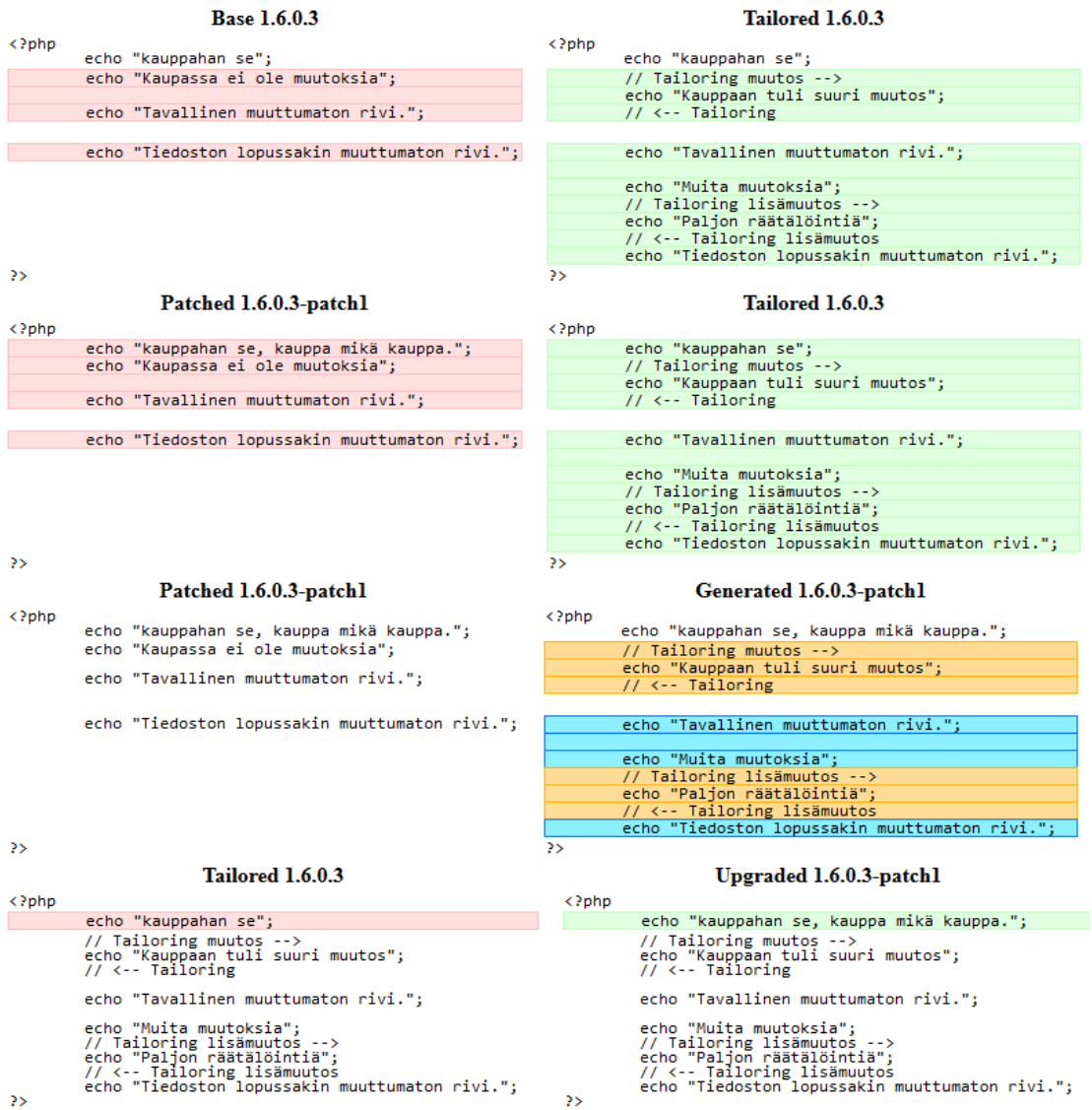
- vaaleanpunainen: vanha rivi
- vaaleanvihreä: uusi rivi
- vaaleansininen: tiedoston generoinnissa muuttunut rivi
- vaaleanoranssi: tiedoston generoinnissa räätälöidyksi riviksi tunnistettu rivi.

Työkalun käyttöliittymässä vertailunäkymän eri tiedostoversioiden nimeämiskäytäntö on seuraava:

- **Base** <versionumero>: kyseinen kantaversio
- **Tailored** <versionumero>: tuotannossa oleva räätälöity versio

- **Patched** <versionumero>-<paikkauksen nimi>: versiokohtainen laastaroitu versio
- **Generated** <versionumero>-< paikkauksen nimi>: työkalun generoima versiokohtainen ja räätälöity versio
- **Upgraded** <versionumero>-<paikkauksen nimi>: tallennettu ja hyväksytty version paikkaus valmiina ladattavaksi tuotantoon
- **Saved** <versionumero>-<paikkauksen nimi>: hyväksytty versiokohtainen paikkaus, valmis käytettäväksi ohjelmistopäivityksessä.

Käyttöliittymän vertailunäkymän nimeämiskäytäntö ja sen värikoodit esitetään kuvassa 6.



