Sami Nurmi

Jatkuva integraatio osaksi PHP-ohjelmistokehitystä

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikan koulutusohjelma
Insinöörityö
17.9.2012

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Sami Nurmi Jatkuva integraatio osaksi PHP-ohjelmistokehitystä 37 sivua 17.9.2012
Tutkinto	Insinööri
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Auvo Häkkinen, yliopettaja Jani Niinimäki, Cabforce Oy

Insinöörityön tavoitteena oli suunnitella ja toteuttaa jatkuvaan integraatioon perustuva testausjärjestelmä osaksi Cabforce Oy -yrityksen ohjelmistokehitystä. Päämääränä oli saavuttaa mahdollisimman tarkoituksenmukainen sekä helppokäyttöinen kokonaisuus. Cabforce Oy tuottaa ohjelmistoaan ketterillä menetelmillä. Ohjelmistokielenä on PHP ja versionhallintana toimii SVN.

Jatkuvan integraation ydinsovellukseksi valittiin Jenkins, joka tukee suoraan yrityksen käyttämää versionhallintaa ja on liitettävissä osaksi PHP-ohjelmistokehitystä. Varsinaisen yksikkötestauksen kehyksenä toimii PHPUnit, mutta työ sisältää lisäksi useita muita testaus- ja analysointityökaluja. Testausvälineistön kytkemisen osaksi järjestelmää mahdollisti koonti- ja käännöstyökalun roolissa toimiva Apache Ant.

Työn keskeisimmät asetustiedostot on sisällytetty versionhallintaan, jolloin niiden päivittäminen on sulavaa. Testaus- ja analysointityökalut asennettiin käyttämällä hyväksi PEAR:ia. Näin niiden pitäminen ajan tasalla on myös automatisoitua.

Lopputuloksena syntyi toimiva, yhtenäinen ja kattava jatkuvaan integraatioon perustuva testausjärjestelmä. Järjestelmä jäi käyttöön Cabforce Oy:lle ja sen päälle on kehitetty testejä PHPUnitilla. Cabforce Oy mainitsi etenkin automaattisen koostamisen helpottavan ohjelmistokehitystään.

Avainsanat	jatkuva integraatio, testaus, php, Jenkins

Author Title Number of Pages Date	Sami Nurmi Continuous integration as part of PHP Software Development 37 pages 17.9.2012
Degree	Engineer
Degree Programme	Information technology
Specialisation option	Software engineering
Instructor(s)	Auvo Häkkinen, principal lecturer Jani Niinimäki, Cabforce Oy

The purpose of this final year project was to design and develop an automated testing system based on continuous integration as part of software development at Cabforce Oy. The aim was to create as appropriate and user friendly structure as possible. Cabforce Oy uses agile methods to develop their software. The programming language is PHP and the version control system is SVN.

Jenkins, which supports the version control system in use at Cabforce Oy and which can be integrated into PHP software development was chosen as the main tool for the continuous integration. The actual unit testing framework was PHPUnit, but many other testing and analyzing tools were also utilized in this project. It was possible to integrate these testing tools in this project with the help of Apache Ant which is the build tool used in this project.

The most important configuration files of this project have been included in the version control. Therefore the updating of these files is smooth. The testing and analyzing tools were installed from PEAR so keeping these updated is also automatized.

The outcome of this project is a functional, solid and comprehensive testing system based on continuous integration. The system is in use at Cabforce Oy and they have developed some PHPUnit tests on it. Feedback from Cabforce Oy indicates that especially the automated compiling has been found useful for their software development process.

Keywords	continuous integration, testing, php, Jenkins

Sisällys

1 Johdanto 1

2 Ketterä ohjelmistokehitys ja jatkuva integraatio 2

3 Jatkuvan integraation osat 4

3.1 Jatkuvan integraation ydin 4
3.2 Jenkins 6
3.3 Koontityökalu 12
3.4 Jatkuva integraatio ja PHP 15
3.5 Versionhallinta jatkuvassa integraatiossa 17

4 PHP-sovelluksen analysointi ja testaus 17

4.1 PHPUnit 18
4.2 Muut testaus- ja analysointityökalut 20
4.3 Tulosten käsittely 27

5 Valmis järjestelmä 30

6 Yhteenveto 32

Lähteet 36

1 Johdanto

Jatkuva integraatio on hyvin tavallinen ja melko vakiintunut tapa automatisoida ohjelmistotestausta esimerkiksi Java-ohjelmoinnissa. Tässä työssä otetaan kantaa jatkuvan integraation tarpeellisuuteen osana PHP-ohjelmistokehitystä sekä yleisellä tasolla PHP:n piirteisiin, jotka vaikuttavat jatkuvan integraation toteutustapaan. Yritys, johon tämä työ on tehty, toteuttaa kehityksensä ketterillä menetelmillä. Näin ollen työssä on pyritty ottamaan erityisesti huomioon jatkuvan integraation ominaisuuksia, jotka edesauttavat testausautomaation sulavaa toimintaa osana ketterää ohjelmistokehitystä.

Työssä suunnitellaan ja toteutetaan Cabforce Oy:lle jatkuvaan integraatioon perustuva testausjärjestelmä. Cabforce Oy on vuonna 2009 perustettu, nopeasti kasvava matkailualan teknologiayritys, joka kehittää tilausvälitysjärjestelmää matkailualan yrityksille sekä näihin liittyville tilauskanaville. Yrityksen kehittämä PHP-pohjainen ohjelmisto on kasvanut siihen pisteeseen, että sille on ajankohtaista suunnitella ja toteuttaa automatisoitu testausjärjestelmä.

Yrityksen ohjelmistokehityksessä käytetään SVN-versionhallintaa, joten jatkuvan integraation ytimen on oltava sen kanssa yhteensopiva. Työn tarkoitus on tutkia ja etsiä sopivat työkalut liitettäväksi yhdeksi kokonaisuudeksi tehostamaan yrityksen ohjelmistokehitystä. Sovelluksia valittaessa pääpainona on niiden tarkoituksenmukaisuus. Työssä esitellään mukana olevat komponentit, perustellaan niiden tarpeellisuus ja perehdytään niiden toimintoihin osana järjestelmää.

Testauksen pääpaino asettuu PHPUnitin avulla tapahtuvaan yksikkötestaukseen, mutta tämä projekti käsittää useita muitakin testaus- ja analysointityökaluja. Näillä pyritään parantamaan ohjelmistotuotteen laatua ja tuottamaan mittaustuloksiin perustuvaa dokumentaatiota kehitysprosessista.

Työssä esitetään jatkuvan integraation käyttöönotto, joka aloitetaan suunnittelutasolta ja saatetaan toimivaksi järjestelmäksi. Ensimmäisessä luvussa hahmotellaan ketterän ohjelmistokehityksen päämääriä sekä jatkuvan integraation merkitystä sen osana. Tä-

män jälkeen tutkitaan jatkuvan integraation rungoksi sopivia työvälineitä ja pohditaan seikkoja, jotka ratkaisevat käytettävän ohjelmiston valinnan.

Luvussa 3 esitellään yksityiskohtaisemmin valituksi tullut jatkuvan integraation sovellus. Luvussa mainitaan lyhyesti käytettävät analysointi- ja testaustyökalut sekä kuvaillaan, kuinka testausjärjestelmä liitetään jatkuvaan integraatioon. Luku esittelee Apachen Antin roolin osana järjestelmää ja selvittää sen soveltuvuutta PHP-ohjelmistokehitykseen. Luvun 3 lopussa kerrotaan SVN-versionhallinnan osuudesta järjestelmässä ja siitä, mitä seikkoja on otettava huomioon kytkettäessä sitä mukaan.

Luku 4 käsittelee tähän projektiin liitetyt analysointi- ja testaustyökalut. Aluksi luvussa otetaan kantaa testauksen tarpeellisuuteen sekä automatisoinnin tuomiin etuihin. Tämän jälkeen luvussa esitellään mukana olevat testaus- ja analysointityökalut sekä tarkennetaan kunkin osuutta järjestelmässä. Neljännen luvun loppu keskittyy testitulosten käsittelyyn.

Viidennessä luvussa tarkastellaan laajemmin aikaansaatua kokonaisuutta. Siinä selvennetään koko järjestelmän toimintaa sekä pohditaan ratkaisun vahvuuksia ja heikkouksia. Luvussa selvitetään järjestelmän nykyinen toiminta ja pohditaan seikkoja, joita olisi hyvä ottaa huomioon jatkokehityksen yhteydessä.

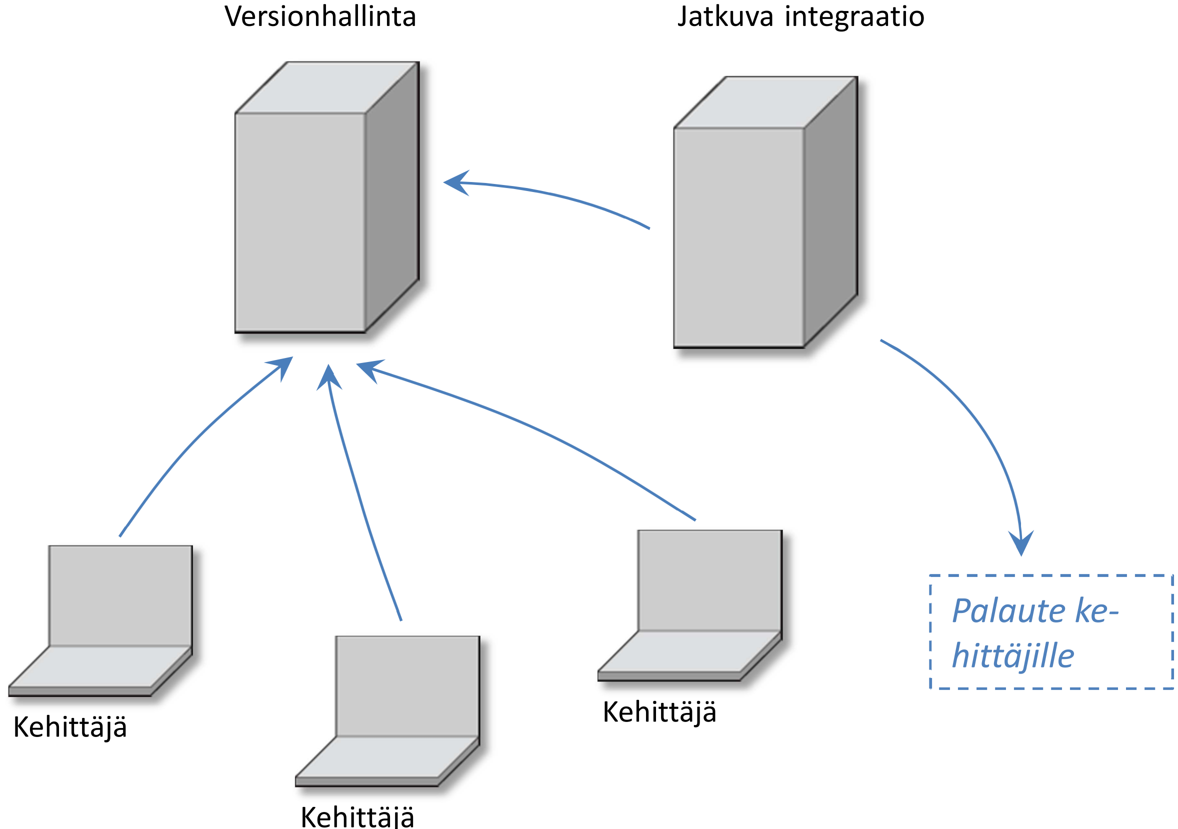
2 Ketterä ohjelmistokehitys ja jatkuva integraatio

Ketterä ohjelmistokehitys on perinteisempiin ohjelmistokehitysmalleihin verrattuna joustavampi menetelmä. Sen eräs keskeinen tavoite on edistää asiakaslähtöisyyttä ohjelmistokehityksessä. Tässä tärkeimpiä kulmakiviä ovat toimivan ohjelmiston tuottaminen sekä reagoiminen asiakkaan taholta tuleviin muutostarpeisiin [1].

Nopea muutoksiin vastaaminen vaatii tuotantoprosessin jaottelua pienempiin sykleihin, jolloin ohjelmistotuotetta koostetaan usein [2]. Näin voidaan välttää sitä, että olemassa oleva tuote ei roiku keskeneräisinä osina tietovarastossa tai kehittäjien koneilla. Muutoksia tehtäessä kehityksen alla oleva ohjelmisto tulisi olla saatavilla kokonaisuudessaan tietovarastosta ja sen pitäisi olla muokattavissa nopeasti.

