WEB-SOVELLUSTEN TESTAUKSEN AUTOMATISOINTI

Jarkko Jäsberg

Opinnäytetyö
Huhtikuu 2011

Ohjelmistotekniikka
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

Tekijä(t) JÄSBERG, Jarkko	Julkaisun laji Opinnäytetyö	Päivämäärä 12.04.2011
	Sivumäärä 37	Julkaisun kieli Suomi
	Luottamuksellisuus ( )	Verkkojulkaisulupa myönnetty (X)

Työn nimi
WEB-SOVELLUSTEN TESTAUKSEN AUTOMATISOINTI

Koulutusohjelma
Ohjelmistotekniikka

Työn ohjaaja(t)
SALMIKANGAS, Esa

Toimeksiantaja(t)
Kilosoft Oy,
RANTAMÄKI, Anssi

Tiivistelmä

Opinnäytetyön tarkoituksena oli määrittää tarvittavat ohjelmat, työkalut ja sovelluskehykset samantapaisille testausautomaation toteuttamisprojekteille web-sovelluksille.

Työssä kuvataan testausautomaation toteuttamisen eri vaiheita ja annetaan ohjeita siitä kuinka automaatio tulisi toteuttaa eri vaiheiden aikana. Työssä esitellään erilaisten työkalujen, ohjelmien ja sovelluskehysten vaatimuksia ja ominaisuuksia, joita tarvitaan samantapaisissa testausautomaation toteuttamisprojekteissa. Työssä puhutaan myös ketteriin ohjelmistonkehitysmenetelmiin liittyvästä jatkuvasta integraatiosta ja siitä kuinka testausautomaatio tulisi liittää integraatiopalvelinohjelmiin.

Integraatio-, kuormitus- ja käyttöliittymä testit toteutettiin prototyyppiohjelmaan, joka rakennettiin opinnäytetyöprosessin aikana. Testit toteutettiin tässä työssä esiteltyjen työkalujen, ohjelmien ja sovelluskehysten avulla. Kaikki testit liitettiin prototyyppiohjelman rakentamisautomaatioon, jolloin pystyttiin ajamaan kaikki testit automaattisesti oikeassa järjestyksessä ja saamaan HTML-pohjaiset testitulokset testien ajamisen jälkeen.

Työn tuloksena saatiin määritettyä tarvittavat työkalut, ohjelmat ja sovelluskehykset samantapaisille testausautomaation toteuttamisprojekteille. Työssä otetaan huomioon myös kustannustehokkuus ja kaikki työssä käytettävät ohjelmat ovat saatavilla ilmaiseksi.

Avainsanat (asiasanat)
testaus, testausautomaatio, jatkuva integraatio, selenium, jmeter, junit.

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
JÄSBERG, Jarkko	Bachelor´s Thesis	12042011
	Pages 37	Language Finnish
	Confidential ( ) Until	Permission for web publication (X)

Title
WEB APPLICATIONS TEST AUTOMATION

Degree Programme
Software engineering

Tutor(s)
SALMIKANGAS, Esa

Assigned by
Kilosoft Ltd,
RANTAMÄKI, Anssi

Abstract

The purpose of this bachelor's thesis was to determine the necessary tools, software and frameworks for similar test automation implementation projects used for web-based applications.

This study contains the description of different phases of the test automation implementation process and the thesis discusses how test automation should be implemented on different phases of the process. The thesis presents features and requirements of tools, software and frameworks needed for similar test automation implementation projects. The thesis also discusses an agile software development technique called continuous integration and how test automation should be integrated into continuous integration systems.

Integration-, load- and user interface tests were made to the prototype application which was built during the process of this thesis. The tests were made with tools, software and frameworks presented in this thesis. All tests were attached to the prototype application build automation and after that it was possible to run all tests automatically in the right order and get HTML based test results after the execution of the tests.

As a result of this thesis necessary tools, software and frameworks were determined for a similar test automation implementation project. The thesis also takes into account cost effectiveness; all software used in this thesis are available for free.

Keywords

test, test automation, continuous integration, selenium, jmeter, junit.

Miscellaneous

SISÄLTÖ

KUVIOT

KÄSITTEET

FTP (File transfer protocol)

Tiedostojen siirtämiseen tarkoitettu protokolla. FTP-yhteys toimii asiakas palvelin periaatteella.

HTML (Hypertext markup language)

Standardoitu merkkauskieli web-sivujen tekoon. Uusin HTML 5.0 versio julkaistiin tammikuussa 2008.

HTTP(S) (Hypertext transfer protocol)

Protokolla web-selaimien ja HTTP-palvelimien väliseen tiedon siirtoon. HTTPS protokollassa tiedot salataan ennen lähetystä SSL- tai TSL-protokollaa käyttämällä.

Jmeter

Web-pohjaisten ohjelmien kuormitus- ja kestävyystestausohjelma. Soveltuu monien eri rajapintojen testaamiseen.

JDBC (Java database connectivity)

Java-ohjelmointikielen standardoitu tapa tietokantojen käyttöä varten. Tarjoaa metodit tietojen kyselyyn ja päivittämiseen relaatio-tietokantoihin.

Junit

Sovelluskehys Java-ohjelmointikielellä toteutettujen ohjelmien yksikkötestausta varten. Yksi xUnit-sovelluskehysperheen sovelluskehyk-

sistä.

LDAP (Lightweight Directory Access Protocol)

Hakemistopalvelujen käyttöön tarkoitettu verkkoprotokolla. Yksikertaisempi versio X.500 hakemistopalvelulle. Hakemistorakenne on avain-arvo pareista koostuva puurakenne.

Maven

Java-ohjelmointikielellä toteutettujen projektien rakentamisen, raportoinnin ja dokumentoinnin automatisointiin tarkoitettu ohjelma.

Selenium

Web-pohjaisten ohjelmien käyttöliittymän testausjärjestelmä. Sisältää monenlaisia ohjelmia ja sovelluskehyksiä käyttöliittymätestauksen toteuttamiseksi.

SMTP (Simple mail transfer protocol)

Sähköpostiviestien välittämiseen tarkoitettu protokolla.

SOAP (Simple object access protocol)

Tietoliikenneprotokolla joka mahdollistaa proseduurien etäkutsun. Pohjautuu XML-kieleen ja toimii useiden eri protokollien ylitse.

Spring

Avoimen lähdekoodin sovelluskehys Java/J2EE pohjaisten ohjelmien toteuttamiseen.

Spring Roo

Web-pohjaisten ohjelmistojen kehitystyökalu Spring sovelluskehystä

varten.

XML (Extensible Markup Language)

Merkintäkieli, jolla tiedon merkitys on kuvattavissa tiedon sekaan. XML-kieltä käytetään formaattina tietojen välitykseen ja tallentamiseen.

Xpath (XML path language)

Kieli XML-pohjaisten dokumenttien osien osoittamiseen ja rakenteeseen perustuvaan tietojen muokkaamiseen ja luontiin. Xpath perustuu XML-pohjaisten dokumenttien puumuotoiseen esitystapaan.

1. TYÖN LÄHTÖKOHDAT

1.1 Toimeksiantaja

Kilosoft Oy on verkonhallintatuotteita, ohjelmistokehitystä ja laadunvarmistusta tarjoava ohjelmistoalan yritys. Kilosoft työllistää tällä hetkellä noin 80 henkilöä Jyväskylässä, Tampereella ja Espoossa. Kilosoftin asiakkaat toimivat maanpuolustuksen, tietoturvateknologioiden, mobiiliteknologioiden, tietoliikenteen, taloushallinnon ja teollisuusautomaation parissa. Kilosoftin liiketoimintaan kuuluvat TitanNMS- ja NetWrapper-teknologioihin pohjautuvien verkonhallintajärjestelmien suunnittelu, toteutus ja myynti sekä tietoturvaraportointipalvelujen tuottaminen FiERS-teknologian avulla. Lisäksi Kilosoft tarjoaa asiakkaille projektitoimituksia, teknologiaresurssointia, konsultointia ja koulutusta. (Kilosoft Yritysinfo 2010.)

1.2 Opinnäytetyön tavoite

Työn tarkoituksena oli määrittää työkalut samantapaisille web-sovelluksten testausprojekteille toimeksiantajalla. Tavoitteena oli tutkia testausautomaation vaiheita, menetelmiä ja konkreettisesta toteutuksesta erilaisten sovelluskehysten ja ohjelmistojen avulla. Työssä käsiteltiin myös ohjelmien rakentamisautomaatiosta huolehtivia työkaluja, joiden avulla testien suorittaminen ja raportoiminen pystyttiin automatisoimaan. Työssä pyrittiin määrittämään mahdollisimman laajasti erilaisia testausmenetelmiä web-sovellusten testaamisen toteutukseen. Työssä käsiteltiin myös ketteriin ohjelmistojenkehitysmenetelmiin liittyvää jatkuva integraatio periaatteen mukaista käytäntöä ja sitä, kuinka testausautomaatio tulisi liittää osaksi jatkuvaa integraatiojärjestelmää.

2. TESTAUSAUTOMAATION TOTEUTUKSEN VAIHEET

2.1 Yleistä

Testausautomaation tarkoituksena on tuoda säästöjä testauksen toteuttamiseen ja vähentää testien ajamiseen kuluvaa aikaa. Testien ajon nopeutumisen takia pystytään ajamaan suuria määriä testejä lyhyessä ajassa. Lisäksi testausautomaatiolla pystytään vähentämään riskejä, koska automatisoidut testit ajetaan aina samalla tavalla, mutta manuaalisessa testauksessa testejä ei välttämättä ajeta joka kerta samalla tavalla ihmisten tekemien virheiden takia. (Fewster & Graham 1999, 346-347.)