Kuva 6 Päivitysnäkymän kaikki vertailut

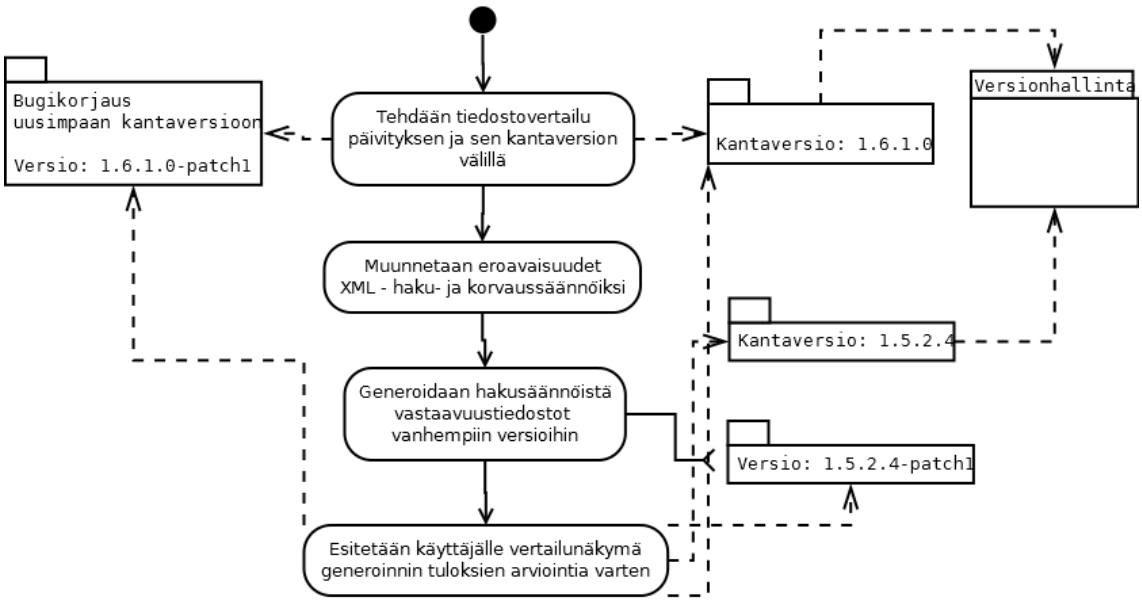
Kuvassa 6 näytetään päivitysnäkymän tilanne, jossa asentaja on hyväksynyt työkalun tekemät vertailu- ja generointipäätelmät klikkaamalla päivitysnäkymän tiedostolistasta tiedostonimen työkaluista ”Commit”-linkkiä. Kuvassa tuotantoympäristön versiota esitetään otsikolla ”Tailored 1.6.0.3” ja asentajan hyväksymää räätälöintikohtaista päivitystä nimellä ”Upgraded 1.6.0.3-patch1”.

Eri näkymien sisällöt ja niihin liitetyt työkalut käydään läpi tämän kappaleen alaotsikoissa.

2.2.1 Paikkauksen generointinäkymä

Paikkauksen generointinäkymä on työkalulle etusivunäkymä.

Ohjelmistopäivityksen versiokohtaisten tiedostojen luominen toteutetaan työkalussa kuvan 7 mukaisin askelin.

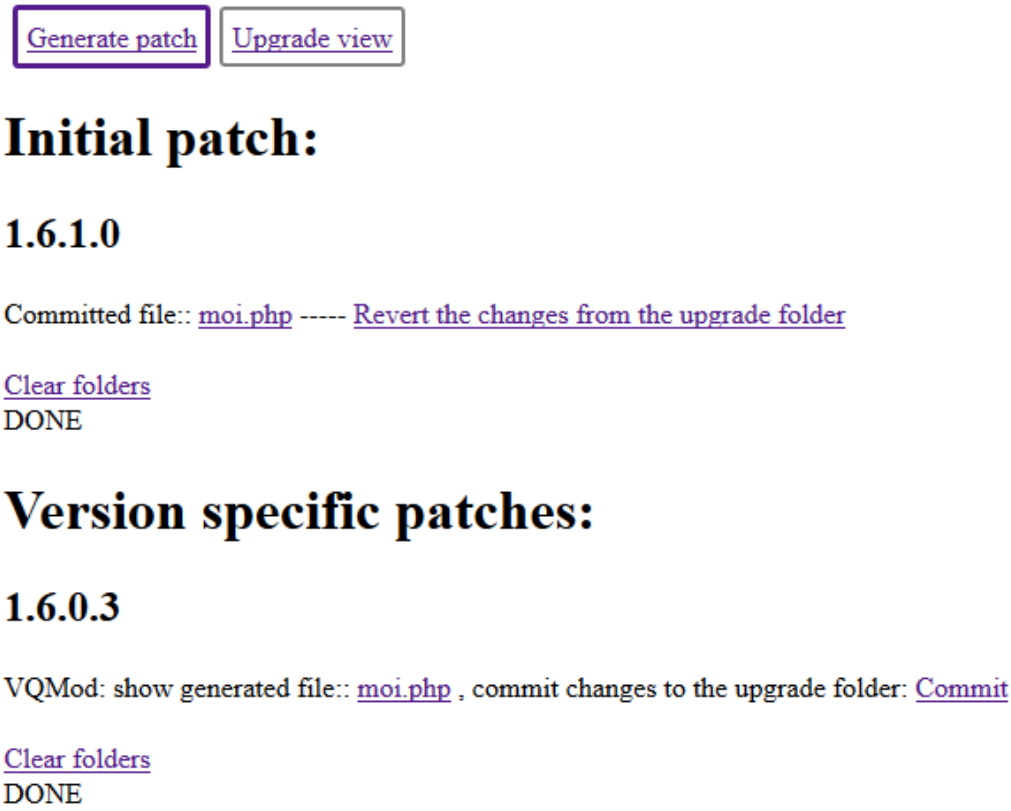


Kuva 7 Kaavio versiokohtaisten päivityspakettien luomisesta

Kuvan 7 askeleet ovat tekstimuodossa seuraavat:

- tehdään vertailu päivityksen tiedostojen ja sen kantaversion vastaavien tiedostojen välillä
- muunnetaan vertailun eroavaisuudet vQmod-kirjaston haku- ja korvaus-säännöiksi
- ajetaan vQmod-säännöt kantaversiota vanhempiin kantaversioihin
- näytetään säännöistä generoidut versiokohtaiset päivitystiedostot päivittäjälle vertailumuodossa.

Generointinäkymän versiolistauksessa näytetään ensimmäisenä yksikkönä se versio josta paikkaus generoidaan päivityksen muihin versioihin. Kuvassa 8 kyseinen yksikkö näkyy otsikon ”Initial patch” alapuolella.



