KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittely

Jarmo Vetoniemi

JAVA-SUORITUSYMPÄRISTÖN NOPEAN KEHITYKSEN SOVELLUSKEHYKSET

Opinnäytetyö
Toukokuu 2015

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Toukokuu 2015** **Tietojenkäsittely** Karjalankatu 3 80200 Joensuu

Tekijä(t)
Jarmo Vetoniemi

Nimeke
Java-suoritusympäristön nopean kehityksen ohjelmistokehykset

Toimeksiantaja
Elbit Oy

Tiivistelmä

Nopean kehityksen sovelluskehykset herättävät kiinnostusta, koska taloudellisesti tiukkana aikana ja jatkuvasti muuttuvassa maailmassa halutaan tehostaa sovelluskehitystä ja mahdollistaa eri ideoiden nopea kokeileminen käytännössä. Opinnäytetyössä perehdyttiin Java-suoritusympäristön päällä toimiviin nopean kehityksen sovelluskehyksiin. Tavoitteena oli selvittää, miten niistä voi valita sopivimman ja parhaimman käyttöönsä sekä mitä hyötyjä niiden käytöstä on.

Opinnäytetyössä perehdyttiin ensin teoreettisesti eri sovelluskehyksiin. Sovelluskehyksistä valittiin kaksi kokeiltavaksi, ja sopivammaksi osoittautuneella toteutettiin sovellus. Samalla selvitettiin, saavutettiinko sovelluskehyksen käytöllä tehokkuutta ja kustannushyötyjä. Lisäksi verrattiin nopean kehityksen ohjelmistokehysten käyttöä perinteiseen Java EE -sovelluskehitykseen.

Sovelluskehystä valitettaessa päädyttiin käyttämään 20:tä eri kriteeriä. Niiden perusteella valittiin Grails-sovelluskehys. Sitä käytettäessä tuli esille monia etuja, mutta myös haittoja ilmeni.

Kieli	Sivuja 39
suomi	

Asiasanat

Grails, Java, Java EE, JVM, Play, sovelluskehys

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
May 2015
Business Information Technology

Karjalankatu 3
80200 Joensuu
FINLAND

Author (s)
Jarmo Vetoniemi

Title
High Velocity Frameworks for Java Virtual Machine

Commissioned by
Elbit Oy

Abstract

High velocity frameworks or frameworks for high-velocity development arouse interest because during the economically tough times and in a rapidly changing world enterprises wish to accelerate application development and test if an idea will be viable for them. This thesis focuses on the Java-based high velocity application frameworks executed upon Java Virtual Machine. The goal was to find out how to choose the most suitable framework and what benefits are acquired by using a high velocity framework.

At first different frameworks were examined theoretically. Then two application frameworks were chosen for practical trials and the better one was used to implement an application. This way it was possible to see which application framework was the best option and if efficiency and cost benefits were achieved. Furthermore the use of rapid development frameworks were compared with traditional Java EE application development.

20 different criteria were used when choosing the best application framework and the Grails framework was chosen. Using it revealed many advantages, but also disadvantages were discovered.

Language
Finnish

Pages 39

Keywords
framework, Grails, Java, Java EE, JVM, Play

Sisältö

Lyhenteet

CRUD	Lyhenteellä tarkoitetaan luonti-, luku-, päivitys- ja poistotoimintoja (Create, Read, Update, Delete) (Wikipedia 2015a).
JVM	Java Virtual Machine on sovellusalusta, jonka päällä Java-tavukoodia ajetaan (Wikipedia 2015b).
MVC	Malli-näkymä-kontrolli-sovellusmalli (Model View Controller) (Wikipedia 2015c).
MVP	Minimum viable product. Tuote, jossa on minimimäärä ominaisuuksia, joiden avulla näkee, onko tuotteella käyttöä (Wikipedia 2015d).
REST	REST (Representational State Transfer) on HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen (Wikipedia 2015e).
SOAP	SOAP (Simple Object Access Protocol) on tietoliikenneprotokolla, jonka pääasiallisena tehtävänä on mahdollistaa proseduurien etäkutsu. Sen erityispiirteinä on pohjautuminen XML-kieleen ja toimiminen useiden eri protokollien yli. Sitä käytetään pääasiassa HTTP-protokollan yli. (Wikipedia 2015f).
Spock	Spock on testaus- ja määrittelykehikko Java- ja Groovy-sovelluksille. Se poikkeaa muista vastaavista ilmaisukykyisen määrittelykielensä takia (Spock 2015).
WSDL	Web Service Description Language (WSDL) on W3C:n määrittämä XML-perustainen kieli, jolla kuvataan tietoverkossa tarjolla oleva web-teknologioihin perustuva palvelu, eli Web Service (Wikipedia 2015g).

1 Johdanto

Nopean kehityksen sovelluskehykset (high velocity frameworks tai framework for high-velocity development) ovat herättäneet paljon kiinnostusta viime aikoina. Taloudellisesti tiukkana aikana ja nopeasti muuttuvassa maailmassa halutaan nopeuttaa sovelluskehitystä ja mahdollistaa eri ideoiden nopea kokeileminen käytännössä. Sovelluskehys on ohjelmistotuote, joka muodostaa rungon sen päälle rakennettavalle ohjelmistolle. Se helpottaa tavallisempien toimintojen toteutusta, mm. tietokannankäsittelyä, istunnon hallintaa ja tietoturvaa. Nopean kehityksen sovelluskehykset lupaavat tehostaa sovelluskehitystä vielä enemmän mm. tehokkaamman ohjelmointikielen ja sovelluskehittimen avulla (Smith & Ledbrook 2014, 4).

Opinnäytetyössä perehdytään Java-suoritusympäristön (Java Virtual Machine eli JVM) päällä toimiviin nopean kehityksen sovelluskehyksiin. Esimerkkejä tällaisista kehyksistä ovat Grails, Play ja Vaadin. Tutkimuksessa vertaillaan eri sovelluskehyksiä keskenään. Myös niitä verrataan perinteiseen Java EE -kehitykseen. Tarkoitus on selvittää, mitä JVM-sovelluskehystä kannattaa käyttää nopeassa sovelluskehityksessä ja toisaalta saavutetaanko sovelluskehyksillä tehokkuutta ja kustannushyötyjä. Opinnäytetyö tehtiin Elbit Oy:n toimeksiannosta. Samanaikaisesti opinnäytetyön tekemisen aloittamisen kanssa (kesä-heinäkuu 2014) tehtiin Elbit Oy:lle vertailu kahdesta sovelluskehyksestä ja valittiin niistä toinen, jolla toteutin yritykselle MVP-sovelluksen (elokuussa 2014). Näin sain myös käytännön kokemusta sovelluskehyksistä ja niiden sopivuudesta MVP:n tekemiseen. Näin sain myös jonkinlaista käsitystä siitä, ovatko nopean kehityksen sovelluskehykset sopivia vain prototyyppien ja MVP-sovellusten tekoon, vai voisiko niillä tehdä myös tuotantokäytössä olevia sovelluksia.

Minulla on yli 20 vuoden työkokemus sovelluskehityksestä. 1990-luvulla käytin eniten Visual Basic -ympäristöä, joka oli sen ajan nopean sovelluskehityksen työkalu. Viime aikoina olen tehnyt kehitystyötä eniten Java EE -ympäristössä, jossa kehitystyö ei ole aina kovin tehokasta (Evans & Verburg 2013, 203), ja saatoin verrata nyt tutkimiani sovelluskehyksiä siihen.

Johdanto-luvun jälkeen toisessa luvussa käsitellään sovelluskehyksiä yleensä. Kolmannessa luvussa esitellään tarkemmin Grails-sovelluskehyksen ominaisuuksia. Neljännessä luvussa kertaan kokemuksista, joita saatiin kokeillessani Grails-sovelluskehystä käytännössä. Viidennessä luvussa esitellään tulokset ja johtopäätökset hankitun teoriatiedon ja käytännön kokemusten pohjalta. Lopuksi pohditaan vielä opinnäytetyön tekemisen prosessia ja tulevaisuuden näkymiä.

Opinnäytetyön tarkoituksena on vastata seuraaviin kysymyksiin:

- Mitä nopean kehityksen sovelluskehykset ovat ja miksi niitä tarvitaan?
- Miten voi valita sopivan sovelluskehyksen?
- Soveltuvatko nopean kehityksen sovelluskehykset demojen tai MVP-sovellusten tekoon?
- Mitä etuja niiden käytöstä on?
- Millaista sovelluskehysten käyttäminen on käytännössä?

2 Sovelluskehykset

2.1 Java-ohjelmointikielen ongelmat

Miksi tarvitaan nopean kehityksen sovelluskehyksiä? Jos on käyttänyt Javaa suuren sovelluksen tekoon, todennäköisesti huomaa, että koodi on monisanaista, jopa kömpelöä, ja toivoisi, että asiat voisi tehdä helpommin. Kuitenkin Java-suoritusympäristö (Java Virtual Machine eli JVM) on hyväksi havaittu ja tehokas sovellusympäristö, joten siksi Java on monen valinta. Tämän vuoksi on myös kehitetty muita ohjelmointikieliä, jotka kääntävät koodin suoritettavaksi JVM:n päällä, kuten Groovy, jota Grails-sovelluskehys käyttää, sekä Scala, jota Play-sovelluskehys käyttää. (Evans & Verburg 2013, 194.)

Ohjelmointikieliä jaotellaan useimmiten seuraavien piirteiden perusteella:

- tulkittu vai käännetty
- dynaamisesti tyypitetty vai staattisesti tyypitetty
- imperatiivinen vai funktionaalinen ohjelmointi.

Java on ajonaikaisesti käännetty, staattisesti tyypitetty, imperatiivinen kieli (Benjamin & Martijn 2013, 199). Imperatiivisessa ohjelmoinnissa ongelman ratkaisu kuvataan yksiselitteisesti vaihe vaiheelta riittävällä tarkkuudella käyttäen ohjelmointikielen komentoja. Komennot muuttavat suorituksen tilaa muokkaamalla laitteiston muistiin tallennettuja muuttujan arvoja. Imperatiivinen paradigma on tietokonelaitteiston luonnollinen tapa toimia. (Wikipedia 2015h.)

Imperatiivisen kielen rajoitteita ja tehottomuutta on ratkaistu Groovyssä lisäämällä funktionaalisen ohjelmoinnin piirteitä Java-kieleen (Benjamin & Martijn 2013, 192). Funktio voidaan laittaa muuttujaan ja antaa parametrina toiseen funktioon, jonka toiminta muuttuu tuon parametrina annetun funktion mukaisesti (Agrawal 2012).