2.2 Suunnittelu

Testitapausten suunnittelu määrittää, miten jonkin ohjelmiston osa tullaan testaamaan. Testitapausten suunnittelun pitäisi tuottaa testejä, joilla on tietyt arvot testin tarvitsemalle oletustilalle, lähetettävällä tiedolle ja oletetuille tuloksille. (Fewster & Graham 1999, 14.) Jokaiselle ohjelmistolle on olemassa suuri määrä mahdollisia testejä ja näistä testeistä tulisi valita automatisoitaviksi vain sellaiset, joilla saadaan havaittua mahdollisimman suuri virhemäärä testattavasta ohjelmistosta (mts. 4). Testejä ei kannata automatisoida tilanteissa, joissa testejä tullaan ajamaan todella harvoin tai tuloksien tarkistaminen ihmisiltä onnistuu helposti, mutta ohjelmallisesti se on todella vaikeata (mts. 22).

2.3 Toteutus

Testitapaukset pitäisi pystyä ajamaan aina kun ohjelmasta tehdään uusi versio, jotta pystytään varmentamaan ohjelman vanhojen ominaisuuksien toiminta (Loveland, Miller, Prewitt & Shannon 2004, 170). Testiohjelman koodin tulisi olla helposti muokattavissa ja uudestaan käytettävissä. Fyysisten resurssien osoitteet tulisi luoda konfiguroitaviksi eikä kirjoittaa suoraan testausohjelman lähdekoodiin. Päällekkäisten testien toteuttamista tulisi välttää, koska on turhaa testata jotain, minkä jo toinen testitapaus on testannut. Yhden testitapauksen tulisi testata vain tiettyä toiminnallisuutta testattavasta ohjelmasta. (Mts. 178-181.)

Testiohjelman dokumentointi on yhtä tärkeää kuin minkä tahansa muunkin ohjelman dokumentointi. Huonosti dokumentoitujen testien kanssa työskenteleminen kuluttaa vain turhaan aikaa. Testien dokumentointi tulisi pitää oikealla tasolla ja käytännöllisenä. Ohjelmistotestaajan tulisi saada dokumentaatiosta selville, mitä kukin testi tekee.(Fewster & Graham 1999, 198.) Usein on turhaa tehdä erillistä dokumentaatiota testitapauksien toiminnasta, koska testien riittävä kommentointi kertoo, mitä ohjelma testaa (Loveland, Miller, Prewitt & Shannon 2004, 178).

2.4 Testien suorittaminen

Automatisoidut testit voidaan suorittaa joko manuaalisesti ajamalla ne ohjelmointiympäristöstä tai automaattisesti liittämällä ne osaksi ohjelman rakentamisprosessia. Täysin testatut projektit voivat sisältää yhtä paljon testi luokkia kuin ohjelman toiminnallisuuteen tarvittavia luokkia. Tämän takia ei voida olettaa, että ohjelmistotestaajat ajaisivat kaikki testit päivittäin manuaalisesti. Testit pystytään liittämään osaksi ohjelman rakentamisprosessia käyttämällä esimerkiksi

Maven tai Ant työkaluja. Testitulosten raportointi pystytään myös automatisoimaan edellä mainituilla työkaluilla, jolloin testien suorittamisen jälkeen saadaan HTML-pohjaiset raportit testien tuloksista. (Tahchiev, Leme, Massol & Gregory 2010, 134-168.)

2.5 Ylläpito

Kun ohjelmistosta julkaistaan uusi versio, tarvitaan aina uusia testejä ja vanhojen testien muuttamista toimimaan uuden ohjelmiston mukaisesti. Osa vanhoista testeistä saattaa tulla tarpeettomiksi, koska niiden testaama toiminnallisuus puuttuu kokonaan uudesta versiosta tai uudet testit korvaavat niiden toiminnallisuuden. (Fewster & Graham 1999, 191.)

Testien ylläpitoon vaikuttaa moni asia kuten,

- testien määrä
- testien käyttämän tiedon määrä
- testien ajamiseen kuluva aika
- testien dokumentaatio
- testien monimutkaisuus.

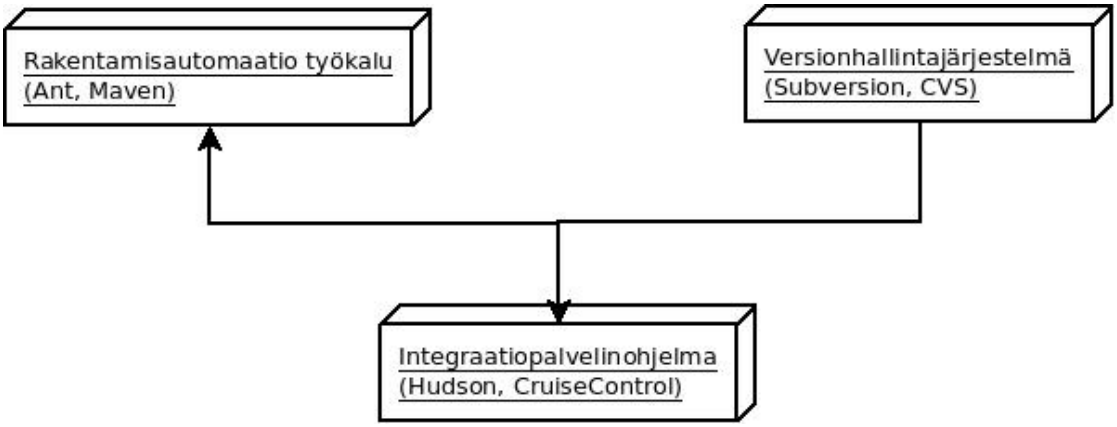
Mitä enemmän on testejä, sitä enemmän kuluu aikaa ja vaivaa testien ylläpitämiseen. Jos testit käyttävät jotain ulkopuolista tietolähdettä, tarvitsee myös kyseiset tiedot päivittää ohjelmaa ylläpidettäessä. Testien suorittamiseen kuluva aika vaikuttaa myös testien ylläpidon kestoon, koska vikojen etsintä pitkään kestävistä testeistä on hitaampaa. Pitkäkestoiset testit voivat myös testata useampaa asiaa samassa testissä, jolloin yhden asian korjaaminen testistä ei välttämättä korjaa koko testiä. Testien ylläpito vaikeutuu huomattavasti jos testien dokumentaatio on huono. Testit tulisi pitää mahdolli-

simman yksinkertaisina, koska monimutkaisten testien ymmärtämiseen ja ylläpitämiseen kuluu paljon aikaa. (Fewster & Graham 1999, 192-198.)

3. JATKUVA INTEGRAATIO

3.1 Yleistä

Ketteriin ohjelmistonkehitysmenetelmiin liittyvässä jatkuva integraatio periaatteen mukaisessa prosessissa on tarkoitus välttää integraatiosta koituvat haitat ja nopeuttaa uusien ominaisuuksien integroimista olemassa olevaan ohjelmaan. Jatkuvasta integroinnista huolehtiva järjestelmä voi koostua esimerkiksi integraatiopalvelinohjelmasta, ohjelman rakentamisautomaatiosta huolehtivasta ohjelmasta ja versionhallintajärjestelmästä (ks. kuvio 1).



KUVIO 1: Integraatio järjestelmän toiminta

3.2 Ohjelmiston rakentaminen

Yksi tärkeimmistä tehtävistä jatkuva integraatio periaatteen mukai-

sessa prosessissa on taata sulavasti toimiva ohjelmiston rakentamisprosessi (build process), jossa vältetään integraatiosta johtuvat ohjelmiston kehityksen viivästykset ja haitat (Smart, J. 2009). Ohjelmiston rakentamisprosessiin kuuluvat esimerkiksi lähdekoodien kääntäminen, tarvittavien tiedostojen kopiointi, tietokannan luonti ja yksikkötestien ajo, joten rakentamisprosessin tekeminen käsin voi viedä todella paljon aikaa. Tämän takia jatkuva integraatio periaatteen mukaisessa prosessissa ohjelman rakentaminen automatisoidaan tähän tarkoitukseen tehtyjen työkalujen avulla. (Fowler 2006.)

Java-ohjelmointikielellä toteutettujen ohjelmien käytetyimpiä rakentamisprosessista huolehtivia työkaluja ovat Ant ja Maven. Molempien työkalujen pääasiallisena tarkoituksena on ajaa muita ohjelman rakentamisprosessissa tarvittavia työkaluja. Molempien työkalujen toiminta perustuu projektiin liitettävään XML-pohjaiseen konfiguraatio tiedostoon, johon määritetään rakentamisprosessin tarvitsemat tiedot. (Tahchiev, Leme, Massol & Gregory 2010, 134-168.)

3.3 Automatisoitujen testien suorittaminen

Automatisoitujen testien suorittaminen liitetään rakentamisprosessiin, koska tällä tavalla saadaan tietoa mahdollisista virheistä nopeasti ja tehokkaasti (Fowler 2006). Rakentamisprosessi voi sisältää esimerkiksi yksikkö-, integraatio- ja käyttöliittymätestejä. Ohjelman rakentaminen tulisi kuitenkin toimia nopeasti ja tämä vaikuttaa automatisoitujen testien suunnitteluun ja niiden ajamiseen. Suunnitteluvaiheessa olisi hyvä yrittää välttää pitkään kestäviä ja tehottomia testejä. Integraatio- ja yksikkötestit tulisi toteuttaa toisistaan erillään. Käyttöliittymätestien suoritus voi kestää todella pitkän aikaa, joten tarvittaessa testejä tulisi ajaa usealla tietokoneella yhtäaikaisesti. Eri testityypit tulisi ajaa useassa eri sekvenssissä. Esimerkiksi aluksi ajetaan yksikkötestit ja tämän jälkeen jos yksikkö-

testit menevät lävitse, ajetaan integraatiotestit. Tällä tavalla ohjelmiston rakentamisprosessiin liitettävän testausautomaation ajamiseen käytettävä aika saadaan pysymään lyhyenä. (Smart 2009.)

