TAMPEREEN
AMMATTIKORKEAKOULU

HRD-TUOTTEEN JA -TYÖMENETELMIEN KEHITTÄMINEN

Sami Hellsten

Opinnäytetyö
Huhtikuu 2018
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

HELLSTEN, SAMI:
HRD-tuotteen ja -työmenetelmien kehittäminen

Opinnäytetyö 67 sivua, joista liitteitä 14 sivua
Huhtikuu 2018

Opinnäytetyössä perehdytään Drupal 8 -ohjelmistokehitykseen käsittäen jatkuvan kehityksen kannalta oleellisimmat työkalut ja työskentelymenetelmät. Opinnäytetyön taustalla on Mediamaisteri Oy:n osaamisen kehittämiseen ja koulutusten organisointiin suunnattu HRD-ohjelmisto, jonka kehittämisessä Drupal 8 -versiolle opinnäytetyön tekijä on merkittävässä roolissa.

Työssä painotetaan Drupalin backend-kehitystä sekä erityisesti niitä asioita, joita Drupal 7 -ohjelmistokehitystä tehnyt henkilö saattaisi tehdä tavoilla, jotka eivät ole tarkoituksenmukaisia oikeaoppisessa Drupal 8 -kehityksessä. Lisäksi työssä esitetään sellaisia DevOps-puolen asioita, jotka ohjelmistokehittäjän on hyvä tietää helpottaakseen niin omaansa kuin oman työryhmänsä työskentelyä.

HRD on ohjelmistotuote, jota myydään muokattavana ja räätälöitävänä asiakkaan tarpeisiin. Työssä esitetään teoreettisen asiakasvaatimuksen toteuttaminen siten, että on mahdollista ymmärtää, miksi kyseiset työvaiheet on syytä toteuttaa tarvetta varten tehtävässä ohjelmistokoodissa. Toteutettu ohjelmakoodi on myös työn liitteenä, jossa on lisäksi nähtävissä myös ne osat ohjelmakoodia, jotka varsinaisesta opinnäytetyöraportista on tilan säästämiseksi jätetty pois.

Asiasanat: drupal 8, drupal, php, backend, devops, sisällönhallintajärjestelmä, cms

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

HELLSTEN, SAMI:
Developing HRD Product and Working Methods

Bachelor's thesis 67 pages, appendices 14 pages
April 2018

The purpose of this thesis is to introduce Drupal 8 development for the reader, including the most essential tools and working methods for continuous development. The background of this thesis is the human resource development and training organizing software HRD, which is product of the Mediamaisteri Oy. Author of this thesis has been in major role on the development of Drupal 8 version of HRD.

Thesis emphasizes Drupal's backend development, and in particular the issues which developer with Drupal 7 knowledge might do in a way that is not meant to be done in Drupal 8 development. Thesis also includes some of the things about DevOps that developer should know to facilitate their own work, as well as to facilitate the work of the working group.

HRD is a software product that is sold customizable as well as tailored to the needs of the customer. This thesis goes through the development of solution for theoretical customer need so that the reader can understand why these work steps should be implemented in a software when it's being developed. The complete code is also attached to the thesis, where reader can also see the parts of the program code that was not included to the actual thesis to save space.

Key words: drupal 8, drupal, php, backend, devops, content management system, cms

SISÄLLYS

LYHENTEET JA TERMIT

Backend	Se osuus ohjelmiston toimintaa, jolla ei ole graafista käyttöliittymää.
Bugi	Ohjelmointivirhe.
CMS	Content Management System, sisällönhallintajärjestelmä.
Coder	Drupal-moduuli ohjelmakoodin standardointiin.
Composer	Komentorivityökalu, pakettienhallintajärjestelmä.
DevOps	Ohjelmiston ja tuotannon väliset operaatiot.
Docker	Virtualisointialusta.
Drupal	Avoimen lähdekoodin sisällönhallintajärjestelmä.
Drupal Console	Komentorivityökalu Drupal-kehitykseen.
Drush	Komentorivityökalu Drupal-kehitykseen.
Facets	Drupal-moduuli.
GitLab	Projektienhallintajärjestelmä.
HRD	Human Resource Development, henkilöstön resurssien kehitys- ja hallintatyökalu.
Interface	Olioluokan esittelytiedosto.
MaisteriLMS	Verkko-oppimisympäristö.
Pipeline	Sarja automatisoituja työvaiheita.
Skripti	Lyhyt ohjelmakoodin osa, joka voidaan suorittaa ilman kääntämistä. Tässä yhteydessä painotus lyhyessä, sillä PHP on ohjelmointikieli, jota ei erikseen käännetä, vaan se tulkataan suorituksen aikana.
State	Drupal-ytimen tarjoama tilapalvelu.

.

1 JOHDANTO

Tämän opinnäytetyön tavoitteena oli tiivistää tuotepohjaisen Drupal 8 -pohjaisen sovelluksen työnkulku siten, että se minimoisi ohjelmistokehittäjän kynnystä siirtyä Drupal 7 -kehityksestä Drupal 8 -kehitykseen. Siirtyminen tarkoittaa funktionaalisen ohjelmoinnin korvaamista lähes kokonaisuudessaan olio-ohjelmoinnilla. Lisätavoitteena on antaa lukijalle ymmärrys Drupal 8 -pohjaisesta ohjelmistokehityksestä ilman kokemusta CMS-julkaisujärjestelmistä (Content Management System). Drupal on avoimen lähdekoodin CMS, jota käytetään runkona useissa sivustoissa sekä palveluissa.

Ensimmäiset Drupal 8 -pohjaiset HRD-toteutukset (Human Resource Development) tehtiin ilman tuotelähtöistä suunnittelua mukaillen vanhempien versioiden toteutuksia. Kolmannen Drupal 8 -projektin kohdalla projektien yhtenäistäminen nousi kuitenkin ensisijaiseksi, ja tällöin olemassa olevien projektien ohjelmakoodien yhdisteleminen aloitettiin.

Työn taustalla on kehitysprojekti, joka alkoi kesäkuussa 2016 ja on aktiivisessa kehityksessä edelleen. Tässä opinnäytetyössä ei kuitenkaan paneuduta erikseen jo tehtyjen toiminnallisuuksien toteutukseen, sen sijaan tässä esitetään teoreettisen ja yksinkertaisen tarpeen mukainen toteutus, jota olisi kuitenkin mahdollista tehokkaasti soveltaa oikeassa käyttötarkoituksessa.

Työssä päädyttiin myös kehittämään HRD:n työnkulkua, jotta ohjelmistokehityksestä saatiin virtaviivaisempaa sekä tehokkaampaa. Työn viimeisessä luvussa esitetään vielä erikseen pohdintaa Drupal 8 -pohjaisen tuotteen suurimmista ongelmakohdista ja mahdollisuuksista.

2 HRD-TUOTE

HRD on työkalu, joka tarjoaa ratkaisun henkilöstön osaamisen kartoittamiselle, seurannalle ja raportoinnille. Sen avulla on myös mahdollista tarjota henkilöstölle vaadittuja, suositeltuja tai vapaaehtoisia verkko-, luokkahuone- sekä monimuotokoulutuksia. Koulutuksia on mahdollista myös rajata eri henkilöstöryhmille esimerkiksi työtehtävien sekä toimipisteen mukaan.

HRD mahdollistaa myös osaamispolut, jolloin on mahdollista, että osaamismerkinnän voi suorittaa esimerkiksi yksittäisellä koulutuksella tai se voi vaatia useamman koulutuksen suorittamisen. Lisäksi koulutuksia voidaan määritellä tarjottavaksi vasta, kun jokin aiempi koulutus on suoritettu hyväksytysti.

HRD voidaan myös integroida muihin järjestelmiin, jolloin järjestelmään on mahdollista tuoda esimerkiksi käyttäjätiedot, koulutukset, osaamismerkinnät ja -polut automatisoidusti. Useimmiten HRD tarjotaan yhdessä Moodleen pohjautuvan MaisteriLMS:n kanssa, jolloin koulutusten ylläpitäminen, käyttäjätuonti ja kirjautuminen järjestelmään toimivat MaisteriLMS:n kautta. HRD on kuitenkin mahdollista integroida käytännössä mihin tahansa järjestelmään, joka tarjoaa riittävän rajapinnan tiedon kulkemiselle.

Henkilöstölle HRD tarjoaa helpon tavan etsiä itseään kiinnostavia tai itselleen pakollisia koulutuksia sekä tarkastaa ja selata omia meneillään olevia koulutuksia. Lisäksi henkilö voi kätevästi seurata HRD:n avulla omaa koulutushistoriaansa, osaamismerkintöjään ja sertifikaattejaan sekä näiden mahdollista uusimistarvetta.

Esimiehille HRD tarjoaa mahdollisuuden ilmoittaa alaisiaan näitä koskeviin koulutuksiin esimerkiksi niissä tapauksissa, ettei henkilöstöllä ole pääsyä järjestelmään esimerkiksi niissä tilanteissa, kun alaisella ei ole mahdollisuutta käyttää tietokonetta työajalla tai alaiselta puuttuu riittävä tietotekninen osaaminen. Lisäksi esimiehille HRD tarjoaa helpon näkymän tarkastaa, ketkä alaisistaan ovat suorittaneet kunkin vaaditun koulutuksen tai ovatko alaiselta vaaditut sertifikaatit edelleen voimassa.

Kouluttajalle HRD mahdollistaa helpon tavan seurata osallistujamääriä, ilmoittaa osallistujille koulutukseen liittyvistä muutoksista, merkitä osallistujien paikallaolo koulutuksesta ja esimerkiksi perua koulutuksen sekä tarjota tilalle vastaavaa koulutusta. Kouluttajan on myös helppo tarkastaa kulloinkin meneillään olevat koulutukset, joissa on itse kouluttajana.

Henkilöstöhallinnolle HRD tarjoaa monipuoliset raportit koko henkilöstön osaamisista, vaadittujen koulutusten suorituksista tai suoritusten puutteesta sekä koulutusten kustannuksista veroraportteineen. Lisäksi se mahdollistaa koulutushistorian luonnin jälkikäteen.

HRD on suunnattu lähtökohtaisesti yrityksille, joilla on tarve kehittää henkilöstönsä osaamista ja seurata henkilöstön osaamistasoa. Pääasiallinen kohderyhmä HRD-tuotteelle ovat keskisuuret ja suuret yritykset, mutta raporttiensa ansiosta HRD on taloudellinen vaihtoehto myös pienille yrityksille. Tuotepohjaisena ratkaisuna HRD löytää paikkansa yrityksen käytöstä ilman pienintäkään räätälöintiä, joten se tarjoaa kustannustehokkaan ratkaisun yrityksille, jotka ovat kiinnostuneet omasta sekä henkilöstönsä osaamisesta ja sertifioinneista.

Tuotepohjainen HRD on tarkoitettu yrityksen oman henkilöstön osaamisen ja koulutuksien hallinnointiin, mutta koska tuote on rakennettu räätälöitäväksi, on yrityksen mahdollista kohtuullisella räätälöinnillä ottaa HRD käyttöön tarjotakseen ja markkinoidakseen koulutuksia myös ulkopuolisille yrityksille ja henkilöille. Lopullinen käyttötarkoitus HRD-tuotteelle on aina asiakaskohtainen ja mahdollisen räätälöinnin osuus riippuu asiakkaan tarpeista.

3 TYÖKALUT JA TEKNOLOGIAT

Soveltuvien työkalujen ja teknologioiden valintaan vaikuttivat erityisesti niiden monipuolisuus, pohjautuminen avoimeen lähdekoodiin, tietoturvallisuus sekä niiden jatkuva kehitys. Tarkoituksena ei kuitenkaan ole lisätä työmäärää sellaisilla järjestelmillä, jotka vaatisivat huomattavan määrän työtunteja järjestelmän ylläpitoon.

Alla on listattuna työkaluja ja teknologioita, jotka ovat nousseet HRD:n kehityksessä keskeiseen rooliin ohjelmoijan näkökannasta.

3.1 GitLab-projektienhallintajärjestelmä

GitLab on sovelluskehitykseen käytettävä alusta, joka mahdollistaa projektien ylläpitämisen esimerkiksi tarjoamalla lähdekoodien ja muiden ohjelmalle keskeisten tiedostojen keskitetyn tallennuspaikan ja versioinnin. Lisäksi GitLab tarjoaa hyvät työkalut projektin hallinnointiin ja dokumentointiin.

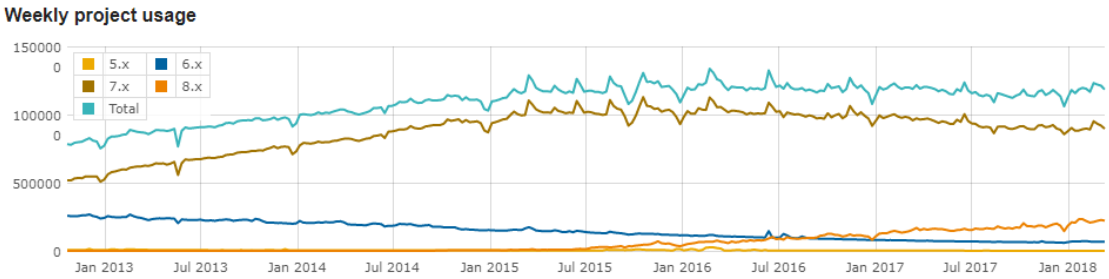
GitLabia on mahdollista käyttää suoraan GitLabin tarjoamana palveluna, jolloin riittää, kun käyttäjä rekisteröityy palveluun, yksittäinen käyttäjä voi palveluun rekisteröitymällä luoda rajattoman määrän julkisia ja yksityisiä projekteja. Toinen vaihtoehto on ylläpitää omaa versiota GitLabista, jolloin ohjelmistokehitys ja projektit sekä niihin liittyvä tietojen jakaminen voidaan pitää tehokkaammin omissa käsissä.

GitLabista on kaksi versiota, GitLab Community Edition (CE) on kaikille ilmainen, avoimen lähdekoodin alainen versio ja GitLab Enterprise Edition (EE) on maksullinen, yli sadan käyttäjän organisaatioille suunnattu versio. Muita vastaavia järjestelmiä ovat esimerkiksi GitHub sekä Bitbucket. Mediamaisteri on valinnut näistä vaihtoehdoista GitLab Community Edition (CE):n sen tarjotessa kaikki ohjelmistokehityksen hallinnointiin oleelliset työkalut avoimen lähdekoodin muodossa.

3.2 Drupal-sisällönhallintajärjestelmä

Drupal on avoimen lähdekoodin sisällönhallintajärjestelmä, joka mahdollistaa niin pienten, suurten kuin monipuolistenkin sivustojen luomisen ilman, että tekijän tarvitsee kirjoittaa lainkaan ohjelmakoodia. Drupalin kehityksestä vastaa pääasiassa Drupalin oma yhteisö, johon kuka tahansa voi liittyä. Toisin kuin aiemmat versiot, Drupal 8 on Symfony-pohjainen.

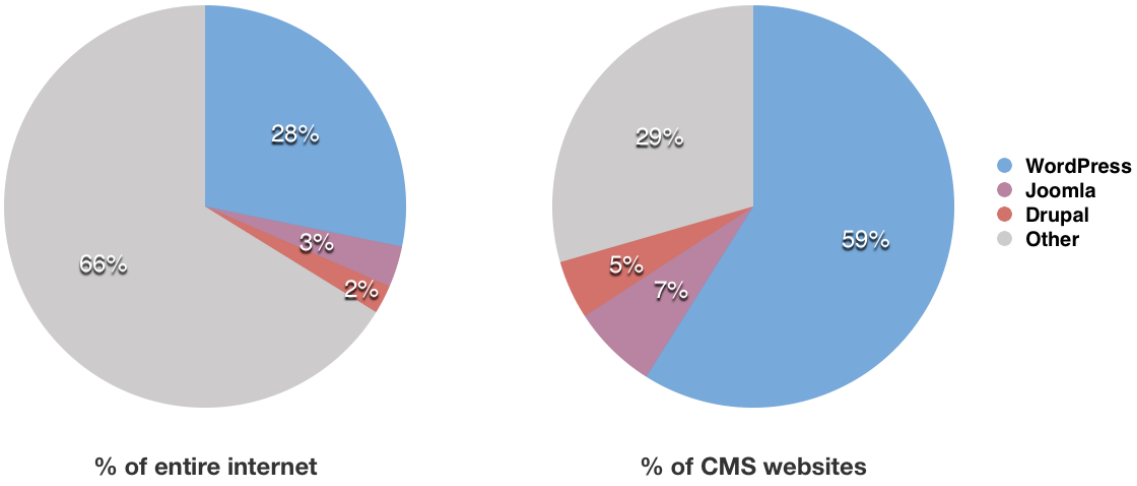
Drupal tarjoaa jo oletuksena laajat työkalut sivuston kehittämiseen, mutta tarvittaessa yhteisöstä löytyy vapaaehtoisten tekemiä moduuleja, joilla omaa sivustoaan voi laajentaa. Yksittäinen moduuli voi tarjota vaikkapa pelkän kentän tai kokonaisen sivuston. Monet vanhempien versioiden suosituimmista lisämoduuleista on sulautettu uusimpaan Drupalin ytimeen, kuten Entity Reference sekä Views.