Kuva 8 Generoinnin alkutilanne "Initial patch" ja versiokohtainen tilanne "Version specific patches"

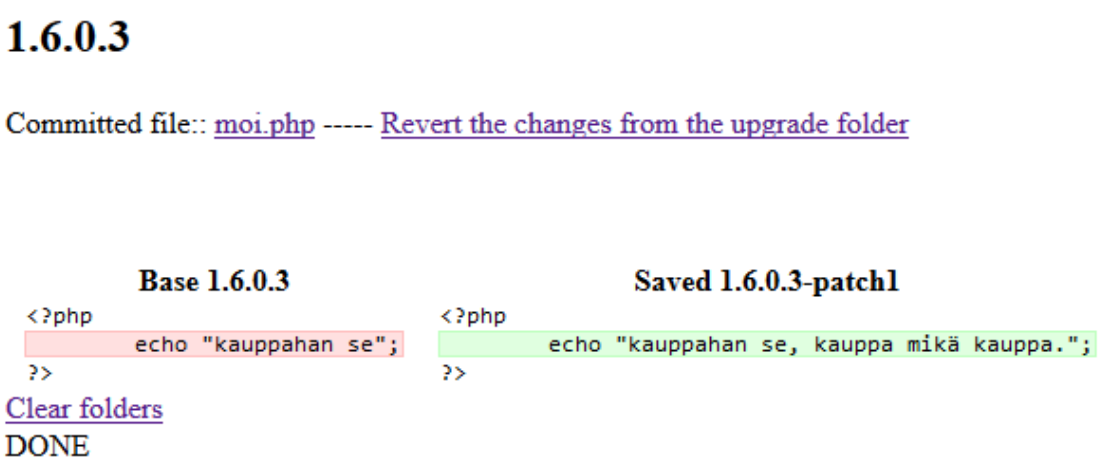
Kuvan 8 otsikon ”Version specific patches” alla listataan päivityksen ne versiot joihin näkymälistan ensimmäisen version muutokset kopioidaan.

Työkalu jättää vertailujen ja generointujen tiedoston onnistumisen harkinnan asentajan tehtäväksi. Asentaja käy läpi generoidut päivitystiedostot ja joko hyväksyy generoidut

muutokset tai korjaa ongelmat käyttäen apukeinona työkalun ilmoittamia virheilmoituksia.

Kuvassa 5 esitettiin näkymä jossa asentajalle näytetään tiedostovertailun ja generoinnin tulokset. Asentaja voi hyväksyä kyseisen muutoksen klikkaamalla tiedoston ohessa olevaa ”Commit” –linkkiä. Linkin klikkaaminen kopioi generoidun tiedoston version hyväksyttyjen paikkausten kansioon.

Kuvassa 9 näkyy paikkauksen generointinäkymän kuva kun tiedosto on hyväksytty päivityskäyttöön.



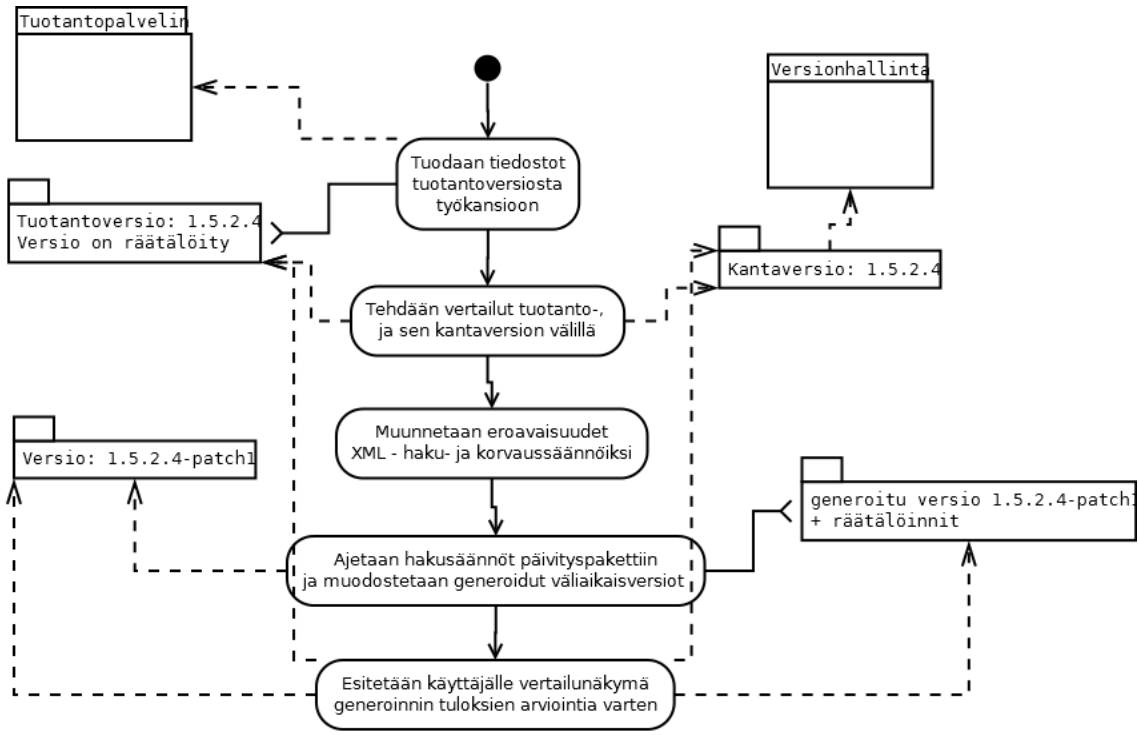
Kuva 9 Hyväksytty 1.6.0.3 version paikkaus tiedostolle moi.php

Kuvassa 9 hyväksytty paikkaus on mahdollista perua klikkaamalla linkkiä ”Revert the changes from the upgrade folder”.