Staattisesti tyypitetyssä kielessä muuttuja voi sisältää vain tietyn tyyppisiä arvoja. Sen sijaan dynaamisessa kielessä muuttuja voi sisältää eri aikoina erityyppisiä arvoja. Esimerkiksi Goovy ja JavaScript ovat dynaamisia kieliä. Seuraava esimerkki valaisee asiaa (Evans & Verburg 2013, 119):

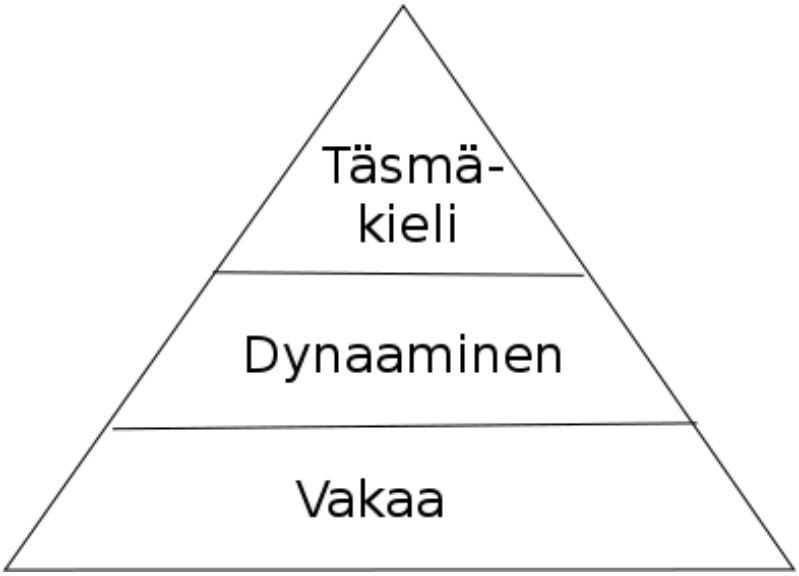
```
var vastus = 40;                //luo vastaus-muuttuja ja aseta numeerinen arvo
vastaus = vastaus + 3           //lisää arvoon 3
vastaus = ”Mikä on vastaus? Vastaus on ” + vastaus;   //liitä merkkijono vastaukseen
```

Esimerkissä vastaus-muuttaja sisältää aluksi numeerisen arvon, mutta lopuksi tekstiä. Lisäksi plus-operaatio aluksi suorittaa yhteenlaskua, mutta myöhemmin liittää kaksi tekstiä yhteen, eli operaation suoritus määräytyy dynaamisesti muuttujan sen hetkisen arvon tyypin mukaan. (Evans & Verburg 2013, 119.)

Groovy-kielessä imperatiiviseen ja staattiseen Java-kieleen on lisätty funktionaalisuutta ja dynaamisuutta. Näin kirjoitettavan koodin määrä vähenee ja kehitystyön pitäisi olla joustavampaa ja tehokkaampaa, mistä ei kuitenkaan vielä ole saatavissa selvää tutkimustietoa.

Tehokkuutta voidaan saavuttaa myös monikieliohjelmoinnilla. Monikieliohjelmoinilla JVM-ympäristössä tarkoitetaan ohjelmistoprojektia, jossa käytetään Javan lisäksi yhtä

tai useampaa muuta kieltä, joka suoritetaan JVM:n avulla. Kielet voidaan luokitella kuvion 1 mukaisesti kolmeen tasoon (Evans & Verburg 2013, 202):



Kuvio 1. Monikieliohjelmoinnin pyramidi (Evans & Verburg 2013, 202).

Java ja Scala ovat vakaita, kun taas Grails on dynaaminen kieli. Taulukossa 1 selitetään näitä termejä:

Taulukko 1. Monikieliohjelmoinnin tasot.

Nimi	**Kuvaus**	**Esimerkki**
Täsmäkieli (domain specific)	Täsmäkieli on ohjelmointikieli, joka on suunniteltu tietylle sovellusalueelle.	Apache Camel DSL, Drools, Web templating, HTML, CSS, SQL
Dynaaminen (dynamic)	Nopea, tuottava ja joustava toiminnallisuuden kehittäminen.	Groovy, Jython, Clojure, JRuby, JavaScript
Vakaa (stable)	Perustoiminnallisuus, vakaa, hyvin testattu, suorituskykyinen.	Java, Scala

Javan ominaisuudet tekevät siitä hyvän ohjelmointikielen, kun kehitetään vakaita ja pitkäikäisiä järjestelmiä. Toisaalta nämä samat hyvät ominaisuudet ovat taakka, kun tehdään dynaamisia tai tietyn sovellusalueen ratkaisuja. Javassa esimerkiksi

- uudelleen kääntäminen on työlästä ja aikaa vievää

- staattiset muuttujat ovat kankeita ja johtavat pitkiin koodin uudelleenkirjoituksiin
- kehitystyö on raskas prosessi
- ohjelmointikieli ei ole luontevaa tietyillä sovellusalueilla (Evans & Verburg 2013, 203).

Näitä Javan ongelmia ratkovat nopean kehityksen sovelluskehykset ja muut JVM:n päällä toimivat ohjelmointikielet.

2.2 Nopean kehityksen JVM-sovelluskehykset

Nopea web-kehitys on tärkeää. Yritysten ja erityisesti aloittavien yritysten täytyy pystyä kehittämään uusi tuote tai uusi ominaisuus nopeasti markkinoille. Loppukäyttäjät odottavat jatkuvasti uusia ominaisuuksia ja nopeita korjauksia ongelmiin. Valitettavasti Java ja monet Java-kehykset eivät ole sopivia nopeaan kehitykseen. Siksi on kehitytty nopean kehityksen JVM-kehyksiä, kuten Grails, Play ja Vaadin, jotka nopeuttavat sovelluskehitystä. (Evans & Verburg 2013, 380.)

2.3 Valintakriteerit

Jotta voisi valita itselleen sopivan nopean kehityksen sovelluskehyksen, täytyy miettiä omia tarpeita ja miten kukin sovelluskehys täyttää nämä tarpeet. Hyvän päätöksen tekeminen sisältää seuraavat vaiheet (Raible 2014, 20):

1. Selvitä tavoitteet.
2. Arvioi tavoitteiden tärkeys.
3. Listaa vaihtoehdot.
4. Arvioi miten kukin vaihtoehdoista vastaa tavoitteitasi.
5. Tee valinta.
6. Muuta tavoitteitasi.

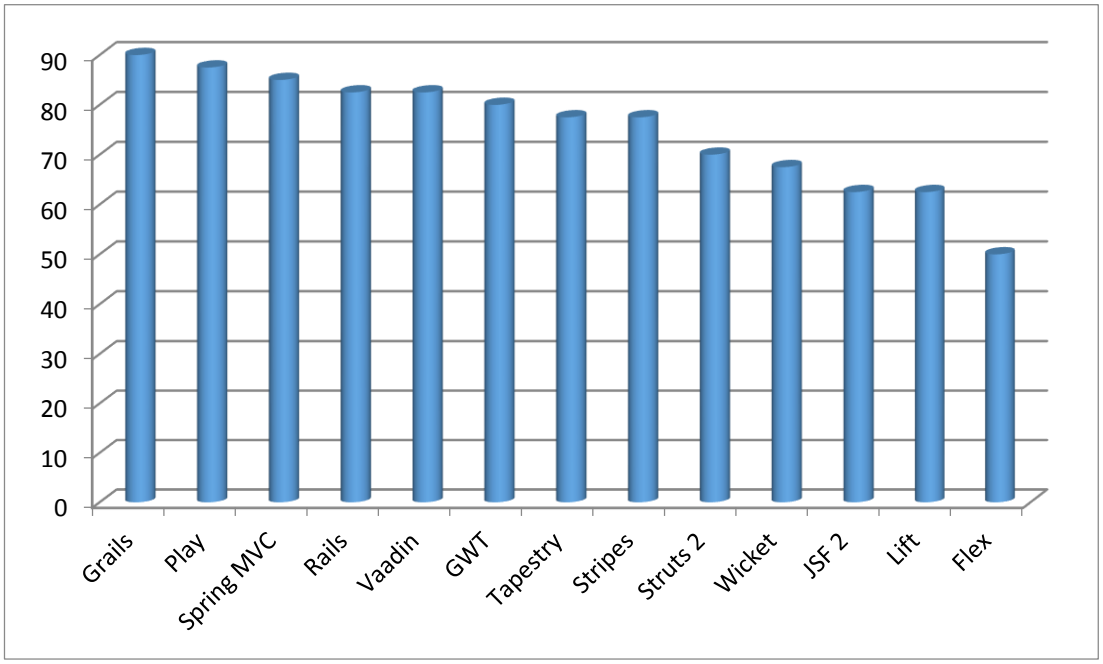
Lisäksi Matt Raible on päätynyt 20 kriteeriin (taulukko 2) web-kehystä valittaessa:

Taulukko 2. Kriteerit web-kehystä valittaessa (Evans & Verburg 2013, 384).

Kriteeria	**Esimerkki**
Kehittäjän tuottavuus	Voiko kehittää CRUD-sovelluksen 1 vai 5 päivässä?
Kehittäjien käsitys	Onko se mukava käyttää?
Oppimiskynnys	Olenko tuottava viikon vai kuukauden päästä?
Projektin tila	Onko projekti kriittisessä tilanteessa?
Kehittäjien saatavuus	Onko saatavilla kehittäjiä, joilla on osaamista?
Trendi	Onko tulevaisuudessa saatavilla kehittäjiä?
Mallit	Voiko noudattaa Don't repeat yourself (DRY) eli Älä Toista Itseäsi -periaatetta (ÄTI)
Komponentit	Onko valmiita komponentteja, esimerkiksi kalenteri tai päivämäärävalitsin?
Ajax	Tukeeko se asynkronisia JavaScript kutsuja?
Lisäosat (Plugins, add-ons)	Voiko helposti lisätä esim. Facebook-integraation?
Skaalautuvuus	Selviytyykö esim. yli 500 yhtaikaisesta käyttäjästä?
Tuki testaukselle	Voiko tehdä testivetoista kehitystä?
I18n ja l10n	Tukeeko se monikielisyyttä ja lokalisointia?
Validointi	Voiko helposti tarkistaa käyttäjien syötteet ja antaa nopeaa palautetta?
Monikielisyyden tuki	Voiko käyttää esimerkiksi sekä Javaa että Goovyä?
Dokumenttien ja oppaiden taso	Ovatko yleisimmät käyttötapaukset ja kysymykset dokumentoituja?
Kirjallisuuden saatavuus	Ovatko alan asiantuntijat käyttäneet sitä ja kertoneet kokemuksistaan?
REST-tuki	Tukeeko se http-protokollan käyttöä?
Mobiili-tuki	Onko helppo tukea Android-, iOS- ja muita mobiililaitteita
Riskien taso	Ollaanko tekemässä sovellusta ”ydinvoimalan” vai ”ruokareseptien” hallintaan?