3.4 Käyttöönotto

Ohjelmiston käyttöönotto (deployment) vaatii monien eri toimintojen suorittamista ja on käsin tehtynä todella paljon aikaa vaativa tehtävä. Ohjelman käyttöönotto pystytään automatisoimaan täysin siihen tarkoitetuilla työkaluilla. Käyttöönottoprosessi vaatii yleensä paljon enemmän suoritettavia tehtäviä kuin ohjelman rakentaminen.(Smart 2009)

Käyttöönottoprosessiin liittyvät toiminnot kuten,

- merkitä lähdekoodi käytettäväksi julkaisu versiossa
- kääntää lähdekoodi
- ajaa yksikkö- ja integraatiotestit
- raportoida ohjelman rakentamisesta
- ohjelman käyttöönotto kohdeympäristössä
- ajaa tietokanta skriptit
- ajaa suorituskykytestit ja toiminnalliset testit
- ilmoittaa sidosryhmille uusimmasta julkaisusta. (Smart 2009.)

3.5 Tietojen välitys

Jatkuvan integraation tärkeimpiä asioita on kommunikointi kaikkien projektin sidosryhmien jäsenten kanssa. Heidän pitäisi helposti

saada selville järjestelmän tila ja siihen tehdyt muutokset. Ohjelmistokehittäjille tärkeimpänä tietona on järjestelmän tämänhetkisen kehitysversion tila. Integraatiopalvelinohjelmistojen (continuous integration server) avulla pystytään pitämään yllä kattavaa tietoa siitä, ketkä ohjelmaa ovat muuttaneet ja mikä kehitettävän ohjelman tila on tällä hetkellä. Tällä tavalla kaikki projektin sidosryhmät pysyvät tietoisena ohjelmiston tämän hetkisestä tilasta ja muutoksista.(Fowler, M. 2006.)

3.6 Keskitetyt tietovarastot

Yleensä ohjelmistoprojektit sisältävät todella monia tiedostoja, joita tarvitaan ohjelman rakentamista varten. Kun projektin parissa työskentelee useita henkilöitä, vaatii tiedostojen hallinta todella paljon työtä. Projektin sisältämien tiedostojen keskittämistä varten on olemassa versionhallintajärjestelmiä (version control systems) ja tietovarastoja (repositories). Näiden järjestelmien avulla on mahdollista ylläpitää tietoa projektin eri tiedostojen versiosta ja tehdä projektista useita eri haaroja (branches), jolloin projekti pystytään jakamaan eri kehitysvaiheisiin. Projektin jakamista eri haaroihin käytetään esimerkiksi siinä tapauksessa, kun halutaan tehdä erillinen versio julkaistusta ja kehityksessä olevasta versiosta. (Fowler, M. 2006.)

3.7 Integraatiopalvelinohjelmat

Integraatiopalvelinohjelmien tarkoituksena on automatisoida ohjelman rakentamisprosessi sekä raportoida projektin rakentamisprosessissa tulevista virheistä ohjelmistokehittäjille. Integraatiopalvelinohjelmia käytetään tietovarastoissa ja versionhallintajärjestelmissä olevien projektien tilan seuraamiseen. Kun ohjelmistokehittäjät

päivittävät tietovastoissa olevaa ohjelmaa, tulee integraatiopalvelinohjelman hakea esimerkiksi tietyin aikavälein automaattisesti uusimmat lähdekoodit tietovarastosta, kääntää ne ja ajaa tarpeelliset testit, sekä raportoida koostamisprosessissa tulevista virheistä takaisin ohjelmistokehittäjille. Yksinkertaisimmillaan palvelin lähettää viestin sähköpostiin rakentamisprosessissa tulleista virheistä. (Fowler 2006.)

4. TYÖKALUT TESTAUKSEN AUTOMATISOINTIIN

4.1 Yleistä

Testausautomaation toteutusta varten on olemassa suuri määrä erilaisia sovelluskehyksiä ja ohjelmia. Sovelluskehyksiä on tehty monille eri ohjelmointikielelle ja monenlaiseen testaamiseen. Ohjelmia ja sovelluskehyksiä saatavilla kaupallisina, ilmaisina ja vapaaseen lähdekoodiin perustuvina.

Sovelluskehykset ja ohjelmat

Työssä käytettiin Junit-sovelluskehystä integraatiotestien toteuttamiseen. Junit-sovelluskehys toimi myös pohjana käyttöliittymätestien toteutukselle. Sovelluskehyksen valintaan vaikutti paljon sen laajennettavuus eri käyttötarkoituksiin. Sovelluskehys pystyttiin liittämään lisäosien avulla rakentamisautomaatiosta huolehtivan Maven ohjelman konfiguraatiotiedostoon, jolloin pystyttiin automatisoimaan testien suorittaminen ja raportointi.

Käyttöliittymätestien toteuttamiseen käytettiin Selenium-testausjärjestelmää. Järjestelmän valintaan vaikuttivat sen todella monipuoliset ominaisuudet ja siihen saatavat lisäosat. Selenium-järjestelmän avulla pystytään esimerkiksi tekemään käyttöliittymätestit nauhoittamalla selaimessa tehtyjä toimintoja ja tämän jälkeen muuttamaan nauhoitetut toiminnot Selenium client kirjaston ymmärtämäksi koodiksi eri ohjelmointikielille. Toinen vaihtoehto testien toteuttamiseen on ohjelmoida testit suoraan testausohjelmaan, jolloin pystytään saavuttamaan modulaarisempi ja ylläpidettävämpi käyttöliittymätestien toteutus. Kun käyttöliittymätestien pohjana käytettiin Junit-sovelluskehystä pystyttiin ne myös liittämään rakentamisautomaatioon, jolloin testien suorittaminen ja raportointi saatiin automatisoitua.

Kuormitustestien toteuttamiseen käytettiin Jmeter ohjelmaa. Jmeter ohjelman valintaan vaikuttivat sen monipuoliset ominaisuudet sekä testien luonnin helppous graafisessa käyttöliittymässä. Lisäksi Jmeter-testien suoritus ja raportointi pystyttiin liittämään osaksi Maven-ohjelmalla toteutettua rakentamisautomaatiota.

4.1 JUNIT

Yleistä

Junit sovelluskehys on tarkoitettu Java-ohjelmointikielellä toteutettujen ohjelmien yksikkötestaukseen. Junit-sovelluskehyksen avulla pystytään ajamaan yksikkötestit toisistaan erillään ja yksi kerrallaan. Testien löytämät virheet ja niistä muodostetut raportit pystytään käsittelemään testikohtaisesti. Junit-sovelluskehys on suunniteltu noudattamaan seuraavia sääntöjä,

- Sovelluskehys auttaa kirjoittamaan käytännöllisiä testejä.

- Sovelluskehyksen avulla voidaan kirjoittaa testejä, jotka säilyttävät arvonsa ajan kuluessa.

- Sovelluskehyksen avulla toteutettujen testien koodia pystytään käyttämään uudelleen.

(Tahchiev, Leme, Massol & Gregory 2010, 8-9).

Junit on yksi xUnit perheeseen kuuluvista sovelluskehyksistä. Ensimmäinen versio xUnit perheen sovelluskehyksistä kehitettiin Small-Talk-kielelle, jolloin sen nimeksi annettiin SUnit. Ensimmäisenä suuren yleisön tietoon tuli javalle muunnettu JUnit, jonka suuren suosion takia sovelluskehystä alettiin toteuttamaan myös muille ohjelmointikielille.(Flower,M. xUnit)

xUnit sovelluskehyksiä on toteutettu monille ohjelmointikielille kuten,

- Java (JUnit)
- C++ (CppUnit)
- PHP (PHPUnit)
- .Net (Nunit)
- Python (PyUnit)
- ABAP (ABAP unit)
- Cobol (CobolUnit)
- Fortran (FortranUnit)

Ominaisuudet

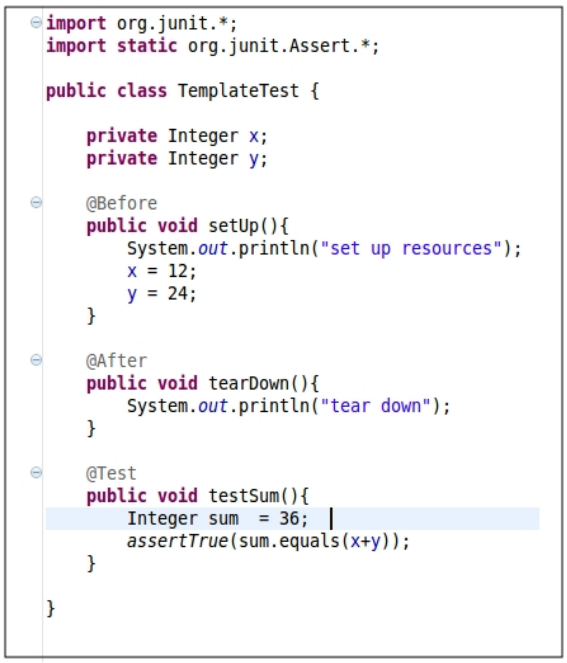
Sovelluskehyksen jokaiselle testille on olemassa oma luokkien lataaja (class loader). Testeissä tarvittavien resurssien alustamiseen ja

testien alku- ja jälkitilan käsittelyssä käytetään sovelluskehyksen tarjoamia annotaatioita (annotations). Monenlaiset tiedon varmentamiseen tarkoitetut metodit auttamat virheiden paikantamisessa. Junit on mahdollista integroida monenlaisiin työkaluihin ja ohjelmointiympäristöihin kuten,

- Ant
- Maven
- Eclipse
- NetBeans
- IntelliJ
- Jbuilder.