Kuvissa 4,5,8 ja 9 näytetyn ”Clear folders” –linkin klikkaaminen pakottaa koko kyseisen version kaikkien tiedostojen uudelleen lataamisen ja generoimisen. Tiedostoja jotka oltiin jo hyväksytty ei kuitenkaan käsitellä uudestaan.

2.2.2 Paikkauksen asentaminen tuotantoversioon

Versiokohtainen ohjelmistopäivitys asennetaan tuotantoversioon kuvan 10 mukaisesti.



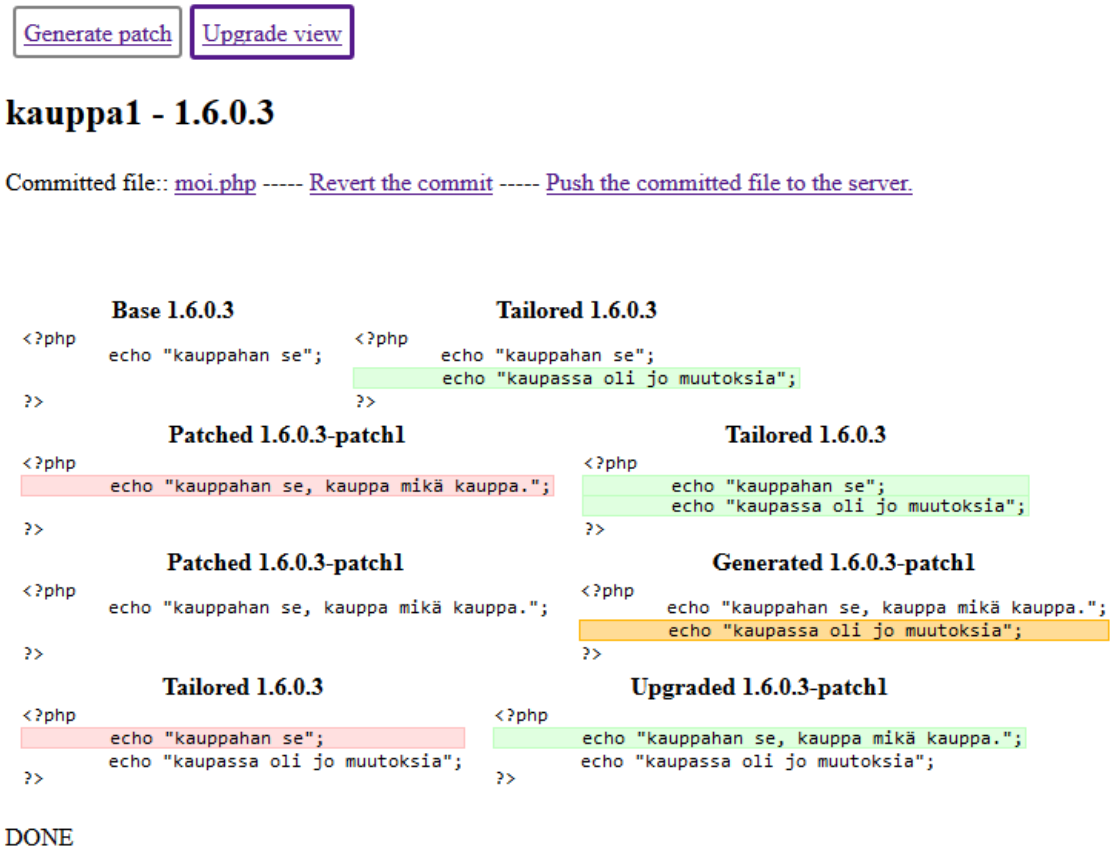
Kuva 10 Versiokohtaisen päivityksen vieminen räätälöityyn tuotantoversioon

Kuvan 10 askeleet ovat tekstimuodossa seuraavat:

- kopioidaan päivitykseen liittyvät tiedostot asiakkaan tuotantoversiosta päivitystyökalun asennuskansioon
- tehdään vertailu asiakkaan tiedostojen ja kyseisen kantaversion välillä
- muunnetaan vertailun eroavaisuudet vQMod-kirjaston haku- ja korvaus-säännöiksi
- ajetaan luodut vQmod-säännöt kyseisen version päivitystiedostoihin
- näytetään säännöistä generoidut versio- ja räätälöintikohtaiset päivittäjälle vertailumuodossa.

Työkalu jättää asentajan tehtäväksi arvioida onnistuiko asiakkaan räätälöintien automaattinen siirto uuteen versioon.

Kuvassa 11 näytetään käyttöliittymän päivitysnäkymästä vertailunäkymä kun asentaja on hyväksynyt räätälöinnit huomioivan generoitudun päivitystiedoston. Kuvassa näytetään kaikki generointiin liittyvät vertailut järjestyksessä siten, että vanha versio on vasemmalla puolella ja uusi oikealla.



Kuva 11 Päivitysnäkymän tiedostovertailut kun kyseinen päivitys on hyväksytty

Kuvassa 11 rivi:

1. vertaa version alkutilannetta ja tuotannossa olevan saman version nykyistä tilannetta
2. vertaa kyseisen version paikkauksen ja tuotannossa olevan version eroja
3. vertaa paikkausversion sekä tuotantoversion ja paikkausversion yhdistelmästä luodun generoidun version välisiä eroja
4. vertaa tuotantoversion alkuperäistä tilannetta hyväksyttyyn päivitysversioon.

Kun asentaja on hyväksynyt versio- ja räätälöintikohtaisen päivityksen, ladataan päivitystiedosto asiakkaan tuontantoversioon klikkaamalla näkymän linkkiä ”Push the committed file to the server”. Päivittäjä voi peruuttaa tekemänsä tuotantoversiomuutoksen klikkaamalla linkkiä ”Push the original file to the server”.

2.3 Työssä käytetty työkalut

Työssä on käytetty seuraavia työkaluja:

- vQmod 2.4.1 standard (Qphoria & Gilford)
- PHP Diff implementation (Morley)
- phpseclib (Wigginton ym.)