Kriteereitä on monia ja jokaisen täytyy itse miettiä, mitkä niistä ovat itselle tai projektille tärkeimpiä. Matt Raible on äskettäin tehnyt arvion yleisimmistä JVM-kehyksistä, jonka tulokset ovat kuviossa 2. Arvion painotuksia voi myös itse muokata taulukon

kautta, joka on saatavilla osoitteesta http://bit.ly/jvm-frameworks-matrix. (Evans & Verburg 2013, 384, 285.) Parhaimmat arviot saivat Grails ja Play.



Kuvio 2. Matt Raiblen painotettu arvio JVM-kehyksistä (Evans & Verburg 2013, 385).

Mielestäni tarjolla olevista kehyksistä kannattaa valita teoriatiedon perusteella muutama, joihin perehtyy tarkemmin ja tekee käytännön kokeiluja joidenkin ominaisuuksien osalta. Näin saa myös käytännön kokemusta valintansa pohjaksi. Minä perehdyin tarkemmin Play- ja Grails-sovelluskehyksiin. Lopulta päädyin tekemään MVP-sovelluksen Grails-sovelluskehyksen avulla, koska oppimiskynnys oli matalampi, lisäosia oli enemmän saatavilla, kehittäjän tuottavuus oli parempi ja dokumentointi oli paremmalla tasolla. Grailsin lisäksi käytin AngularJs:ää ja Bootstrapia käyttöliittymän toteutukseen.

2.4 Play

Play on Scala-kieleen pohjautuva nopean kehityksen sovelluskehys. Myös Java-kieltä voisi käyttää, mutta suurin osa näyttäisi käyttävän Scala-kieltä, sillä suurin osa valmiista esimerkkiratkaisuista on tehty Scala-kielellä ja Java-kielisiin toteutuksiin ei löytynyt niin paljon ohjeita. Palvelimena on Akka-palvelin, joka tarjoaa hyvää suorituskykyä, sillä suoritus on asynkronista ja ei-blokkaavaa actor-pohjaista toteutusta käytettäessä

(Hatami 2012, 33), joten toisen tehtävän suorittaminen voidaan aloittaa ennen kuin ensimmäinen tehtävä on suoritettu loppuun asti.

Seuraavassa on joitain kokemuksiani Play-sovelluskehyksen käytöstä.

Asennus

- Asentaminen oli helppoa. Latasin ohjelman zip-paketin koneelleni ja asensin JDK:n. Sitten aloitin käytön käynnistämällä Activator-ohjelman.

Ensimmäisen sovelluksen teko

- Playn mukana tulee valmiita mallisovelluksia, joita testaamalla ja koodia tutkimalla voi perehtyä siihen, mitä ja miten sovelluksia voi tehdä.

Valmiit komponentit ja niiden käyttö

- Play-sovelluskehykseen on saatavilla muiden tekemiä valmiita komponentteja. Monet komponentit toimivat vain vanhassa 1.x-versiossa eivätkä nykyisissä 2.x-versioissa.

Edut

- Valmiit mallisovellukset nopeuttavat alkuun pääsyä.
- Suorituskyky on hyvä ja suoritus on yleensä ei-blokkaavaa.
- Voi käyttää funktionaalista Scala-kieltä.
- Sovelluskehitys on nopeampaa, koska koodin tehdyt muutokset näkyvät heti ilman uudelleenkäynnistystä tai tarvetta kääntää uudestaan.
- Hyvä virheenkäsittely. Play näyttää selkeän virheilmoituksen sekä koodin, joka virheen aiheutti.

Heikkoudet

- Koska Playn 1.x-versio ei ole yhteensopiva 2.x-version kanssa, monet vanhat esimerkit ja valmiit komponentit eivät enää toimineet. Vanhoja komponentteja vaikutti olevan enemmän kuin uusia.
- Vaikka Java-kieltä voi käyttääkin, niin suurin osa esimerkeistä on Scala-kielellä. Jos ennestään osaa vain Java-kielen, tulee opiskella myös Scala.
- Dokumentointi ei kaikilta osin ole ajan tasalla.

- Sovellusten kääntämistä varten Playn käyttää SBT-työkalua, joka on kyllä tehokas, mutta vaikea käyttää ja vaatii paljon opiskelua. (Brikman 2013.)

2.5 Grails

Grails on Groovy-kieleen pohjautuva nopean kehityksen sovelluskehys. Sen toiminnallisuuden perustana on toisten osapuolien kehittämiä ohjelmistoja, kuten Spring, Hibernate, JUnit ja Tomcat-server. Kehityksessä noudatetaan sovittuja tapoja ennemmin kuin konfiguroinnin avulla kerrotaan joka asia (sopimus mieluummin kuin konfigurointi eli *convention over configuration*) (Evans & Verburg 2013, 385). Esimerkiksi kooditiedostojen tulee olla sovituissa hakemistoissa ja sovitulla tavalla nimetty. Tämä mielestäni helpottaa ja nopeuttaa kehitystä, kun ei tarvitse muokata aina konfigurointitiedostoja ja toisen tekemä koodikin on nopeampi ymmärtää, kun asiat on toteutettu aina samalla sovitulla tavalla.

Seuraavassa on joitain kokemuksiani Grailsin käytöstä:

Asennus

- Grailsin asennus oli helppo. Latasin ohjelman zip-paketin ja purin sen haluamaani kansioon esim. C:\grails-2.4.1. Tämän jälkeen piti asettaa joitakin ympäristömuuttujia, kuten GRAILS_HOME= C:\grails-2.4.1.

Ensimmäisen sovelluksen teko

- Sovelluksen hakemistorakenne on vakio, mikä helpottaa alkuun pääsyä ja toisten tekemien esimerkkiohjelmien ymmärtämistä ja muokkausta.
- Scaffoldingin eli koodin automaattisen generoinnin avulla voi generoida kontrollereja ja näkymiä tietokantaobjektien pohjalta. Tämä nopeuttaa alkuun pääsyä ja voi nopeasti tehdä erilaisia kokeiluja.
- Valmiita esimerkkisovelluksia on saatavilla paljon.

Valmiit komponentit ja niiden käyttö

- Valmiita plugineja on noin tuhat, ja ne löytyvät osoitteesta https://grails.org/plugins/?filter=all

- Pluginit nopeuttavat kehitystä ja tarjoavat valmiita ratkaisuja yleisiin tarpeisiin, kuten tietoturva, sisäänkirjautuminen, sähköpostinlähetys jne.
- Pluginien mukana tulee usein myös esimerkkiohjelma, josta saa mallia sen käytöstä.
- Joskus pluginien käytössä on ongelmia versioiden kanssa. Pluginin tietty versio toimii vain joidenkin Grails-versioiden kanssa ja jos käyttää monia plugineja, niin pitää käyttää sellaista Grails-versiota, joka toimii kaikkien käyttämiesi pluginien kanssa. Tämän vuoksi ei useimmiten kannata heti käyttää ihan uusinta Grails-versiota, koska pluginit eivät vielä tue sitä. Lisäksi jotkut pluginit ovat riippuvaisia toisistaan, ja jos päivittää yhdestä pluginista uudemman version, niin tulee päivittää myös siitä riippuvat pluginit.

Edut

- Groovy-kielen käyttö helppoa, jos Java on ennestään tuttu.
- Valmiit pluginit nopeuttavat kehitystä.
- Scaffoldingin avulla pääsee nopeasti alkuun.
- Koodissa ei ole niin paljon toistoa kuin Java-koodissa.
- Ohjaa testivetoiseen kehitykseen.
- Sovelluksen voi asentaa monille suosituille Java EE -palvelimille, kuten Tomcatiin, WildFlyhyn, WebLogiciin jne.
- Tarvittaessa voi käyttää Javan kirjastoja.

Heikkoudet

- Automaattisesti generoidut “näkymättömät” metodit nopeuttavat kehitystä, mutta eivät aina toimi niin kuin pitäisi. Esimerkiksi tietokantahakuja varten on automaattisesti olemassa metodit, kuten findById, findByLastName tai findByLastNameOrFirstName. Yksinkertaiset metodit toimivat, mutta monimutkaisemmat haut eivät useinkaan toimineet, vaan piti itse koodata metodi, joka teki saman asian toisella tavalla ja toimi.

2.6 Muita nopean kehityksen sovelluskehyksiä

Vaadin on Suomessa kehitetty sovelluskehys. Aluksi se sisälsi vain käyttöliittymäkomponentteja, mutta nykyään se sisältää kattavasti kaikki osa-alueet. Sen käyttöliittymäkomponentit pohjautuvat teknologioihin kuten Javascript, HTML5 ja Google Web Toolkit (GWT) (Wikipedia 2015i).

Clojure poikkeaa suuresti Javasta, Grailsista tai Playsta. Se pohjautuu yhteen vanhimmista ohjelmointikielistä, nimittäin Lispiin, mutta sillä tehdyt sovellukset voi suorittaa Javan ajoympäristössä (JVM). Näin ollen on mahdollista kutsua Clojure-koodista myös Java-koodia. Clojure yhdistää Lispin vahvuudet joihinkin nykyajan teknologioihin. (Benjamin & Martijn 2013, 279.)

Muita suosittuja JVM-Web-sovelluskehyksiä ovat Spring MVC ja JSF, mutta niitä ei pidetä nopean kehityksen sovelluskehityksinä (Hatami 2012, 8, 10 ja 11), joten en käsittele niitä tässä yhteydessä. Springin käyttöä helpottamaan on kehitetty Spring Boot, joka on yleistynyt vasta aivan viime aikoina eikä ollut esillä, kun tätä opinnäytetyötä alettiin tehdä.

2.7 Hyödyllisiä lisäosia sovelluskehyksen rinnalle

Sovelluskehyksistä ei aina löydy kaikkia tarvittavia ominaisuuksia tai kolmannen osapuolen ratkaisu tarjoaa paremman ratkaisun asiaan. Itse käytin AngularJs- ja Bootstrap-lisäosia. AngularJs:n avulla voi tehdä sivustoja, jotka muuten vaatisivat JavaScript-koodin kirjoittamista, joka on hankalaa ja virhealtista. Bootstrap helpottaa ulkoasun tekemistä hyvän näköiseksi ja sellaiseksi, joka toimii useimmissa selaimissa (Wikipedia 2015j).