Ohjelmiston toimivuuden parantaminen vaatii virheiden minimoimista ohjelmistotuotteesta. Virheiden esiintyminen ohjelmistossa liittyy usein koostamiseen, jossa ohjelmiston eri osa-alueita kasataan yhdeksi kokonaisuudeksi [3]. Lisäksi koska ketterissä menetelmissä koostetaan tuotetta usein, testaustoimenpiteet virheiden karsimiseksi on kannattavaa kohdistaa ohjelmiston koostamisen yhteyteen.

Jatkuva integraatio on käytäntö, jossa kehityksen alla olevaan ohjelmistoon tehtävät muutokset sekä lisäykset koostetaan, testataan ja integroidaan säännöllisesti [3]. Yleinen malli jatkuvasta integraatiosta koostuu versionhallintapalvelimesta, testausjärjestelmästä sekä jatkuvasta integraatiosta huolehtivasta koontipalvelimesta (Kuva 1).



Kuva 1. Yleiskuva jatkuvasta integraatiosta.

Kuva 1 esittää jatkuvan integraation yleistä perusrakennetta. Pääpiirteittäin prosessi alkaa, kun ohjelmoija lisää oman koodinsa tietovarastoon. Jatkuvan integraation työkalut sisältävä palvelin tarkkailee versionhallintapalvelinta tietyin väliajoin. Kun versionhallintapalvelimella tapahtuu muutos, jatkuvan integraation palvelin hakee ohjelmakoodit käsiteltäväkseen. Tämän jälkeiset toimenpiteet riippuvat jatkuvan integraation

palvelimen ydinsovelluksien ominaisuuksista. Se, mitä koodille tehdään, riippuu tuotannossa käytettävästä ohjelmointikielestä. Pääsääntöisesti tässä suoritetaan kääntäminen, koostaminen ja testaus. Lisäksi kuvan 1 esittämällä tavalla jatkuvan integraation palvelin voidaan asettaa tiedottamaan kehittäjiä prosessin tapahtumista, kuten esimerkiksi testauksessa esiintyvistä virheistä.

3 Jatkuvan integraation osat

Jatkuva integraatio on termi laajalle kokonaisuudelle, johon liittyy vaihteleva määrä erilaisia osia. Näitä ovat esimerkiksi fyysiset laitteet, erilliset ohjelmistot sekä niiden yhteistoimintaa tukevat käytännöt. Luku esittelee työssä käytetyt välineet sekä hahmottelee toimenpiteet niiden liittämiseksi yhdeksi toimivaksi kokonaisuudeksi. Luvussa otetaan myös kantaa siihen, miksi esitellyt työkalut on päädytty ottamaan käyttöön.

3.1 Jatkuvan integraation ydin

Jatkuvan integraation keskeisin osa on ohjelmisto, jolla liitetään muita työvälineitä yhteen ja mahdollistetaan prosessin automatisointi. Kun valittaan järjestelmää jatkuvan integraation perustaksi, on aluksi mietittävä kokonaisuutta, johon se tullaan liittämään. Keskeisiä valintaan vaikuttavia tekijöitä ovat ympäristössä käytettävät ohjelmointikielet sekä muut järjestelmät, joita jatkuva integraatio tulisi käyttämään.

Työssä pyrittiin löytämään työkalu, joka vastaisi hyvin tarpeita, mutta ei olisi tarpeettoman monimutkainen ja suuri. Järjestelmän ohjelmointikielen perusteella keskityttiin aluksi tutkimaan puhtaasti PHP:tä varten kehitettyjä jatkuvan integraation ratkaisuja, jotta lopputulos olisi kevyt ja tarkoituksenmukainen. Tutkimusvaiheessa löytyi vain yksi mainitsemisen arvoinen, nimenomaan PHP:tä varten kehitetty jatkuvan integraation alusta. Tämä ohjelma on nimeltään Xinc [4]. Jatkettaessa etsintöjä löytyi kaksi lupaavaa Java-pohjaista ohjelmistoa, joilla on tehty jatkuvaa integraatiota myös PHP-järjestelmille. Näistä toinen on Hudson ja toinen CruiseControl-ohjelman integraatio nimeltään phpUnderControl.

Valinnan vapauden niukkuus johtuu ainakin osittain siitä, etteivät PHP-kielen tapaiset ohjelmistot tarvitse yleensä varsinaista käännöstyötä ja ovat suoritettavia sellaisenaan palvelimella. Käytännössä ohjelmistoja on yksinkertaisesti vain siirrelty tiedonsiirtotyö-

kaluilla suoraan palvelimille, joilla niiden on haluttu pyörivän. Nykyisimmin web-ohjelmistojen lisääntyessä ja kasvaessa hyvää vauhtia, on huomattu, että käännöstyökaluja käyttämällä isojen web-ohjelmistojen virhealttiutta ja toimivuutta on voitu parantaa melkoisesti.

Jatkuvan integraation työkalulle asetettiin vaatimus, jonka mukaan sen pitäisi olla käytettävissä myöhemmin myös Android-järjestelmälle. Tämä rajasi hieman vaihtoehtoja, kun kyseinen järjestelmä pohjautuu Javaan. Xincistä oli siis luovuttava, ja valinta oli tehtävä kahden Java-pohjaisen järjestelmän, Hudsonin ja CruiseControlin, väliltä.

Jatkuvan integraation pystyttäminen on sen verran aikaa ja työtä vaativa urakka, että asennuksien ja testailujen sijaan on älykkäämpää kerätä tietoa ja muiden käyttäjien mielipiteitä. Näiden tulosten pohjalta voi tehdä omia tarpeita vastaavan valinnan jatkuvan integraation työvälineelle.

Tutkittaessa lukuisia aihetta käsitteleviä kirjoituksia ja keskustelualueita Hudson osoittautui asennustöiltään helpommaksi [5]. Lisäksi sen paljon kehuttu web-käyttöliittymä näytti vakuuttavalta. Harkinnassa otettiin erityisesti huomioon yrityksen tilanne, johon tämä työ on tarkoitettu. Tuoreen ja pienehkön yrityksen resurssit huomioon ottaen jatkuvan integraation yhdeksi tärkeäksi kriteeriksi nousee helppokäyttöisyys. Tätä puoltaa esimerkiksi Hudsonin selkeärakenteinen web-pohjainen käyttöliittymä. Lisäksi Hudsonin laaja lisäosavalikoima tarjoaa laajennettavuutta, joka saattaa tulla erityiseen tarpeeseen hyvässä vauhdissa olevalle ohjelmistokehitykselle.

Nykyisin Hudson on haarautunut kahdeksi eri projektiksi, joista kumpaakin kehitetään edelleen. Alkuperäistä nimeä kantava ohjelmisto kuuluu Oraclelle ja toista haaraumaa kehittää erillinen monitahoinen yhteisö. Tämä projekti tunnetaan nimellä Jenkins [7]. Haarautuminen ja nimen muutos ovat seurausta infrastruktuuriin liittyvistä ongelmista ja kyseisten tahojen välisistä sopimuksista [6].

Hudson ja Jenkins ovat teknisessä mielessä miltei yksi ja sama ohjelmisto. Keskustelupalstoja selaillen kummastakin löytyy varsin positiivisia kokemuksia PHP-kehityksessä [5]. Jenkinsin internetsivustoilta tosin löytyy kokonainen projekti, joka pyrkii standar-

doimaan käytännön jatkuvalle integraatiolle PHP-sovelluskehityksessä [8]. Jenkins toimii jatkuvan integraation selkärankana tässä työssä.

3.2 Jenkins

Jenkins on palvelimella käytettävä Java-sovellus ja se tarvitsee toimiakseen Java JRE 1.5:n tai uudemman. Jenkinsin voi asentaa ja sitä voi käyttää useammallakin eri tavalla. Esimerkiksi monille Linux-jakeluille löytyy tietovarastoista suoraan asennuspaketti. Tässä työssä palvelimen käyttöjärjestelmänä on CentOs 5.5. Asennuspalvelimella ei ole tarkoitus käyttää muita Java-sovelluksia, joten alustaksi on kannattavaa käyttää mahdollisimman yksinkertaista ja kevyttä alustaa. Jenkinsin kotisivuilta ladattavaan war-pakkaukseen onkin sisällytetty huomattavasti tavanomaisia alustoja kevyempi Winstone-palvelin, jolloin ylimääräisiltä asennuksilta vältytään. Winstonea hyödyntämällä Jenkinsin käyttöönotto tapahtuu lataamalla jenkins.war-pakkaus palvelimelle ja suorittamalla se. Jenkins tarvitsee toimiakseen työhakemiston, johon se tallentaa asetukset ja tietovarastosta poimitut tiedostot. Jos Jenkins ei käynnistyessään löydä työhakemistoaan, se luo sellaisen. Oletusarvoisesti Jenkinsin työhakemisto sijaitsee käyttäjän kotihakemistossa ja sen nimi on .jenkins. Työhakemiston nimen ja sijainnin voi muuttaa ympäristömuuttujan avulla.

Tässä projektissa Jenkinsin työhakemisto oli määritettävä paikkaan, jossa kohteena olevan sovelluksen voi suorittaa täysivaltaisesti, jolloin myös sen testaus on mahdollista suorittaa kattavammin. Tässä kohtaa on otettava huomioon esimerkiksi testattavan sovelluksen tietokantayhteydet. Työhakemiston määrittäminen tapahtuu asettamalla arvo ympäristömuuttujalle *$JENKINS_HOME*. Jottei tätä tarvitse asettaa uudelleen aina ennen Jenkinsin käynnistämistä, sen asettaminen kannattaa kirjoittaa komentosarjaksi paikkaan, jossa asia hoituu automaattisesti. Kuten Linux-järjestelmissä yleisesti, CentOs 5.5 -käyttöjärjestelmässä tällainen paikka on */etc/profiles.d*. Tässä hakemistossa olevat komentosarjat suoritetaan automaattisesti, joten Jenkinsin työhakemiston ympäristömuuttujan asetuskomennon kirjoitettiin sinne.

Seuraava asia, mitä Jenkins käynnistyessään tekee, on http-portin kuunteleminen. Oletuksena portti on 8080. Sen voi myös muuttaa ympäristömuuttujan avulla, jos kyseinen portti on esimerkiksi varattu muuhun käyttöön. Tässä projektissa portin vaihto oli lopul-

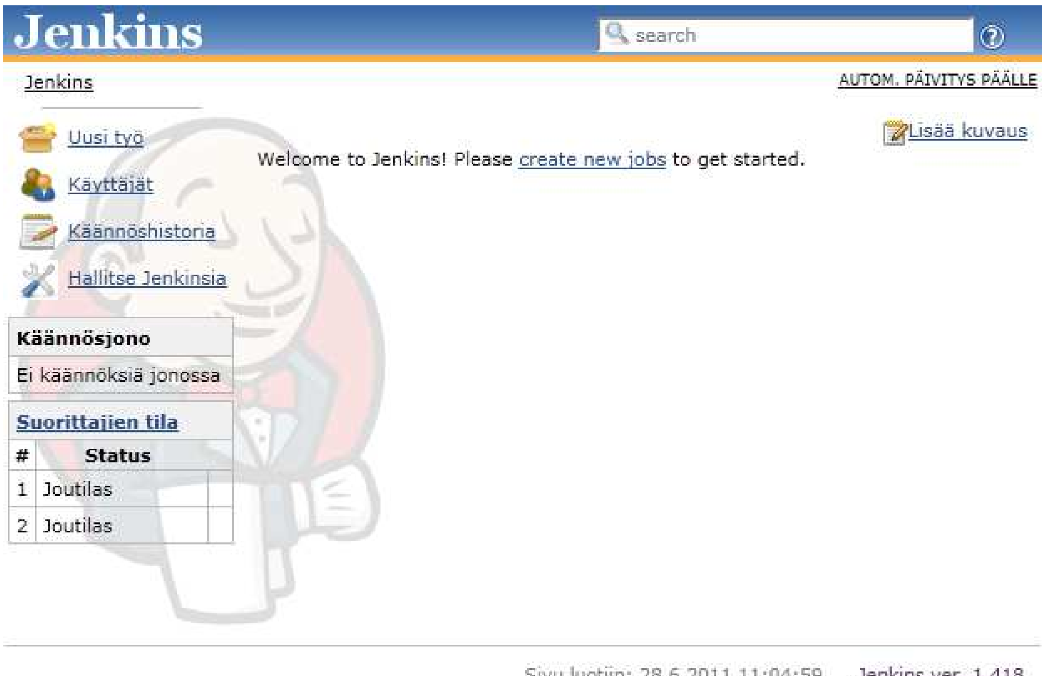
ta tehtävä, koska portti valjastettiin toisen ohjelman käyttöön. Portin vaihto onnistuu käynnistämällä Jenkins muuten normaalisti, mutta asettamalla sille parametrina haluttu portti:

```
java -jar jenkins.war --httpPort=9090
```

Esimerkissä "java –jar" on käsky Jenkinsin war-muotoisen tiedoston käynnistämiseksi. Valitsin "--httpPort=9090" asettaa Jenkinsin käyttämään porttia 9090.