Testitapausten toteuttaminen

Junit-sovelluskehyksen avulla toteutettujen testimetodien tulee käyttää aina @Test annotaatiota. Testien tarvitseman alku- ja jälkitilan määrittelyssä voidaan käyttää @Before, @After, @BeforeClass ja @AfterClass annotaatioita. Sovelluskehyksen avulla testien käyttämä tieto on myös mahdollista tallentaa erilliseen XML-pohjaiseen tiedostoon ja käyttää sitä @DataSet annotaation avulla. Testien käyttämän tiedon varmentamisessa voidaan käyttää sovelluskehyksen tarjoamia metodeja. Näitä sovelluskehyksen tarjoamia ominaisuuksia on mahdollista käyttää testien toteuttamisessa (ks. Kuvio 2).



```
import org.junit.*;
import static org.junit.Assert.*;

public class TemplateTest {

    private Integer x;
    private Integer y;

    @Before
    public void setUp(){
        System.out.println("set up resources");
        x = 12;
        y = 24;
    }

    @After
    public void tearDown(){
        System.out.println("tear down");
    }

    @Test
    public void testSum(){
        Integer sum  = 36;
        assertTrue(sum.equals(x+y));
    }

}
```

KUVIO 2: Junit-testi esimerkki

Tulosten raportointi

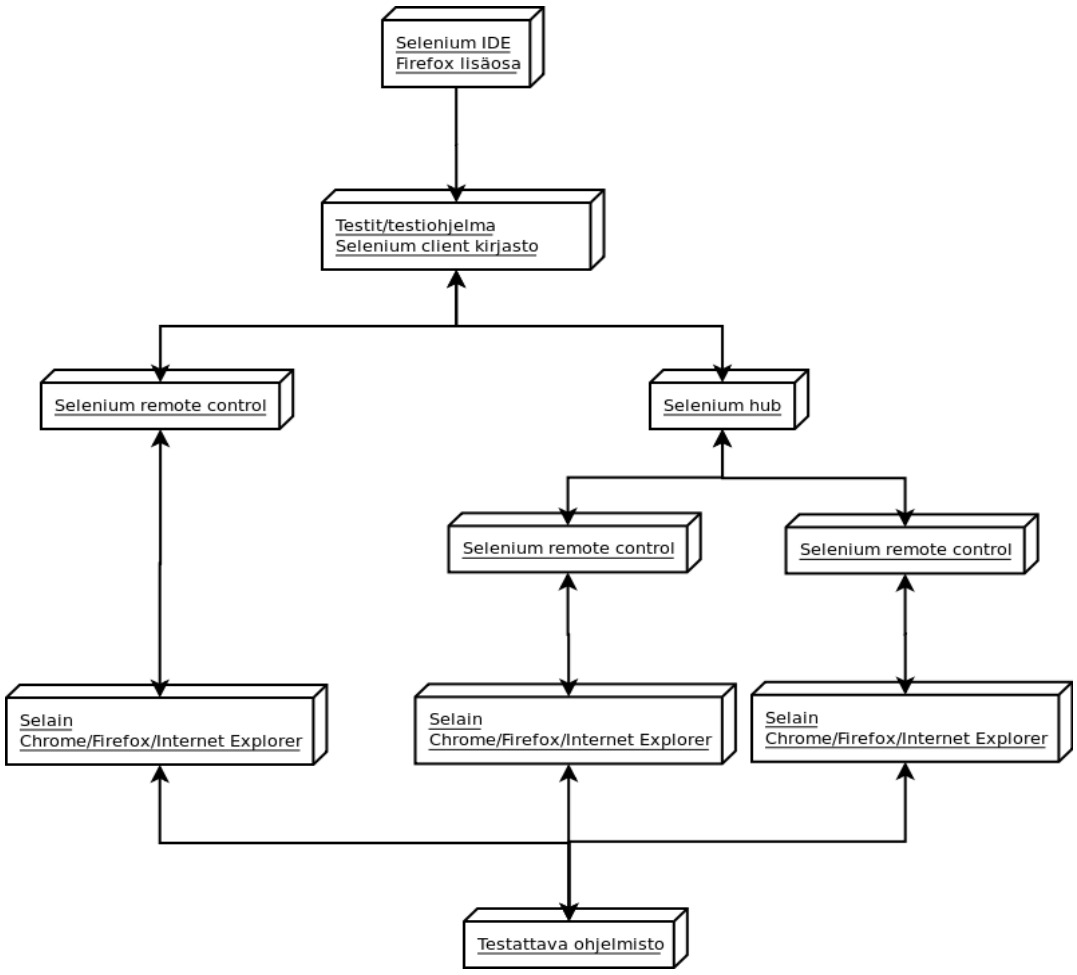
Suorittamisen jälkeiset testiraportit voidaan lukea esimerkiksi integroimalla se ohjelmointiympäristöön tai liittämällä se projektin rakentamisautomaatiosta huolehtivan ohjelman konfiguraatiotiedostoon. Ohjelmointiympäristöissä testien tuloksen voidaan lukea suoraan käyttöliittymästä ajon aikaisesti. Rakentamisautomaatiosta huolehtivien ohjelmien avulla voidaan testituloksista generoida XML- ja HTML-pohjaisia raportteja testien ajon jälkeen.

4.2 SELENIUM

Ominaisuudet

Selenium-testausjärjestelmä on suuri joukko erilaisia ohjelmia ja sovelluskehyksiä, jotka tukevat käyttöliittymätestauksen toteutusta web-pohjaisiin ohjelmistoihin (ks. kuvio 3). Järjestelmän pääasiallise-

na tarkoituksena on varmentaa web-sivuilla näkyvää tietoa ja simuloida web-sivujen käyttöä. Seleniumia voidaan käyttää Windowssissa, Macissa ja Unix-pohjaisissa järjestelmissä. Selenium vaatii käyttöjärjestelmältä Java tuen toimiakseen. Käyttöliittymätestit voidaan suorittaa Firefox, Chrome ja Internet Explorer selaimissa. (Selenium documentation 2010.)

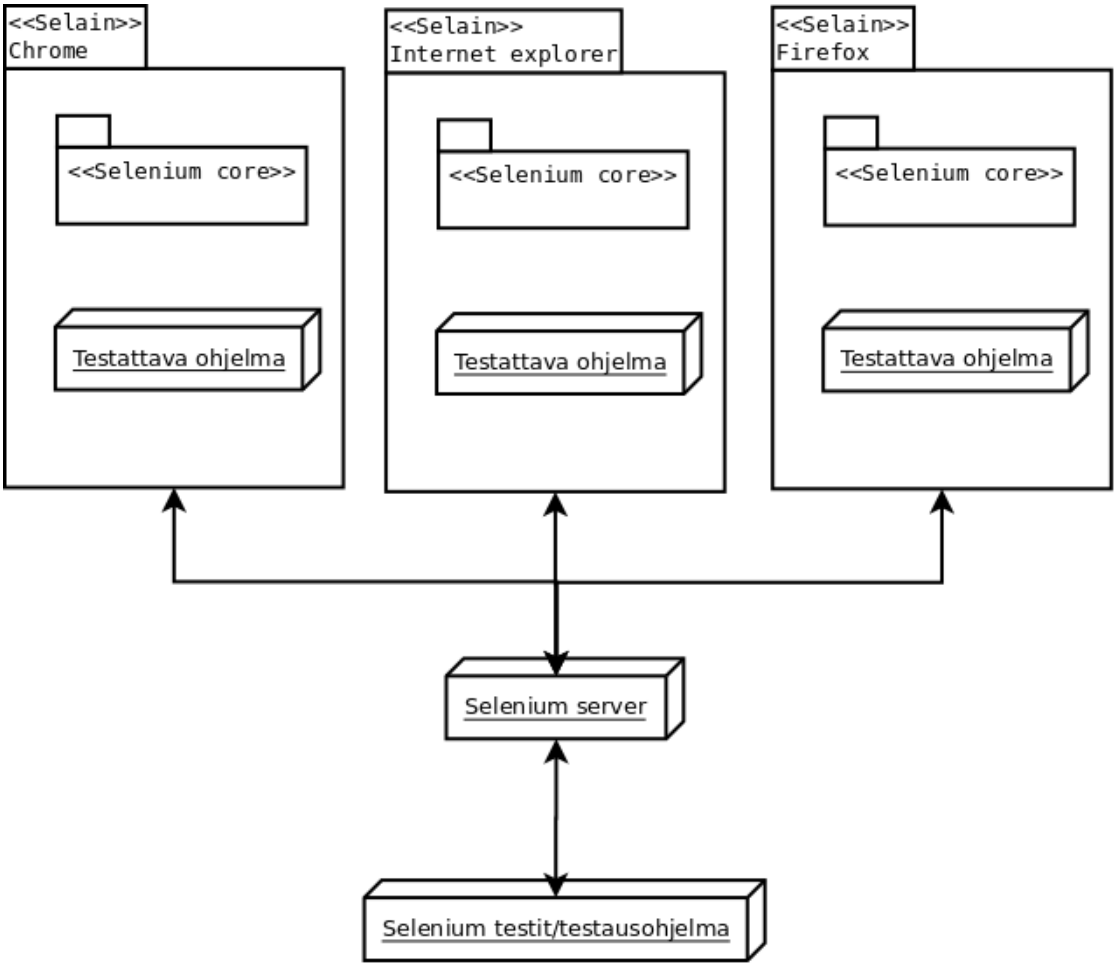


KUVIO 3. Selenium-testausjärjestelmän toiminta.

Selenium server

Selenium server-palvelimenohjelman tarkoitus on toimia siirtämässä tietoa testiohjelman ja selaimen välissä. Testausohjelman toiminta perustuu Selenium server-palvelinohjelmalle lähetettäviin komentoihin, jotka palvelinohjelma lähettää eteenpäin Selenium corelle.

Selenium core on vain joukko javascript-funktioita, jotka käskevät selaimen omaa javascript-tulkkia. Kun Selenium core on käsitellyt käskyt, se palauttaa tulokset Selenium server-palvelinohjelmalle, ja tämä taas lähettää tulokset takaisin testiohjelmalle. Tällä tavalla komennot välittyvät testausohjelmasta selaimeen ja testausohjelma saa takaisin tulokset tehdyistä komennoista (ks. kuvio 4) (Selenium documentation 2010).



KUVIO 4: Selenium server-palvelinohjelman toiminta.