2.7.3 AngularJs

AngularJs on se, mitä HTML olisi ollut, jos se olisi suunniteltu web-sovellusten tekoon. Se soveltuu hyvin Single Page Architecture (SPA) -sovellusten tekoon, jossa paljon toiminnallisuutta on yhdellä web-sivulla, jota ei ladata joka toimenpiteen jälkeen uudes-

taan vaan sivulla olevaa tietoa muokataan dynaamisesti. AngularJs:n käyttämisen ansiosta sovelluksen koodaajan ei tarvitse tehdä JavaScript-koodia, vaan tarvittava koodi generoituu AngularJs:n määritysten avulla. Tämä mielestäni helpottaa paljon kehitystä, kun AngularJs huolehtii siitä, että sovelluksen JavaScript toimii eri selaimissa (Safari, Chrome, Firefox, Opera 15, IE9 ja mobiiliselaimista Android, Chrome Mobile, iOS Safari (AngularJs.org 2015)).

AngularJs kehittyi JavaScript MVC -kehyksestä systeemiksi joka

- tarjoaa yksinkertaisen HTML-mallin mukaisen kielen, jota on helppo laajentaa
- sisältää tehokkaan, kaksisuuntaisen tietojen sitomisen (data binding) niin että käyttöliittymäkerroksen ja sovelluslogiikkakerroksen objektit pysyvät samassa tilassa
- ei vaadi selaimen DOM-objektien käsittelyä. AngularJs huolehtii käyttöliittymäobjektien päivittämisestä.
- sisältää laajan ja kasvavan laajennusmoduulien ekosysteemin
- on Googlen tukema ja käyttämä

(Smith & Ledbrook 2014, 363, 364).

AngluarJs:ää voi hyvin käyttää yhdessä REST-rajapintojen kanssa. REST-rajapinnan voi toteuttaa esimerkiksi Grailsin avulla, joka muodostaa yhteyden tietokantaan, muokkaa tietoa tarvittavalla tavalla ja tarjoaa sen REST-rajapinnan kautta käyttöliittymälle.

2.7.3 Bootstrap

Bootstarp on ilmainen, avoimen lähdekoodin HTML-, CSS- ja JavaScript-sovelluskehys, jonka avulla voi kehittää mukautuvia web-sivustoja ja web-sovelluksia. Käyttämällä Bootsrapiä sivusto mukautuu käyttäjän selaimen ja näyttökoon mukaan sekä toimii myös mobiililaitteissa (Wikipedia 2015j).

3 Grails

Seuraavaksi käsittelen tarkemmin Grailsin sovelluskehystä. Käsittelen sen ominaisuuksia, sovellusten rakennetta ja ohjelmointia Groovy-ohjelmointikielellä.

3.1 Ominaisuudet

Grails on web-sovelluskehys, joka pohjautuu Java- ja Groovy-ohjelmointikieliin ja jolla tehtyjä ohjelmia voi suorittaa olemassa olevilla Java-palvelimella, kuten Tomcatilla ja Jettyllä (Smith & Ledbrook 2014, 468). Muutamassa minuutissa voi luoda uuden projektin ja sovelluksen generoimalla sen scaffolding-ominaisuudella. Grails pohjautuu *sopimus mieluummin kuin konfigurointi* -ideaan, joka liittää sovelluksen osat yhteen nimeämiskäytäntöjen eikä xml-konfigurointitiedostojen avulla. Grails-sovelluskehyksen avulla voi aloittaa kehitystyön heti ilman mitään konfigurointeja. Tämä on mahdollista, koska se tarjoaa automaattisesti Tomcat-webserverin ja HSQLDB-tietokannan käytettäviksi. (Kumar 2013.)

Tietokantaobjektit kuvataan GORMin (Grails Object Relational Mapping) avulla. Se pohjautuu Hibernateen, joka on monille tuttu Java EE -puolelta, mutta GORM peittää sen monimutkaisuuksia. Grails käyttää Java EE -arkkitehtuuria pohjanaan ja Springiä sovelluksen rakentamiseen riippuvuusinjektioiden avulla. Sovelluksen kääntämistä varten Grails käyttää omaa järjestelmäänsä nimeltä Gant, joka versiosta 3 lähtien on korvattu Gradle-työkalulla. (Kumar 2013.)

Grails-sovellus koostuu malleista, kontrollereista ja näkymistä (MVC-malli). Lisäksi siinä voidaan käyttää palveluja ja valmiita plugineja. Kuviossa 3 esitellään Grails-arkkitehtuuri.

Kuvio 3. Grails-arkkitehtuuri (Kumar 2013).

3.2 Sovelluksen rakenne

Grails-sovelluksen hakemistorakenne on aina sama. Kun luo sovelluksen komennolla *grails create-app*, luodaan seuraavat hakemistot:

```
%PROJECT_HOME%
    + grails-app
       + conf               ---> konfigurointi tiedostot
          + hibernate       ---> valinnainen, hibernaten konfigurointi
          + spring          ---> valinnainen, spring konfigurointi
       + controllers        ---> controller luokat
       + domain             ---> domain luokat
       + i18n               ---> tekstit eri kielillä
       + services           ---> palvelus
       + taglib             ---> tag kirjastot
       + util               ---> erityiset työkalu luokat
       + views              ---> näkymät
          + layouts         ---> ulkoasut
   + lib                   ---> kirjastot
   + scripts               ---> scriptit
   + src
       + groovy            ---> valinnainen; Groovyn lähdekoodi
                                (muun tyyppiset kuin grails-app
```

```
                                hakemiston alla)
    + java                ---> valinnainen; Java lähdekoodi
+ test                    ---> generoidut testiluokat
+ web-app                 --->
    + WEB-INF             --->
```

Tärkeimmät hakemistot ovat conf-hakemisto, jossa on konfigurointitiedostoja, sekä domain-, views- ja controllers-hakemisto, joissa on MVC-mallin mukaisesti sovelluksen koodi. Test-hakemistossa on koodi testivetoista kehitystä varten.

3.3 Groovy-kieli

Groovy-ohjelmointikieli kehitettiin 2003. Se on dynaaminen, käännetty kieli, jonka syntaksi on Javan tapainen mutta joustavampi (Evans & Verburg 2013, 204).
Eroja Javaan ovat muun muassa seuraavat:

- Muuttujan tyyppiä ei ole pakko määritellä.
- Puolipiste rivin lopussa ei ole pakollinen.
- For-silmukassa voidaan käyttää yksinkertaisempaa syntaksia: for (i in 0…<10) {…}.
- Luokan muuttajat ovat oletuksena public- eikä private-tyyppisiä, joten ei ole pakollista koodata get- ja set-metodeja niiden käsittelyä varten.

Groovyssä on joitain ominaisuuksia joita ei ole vielä Javassa:

- GroovyBeans eli yksinkertaisemmat JavaBeans-luokat, joissa ei ole get- ja set-metodeja ja joissa on automaattinen konstruktori
- Null-objectien käsittely ?. syntaksilla
- Elvis-operattori (?:), joka on vielä lyhempi tapa tehdä *if-else*-rakenne
- Groovy- merkkijonot, jotka voivat sisältää muuttujan arvolla korvattavia kohtia
- Helpompi XML-käsittely. (Evans & Verburg 2013, 226.)

3.4 Domain-objektit

Sovellusalueen luokat eli domain-luokat ovat yleensä ensimmäinen asia, joka Grails-sovellukseen tehdään. Niissä määritellään olio-luokat, joita sovelluksessa käsitellään, ja

niiden perustella generoituvat automaattisesti tietokantataulut. Kontrollerien ja näkymien nimet johdetaan sovellusalueen luokkien nimestä. Esimerkiksi jos henkilö-luokka on nimeltään Person, niin sen kontrollerin nimi on aina PersonController ja näkymän nimi on aina PersonView. Domain-luokassa määritellään käytettävät tietokentät, oikeellisuustarkistukset ja luokkien väliset riippuvuudet. (Kumar 2013.)
Domain-luokka voidaan luoda komennolla:
grails create-domain-class domain-class-name
Komento luo sekä domain-luokan että sille testiluokan.

GORM (Grails Object-Relational Mapping) on Grailsin kerros Hibernaten päällä, joka piilottaa Hibernatessa olevaa monimutkaisuutta ja jonka ansiosta domain-luokat ovat yksinkertaisempia. Tietokantalähde, jota sovellus käyttää, määritellään tiedostossa ***grailsapp/conf/DataSource.groovy***.

3.5 Kontrollerit ja näkymät

Kontrollerit ja näkymät voidaan generoida domain-luokan mukaan ja ne toteuttavat automaattisesti CRUD-toiminnot. Komento *grails generate-all domain-class-name* luo kontrollerin ja näkymät. Esimerkiksi grails generate-all Person loisi PersonController-luokan ja tarvittavat näkymät henkilön luontia, lukemista, päivittämistä ja poistoa varten. Generoitua koodia voidaan itse muokata. Jos luokkia ja niiden koodia ei haluta generoida, voidaan ne kirjoittaa alusta asti itsekin. Kontrolleri-luokat voidaan helposti määrittää myös tarjoamaan REST-rajapinnan.

3.6 Testivetoinen kehitys

Grails ohjaa testivetoiseen kehitykseen luomalla uusista domain- ja controller-luokista myös uudet testiluokat yksikkötestausta varten. Testivetoinen kehitys on ollut osa sovelluskehitysalaa jo jonkin aikaa. Sen lähtökohta on, että ensin kirjoitetaan testitapaus ja vasta sitten koodi, joka toteuttaa ominaisuuden ja jota parannetaan asteittain. Esimerkiksi jos haluaa toteuttaa ominaisuuden, joka liittää kaksi tekstijonoa yhteen (esim.

”*foo”* ja ”*bar*”), tulee kirjoittaa ensin testi (testaa että tulos on ”*foobar*”), joka varmistaa että implementointi on oikein. (Evans & Verburg 2013, 313, 314).

Miksi tehdä testivetoista kehitystä ja mitkä ovat sen hyödyt? Epävarmuuden ja pelon poistaminen on tärkeimpiä syitä. Kun muokkaa olemassa olevaa koodia, ei tarvitse pelätä, että huomaamatta hajottaa sen toiminnallisuuden. Testit kertovat, toimiiko koodi edelleen. Muita hyötyjä ovat:

- Selvempi koodi. Testi ohjaa kirjoittamaan vain sen koodin, mitä tarvitaan.
- Parempi suunnittelu. Jotkut puhuvat testivetoisesta suunnittelusta.
- Suurempi joustavuus. Testivetoinen kehitys rohkaisee koodaamaan selkeitä rajapintoja.
- Nopea vaste. Voi saada selville virheet heti, eikä vasta myöhemmin tuotannossa.

(Evans & Verburg 2013, 314.)

Testivetoista kehitystä voi tehdä neljällä tasolla:

1. yksikkötestaus
2. integrointitestaus
3. systeemitestaus
4. systeemi-integrointitestaus (Evans & Verburg 2013, 315).

Kuitenkin Grails-kehityksessä testaus jaetaan neljään vaiheeseen.