Kun Jenkins on käynnistetty ja portti asetettu, Jenkinsin web-käyttöliittymän saa avattua internetselaimeen kirjoittamalla palvelimen http-osoiteen ja sen perään käytettävän portin.

Kun Jenkins on saatettu toimintakuntoon, sen web-käyttöliittymästä voi tehdä suurin piirtein kaiken tarvittavan. Jenkinsin peruskäyttö, eli projektien luominen, käännöstöiden hallinta ja tulosten seuranta tapahtuu nimenomaan täällä. Web-käyttöliittymästä onnistuu myös hallintaan liittyvät toiminnot, kuten esimerkiksi käyttäjätilien hallinta, liitännäisten hallinta, lokitietojen tarkkailu ja päivitystoimenpiteet. Kuva 2 on otos Jenkinsin web-käyttöliittymästä, kun se on ensimmäistä kertaa käynnistetty palvelimelle oletusasetuksin.



Kuva 2. Jenkinsin web-käyttöliittymä

Kuvan vasemmassa laidassa ylimmäisenä on Jenkinsin keskeisimmät hallintaan liittyvät työvälineet. Näiden alapuolelle listautuu jonossa olevia käännöstöitä. Jenkinsiin pääsee tässä vaiheessa kuka tahansa, joten turvallisinta on aloittaa käyttäminen asettamalla sinne käyttäjä ja tälle salasana. Tämä onnistuu kuvassa 2 näkyvän ”Hallitse Jenkinsiä” -painikkeen alta löytyvästä hallintapaneelista, joka puolestaan on esitetty kuvassa 3.

Security Realm

- ○ Delegate to servlet container
- ◉ Jenkins's own user database
  - ☐ Salli käyttäjien rekisteriötyminen
- ○ LDAP
- ○ Unix user/group database

Authorization

- ○ Anyone can do anything
- ○ Legacy mode
- ○ Logged-in users can do anything
- ◉ Matrix-based security

User/group	Overall		Slave		Job						Run		View			SCM
	Administer	Read	Configure	Delete	Create	Delete	Configure	Read	Build	Workspace	Delete	Update	Create	Delete	Configure	Tag
ci_admin	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☐
Anonymous	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Kuva 3. Turvallisuusasetukset, hallintapaneeli

Kuva 3 on hallintapaneelista ja siihen on rajattu näkymään Jenkinsin tietoturva-asetukset. Autentikoituminen on tässä tapauksessa toteutettu käyttäen Jenkinsin omaa käyttäjätietokantaa.

Jenkinsin käyttäminen aloitetaan luomalla uusi työ, johon yleensä liitetään yksi kokonainen ohjelmistoprojekti. Jenkinsiin voi luoda useampia projekteja ja määrittää kullekin omat erilliset asetukset. Näin Jenkinsissä voi hallita samanaikaisesti useampia eri vaatimuksia omaavia ohjelmistoprojekteja. Kuvassa 4 on näkyvillä Jenkinsiin lisättyjä projekteja.

Kuva 4. Jenkins-projektit

Kuva 4 on Jenkinsin etusivulta ja siinä näkyy luodut projektit sekä niiden tilanteet. Sininen pallo kertoo, että käännös on onnistunut, ja punainen, että se on epäonnistunut. Keltainen pallo osoittaa, että käännös on onnistunut, mutta se sisältää niin paljon virheitä, että tulos on epävakaa. Pallojen vieressä oleva säätilaa kuvaava ilmaisin kertoo enemmänkin koodin laadusta, johon vaikuttavat testausprosessissa mukana olevat analysointi- ja testaustyökalut. Tässä näytettävät kuvakkeet ovat aurinkoinen, puolipilvinen, sateinen ja ukkosmyrsky.

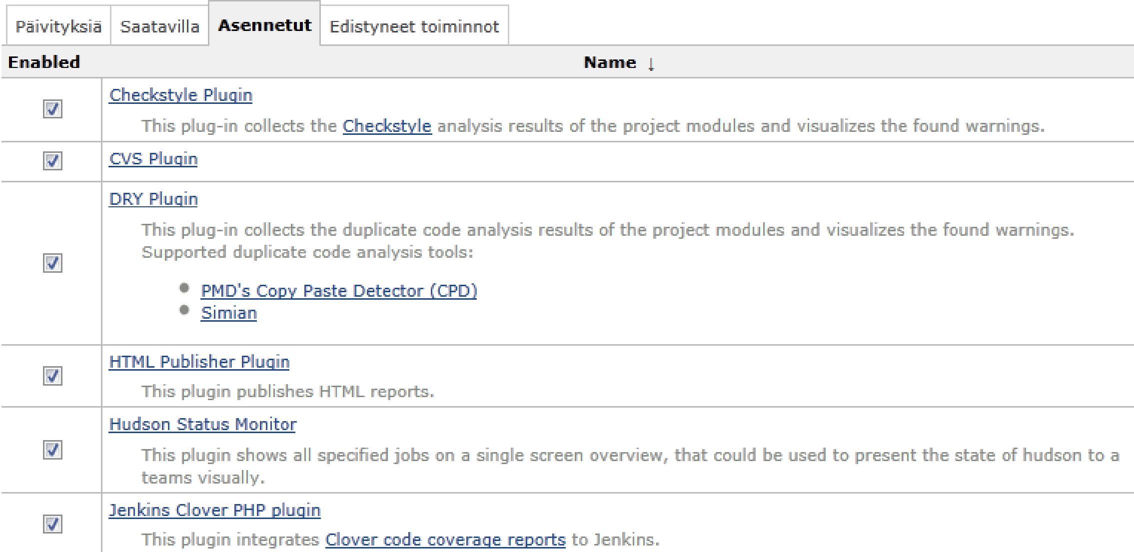
Jenkinsin päivittäminen onnistuu helpoiten suoraan web-käyttöliittymästä. Jos Jenkinsistä on saatavilla uudempi versio, käyttöliittymän ilmoittaa tästä käyttäjälle hallintapaneelissa (kuva 5).

Kuva 5. Jenkinsin päivittäminen

Käyttäjän on helppo päivittää ohjelmisto painamalla kuvassa näkyvää päivityspainiketta. Päivitys ei sotke Jenkinsiin tehtyjä asetuksia, mutta vaatii uudelleenkäynnistyksen.

Koska Jenkins pitää datan erillisessä työhakemistossa, sen voi päivittää turvallisesti myös komentoriviltä käsin yksinkertaisesti korvaamalla vanhan war-pakkauksen uudemmalla. Ennen päivitystä Jenkins on suljettava ristiriitatilanteiden ehkäisemiseksi.

Jenkinsin laajennettavuus ja monikäyttöisyys perustuvat erinäisten osien liitettävyyteen ja Jenkinsin omaan laajaan liitännäisvalikoimaan. Tämä mahdollistaa Jenkinsin käyttämisen hyvin erilaisissa ympäristöissä. Esimerkiksi kun ohjelmointikieli on jokin vähän epätavanomaisempi, liitännäisvalikoimasta saattaa löytyä apuväline, jolla sille voi toteuttaa Jenkinsin päällä toimivan jatkuvan integraation. Myös tässä työssä, kun kohdesovelluksen ohjelmointikieli on PHP, liitännäisvalikoima mahdollisti Jenkinsin käyttämisen jatkuvan integraation ytimenä. Liitännäisiä voi asentaa ja poistaa joko komentoriviltä tai web-käyttöliittymästä. Jälkimmäinen vaihtoehto on helpompi, koska käyttöliittymään on rakennettu kattava hallintapaneeli lisäosia varten.



Kuva 6. Liitännäisten hallintapaneeli

Kuva 6 on Jenkinsin liitännäisten hallintasivulta, joka löytyy Jenkinsin hallintapaneelin alta. Kustakin lisäosasta löytyy lyhyt kuvaus ja versionumero. Liitännäisiä otetaan käyttöön aktivoimalla niiden vasemmalla puolella näkyviä valintaruutuja. Tarkempia tietoja ja käyttäjäkokemuksia varten jokainen liitännäinen sisältää linkin itseään koskevalle internetsivulle.

Ohjelmistot käännetään kussakin projektissa erikseen. Oletusarvoisesti tämä vaatii käyttöliittymän avaamisen ja napinpainalluksen, mutta toimenpiteen voi myös automatisoida. Tähän on olemassa erilaisia keinoja:

- Käännöstyön voi asettaa alkamaan, kun kaikki muut projektit on käännetty.
- Käännöksen voi aloittaa käyttämällä liipaisimena jotakin ulkoista tapahtumaa, kuten esimerkiksi palvelimella sijaitsevan sovelluksen suorittamista.
- Käännösprosessin voi ajastaa suoritettavaksi tietyin väliajoin, kuten esimerkiksi päivittäin tai viikoittain.
- Jenkinsin voi määrätä tutkimaan tietovarastoa säännöllisin väliajoin, jolloin käännös aloitetaan, kun tietovarastossa on tapahtunut jokin muutos.

Tässä työssä noudatetaan viimeisintä vaihtoehtoa, eli Jenkins on asetettu tutkimaan tietovarastoa säännöllisin väliajoin. Tämä vaihtoehto on järkevin valinta silloin, kun halutaan pitää yllä mahdollisimman tuoretta versiota kohdeohjelmistosta, eikä koontiprosessi riipu mitenkään muista projekteista. Tietovaraston tarkkailujen välinen aika on

tässä projektissa asetettu yhdeksi minuutiksi. Jos tietovarastoon on tänä aikana tullut muutos, Jenkins noteeraa tämän ja aloittaa määrättyjen toimenpiteiden suorittamisen. Kuva 7 selventää käännöksen automatisointia.

Build Triggers
Build after other projects are built
Trigger builds remotely (e.g., from scripts)
Build periodically
Poll SCM
Schedule * * * * *
Build
Invoke Ant
Ant Version Default
Targets
Add build step

Kuva 7. Käännösten hallinta

Kuvan 7 esimerkissä käännösprosessin automatisointi on toteutettu asettamalla Jenkins tutkimaan säännöllisin väliajoin tietovarastoa. Kuvan *Schedule*-kohdan syötekentässä määritellään halutunlainen viive tietovaraston tarkkailujen välille. Tässä hyödynnetään Unix-pohjaista, cron-tyyppistä ajastuspalvelua. Kuvassa näkyvä syöte "viisi tähteä" määrää Jenkinsin tutkimaan tietovarastoa minuutin välein. Käännösprosessiin voi lisätä vaikka loputtomiin erilaisia kohteita, kuten kuvassa näkyvä *Invoke Ant* -kohta. Jenkins suorittaa kaikki lisätyt kohteet järjestyksessä. Tässä työssä ainoaksi kohteeksi on asetettu Apachen Ant [10], koska se on hallinnoiva osa, joka hoitaa kaikki tämän projektin käännöstyössä vaadittavat toimenpiteet. Antista kerrotaan tarkemmin seuraavassa luvussa.

3.3 Koontityökalu

Kun jatkuvan integraation työväline, tässä tapauksessa Jenkins, hakee ohjelmakoodin tietovarastosta, seuraava askel on koodille tehtävä jatkokäsittely. Tämä tarkoittaa kääntämistä, paketointia, testausta sekä muita tarvittavia toimenpiteitä. Tätä varten

löytyy erillisiä sovelluksia, joita kutsutaan koonti- ja käännöstyökaluiksi. Useimmiten sama työkalu suorittaa siis ohjelmakoodin kääntämisen sekä koostamisen. Tässä työssä käsiteltävä kohdeaineisto on PHP-koodia, joka ei vaadi kääntämistä, kuten esimerkiksi Java. Näin koodin jatkokäsittelystä on oikeampaa käyttää termiä koostaminen ja sen tekevää ohjelmistoa nimittää koontityökaluksi.

Työn suunnitteluvaiheessa oltiin aluksi kallistumassa erityisesti PHP:tä varten suunniteltuun Phingiin [9], joka pohjautuu Apachen kehittämään Ant-työkaluun [10]. Phing olisi ollut varsin tarkoituksenmukainen väline PHP-sovelluskehitykseen painottuvassa ympäristössä. Se ei esimerkiksi vaadi Javan asentamista. Suunnitteluvaiheessa selvisi kuitenkin, että lähitulevaisuudessa samassa ympäristössä tullaan kehittämään myös Java-pohjaista Android-sovellusta. Phingin soveltuvuus on tätä varten vähintäänkin epävarmaa. Lisäksi, koska työssä on päämääränä saavuttaa mahdollisimman yhdenmukainen lopputulos, on viisaampaa hoitaa koostamiseen vaaditut toimenpiteet yhden työkalun voimin.