Työssä on muokattu vQmod-kirjastoa siten, että sitä voi ajaa useampaan eri tiedostopolkuun ja sen aiheuttamat virheviestit saadaan kaapattua kutsuvan olioluokan myöhempää käyttöä varten. Kirjastosta on muokattu:

- VQMod::bootup –metodi:
  - staattisten muuttujien alustaminen, että kirjasto olisi mahdollista ladata uudestaan uudessa ajoympäristössä
  - parametri $src_files_folder ja sen alustaminen kirjastoon lisättyyn saman nimiseen muuttujaan
- VQMod::checkMods –metodi:
  - tutkittavan tiedostopolun muuntaminen siten että sitä etsitään $src_files_folder –muuttujan kansionimestä.
  - modObject:n applyMod() -metodin paluuarvon kaapppaus luokkaan lisättyyn $error-muuttujaan
- VQMod::_cacheName() –funktio:
  - lisätty tunniste tiedostonimeen
- VQModObject::applyMod() – metodi
  - Lisätty metodille virhetilaanteessa palautusarvo
- Lisätty kirjastoon:
  - GetCachenameFromFileame() –metodi
  - GetFilenameFromCachename() –metodi
  - $src_files_folder –muuttuja
  - $errors –muuttuja
  - GetErrors() –metodi

3 JATKOKEHITYS

Työn tulos ei ole sellaisenaan valmis konkreettiseen käyttöön. Ennen tuotantokäyttöä työkalua pitää kehittää käyttäjälle ystävälliseen suuntaan. Työkalun käyttöön ja asetuksien määrittämiseen pitää luoda yksiselitteiset toimintaohjeet. Asetuksien määrittämiseen on järkevää kehittää jonkinlainen käyttöliittymä, johon on liitetty myös kehitysympäristöön liittyvät versioselaimet.

Työkalun ajonaikaisia ilmoituksia pitää myös kehittää. Esimerkiksi tilanteissa joissa työkalu ei onnistunut versio- tai asiakaskohtaisen korjauksen muodostamisessa lähdekoodin uudelleen järjestelyjen takia, on tärkeää antaa asentajalle ohjeet kuinka ongelma korjataan manuaalisesti.

Työkalua on kehitetty testiympäristössä ja testitiedostoilla. Osa työkaluun liittyvistä ongelmista huomataan todennäköisesti vasta silloin kun sitä integroidaan johonkin kehitysympäristöön. Vastaantulevia ongelmia voisi olla esimerkiksi kyseisen ohjelmiston asennuspakettien työkalulle toimittaminen tai että tietynlainen tilanne pitäisi erikseen huomioida käsiteltävän lähdekoodin vertailuissa.

3.1 Tiedostojen vertaamisen muistinkäyttö

Työssä käytetty Stephen Morleyn tiedostojen vertailutyökalu ei ole optimoitu suuria tiedostokokoja vaten. Työkalu lukee vertailtavista tiedostoista kaikki rivit ensimmäisen ja viimeisen muutoksen välillä. Jos lähdekooditiedosto on suuri ja muutokset sijaitsevat alku- sekä loppupuolella tiedostoa, käyttää työkalu tiedon muistiin tallentamiseen vastaavasti koko erojen välisen määrän rivitietoja. Yksittäinen tutkittava rivi vie tietuekokoonsa nähden moninkertaisen muistimäärän. Jos tiedostokoko on suuri, saattaa vertailun suorittaminen keskeytyä PHP:n asetuksissa määritellyn käyttömuistirajan ylittymisvirheeseen.

Vertailutyökalun muistinkäyttöä pitää joko optimoida tai vaihtaa työkalu järkevämpään vastaavaan. Työn toteutuksessa ongelmasta ollaan päästy eroon yksinkertaisesti lisäämällä ohjelmallisesti palvelimen PHP:n sallittua muistimäärää, mutta itse ongelma on ratkaisematta.

3.2 Käyttöympäristön vaihtaminen

Työkalu on toteutettu PHP-ohjelmointikielellä. PHP-kieltä työkalussa vaativat kuitenkin vain työssä käytetyt kirjastot. Monet työkalun tarjoamat toiminnallisuudet on muunnettavissa esim. Python –scriptimuotoon. Olennaista työkalun PHP-kirjastoissa on vQmod:n tapa generoida tiedostoja. Sen tarvitsemien sääntöjen muodostamiseen on kuitenkin mahdollista käyttää toisia toteutustapoja. Tästä voisi olla hyötyä esimerkiksi jos työkalun vertailunäkymät halutaan totetuttaa ulkopuolisilla työkaluilla.

3.3 Tiedon siirtämisen protokolla

Työkaluun on asetettu tiedonsiirtoprotokollaksi SSH ja sen PHP-kieliseen toteutukseen käytetään phpseclib-kirjastoa (Wigginton ym.). Tunnistautuminen asiakkaan järjestelmän palvelimeen tapahtuu joko tunnus - salasana tai yksityinen avain - julkinen avain -yhdistelmällä. Protokolla mahdollistaa suoran ja salatun tietoyhteyden asiakkaan tuotantoversioon, mutta sisältää sellaisenaan myös riskejä. Päivitysten jakelutyökaluissa eri tuotantoversioiden palvelimien kirjautumistiedot tulee olla työkalulle luettavassa muodossa. Vaikka kirjautumistiedot säilytettäisiin näennäeisesti turvatussa ympäristössä on luettava muoto aina tietoturvariski.

Päivityksien siirtoa varten on mahdollista totetuttaa pieni päivityspalvelu jota voitaisiin käyttää suoraan ohjelmiston hallinnasta. Palvelun avulla ohjelmiston käyttäjälle on mahdollista ilmoittaa jos ohjelmistoon on saatavilla korjauspäivitys jonka ohjelmiston toteuttaja on hyväksynyt soveltuvaksi kyseiselle asennukselle. Palvelun on mahdollista toimittaa ja asentaa päivityspaketti, ja sen avulla päivitys on mahdollista myös peruuttaa.