1.	Yksikkö	Täysin eristetty testi, joka ei käytä tietokantaa tai Grails-ympäristöä.
2.	Integrointi	Käyttää Grails-ympäristöä ja GORM käyttää oikeaa tietokantaa. Ei servlet-kontaineria.
3.	toiminnallisuus	Sovellus ajetaan servlet-kontainerissa ja testitapaukset suoritetaan http:n kautta.
4.	muu	Harvoin käytetty, esimerkiksi asennus-skriptien testaus. (Smith & Ledbrook 2014, 230.)

Testejä voi ajaa *test-app*-komennolla, joka testaa koko sovelluksen. Komennolle voi halutessaan antaa parametrina, minkä vaiheen testi ajetaan ja ajetaanko JUnit- vai Spock-testi (Smith & Ledbrook 2014, 231).

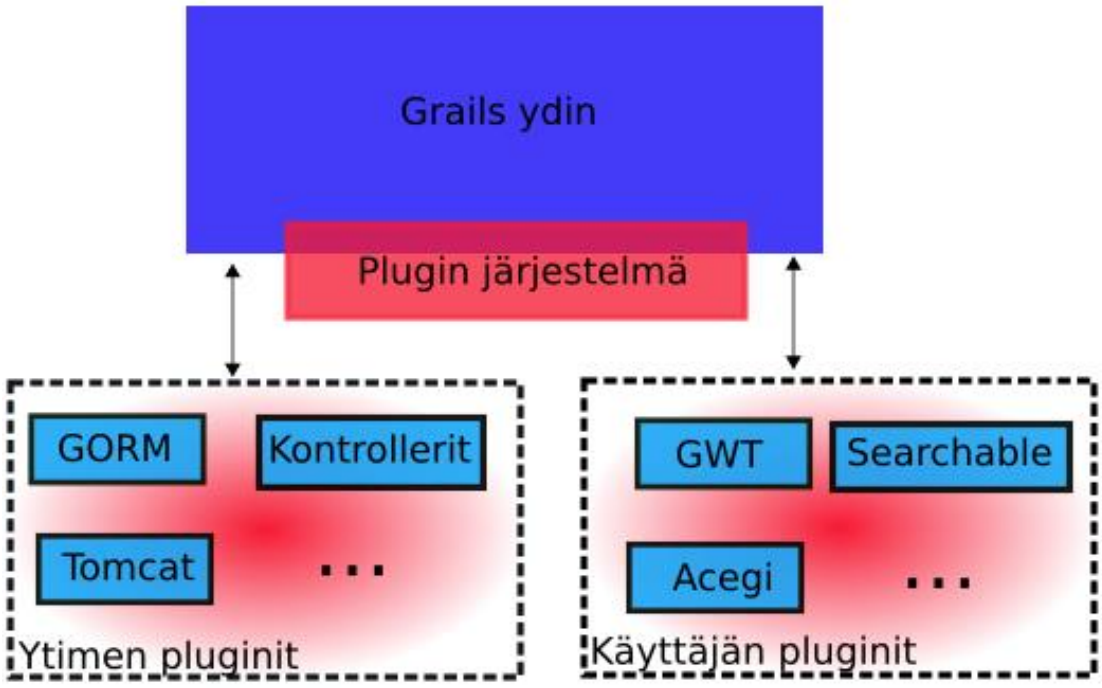
3.7 Scaffolding eli automaattinen koodin generointi

Grails voi generoida koodia ja luoda CRUD-sovelluksen automaattisesti domain-luokan pohjalta. Sen voi tehdä kahdella tavalla. Ensinnäkin voi käyttää generate-komentoja, jolloin luodaan kooditiedostot, joita voi myös muokata. Toinen tapa on tehdä *def scaffold=true* -määritys domain-luokkaan, jolloin Grails generoi tarvittavan koodin suorituksen aikana. Koodin generointi tehdään mallitiedostojen pohjalta. Tarvittaessa mallitiedostoja voi muokata haluamakseen. (Kumar 2013.)

Scaffolding-ominaisuus mahdollistaa nopean alkuun pääsyn sovelluskehityksessä. Mielestäni sen avulla voi tehdä nopeasti yksinkertaisia sovelluksia ja esitellä perustoiminnallisuutta. Generoituva koodi on kuitenkin harvoin sellaista, että sitä voisi käyttää monimutkaisemmissa ja oikeassa käytössä olevissa sovelluksissa.

3.8 Pluginit

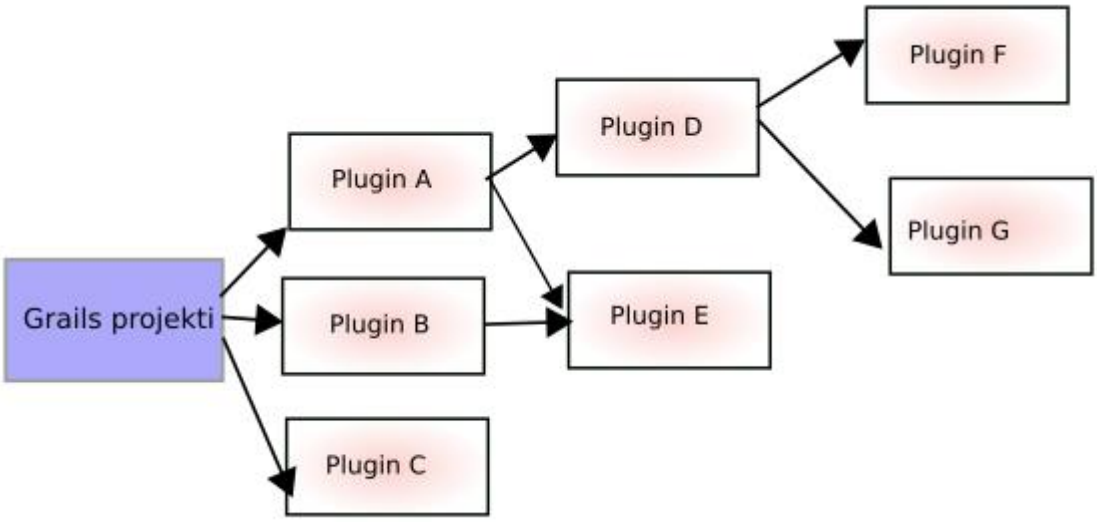
Grailsin ydintoiminallisuutta voi laajentaa lisäosilla, plugineilla (kuvio 4). Itse asiassa jotkin Grails-paketin perustoiminnoista, kuten GORM, on teknisesti toteutettu pluginina. Lisäksi Grailsin loppukäyttäjät voivat tehdä omia lisäosia, joita sitten muut voivat ottaa käyttöönsä. Lisäosia on saatavilla yli 1200 ja ne löytyvät Grailsin verkkosivustolta https://grails.org/plugins/.



Kuvio 4**.** Plugin-arkkitehtuuri (Smith & Ledbrook 2014, 262).

Grailsin ydin tarjoaa plugin-rajapinnan, jonka kautta sen toiminnallisuutta voi laajentaa joko ytimen mukana vakiona tulevilla plugineilla tai käyttäjien tekemillä plugineilla. Lisäosien avulla voi toteuttaa helposti esim. ajastettuja toimintoja, REST-rajapintoja, jQuery-toiminallisuutta, Facebook-autentikointia jne. (Evans & Verburg 2014, s. 398)

Plugin otetaan käyttöön lisäämällä grails-app/conf/BuildConfig.groovy-tiedostoon pluginin nimi ja versio. Monet lisäosat ovat itsenäisiä, mutta osa lisäosista on riippuvaisia muista lisäosista (kuvio 5), jotka tulee ottaa käyttöön ensin ja joiden tulee olla versioltaan yhteensopivia (Smith & Ledbrook 2014, 267). Osa lisäosista on myös riippuvaisia Grailsin versiosta, eivätkä ehkä toimi vanhempien tai uudempien versioiden kanssa.



Kuvio 5. Esimerkki pluginien riippuvuuksista toisistaan (Smith & Ledbrook 2014, 267).

3.9 REST-rajapinnat

REST (REpresentational State Transfer) on arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen. Se on esitetty usein SOAP- ja WSDL-mallien korvaajana. REST pohjautuu HTTP-protokollaan, jossa operaatioina ovat GET, POST, PUT ja DELETE. Nämä operaatiot vastaavat lue-, luo-, muokkaa- ja poista-operaatioita (Smith & Ledbrook 2014, 329). REST-rajapinnan avulla sovellus voi tarjota rajapintoja toisten sovellusten käyttöön tai käyttöliittymän ja sovelluslogiikan välinen tiedonvaihto voidaan toteuttaa sillä.

Grails tarjoaa monia tapoja toteuttaa REST-rajapintoja. (12.1.2) Nopein lähestymistapa on merkitä jokin luokista resurssiksi seuraavalla tavalla:

```
package com.grailsinaction
import grails.rest.resource
@Resource(uri=”/posts”)
class Post {
…
}
```

Tämä luo kontrollerin, joka tarjoaa JSON- ja XML-rajapinnan, jonka kautta voi lukea ja muokata tietoa. (Smith & Ledbrook 2014, 331.)

4 Käytännön kokemukset

4.1 Sovelluskehyksen valinta

Tein Elbit Oy:n toimeksiannosta MVP-sovelluksen. Aluksi selvitin parin viikon ajan, mitä Java-suoritusympäristössä toimivaa nopean kehityksen sovelluskehystä toteutuksessa kannattaisi käyttää. Tutkin netissä olevia vertailuja (Raible 2014) ja hankin yleistä tietoa aiheesta. Tämän pohjalta valitsin Grails- ja Play-sovelluskehykset tarkempaan tarkasteluun ja tein viikon ajan joitain testejä niillä. Testasin mm. millaista niillä on tehdä yksinkertainen sovellus, jossa on sisäänkirjautuminen ja mahdollisuus lähettää sähköpostia.

Taulukossa 3 on oma käsitykseni näistä kahdesta sovelluskehyksestä käyttäen Matt Raiblen 20 kriteeriä.

Taulukko 3. Grails- ja Play-sovelluskehysten arviot Matt Raiblen kriteereillä.