Vaikka Phingin isä, Ant on Javaa varten kehitetty sovellus, sen xml-käännöstiedosto on varsin monikäyttöinen. Siitä tuntui myös löytyvän runsaasti hyviä kokemuksia PHP-kehityksessä. Antia on lisäksi käytetty luvussa 3.4 mainitussa Jenkinsin PHP-projektissa *Template for Jenkins Jobs for PHP Projects* [8], josta tässä projektissa on otettu mallia. Tässä työssä koontityökalun roolissa toimii siis Apache Ant.

Antin toiminta määräytyy sen xml-asetustiedoston mukaan. Sinne kirjoitetaan komennot, joita Antin halutaan suorittavan. Ant on erityisesti Javaa varten kehitetty käännös- ja koontityökalu. Sen yleisimmät tehtävät ovat kääntäminen, koostaminen ja riippuvuuksien hallitseminen. Tässä työssä Antin tärkein tehtävä on testausprosessiin liittyvien riippuvuuksien hallinta. Antin sisältämä *exec*-suorituskomento mahdollistaa sen käyttämisen tehokkaasti PHP-testausjärjestelmän tukipilarina. Exec-suorituskomennon avulla on mahdollista ajaa kaikki luvussa 3.4 mainitut PHP-testaustyökalut oikeassa järjestyksessä ja ohjata niiden tulokset haluttuun paikkaan. Seuraavaksi esimerkki *exec*-toiminnon käytöstä.

```
<target name="phpunit" description="Run unit tests using PHPUnit
and generates junit.xml and clover.xml">
```

```
  <exec executable="phpunit" failonerror="true"/>
</target>

<target name="phpmd" description="Generate pmd.xml using PHPMD">
  <exec executable="phpmd">
    <arg path="${source}" />
    <arg value="xml" />
    <arg value="${basedir}/build/phpmd.xml" />
    <arg value="--reportfile" />
    <arg value="${basedir}/build/logs/pmd.xml" />
  </exec>
</target>
```

Esimerkissä PHPMD-testaustyökalun [14] suorittava komento lukee sille määrätyt säännöt tiedostosta ”phpmd.xml” ja tuottaa siitä tuloksiin perustuvan xml-muotoisen raportin Jenkinsiä varten. PHPUnitin [11] tapauksessa raportin tuottamista ei määritellä Antissa, vaan se on määritelty PHPUnitin asetustiedostoon, josta kerrotaan lisää luvussa 4.1.

Toinen tärkeä Antin ominaisuus tässä projektissa on tiedostojen kopiointi. Kun testausautomaation prosessi on hahmoteltu, on selvitettävä jokin keino siirtää testauksesta läpimennyt ohjelma eteenpäin. Yksi vaihtoehto olisi hoitaa asia Jenkinsin laajasta lisäosavarastosta löytyvillä liitännäisillä. Asian voisi hoitaa esimerkiksi SVN-lisäosalla, jolla on mahdollista lähettää käsiteltävä aineisto uuteen tietovarastoon. Parempi vaihtoehto tämän projektin kohdalla on kuitenkin pitää tiedostoihin ja hakemistoihin liittyvät asetukset kohdistettuna yhteen paikkaan, jolloin niitä on helpompi hallita. Ant on tässä kätevä valinta. Siitä löytyy suoraan kopiointikomento *copy*, joka kirjoitetaan halutun kohteen sisälle Antin xml-asetustiedostoon. Ant on lisäksi sisällytetty tässä projektissa jo valmiiksi tietovarastoon, joten käsitellyn aineiston siirtoon liittyvät komennot ovat helposti päivitettävissä.
Antin ottamisessa osaksi tätä työtä oli kuitenkin eräs ongelma. Se vaatii toimiakseen Javan. Osoittautui myös, että CentOs 5.5 asennuspalvelin ei sisältänyt muuta kuin OpenJDK:n. CentOs ei tarjoa tukea Sun Javalle, eikä Sunin sivuilta ole ladattavissa toimivaa versiota kyseiselle käyttöjärjestelmälle. CentOs:n tietovarastosta on löydettävissä ainoastaan OpenJDK, jonka yhteistoiminta Antin kanssa vaatii tältä vähintään versi-

on 1.7.1. Aluksi ongelmaa ratkottiin päivittämällä Ant sekä OpenJDK. Näin Ant saatiin pyöräytettyä kyllä käyntiin, mutta sen sisältämät suorituskomennot eivät toimineet. Lopulta oli tehtävä manuaalinen asennus Sunin Javalle sekä Apachen Antille. Tämä tarkoittaa tiedostojen lataamista palvelimelle, sekä ympäristömuuttujien luomista kohdehakemistoihin. Jottei ympäristömuuttujia tarvitsisi aina uudestaan manuaalisesti asetella, se kannattaa hoitaa kirjoittamalla automaattinen komentosarja niiden asettamista varten.

Kuten luvussa 3.2 on esitelty, Jenkinsin työhakemiston ympäristömuuttujan asettamista varten on jo valmiiksi kirjoitettu komentosarja hakemistoon */etc/profile.d*. Antin ja Javan vaatimien ympäristömuuttujien asetuskomentosarjat kirjoitettiin siis samaan paikkaan. Seuraavassa ovat esiteltynä kyseiset komentosarjat.

```
JAVA_HOME=/usr/lib/jdk1.6.0_26/
export JAVA_HOME
PATH=$PATH:$JAVA_HOME/bin

ANT_HOME=/usr/local/ant
export ANT_HOME
PATH=$PATH:$ANT_HOME/bin
```

Yllä olevat komennot ovat Javan ja Antin ympäristömuuttujien asettamista varten. Kumpikin asetetaan samalla tyylillä. Ensin annetaan muuttujalle polku ja tämän jälkeen muuttujat lisätään arvoineen ympäristömuuttujiksi käyttöjärjestelmään.

3.4 Jatkuva integraatio ja PHP

Kun ohjelmointikielenä on web-pohjainen PHP, jatkuvan integraation toimenpiteet ovat osaltaan hieman poikkeavat verrattuna perinteisempiin kieliin, kuten Javaan tai C++:aan. Kuten esimerkiksi luvussa todettiin, ”käännöstyökalun” pääpaino on aivan toisaalla kuin ohjelmakoodin kääntämisessä. Eräs toinen eroavaisuus löytyy testauspuolelta. Web-sovellusten testaaminen on parasta tehdä ympäristössä, jossa se tulee toimimaan, eli suoraan palvelimella. Ympäristö saattaa olla rakenteeltaan hyvin monimutkainen ja sen simuloiminen paikallisesti ohjelmoijien työasemilla saattaa olla jopa

mahdottomuus. Jatkuvan integraation pääpaino siis siirtyy web-sovelluksissa kääntämisen sijaan kokonaisuuden koostamisen ja integroimisen puolelle.

Luvussa 3.2 esitelty jatkuvan integraation työväline Jenkins on kehitetty Javalle ja sisältää suurimmaksi osaksi työkaluja suoraan sitä varten. Jenkinsille on kuitenkin myös kehitetty paljon lisäosia PHP:tä silmälläpitäen. Jenkinsin internetsivuilla on esitelty kokonainen projekti, jonka tarkoituksena on standardoida PHP:n jatkuvaa integraatiota osana Jenkinsiä. Kyseinen hanke käyttää nimeä *Template for Jenkins Jobs for PHP Projects* [8]. Tässä projektissa on käytetty pohjana kyseistä standardointihanketta ja käytetty siinä mainittuja PHP:n testaus- ja analysointityökaluja:

- PHP_Depend tutkii sovelluksen laatua ja monimutkaisuutta.
- PHPMD etsii ohjelmistosta virheitä ja epäoptimaalista koodia.
- PHPUnit toimii yksikkötestauskehyksenä.
- PHP_CodeSniffer puuttuu epäsiistiin, vaikeasti luettavaan sekä hyvien tapojen vastaisesti kirjoitettuun koodiin.
- PHPDocumentor tuottaa pohjan ohjelmiston dokumentaatiolle.

Työkalut koskevat pääosin ohjelmakoodin analysointia ja testaamista. Niihin palataan tarkemmin testausta käsittelevässä luvussa 4. Jotta PHP:n testaus- ja analysointityökalujen tulokset voidaan käsitellä, Jenkinsiin on asennettava niitä vastaavat liitännäiset. Tässä työssä Jenkinsiin asennettiin lisäosat:

- Checkstyle esittää PHP_CodeSniffer-ohjelmiston tulokset.
- Clover PHP käsittelee PHPUnitin testituloksia.
- DRY ilmaisee ohjelmakoodista paljastuneet duplikaatit.
- HTML Publisher-liitännäisen avulla tulostetaan raportteja web-käyttöliittymässä.
- JDepend käsittelee PHP_Dependin lokitiedostot.
- Plot generoi kaavioita analysointitulosten perusteella.
- PMD käsittelee PHPMD:n tulokset.
- Violations on testaustuloksia kokoava lisäosa.
- xUnit kytkee PHPUnitin Jenkinsiin.

Asentaminen onnistuu helpoiten suoraan web-käyttöliittymästä, mutta on mahdollista myös komentoriviltä.

3.5 Versionhallinta jatkuvassa integraatiossa

Cabforce Oy:n versiohallintaohjelmistona toimii Subversion, eli SVN. Jatkuvassa integraatiossa versionhallinta on ensimmäinen osa, joka suoraan näkyy kehittäjille ja sen toimivuus on edellytys jatkuvalle integraatiolle. Versionhallinta toimii omana erillisenä osanaan huolehtien ohjelmiston versioinnista eikä sille ei tarvitse tehdä mitään asennukseen liittyviä toimenpiteitä. Jatkuvan integraation palvelin puolestaan on vastuussa versionhallintaan tehdyistä muutoksista. Ohjelmoijien lisätessä tuotoksiaan tietovarastoon versionhallinnan tärkein tehtävä jatkuvan integraation näkökulmasta on ylläpitää viimeisintä versiota ohjelmasta. Toisaalta jatkuvan integraation yksi tärkeimmistä päämääristä onkin, että versionhallinnasta löytyy jatkuvasti toimiva kokonaisuus. Tämä tarkoittaa sitä, että jos versionhallintaan päätyy virheellistä koodia, jatkuva integraatio informoi asiasta kehittäjiä (kuva 1). Näin tietovarastoon joutuneet virheet voidaan korjata nopeasti.

Kun jatkuvassa integraatiossa käytettävä testausjärjestelmä huolehtii virheettömän koodin pitämistä tietovarastossa, kannattaa huomioida, että myös testit kannattaa pitää tietovarastossa. Silloin myös viimeisimmät ja kehittyneimmät testit löytyvät sieltä.

4 PHP-sovelluksen analysointi ja testaus

Testaaminen on aina tärkeä osa ohjelmistokehitystä. Luonnollisesti se on sitäkin tärkeämpää, mitä monimutkaisempi kokonaisuus on kyseessä. Kaikki ohjelmoijat tekevät virheitä ja ne on yksinkertaisesti korjattava ennemmin tai myöhemmin. Mitä aikaisemmin virheisiin puututaan, sitä kustannustehokkaampaa ohjelmistoalan liiketoiminnasta tulee. Virheiden huomaaminen ajoissa vähentää myös virheiden kertymistä toistensa päälle. Tämä on tärkeätä, koska myöhemmin korjatessa jotakin erityisen kriittistä virhettä saatetaan huomata, että siihen johtavat syyt ovat jo syvällä järjestelmässä. Näin korjaamisesta tulee erittäin kallis ja aikaa vievä prosessi. Kriittisimmät virheet ovat tietenkin etusijalla koko ohjelmiston toimivuuden kannalta, mutta pienempiäkään virheitä kannata ohittaa olankohautuksella. Niihin puuttumisella nimittäin saattaa olla suuri merkitys ohjelmiston tuottavuuden kannalta.

Varsinaisten virheiden lisäksi ohjelmakoodiin päätyy usein myös epäjohdonmukaista ja turhaan resursseja vievää sisältöä. Kun testauksessa pureudutaan tähän ongelmaan,

saadaan johdonmukaisempi ja kustannustehokkaampi tulos. Kun kyse on jatkuvasta integraatiosta, testausta pyritään automatisoimaan. Näin ollen testauksesta itsestään ei tule resurssikysymys. Jatkuva integraatio mahdollistaa lisäksi testitulosten pohjalta tehtävän välittömän palautteen suoraan ohjelmoijille. Tämä tehostaa ohjelmoijien kehittymistä ja kykyä tuottaa virheettömämpää ja johdonmukaisempaa koodia vastaisuudessa.