Week	5.x	6.x	7.x	8.x	Total
March 11, 2018	873	65,735	895,870	221,050	1,183,528
March 4, 2018	898	65,885	918,426	226,384	1,211,593
February 25, 2018	896	66,529	932,138	219,998	1,219,561

KUVA 1: Drupal-ytimen versioiden käyttötilastot (Drupal 2018)

Alkunsa Drupal on saanut jo vuonna 2000, ja sen nykyinen ydinversio, Drupal 8, on julkaistu marraskuussa 2015. Nykyistä versiota edeltävä Drupal 7 on kuitenkin aktiivisessa kehityksessä ja yleisesti käytössä. Suurin osa sivustoista pyörivät edelleen vanhemmilla versioilla (kuva 1). Nykyisen version verkkaista käyttöönottotahtia osaltaan selittää se, että moni Drupal 7:n suosituimmista moduuleista on yhä varhaisessa kehitysvaiheessa Drupal 8:lle, eikä niistä täten ole tarjolla vakaita versioita. Osaltaan myös hidasta tahtia selittää se, että Drupal 8 on suurin muutos koko Drupalin historiassa. Vanhempien Drupalien ollessa enemmän funktionaalisia PHP-ohjelmistokehyksiä, on Drupal 8 lähes puhtaasti olio-ohjelmointia.



KUVA 2: Drupal verrattuna muihin alustoihin (Briteweb 2018)

Vastaavia järjestelmiä kuin Drupal ovat esimerkiksi Joomla sekä WordPress, näistä Drupal on mielletty vaikeimpana, mutta myös tehokkaimpana. Osaltaan juurikin helppous selittää, miksi Drupal ei välttämättä ole pienempien sivustojen valinta (kuva 2). HRD:n rungoksi Drupal valikoitui juuri sen tehokkuuden ja muokattavuuden vuoksi.

3.3 Docker-virtualisointialusta

Ohjelmistokehityksessä on tärkeää, että oma kehitysympäristö vastaa mahdollisimman tarkasti sitä oikeaa ympäristöä, jossa ohjelma lopullisessa muodossaan tulee pyörimään. Tällöin voidaan käytettyjen ohjelmistojen ja niiden versioiden yhteensopivuus varmistaa jo kehitysvaiheessa ennaltaehkäisten täten tuotannossa mahdollisesti ilmeneviä yllätyksiä, kun jokin asia ei toimikaan, kuten sen on odotettu toimivan.

HRD:n osalta toimivuuden varmistaminen on erityisen tärkeää, sillä sen sisältämät tiedot on tarkoitettu nähtäväksi vain asianomaisille, eikä vapaata pääsyä järjestelmään yleisesti ottaen ole. HRD on usein myös integroitu muihin järjestelmiin, joten toimivuudesta molempiin suuntiin on oltava varmuus.

Ratkaisuna tälle on kehityksessä valittu Docker, joka tarjoaa mahdollisuuden virtualisoida kehitysympäristö hyvin lähelle tuotantoympäristöä, ja sen julkisesta kirjastosta löytyy laaja valikoima valmiita kehitysympäristöjä. Myös Mediamaisterin käyttämä Drupal-kehitysympäristö on julkisesti saatavilla Docker Hubista.

Dockerin skaalautumiskyvyn myötä Dockeria on tarkoitus käyttää myöhemmässä vaiheessa myös tuotantokäytössä, sillä toimiessaan yksittäisenä virtuaaliympäristönä se poistaa eston päivittää ohjelmistoversioita niistä ympäristöistä, joissa päivittäminen on mahdollista. Ilman Dockeria tai muuta virtualisointia on esimerkiksi PHP:n versio hyvin usein paljon toivottua vanhempi, sillä palvelin sisältää jonkin yksittäisen ohjelmiston, joka ei tue uudempaa versiota.

3.4 Composer-työkalu

Yksittäisessä HRD-projektissa voi olla käytössä useita kymmeniä Drupal-moduuleja tai muita PHP-liitännäisiä, joista osa on samoja kuin muissa HRD-projekteissa ja osa taasen sellaisia, ettei niitä ole missään muussa projektissa. Vaikka jokaista moduulia ja liitännäistä kehitetään jatkuvasti, on niiden kehittymisvauhti täysin yksilöllistä. Tuotantokäytössä olevan asiakasprojektin tietoturvallinen ylläpitäminen vaatii, että kaikki siinä käytetyt kirjastot ovat ajan tasalla. Tällainen seurantatarve olisi lähes mahdotonta järjellisesti toteuttaa ilman siihen soveltuvaa työkalua.

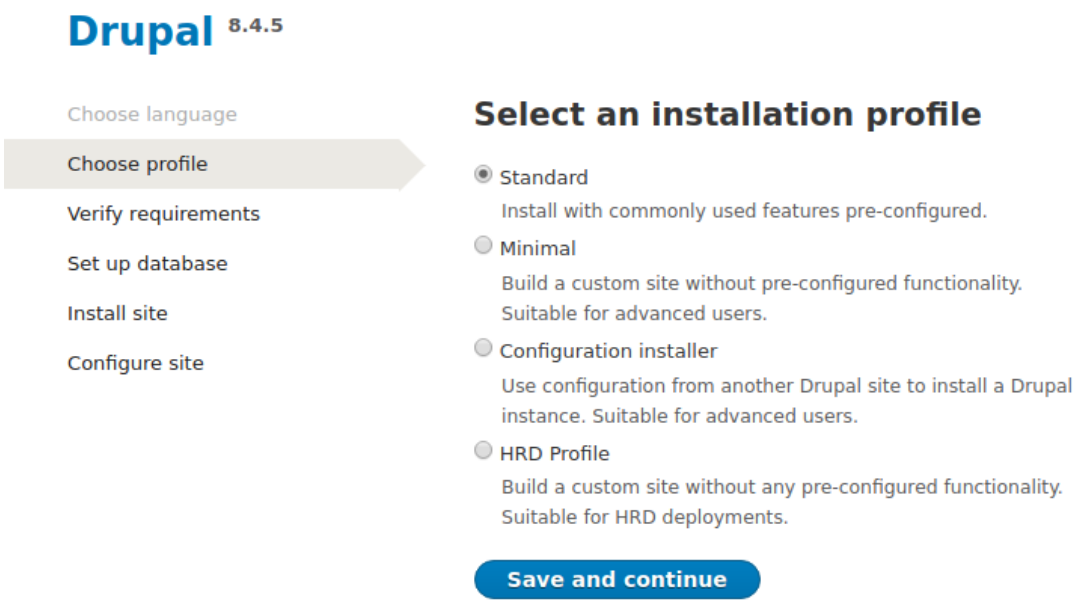
Composer on tähän tarkoitukseen tehty pakettienhallintajärjestelmä, se tarjoaa helpon tavan tarkistaa nykyiset käytössä olevat versiot ja niiden mahdolliset päivitystarpeet. Composer hallinnoi projektin ulkopuolisia riippuvuuksia kahden tiedoston avulla, joista toinen kertoo, mitä paketteja on tarkoitus käyttää, ja toinen kertoo taasen, mitkä versiot kustakin paketista on käytössä. Näiden kahden tiedoston avulla projektin versiohallintaan ei tarvitse sisällyttää yhtään projektin ulkopuolisen liitännäisen tiedostoja, vaan kukin ohjelmistokehittäjä ja käyttöympäristö voidaan asentaa täsmälleen samaan tilaan Composerin avulla.

Yksi Composerin tarjoamista eduista ohjelmistokehittäjän osalta on sen tarjoama mahdollisuus lisätä riippuvuuksia pelkästään kehitysympäristötasolle, jolloin tuotantoon ei ladata vastaavia paketteja lainkaan. Vastaavan edun Composer tarjoaa sen kyvyllä etsiä käytettävien pakettien keskinäisiä riippuvuuksia. Mikäli esimerkiksi projekti ajetaan ympäristössä, jossa on käytössä PHP 5.6, voidaan tämä määrittää Composerin asennustiedostoon, jolloin se ei päivitä mitään käytettyä pakettia sellaiseen versioon, mikä vaatisi PHP 7:n käyttöä.

Vaikka Composer on pakettienhallintajärjestelmä, se ei kuitenkaan ole suoranaisesti sama asia kuin Debian-pohjaisissa käyttöjärjestelmissä käytettävä APT tai Red Hat -pohjaisissa käyttöjärjestelmissä käytettävä YUM. Erona varsinaiseen pakettienhallintajärjestelmään Composer lataa paketit yksittäisen projektin alle, eikä koko käyttöjärjestelmän käytettäväksi, vaikkakin Composer tukee myös projektin ulkopuolisia asennuksia. Suoraan verrannollisia sovelluksia ovat JavaScript-kirjastoille käytettävä npm ja Ruby-kirjastoille käytettävä Bundler.

3.5 Drush-työkalu

Drush on komentorivityökalu, joka on kehitetty Drupalia varten. Se on suunnattu erityisesti ohjelmistokehittäjille ja sivustojen ylläpitäjille. Drush tarjoaa jo lähtökohtaisestikin runsaasti erilaisia toimintoja, mutta samalla se tarjoaa myös rajapinnan, jota Drupal moduulit voivat hyödyntää tarjoten Drushin kautta omia toimintojaan.



KUVA 3: Drupalin asennuksen oletuskäyttöliittymä

Esimerkiksi jo asennusvaiheessa, kun Drupal vasta otetaan käyttöön, on asiansa tuntevalle ohjelmistokehittäjälle Drushista suuri hyöty, sillä Drupalin omassa käyttöliittymässä on monia välivaiheita, joista suuri osa on kehittäjän silmissä tarpeettomia. Kuvan 3 mukaisen kuusiosaisen asennusvaiheen voi kätevästi suorittaa yhdellä komennolla komentoriviltä (komentorivi 1).

```
$ drush si --site-name="My Site" --db-url="mysql://root:root@db/drupal"
You are about to DROP all tables in your 'drupal' database.
Do you want to continue? (y/n): y
Starting Drupal installation. This takes a while. Consider using the [ok]
--notify global option.
Installation complete.  User name: admin  User password: pWBzWRpFVa  [ok]
Congratulations, you installed Drupal!                              [status]
```

KOMENTORIVI 1: Drupalin asentaminen komentoriviltä

Drush tarjoaa myös erityisen helpon ratkaisun sivuston asetusten viemiselle ja tuomiselle, jolloin ohjelmistokehittäjän on mahdollista tehdä muutokset omassa kehitysympäristössään, viedä ne sieltä versionhallintaan ja sieltä edelleen muille kehittäjille tai tuotantoon. Koska Drush on komentorivityökalu, se samalla mahdollistaa sen käyttämisen helposti esimerkiksi automaattisissa testeissä sekä automaattisessa tuotannon päivityksessä. Esimerkiksi muuttuneiden asetusten käyttöönottaminen onnistuu komentorivi 2:n mukaisella komennolla.

```
$ drush cim
 Collection  Config                          Operation
             hrd_messages.language_settings  create
Import the listed configuration changes? (y/n): y
Synchronized configuration: create hrd_messages.language_settings.   [ok]
Finalizing configuration synchronization.                            [ok]
The configuration was imported successfully.                         [suc-
cess]
```

KOMENTORIVI 2: Asetusten tuominen tiedostosta Drupal-järjestelmään

3.6 Drupal console -työkalu

Drupal Console on Drupalia varten kehitetty työkalu, joka helpottaa erityisesti ohjelmistokehittäjien arkea tarjoten työkalut moniin eri tilanteisiin ja tarpeisiin. Ilman Drupal Consolea esimerkiksi Drupal-moduulin luominen on hidasta, sillä vaikka moduulin teoreettiseen luomiseen riittää yksi tiedosto, toiminnallinen moduuli tarvitsee huomattavan määrän ohjelmakoodia, jotta Drupal tunnistaa sieltä halutut asiat.

HRD:n kehittämisessä Drupal Console on tärkeässä roolissa, sillä sen avulla ohjelmistokehittäjän ei tarvitse erikseen muistaa, mitkä kaikki tiedostot tai ohjelmakoodit moduulista täytyy löytyä, ennen kuin se on täysin yhteensopiva koko muun järjestelmän kanssa. Drupal Consolen avulla ohjelmistokehittäjä voi keskittyä suoraan tekemään ohjelmakoodia niihin osiin, joihin halutaan muutoksia.

3.7 Coder-moduuli

Drupalille on olemassa oma standardi ohjelmakoodia varten, jota noudattelemalla koodi säilyy helposti luettavana ja yhtäläisenä riippumatta siitä, kuka sen on tehnyt. Standardi sisältää määritykset niin sisennyksille, koodin muotoilulle kuin erilaisille käyttötavoillekin.

Drupal Coder on Drupalista riippumaton Drupal-moduuli, joka tarjoaa Drupalin standardit kahtena erillisenä lisäosana PHP CodeSniffer -nimiseen ohjelmaan. PHP CodeSniffer on avoimen lähdekoodin työkalu, jolla on mahdollista tarkastaa ohjelmakoodin - myös muun kuin PHP:n - standardinmukaisuus.

Ohjelmakoodin standardointi voi tuntua alkuun raskaalta ja hankalalta, sillä esimerkiksi ohjelmakoodi 1:n mukainen - täysin toimiva - ohjelmakoodi on suoritettavissa, eikä siitä aiheudu välttämättä missään vaiheessa virheitä tai virheilmoituksia.

```
<?php

namespace Drupal\my_module;
use Drupal\Core\Controller\ControllerBase;
use Drupal\Core\DependencyInjection\ContainerInjectionInterface;

interface myInterface extends ContainerInjectionInterface {}

class myBaseClass extends ControllerBase implements myInterface {}

class myClass extends myBaseClass {

  function myFunction () {
    $user = \Drupal::currentUser()->getAccountName();
    return $user;
  }
}
$class = new myClass(); $echo = $class->myFunction();
echo $echo;
```

OHJELMAKOODI 1: Esimerkki toimivasta, ei-standardinmukaisesta ohjelmakoodista

Mikäli ohjelmakoodi 1:n ajaa Drupal Coderin läpi (komentorivi 3) tarkistaakseen mahdollisten virheiden varalta, on edellä mainitussa ohjelmakoodissa hyvinkin paljon parantamisen varaa. On syytä ottaa huomioon, että vaikka kaikki komentorivi 3:ssa mainitut virheet korjattaisiin edellä mainittuun ohjelmakoodiin, sisältäisi se silti monia sellaisia

piirteitä, joita yleisesti ei pidetä hyvien ohjelmointitapojen mukaisina, kuten esimerkiksi se, että vain yksi luokka tai luokan esittely tulisi olla edustettuna yhdessä tiedostossa.

```
$ drupalcs modules/custom/my_module/src/myClass.php

FILE: modules/custom/my_module/src/myClass.php
----------------------------------------------------------------------
FOUND 13 ERRORS AFFECTING 8 LINES
----------------------------------------------------------------------
  3 | ERROR | [x] There must be one blank line after the namespace
    |       |     declaration
  7 | ERROR | [ ] Interface name must begin with a capital letter
  7 | ERROR | [x] Missing interface doc comment
  9 | ERROR | [ ] Class name must begin with a capital letter
  9 | ERROR | [x] Missing class doc comment
 11 | ERROR | [ ] Class name must begin with a capital letter
 11 | ERROR | [x] Missing class doc comment
 13 | ERROR | [x] Visibility must be declared on method "myFunction"
 13 | ERROR | [x] Missing function doc comment
 13 | ERROR | [x] Expected 0 spaces before opening parenthesis; 1
    |       |     found
 16 | ERROR | [x] Expected 1 blank line after function; 0 found
 17 | ERROR | [x] The closing brace for the class must have an empty
    |       |     line before it
 19 | ERROR | [x] Expected 1 newline at end of file; 0 found
----------------------------------------------------------------------
```

KOMENTORIVI 3: Ohjelmakoodi 1:n standardinmukaisuus testattuna

4 KEHITYSTARVE

Työn aloitusvaiheessa HRD oli ollut olemassa jo vuosia. Ylläpidossa olleet toteutukset oli päivitetty Drupal 7 -pohjaisiksi, joista vanhimmat olivat HRD 3 -pohjaisia ja uudemmat HRD 4 -pohjaisia. Näiden kahden välillä ei vielä ollut suuria eroja, mutta HRD:n siirtyessä Drupal 8:lle jo yksittäisen ison versionumeron sisällä alkoi olla isoja eroavaisuuksia.