Kriteeria	**Esimerkki**	**Grails**	**Play**
Kehittäjän tuottavuus	Voiko kehittää CRUD-sovelluksen 1 vai 5 päivässä?	CRUD-sovelluksen sai aikaiseksi muutamassa tunnissa.	Jos käytti valmista sovellusmallia, niin CRUD sovelluksen sai tehtyä nopeasti, mutta muokkaus työläämpää kuin Grailsissä.
Kehittäjien käsitys	Onko se mukava käyttää?	Mukava käyttää, vaikka välillä törmää ylitsepääsemättömiltä vaikuttaviin ongel-	Mukava käyttää. Useat kehitystyökalut (Eclipse, IntelliJ) tukevat.

		miin. Useat kehitystyökalut (Eclipse, IntelliJ) tukevat.	
Oppimiskynnys	Olenko tuottava viikon vai kuukauden päästä?	Jo viikossa saa joitain aikaiseksi, mutta kaikkiaan oppimiseen kuluu useampi viikko.	Suurempi oppimiskynnys kuin Grailsissä.
Projektin tila	Onko projekti kriittisessä tilanteessa?	Ei kriittinen projekti.	Ei kriittinen projekti.
Kehittäjien saatavuus	Onko saatavilla kehittäjiä, joilla on osaamista?	Ei paljon, mutta Java-kehittäjät oppivat nopeasti Goovyn.	Ei paljon.
Trendi	Onko tulevaisuudessa saatavilla kehittäjiä?	Ehkä	Ehkä
Mallit	Voiko noudattaa Don't repeat yourself (DRY) eli Älä Toista Itseäsi -periaatetta (ÄTI)	Kyllä	Kyllä
Komponentit	Onko valmiita komponentteja, esimerkiksi kalenteri tai päivämäärävalitsin?	Lisäosien kautta saatavilla.	Lisäosien kautta saatavilla, mutta monesti toimivat vain vanhemmassa versiossa.
Ajax	Tukeeko se asynkronisia JavaScript kutsuja?	Kyllä	Kyllä
Lisäosat (Plugins, add-ons)	Voiko helposti lisätä esim. Facebook integraation?	Noin tuhat lisäosaa, jotka helposti löydettävissä yhdestä sivustolta.	Noin sata eikä vielä ole virallista sivustoa niitä varten.
Skaalautuvuus	Selviytyykö esim. yli 500 yhtaikaisesta käyttäjästä?	Skaalautuu hyvin.	Skaalautuu erinomaisesti.
Tuki testaukselle	Voiko tehdä testivetoista kehitystä?	Kyllä	Kyllä

I18n ja l10n	Tukeeko se monikielisyyttä ja lokalisointia?	Kyllä	Kyllä
Validointi	Voiko helposti tarkistaa käyttäjien syötteet ja antaa nopeaa palautetta?	Kyllä	Kyllä
Monikielisyyden tuki	Voiko käyttää esimerkiksi sekä Javaa että Goovyä?	Voi käyttää Java- ja Groovy-kieltä.	Voi käyttää Java- ja Scala-kieltä.
Dokumenttien ja oppaiden taso	Ovatko yleisimmät käyttötapaukset ja kysymykset dokumentoituja?	Erinomainen	Kohtalainen
Kirjallisuuden saatavuus	Ovatko alan asiantuntijat käyttäneet sitä ja kertoneet kokemuksistaan?	Kirjallisuutta on saatavilla, mutta ei aina vastaa uusinta versiota.	Kirjallisuutta on jonkin verran saatavilla.
REST-tuki	Tukeeko se http protokollan käyttöä?	Kyllä	Kyllä
Mobiilituki	Onko helppo tukea Android-, iOS- ja muita mobiili-laitteita	Ei	Ei
Riskien taso	Ollaanko tekemässä sovellusta ”ydinvoimalan” vai ”ruokareseptien” hallintaan?	Sovellus ei ole riskialtis.	Sovellus ei ole riskialtis.

Valitsin Grails-sovelluskehyksen, koska Groovy-kielen käyttö on helpompaa Java-kehittäjälle kuin Scalan käyttö, sen valmiit pluginit nopeuttavat kehitystä ja scaffoldingin avulla pääsee nopeasti alkuun. Play-sovelluskehyksen heikkouksina olivat se, että 1.x-version ollessa ei-yhteensopiva 2.x-version kanssa, monet vanhat esimerkit ja valmiit komponentit eivät enää toimineet ja vanhoja vaikutti olevan enemmän kuin uusia. Lisäksi vaikka Java-kieltä voikin käyttää, niin suurin osa esimerkeistä on Scala-kielellä,

jonka opiskelu on työläämpää kuin Groovyn. Lisäksi Grailsin dokumentointi on paremmalla tasolla kuin Playn.

4.2 Kehitystyö Grails-sovelluskehyksellä

Grailsillä pääsi sovelluskehityksen alkuun nopeasti. Asennus oli helppo ja uuden sovelluksen luominen tapahtui yhdellä komennolla. Sovelluksen domain-luokat oli helppo määrittää ja scaffolding-toiminnolla saattoi nopeasti kehittää malli- ja näkymä-luokat.

Pluginien käyttö nopeutti monien ominaisuuksien tekoa. Esimerkiksi kun tein Google-, Facebook- ja LinkedIn-kirjaumiset, niin ensimmäisenä tekemäni Google-kirjautuminen kesti tehdä muutaman päivän, koska opiskeluun meni paljon aikaa. Tämän jälkeen kuitenkin Facebook-kirjaumisen toteutin tunnissa ja LinkedIn-kirjaumisen varttitunnissa. Havaitsin, että uusien pluginien käyttöönotto kannattaa aloittaa kokeilemalla sen mukana useimmiten tulevaa esimerkkiohjelmaa. Näin saattoi varmistua, että ko. plugin yleensäkin edelleen toimii käytettävässä versiossa ja ympäristössä. Tämän kokeilun jälkeen sen saattoi ottaa käyttöön omaan projektiin.

Grailsin dokumentaatio on hyvällä tasolla ja sitä kautta saa tietoa, miten yksittäiset luokat tai niiden metodit toimivat. Tietoa kokonaisuuden kannalta parhaista toteutustavoista oli kuitenkin hankala löytää. Tietoa oli koottava hajanaisesti eri paikoista. Vasta heinäkuussa 2014 julkaistiin Grails in Action -kirja, josta olisi saanut lähes kaiken tarvittavan tiedon yhdestä paikasta. Sain tuon kirjan käsiini vasta syyskuun alussa, kun koodaustyö oli jo tehty.

5 Tulokset ja johtopäätökset

Opinnäytetyön tarkoituksena oli perehtyä Java-suoritusympäristössä toimiviin nopean kehityksen sovelluskehyksiin ja valita niistä yksi MVP-sovelluksen toteutusta varten. Näin saatiin teoriatiedon lisäksi käytännön kokemusta Grails-sovelluskehyksestä. Lisäksi saatoin verrata sitä Java EE -sovelluskehitykseen, josta minulla on aikaisempaa kokemusta useita vuosia.

5.1 Sovelluskehyksen valinta

Sovelluskehyksen valinta on monimutkainen prosessi ja yhtä ainoaa oikeaa valintaa ei ole. Jokaisella kehyksellä on omat etunsa ja haittansa. Sopivan valinta riippuu monesta seikasta, kuten aiemmin on tullut esille. Jossakin projektissa jokin tietty kriteeri voi olla erityisen tärkeä, ja näin ollen siinä projektissa päädytään käyttämään tämän takia sovelluskehystä, jossa tuo ominaisuus on vahva. Täysin kattavan vertailun tekeminen useista sovelluskehyksistä olisi vienyt paljon aikaa, joten osaltaan kannattaa luottaa toisten tekemiin vertailuihin ja niiden pohjalta valita kaksi tai kolme sovelluskehystä, joihin perehtyy tarkemmin. Valitsin käytettäväksi Grails-sovelluskehyksen edellisessä luvussa käsiteltyjen syiden pohjalta.

5.2 Nopean kehityksen sovelluskehykset verrattuna Java EE:hen

Grails-ympäristöllä sovelluksen toteuttaminen vaikutti olevan tehokkaampaa kuin Java EE -ympäristöä käytettäessä. Ensinnäkin koodirivejä tarvitaan vähemmän, koska Groovy-kieli on tehokkaampaa kuin Java ja kaikkia metodeja, esim. get- ja set-metodeja, ei useinkaan tarvitse itse toteuttaa. Toiseksi jos löysi sopivan pluginin, niin ominaisuus saattoi olla hyvinkin nopea toteuttaa. Pluginit oli helppo löytää, koska ne on koottu yhdelle web-sivulle. Kolmanneksi Grails-koodista asiat on helpompi löytää, koska hakemistorakenne on vakio ja luokat tulee nimetä sovitulla tavalla. Neljänneksi tämä *sopimus mieluummin kuin konfigurointi* -periaate vähentää konfigurointitiedostojen määrää, mikä myös helpottaa ja nopeuttaa kehitystä. Konfigurointitiedostoja minun täytyi muo-

kata vain, kun lisäsi uusia plugineja käyttöön tai muutti tietokannan nimeä. Viidenneksi koodausta nopeutti se, että sovellusta ei tarvitse erikseen kääntää ja käynnistää uudestaan joka muutoksen jälkeen. Periaatteessa kääntämisen pitäisi tapahtua aina automaattisesti eikä sovellusta tarvitse käynnistää uudelleen, mutta useiden muutosten jälkeen sovellus piti yleensä kuitenkin käynnistää uudestaan ennen kuin muutos tuli voimaan. Kuudenneksi tehokkuutta lisäsi se, että toteuttamisvaihtoehtoja ei ole niin paljon. Joskus Java EE -ympäristössä asian voi tehdä niin monella eri tavalla, että parhaimman etsiminen ja jonkin toteutustavan valitseminen vaatii paljon aikaa ja vaivaa.

Seuraavassa listassa ovat Grailsin edut ja haitat verrattuna Java EE -kehitykseen.
Grailsin edut:

- Scaffolding-ominaisuuden avulla pääsee nopeasti alkuun ja saa heti jotain näkyvää aikaiseksi.
- Ei juurikaan konfigurointia.
- Vakioitu hakemistorakenne ja vakioidut luokkien nimeämiskäytännöt auttavat lukemaan myös muiden koodia ja löytämään sieltä asiat helpommin ja nopeammin.
- Valmiit pluginit, joiden avulla voi toteuttaa jotkin ominaisuudet hyvinkin nopeasti.
- Käytettävissä tehokkaampi Groovy-kieli.

Grailsin heikkoudet:

- Grails muuttuu vielä nopeasti, mikä voi aiheuttaa yhteensopivuusongelmia eri versioiden välillä ja vanhemmissa versioissa olevia virheitä ei enää korjata.
- Dokumentaatiosta pitää aina selvittää, mitä versiota se koskee. Tämä ei kuitenkaan Grailsin dokumentaatiossa ollut niin suuri ongelma.
- Pluginin tietty versio toimii Grailsin tietyn version kanssa.
- Grailsissä bugeja. Erityisesti tietokantaoperaatiot eivät aina tee sitä, mitä pitäisi. Täytyy etsiä vaihtoehtoisia tapoja tehdä sama asia tai jopa turvautua tekemään asia suoraan SQL-lauseilla.