Alun perin suunniteltiin, että tässä projektissa käytetään vain yhtä laajempaa testaussovellusta. Ohjelma nimeltä PHPUnit oli melkoisen varma valinta perustuen työn toteuttajan aikaisempaan kokemukseen siitä. Kyseinen ohjelma on tuntunut suhteellisen näppärältä ja tehokkaalta työkalulta. Kun tutkittiin aiheeseen liittyviä keskusteluita ja blogeja eri internetsivuilta, PHPUnit vakuutti soveltuvuutensa myös jatkuvan integraation osaksi. Myös Hudsonin ja Jenkinsin käyttäjät usein tuntuvat päätyvän sen käyttöönottoon. Varsinkin luvussa 3.4 esitelty Jenkinsin malli jatkuvasta integraatiosta PHP:lle varmisti PHPUnitin käyttämisen osana tätä työtä.

4.1 PHPUnit

PHPUnit on ohjelmistojen yksikkötestaukseen tarkoitettu testauskehys. Se helpottaa yksikkötestien kirjoittamista ja suorittamista sekä sillä voi myös analysoida testituloksia. PHPUnit kuuluu xUnit-perheeseen, joka on nimitys kokoelmalle yksikkötestaukseen tarkoitetuista työkaluista. Sen tarkoituksena on pitää yllä runkoa, joka levittää yhdenmukaista arkkitehtuuria useammalle eri ohjelmointikielille ohjelmistotestauksessa. Tähän joukkoon kuuluu esimerkiksi hyvin tunnettu Javan testaukseen tarkoitettu JUnit, joka itse asiassa joukon ensimmäisenä ohjelmistona on johtanut xUnit-nimityksen. Useimmat xUnit-sarjan ohjelmistojen lähdekoodeista ovat avoimia ja vapaasti saatavilla, kuten myös PHPUnitin.

Jenkins käyttää PHPUnitia Antin avulla luvussa 3.2 kuvatulla tavalla. Antiin on siis kirjoitettu pelkästään PHPUnitin suorituskomento. PHPUnitin toimintaan vaikuttavat muut asetukset on tässä projektissa kirjoitettu erilliseen asetustiedostoon nimeltä *phpunit.xml.dist*. Sinne määritellään kaikki kohteet, joista PHPUnitin halutaan etsivän testattavaa koodia. Myös testaustulosten julkaisu määritellään sinne. Kun *phpunit.xml.dist*-tiedosto löytyy työhakemistosta, PHPUnit käyttää sitä asetustiedostonaan. PHPUnitin asetustiedosto sisällytettiin versionhallintaan, jotta sitä olisi helppo päivittää. Tässä

työssä *phpunit.xml.dist*-tiedosto generoi PHPUnit testien pohjalta lokitiedostot *junit.xml* ja *clover.xml*. Näistä *junit.xml* pitää sisällään tuloksen varsinaisista yksikkötesteistä ja *clover.xml* sisältää kokonaisvaltaisempaa analyysia koodista. Lisäksi *phpunit.xml*.dist tuottaa html-muotoisen raportin, joka näytetään Jenkinsin web-käyttöliittymässä. Jenkins tarvitsee xUnit-lisäosan käsitelläkseen PHPUnitin tuloksia. PHPUnitin tuloksia käsitellään Jenkinsin lisäosilla Clover PHP ja HTML-Publisher.

PHPUnitia käyttämällä voi järjestää testitapauksia sarjoihin. Tämä helpottaa testien pitämistä järjestyksessä ja helpottaa niiden muokkaamista. Erityisesti hyötyä saadaan, kun halutaan ajaa vaihdellen erityyppisiä testejä. Testien jakaminen sarjoihin tapahtuu *phpunit.xml.dist*-tiedostossa. Se tehdään *testsuites*-osiossa siten, että sinne lisätään tunniste kutakin testisarjaa varten.

```
<testsuites>
  <testsuite name=””>
    <directory suffix=”Test.php”>phpunit_testfiles</directory>
  </testsuite>
  <testsuite name=””>
    <directory suffix=”Test.php”>
      phpunit_testfiles/instance_tests
    </directory>
  </testsuite>
</testsuites>
```

Tässä työssä suoritettavat testit on jaoteltu kansioittain ja testit otetaan käyttöön etsimällä kaikki ”Test.php”-päätteiset testitiedostot kustakin kohdehakemistosta. Kukin testitiedosto sisältää PHP-luokan, johon kirjoitetaan funktioita yksikkötestejä varten. Funktioiden nimien täytyy alkaa sanalla *test*, jotta PHPUnit osaa käyttää niitä. Jotta PHPUnitin toiminnot saadaan käyttöön luokassa, siihen on kirjoitettava extends-määritys PHPUnitia varten. PHPUnit sisältää joukon vertailufunktioita (*assertions)*, jotka helpottavat testien kirjoittamista. Seuraava koodinpätkä on yksinkertainen esimerkki PHPUnitin testausluokasta.

```
class InstanceTest extends PHPUnit_Framework_TestCase {
  public function testFunktio() {
```

```
    $this->assertEquals(“odotettu tulos”,“saatu tulos”);
  }
}
```

Esimerkissä otetaan PHPUnit-kehys käyttöön extends-määrityksellä. Tämä saa aikaan sen, että luokan sisältä löytyvät oikein nimetyt funktiot suoritetaan automaattisesti. Esimerkissä näkyvä *assertEquals*-funktio vertailee kahta muuttujaa ja palauttaa virheen, jos ne eivät vastaa toisiaan.

PHPUnit kytkeytyy Jenkinsiin xUnit-lisäosan avulla. Jenkinsissä xUnit on lisäosa, joka muuttaa muiden xUnit-perheeseen kuuluvien testausvälineiden, kuten tässä tapauksessa PHPUnitin, testaustulokset JUnit-muotoisiksi. Näin Java-pohjaisella Jenkinsillä voi julkaista myös muiden ohjelmointikielien yksikkötestien tuloksia. Jenkinsin xUnit-liitännäinen luo raportin näytettäväksi web-käyttöliittymässä. Kuvassa 8. on esitelty PHPUnitin tulokset Jenkinsin web-käyttöliittymässä.

Test Result : (root)

0 epäonnistunutta (±0)

23 testiä (±0)
Vei 52 ms
Lisää kuvaus

Kaikki testit

Class	Kesto	Epäonnistui	(muutos)	Ohitettu	(muutos)	Yhteensä	(muutos)
ClientAPITest	1 ms	0		0		2	
InstanceTest	32 ms	0		0		1	
MyClassTest	1 ms	0		0		1	
TestAPITest	16 ms	0		0		19	

Kuva 8. PHPUnit

Kuvassa näkyy suoritetut testit ja niiden kestoajat. Jos yksikin PHPUnitiin kirjoitettu testi epäonnistuu, Jenkins ei suorita PHP-ohjelman tulkintaa loppuun, vaan antaa virheilmoituksen.

4.2 Muut testaus- ja analysointityökalut

Varsinainen yksikkötestaus jää tässä työssä PHPUnitin harteille. Mutta kuten luvussa 3.4 on mainittu, Jenkinsin PHP-projekti tuo mukanaan joukon muitakin käyttökelpoisia välineitä, joilla pureutua ohjelmiston heikkouksiin.

PHP_Depend on staattista analyysiä tekevä sovellus. Se tutkii ohjelmakoodia ja mittaa sieltä tunnuslukuja, jotka kuvaavat sovelluksen laatua. Esimerkiksi ohjelman monimutkaisuuden arvoon vaikuttaa koodista löytyvien loogisten operaattoreiden määrä. PHP_Depend tuottaa lokitiedoston Jenkinsistä löytyvälle JDependille, joka puolestaan mahdollistaa PHP_Dependin tulosteiden näyttämisen Jenkinsissä.

PHPMD (PHP Mess Detector) [14] on suositusta Javalle tarkoitetusta PMD-ohjelmasta PHP:tä varten tehty vastine. Se on edellä mainitun PHP_Dependin [13] sivussa kehitetty ohjelmisto, joka etsii koodista suoranaisia virheitä. Se pyrkii löytämään myös epäoptimaalista koodia ja turhan monimutkaiseksi kirjoitettuja lausekkeita. Lisäksi se etsii käyttämättömiä metodeja ja parametreja. PHPMD:n toiminta perustuu sille asetettaviin sääntöihin [15], jotka kirjoitetaan Jenkinsin työhakemistosta löytyvän *build*-kansion juureen. Sääntötiedoston nimen voi itse päättää, kunhan se esitellään oikein Antin xml-tiedostossa. Tässä projektissa sen nimi on phpmd.xml.

```
<ruleset name="Cabforce PHPMD ruleset">
  <description> Cabforce PHPMD ruleset </description>

  <rule ref="rulesets/unusedcode.xml"/>
  <rule ref="rulesets/codesize.xml/CyclomaticComplexity" />
  <rule ref="rulesets/naming.xml">
    <exclude name="ShortVariable"/>
    <exclude name="LongVariable"/>
  </rule>
</ruleset>
```

Edellinen esimerkki PHPMD-sääntötiedostosta. Kaikki mukaan halutut säännöt kirjoitetaan *ruleset*-tagin sisälle omiksi *rule*-osioikseen. PHPMD:ssä käytettävät säännöt jaetaan neljään yläluokkaan, joiden alle ne sijoittuvat tarkemmin.

Kokoon ja monimutkaisuuteen puuttuvat säännöt (Code Size Rules):

- *CyclomaticComplexity* eli syklomaattinen kompleksisuus mittaa koodin monimutkaisuutta. Tulos määräytyy *if-, while-, for- ja case*-rakenteiden summasta.
- *NPath complexity* laskee mahdollisia suoritettavia polkuja funktioissa. Se on siis arvo jatkuvien sisäkkäisten toimintojen määrälle funktioissa.
- *ExcessiveMethodLength* analysoi metodien pituuksia. Jos tämä virhe esiintyy metodissa, sitä täytyisi jakaa pienempiin osiin.
- *ExcessiveClassLength* tarkkailee koodirivien määrää luokissa. Jos arvo on liian suuri, hallinta saattaa olla vaikeata.
- *ExcessiveParameterList* ilmoittaa, jos funktiossa on liikaa parametreja.
- *ExcessivePublicCount* laskee public-tyyppisten metodien määrää luokissa.
- *TooManyFields* laskee kenttien määrää luokissa.
- *TooManyMethods* laskee metodien määrää luokissa.
- *ExcessiveClassComplexity* esittää luokan monimutkaisuutta. Mitä monimutkaisempi rakenne luokalla on, sitä vaivalloisempaa sen muokkaaminen on.

Suunnittelusäännöt (Design Rules):

- *ExitExpression* antaa virheen, jos koodin joukossa käytetään *exit*-funktiota.
- *EvalExpression* ilmoittaa, jos koodin seasta löytyy *eval*-funktioita.
- *GotoStatement* etsii *goto*-funktioita koodista ja ilmoittaa, jos löytyy.
- *NumberOfChildren* laskee lapsiluokkien määrän ja ilmoittaa, jos niitä on liikaa.
- *DepthOfInheritance* ilmoittaa, jos luokka perii liian montaa ylempää luokkaa.
- *CouplingBetweenObjects* ilmoittaa liiallisista olioiden välisistä riippuvuuksista.

Nimeämis-säännöt (Naming Rules):

- *ShortVariable* ilmoittaa, jos muuttujalla on liian lyhyt nimi.
- *LongVariable* ilmoittaa, jos muuttujalla on liian pitkä nimi.
- *ShortMethodName* ilmoittaa, jos metodilla on liian lyhyt nimi.
- ConstructorWithNameAsEnclosingClass varoittaa, jos konstruktori on saman niminen kuin luokka.
- *ConstantNamingConventions* ilmoittaa, jos luokan nimi alkaa pienellä tai jos muuttujan nimi alkaa isolla.
- *BooleanGetMethodName* ilmoittaa, jos koodista löytyy väärin nimettyjä boolean-arvon palauttavia metodeita.

Käyttämättömyys säännöt (Unused Code Rules):

- *UnusedPrivateField* ilmoittaa käyttämättömistä private-kentistä.
- *UnusedLocalVariable* ilmoittaa käyttämättömistä muuttujista.
- *UnusedPrivateMethod* ilmoittaa käyttämättömistä private-metodeista.
- *UnusetFormalParameter* ilmoittaa käyttämättömistä parametreista.

PHPMD:n tuottaman lokitiedoston tulokset käsitellään Jenkinsissä PMD-liitännäisen avulla. PMD tulkitsee PHPMD:n tuottaman lokitiedoston ja tulostaa siitä löytyvät virheet ja varoitukset Jenkinsin web-käyttöliittymään (kuva 9).