Selenium client

Selenium client-kirjaston avulla toteutetun testiohjelman pohjana kannattaa käyttää jotain yksikkötestausjärjestelmää, kuten Junit tai TestNG. Testitapauksien tekemiseen käytetään Selenium client-

kirjaston tarjoamia funktiota ja luokkia, joiden avulla testiohjelma pystyy lähettämään komentoja Selenium server-palvelinohjelmalle. Selenium client-kirjasto tarjoaa todella monipuoliset mahdollisuudet varmentamaan ja käsittelemään web-sivustolla olevaa tietoa. Client-kirjaston avulla pystytään muokkaamaan ja lukemaan lomake elementtien sisällä olevaa tietoa. Sen avulla pystytään etsimään mitä tahansa sivustolla näkyvää tietoa esimerkiksi Xpath-lauseiden avulla ja myös web-sivujen lataukseen kestoon on helppo tehdä aikarajoituksia. Client-kirjaston funktiot lähettävät komennot Selenium server-palvelinohjelmalle, joka lähettää komennon eteen-päin selaimelle. Kun komennot ovat käsitelty selaimen puolella, palauttaa Selenium server-palvelinohjelma tiedon komennon suorit-tamisen onnistumisesta takaisin testiohjelmalle (ks. kuvio 4). (Selenium documentation 2010.)

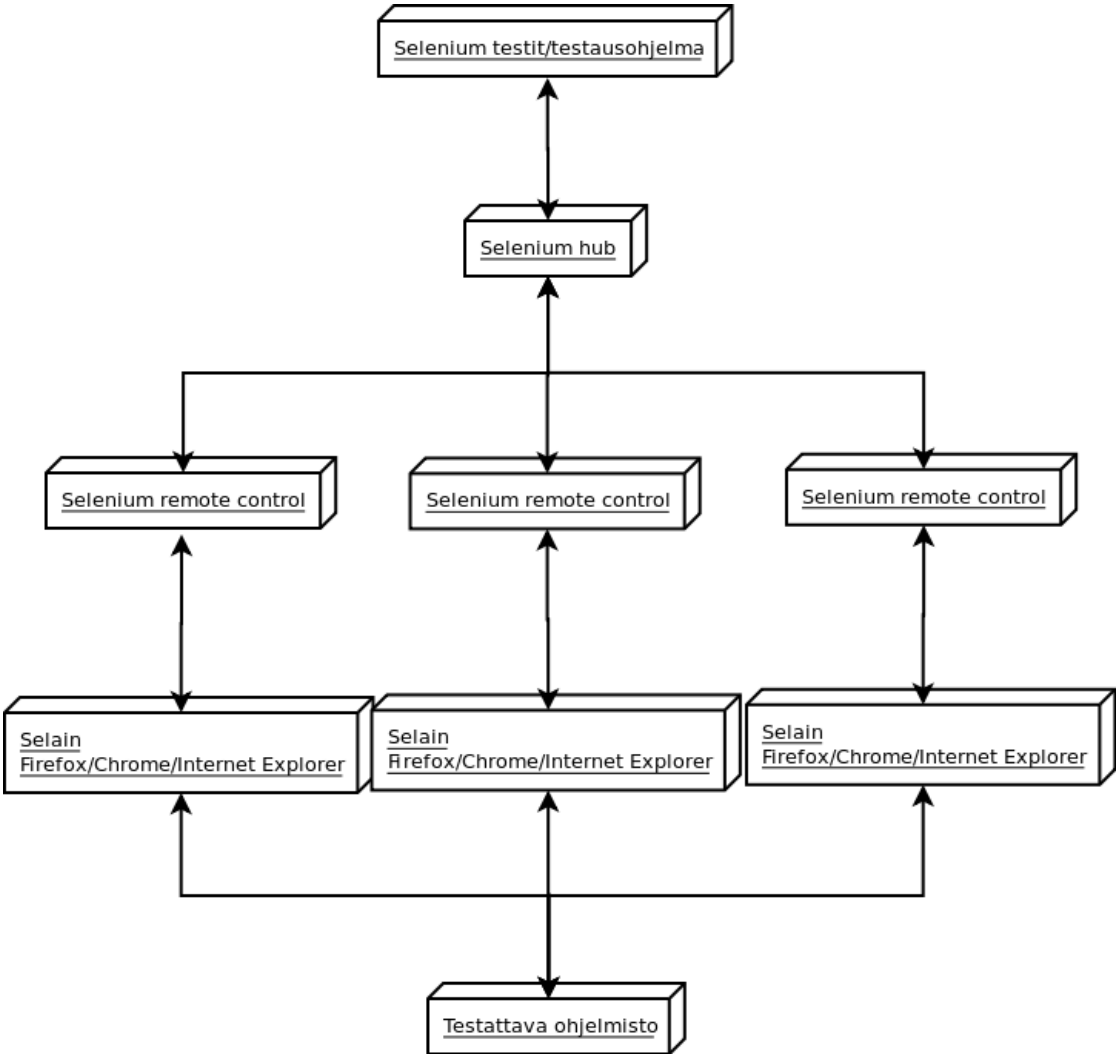
Selenium IDE

Selenium IDE on Firefox selaimeen asennettava lisäosa, joka auttaa paljon testausautomaation toteutuksessa. Selenium IDE:llä on mahdollista nauhoittaa manuaalisesti tehtyjä testejä ja ajaa niitä uudestaan. Ohjelmassa luodut testit voidaan muuttaa Selenium client-kirjastoa käyttäväksi ohjelma koodiksi, jolloin nauhoittamalla tehdyt testit saadaan osaksi testausohjelmaa. Selenium IDE:n avulla pystytään etsitään web-sivuilla olevien HTML-elementtien Xpath lauseita, joka helpottaa itse kirjoitettujen testien luomista. (Selenium documentation 2010, Selenium IDE.)

Selenium Grid

Selenium Grid:n avulla pystytään rakentamaan testaus verkkoja,

joissa pystytään suorittamaan käyttöliittymätestausta useilla eri koneilla yhtäaikaisesti. Tällä tavalla pitkään kestävien testiajojen aikaa pystytään pienentämään ja ajamaan yhtäaikaisesti testejä monille eri selaimille (ks. kuvio 5). (Selenium grid, how it works, 2010.)



KUVIO 5: Selenium grid-ohjelman toiminta.

WebDriver

WebDriver on Seleniumin kehitteillä olevan version mukana tuleva sovelluskehys. WebDriverin suurimpana erona on Selenium client-kirjastoon verrattuna on, että se ei ole enää riippuvainen selaimen sisäisestä javascript-tulkista ja sillä pystytään ajamaan käyttöliitty-

mä testejä ilman selainta. Tällöin WebDriver lataa web-sivun lähdekoodin ja tulkitsee tämän perusteella sivulla näkyvät HTML-elementit. Tämä nopeuttaa testien suoritusta huomattavasti, mutta ei anna niin tarkkoja tuloksia testien onnistumisesta. WebDriver tulee myös tarjoamaan paremmin suunnitellun ohjelmointirajapinnan, jossa esimerkiksi web-sivujen HTML-elementtejä käsitellään olioina. (Selenium documentation 2010, Web Driver.)

4.3 JMETER

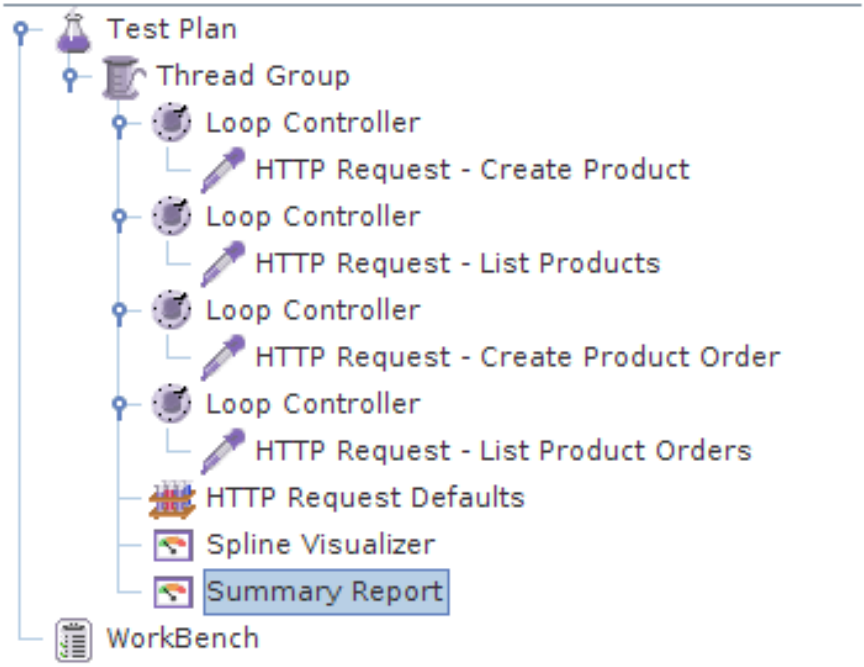
Vaatimukset ja ominaisuudet

JMeter on testausohjelma, jolla on helppo toteuttaa kestävyys- ja kuormitustestejä erilaisille rajapinnoille. JMeter toimii Windowsissa, Macissa ja Unix-pohjaisissa käyttöjärjestelmissä. JMeter tarvitsee Java tuen käyttöjärjestelmältä. Tällä hetkellä JMeter:llä pystytään ajamaan testejä esimerkiksi web-sivustoille(HTTP, HTTPS), SOAP-rajapinnoille, tietokantoihin (JDBC), LDAP hakemistopalveluihin ja sähköpostipalvelimille (SMTP). JMeter ohjelmiston tarkoituksena on simuloida useiden käyttäjien yhtäaikaista käyttöä testattavalle rajapinnalle. JMeterin avulla pystytään luomaan testaus verkkoja, missä useat eri koneet ajavat testejä samalle testattavalle rajapinnalla. Tällä tavalla voidaan simuloida todella suuria käyttäjä määriä. (Jmeter user manual)

Testaussuunnitelman toteutus

JMeter testauksen toteutus aloitetaan tekemällä testaussuunnitelma (test plan). Testaussuunnitelma sisältää kaikki Jmeter ohjelman testauselementit, joita testaamisen suorittamiseen tarvitaan. JMeterin testaussuunnitelma on hierarkkinen puurakenne (ks. kuvio 6).