Drupal 8 -versio HRD:stä oli alkujaan ohjelmallisesti muunnettu Drupal 7 -versiosta oletetun jatkokehityksen helpottamiseksi. Loppuosa muunnosta oli tarkoitus tehdä aiempien HRD 4 -toteutuksien mukaisesti, mutta kehityksessä siirryttiin nopeasti tuotepohjaiseen ajattelumalliin.

Heti alussa oli selvää, että versionumerointi tulisi muuttaa käyttämään niin sanottua semanttista versionumerointia, jolla tarkoitetaan versionumeroiden kasvattamista siten, että versionumero jaetaan kolmeen pisteellä erotettuun osaan, josta viimeinen, vähiten merkitsevä numero tarkoittaa virhekorjausta. Keskimmäinen numero tarkoittaa uutta toiminnallisuutta, mutta taaksepäin yhteensopivaa versioita. Vasemmanpuoleisin, merkitsevin numero tarkoittaa isoja, ohjelmistoarkkitehtuurillisia muutoksia, jotka eivät enää ole taaksepäin yhteensopivia. HRD siirtyi myöhemmin ison arkkitehtuurimuutoksen johdosta semanttiseen versionumerointiin, jolloin julkaistiin HRD 5.0.0, nykyisen version ollessa 5.4.1 kirjoitushetkellä.

4.1 Ylläpidettävän ohjelmakoodin määrä

Yksi suurimpia kehitystarpeita oli ylläpidettävän koodin määrä, sillä jokainen käytössä oleva projekti vaatii ylläpitoa aina jossain määrin. Ylläpitoon varattu henkilötyötuntien määrä on kuitenkin rajallinen, joten kaikki niin sanottuun turhaan työhön käytetty aika on pois jostain muualta, tässä tapauksessa yleensä jatkokehityksestä.

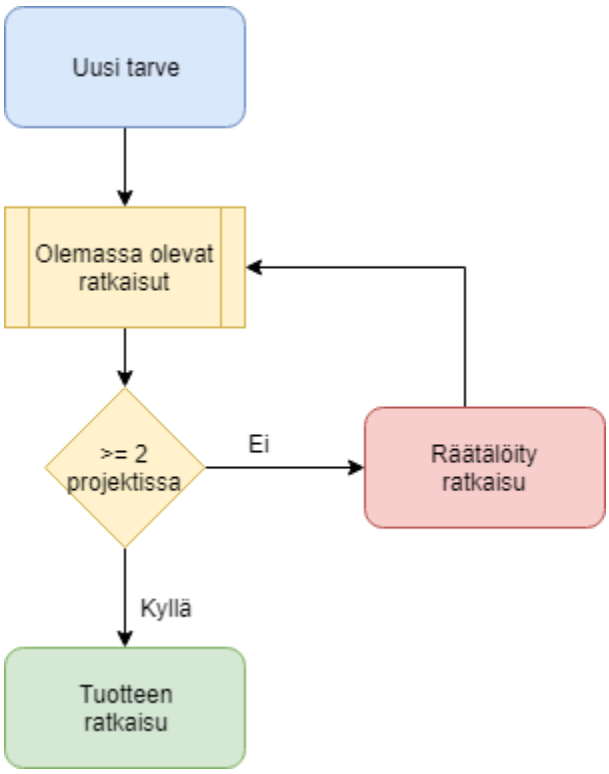
Ensimmäinen vaihe ylläpidettävän koodin määrään puuttumiseksi oli ohjelmakoodin radikaali vähentäminen tuotteistamalla ohjelmisto. Käytännössä se tarkoitti kaikkien projektien käytössä olevien toiminnallisuuksien listaamista ja vertailua, mikäli sama toiminnallisuus oli käytössä kahdessa tai useammassa projektissa. Mitä useammassa projektissa

näennäisesti samanlainen toiminto oli käytössä, sitä oleellisemmaksi nousi sen konkreettinen ymmärtäminen, sillä vaikka toiminto näennäisesti olisikin sama, saattoi sen käyttötarkoituksissa olla eroavaisuuksia, ja täten yhdistetyn ohjelmakoodin piti pystyä palvelemaan kaikkia mahdollisia käyttötarkoituksia.

Toisena vaiheena ylläpidettävän koodin määrän vähentämiseksi otettiin linjaukseksi, että kaikkien projektien pitää pystyä noudattamaan ainakin suurimmaksi osaksi samaa päivityspolkua, jolloin yksittäisen projektin päivittäminen uusimpaan versioon ei vaatisi erikseen perehtymistä kyseiseen projektiin. Lisäksi tämä tarkoitti sitä, että jatkossa kaikki projektit pyritään pitämään uusimmassa mahdollisessa versionumerossa, jolloin mahdollisen ohjelmavirheen paljastuessa ei tarvitsisi tehdä korjausta kuin yhteen versioon.

4.2 Tuotelähtöisyys

Ennen HRD 4 -version aloittamista ohjelman kehitys oli pohjautunut puhtaasti asiakkaan määrittelemiin tarpeisiin, projektit olivatkin käytännössä aina erikseen kyseistä asiakasta varten tehtyjä tilaustöitä. Käyttöönottojen lisääntyessä tämä ratkaisu ei kuitenkaan ole kestävä, sillä vaikka osaaminen säilyy projektista toiseen ja täten nopeuttaa vastaavan toteutuksen uudelleentekemistä, ei toisistaan huomattavasti eriävien projektien ylläpitäminen ole nopeaa. Tämä vaatii kehittäjältä aina oman aikansa perehtyä projektin määrittelyihin, jos ohjelmaan tarvitsee tehdä esimerkiksi korjaus tai tietoturvapäivitys.



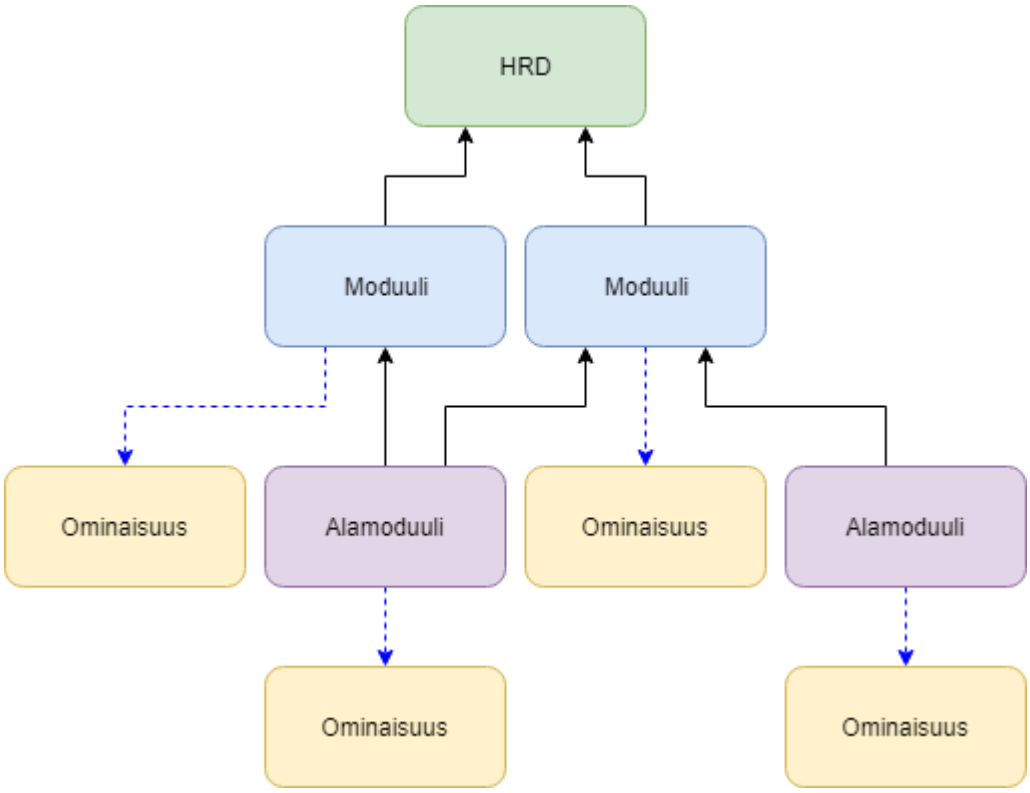
KUVA 4: Tuotteeseen kuuluvan ohjelmakoodin vaatimus

Ratkaisuna tähän otettiin tulevalle jatkokehitykselle linjaus, että HRD kehitetään tuotepohjaisena, jos kahdella tai useammalla asiakkaalla on tarve samalle tai lähes samalle ominaisuudelle, se kuuluu tuotteeseen (kuva 4). Mikäli asiakkaan tarve on sellainen, ettei sitä voida olemassa olevilla ratkaisuilla toteuttaa, lasketaan se räätälöinniksi. Mikäli kuitenkin tarve on sellainen, että sitä tullaan selvästi tarvitsemaan myöhemmissä käyttöönotoissa, voi kyseessä olla tuotekehityksen piiriin laskettavissa oleva työ.

Vaikka ohjelmiston tuotepohjainen kehittäminen hidastaakin jossain määrin yksittäisen toiminnallisuuden kehittämistä ja käyttöönottoa, sillä toimivuus ja yhteensopivuus tulee varmistaa aikaisempien toteutuksien kanssa, tarjoaa se suuren hyödyn niin ohjelmistoa kehittävälle yritykselle, kuin asiakasyritykselleKin. Tuotepohjaisena ratkaisuna kehitetty toiminnallisuus on täten käytössä useammassa projektissa. Ohjelmistoa kehittävä yritys säästää aikaa sekä rahaa, kun samaa toiminnallisuutta ei tarvitse uudelleen suunnitella ja ohjelmoida. Asiakasyritykselle taasen tuotelähtöisyys näkyy eritoten mahdollisten ohjelmointivirheiden varalta, mitä useammassa projektissa sama toiminnallisuus on käytössä, sitä varmemmin ja nopeammin mahdollinen ohjelmointivirhe löytyy ja tulee raportoiduksi. Samalla mitä useammasta lähteestä mahdollisen ohjelmavirheen aiheuttamaa dataa on luettavissa, sitä nopeammin se on paikannettavissa ja korjattavissa.

4.3 Modularisointi

Sellainen ominaisuus, joka on käytössä kahdella tai useammalla asiakkaalla, voidaan laskea kuuluvaksi tuotteeseen. Tuotteen perusrunko haluttiin kuitenkin pitää mahdollisimman kevyenä ja yksinkertaisena, tarjoten vain oleellisimmat ja välttämättömimmät ominaisuudet. Tähän ratkaisuna otettiin ohjelmiston kehityksessä käyttöön modularisointi.



KUVA 5: Moduulien riippuvuuksien havainnekuva

Modularisoinnilla tarkoitetaan ohjelmakoodin hajauttamista pienempiin osiin, jotka pyritään kehittämään siten, että niissä ei ole lainkaan, tai on korkeintaan muutamia riippuvuuksia muihin moduuleihin. Moduuli voi olla riippuva toisesta moduulista, mutta kaksi moduulia eivät voi olla riippuvia toisistaan. Sellaisessa tapauksessa, jossa jokin toiminnallisuus vaatisi kahden moduulin keskeistä riippuvuutta, tällainen ohjelmakoodi eriytetään omaan moduuliinsa, joka voi olla joko oma päätason moduulinsa tai olemassa olevan moduulin alamoduuli. Kuvassa 5 riippuvuudet ovat kuvattu mustilla nuolilla ja moduulien tarjoamat ominaisuudet sinisillä.

Tavoite on kuitenkin luoda ohjelmakoodista mahdollisimman joustavaa, jolloin moduulin tarjoama ominaisuus kirjoitetaan siten, että se aktivoituu vasta, kun kaikki sille oleelliset moduulit ovat käytössä. Tämä mahdollistaa esimerkiksi tilanteen, että moduuli, joka tarjoaa koulutustyypin sekä siihen liittyvän sähköpostipohjan, ei riipu sähköposteja käsittelevästä moduulista, vaan sähköpostipohja tulee käyttöön automaattisesti niissä tilanteissa, kun sähköposteja käsittelevä moduuli otetaan käyttöön. Tämmöisessä tilanteessa sähköpostipohjaa ei ole ideaalista eriyttää omaksi alamoduulikseen, sillä se lisäisi turhaan dokumentoitavien moduulien määrää.

Kullekin päätason moduulille on myös nimetty yrityksen sisällä ylläpitäjä, jonka vastuulla on olla perillä siitä, mitä kyseinen moduuli tarjoaa, mihin tilanteisiin se soveltuu ja mitä kehitystarpeita sillä on. Moduulien kehittämisestä vastaavat kaikki ohjelmistokehittäjät yhdessä, mutta moduulin ylläpitäjällä on päätäntävalta siitä, minkä muutoksen hyväksyy kyseiseen moduuliin. Tällä osallaan pyritään vaikuttamaan siihen, minkä verran muistettavia asioita yksittäisellä ohjelmistokehittäjällä on ja vähentämään ylimääräistä työkuormaa sekä turhia työtunteja.

4.4 Haasteet

Haasteita työlle muodostui muutamista eri osa-alueista. Olemassa oleviin toteutuksiin tehtyjen ominaisuuksien puutteellinen dokumentointi asetti paineita tuotteen riittävän kattavalle määrittelylle. Lisähaasteen määrittelylle aiheutti se, että kahdelle projektille oli luvattu toimitus ennen kuin tuotteen ominaisuuksia päästiin määrittämään.

Suurimmat haasteet tulivat Drupalin varhaisesta käyttöönotosta, sillä vaikka Drupal itsessään oli julkaistu jo täysin tuettuun Drupal 8 -versioon, ei monetkaan Drupalin valinnaisista moduuleista vielä tukeneet uutta ydintä. Käytännössä tämä tarkoitti sitä, että tuotekehityksestä piti varata aikaa myös avoimen lähdekoodin kirjastojen kehittämiseen. Riippuvuus ulkopuolisista toteutuksista sekä epävarmuudet tarvittavista osuuksista aiheuttivat haasteita myös projektien aikatauluttamiselle.

Varhainen käyttöönotto myös tarkoitti sitä, että Drupal 8:n dokumentaatio oli hyvin varhaisessa vaiheessa, eikä siitä aina ollut riittävästi apua ohjelmistokehittäjälle. Tällaisissa tapauksissa ohjelmistokehittäjän oli pakko selvittää suoraan lähdekoodia lukemalla, mitä mikäkin toiminnallisuus pohjimmillaan tekee. Lisäksi koska Drupal 8 oli ollut pitkään

kehityksessä ennen varsinaista julkaisua, osa dokumentaatiosta oli kerennyt jo vanhentua ennen virallista julkaisua. Joissain tapauksissa oli myös vaikeata erottaa, mille Drupal-versiolle löydetty dokumentaatio oli kirjoitettu.

Harhaanjohtavasta ja vajanaisesta dokumentaatiosta johtuen oli myös mahdotonta ennalta arvata, mitkä aiemmin käytetyistä lisämoduuleista ylipäätään olisi tarpeellisia enää uuden version kanssa. Tämä osallaan hidasti myös Drupal-yhteisössä tarjolla olevien moduulien julkaisua uudelle versiolle.

5 OHJELMISTOKEHITYKSEN KULKU

Ohjelmistokehityksen työnkulkua hiottiin koko tuotekehityksen ajan, jotta siitä saatiin mahdollisimman täsmällinen ja nopea, mutta myös riittävän ketterä. Perimmäisenä ideana kehitykselle otettiin, että kukaan ei ole yksittäin vastuussa mistään toteutuksen vaiheesta.

Parhaiten toimivaksi todettu työnkulku on kuvattu järjestyksessä tämän kappaleen alaotsikoissa siten, että kustakin työnkulun välivaiheesta voidaan tarvittaessa palata takaisinpäin aina ensimmäiseen vaiheeseen asti, jos myöhemmissä vaiheissa huomataan alkuperäisen idean vaativan jatkokehitystä.