Java EE:n vahvuudet verrattuna nopean kehityksen sovelluskehyksiin:

- Vakaa ja toimivaksi testattu järjestelmä.

- Paljon tietoa saatavilla, joskus liiankin paljon, josta on vaikea löytää olennainen tai paras mahdollinen.
- Avoimet standardit.

Java EE heikkoudet nopean kehityksen sovelluskehyksiin:

- Java-kieli ei ole niin tehokas ja ketterä kuin Groovy tai Scala.
- Ei valmiita plugineja tai on vaikea löytää toisten tekemiä valmiita komponentteja, koska niitä ei ole koottu yhdelle sivustoille.
- Paljon konfigurointitiedostoja.

Nopean kehityksen sovelluskehitykset soveltuvat MVP:n tekemiseen. Kun tekee ensimmäisen projektinsa nopean kehityksen sovelluskehyksellä, saavutettu hyöty ei tule heti esille, koska uuden opiskeluun menee paljon aikaa. Uskoisin että seuraavissa projekteissa hyöty olisi selvempi, koska monet asiat voidaan tehdä niillä nopeammin kuin perinteisessä Java EE -ympäristöllä.

6 Pohdinta

6.1 Opinnäytetyön prosessi

Opinnäytetyön teko oli pitkä prosessi, joka alkoi kesäkuussa 2014. Työskennellessäni Elbit Oy:ssä tein ensin teoriatutkimusta ja sen jälkeen käytännön kokeiluja. Elokuun loppuun mennessä toteutin MVP-sovelluksen käyttäen Grails-sovelluskehystä. Tämä oli hyvin opettavaa ja toisaalta kiireistä aikaa, eikä paljon jäänyt aikaa opinnäytetyön kirjoittamiseen. Syyskuusta lähtien keskityin enemmän opinnäytetyön kirjoittamiseen ja hankin lisää teoriatietoa aiheesta. Kirjoittamisen sain valmiiksi huhtikuussa 2015. Koska opinnäytetyön prosessi kesti kaikkiaan lähes vuoden, on tuona aikana jo ehtinyt tapahtua joitain muutoksia alalla. Tämän takia olisi ollut parempi toteuttaa opinnäytetyö nopeammalla aikataululla, jotteivät asiat ehdi muuttua kesken kaiken.

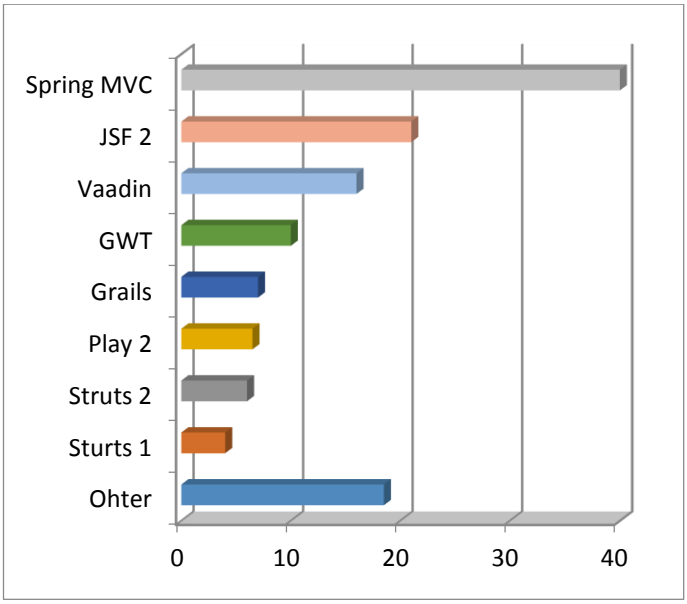
Kaikki lähdeaineisto on ollut englanninkielistä ja joillekin termeille on ollut vaikea löytää suomennosta, mutta mielestäni oli parempi kirjoittaa opinnäytetyö suomeksi suomalaisille. Prosessin aikana olen oppinut paljon sovelluskehyksistä, uusimmista teknologioista, alan nykytrendeistä ja työelämän ongelmien ratkaisemista niitä käyttäen. Opin myös ammattimaista asiantuntijakirjoittamista. Opinnäytetyön aihe oli ajankohtainen ja mielenkiintoinen.

6.2 Tulevaisuus

Nopean kehityksen JVM-sovelluskehysten tulevaisuus on vielä hieman epävarma. Mikään kehikoista ei ole vielä saavuttanut selvää ykkösasemaa eikä niiden suosio ole vielä niin suurta kuin perinteisten kehitysympäristöjen, kuten String MVC:n, J2EE:n, PHP:n tai Microsoft Visual Studion. Rebellabsin julkaisemien tilastojen mukaan nopean kehityksen sovelluskehyksistä eniten suosiotaan on kasvattanut Vaadin, jonka prosenttiosuus käytetyistä web-sovelluskehyksistä on yli kaksinkertaistunut vuodesta 2012 (Hatami 2012) vuoteen 2014 (White 2014a, 11). Seuraavassa on listattuna, montako prosenttia web-sovelluskehysten käyttäjistä käyttää kutakin web-sovelluskehystä ja miten prosenttiosuudet ovat muuttuneet vuodesta 2012 vuoteen 2014:

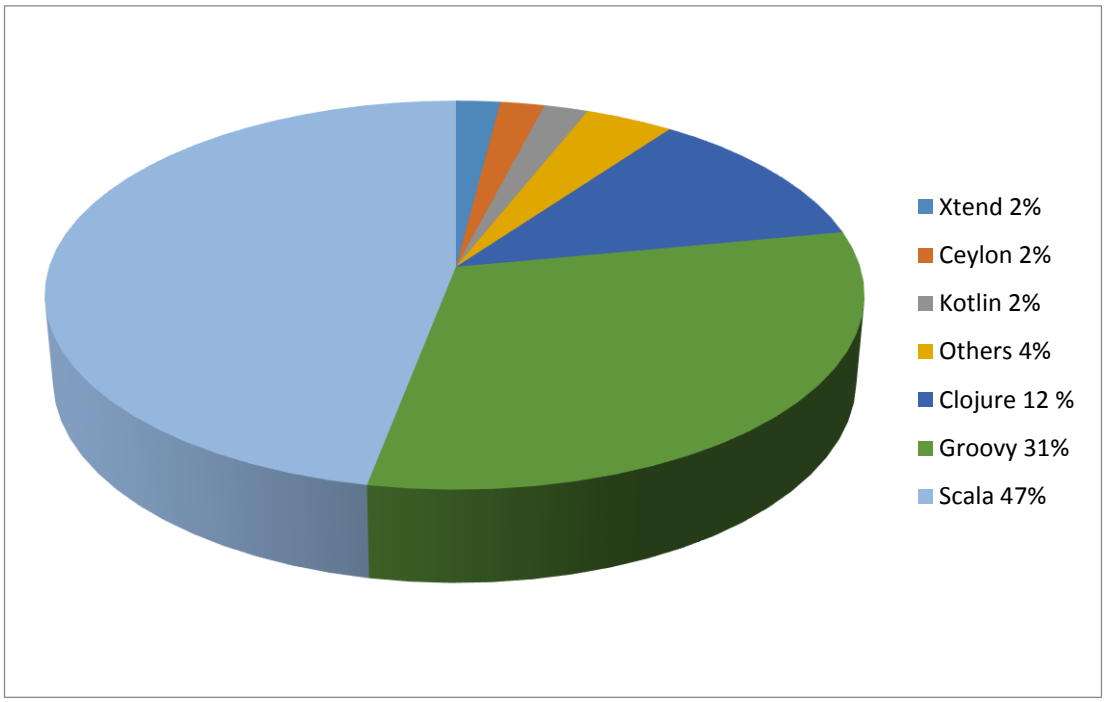
- Vaadin 7 % -> 16 %
- GWT 14 % -> 10 %
- Grails 7 % -> 7 %
- Play 8 % -> 6.5 %
- Spring MVC 30 % -> 40 % (White 2014a, 11)

Kuviossa 5 näkyy, montako prosenttia web-sovelluskehysten käyttäjistä käyttää kutakin web-sovelluskehystä. Spring MVC:n suosio johtuu osaltaan siitä, että noin kolmasosa muiden sovelluskehysten käyttäjistä käyttää rinnalla myös Spring MVC -sovelluskehystä. Myös itse käytin Spring MVC -kehystä Grailsia käyttäessäni, ja sitä varten on olemassa Grails plug-in, jonka ansiosta sen käyttö on helpompaa.



Kuvio 5. Mitä web-sovelluskehystä käytät (White 2014a, 11).

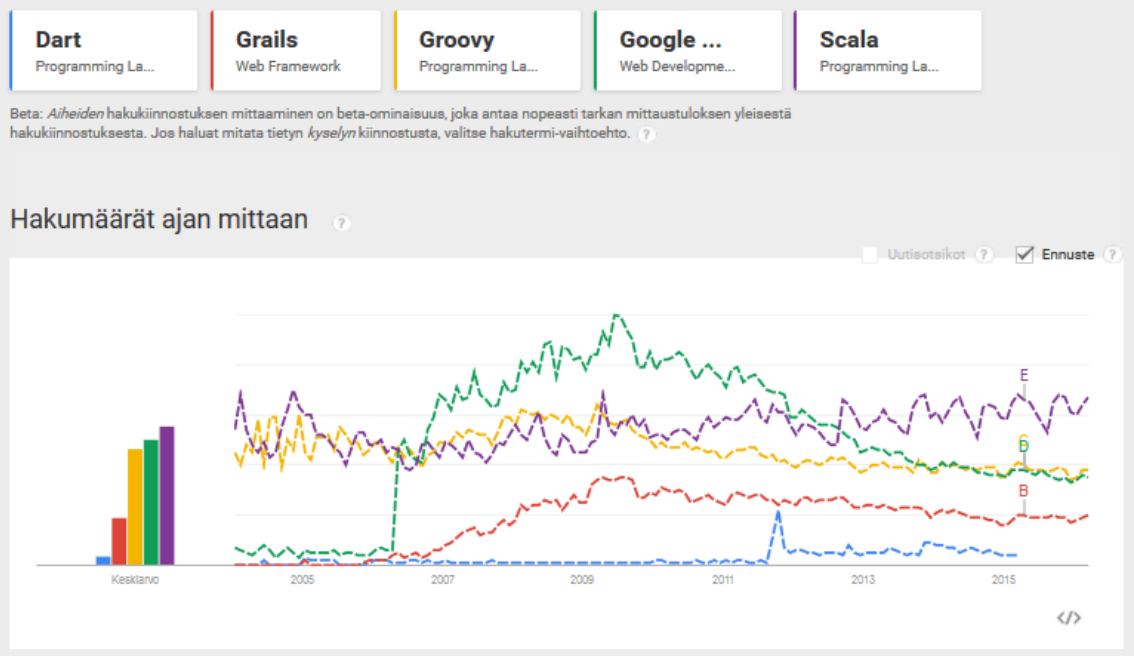
Grails-sovelluskehyksen suosio on pysynyt ennallaan. Play-sovelluskehyksen suosio ei ole kasvanut, vaan pikemminkin pienentynyt. Jostain syystä kuitenkin 47 % Java-sovelluskehittäjistä ilmoittaa haluavansa opetella seuraavana ohjelmointikielenään Scalan (kuvio 6), mikä lisäisi Play-sovelluskehyksen suosiota tulevaisuudessa (White 2014b, 7).