Testejä ajettaessa ohjelma alkaa käymään puurakennetta lävitse ylhäältä alaspäin. Testaussuunnitelman tulee sisältää ainakin yksi Säieryhmä, logiikkaohjain, näytteidenottaja ja näytteidenkuuntelija. Kun kaikki tarvittavat elementit ovat lisätty testaussuunnitelmaan voidaan testit ajaa, joko paikallisella koneella tai testaus verkossa olevalla koneella.



KUVIO 6: JMeter testaussuunnitelma

Testaussuunnitelman elementit

Säieryhmät

Säieryhmät (Thread groups) ovat testaussuunnitelman alimmaisia elementtejä ja ne tulee lisätä puu rakenteen alkuun. Kaikkien logiikkaohjainten ja näytteidenottajien tulee olla säieryhmien alla. Säieryhmillä voidaan määrittää kuinka monessa säikeessä testaussuunnitelma ajetaan ja uusien säikeiden käynnistymiseen kuluva aika, sekä säieryhmän ajo kerrat.

Logiikkaohjaimet

Logiikkaohjaimet (Logic controllers) ohjaavat näytteidenottajia. Logiikkaohjaimilla voidaan vaikuttaa siihen kuinka useasti näytteitä otetaan yhden ajon aikana.

Näytteidenottajat

Näytteidenottajat (Samplers) ottavat näytteitä erilaisista rajapinnoista. Jmeter tällä hetkellä tukee ainakin HTTP-, HTTPS-, FTP-, SMTP-, JDBC- ja SOAP-rajapintojen näytteidenottajia. Näytteidenottajat palauttavat esimerkiksi tietoa siitä kuinka nopeasti testattava rajapinta on vastannut pyyntöön ja onko pyyntöä käsiteltäessä aiheutunut virhetilanne.

Näytteidenkuuntelijat

Näytteidenkuuntelijat (Listeners) ottavat vastaan näytteidenottajilta tulevia näytteitä ja tekevät niistä luettavassa muodossa olevia raportteja. Näytteidenkuuntelijoilla pystytään esimerkiksi listaamaan taulukkoon testien tulokset tai piirtämään kuvaajia suorituskykytesteistä.

Ajastimet

Ajastimilla (Timers) pystytään viivyttämään testejä. Testien ajossa kannatta käyttää ajastimia, koska JMeter saattaa joissain tilanteissa lähettää liian monta pyyntöä palvelimella liian lyhyessä ajassa ja tämän takia palvelin ei pysty vastaamaan kaikkiin lähetettyihin pyyntöihin.

Varmentajat

Varmentajilla (Assertions) pystytään tarkistamaan, että lähetetty pyyntö testattavalle rajapinnalle palauttaa oikean vastauksen.

Varmentajilla on mahdollista testata esimerkiksi, että HTTP-rajapinnalta tullut vastaus sisältää tiettyjä elementtejä.

Konfiguraatioelementit

Konfiguraatioelementeillä (Configuration elements) pystytään määrittämään testaussuunnitelmalle vakio arvoja joita pystytään käyttämään kaikissa testeissä. Konfiguraatioelementtien avulla pystytään esimerkiksi tallentamaan web-sivuston käyttämän evästeiden arvoja tai asettamaan vakio arvoja LDAP tai HTTP pyyntöjen lähettämiseen.

Alku- ja jälkitilakäsittelijät

Alku- ja jälkitilakäsittelijöillä (Pre-prosessors & Post-prosessors) pystytään muokkaamaan testien käyttämiä asetuksia ennen tai jälkeen suorittamisen. Nämä ovat käytännöllisiä jos esimerkiksi halutaan ajaa yksi testi muilla kuin vakioasetuksilla tai ottaa talteen tietoa aikaisemmin ajetun testin palauttamista arvoista.

4.4 MAVEN

Yleistä

Maven on Java ohjelmointikielellä toteutettujen projektien hallitsemiseen tarkoitettu ohjelma. Maven ohjelman toiminta perustuu projekti oliomalliin (project object model, POM). Ohjelman avulla pystytään hallitsemaan projektin kääntämistä, testaamista, käyttöönottoa, raportointia ja dokumentointia. POM mallin on tarkoitus tarjota rakentamiskonfiguraatio (build configuration) yksittäiselle projektille. (POM reference)

POM-malli

Maven ohjelman käyttämä POM-malli liitetään projektiin XML-konfiguraatiotiedostolla. Konfiguraatiotiedosto sisältää kaikki ohjelman tarpeelliset tiedot. Konfiguraatiotiedosto on jaettu osiin sen perusteella minkälaista tietoa kyseinen osio sisältää. Konfiguraatiotiedostosta löytyy omat osiot projektin tietojen tallentamiseen, rakentamisprosessin hallitsemiseen, raportointiin ja ympäristön asetusten määrittämiseen (ks. kuvio 7). (POM reference)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <!-- The Basics -->
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <version>...</version>
  <packaging>...</packaging>
  <dependencies>...</dependencies>
  <parent>...</parent>
  <dependencyManagement>...</dependencyManagement>
  <modules>...</modules>
  <properties>...</properties>

  <!-- Build Settings -->
  <build>...</build>
  <reporting>...</reporting>

  <!-- More Project Information -->
  <name>...</name>
  <description>...</description>
  <url>...</url>
  <inceptionYear>...</inceptionYear>
  <licenses>...</licenses>
  <organization>...</organization>
  <developers>...</developers>
  <contributors>...</contributors>