5.1 Uuden toiminnallisuuden tarve

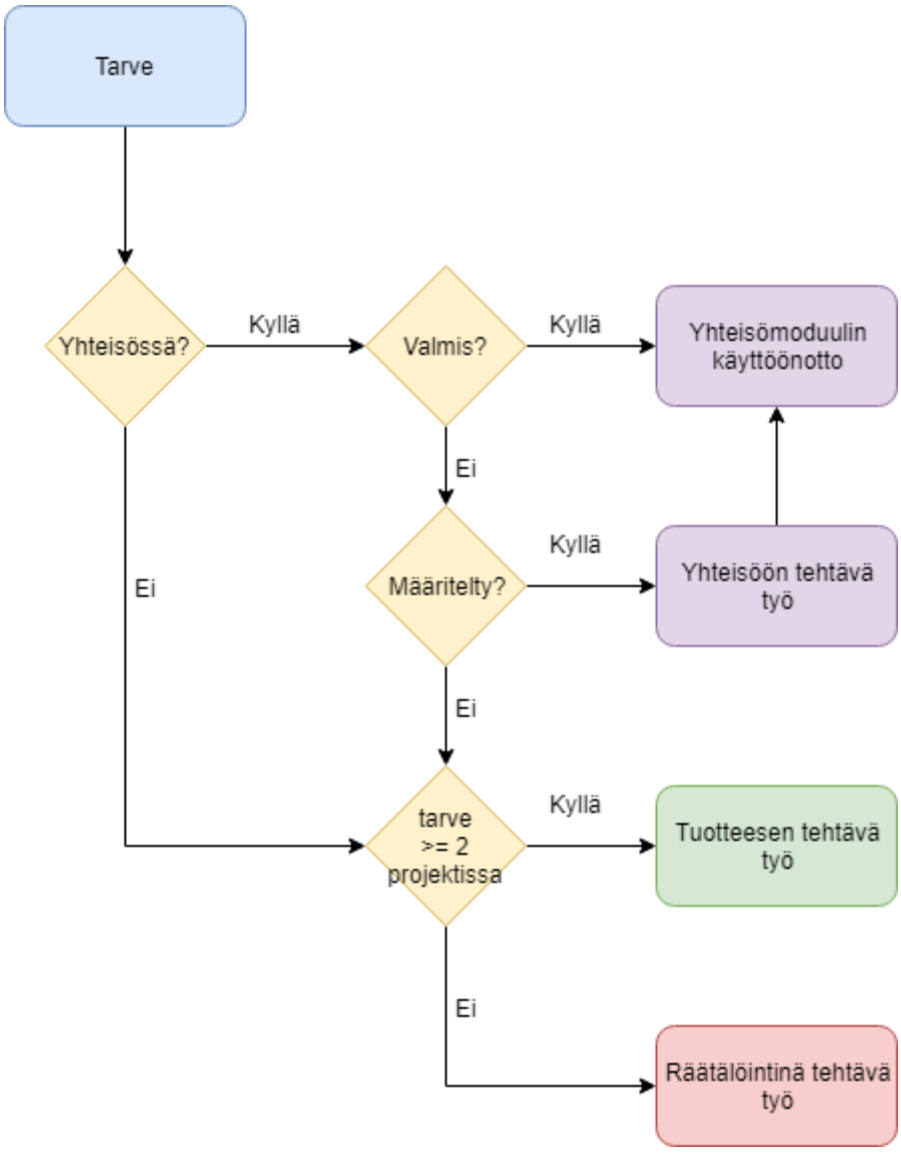
Alkuperäinen tarve on aina asiakaslähtöinen, joten sellaista toimintoa tai toiminnallisuutta, jota yksikään asiakas ei ole toivonut, ei myöskään lähdetä kehittämään. Tällä pyritään vähentämään ylimääräisen työn määrää ja palvelemaan asiakasta tehokkaammin juuri niillä ratkaisuilla, jotka asiakas itse on ensisijaisesti määritellyt.

Konkreettisen tarpeen määrittämiseksi katsottiin, että ohjelmistokehittäjän on hyvä osallistua asiakaspalavereihin, jolloin ohjelmistokehittäjä voi suoraan kertoa, mikäli toivottu ominaisuus on monimutkainen tai vaatisi räätälöintiä. Asiakas ei myöskään aina osaa tai uskalla pyytää sellaista toiminnallisuutta, jonka toteuttaminen olisi huomattavan helppoa ja parhaassa tapauksessa jo toteutettu. Tällaisissa tilanteissa on ehdottoman tärkeätä, että on ymmärretty asiakkaan perimmäinen ongelma, johon ratkaisua ollaan löytämässä.

Ohjelmistoprojektin palavereista tulee herkästi tapahtumia, joissa on paikalla vain edustajat niin asiakasyrityksestä kuin ohjelmistoa toimittavasta yrityksestä. Näille yhteistä on usein se, että kukaan palaveriin osallistuneista henkilöistä ei ole ohjelmiston loppukäyttäjä. Vaikka asiakas näissä tilanteissa osaisikin kertoa suurin piirtein, mitä toimintoja käyttäjät pääasiallisesti missäkin tilanteessa käyttävät tai haluavat käyttää, ei asiakas välttämättä tiedä, miten loppukäyttäjät kokevat ohjelmiston käyttämisen mielekkäimmäksi. Näitä tilanteita varten pyritään saamaan palautetta ohjelmiston loppukäyttäjiltä sekä asettumaan itse loppukäyttäjän asemaan ohjelmistoa kehittäessä.

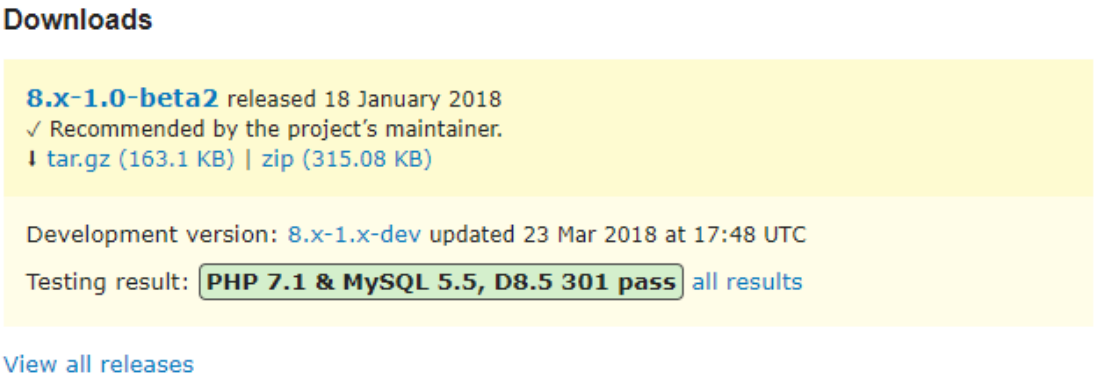
5.2 Toteutusidea

Asiakkaan tarpeen ollessa tiedossa on aika suunnitella sille riittävän kattava ratkaisu. Suunnittelun tavoite on erityisesti pyrkiä vähentämään myöhemmin tehtävän työn määrää, joten mitä yleisemmällä tasolla tunnettu tarve on, sitä parempi. Yleisellä tasolla tunnettuihin tarpeisiin on yleensä olemassa oleva ratkaisu Drupal-yhteisössä, joko jonkin yrityksen toteuttamana, tai jopa yksittäisen henkilön ylläpitämänä. Mitä useammassa paikassa yhteisön tarjoama ratkaisu on, sitä paremmin sen mahdolliset ongelmat on yleensä kartoitettu ja sitä nopeammin niihin myös tarjotaan korjauksia.



KUVA 6: Tarvetta vastaavan työn lokerointi

Mikäli tarpeeseen soveltuva yhteisön tarjoama moduuli on olemassa, se kattaa tarpeen ainakin suurimmaksi osaksi ja se on julkaistu vakaana versiona, voidaan se ottaa käyttöön. Vakaan version olemassaolo on tärkeä seikka, ja jos kaksi moduulia sattuisi tarjoamaan samaan tarpeeseen ratkaisun, otetaan näistä käyttöön ensisijaisesti se, kumpi on vakaassa versiossa tai vaihtoehtoisesti kummalla on suurempi käyttäjäkunta. Kaaviossa 3 on hahmotettu ensisijaista toivetta, jolla tarve toteutetaan.



KUVA 7: Facets-moduulin versiot (Drupal.org, 2018)

Toisinaan tulee tilanteita, jolloin yhteisössä olisi tarjolla moduuli, joka on tehty vastaavaa tarvetta varten, kuin asiakkaan tarve, mutta sitä ei ole joko vielä julkaistu Drupal 8-versiolle tai sen kehitys on varhaisvaiheessa. Varhaiskehitysvaiheessa olevaa julkaisua käyttöönottaessa joutuu aina tapauskohtaisesti harkitsemaan, aiheuttaako sen käyttöönotto myöhemmässä vaiheessa ongelmia. Tapana on, että varhaiskehitysvaiheessa oleva moduuli ei saa päivityspolkua, mikäli ohjelmiston kehittäjät päättävät muuttaa ohjelmakoodin rakennetta. Kuvassa 7 on kohtalaisen suositun yhteisömoduulin Facets tarjolla olevat versiot, eli varhaisvaiheen julkaisu (beta) sekä kehityshaara, jota ei yleisesti suositella käytettävän kuin kehittämismielessä. Facets on puhtaasti uudelleenkirjoitettu Drupal 8-versiolle, ja sen varhaisvaiheen käyttö on suhteellisen turvallista, sillä se ei kirjoita dataa tietokantaan, joka vaatisi mahdollisen arkkitehtuurimuutoksen jälkeen päivityspolun.

Lähtökohtaisesti varhaiskehitysvaiheessa oleva moduuli pyritään auttamaan valmiiksi käyttämällä tuotekehitykseen ohjattua aikaa yhteisöön tehtävään työhön. Tällä pyritään välttämään tilanteet, jotka aiheutuisivat varhaisvaiheisen moduulin käytöstä ja sen päivityspolun puutteesta. Tuotantokäytössä päivityspolku on aina oltava, joten se vaatisi joka

tapauksessa työtä. Lisäksi omana toteutuksena tehty päivityspolku saattaisi olla ristiriidassa myöhemmin moduulille julkaistavalle oikealle päivityspolulle.

5.3 Ohjelmiston kehittäminen omana toteutuksena

Toisinaan tarve voi olla niinkin rajallinen, että se esiintyy sellaisenaan vain HRD -projekteissa. Tällaisissa tapauksissa ohjelmakoodi on pakko toteuttaa itse, joten kaikki siihen käytetty aika on yrityksen näkökannasta kallista, ja se kasvattaa tuotekehityksen kokonaishintaa sekä ylläpidettävää ohjelmakoodin määrää.

Oman toteutuksen suunnittelussa on erityisen tärkeää, että suunnitelmaa ei tarvitse myöhemmin enää uusia. Suunnittelussa otettiin linjaus, että kaikki tuotteeseen tehdyt lisäominaisuudet pitää pystyä ottamaan käyttöön ja poistamaan käytöstä missä tahansa vaiheessa projektia. Lisäksi tuotetulla ohjelmakoodilla pitää olla riittävät testit, joista selviää kyseisen ohjelmakoodin käyttötarkoitukset ja -mahdollisuudet.

Alla on vaiheittain kehitettynä kuvitteellinen tarve toiminnolle, jossa varastoitaisiin dataa kahteen eri paikkaan, joista ensisijaisesti käytettäisiin nopeaa mutta rajallista tietosäiliötä. Mikäli ensisijainen tietovarasto tulisi täyteen, vaihdettaisiin käyttöön hitaampi mutta rajaton tietosäiliö. Esimerkki on kuvitteellinen, jotta se voidaan esittää mahdollisimman lyhyessä muodossa käyden kuitenkin kaikki oleelliset välivaiheet läpi. Esimerkkikoodia ei ole myöskään kommentoitu ohjelmakoodissa itsessään tilan säästämiseksi.

5.3.1 Interface

Interfacet ovat olio-ohjelmoinnissa yleisesti käytettyjä esittelytiedostoja, niin sanottuja minimivaatimuksia tietyn typpiselle olioluokalle. Interfacet eivät ole pakollisia, mutta helpottavat ohjelmoijaa ymmärtämään, mitkä ominaisuudet luokasta ovat varmasti käytettävissä, sillä tiettyä interfacea toteuttavat luokat voivat muuten poiketa toisistaan.

Suunnitteilla olevissa varastoissa on selkeästi yhteistä se, että niistä tarvitsee pystyä hakemaan dataa sekä noutamaan varastoitua dataa. Kyseisille toiminnoille voidaan siis luoda interface ohjelmakoodi 2:n mukaisesti.

```php
<?php

namespace Drupal\my_module;

interface RatedContainerInterface {

  public function limit();

  public function insert($item);

  public function extract();

  public function currentItemCount();

}
```

OHJELMAKOODI 2: Interfacen esittely

Interfacessa on tarjottu ohjelmistokehittäjälle sekä ohjelmalle itselleen tietoon, että jokaisessa tätä interfacea toteuttavassa luokassa on toiminnot uuden tietueen lisäämiseen, viimeisimmän lisätyn tietueen noutamiseen sekä nykyisten säilöttyjen tietueiden määrän tarkastamiseen.

5.3.2 Base class

Koska tiedetään, että oli käytössä kumpi tahansa tietovarasto, dataa käsitellään suurimmaksi osaksi samalla tavalla. Tällaisissa tapauksissa ne osat ohjelmakoodista, jotka ovat käytössä kaikissa tai lähes kaikissa tapauksissa, voidaan kirjoittaa yhdistettyyn luokkaan. Tällaisia kutsutaan pohjaluokiksi (engl. base class).

```php
<?php

namespace Drupal\my_module;

abstract class RatedContainerBase implements RatedContainerInterface {

  protected $limit = 0;

  public function limit() {
    return $this->limit;
  }

  public function insert($item) {
    if (!($this->limit) || $this->limit > $this->currentItemCount()) {
      $this->pushStorage($item);
    }
  }

  public function extract() {
    $storage = $this->loadStorage();
```

```
    $item = array_pop($storage);
    $this->setStorage($storage);
    return $item;
  }

  public function currentItemCount() {
    return count($this->loadStorage());
  }

  private function loadStorage() {
    return \Drupal::state()->get(get_class($this)) ?: [];
  }

  private function pushStorage($item) {
    $storage = $this->loadStorage();
    $storage[] = $item;
    $this->setStorage($storage);
  }

  private function setStorage(array $items) {
    \Drupal::state()->set(get_class($this), $items);
  }

}
```

OHJELMAKOODI 3: Pohjaluokan esittely

Ohjelmakoodi 3:n esimerkissä käytetyssä pohjaluokassa on esiteltynä toteutukset kaikkiin interfacessa esitettyihin funktioihin. Lisäksi luokassa on selkeytetty tietovarastoon koskevaa ohjelmakoodia eriyttämällä se luokan omiin funktioihin, jotka eivät ole näkyvillä luokan ulkopuolella. Luokan yksityiset funktiot ovat sellaisia, johon ohjelmistokehittäjällä eikä ohjelmalla ei ole pääsyä, vaan niitä voidaan kutsua korkeintaan välillisesti luokan muista funktioista.

5.3.3 Class

Drupal 8 eroaa aiemmista versioista sillä, että funktionaalisen ohjelmoinnin sijaan se käyttää pääasiallisesti olio-ohjelmointia. Luokat (engl. class) ovat kokoonpanoja, jotka voivat sisältää erinäisiä julkisia tai ei-julkisia tietueita. Yksinkertaisimmillaan luokka voi olla jopa ilman mitään sisältöä.

Aiemmassa otsikossa esitelty pohjaluokka ei sellaisenaan ole vielä valmis käytettäväksi, vaan se vaatii vielä sitä jatkavan luokan, joka toimii varsinaisena käytettävänä luokkana.