PMD Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
3	3	0

Summary

Total	High Priority	Normal Priority	Low Priority
3	1	2	0

Details

Categories | **Types** | Warnings | Details | New | High | Normal

Type	Total	Distribution
CyclomaticComplexity	2	
ShortMethodName	1	
Total	**3**	

Kuva 9. PMD

Kuva 9 on esimerkki PMD-raportista Jenkinsin web-käyttöliittymässä. Se listaa PHPMD:n xml-asetustiedostoon kirjoitetut säännöt ja näyttää, mitä sääntöjä on rikottu. Se myös pitää kirjaa edellisistä tuloksista, joten se osaa myös näyttää, mitkä rikkeet ovat uusia ja mitkä on korjattu. Tässä tapauksessa PHPMD:n xml-tiedostoon on kirjoitettu säännöt *CyclomatiComplexity* ja *ShortMethodName*. Kuvassa keltaisella näkyvät linjat ovat kevyempiä rikkeitä eli varoituksia. Punaisella merkityt rikkeet on priorisoitu kriittisemmiksi virheiksi.

PHP_CodeSniffer [16] kiinnittää huomiota rikkeisiin johdonmukaisen ja siistin koodin puolesta. Se ei etsi varsinaisia virheitä, vaan auttaa tuottamaan enemmänkin yleisesti hyvien tapojen mukaista koodia. PHP:n lisäksi PHP_CodeSniffer tutkii JavaScriptiä ja CSS:ää, jotka ovat molemmat mukana tämän työn kohteena olevassa järjestelmässä.

PHP_CodeSniffer tarvitsee myös xml-muotoisen asetustiedoston. Sinne kirjoitetaan PHPMD:n tavoin sääntöjä, joita myötäileväksi koodi halutaan tehdä. Tiedosto on tässä työssä sijoitettu samaan paikkaan kuin PHPMD:n sääntötiedosto, eli Jenkinsin työhakemistosta löytyvään build-kansioon. Sen nimi on phpcs.xml.

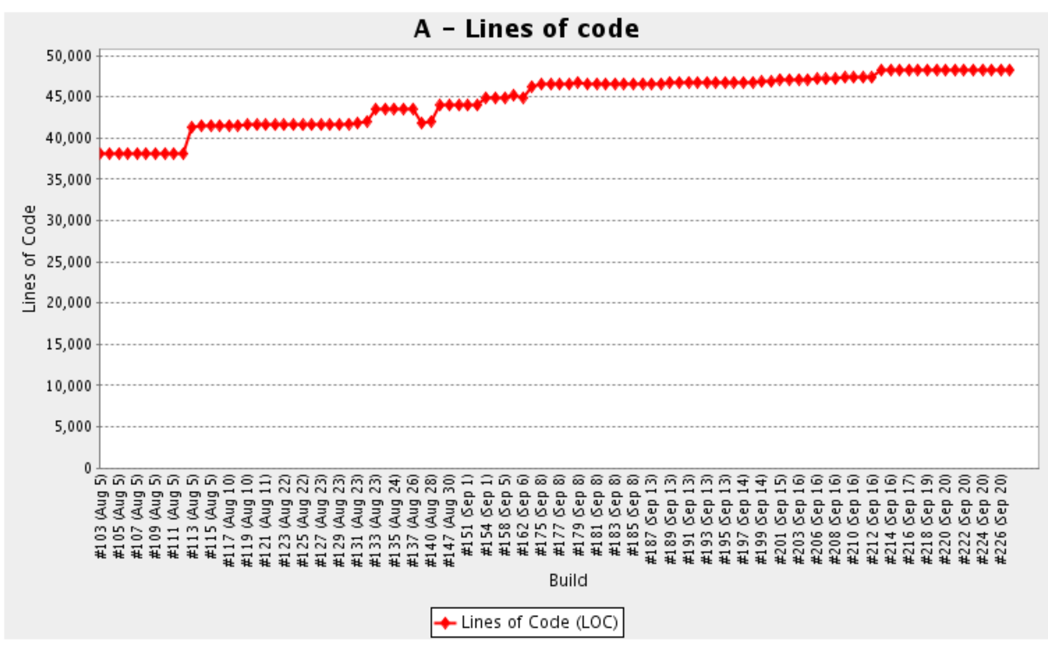
```
<ruleset name="PHP_CodeSniffer">
  <description>Cabforce PHP_CodeSniffer ruleset</description>
  <rule ref="PEAR">
    <exclude name="Generic.WhiteSpace.DisallowTabIndent"/>
  </rule>
</ruleset>
```

Edellinen xml-koodi on esimerkki CodeSnifferin sääntötiedostosta. Kuten PHPMD:ssä, siinä tehdään *ruleset*-merkintä, jonka sisällä kaikki säännöt otetaan mukaan. CodeSnifferin säännöt löytyvät sen internetsivuilta [16], mutta tarkemmin käytössä olevaa versiota vastaavat voi listata asennushakemistosta ” /usr/share/pear/PHP/CodeSniffer”. Tämä onnistuu esimerkiksi komennolla: *find ./ -name "*TabIndentSniff.php" -and -not -name "Abstract*".*

Edeltävän xml-tiedostoesimerkin mukaisesti sääntöjä otetaan käyttöön *exlude*-merkinnällä. Mukaan otettavan sääntötiedoston saa mukaan erottamalla sen hakemistopolun pisteillä ja kirjoittamalla perään sen PHP-tiedoston nimen. Jenkinsin puolella CodeSnifferin tulosten näyttäminen vaatii Checkstyle-lisäosan [17].

PHPCPD (PHP Copy/Paste Detector) [18] kiinnittää huomiota ohjelmakoodissa useampaan kertaan esiintyvään koodiin. Tulokset ohjataan tässä projektissa xml-tiedostoon ja näytetään Jenkinsin web-käyttöliittymässä DRY-lisäosan [19] avulla.

PHPLOC [20] on ohjelma, jolla mitataan sovelluksen kokoa. Se laskee ohjelmistossa mukana olevien kansioiden ja tiedostojen sekä koodirivien määrät. *PHPLOC* laskee lisäksi luokkien, metodien ja funktioiden määrät. Se selvittää myös niiden näkyvyyden. PHPLOC:in tulokset ohjataan csv-tiedostoon ja niitä käsitellään Jenkinsissä työkalulla nimeltä *Plot* [21]. Se on graafinen työkalu, jolla voi tuottaa testituloksiin perustuvia havainnollistavia kaavioita. Kuva 10 esittää Plot-ohjelman generoimaa kuvaajaa.

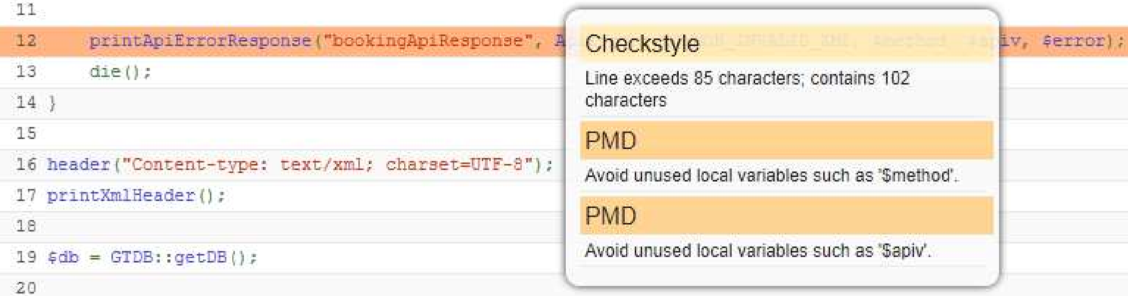


Kuva 10. Plot

Kuvaaja esittää ohjelmiston kasvua. Vasemmassa reunassa oleva mitta-asteikko kuvaa koodirivien määrää. Kuvassa alhaalla puolestaan on päivämäärä, jolloin PHPLOC on laskenut kohteena olevasta ohjelmistosta koodirivien määrän.

PHPLOC on melko monipuolista raporttia tuottava ohjelmisto ja Plotin avulla sen laskemista arvoista voi määritellä paljon erilaisia kaavioita. Mitatut arvot säilytetään Jenkinsin työhakemistossa, joten Plotilla piirretyt kuvaajat perustuvat useamman eri käännöksen PHPLOC-tulokseen. Lisäksi Plotiin on määriteltävissä, kuinka monen käännöksen tuloksia se käyttää kuhunkin kaavioon. Näin kaaviot havainnollistavat ohjelmiston kehitystä tietyllä aikavälillä.

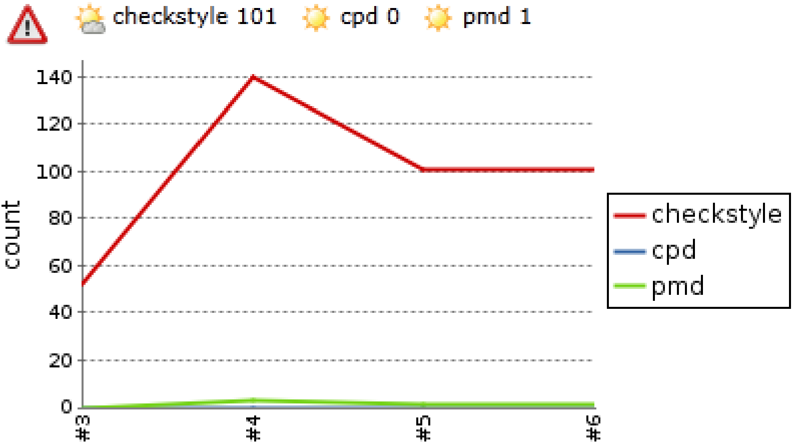
CodeBrowser [22] nitoo testaus- ja analysointityökalujen tuloksia yhteen luodakseen helposti silmäiltävän, kokonaisvaltaisen raportin. CodeBrowser-raportissa ovat mukana tulokset ohjelmista PHPUnit, PHPMD, PHPCPD ja CodeSniffer. CodeBrowser-ohjelman tekemä kooste näytetään Jenkinsin web-käyttöliittymässä kuvan 11 mukaisesti.



Kuva 11. CodeBrowser

Kuva 11. CodeBrowser selventää CodeBrowserin toimintaa web-käyttöliittymässä. CodeBrowser värjää koodista eriväriseksi kaikki kohdat, joissa esiintyy jonkin testaus- tai analysointityökalun löytämä virhe. Kun käyttäjä siirtää hiiren kursorin tällaisen kohdan päälle, CodeBrowser näyttää löydetyt virheet. CodeBrowser on suureksi avuksi, kun testauskehykseen kuuluu monta erillistä työkalua, joista jokainen tuottaa omaa raporttiaan. Ohjelmakoodin tarkastelija voi tutkia virheitä käyttämällä vain yhtä työkalua sen sijaan, että tutkisi erikseen kaikkien työkalujen tuottamia raportteja.

Jenkinsin lisäosa nimeltä *Violations* [23] on myös testaustuloksia tarkasteleva ohjelma. Se kokoaa tulokset ohjelmista Checkstyle, PMD ja CPD. Näiden pohjalta Violations pitää lukua ohjelmistossa esiintyvistä virheistä ja tuottaa kaavioita määrätyllä aikavälillä esiintyvistä virheiden määristä. Kaavioiden avulla on mahdollista seurata, miten virheiden määrä on hallinnassa. Lisäksi kaavioista huomaa helposti, jos kehityksessä tapahtuu äkillisempi muutos, joka johtaa virheiden lisääntymiseen. Kuva 12 on esimerkki Violationsin tuottamasta kaaviosta.



Kuva 12. Violations-kuvaaja

Kuvassa on esimerkki tilanteesta, jossa Violations-ohjelman kuvaajassa esiintyy piikki. Kuvaajan vasen reuna esittää virheiden määrää ja alareuna käännöstä, jossa virheiden määrä on mitattu. Siinä on helposti nähtävissä, että checkstyle-ohjelma on havainnut huomattavan paljon virheitä käännöksessä 4.

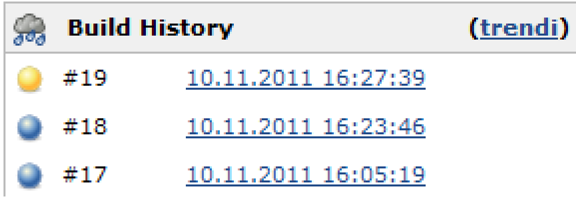
Kun virheitä mukanaan tuova käännös on saatu selville, Jenkinsin web-käyttöliittymästä voi avata kyseisen käännöksen ja tutkia sitä tarkemmin. Näin Violations auttaa selvittämään syitä, jotka lisäävät virheiden määrää ohjelmistossa. Violationsin keräämät tulokset vaikuttavat myös Jenkinsin etusivulla näkyviin, projektien tilaa kuvaaviin ilmaisimiin (Kuva 4). Tämä mahdollistaa useamman projektin tarkkailemisen kerralla siltä varalta, että näissä tapahtuisi äkillisempiä muutoksia virheiden määrässä.