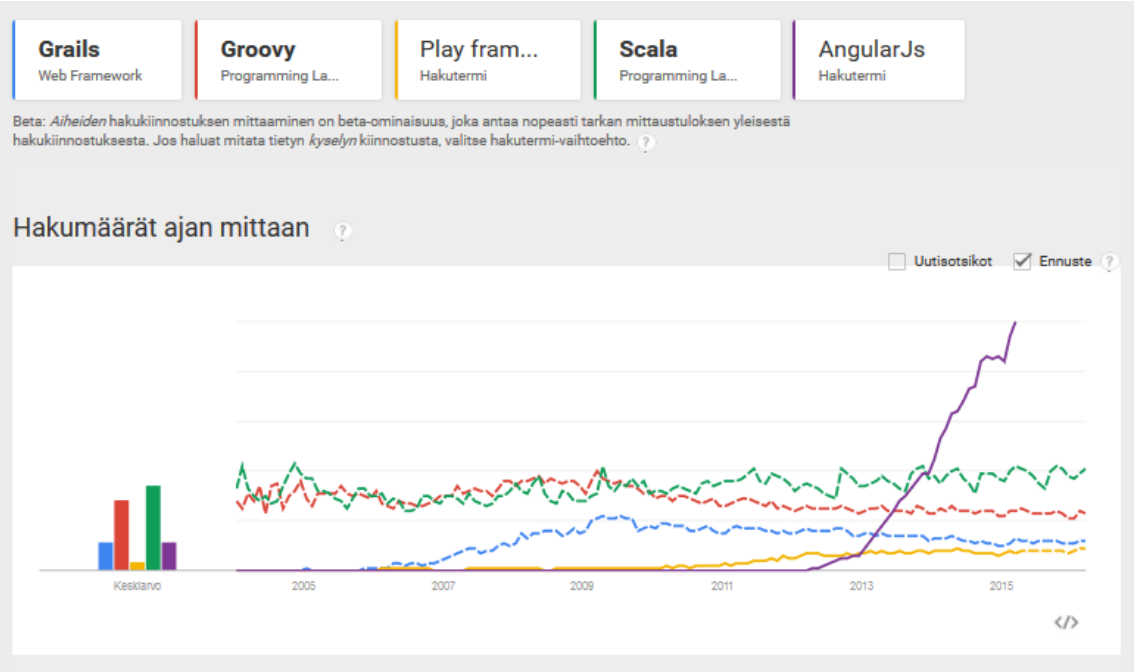
Kuvio 6. Minkä JVM-kielen aiot opiskella (White 2014b, 7).

Grails-sovelluskehyksen tulevaisuutta varjosti tuore uutinen siitä, että sen kehityksen päärahoittaja Pivot päätti lopettaa Grailsin rahoituksen 31.3.2015 mennessä (Grazi 2015). Näin ollen Grails-kehitysprojektin jatkorahoitus oli hetken epävarma. Maaliskuussa Groovy-projekti ilmoitti liittyvänsä osaksi Apache Software Foundationia (Krill 2015a) ja Grails-projektin tekijät siirtyivät Object Computingin (OCI) palvelukseen, joten sovelluskehyksen kehitys näyttäisi jatkuvaan tulevaisuudessakin.

Google Trends -analyysin mukaan Grails oli suosituimmillaan vuonna 2009, jonka jälkeen kasvua ei ole tapahtunut. Groovy-kieli on ollut suunnilleen yhtä suosittu kuin Scala, mutta vuodesta 2011 lähtien Scala on ollut suositumpi. Dart on uusi Googlen kehittämä ohjelmointikieli, jonka on tarkoitus kilpailla JavaScriptin kanssa (kuvio 7). Sen suosio ei ole kuitenkaan kasvanut suuresti toisin kuin AngularJs:n, jonka suosio on voimakkaassa kasvussa (kuvio 8) (Google 2015).



Kuvio 7. Google Trends -käyrät termeille Dart Programming Language, Grails Web Framework, Groovy Programming Language, Google Web Toolkit, Scala Programming Language. Kukin käyrä kertoo, miten ko. teknologian suosio on muuttunut viime vuosina. (Google 2015.)



Kuvio 8. Google Trends -käyrät termeille Grails Web Framework, Groovy Programming Language, Play Framework, Scala Programming Language, AngularJs. Käyristä näkee, miten kiinnostus AngularJs:ää kohtaan on kasvanut voimakkaasti, mutta muiden osalta kiinnostus on pysynyt suunnilleen samana. (Google 2015)

Kokeilin myös Vaadin-termiä haussa ja sen käyrä oli samanlainen kuin Play-frameworkillä. Googlen trendikäyristä voi päätellä, että AngularJs on suosittu, GWT:n suosio on laskussa ja Play- ja Grails-sovelluskehysten suosiossa ei ole tapahtunut suurta muutosta viime aikoina.

Java-kieli yrittää uudistua ja ottaa käyttöön tehokkuutta lisääviä piirteitä muista kielistä. Esimerkiksi uudessa JDK 8 -versiossa on uutena ominaisuutena mm. lambda-ilmaisut ja oletusmetodit. Lambda-ilmaisulla voidaan mm. tehokkaammin ja yksinkertaisemmin tehdä se, mihin ennen tarvittiin nimettömiä luokkia ja toiminnallisia rajapintoja. Javan lambda-ilmaisuilla on samankaltainen syntaksi kuin Scalassa. Näin uusi Java sisältää joitain samoja piirteitä kuin nopean kehityksen kielet Scala ja Groovy. Kuitenkin Javan on vaikeampi kehittyä ja muuttua nopeasti, koska sen käyttäjiä on 9 miljoonaa ja siinä on paljon vanhaa koodia, jonka kanssa on säilytettävä yhteensopivuus. (Benjamin & Martijn 2013, 411.)

Sovelluskehykset kehittyvät joka vuosi ja uusia ilmaantuu. Näin ollen joka vuosi joutuu arvioimaan uudestaan, mikä niistä on paras valinta. Aikaisemmasta tiedosta ja evaluointikokemuksesta on kuitenkin apua, joten aivan alusta ei tarvitse aina aloittaa.

Lähteet

Agrawal, G. 2012. Functional Programming on the JVM. DZone. http://java.dzone.com/articles/functional-programming-jvm. 7.5.2015.

AngularJs.org. 2015. FAQ. https://docs.angularjs.org/misc/faq. 7.5.2015.

Brikman, Y. 2013. What are the pros and cons of the Play Framework 2, for a Java developer? Quora. http://www.quora.com/What-are-the-pros-and-cons-of-the-Play-Framework-2-for-a-Java-developer. 28.4.2015.

Evans, B. & Verburg, M. 2013. Well-Grounded Java Developer. Shelter Island: Manning.

Grazi, V. 2015. Pivotal Pulls Groovy/Grails Funding. InfoQ. http://www.infoq.com/news/2015/01/Pivotal-Pulls-Groovy-Grails-Fund. 1.3.2015.

Google. 2015. Google Trends. www.google.com/trends/ 11.3.2015

Hatami, H. 2012. The curious coders Java web framework comparison. Rebellabs. http://www.slideshare.net/hamedhatami2012/curious-coders-java-web-frameworks-comparison. 4.3.2015.

Krill, P. 2015a. Dumped by Pivotal, Groovy moves to Apache. InfoWorld. http://www.infoworld.com/article/2895752/development-tools/groovy-language-moves-to-apache-dumped-by-pivotal.html.2.4.2015.

Krill, P. 2015b. Pivotal discontinued its support for Grails, but Object Computing has stepped in to hire key Grails developers. InfoWorld. http://www.infoworld.com/article/2909679/java/grails-web-framework-finds-home-at-object-computing.html. 20.4.2015.

Kumar, N. 2013. Groovy & Grails Understanding – Part2. http://techmytalk.com/2013/04/14/groovy-grails-understanding-part2/.

Raible, M. 2014. Comparing JVM Web Frameworks - Raible Designs, http://www.raibledesigns.com/repository/presentations/Comparing_JVM_Web_Frameworks_February2014.pdf. 2.4.2015.

Smith, G., Ledbrook P. 2014. Grails in action. Shelter Island: Manning.

Spock. 2015. https://code.google.com/p/spock/. 11.5.2015.

White, O. 2014a. Java Tools and Technologies Landscape for 2014. Rebel-labs.http://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-for-2014/. 4.3.2015.

White, O. 2014b. Top 4 Java frameworks revealed. Rebellabs. http://zeroturnaround.com/rebellabs/top-4-java-web-frameworks-revealed-real-life-usage-data-of-spring-mvc-vaadin-gwt-and-jsf/. 4.3.2015.

Wikipedia. 2015a. Create, read, update and delete. http://en.wikipedia.org/wiki/Create,_read,_update_and_delete. 11.5.2015.

Wikipedia. 2015b. Java virtual machine. http://en.wikipedia.org/wiki/Java_virtual_machine. 11.5.2015.

Wikipedia. 2015c. MVC-arkkitehtuuri. http://fi.wikipedia.org/wiki/MVC-arkkitehtuuri. 11.5.2015.

Wikipedia. 2015d. Minimum viable product. http://en.wikipedia.org/wiki/Minimum_viable_product. 11.5.2015.

Wikipedia. 2015e. REST. http://fi.wikipedia.org/wiki/REST. 11.5.2015.

Wikipedia. 2015f. SOAP. http://fi.wikipedia.org/wiki/SOAP. 11.5.2015.

Wikipedia. 2015g. WSDL. http://fi.wikipedia.org/wiki/WSDL. 11.5.2015.

Wikipedia. 2015h. Imperatiivinen ohjelmointi. http://fi.wikipedia.org/wiki/Imperatiivinen_ohjelmointi. 27.2.2015.

Wikipedia. 2015j. Bootstrap (front-end framework). http://en.wikipedia.org/wiki/Bootstrap_(front-end_framework) 29. 6.5.2015.

Wikipedia. 2015i. Vaadin. http://en.wikipedia.org/wiki/Vaadin. 6.5.2015.