Aiemmin mainitut tietovarastot olivat muuten ohjelman kannalta samanlaisia, mutta toisessa oli yläraja tallennettavan tietojen kappalemäärälle. Nämä kaksi luokkaa voidaan toteuttaa ohjelmakoodien 4 ja 5 mukaisesti.

```
<?php

namespace Drupal\my_module;

class FastContainer extends RatedContainerBase {

  protected $limit = 5;

}
```

OHJELMAKOODI 4: Nopean, mutta rajallisen tietovarastoluokan esittely

```
<?php

namespace Drupal\my_module;

class UnlimitedContainer extends RatedContainerBase {

}
```

OHJELMAKOODI 5: Hitaan, mutta rajattoman tietovarastoluokan esittely

Kuten ohjelmakoodeista 4 ja 5 voidaan havaita, tietovarastoluokan lopullinen esittely ei vaadi erityisemmin ohjelmakoodia toimiakseen. Rajattoman tietovarastoluokan esittelyksi riittää luokka, joka jatkaa aiemmin luotua pohjaluokkaa, vastaavasti rajallisen tietovarastoluokan esittely vaatii vain asetetun ylärajan toimiakseen odotetusti. Huomioitavaa tässä kohtaa on, että oikeassa toteutuksessa ylärajaa ei voitaisi todennäköisesti ilmoittaa suoraan luokassa niin sanotusti kovakoodattuna, vaan yläraja pitäisi pystyä hakemaan funktiolla, jolloin oikea yläraja haettaisiin tallennettavasta tietovarastosta suoraan. Lisähuomiona sekin, että oikeassa toteutuksessa yläraja olisi helposti sekä suurella todennäköisyydellä järkevämpi toteuttaa niin päin, että tietovarastolta kysytään, onko sillä riittävästi tilaa tallennettavalle tiedolle.

Nyt toteutetut luokat ovat testattavissa ja käytettävissä. Esimerkkinä toiminnallisuuden testaamiseksi voidaan tehdä ohjelmakoodi 6:n mukainen testikoodi.

```
<?php

use Drupal\my_module\FastContainer;
```

```
use Drupal\my_module\UnlimitedContainer;

$containers = [
  new FastContainer(),
  new UnlimitedContainer(),
];

foreach ($containers as $container) {
  for ($i = 1; $i <= 10; $i++) {
    $container->insert($i);
  }
  echo get_class($container);
  echo " has {$container->currentItemCount()} items" . PHP_EOL;
}
```

OHJELMAKOODI 6: Ensimmäinen käsin tehtävä testi

Konsolista suoritettuna kyseisen testin tulos näyttää komentorivi 4:n mukaisesti toimivan odotetusti oikein.

```
$ scr containers.php
Drupal\my_module\FastContainer has 5 items
Drupal\my_module\UnlimitedContainer has 10 items
```

KOMENTORIVI 4: Ensimmäisen käsin tehtävän testin tulos

5.3.4 Dependency injection

Kun ohjelmakoodi on saatu kertaalleen toimimaan, eli siitä on toteutettu soveltuvuusselvitys (engl. Proof of Concept), on ohjelmakoodin jatkokehityksen aika. HRD:n kehityksessä otettiin linjaus, että jos ohjelmakoodi on ylipäätään tarkoitus ottaa pysyvästi käyttöön, tehdään se suoraan sellaiseksi, että sitä on mahdollisimman helppo jatkokehittää.

Aiemmassa tehty pohjaluokka toimii tässä hyvänä esimerkkinä, vaikka ohjelmakoodi toimiikin juuri kuten on haluttu, se ei läpäise standardin vaatimuksia, eikä sitä ole helppo jatkokehittää. Pohjaluokan ohjelmakoodissa on kahdessa eri kohdassa käytetty ulkoisen riippuvuuden asettavaa kutsua.

```
  private function loadStorage() {
    return \Drupal::state()->get(get_class($this)) ?: [];
  }

  private function setStorage(array $items) {
    \Drupal::state()->set(get_class($this), $items);
  }
```

OHJELMAKOODI 7: Korvattava osuus ohjelmakoodista

Jotta ohjelmakoodi 7:n rivit voitaisiin muuntaa sellaisiksi, että pohjaluokka ei riippuisi ulkoisista luokista, otettiin käyttöön riippuvuuksien injektointi (engl. dependency injection). Riippuvuuksien injektoinnilla tarkoitetaan sitä, että itse ohjelmoitava luokka ei hae muualta tarvittavia luokkia, vaan ne toimitetaan luokan rakentajalle. Tällöin vastuu ja päätäntävalta siirretään ohjelmakoodia käyttävälle ohjelmistokehittäjälle.

Riippuvuuksien injektointi mahdollistaa olioluokan esittelyn käyttäen interfaceja, jolloin ohjelmakoodille riittää, kun se saa minkä tahansa sellaisen luokan käytettäväksi, missä on sen tarvitsemat funktiot. Lisäksi injektointi antaa mahdollisuuden kirjoittaa kevyitä ja nopeita testejä, koska testille epäolennaiset vaiheet voidaan luoda olettamuksin.

Jotta aiemmin luotu pohjaluokka voidaan muuttaa, sen ohjelmakoodi 3:n esittelyä muokataan ohjelmakoodi 8:n mukaiseksi.

```
<?php

namespace Drupal\my_module;

use Drupal\Core\DependencyInjection\ContainerInjectionInterface;
use Drupal\Core\State\StateInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

abstract class RatedContainerBase implements RatedContainerInterface, Con-
tainerInjectionInterface {

  protected $state;

  protected $limit = 0;

  // ...
}
```

OHJELMAKOODI 8: Esittelyyn tehtävä muutos

Tämän lisäksi luokkaan täytyy lisätä kaksi uutta funktiota ohjelmakoodi 9:n mukaisesti.

```
  public function __construct(StateInterface $state) {
    $this->state = $state;
  }

  public static function create(ContainerInterface $container) {
    return new static(
      $container->get('state')
    );
  }
```

OHJELMAKOODI 9: Riippuvuuksien injektionin mahdollistavat funktiot

Nyt aiemmin ulkopuoliseksi riippuvuudeksi ohjelmoitu State on pakotettuna luokan luomiselle, joten riippuvuuden aiheuttaneet ohjelmakoodin osat voidaan vaihtaa seuraavanlaisiksi.

```
  private function loadStorage() {
    return $this->state->get(get_class($this)) ?: [];
  }

  private function setStorage(array $items) {
    $this->state->set(get_class($this), $items);
  }
```

OHJELMAKOODI 10: Ohjelmakoodi 7:n korvaavat funktiot.

Nyt ohjelmakoodin toimivuus voidaan testata uudelleen, kun siihen on lisätty luokkien rakentajakutsuihin vaadittu osuus. Ohjelmakoodista 11 myös näkyy, kuinka vastuu koodin oikeaoppisesta käytöstä siirtyy suorittavaa koodia kirjoittavalle ohjelmistokehittäjälle. Nykyisen ohjelmakoodin toimivuuden testaamiseksi tuli koodiin lisätä myös kohta, joka poistaa aiemmin sinne lisätyt asiat.

```
<?php

use Drupal\my_module\FastContainer;
use Drupal\my_module\UnlimitedContainer;

$state = \Drupal::state();

$containers = [
    new FastContainer($state),
    new UnlimitedContainer($state),
];

foreach ($containers as $container) {
    $items = 0;
    while ($container->extract()){
        $items++;
```

```
    }
    echo "Removed {$items} old items from " . get_class($container) .
PHP_EOL;
}

foreach ($containers as $container) {
    for ($i = 1; $i <= 10; $i++) {
        $container->insert($i);
    }
    echo get_class($container);
    echo " has {$container->currentItemCount()} items" . PHP_EOL;
}
```

OHJELMAKOODI 11: Käsin tehtävän testauksen koodi riippuvuuksien injektointien jälkeen

5.3.5 Service

Riippuvuuksien injektoinnista aiheutuu herkästi monia ulkopuolelta asetettavia riippuvuuksia, jolloin niiden lisääminen jokaisessa käytettävässä paikassa on hyvin työlästä. Tällaisia tilanteita varten Drupal tarjoaa mahdollisuuden luoda palveluita (engl. service).

Äskeisessä esimerkissä injektiossa käytettiin ContainerInjection-luokkaa, jota käyttämällä kaikki riippuvuudet ovat tuotavissa luokkaan yhden säiliön (engl. container) avulla. Säiliö tarjoaa rajapinnan kaikkiin Drupalissa oleviin palveluihin. Säiliön idea on sen muokattavuudessa. Normaalissa käyttötarkoituksessa Drupal huolehtii siitä, minkä luokan säiliö palauttaa kutakin palvelua pyydettäessä. Muokattavuuden tarve korostuukin lähinnä vasta automaattisten testien puolella, jossa oikean palvelun sijaan halutaan lähinnä testata erilaisia mahdollisia funktioiden paluuarvoja kullekin riippuvuudelle.

Palvelu on käytännössä vain palvelunimi, jolla jokin luokka rekisteröidään Drupalin käyttämään säiliöön. Palveluita on kuitenkin mahdollista muokata sekä luoda dynaamisesti. Mallitapauksessa käytetyt kaksi eri tietovarastoluokkaa voidaan rekisteröidä ohjelmakoodi 12:n mukaisesti.

```
services:
  rated_container.primary:
    class: Drupal\my_module\FastContainer
    arguments: ['@state']
  rated_container.secondary:
    class: \Drupal\my_module\UnlimitedContainer
    arguments: ['@state']
```

OHJELMAKOODI 12: Palvelun rekisteröinti

Ohjelmakoodi 12:ssa esitellyt palvelut sisältävät myös parametrin State-luokalle, joka on viittaus toiseen palveluun. Riippuvuuksien kirjaaminen ei aina ole välttämätöntä palvelun rekisteröintiin, mutta se on suositeltava tapa, sillä kun ne ovat ilmoitettu argumentteina, Drupalin säiliö pitää huolen niiden olemassaolosta ennen luokan rakennusta. Lisäksi ne helpottavat ohjelmistokehittäjää tulkitsemaan moduulien välisiä riippuvaisuuksia.

Kun palvelut ovat rekisteröity Drupaliin, testikoodista voidaan poistaa sekä esittelyt itse tietovarastoluokille, että niiden rakentajakutsut. Palveluiden luomisen jälkeen testikoodi näyttää ohjelmakoodi 13:n mukaiselta.

```
<?php

$containers = [
    \Drupal::service('rated_container.primary'),
    \Drupal::service('rated_container.secondary'),
];

foreach ($containers as $container) {
    $items = 0;
    while ($container->extract()){
        $items++;
    }
    echo "Removed {$items} old items from " . get_class($container) .
PHP_EOL;
}

foreach ($containers as $container) {
    for ($i = 1; $i <= 10; $i++) {
        $container->insert($i);
    }
    echo get_class($container);
    echo " has {$container->currentItemCount()} items" . PHP_EOL;
}
```

OHJELMAKOODI 13: Käsin tehtävä testi palveluiden avulla

5.3.6 Controller

Tähän asti esimerkkinä tehty toteutus on toiminut odotetusti kuvitteellisen kahden tietovaraston käsittelyyn. Käsittelyä on kuitenkin hidastanut se, että käytettävää ohjelmakoodia tekevän ohjelmistokehittäjän täytyy olla tietoinen siitä, kumpi tietovarasto kulloinkin on soveltuva tiedon tallennukseen.

Ratkaisuna suorittavan ohjelmakoodin yksinkertaistamiseksi on mahdollista luoda kontrolliluokka (engl. controller), joka tällöin pitää huolen siitä, kumpi tietovarasto on milloinkin käytössä, mutta toimii muutoin ulospäin vastaavasti kuin yksittäinen tietovarastoluokka. Tietovarastojen ohjainluokka luotiin tässä tapauksessa ohjelmakoodi 14:n mukaan.

```
<?php

namespace Drupal\my_module\Controller;

use Drupal\Core\Controller\ControllerBase;
use Drupal\my_module\RatedContainerInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

class RatedContainerController extends ControllerBase {

  private $primary;

  private $secondary;

  protected $current;

  public function __construct(RatedContainerInterface $primary, RatedContain-
erInterface $secondary) {
    $this->primary = $primary;
    $this->secondary = $secondary;
    $this->prepareCurrentForInsert();
  }

  public static function create(ContainerInterface $container) {
    return new static(
      $container->get('rated_container.primary'),
      $container->get('rated_container.secondary')
    );
  }

  public function getCurrentContainer() {
    return $this->current;
  }

  public function insert($item) {
    $this->current->insert($item);
    $this->prepareCurrentForInsert();
  }
```

```
  public function extract() {
    $this->prepareCurrentForExtract();
    return $this->current->extract();
  }

  public function currentItemCount() {
    $count = 0;
    $count += $this->primary->currentItemCount();
    $count += $this->secondary->currentItemCount();
    return $count;
  }

  private function prepareCurrentForInsert() {
    if (!($this->primary->limit()) || ($this->primary->limit() > $this->pri-
mary->currentItemCount())) {
      $this->current = $this->primary;
    }
    elseif (!($this->secondary->limit()) || ($this->secondary->limit() >
$this->secondary->currentItemCount())) {
      $this->current = $this->secondary;
    }
    else {
      throw new \Exception('No usable container');
    }
  }

  private function prepareCurrentForExtract() {
    if (($this->current === $this->secondary) && $this->current->currentItem-
Count() === 0) {
      $this->current = $this->primary;
    }
  }

}
```

OHJELMAKOODI 14: Ohjainluokka tietovarastojen käyttöön

Ohjainluokka esiteltiin Drupalille palveluna samalla tapaa (ohjelmakoodi 15), kuin itse tietovarastoluokatkin (ohjelmakoodi 12).

```
services:
  # ...
  rated_container.controller:
    class: Drupal\my_module\Controller\RatedContainerController
    arguments: ['@rated_container.primary', '@rated_container.secondary']
```

OHJELMAKOODI 15: Ohjainluokan palvelun lisäys

Tämän jälkeen testaukseen käytettyä ohjelmakoodia oli mahdollista keventää huomattavasti, jolloin päästiin tilanteeseen, jossa ohjelmistokehittäjän ei edes tarvitse olla tietoinen käytettyjen tietovarastoluokkien rajoista tai siitä, kumpi tietovarasto kulloinkin on käytössä.

```
<?php

$controller = \Drupal::service('rated_container.controller');

$items = 0;
while ($controller->extract()){
  $items++;
}
echo "Removed {$items} old items from containers" . PHP_EOL;

for ($i = 1; $i <= 10; $i++) {
  echo "Inserting item #{$i} into " . get_class($controller->getCurrentCon-
tainer()) . PHP_EOL;
  $controller->insert($i);
}
```

OHJELMAKOODI 16: Käsin tehtävä testi ohjainluokan avulla

Ohjelmakoodi 16:n esimerkkikoodin tuloste poikkeaa aiemmista hieman. Samalla tuloste osoittaa, kuinka kontrolliluokka vaihtaa tietovarastoluokkaa ilman, että ohjelmistokehittäjän tarvitsee puuttua siihen, mihin tieto konkreettisesti tallennetaan (komentorivi 5).

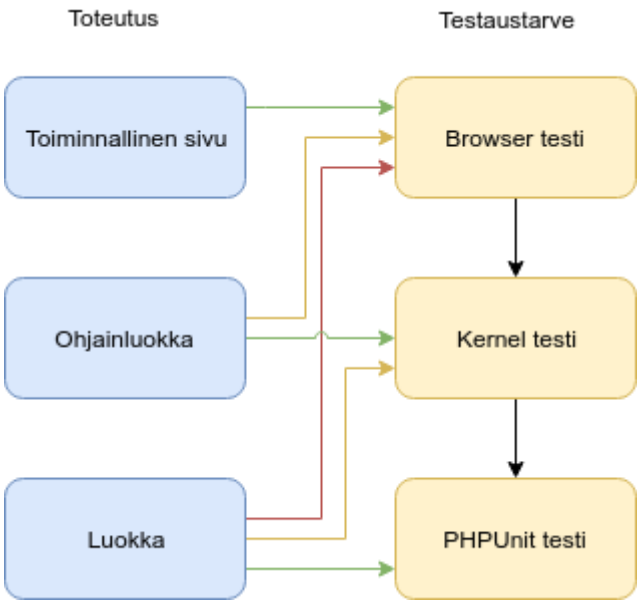
```
$ scr example.php
Removed 10 old items from containers
Inserting item #1 into Drupal\my_module\FastContainer
Inserting item #2 into Drupal\my_module\FastContainer
Inserting item #3 into Drupal\my_module\FastContainer
Inserting item #4 into Drupal\my_module\FastContainer
Inserting item #5 into Drupal\my_module\FastContainer
Inserting item #6 into Drupal\my_module\UnlimitedContainer
Inserting item #7 into Drupal\my_module\UnlimitedContainer
Inserting item #8 into Drupal\my_module\UnlimitedContainer
Inserting item #9 into Drupal\my_module\UnlimitedContainer
Inserting item #10 into Drupal\my_module\UnlimitedContainer
```

KOMENTORIVI 5: Ohjainluokan avulla suoritetun testin tulos

5.3.7 Testaus

Kun uuden ohjelmakoodin kehitysvaiheessa on päästy siihen pisteeseen, että se olisi mahdollista ottaa käyttöön, tulee sen toiminta vielä varmistaa. Aiemmissa välivaiheissa käytettiin toiminnan testaamiseksi skriptiä, jolla on voitu varmistaa ohjelman suorituksen päätyvän haluttuun lopputulokseen.