  <!-- Environment Settings -->
  <issueManagement>...</issueManagement>
  <ciManagement>...</ciManagement>
  <mailingLists>...</mailingLists>
  <scm>...</scm>
  <prerequisites>...</prerequisites>
  <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
  <distributionManagement>...</distributionManagement>
  <profiles>...</profiles>
</project>
```

KUVIO 7: Konfiguraatiotiedoston rakenne.

POM-mallissa ohjelman rakentamisprosessi (building process) jaetaan erillisiin vaiheisiin, joissa jokaisessa pystytään suorittamaan kyseisen vaiheen tarvitsemat operaatiot. Tällä tavalla voidaan esimerkiksi määrittää missä vaiheessa rakentamisprosessia ajetaan eri tyyppiset testit.

Ohjelman rakennusprosessiin POM-mallissa kuuluvat seuraavat vaiheet:

1. ohjelman validointi
2. ohjelman lähdekoodien kääntäminen
3. yksikkötestien suorittaminen
4. paketointi
5. integraatiotestien suorittaminen
6. verifiointi
7. asentaminen
8. käyttöönotto.

Tietovarastot

Maven käyttää ohjelman kirjastojen riippuvaisuuksien hallintaan tietovarastoja (repositories). Tietovarastojen avulla Java-projekteihin ei enää tarvitse käsin etsiä tarvittavia kirjastoja, vaan Maven etsii automaattisesti olemassa olevia kirjastoja tietovarastoista. Maven ohjelman sisältää itsessään paikallisen tietovaraston, josta ohjelma ensisijaisesti etsii projektin tarvitsemia kirjastoja, mutta jos kirjastoa ei löydy paikallisesta tietovarastosta, yrittää Maven etsiä kirjastoa ulkopuolisesta tietovarastosta.(POM reference.)

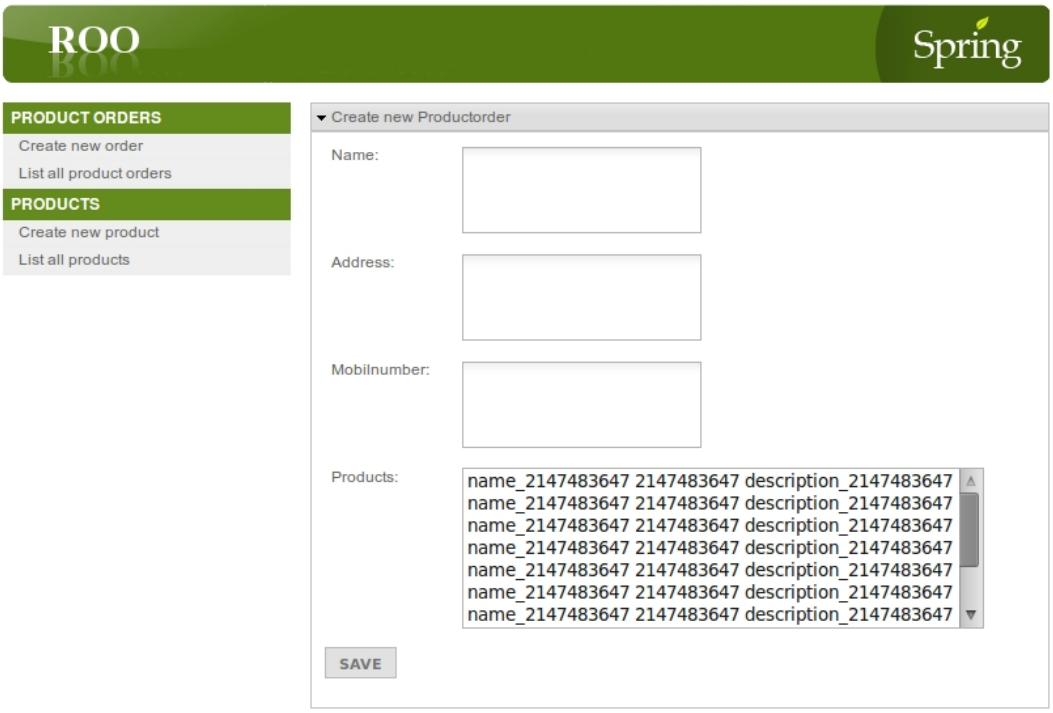
5. TESTAUSAUTOMAATION TOTEUTTAMINEN KÄYTÄNNÖSSÄ

5.1 Yleistä

Testausautomaation toteutusta varten tehtiin verkkopohjainen prototyyppiohjelma, jonka avulla varmistettiin erilaisten testaus tapojen toteuttamista. Ohjelmaan toteutettiin integraatio-, käyttöliittymä- ja kuormitustestit. Testien toteuttamiseen käytettiin Selenium testausjärjestelmää, Junit sovelluskehystä ja Jmeter ohjelmaa (ks. luvut 4-6). Testitien ajo liitettiin testattavan ohjelman rakentamisautomaatiosta huolehtivan Maven ohjelman konfiguraatiotiedostoon, jolloin kaikki testit pystyttiin ajamaan aina kun ohjelmasta käännettiin uusi versio.

5.2 Testattava ohjelma

Testattava verkkopohjainen ohjelma toteutettiin Spring Roo työkalun avulla, jolla generoitiin Spring-sovelluskehystä käyttävä ohjelma. Ohjelma sisälsi selainpohjaisen käyttöliittymän (ks. kuvio 8) ja tietojen tallentamiseen käytetyn MySQL tietokannan. Ohjelmassa käytettiin Hibernate-sovelluskehystä, jolla pystyttiin siirtämään ohjelmassa käytettävien luokkien tietoa suoraan tietokannan tauluihin.

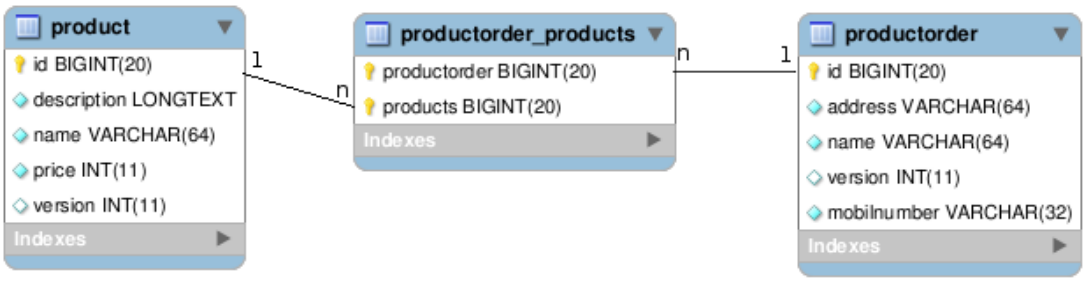


KUVIO 8: Testattavan ohjelman käyttöliittymä

Testattavan ohjelman ominaisuuksiin kuuluivat,

- uusien tuotteiden luonti
- olemassa olevien tuotteiden muokkaus
- tuotteiden poistaminen
- tuotetilausten teko
- olemassa olevien tuotetilausten muokkaaminen
- tuotetilausten poistaminen.

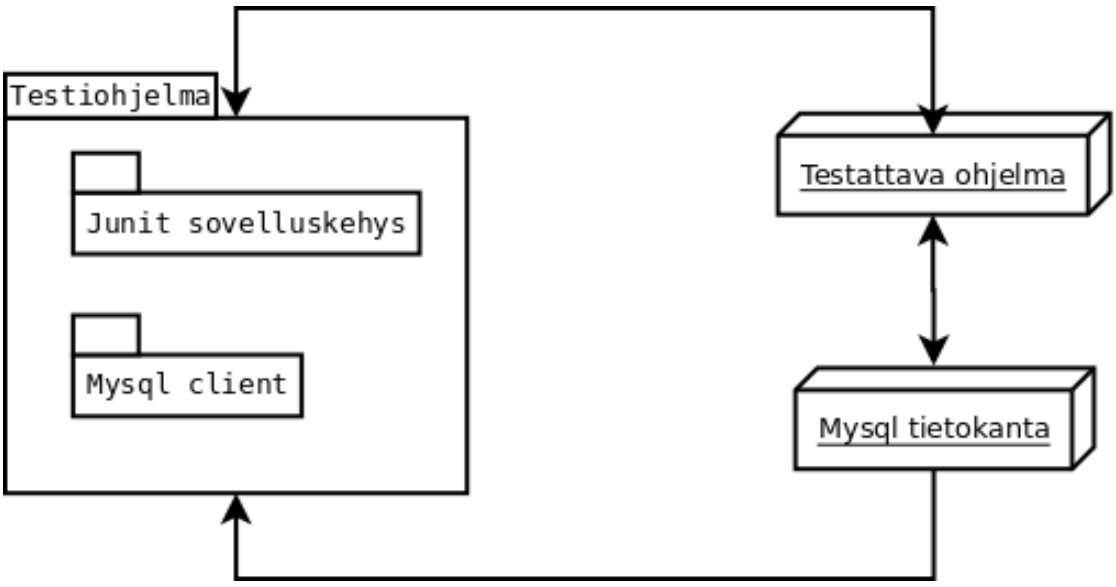
Testattava ohjelma pidettiin mahdollisimman yksinkertaisena, koska työn päätavoitteena oli tutkia vain testausautomaation toteuttamista. Ohjelman tietokanta sisälsi kolme taulua, joihin ohjelman tarvitsemat tiedot tallennettiin (ks. kuvio 9).



KUVIO 9: Tietokannan kuvaus.

5.4 Integraatiotestaus

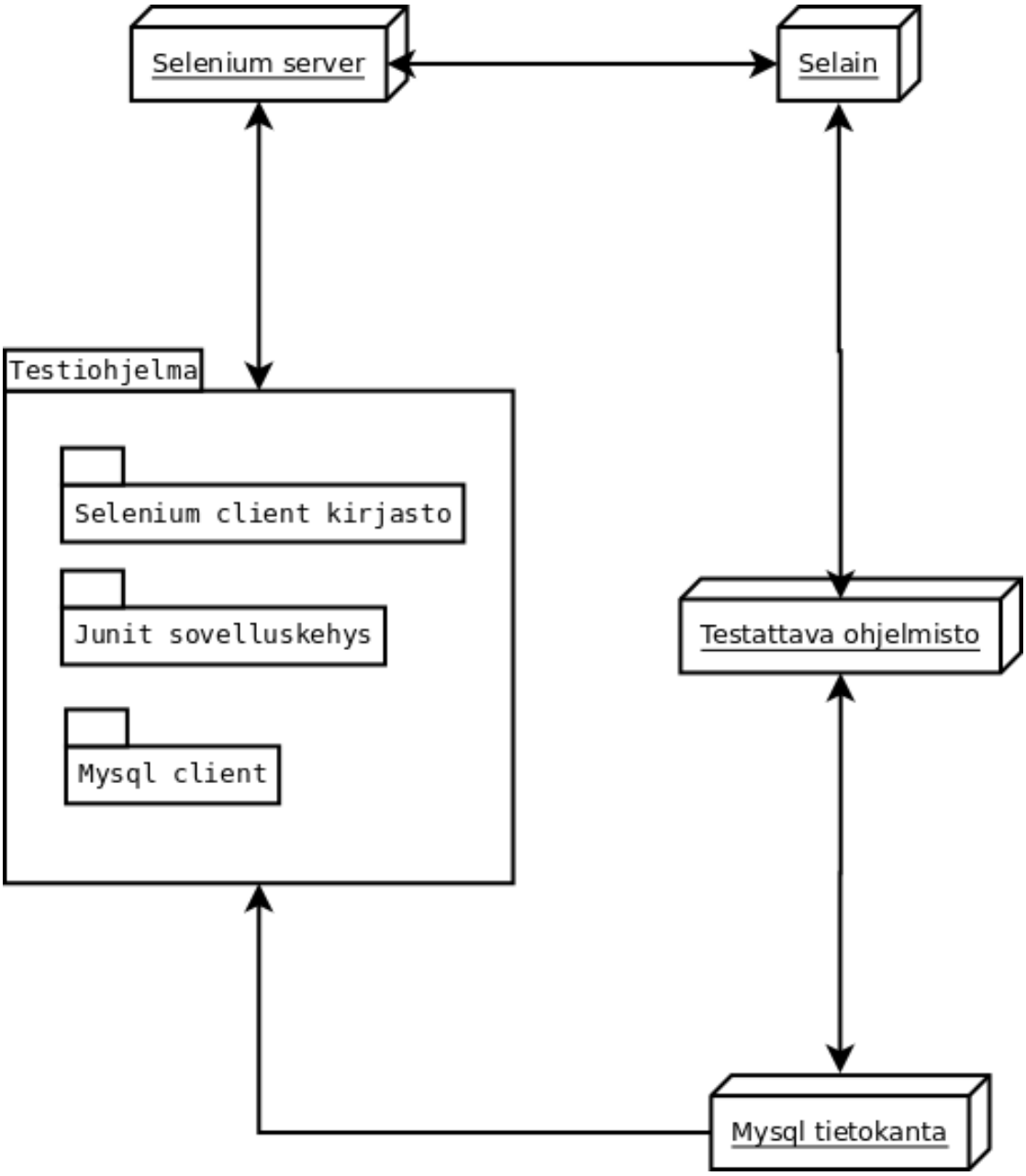
Integraatiotestien toteuttamiseen käytettiin Junit-sovelluskehystä. Integraatiotestien tarkoituksena oli tarkastaa ohjelman entiteetti-luokkien ja tietokannan välistä toimintaa. Hibernate-sovelluskehystä käyttävät ohjelmat koostuvat entiteettiluokista (entity classes), joiden sisältämää tietoa pystytään siirtämään suoraan tietokannan tauluihin. Integraatiotesteillä tarkistettiin entiteettiluokkien ja tietokannan välistä toimintaa (ks. kuvio 10).



KUVIO 10: Integraatiotestien toiminta

5.5 Käyttöliittymätestaus

Käyttöliittymätestit toteutettiin Selenium testausjärjestelmän ja Junit-sovelluskehyksen avulla. Käyttöliittymätestit toteutettiin kaikille järjestelmän ominaisuuksille. Ominaisuuksien toimiminen testattiin lisäämällä ja poistamalla tuotteita ja tuotetilauksia ohjelman käyttöliittymästä. Tämän jälkeen tarkistettiin tietokannasta että kyseiset operaatiot olivat suoritettu onnistuneesti (ks. Kuvio 11).



KUVIO 11: Käyttöliittymä testien toiminta

5.6 Kuormitustestaus

Testattavaan ohjelmaan luotiin kuormitus testit Jmeter ohjelmalla. Jmeter ohjelmalla toteutettiin testit kaikille mahdollisille testattavan ohjelman verkkosivuille. Testien luonti aloitettiin luomalla uusi testaussuunnitelma ja säieryhmä Jmeter ohjelman käyttöliittymästä. Jokaiselle testattavan ohjelman sisältämälle verkkosivulle, luotiin oma logiikkaohjain ja näytteidenottaja. Lisäksi säieryhmän juureen lisättiin HTTP-konfiguraatio elementti, jolla pystyttiin jokaiselle näytteenottajalle määrittämään HTTP-pyyntöjen lähetysosoite.

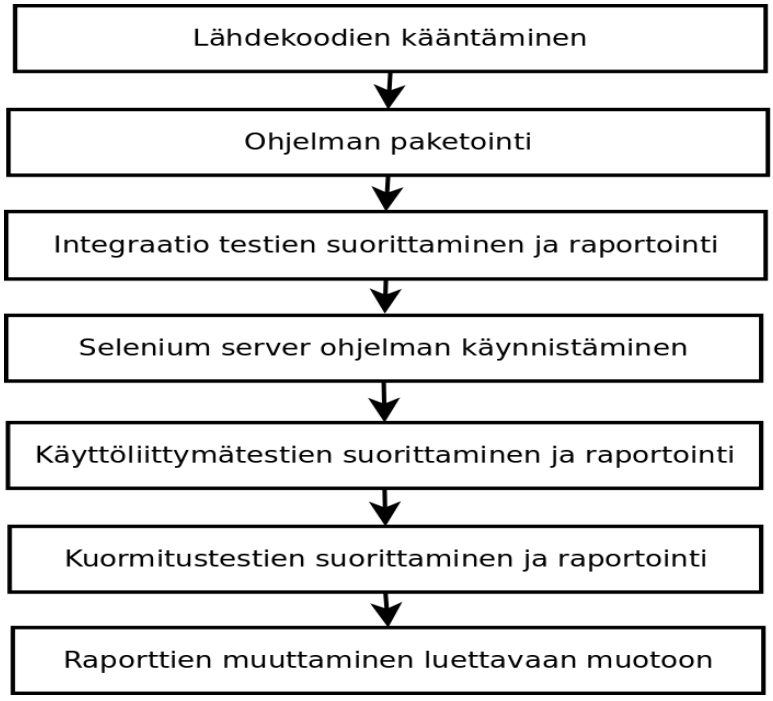
Testeissä kasvatettiin säieryhmän avulla simuloitua käyttäjämäärää ja tarkkailtiin kuinka kauan palvelimelta kestää vastata lähetettyihin pyyntöihin tai kuormittuuko palvelin liikaa, jolloin pyyntöihin ei saada vastausta. Työssä ei kuitenkaan pystytä antamaan realistisia testituloksia, koska testiympäristö ei vastannut todellista web-ohjelmien ajoympäristöä.

5.7 Testien liittäminen osaksi rakentamisautomaatiota