Päivityksien tiedonsiirto on myös mahdollista toteuttaa liittämällä työkalu erilaisin scriptein suoraan versionhallintajärjestelmään johon kyseiset tuotantoversiot on liitetty.

Päivityspalvelulla tai versionhallintaratkaisulla on mahdollista ulkoistaa tietoturvariskit pois työkalusta.

Työkalu käyttää tiedon siirtoon olio-ohjelmoinnin periaattein yhtä tiedonsiirto-luokkaa jota muuttamalla on mahdollista vaihtaa siirtoon käytettävä protokolla koko järjestelmästä.

3.4 Työkalun kehittäminen koko järjestelmän laajuisia päivityksiä varten

Työkalu soveltuu idealtaan järjestelmän laajuisen päivityksen tekemiseen. Koko järjestelmän laajuinen päivitysprosessi on kuitenkin pääosin sellainen, että samaan aikaan ei ole tarvetta päivittää useampaa ympäristöä. Yksittäisten ympäristöjen päivittämisessä tiedostovertailuihin on olemassa graafisia työkaluja kuten Beyond Compare, Meld ja erilaisten versionhallintajärjestelmien graafisten käyttöliittymien mukana tulevat vertailutyökalut. Olemassa olevat työkalut soveltuvat yksittäisten systeemien päivittämiseen todella hyvin ja osa niistä hoitaa tiedostojen siirrot tarvittaessa päivittäjän puolesta.

Työkalun käyttöliittymää on mahdollista kehittää sellaiseksi, että varsinkin yksinkertaisten päivitysten tekeminen on olemassa olevien työkalujen tapaan kätevää. Päivittäjän tulee pystyä vähintäänkin muokkaamaan työkalun tuottamien vertailujen tuloksia suoraan työkalusta, tai tehdä sama asia kutsumalla tiedostovertailuun jotakin työkalun ulkopuolista apuohjelmaa.

4 POHDINTA

Toimivasti järjestetyssä versionhallinnassa eri versioiden paikkaaminen on toteutettavissa merge-toiminnolla, jossa tiedostojen yhdistämisen ongelmatilanteet arvioi, kuten opinnäytetyön työkalussakin, päivityksen asentaja. Toiminnolla voi yhdistää yhteen versioon tehdyt korjaukset ohjelmiston muihin versioihin. Jos ohjelmiston tuotantoversiot on yhdistetty versionhallintajärjestelmään, niin myös päivitysten asentaminen onnistuu versionhallinnan omilla työkaluilla. Merge-toiminnon onnistumista voi testata ennen kuin tiedostoja päivitetään, ja edellisen version palauttaminen on helppoa.

Tuotantoympäristöjen päivittäminen versionhallintajärjestelmän kautta on ohjelmiston käytössä olevien versioiden hallinnan kannalta hyvä idea, mutta sen toteuttamisessa on omat ongelmansa. Tuotantoversiot saattavat sijaita palvelimilla joihin versionhallinnan järjestäminen ei ole mahdollista erilaisten palvelinrajoitusten vuoksi. Palvelimilla ongelmana saattaa olla esim. eri työkalujen käyttörajoitukset tai yksinkertaisesti rajoitettu levytila. Esimerkiksi GIT-versionhallintajärjestelmä tallentaa muutostiedot paikallisesti, joten työkalua käyttämällä versioinnin järjestäminen vie paljon palvelimen levytilaa (Elmueller, 2011).

Tiedonsiirron ja varmuuskopioinnin järjestämiseen on olemassa paljon työkaluja, mutta monille eri palvelinympäristöille yhteisen työkalun löytäminen ja pitkällä aikavälillä käyttämisen jatkaminen voi olla hankalaa. Ohjelmistotuottajan kannalta olisi järkevää, että ohjelmisto osaisi päivittää itse itsensä, tai että päivitys olisi helppo tehdä tuottajan omilla työkaluilla. Silloin koko päivitysprosessi olisi aina ympäristöstä riippumatta tuottajan hallussa. Jos olemassa olevia versionhallintatyökaluja ei ole mahdollista yhdistää tuotantoversioihin, olisi mahdollisen muun prosessin silti järkevää tallentaa muutoshistoria esim. tuottajan paikalliseen versionhallintaan.

Työssä on lähestytty ongelmaa ProsperCart Oy:n nykyisen päivitystenjakotilanteen näkökulmasta, jossa sadoissa eri tuotantoympäristöissä on käytössä kymmeniä ohjelmiston eri versioita, ja suurin osa niistä on räätälöity ohjelmiston käyttäjän tarpeiden mukaisesti. Tuotantoympäristöjä ei ole sidottu kehitysympäristön versionhallintaan muutoin kuin kehitysympäristön paikallisina kopioina siten, että

tuotantoversio ja kehitysversio ei synkronoidu keskenään ilman manuaalista tiedon siirtelyä ja merge-toiminnon käyttämistä.

Työn tuloksena toteutunut työkalu sopii sellaisenaan kyseiseen ympäristöön, jos kyse on kiireellisten ja pienikokoisten korjauspäivitysten tuottamisesta sekä jakelusta, mutta ei ole ainakaan ilman jatkokehittelyä kovinkaan kattava vastaus ympäristön päivitysarkkitehtuurisiin ongelmiin. Työkalu voi toimia työntekoa helpottavana osana kehitysympäristön isoa kokonaisuutta, jossa yhdistyvät versionhallintajärjestelmän ominaisuudet, versiokohtaisten päivitysten luomisen helppous, sekä turvallinen tiedonsiirto eri järjestelmien välillä.

Työssä on tutkittu kuinka ohjelmistotuottajan oman päivitysjärjestelmän toteuttaminen onnistuisi. Työn tulos on onnistunut tiedostojen välisten merge-toimintojen tekemisessä siten, että päivitystä tekevä henkilö voi keskittyvä vain muutosten arviointiin. Työkalulla tuotantoympäristöön tehdyt muutokset ovat peruutettavissa, sillä se tallentaa kyseisen version alkuperäisistä tiedostoista paikalliset kopiot. Muutosten peruutettavuuden vuoksi työkalun käyttö päivitystilanteessa on turvallista. Jos työkalu integroidaan osaksi kehitysympäristöä, tulisi se liittää kehitysympäristön versionhallintajärjestelmään esim. muutoshistorian tallentumisen takia.