Toimivuuden varmistamiseksi myös jatkossa, esimerkiksi niissä tapauksissa, kun tehdystä ohjelmakoodista havaitaan bugi, tai vanhaan luokkaan halutaan toteuttaa uusia toiminnallisuuksia, on erityisen tärkeää olla muuttamatta vanhan ohjelmakoodin toimintalogiikkaa vahingossa. Helpoin tapa varmistaa toimivuus on kirjoittaa toteutukselle testi.



KUVA 8: Toiminnallisuustasoa vastaava testitason hahmotelma

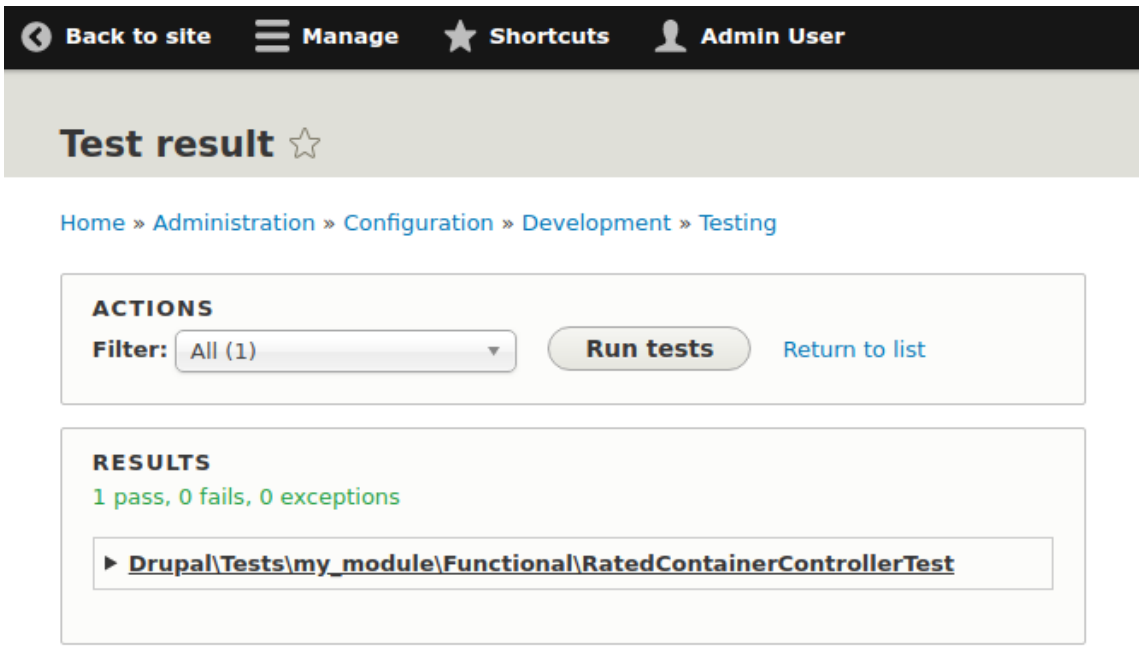
Drupal tarjoaa oletuksena puitteet testaamiselle kolmessa eri tasossa: Unit test, Kernel test sekä Functional test. Functional sisältää vielä erikseen tasot Browser test ja JavaScript test. HRD:n kehityksessä on lähtökohtaisesti pyritty saamaan riittävän kattava testaus ilman erillisiä JavaScript-testejä. Kuvassa 8 on hahmoteltuna kullekin toteutustasolle ominainen testaustaso. Sopiva taso on osoitettu vihreällä nuolella, sekä mahdolliset vaihtoehdot testitasolle keltaisella tai punaisella nuolella, joista keltainen ensisijaisena vaihtoehtona.

Riittävän kattava testaus toiminnallisuuden varmistamiseksi edellä rakennetulle ohjelmakoodille on ohjelmistokehittäjän näkökulmasta vaivattominta tekemällä sille Browser test -tyyppinen testi, joka testaa luodun ohjainluokan kautta kaiken ohjelmallisesti oleellisen toiminnallisuuden (ohjelmakoodi 17).

```
<?php

namespace Drupal\Tests\my_module\Functional;

use Drupal\Tests\BrowserTestBase;

/**
 * Class RatedContainerControllerTest
 *
 * @group my_module
 */
class RatedContainerControllerTest extends BrowserTestBase {

  public static $modules = ['my_module'];

  public $ratedContainerController;

  protected function setUp() {
    parent::setUp();
    $this->ratedContainerController = \Drupal::service('rated_container.con-
troller');
  }

  public function testQuickly() {
    for ($i = 1; $i <= 10; $i++) {
      $this->ratedContainerController->insert($i);
    }
    for ($j = 10; $j >= 1; $j--) {
      $pulled = $this->ratedContainerController->extract();
      $this->assertEquals($j, $pulled);
    }
  }

}
```

OHJELMAKOODI 17: Automatisoitava testi ohjainluokalle.

Nyt testi on ajettavissa koska tahansa Drupalin käyttöliittymästä käsin, jolloin ohjelmistokehittäjä voi varmistaa tekemiensä muutosten yhteensopivuuden vanhan koodin kanssa (kuva 9). Lyhyestä muodostaan huolimatta ohjelmakoodi 17:n mukainen testi pitää huolen siitä, että moduuli on asennettavissa, haluttu palvelu on olemassa, palvelun käyttämät tietoluokat ovat olemassa sekä näiden oleelliset funktiot toimivat.



KUVA 9: Automatisoidun testin tulos testauskäyttöliittymässä

Testiä on mahdollista jatkokehittää vaativammaksi lisäämällä luotuun luokkaan uusia funktioita. Jokainen funktio lähtee kuitenkin niin sanotusti nollatilanteesta, eli niillä ei ole mitään historiatietoja tai muistikuvaa muista ajetuista testeistä. Mikäli useampi testi on riippuvainen jostakin pohjatiedoista, nämä voidaan asettaa setUp-funktiossa.

HRD:n kehityksessä käytänne on, että mikäli sovelluksessa havaitaan virhe, tulee sen korjauksen ohessa tarjota testi, jolla voidaan varmistaa, että sama virhe ei pääse toistumaan uudelleen. Vakavimmissa tapauksissa, kuten integraation virhetoiminnoissa tämä on luonnollinen tapa, mutta myös vähemmän oleellisissa virheissä on todettu hyödylliseksi käyttää hetki testin kirjoittamiseen, sillä yksinkertaisenkin vian selvittämisessä voi usein vierähtää tunteja. Lisäksi mitä yksinkertaisempi virhe on, sitä helpompi sille yleisesti ottaen on kirjoittaa vastaava testi.

5.4 Testausputki (pipeline)

5.4.1 Code standard -vaihe

Ohjelmakoodin tarkastamisen ensimmäinen vaihe on nimeltään Code standard, tässä vaiheessa tarkastetaan, onko toteutettu ohjelmakoodi Drupalin koodistandardien mukainen. Standardit yhtenäistävät ohjelmakoodin muotoilun sellaiseksi, että suoraan kirjoitettua koodia katsomalla ei pysty kertomaan, kuka kyseisen osuuden on kirjoittanut.

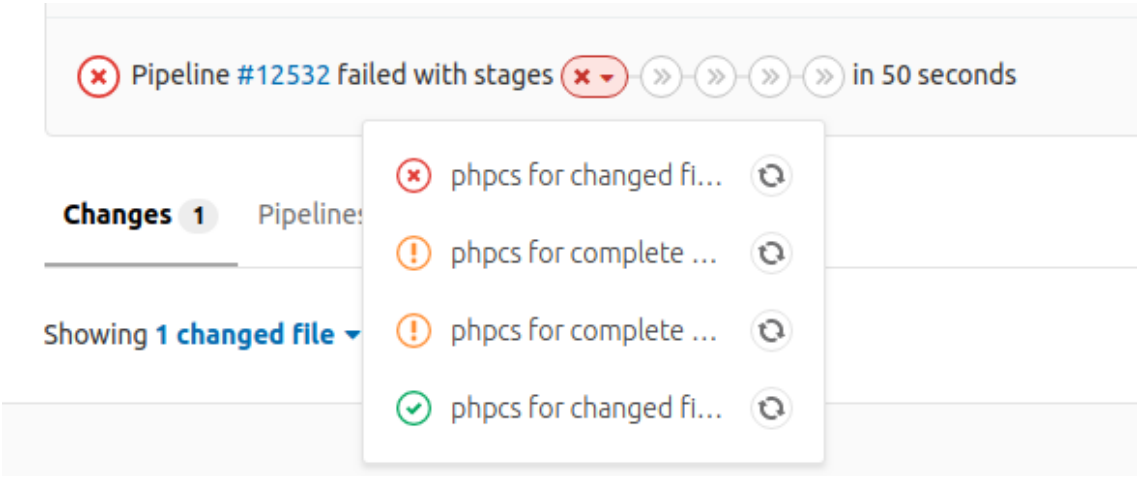
Koodin yhtenäistämisellä on ensisijaisena tarkoituksena edesauttaa sitä, että kuka tahansa samojen standardien parissa työskentelevä pystyy lukemaan tuotettua ohjelmakoodia helposti ja ymmärtämään siinä tapahtuvat asiat vaivattomasti. Sivuvaikutuksena koodin standardointi myös usein ohjaa toimivampaan toteutustapaan, joka taasen on HRD:n kehityksessä mielletty hyväksi tavaksi saada rakentavaa palautetta.

Drupalin valinnaisista lisämoduuleista löytyvää Drupal Coderia on mahdollista käyttää omassa työympäristössä sellaisenaan, mutta toisinaan kiireessä tämä tarkastusvaihe saattaa päästä unohtumaan. Tähän ratkaisuna lisättiin GitLabissa suoritettavaan testausputkeen uusi välivaihe, joka tarkastaa muuttuneiden tiedostojen standardinmukaisuuden.

```
phpcs for changed files:
  image: willhallonline/drupal-phpcs
  tags:
    - d8test
  stage: prebuild
  cache:
    key: "$CI_PIPELINE_ID"
    untracked: true
  only:
    - branches
  except:
    - master
  script:
    - CHANGED_FILES=$(git diff --name-only --diff-filter=d origin/master...
|tr '\n' ' ')
    - if [ -n "$CHANGED_FILES" ]; then phpcs --standard=Drupal,DrupalPractice
$CHANGED_FILES; else echo "No changed files"; fi
```

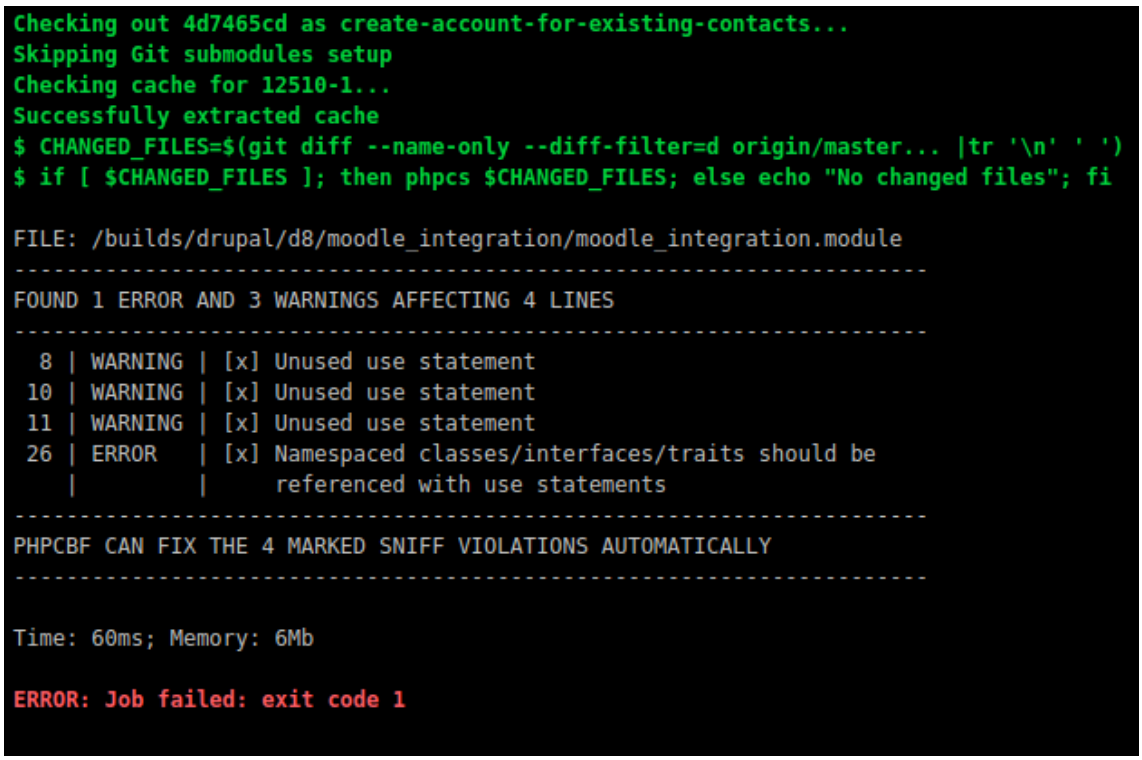
OHJELMAKOODI 18: Ohjelmakoodin standardinmukaisuuden automaattinen testaus

Kun ohjelmakoodin 18 mukainen osio lisätään GitLabin testausputkeen, lisäyksen ansioista ohjelmistokehittäjä saa palautteen lähes välittömästi, mikäli tehdyissä muutoksissa olisi jotain standardista poikkeavia virheitä. Tämä koettiin suurena parannuksena, sillä aiemmin kun automaattista standardien tarkastusta ei vielä ollut ja niiltä osin tehty ohjelmamuutos ei ollut hyväksyttävä, joutui tekijä palaamaan saman työn pariin, jonka oli mahdollisesti jo täysin unohtanut.



KUVA 10: Standardinmukaisuuden tarkasteluvaihe. Vanhojen tiedoston epästandardinmukaisuus on sallittu, mutta uusien ja muokattujen tiedostojen osalta testausputki keskeytyy, mikäli muutokset eivät vastaa standardeja

Toteutettu testausputki estää läpimenon vain siinä tapauksessa, että standardivirhe on muuttuneissa tiedostoissa (kuva 10). Testausputkessa ajetaan samalla toinen tarkastus, joka tarkistaa koko projektin koodin standardivirheiden varalta, mutta mikäli niitä on jo entuudestaan ollut olemassa, eivät ne myöskään estä tulevaa muutosta.



KUVA 11: Standarditestin raportti epäonnistuneen testausputken osalta

Lisäksi tarkastus antaa välittömän palautteen (kuva 11), mikäli ohjelmakoodissa on esimerkiksi tuntemattomia tai käyttämättömiä muuttujia, jolloin ohjelmistokehittäjä voi tehdä tarvittavat muutokset vielä, kun muutos, jota ylipäätään on tekemässä, on hyvässä muistissa. Tämä on koettu HRD:n kehityksessä huomattavasti ohjelman vakautta ja toimivuutta edistävänä.

5.4.2 Build-vaihe

Toisena vaiheena testausputkessa on Build-vaihe, jossa testataan koko projektin asennettavuus sekä asetusten tuominen järjestelmään. Tämä vaihe on erityisen oleellinen moduulien, sekä tuotekehityksessä käytettävän demon osalta, sillä molempien asennus täytyy onnistua ilman, että asiantuntija tai ohjelmistokehittäjä puuttuu projektin pystytykseen.

Epäonnistuessaan tämä vaihe paljastaa pääsääntöisesti, mikäli moduuleilla on sellaisia riippuvuuksia, joita ei ole huomioitu oikeaoppisesti. Vaihe voi epäonnistua myös siinä tapauksessa, että moduulin tarjoamissa asetustiedostoissa on jotain sellaista, mikä ei sellaisenaan ole Drupal-projektiin asennettavissa.

Mikäli asennusvaihe läpäistään onnistuneesti, tallentaa GitLab onnistuneesta asennuksesta vedoksen (engl. artifact), jota hyödynnetään muissa vaiheissa, mitkä ovat riippuvaisia Drupalin olemassaolosta. Vedoksesta on myös erityisesti hyötyä niissä tapauksissa, kun vaihe epäonnistuu, eikä syy ole täysin selvä. Tällaisessa tapauksessa ohjelmistokehittäjä voi ladata vedoksen omaan ympäristöönsä ja selvittää ongelman syntyperän.