Testattavassa ohjelmassa käytettiin Maven ohjelmaa hoitamaan projektin rakentamisprosessia. Testien liittämiseksi Maven ohjelmaan jouduttiin asentamaan lisäosat testien suorittamista ja raportointia varten.

Junit-sovelluskehyksellä toteutettujen testien liittämiseksi rakentamisautomaatioon tarvittiin lisäosat testien suorittamiselle ja raportoimiselle. Maven ohjelmaan asennettiin Surefire-lisäosa, jonka avulla pystyttiin ajamaan ja luomaan XML-pohjaisia raportteja Junit sovelluskehyksen avulla toteutetuista testeistä. Surefire-lisäosan avulla pystyttiin lohkomaan erilaisten testien suorittamista ohjelman rakentamisen eri vaiheisiin. Tällä tavalla pystyttiin suorittamaan integraatio- ja käyttöliittymätestit ohjelman rakentamisen eri vaiheissa.

Integraatio- ja käyttöliittymä testien raportoimista varten asennettiin Surefire-report-lisäosa. Lisäosan avulla pystyttiin muodostamaan erilliset HTML-pohjaiset raportit aikaisemmin luoduista XML-pohjaisista integraatio- ja käyttöliittymätestien raporteista. Kuormitustestien liittämiseksi osaksi ohjelman koostamisprosessia Maven ohjelmaan piti asentaa kaksi lisäosaa. Kuormitustestien suorittamista ja raportointia varten Maven ohjelmaan asennettiin jmeter lisäosa. Lisäosa tarvitsi Jmeter ohjelmalla toteutetun testaus suunnitelman jmx tiedoston kuormitustestien ajamista varten. Lisäosan avulla pystyttiin generoimaan XML-pohjaisia raportteja testien suorituksesta. Maven ohjelmaan piti lisäksi asentaa XML-lisäosa, jolla testiraportit pystyttiin muuttamaan HTML-pohjaisiksi ja helpommin luettaviksi. Kun kaikki testit saatiin liitettyä Maven ohjelman konfiguraatiotiedostoon, pystyttiin yhdellä komennolla kääntämään ohjelmasta uusi versio, sekä ajamaan ja raportoimaan kaikki testit oikeassa järjestyksessä (ks. kuvio 12).



KUVIO 12: Testien suorittaminen Maven ohjelman avulla

6. TULOKSET

Työn tuloksena saatiin todennettua työssä käytettyjen ohjelmien, sovelluskehysten ja työkalujen soveltuvuus testausautomaation toteuttamiseksi web-sovelluksille. Ohjelmien toiminta testattiin tekemällä prototyyppiohjelma, johon toteutettiin integraatio-, käyttöliittymä- ja kuormitustestit. Lisäksi työssä annettiin ohjeita siitä kuinka testausautomaation eri vaiheet tulisi toteuttaa ja kuinka testausautomaatio tulisi liittää osaksi jatkuvaa integraatiojärjestelmään.

Työssä ei kuitenkaan pystytä toteamaan testausautomaatiolla saavutettavia hyötyjä, koska niiden todistamiseksi tarvittaisiin pitkäaikaisia tuloksia niin manuaalisen kuin automaattisenkin testauksen toteuttamisesta ja käyttämisestä. Kuitenkin kuten (Fewster & Graham 1999, 205) toteaa, että testausautomaation toteuttaminen projekteihin, joissa testejä ajetaan useaan kertaan, tulee halvemmaksi kuin testien toteuttaminen manuaalisesti. Testausautomaation avulla voidaan vähentää ohjelmaan liittyviä riskejä, pystytään ajamaan testit nopeasti ja säästämään rahaa pidemmällä aikavälillä (mts. 346-347). Lisäksi työssä käytetyt ohjelmat ovat täysin ilmaiseksi saatavilla, joten automaation toteuttamisessa syntyvät kustannukset tulevat ainoastaan testausautomaation toteuttamisesta ja testien suorittamisesta.

7. YHTEENVETO

Opinnäytetyöprojekti oli todella mielenkiintoinen ja pääsin sen aikana oppimaan paljon uusia asioita ohjelmistotestauksesta. Ohjelmistotestaaminen on aiheena niin laaja käsite, että vaikein asia

projektissa olikin rajata työn aihealue tarpeeksi suppeaksi, jotta saisi käsiteltyä tiettyä kokonaisuutta tarpeeksi tarkasti.

Työn tekemisen aloitin etsimällä aiheeseen liittyvää kirjallisuutta ja internetistä löytyviä artikkeleja. Kun olin löytänyt tarpeeksi materiaalia kirjallisen osuuden pohjaksi, aloitin käytännön työn toteuttamalla prototyyppiohjelman testausautomaatiota varten. Tämän jälkeen toteutin integraatio, käyttöliittymä- ja kuormitustestit. Viimeisenä liitin kaikki testit projektin rakentamisautomaatioon, jolloin kaikki testit pystyttiin ajamaan ja raportoimaan automaattisesti oikeassa järjestyksessä.

Mielestäni tässä työssä annetaan kattava kuvaus tarvittavien ohjelmien, työkalujen ja sovelluskehysten käyttämisestä testausautomaation toteuttamiseksi web-sovelluksille. Lisäksi työn sisältämässä testausautomaation toteutuksessa on otettu huomioon kustannustehokkuus ja kaikki työssä käytetyt ohjelmat ovat saatavilla täysin ilmaiseksi.

LÄHTEET

Apache JMeter user's manual. Viitattu 01.09.2010. http://jakarta.apache.org/jmeter/usermanual/index.html.

Fewster, M & Graham, D. 1999. Software test automation, effective use of test execution tools.

Fowler, M. 2006. Continuous Integration. Viitattu 05.09.2010. http://martinfowler.com/articles/continuousIntegration.html

Fowler, M. xUnit. Viitattu 15.09.2010. http://www.martinfowler.com/bliki/Xunit.html

Kilosoft yritysinfo 2010. Viitattu 03.10.2010. http://www.kilosoft.fi/yritysinfo.html

Loveland, S., Miller, G., Prewitt, R. & Shannon, M. 2004. Software testing techniques: finding the defects that matter.

POM reference 2010. Viitattu 13.09.2010. http://maven.apache.org/pom.html

Smart, J. 2009. Where To Now with Build Automation?. Viitattu 05.09.2010. http://www.infoq.com/articles/build-automation-ci-atlassian

Selenium documentation, 17.10.2010. Viitattu 01.09.2010. http://seleniumhq.org/docs/

Selenium Grid, How it works. Viitattu 01.09.2010. http://selenium-grid.seleniumhq.org/how_it_works.html

Tahchiev, P., Leme, F., Massol, V. & Gregory, G. 2010. Junit in action, second edition.