Työssä käytetyistä tekniikoista vQmod-kirjaston tiedostojen generointiominaisuus vaikutti toimivan erinomaisen hyvin. Kirjaston XML-hakusääntöjen syntaksi on selkeä ja käyttäminen yksinkertaista. Työssä ei hyödynnetä kirjaston päätoiminnallisuutta eli suoritettavien tiedostojen ajonaikaista korvaamista generoiduilla vastaavuuksilla. Jos kirjasto on integroitu osaksi päivitettävää ohjelmistoa, toiminnallisuudella olisi mahdollista testata päivityksen eri osia ns. virtuaalisesti lisäämällä päivitystyökalun vQmod-välimuistitiedostot tuotantoversion vQmod-välimuistikansioon. Tuolloin kirjasto lukisi ajettavat tiedostot välimuistista, eikä tuotantoversion tiedostoihin tehtäisi testauksessa mitään muutoksia. Tietokantamuutosten osalta virtuaalisuus olisi toteutettavissa siten, että tuotantoversion tietokannasta luotaisiin testausta varten esimerkiksi SQLite3-tyyppinen kopio.

Toinen kirjaston erinomaisista puolista on sen pääasiallinen käyttötarkoitus eli paikallisten muutosten modularisoiminen kohdejärjestelmässä. Erilaiset ohjelmiston paikalliset muutokset on mahdollista muuntaa vQmod-säännöiksi, jolloin ne toimisivat

helposti siirreltävinä ja huomioitavina ohjelman liitännäisinä. Uudet paikalliset ominaisuudet ja muutokset on mahdollista kirjoittaa testiympäristössä suoraan lähdekoodiin, ja sitten opinnäytetyön tuotoksella generoida vQmod-kirjastomuotoon.

Projektin parissa työskentely on ollut antoisaa. Käsiteltävänä on ollut oikea tarve ja erilaisten päivitystöiden kanssa aikaisempi työskentely on antanut hyvin tietoa tarpeen yksityiskohdista. Työnteko on ollut pääosin itsenäistä, mutta siihen on saatu työnantajalta konsultaatiota ja sekä henkistä että ajankäytöllistä tukea.

LÄHTEET

Morley, S. A diff implementation for PHP. Luettu 20.10.2014.
http://code.stephenmorley.org/php/diff-implementation/

”Qphoria” & Gilford, J. 2014, vQmod™. vQmod official repository. Luettu 20.10.2014.
https://github.com/vqmod/vqmod/wiki

Wigginton, J. & Monnerat, P. & Fischer, A. & Petrich, H. phpseclib. Luettu 20.10.2014.
http://phpseclib.sourceforge.net/

Munro, J. 2013. Using SSH and SFTP with PHP, Luettu 10.8.2014
http://www.sitepoint.com/using-ssh-and-sftp-with-php/

Elmueller, M. 2011, Using git as an autoupdate mechanism, Luettu 18.11.2014.
http://micha.elmueller.net/2011/03/git-autoupdate-mechanism/

LIITTEET

Liite 1. Tiedostojen suodatusluokka

```
class UpgradeFileFilter extends RecursiveFilterIterator
{
    // Filetypes in regex-format
    private $filetypes = "(php|js|css)";
    private $recursive_diff;
    private $filename_override;

    /*
     * A DirectoryIterator filter class constructor
     * @param RecursiveIterator $i
     * @param boolean $recursive_diff If sub-directories should be accepted
     * @param array $filename_override A optional list of filenames. If applied, only
            these files will be accepted
     */
    function __construct(RecursiveIterator $i, $recursive_diff=true,
            $filename_override = array())
    {
        $this->recursive_diff = $recursive_diff;
        $this->filename_override = $filename_override;
        parent::__construct($i);
    }
    public function accept()
    {
        $filename = $this->current()->getFilename();

        // Skip hidden files and directories.
        if ($filename[0] === '.')
        {
            return false;
        }

        if ($this->isDir())
        {
            return ($this->recursive_diff);
        }
        else
        {
            if (!empty($this->filename_override))
            {
                return (in_array($filename, $this->filename_override));
            }
            else
            {
               return (preg_match('/^.+\.'.$this->filetypes.'$/i', $filename) === 1);
            }
        }
    }
}
```

Liite 2. Osia tiedostovertailun toteutuksesta

Liitteessä on logiikkaa joka on S. Morleyn class.Diff.php –kirjastosta

```
/* SNIP */
$this->vqcreator->setFilename($clear_filepath);

// compare tailored files to the base version files line by line
$diff = Diff::compareFiles($filepath_base, $filepath_tailorings);

// Logic from Diff::toTable()
$index = 0;
while ($index < count($diff))
{

    $last_unmodifieds = array();
    $last_deletes = array();
    $last_inserts = array();
    $modified = false;

    switch ($diff[$index][1])
    {
    case Diff::UNMODIFIED:
        $last_unmodifieds = $this->getDiffLinesForType($diff, $index,
             Diff::UNMODIFIED);
        break;
    case Diff::DELETED:
        $modified = true;
        $last_deletes = $this->getDiffLinesForType($diff, $index, Diff::DELETED);
        $last_inserts = $this->getDiffLinesForType($diff, $index, Diff::INSERTED);
        break;
    case Diff::INSERTED:
        $modified = true;
        $last_inserts = $this->getDiffLinesForType($diff, $index, Diff::INSERTED);
        break;
    }
    /* SNIP */
    $this->vqcreator->AppendRules($modified, $last_unmodifieds, $last_deletes,
             $last_inserts);

    $last_unmodifieds = $last_deletes = $last_inserts = null;
    unset($last_unmodifieds, $last_deletes, $last_inserts);
}
$diff = null;
unset($diff);
$this->vqcreator->Commit(); // Throws

/* SNIP */
// Logic from Diff::getDiffLinesForType()
public function getDiffLinesForType($diff, &$index, $type)
{
    $changes = array();
    while ($index < count($diff) && $diff[$index][1] == $type)
    {
        $change = $diff[$index][0];
        $changes[] = $change;
        $index++;
    }
    return $changes;
}
```