Samaan vaiheeseen on suunnitteilla myös toinen tarkastus, jossa käytäisiin läpi projektin päivitettävyys. Tämä osio pitäisi sisällään edellä mainitun asennusosion mukaisen asennuksen edellisestä virallisesta versiosta, jonka jälkeen tarkastettaisiin, onko nykyinen ohjelmakoodi päivitettävissä. Tämä tarkastus tehdään nykyisin käsin suljetussa ympäristössä aidolla tuotantosisällöllä, jotta voidaan varmistaa päivityksen onnistuminen. Automaattista osiota ei ole pidetty erityisen kriittisenä kehitystarpeena, sillä tuotantosisältöä ei ole mahdollista sisällyttää automatisoituun vaiheeseen, eikä vastaavan sisällön tuottaminen testien avulla ole yksinkertaista.

5.4.3 Testit

Asennusvaiheen ollessa valmis ja Drupal on käytettävissä gitlabin tarjoaman vedoksen kautta, on aika suorittaa Drupalin moduuleita ja projektia varten kirjoitetut automaattitestit. Automaattitestit suoritetaan kolmessa eri vaiheessa niiden tasojen mukaan.

Uutta toiminnallisuutta tehdessä HRD:n kehityksessä otettiin käytäntö, että vaikka kullekin toiminnallisuuden tasolle on oma vastaava testitaso, täytyy sama toiminnallisuus olla varmistettuna myös ylemmän tason testissä. Suoranaisesti tämä tarkoittaa sitä, että kun uusi toiminnallisuus tehdään, tehdään sille ensisijaisesti raskain testi ensisijaisesti, kevyemmät testivaiheet tehdään jatkokehityksenä tai mikäli sama toiminnallisuus aiheuttaa usein testivaiheen epäonnistumisen.

GitLabin automaattitestit vievät toisinaan hyvinkin paljon aikaa, jonka lisäksi useat eri projektit käyttävät samaa testejä suorittavaa palvelinta (engl. runner). Tästä syystä testeihin aika-ajoin voi tulla pitkätkin ruuhkat, joka osallaan tarkoittaa, että ohjelmistokehittäjä joutuu odottamaan palautetta tekemästään työstä, sekä mikäli testausputki kuitenkin epäonnistuu, muut ohjelmistokehittäjät ovat joutuneet odottamaan omien testausputkien alkamista käytännössä turhaan.

Testien osalta ruuhkautumista purettiin siten, että testit pilkottiin kolmeen eri tasoon, joista ensimmäisenä ajetaan PHPUnit-testit. Unit-testit ovat mahdollisia ajaa minimaalisilla riippuvuuksilla testattavan ohjelmakoodin ulkopuolisista tekijöistä. Käytännössä tämä tarkoittaa sitä, että Unit-testit voidaan ajaa ilman, että Drupalia on asennettu lainkaan. Tällöin yksittäisen funktion sisällä käytetyt asiat oletetaan ja vain funktion ulostulo tarkastetaan. Unit testit ovat hyvin nopeita, mutta olettamuksista johtuen luotettavia vain tietyissä määrin.

Toisena testivaiheena suoritetaan Kernel-testit, jolloin Drupal on jo jossain määrin asennettu. Asennus voi olla joko kokonainen puhtaasti asennettu Drupal tai erikseen testikäyttöön tehty tietokanta-alustus. Kernel-testit mahdollistavat tehokkaasti yksittäisen moduulin asennuksen varmistamisen, sekä kaiken taustalla (engl. backend) tapahtuvan ohjelmakoodin testaamisen. Kernel-testien suoritusajassa ei kuitenkaan ole merkittävää eroa ylemmän tason Browser-testiin ja osittaisesta Drupalin asennuksesta johtuen Kernel-testit voivat jopa epäonnistua sellaisista syistä, jotka eivät vaikuta kokonaisessa asennusympäristössä.

Kolmantena testivaiheena suoritetaan viimein Browser-testit, joiden aikana Drupal on käytettävissä kokonaisuudessaan. Tämä testitaso pystyy liikkumaan Drupalin käyttöliittymässä ja käyttämään niitä toiminnallisuuksia, joita oikea loppukäyttäjäkin tulee käyttämään. Browser-testeissä on lisäksi mahdollista käyttää kaikkia alempien testitasojen ominaisuuksia, joten taustalla tapahtuvien asioiden testaus on myös mahdollista. Tämä testitaso edellyttää, että Drupal on kokonaisuudessaan saatu asennetuksi, joten se testaa myös itsessään kirjoitetun testin lisäksi tehdyn ohjelmakoodin yhteensopivuutta muun ohjelmakoodin kanssa. Browser-testeissä ei enää ole muita olettamuksia kuin mitattavien paluuarvojen oikeellisuus, joten sen voidaan katsoa olevan riittävän luotettava kertomaan, toimiiko jokin ominaisuus.

Drupal-testejä suorittavassa vaiheessa on havaittu jatkokehityksen tarpeita muun muassa PHPUnit-testien osalta, sillä nykyisellään ne suoritetaan vasta, kun GitLab on suorittanut itse asennusvaiheen ja luonut tästä vedoksen. Unit-testit tulisi siirtää ajettavaksi ennen tätä vaihetta, mutta ovat nykyisellään sidoksissa vanhaan Simpletest-moduuliin joka puolestaan vaatii osittaisen asennuksen olemassaolon.

Lisäkehitystarve on myös GitLab testausputkia suorittavissa testipalvelimissa, sillä testien määrä lisääntyy koko ajan, eikä jaettu testipalvelin skaalaudu tarpeen mukaisesti. Nykyisellään ruuhkaisimpaan aikaan yksittäinen testausputki saattaa olla jopa tunteja jonossa ennen kuin testipalvelin pääsee sitä suorittamaan. Ratkaisuna tähän olisi pilvipalvelussa suoritettava Docker-kontti, jota skaalattaisiin testitarpeen mukaan. Käytännössä tämä tarkoittaisi sitä, että uuden testausputken syntyessä kyseiseen pilvipalveluun luotaisiin samalla automaattisesti kyseiselle testausputkelle omistautunut testipalvelin. Kyseinen palvelin olisi elossa niin kauan, kun testausputki on suorituksessa tai sen tarjoama vedostiedosto on voimassa. Näin toimiessaan testausputket eivät ruuhkauttaisi toisiaan, eikä missään tilanteessa yksittäinen palvelin pyörisi tyhjillään, toisin kuin nykyinen joka on tyhjillään aina työaikojen ulkopuolella.

5.5 Tarkastusvaihe

Yksittäisen ohjelmistokehittäjän tekemä tuotos voi toisinaan olla hyvinkin kapeakatseisesti tehty. Etenkin pitkään työstetty ominaisuus on omiaan aiheuttamaan puhtaita ajatusvirheitä, sekä testaamattomia osuuksia. Voi myös olla, että toiminnallisuus toimii juuri kuten pitääkin, mutta se on toteutettu tarpeettoman vaikeasti. Näiden virheiden välttämiseksi HRD:n kehityksessä on otettu käyttöön koodin katselmointi eli tarkastusvaihe. Tässä vaiheessa toinen ohjelmistokehittäjä tarkastaa tehdyn työn siinä määrin, kun se on oleellista.

Katselmoinnilla pyritään jakamaan vastuu tehdystä työstä, jolloin mahdollisen ohjelmistovirheen löytyessä ei kenenkään tarvitse yksin kantaa vastuuta tai tuntea syyllisyyttä virheestä. Katselmointivaiheella jaetaan myös osaamista eri ohjelmistokehittäjien välillä ja usein tarkastusvaihe opettaakin enemmän tarkastajaa kuin tekijää.

Tarkoitus katselmoinnissa on olla olettamatta mitään. Nopea vilkaisu ohjelmakoodiin ei useinkaan kerro kuin korkeintaan räikeimmät virheet. Tarkastajan tulee olla perillä siitä, mitä kyseisellä tuotoksella ylipäätään haluttiin tehdä, joten jos tarkastaja joutuu varmistamaan tekijältä mitään kyseiseen tuotokseen liittyen, on kyseinen asia syytä dokumentoida erikseen. Tarkastusvaihe on kuitenkin kytköksissä tuotettuun ohjelmakoodiin myös myöhemmässä vaiheessa, joten kaikkea käytyä keskustelua ei tarvitse siirtää ohjelmakoodiin tai sitä vastaavaan työpyyntöön.

Oleellisinta etenkin uuden ohjelmakoodin tarkastamisessa on ajatella tuotettua ohjelmakoodia siltä näkökannalta, mitä tekijä itse ei välttämättä ole huomannut katsoa. Tähän oivallisena tapana on tutustua ensin tehtyyn ohjelmakoodiin ja pyrkiä käyttämään sitä jotenkin, ennen kuin perehtyy asiaan liittyvään dokumentaatioon. Mikäli ohjelmakoodi päätyy esimerkiksi poikkeukseen, on kyseinen osuus todennäköisesti tehty liian rajatulla olettamuksella siitä, miten kyseistä osuutta tultaisiin myöhemmin käyttämään.

Mikäli tehty työ vastaa tarkastavan henkilön mielestä työpyynnössä esitettyjä vaatimuksia, työ yhdistetään (engl. merge) projektin versiohallinnan päähaaraan ja työpyyntö suljetaan. Joissain tapauksissa työpyyntö on laaja kokonaisuus, jolloin tarkastamisen selkeyttämiseksi työ suoritetaan useammassa osassa, näissä tapauksissa tarkastaja merkitsee työpyyntöön, mitkä osuudet ovat tehtyjä. Lisäksi jos tarkastusvaiheessa huomataan, että toimiva ratkaisu vaatii muutoksia myös muihin kokonaisuuksiin, tai se muulla tavoin paljastaa ongelmakohdan itse työpyyntöön liittymättömästä osuudesta, avataan näissä tapauksissa uusi työpyyntö, johon merkitään kaikki siihen mennessä selvinneet seikat kyseisestä tarpeesta.

5.6 Manuaalinen testaus

Vaikka testausputkeen kuuluu monia erilaisia testejä, ne eivät vielä ota kantaa loppukäyttäjän kokemukseen järjestelmän parissa. Tämän vuoksi HRD:n kehityksessä on otettu käytänne, että koko järjestelmä on testattavissa erillisessä testiasennuksessa, johon on pääsy sekä projektin asiakasyrityksellä, että kehitystä tekevällä yrityksellä.

Erillinen testiasennus mahdollistaa eri tietoteknisellä tasolla olevien tutustua järjestelmän käyttöön, ilman vaaraa siitä, että väärinkäsitys tai virheellinen toiminto vaikuttaisi oikeaan, merkitykselliseen tietokantaan. Loppukäyttäjä joka ei tunne järjestelmää tai sen osuutta entuudestaan on myös koettu hyväksi tavaksi mitata käyttöliittymien helppokäyttöisyyttä.

Manuaalisessa testauksessa pääpaino on kuitenkin järjestelmän toimivuuden ja vakauden varmistamisessa. Versiohallinnan päähaaran muuttuessa käynnistyy testausputki, joka päivittää testiasennuksen uusimpaan haaraan, ja viimeistään siinä vaiheessa kun yksittäinen kokonaisuus on saatu hyväksytysti läpi päähaaraan, on aika tehdä täysimittainen tar-

kastus testiympäristössä. Tällöin tarkastus tapahtuu käyttöliittymän kautta, jolloin tarkastetaan esimerkiksi kurssille ilmoittautuminen, siitä syntyvät raporttien osuudet sekä kustannustiedot. Samalla seurataan palvelimen lokitietoja, joista paljastuu sellaiset ongelmat, mitkä eivät välttämättä käyttöliittymästä käsin näy.

Tämä vaihe on koettu erittäin tärkeäksi etenkin niissä tilanteissa, missä tietokantaa on jouduttu päivittämään. Mikäli tietokanta joudutaan päivittämään, eikä erityisen huolellista testausta ole asian suhteen tehty, voi siitä aiheutua tarkastusta suurempi työ, mikäli tietoja joudutaan valikoidusti palauttamaan varmuuskopioista.

5.7 Käyttöönotto

Viimeinen vaihe kehityskulkua ennen uutta kierrosta on käyttöönotto, tällöin ohjelmistosta julkaistaan uusi versio, joka vielä erikseen hyväksytetään testiympäristössä asiakasyrityksellä ja sovitaan ajankohta tuotannon päivittämiseen. Uudesta versiosta tehdään myös yhteenveto, jossa listataan kaikki ne asiat, jotka ovat jollain tapaa vaikuttanut järjestelmän toimintaan.

Poikkeuksena normaalille käyttöönotolle ovat Drupalin sekä sen moduulien tietoturvapäivitykset. Tällaisissa tapauksissa tietoturvapäivitys pyritään saamaan normaalin tuotantopäivityksen käyttöönottoon mukaan, mutta joissain tapauksissa päivitystarve on niin kriittinen, ettei normaalille tarkastuskierrokselle ole aikaa. Mikäli tuotanto joudutaan päivittämään ilman asiakasyrityksen vahvistusta, tällöin tietoturvapäivitys tehdään sille versiolle, joka tuotannossa viimeisenä on ollut käytössä. Tällä on pyritty siihen, että ilman asiakasyrityksen hyväksyntää järjestelmässä ei loppukäyttäjän osalta muutu oleellisesti mitään.

Tuotantopäivityksen jälkeen siirrytään niin sanotulle uudelle kierrokselle, joka tarkoittaa ohjelmistokehityksen puolella järjestelmän valvontaa ja mittausta sekä seuraavan vaiheen toteutuksen suunnittelemista. Käytännössä seuraavan kierroksen työmäärä saattaa olla hyvinkin tiedossa jo edellisellä kierroksella, joissain tapauksissa siihen liittyvä työ saattaa olla osin jopa tehtynä. Oleellista on kuitenkin, että kierroksen alettua siihen ei tuoda enää uusia isompia lisäyksiä. Käynnissä olevaan kierrokseen voi silti tulla korjauksia, täsmen-

nyksiä sekä parannuksia. Näin toimimalla on pystytty määrittelemään jo varhaisessa vaiheessa suuntaa antava aikataulu seuraavan vaiheen valmistumiselle, joka tarkentuu kierroksen edetessä entisestään.

6 POHDINTA

Opinnäytetyön tarkoitus oli tuoda esille Drupal 8 -ohjelmistokehityksen niitä asioita, jotka etenkin Drupal 7:n kanssa aiemmin työskennellyt ohjelmistokehittäjä herkästi jättää huomiotta. Drupal 8 tukee isoa osaa myös Drupal 7 -tyylisistä toimintatavoista, joten ohjelmistokehittäjällä voi olla vaikeuksia hahmottaa, mikä on nykyinen tapa toimia. Työn tavoite saavutettiin esimerkkiohjelman avulla, se korostaa toiminnallisuuksien käärimistä paketteihin, joita tarjotaan palvelujen avulla.

Kokonaisen ohjelmiston päivittämisessä Drupal 7:sta Drupal 8:aan tulisi olla todella hyvä ymmärrys siitä, mitä ohjelmistolla on tarkoitus tehdä. Drupal 8:n ydin on muuttunut niin paljon aiemmista versioista, että hyvin suurta osaa vanhoista liitännäisistä tai edes arkkitehtuureista ei tarvita saman toiminallisuuden toteuttamiseksi. Vahva suositus on etenkin moduulien mutta myös projektien suhteen, että ne kirjoitetaan Drupal 8:n kohdalla kokonaan uudestaan. Ytimen tarjoamia ominaisuuksia hyödyntäen hyvin moni sellaisista asioista, jotka ovat aiemmin vaatineet erikseen ohjelmointia, on nykyään suoraan toteutettavissa käyttöliittymästä käsin.

HRD:n päivitys Drupal 8:aan aloitettiin hyvin varhaisessa vaiheessa, jonka vuoksi monissa tapauksissa dokumentaatiosta oli enemmän haittaa kuin hyötyä. Myös monissa Drupal-moduuleissa on havaittavissa, ettei kehittäjä välttämättä ole tiennyt, kuinka helposti jonkin toiminnallisuuden tai ominaisuuden olisi voinut toteuttaa.

Ohjelmistojen kehittäjien kannattaa tutustua olemassa oleviin Drupal-ytimen ohjelmakoodeihin, kun haluavat toteuttaa jonkin ominaisuuden. Lähes poikkeuksetta kaikki on mahdollista toteuttaa annotaatioiden avulla, jolloin tehtävä ominaisuus on dynaamisesti käytettävissä muuallakin, kuin minne se alun perin on haluttu.