PHPDocumentor [24] on ohjelma, jolla tuotetaan automaattisesti PHP-koodista dokumentointi. Se esittelee ohjelmakoodin rakenteen ja erittelee luokat, funktiot ja muuttujat. PHPDocumentorin käyttö ei sellaisenaan käy ohjelmiston dokumentoinnista, mutta se luo hyvän pohjan ja mallin sille. Jenkinsin web-käyttöliittymään saa selkeän tulosteen PHPDocumentorin tuottamasta aineistosta.

Kaikki edellä mainitut testaus- ja analysointityökalut käyttävät päälevityskanavanaan PEAR:ia. Tämä on PHP:n sisarprojekti, joka tarjoaa julkaisujärjestelmän PHP-komponenteille. Testaus- ja analysointiohjelmistojen asennus on johdonmukaista PEAR:ia käyttämällä. Näin myös niiden päivittäminen automatisoituu ja ne pysyvät uusimman PHP-version mukaisina.

4.3 Tulosten käsittely

Jenkinsin web-käyttöliittymän etusivulla (Kuva 4. Jenkins-projektit) on esitetty yhteenveto testaus- ja analysointityökalujen tuloksista. Näin käyttäjät näkevät nopeasti kaikkien Jenkinsiin sisällytettyjen projektien yleisen tilanteen. Värikoodit (sininen, keltainen ja punainen) sekä sääilmaisimet helpottavat tilanteen yleistä katselmointia. Jos projektin väri on esimerkiksi vaihtunut sinisestä keltaiseen, tämä viestii ohjelmoijille, että virheiden esiintymisen sallittu raja on ylitetty ja toimenpiteisiin on ryhdyttävä. Ensin on otettava selvää tilanteen aiheuttajista. Se on tehokkainta aloittaa avaamalla projekti ja tarkastelemalla sen käännöshistoriaa (Kuva 13. Käännöshistoria).



Kuva 13. Käännöshistoria

Kuvassa on esimerkki tilanteesta, jossa projektiin on tullut liikaa virheitä. Listassa näkyvä, keltaisella pallolla kuvattu käännös ilmoittaa selkeästi käännöksen, jossa virheiden esiintymisen sallittu raja on ylittynyt. Kuvassa näkyvät käännökset toimivat linkkeinä kullekin käännökselle.

Kun Jenkinsin käännöshistoriasta on paikannettu kohta, jossa virheitä on esiintynyt liikaa, voi kyseisen käännöksen avata yksityiskohtaisempaan näkymään. Näin käännöksen virheitä on mahdollista päästä tutkimaan tarkemmin (Kuva 14).

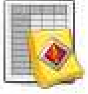
Käännös #19 (10.11.2011 16:27:39)

Revisio: 141
Muutokset

1. Lisatty Validate.php ja backup.php (yksityiskohdat)

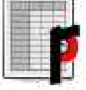
Started by an SCM change

Started by user Sami



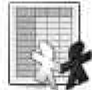
Checkstyle: 298 warnings from one Checkstyle file.

- 196 new warnings



PMD: 25 warnings from one PMD file.

- 24 new warnings



Duplicate Code: 0 warnings from one analysis file.

- No warnings since build 1.
- New zero warnings highscore: no warnings for 39 days!

Kuva 14. Avattu käännös

Kuva 14 on esimerkki avatusta käännöksestä, joka on aiheuttanut liikaa virheitä projektiin. Jenkins pitää kirjaa tietovarastoon tehdyistä muutoksista ja näyttää muutoksia tehneiden kehittäjien kommentit kussakin käännöksessä. Käännöksistä on helppo havaita kunkin testaus- ja analysointityökalun havaitsemat rikkeet. Kuten kuvasta käy ilmi, Jenkins näyttää rikkeiden kokonaismäärän sekä myös tarkasteltavan käännöksen yksinään tuomat rikkeet. Kuvassa näkyvien analysointityökalujen linkkien kautta pääsee kyseisen työkalun osiolle, jossa virheet on eritelty tarkemmin.

Jos käännöksessä esiintyy punainen ilmaisin, kyseessä on yleensä vakavampi, ohjelman toiminnallisuuteen vaikuttava virhe. Tällainen aiheutuu tavallisesti PHPUnitin havaitsemasta virheestä. Kun tällainen tilanne tulee, kannattaa avata ongelman aiheuttava käännös ja tarkastaa PHPUnitin tilanne. Jos siinä esiintyy virheitä, käännöksestä on avattavissa PHPUnitin tuottama virheraportti (Kuva 15).

Failed

ClassTest.testParameter

Stacktrace

```
ClassTest::testParameter
Failed asserting that <integer:2> matches expected <integer:1>.

/home/sami/.jenkins/jobs/ProjectX/workspace/tests/ClassTest.php:9
```

Kuva 15. PHPUnit virheraportti

Virheraportissa näytetään testi, joka on löytänyt virheen sekä syy, josta virhe aiheutuu. Kuvan esittämässä tapauksessa virhe on testistä, jossa funktiolle syötetty parametri ei tuota halutunlaista paluuarvoa. Testiin on määritelty odotettavaksi paluuarvoksi 1, mutta se on saanut arvon 2. Virheraportista näkyy lisäksi polku, jossa virheen löytänyt testi sijaitsee.

Jenkins mahdollistaa virheraporttien ohjaamisen sähköpostiosoitteisiin. Tämä on hyödyllinen toiminto sikäli, että virheellisen ohjelmakoodin tietovarastoon toimittamisesta seuraa välitön palaute ja siihen voidaan puuttua nopeasti. Virheraportit ohjataan sähköpostiin Jenkinsin web-käyttöliitymästä (Kuva 16).

E-mail Notification

SMTP server

Default user e-mail suffix

System Admin E-mail Address

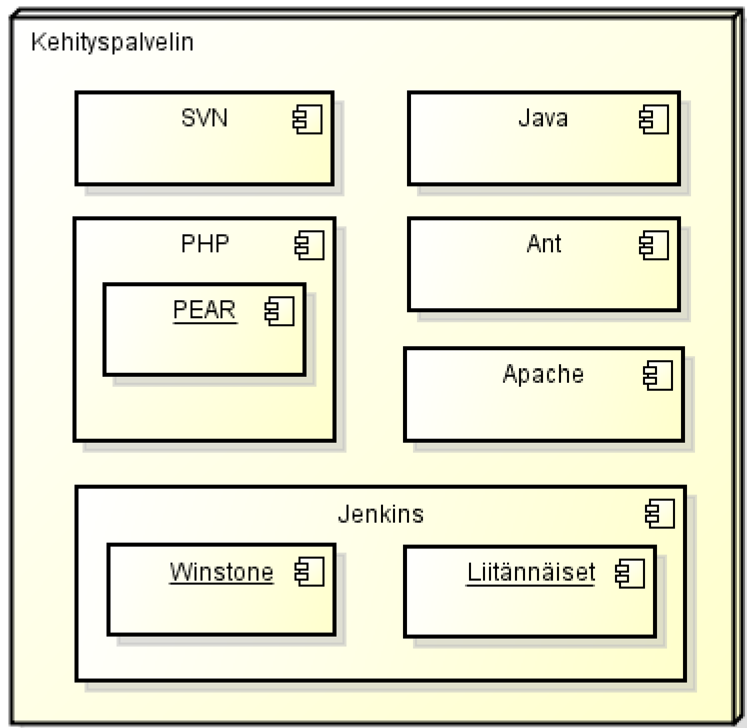
Kuva 16. Virheraportin ohjaus sähköpostiin

Sähköpostiasetuksiin on kirjoitettava SMTP-palvelimen osoite sekä sähköpostiosoite, johon virheraportti toimitetaan.

5 Valmis järjestelmä

Tässä projektissa käsiteltävä kokonaisuus on melko laaja. Mukana on suurempia palvelinohjelmistoja sekä erityisesti yhtenäistä järjestelmää varten kerättyjä pienempiä sovelluksia. Tässä luvussa esitetään järjestelmän eri osa-alueet omina kerroksinaan ja tiivistetään lopputulosta siten, että aikaansaadun järjestelmän vahvuudet ja heikkoudet olisi eriteltävissä mahdollisimman selkeästi. Lopputulosta on hyvä tarkastella siinä mielessä kriittisesti, että toteutusta on tulevaisuudessa mahdollista parannella.

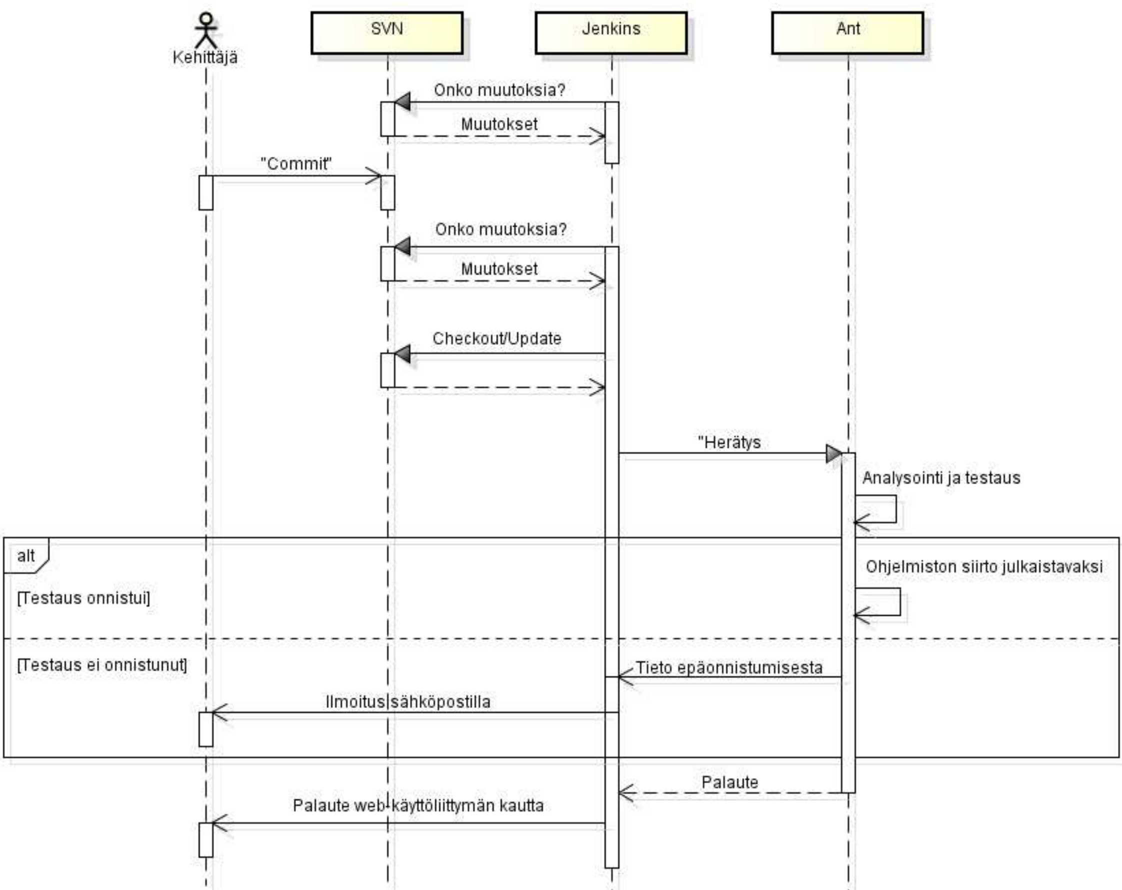
Kokonaiskuvaa hahmottelee parhaiten sen tarkastelu korkeammalta tasolta. Fyysisten laitteiden osalta järjestelmä on yksinkertainen, sillä kaikki ohjelmistot on asennettu yhdelle virtuaalipalvelimelle. Kullakin palvelinohjelmistolla puolestaan on oma vastuunsa pienempien kokonaisuuksien hallitsemisesta (Kuva 17).



Kuva 17. Järjestelmän sijoittelukaavio

Kuvassa 17 on järjestelmän sijoittelukaavio, jossa on eritelty järjestelmän suurimmat ja keskeisimmät osat. Se näyttää tämän työn kokonaiskuvan ja selventää komponenttien sijaintia toisiinsa nähden.