Liite 3. Tiedostojen siirtoluokka

```
class Transfer extends SSH2SCP
{
    private static $instance;
    private static $current_shop = "";
    public static function getInstance($current_shop)
    {
        if (!isset(self::$instance) || (self::$current_shop != $current_shop))
        {
            self::$instance = new self($current_shop);
        }
        return self::$instance;
    }

    public static $connection_params = array();
    public function __construct($current_shop)
    {
        try
        {

            if (isset(Config::$shops[$current_shop]['username']) &&
                        isset(Config::$shops[$current_shop]['password']))
            {
                parent::__construct(Config::$shops[$current_shop]['hostname'],
                        new SSH2Password(
                                    Config::$shops[$current_shop]['username'],
                                    Config::$shops[$current_shop]['password']));
            }
            else if (isset(Config::$shops[$current_shop]['username']) &&
                        isset(Config::$shops[$current_shop]['public_key']) &&
                        isset(Config::$shops[$current_shop]['private_key']))
            {
                parent::__construct(Config::$shops[$current_shop]['hostname'],
                        new SSH2Key(Config::$shops[$current_shop]['username'],
                        Config::$shops[$current_shop]['public_key'],
                        Config::$shops[$current_shop]['private_key']));
            }
        }
        catch (Exception $e)
        {
            throw new Exception("SCP::Fatal exception: ". $e->getMessage());
        }
    }
}
```

Liite 4. vQmod-sääntöjen generoimisen logiikkatoteutukset

```
/*
 * Generates xml search rules from the diff data arrays
 * @param bool $modified if there are differences in the current file comparison
 * @param array $last_unmodifieds a list of last unmodified lines
 * @param array $last_deletes a list of last deleted lines
 * @param array $last_inserts a list of added lines
 */
function AppendRules($modified, $last_unmodifieds = array(), $last_deletes = array(),
$last_inserts = array())
{

/*
 * Prepare some search data from last unmodified lines
 */
if (count($last_unmodifieds))
{
    $this->offset_from_last_unmodified_line = 0;
    foreach ($last_unmodifieds as $key => $value)
    {
        // Only add unmodified lines to search params if it contains non-whitespace
             contents
        if (trim($value) != "")
        {
            $this->last_unmodified_line = $value;
            if (!isset($this->search_line_indexes["$this->last_unmodified_line"]))
            {
                $this->search_line_indexes["$this->last_unmodified_line"] = 0;
            }
            $this->search_line_indexes["$this->last_unmodified_line"]++;
            $this->offset_from_last_unmodified_line = 0;
        }
        else
        {
            $this->offset_from_last_unmodified_line++;
        }
    }
}

/*
 * If lines are deleted
 */
if ($linecount = count($last_deletes))
{
    $position = "replace";
    $search = $last_deletes[0];

    $offset = $linecount-1;
    $add = "";

    // If the first deleted line was empty we have to fetch the search rule from last
            unmodified lines.
    if (trim($search) == "")
    {
        // If the operation is at the top of the file.
        if (empty($this->last_unmodified_line))
        {
            $position = "top";
            $search = "";
            $index = 0;
        }
        else
```

```
            {
                // Instead of just deleteting, overwrite the line with the last
                            unmodified line.
                // This might drop some empty lines from the generated file
                $search = $this->last_unmodified_line;
                $add = $this->last_unmodified_line;
                $index = $this->search_line_indexes["$this->last_unmodified_line"];
            }
        }
        else
        {
            if (!isset($this->search_line_indexes["$search"]))
            {
                $this->search_line_indexes["$search"] = 0;
            }
            $this->search_line_indexes["$search"]++;
            $index = $this->search_line_indexes["$search"];
        }

        /*
         * Or if lines are replaced with new lines
         */
        if ($insert_lint_count = count($last_inserts))
        {
            $add = implode("\n", $last_inserts);
        }

        $this->xml_strings[$this->upgrade_id] =
         $this->CreateVQModXMLOperationString($position, $search, $add, $index, $offset);
    }

    /*
     * Or if lines are inserted
     */
    else if (count($last_inserts))
    {
        if (empty($this->last_unmodified_line))
        {
            $position = "top";
            $search = "";
            $index = 0;
            $offset = 0;
        }
        else
        {
            $position = "after";
            $search = $this->last_unmodified_line;
            $index = $this->search_line_indexes["$this->last_unmodified_line"];
            $offset = $this->offset_from_last_unmodified_line;
        }
        $add = implode("\n", $last_inserts);

        $this->xml_strings[$this->upgrade_id] =
         $this->CreateVQModXMLOperationString($position, $search, $add, $index, $offset);
    }
```

Liite 5. AJAX-kutsujen vastaanottajaluokka

```
class AjaxHandler
{
    public $caller = null;
    public $action = null;

    public function __construct()
    {
        if(isset($_POST['action']) && isset($_POST['caller']))
        {
            $result = array("message" => "Unkown error", "success" => false);

            $action = $this->action = $_POST['action'];
            $_POST['action'] = null;
            unset($_POST['action']);

            $caller = $this->caller = $_POST['caller'];
            $_POST['caller'] = null;
            unset($_POST['caller']);

            if(method_exists($this->caller, $this->action))
            {
                try
                {
                    ob_start();
                    $caller::$action($_POST);
                    $result = ob_get_clean();
                }
                catch (Exception $e)
                {
                    $result["message"] = $caller."::".$e->getMessage();
                }
            }

            if (is_array($result))
            {
                $result = json_encode($result);
            }
            echo $result;
            exit;
        }
    }
}
```