LÄHTEET

Briteweb. Comparison. Luettu 9.4.2018.
https://briteweb.com/ideas/wordpress-vs-drupal-statistics-trends-non-profit/

Coder. Documentation. Luettu 23.3.2018.
https://www.drupal.org/project/coder

Composer. Documentation. Luettu 9.4.2018.
https://getcomposer.org/doc/

Drupal console. Documentation. Luettu 24.3.2018.
https://hechoendrupal.gitbooks.io/drupal-console/

Drupal, About. Luettu 9.4.2018
https://www.drupal.org/about

Drupal. Coding standards. Luettu 23.3.2018.
https://www.drupal.org/docs/develop/standards

Drupal. Testing. Luettu 9.4.2018.
https://www.drupal.org/docs/8/testing

Drupal. Usage. Luettu 9.4.2018.
https://www.drupal.org/project/usage/drupal

Drush. Documentation. Luettu 23.3.2018.
http://docs.drush.org/en/master/

Facets. Documentation. Luettu 9.4.2018.
https://www.drupal.org/project/facets

GitLab. Community Edition. Luettu 9.4.2018.
https://gitlab.com/gitlab-org/gitlab-ce

Mediamaisteri. Verkko-oppimisympäristöt. Luettu 24.3.2018.
https://www.mediamaisteri.com/fi/digitaaliset-oppimisymparistot

PHP. Abstract class. Luettu 28.3.2018.
http://php.net/manual/en/language.oop5.abstract.php

PHP. Class. Luettu 28.3.2018.
http://php.net/manual/en/language.oop5.basic.php

PHP. Interfaces. Luettu 28.3.2018.
http://php.net/manual/en/language.oop5.interfaces.php

PHP-DI. Dependency injection. Luettu 5.4.2018
http://php-di.org/doc/understanding-di.html

Semver. Semantic versioning. Luettu 9.3.2018.
https://semver.org

StackShare. Comparison. Luettu 9.4.2018.
https://stackshare.io/stackups/bitbucket-vs-github-vs-gitlab

WebsiteSetup, Comparison. Luettu 9.4.2018.
https://websitesetup.org/cms-comparison-wordpress-vs-joomla-drupal/

LIITTEET

Liite 1. example.php

```
<?php

/**
 * @file
 * This file contains example use case of rated containers.
 */

/* @var \Drupal\my_module\Controller\RatedContainerController $controller */
$controller = \Drupal::service('rated_container.controller');
/* @var \Drupal\my_module\RatedContainerInterface $fast */
$fast = \Drupal::service('rated_container.primary');
/* @var \Drupal\my_module\RatedContainerInterface $unlimited */
$unlimited = \Drupal::service('rated_container.secondary');

$insert = function ($times) use ($controller, $fast, $unlimited) {
  for ($i = 1; $i <= $times; $i++) {
    echo "Inserting item #{$i} into " . get_class($controller->getCurrentCon-
tainer()) . PHP_EOL;
    $controller->insert($i);
  }
  echo get_class($fast) . ' has ' . $fast->currentItemCount() .
    ' items and ' . get_class($unlimited) . ' has ' .
    $unlimited->currentItemCount() . ' items' . PHP_EOL;
};

$extract = function ($times) use ($controller, $fast, $unlimited) {
  for ($i = 1; $i <= $times; $i++) {
    $item = $controller->extract();
    echo "Extracting item #{$i} from " . get_class($controller->getCur-
rentContainer()) . '. ';
    echo $item ? "Found {$item}." . PHP_EOL : "Oh no, got nothing!" .
PHP_EOL;
  }
  echo get_class($fast) . ' has ' . $fast->currentItemCount() .
    ' items and ' . get_class($unlimited) . ' has ' .
    $unlimited->currentItemCount() . ' items' . PHP_EOL;
};

$items = 0;
while ($controller->extract()) {
  $items++;
}
echo "Removed {$items} old items from containers" . PHP_EOL;

$insert(6);
echo "Should be 5 / 1" . PHP_EOL;
$extract(1);
echo "Should be 5 / 0" . PHP_EOL;
echo "Current is " . get_class($controller->getCurrentContainer()) . PHP_EOL;
$insert(1);
echo "Should be 5 / 1" . PHP_EOL;
$extract(6);
echo "Should be 0 / 0" . PHP_EOL;
```

Liite 2. my_module.info.yml

```
name: 'My module'
type: module
description: 'My Awesome Module'
core: 8.x
package: 'Custom'
```

Liite 3. my_module.module

```
<?php

/**
 * @file
 * Contains my_module.module.
 */

use Drupal\Core\Routing\RouteMatchInterface;

/**
 * Implements hook_help().
 */
function my_module_help($route_name, RouteMatchInterface $route_match) {
  switch ($route_name) {
    // Main module help for the my_module module.
    case 'help.page.my_module':
      $output = '';
      $output .= '<h3>' . t('About') . '</h3>';
      $output .= '<p>' . t('My Awesome Module') . '</p>';
      return $output;

    default:
  }
}

/**
 * Implements hook_theme().
 */
function my_module_theme() {
  return [
    'my_module' => [
      'render element' => 'children',
    ],
  ];
}
```

Liite 4. my_module.services.yml

```
services:
  rated_container.primary:
    class: Drupal\my_module\FastContainer
    arguments: ['@state']
  rated_container.secondary:
    class: \Drupal\my_module\UnlimitedContainer
    arguments: ['@state']
  rated_container.controller:
    class: Drupal\my_module\Controller\RatedContainerController
    arguments: ['@rated_container.primary', '@rated_container.secondary']
```

Liite 5. tests/src/Functional/RatedContainerControllerTest.php 1 (2)

```
<?php
namespace Drupal\Tests\my_module\Functional;
use Drupal\my_module\Controller\RatedContainerController;
use Drupal\my_module\FastContainer;
use Drupal\my_module\UnlimitedContainer;
use Drupal\Tests\BrowserTestBase;
/**
 * @coversDefaultClass \Drupal\my_module\Controller\RatedContainerController
 *
 * @group my_module
 */
class RatedContainerControllerTest extends BrowserTestBase {
  /**
   * {@inheritdoc}
   */
  public static $modules = ['my_module'];
  /**
   * Controller to use in tests.
   *
   * @var \Drupal\my_module\Controller\RatedContainerController
   */
  public $ratedContainerController;
  /**
   * {@inheritdoc}
   */
  protected function setUp() {
    parent::setUp();
    $this->refreshController();
  }
  /**
   * Refresh controller if containers were manipulated elsewhere.
   */
  private function refreshController() {
    $this->ratedContainerController = \Drupal::service('rated_container.con-
troller');
  }
  /**
   * @covers ::insert
   * @covers ::currentItemCount
   */
  public function testInsert() {
    for ($i = 0; $i < 30; $i++) {
      $this->assertEquals($i, $this->ratedContainerController->currentItem-
Count());
      $this->ratedContainerController->insert($i);
      $this->assertEquals($i + 1, $this->ratedContainerController->cur-
rentItemCount());
    }
  }
  /**
   * @covers ::insert
   */
  public function testInsertException() {
    /** @var \Drupal\my_module\FastContainer $fastContainer */
    $fastContainer = \Drupal::service('rated_container.primary');
    $this->assertInstanceOf(FastContainer::class, $fastContainer);
    $this->assertEquals(5, $fastContainer->limit());
    $controller = new RatedContainerController($fastContainer, $fastCon-
tainer);
```

```
    $controller->insert('no exception');
    $controller->insert('no exception');
    $controller->insert('no exception');
    $controller->insert('no exception');
    $this->setExpectedException(\Exception::class);
    $controller->insert('containers are full.');
  }
  /**
   * @covers ::extract
   * @covers ::insert
   * @covers ::getCurrentContainer
   */
  public function testExtract() {
    $primaryValues = ['dog', 'cat', 2, 3, 5];
    $secondaryValues = ['car', 'bike', 100, 200, 300];
    foreach ($primaryValues as $primaryValue) {
      $this->assertInstanceOf(FastContainer::class, $this->ratedContainerCon-
troller->getCurrentContainer());
      $this->ratedContainerController->insert($primaryValue);
    }
    foreach ($secondaryValues as $secondaryValue) {
      $this->ratedContainerController->insert($secondaryValue);
      $this->assertInstanceOf(UnlimitedContainer::class, $this->ratedContain-
erController->getCurrentContainer());
    }
    $this->assertEquals(300, $this->ratedContainerController->extract());
    $this->assertEquals(200, $this->ratedContainerController->extract());
    $this->assertEquals(100, $this->ratedContainerController->extract());
    $this->assertEquals('bike', $this->ratedContainerController->extract());
    $this->assertEquals('car', $this->ratedContainerController->extract());
    $this->assertEquals(5, $this->ratedContainerController->extract());
    $this->assertEquals(3, $this->ratedContainerController->extract());
    $this->assertEquals(2, $this->ratedContainerController->extract());
    $this->assertEquals('cat', $this->ratedContainerController->extract());
    $this->assertEquals('dog', $this->ratedContainerController->extract());
  }
}
```

Liite 6. src/UnlimitedContainer.php

```
<?php
namespace Drupal\my_module;
/**
 * Unlimited container, lacks speed.
 *
 * @package Drupal\my_module
 */
class UnlimitedContainer extends RatedContainerBase {
}
```

Liite 7. src/RatedContainerInterface.php

```php
<?php
namespace Drupal\my_module;
/**
 * Interface for containers.
 *
 * @package Drupal\my_module
 */
interface RatedContainerInterface {
  /**
   * Maximum storage size of this container.
   *
   * @return int
   *   Maximum storage size.
   */
  public function limit();
  /**
   * Insert item into storage.
   *
   * Item may be any object.
   *
   * @param mixed $item
   *   Item to be saved.
   */
  public function insert($item);
  /**
   * Remove and return latest item from storage.
   *
   * @return mixed
   *   Most recent item from storage.
   */
  public function extract();
  /**
   * Amount of items in storage currently.
   *
   * @return int
   *   Amount of items.
   */
  public function currentItemCount();
}
```

Liite 8. src/RatedContainerBase.php 1 (2)

```php
<?php
namespace Drupal\my_module;
use Drupal\Core\DependencyInjection\ContainerInjectionInterface;
use Drupal\Core\State\StateInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;
/**
 * Default behaviour of any storage.
 *
 * Class that extends this one may override any function but desired require-
ment
 * is only to override limit if storage is not unlimited.
 *
 * @package Drupal\my_module
 */
abstract class RatedContainerBase implements RatedContainerInterface, Con-
tainerInjectionInterface {
  /**
   * State service, used to replicate storage.
   *
   * @var \Drupal\Core\State\StateInterface
   */
  protected $state;
  /**
   * Limit of the given container.
   *
   * Defaults to zero, which means unlimited.
   *
   * @var int
   */
  protected $limit = 0;
  /**
   * RatedContainerBase constructor.
   *
   * @param \Drupal\Core\State\StateInterface $state
   *   State service.
   */
  public function __construct(StateInterface $state) {
    $this->state = $state;
  }
  /**
   * {@inheritdoc}
   */
  public static function create(ContainerInterface $container) {
    return new static(
      $container->get('state')
    );
  }
  /**
   * {@inheritdoc}
   */
  public function limit() {
    return $this->limit;
  }
  /**
   * {@inheritdoc}
   */
  public function insert($item) {
```

Liite 8. src/RatedContainerBase.php 2 (2)

```
    if (!($this->limit()) || $this->limit() > $this->currentItemCount()) {
      $this->pushStorage($item);
    }
  }
  /**
   * {@inheritdoc}
   */
  public function extract() {
    $storage = $this->loadStorage();
    $item = array_pop($storage);
    $this->setStorage($storage);
    return $item;
  }
  /**
   * {@inheritdoc}
   */
  public function currentItemCount() {
    return count($this->loadStorage());
  }
  /**
   * Fetch storage content.
   *
   * @return array
   *   All storage items in array.
   */
  private function loadStorage() {
    return $this->state->get(get_class($this)) ?: [];
  }
  /**
   * Add one item to storage stack.
   *
   * @param mixed $item
   *   Item to be inserted in storage.
   */
  private function pushStorage($item) {
    $storage = $this->loadStorage();
    $storage[] = $item;
    $this->setStorage($storage);
  }
  /**
   * Override current storage.
   *
   * @param array $items
   *   Items to be inserted in storage.
   */
  private function setStorage(array $items) {
    $this->state->set(get_class($this), $items);
  }
}
```

Liite 9. src/FastContainer.php

```
<?php
namespace Drupal\my_module;
/**
 * Fast and limited container.
 *
 * @package Drupal\my_module
 */
class FastContainer extends RatedContainerBase {
  protected $limit = 5;
}
```

Liite 10. src/Controller/RatedContainerController.php 1 (3)

```php
<?php
namespace Drupal\my_module\Controller;
use Drupal\Core\Controller\ControllerBase;
use Drupal\my_module\RatedContainerInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;
/**
 * Multi-container controller.
 *
 * This controller allows using two different containers simultaneously.
 * Controller will automatically switch to secondary container once the pri-
mary
 * one is full. Switch happens back to primary once secondary is empty.
 *
 * @package Drupal\my_module\Controller
 */
class RatedContainerController extends ControllerBase {
  /**
   * Primary container.
   *
   * @var \Drupal\my_module\RatedContainerInterface
   */
  private $primary;
  /**
   * Secondary container.
   *
   * @var \Drupal\my_module\RatedContainerInterface
   */
  private $secondary;
  /**
   * Currently selected container.
   *
   * Current is calculated after insert and before extract.
   *
   * @var \Drupal\my_module\RatedContainerInterface
   */
  protected $current;
  /**
   * RatedContainerController constructor.
   *
   * @param \Drupal\my_module\RatedContainerInterface $primary
   *   Primary container.
   * @param \Drupal\my_module\RatedContainerInterface $secondary
   *   Secondary container.
   *
   * @throws \Exception
   *   In case both of the containers is already full.
   */
  public function __construct(RatedContainerInterface $primary, RatedContain-
erInterface $secondary) {
    $this->primary = $primary;
    $this->secondary = $secondary;
    $this->prepareCurrentForInsert();
  }
  /**
   * {@inheritdoc}
   */
  public static function create(ContainerInterface $container) {
```

Liite 10. src/Controller/RatedContainerController.php 2 (3)

```
    return new static(
      $container->get('rated_container.primary'),
      $container->get('rated_container.secondary')
    );
  }
  /**
   * Get currently used container.
   *
   * @return \Drupal\my_module\RatedContainerInterface
   *   Currently used container.
   */
  public function getCurrentContainer() {
    return $this->current;
  }
  /**
   * Insert item into storage.
   *
   * Item may be any object.
   *
   * @param mixed $item
   *   Item to be saved.
   *
   * @throws \Exception
   *   If using two limited containers, inserting item that reaches limits,
   *   will cause exception. Surrounding insert with try-catch will prevent
   *   this from causing problem in case desired next operation is to ex-
tract.
   */
  public function insert($item) {
    $this->current->insert($item);
    $this->prepareCurrentForInsert();
  }
  /**
   * Remove and return latest item from storage.
   *
   * @return mixed
   *   Most recent item from storage.
   */
  public function extract() {
    $this->prepareCurrentForExtract();
    return $this->current->extract();
  }
  /**
   * Amount of items in storage currently.
   *
   * @return int
   *   Amount of items.
   */
  public function currentItemCount() {
    $count = 0;
    $count += $this->primary->currentItemCount();
    $count += $this->secondary->currentItemCount();
    return $count;
  }
  /**
   * Change current container if required for insert.
   *
   * Prefers primary source but switches to secondary if primary is full.
```

```
   *
   * @throws \Exception
   *   If both container has already reached their limits.
   */
  private function prepareCurrentForInsert() {
    if (!($this->primary->limit()) || ($this->primary->limit() > $this->pri-
mary->currentItemCount())) {
      $this->current = $this->primary;
    }
    elseif (!($this->secondary->limit()) || ($this->secondary->limit() >
$this->secondary->currentItemCount())) {
      $this->current = $this->secondary;
    }
    else {
      throw new \Exception('No usable container');
    }
  }
  /**
   * Change current container if required for extract.
   *
   * Changes container back to primary if secondary is currently used,
   * desired next operation is extract and secondary container is empty.
   */
  private function prepareCurrentForExtract() {
    if (($this->current === $this->secondary) && $this->current->currentItem-
Count() === 0) {
      $this->current = $this->primary;
    }
  }
```