Palvelinohjelmistojen asentaminen vain yhdelle fyysiselle laitteelle ja niiden suorittaminen siellä on hallinnan kannalta helppoa, mutta jatkuvaan integraatioon liittyvä testaus on kuitenkin melko raskasta palvelimen muistille ja prosessorille. Testausten suorittaminen esiintyy kuormituspiikkeinä ja se saattaa hidastaa muuta palvelimella suoritettavaa toimintaa. Tähän tulisi kiinnittää erityistä huomiota, jos samalla palvelimella on tarkoitus suorittaa muuta kriittistä toimintaa. Todennäköisesti tämänkin projektin kohteena olevan järjestelmän ja testiaineiston kehittyessä on varmempaa hajauttaa jatkuvaa integraatiota erilliselle palvelimelle. Kuormitus riippuu tietysti myös siitä, miten usein prosessi suoritetaan. Tämä puolestaan heijastuu suoraan kehittäjien määrästä ja aktiivisuudesta. Kuva 18 selventää, miten jatkuvan integraation prosessi suoritetaan ja mitä toimenpiteitä se saa aikaiseksi.



Kuva 18. Järjestelmän sekvenssikaavio

Kuvan sekvenssikaavio selventää toimenpiteitä, joita ohjelmoijan versionhallintaan lähettämä tuotos saa aikaiseksi. Kuvassa kehittäjän oikealla puolella olevat palkit ovat järjestelmän tärkeimmät toimivat osat. Jenkins tutkii säännöllisesti tietovarastoa ja, kun tietovarastossa esiintyy muutos, se hakee tuotoksen omaan työtilaansa. Tämän jälkeen Jenkins herättää Antin, joka puolestaan suorittaa koodille analysoinnin ja testauksen käyttäen apunaan luvussa 4 esiteltyjä työvälineitä. Kun testaus menee onnistuneesti läpi, Ant huolehtii ohjelmiston julkaisusta. Jos testauksessa ilmenee virheitä, Ant palauttaa tiedon Jenkinsille, joka puolestaan huolehtii sähköpostin lähettämisestä ohjelmoijille. Jenkins pitää koko ajan kirjaa tapahtumista ja kehittäjät voivat seurata tilannetta esimerkiksi web-käyttöliittymästä.

6 Yhteenveto

Jatkuvaan integraatioon perustuva testausjärjestelmä suunniteltiin vastaamaan mahdollisimman hyvin yrityksen tarpeita. Projekti aloitettiin tietojen keräämisellä ja koko-

naisuuden hahmottelemisella. Näin jo alkuvaiheessa saatiin kohtuullinen käsitys siitä, millaisen järjestelmän osaksi jatkuva integraatio oli suunniteltava. Tällöin oli myös mahdollista rajata tarkemmin työkaluja, joita tähän projektiin tulisi mukaan.

Kun tämän projektin toteuttaminen aloitettiin, yrityksen ohjelmistokehityksen testaukset oli suoritettava täysin manuaalisesti. Myös tietovarastoon tuotettu koodi oli kasattava käsin yhdeksi kokonaisuudeksi oikeaan paikkaan. Järjestelmän valmistuessa oli selvää, että testauksen ja koostamisen automatisoiminen jättää tulevaisuudessa paljon työtunteja pois.

Alkuperäisen suunnitelman mukaan testausjärjestelmän osalta projektiin oli tarkoitus liittää vain PHPUnit hoitamaan ohjelmistokehityksessä tapahtuva yksikkötestaus. Luvussa 3.4 esitelty ”Template for Jenkins Jobs for PHP Projects” -standardointihanke toi kuitenkin mukanaan uusia näkökantoja ja laajensi testausvälineistön kirjoa. Sen lisäksi, että järjestelmä hoitaa alkuperäisen suunnitelman mukaisen yksikkötestauksen ja koostamisen automatisoinnin, sillä voi nyt suorittaa huomattavasti kokonaisvaltaisempaa testausta ja analysointia. Cabforce Oy voi järjestelmän myötä puuttua kohtiin, jotka heikentävät ohjelmiston laatua sekä ohjelmistokehityksen kustannustehokkuutta. Lisäksi yritys pystyy tuottamaan järjestelmän avulla seurantatietoa, siitä miten tehokkaasti ohjelmistoa kehitetään milläkin hetkellä ja miten nopeasti ohjelmisto kasvaa. Tämä auttaa ennakoimaan resurssien tarvetta tulevaisuudessa.

Jenkinsin asentaminen jatkuvan integraation perustaksi osoittautui melko helpoksi ja kuitenkin suhteellisen monipuoliseksi vaihtoehdoksi. Versionhallinnan osalta käyttöönotto sujui vaivattomasti. Koska Jenkins tukee suoraan Cabforce Oy:n käytössä olevaa SVN-versionhallintaa, ohjelmakoodin saaminen jatkuvan integraation alle tapahtui yksinkertaisesti antamalla tarvittavat tiedot ja painamalla nappia Jenkinsin web-käyttöliittymästä. Ohjelmakoodin koostaminen käyttövalmiiksi web-sovellukseksi onnistui myös vähin vaivoin kirjoittamalla tätä varten komentosarja työhön sisällytettyyn Ant-asetustiedostoon.

Suurimmat ongelmat esiintyivät muiden palvelinohjelmistojen kanssa. Esimerkiksi kehityspalvelimen käyttöjärjestelmän oikut Javan sekä Antin suhteen toivat hieman mutkia matkaan, mutta niistäkin päästiin etenemään manuaalisien asennuksien voimin.

Tähän työhön on sisällytetty useita eri testaus- ja analysointityökaluja. Niiden toimintojen tutkiminen ja tarpeellisuuden selvittely kahmaisi paljon aikaa suunnitteluvaiheessa. Myös teknisen toteutuksen puolella näiden sovelluksien asentelu ja virittäminen toimintakuntoon vaati kohtuullisen paljon työtä. Lisäksi lopullisen toimintakunnon aikaansaamiseksi on laskettava mukaan myös eri testausjärjestelmien tarkemmat säädöt sekä yksikkötestien suunnittelu ja kirjoittaminen.

Työn tarkoituksena oli selvittää, miten jatkuva integraatio ja sen tuoma testausautomaatio on tuotavissa osaksi PHP-ohjelmistokehitystä. Järjestelmään sisällytetyt työkalut yhdessä saavat aikaiseksi melko vankan kokonaisuuden PHP-sovellusten testaamiseen ja muutenkin hyvän pohjan yrityksen ohjelmistotestaukselle. PHP:n testaustyökaluja silmälläpitäen noudatettiin pääsääntöisesti luvussa 3.4 esiteltyä "Template for Jenkins Jobs for PHP Projects" -sivustoa. Tämä osoittautui kattavaksi kokonaisuudeksi PHP:n analysointia ja testausta varten. Lisäksi valmiin hyväksi havaitun toteutustavan käyttö ohjenuorana nopeutti työn edistymistä tuoden osansa kustannustehokkuuteen. Sivusto kuitenkin valitettavasti esitteli varsin suppeasti ja pintapuolisesti toimenpiteitä tällaisen projektin aikaansaamiseksi. Esimerkiksi PEAR-paketit vaativat tiettyjen beta-vaiheessa olevien pakettien asentamisen toimiakseen. Lisäksi palvelimelle on kirjoitettava CodeSniffer- ja PHPmd-sovelluksia varten xml-tiedostot, joista sivustolla ei ollut viittausta enempää tietoa. Sivusto ei myöskään juuri ottanut kantaa asennettavien ohjelmien kytköksiin, eikä tarpeellisuuteen. Tämän tapaisen projektin toteuttajan harteille jää siis paljon omakohtaista tutkimustyötä. Esimerkiksi testausohjelmien kuormittavuus palvelinta kohtaan jää avoimeksi ilman kokeilua.

Kun testauspuolen sovellukset on saatu kasaan ja jatkuvan integraation palvelin pyörittämään testit automaattisesti sekä koostamaan tuotos kokonaiseksi web-sovellukseksi, päivittäinen työmäärä testauksen sekä koostamisen osalta vähenee huomattavasti. Jatkuvaan integraatioon perustuvan testausjärjestelmän ohessa kehitettävien ohjelmistojen laatu ja virheettömyys paranevat. Lisäksi ohjelmiston kasvua kuvaavan statistiikan avulla on mahdollista varautua mahdollisiin tulevaisuudessa tapahtuviin resurssivaatimuksiin. Koska Jenkins pitää tallessa tietoa edellisien käännöksien tapahtumista, järjestelmä tuottaa havainnollista tietoa myös ohjelmiston laadun kehittymisestä. Kun jatkuva integraatio nopeuttaa yrityksen ohjelmistokehitystä, se saa aikaiseksi myös

kyvyn vastata nopeasti ohjelmistoon kohdistuviin muutostarpeisiin. Tästä on erityistä hyötyä yritysten välisessä kilpailussa.

Jatkuvan integraation pystyttäminen osaksi PHP-ohjelmistokehitystä on varsin perusteltua, kun pohjataan säästyviin resursseihin sekä virheettömämpien ohjelmistotuotteiden aikaansaamiseen. Toteuttaminen vaatii jonkin verran aikaa ja vaivaa, mutta maksaa itsensä varmasti takaisin myöhemmin.

Lähteet

[1] Ketterän ohjelmistokehityksen julistus [Verkkodokumentti] [Viitattu 5.6.2011], saatavissa: http://www.agilemanifesto.org/.

[2] Ketteryys [Verkkodokumentti] [Viitattu 5.6.2011], saatavissa: http://www.ketteratkaytannot.fi/fi-FI/Ketteryys/IteraatiotJaInkrementit/.

[3] Martin Fowler, Continuous Integration [Verkkodokumentti] [Viitattu 5.6.2011]. Saatavissa : http://www.martinfowler.com/articles/continuousIntegration.html.

[4] Xinc, saatavissa: http://code.google.com/p/xinc/. Luettu 6.6.2011.

[5] 10 reasons to switch from CruiseControl to Hudson [Verkkodokumentti] [Viitattu 10.5.2011] Saatavissa: http://blog.fedecarg.com/2009/03/05/10-reasons-to-switch-from-cruisecontrol-to-hudson/.

[6] Hudson [Verkkodokumentti] [Viitattu 14.6.2011]. Saatavissa: http://en.wikipedia.org/wiki/Hudson_(software).

[7] Jenkins, saatavissa: http://jenkins-ci.org/. Luettu 6.6.2011.

[8] Template for Jenkins Jobs for PHP Projects [Verkkodokumentti] [Viitattu 15.06.2011], saatavissa: http://jenkins-php.org/.

[9] Phing, saatavissa http://www.phing.info/trac/. Luettu 6.6.2011.

[10] Ant, saatavissa: http://ant.apache.org/. Luettu 10.6.2011.

[11] PHPUnit, saatavissa: http://pear.php.net/package/PHPUnit/redirected. Luettu 4.6.2011.

[12] PHPUnit Manual [Verkkodokumentti] [Viitattu 15.6.2011], saatavissa: http://www.phpunit.de/manual/current/en/.

[13] PHP_Depend, saatavissa: http://pdepend.org/. Luettu 12.6.2011.

[14] PHPMD, saatavissa: http://phpmd.org/. Luettu 12.6.2011.

[15] PHPMD, Howto create a custom ruleset, [Verkkodokumentti] [Viitattu 15.6.2011], saatavissa: http://phpmd.org/documentation/creating-a-ruleset.html.

[16] PHP_CodeSniffer, saatavissa: http://pear.php.net/package/PHP_CodeSniffer/. Luettu 13.6.2011.

[17] Checkstyle, saatavissa: https://wiki.jenkins-ci.org/display/JENKINS/Checkstyle+Plugin . Luettu 12.6.2011.

[18] PHPCPD, saatavissa: https://github.com/sebastianbergmann/phpcpd. Luettu 13.6.2011.

[19] DRY, saatavissa: https://wiki.jenkins-ci.org/display/JENKINS/DRY+Plugin. Luettu 14.6.2011.

[20] PHPLOC, saatavissa: https://github.com/sebastianbergmann/phploc. Luettu 14.6.2011.

[21] Plot, saatavissa: https://wiki.jenkins-ci.org/display/JENKINS/Plot+Plugin. Luettu 14.6.2011.

[22] CodeBrowser, saatavissa: http://blog.mayflower.de/archives/464-PHP_CodeBrowser-Release-version-0.1.0.html. Luettu 14.6.2011.

[23] Violations, saatavissa: https://wiki.jenkins-ci.org/display/JENKINS/Violations. Luettu 15.6.2011.

[24] PHPDocumentor, saatavissa: http://www.phpdoc.org. Luettu 15.6.2011.