Niels Doekemeijer
Camiel Steenstra

OTEC Tool

Orientation

Commissioned by Bluerise and TU Delft

Under supervision of
Paul Dinnissen
Claudia Hauff

May 7, 2013

Preface

This document is the second of three reports. This report describes the orientating research performed for the OTEC Tool. Multiple web frameworks are tested and the most suited one is chosen. This project, conducted as part of the TU Delft Computer Science Bachelor program, is carried out at Bluerise. Bluerise is a technology provider specialized in Ocean Thermal Energy.

More information about the project can be found in the first report, the Project Plan.

We would like to thank Bluerise for giving us the opportunity to perform this project. We would also like to thank our supervisors for guiding us; Paul Dinnissen on behalf of Bluerise and Claudia Hauff on behalf of the TU Delft.

Delft, May 7, 2013

Niels Doekemeijer
Camiel Steenstra

Contents

1 Introduction

Web applications are a hot programming topic. New web applications emerge every day and most of these applications require a wide variety of the same functionalities. In order to speed up the implementation process, programmers can use frameworks. These frameworks take care of common challenges and try to reduce the number of redundant tasks. There are hundreds of frameworks available; choosing the right one is not an easy task. Frameworks differ in functionality, techniques, mindset, style and efficiency. Choosing the right framework eases programming and helps with creating a secure, well performing web application.

In this report, different aspects of frameworks are researched in order to find the best suitable framework for the Bluerise OTEC tool. Comparisons are done on different levels, taking into account the popularity, the online community/support, the documentation and usability. This form of research is loosely based on the framework comparison process described by Raible 3.

Important functionalities for the ideal framework are described in chapter 2. The most popular frameworks are determined in chapter 3 and an initial selection of frameworks will be made. The preselected frameworks will be compared in chapter 4. Finally, the best framework will be recommended in chapter 5.

2 Functionalities

A framework can help with many different aspects of an application. A few components present in most frameworks are briefly discussed. These components take precedence according to the application requirements as discussed in the Project Plan[1].

Database access

When creating an application that contains elements that need permanent storage, a storage system is required. In (large) web applications, the storage system is usually a relational database. Accessing this database can be troublesome and is prone to errors when user input is used. Object Relational Mapping (ORM) provides an extra layer of abstraction between the developer and the database. A good ORM component makes it easy to access the database securely.

Input validation

User input always needs to be validated before it is used. A good structured way of handling validation and providing feedback to the user can save a lot of time. A good framework thus provides an easy format for handling validation.

Design Patterns

There are multiple design patters for software to structure code. For example, a widely used pattern for web development is the Model-View-Controller (MVC) pattern. This pattern separates data (Model) from logic (Controller) and visualization (View). The model stores a data object, while the controller manipulates it. Views visualize this model data and invoke controller actions. Separating these segments of code makes it easier to exchange a segment; e.g. a view specialized for mobile phones.
In web development, this pattern can be simply summarized as follows: the view is the actual web page in the browser (client), database/file interaction (server) is done in the model and the controller performs actions and handles page flow (server).
A good framework follows a suited design pattern and makes this pattern easy to implement. In case of MVC, the framework can provide ORM for the model and templates for the view. Templates ensure the separation of client and server code and only handle the visualization of the data.

[1]Project Plan: the report describing the assignment and approach for the OTEC Tool.

3 Selection

The ideal framework for this project does not have to meet any special requirements. The needed features are common and tons of complying frameworks can be found. However, it is difficult to automatically benchmark security and code quality. Manually checking every framework is time consuming and undesirable.

To address this issue, the team has decided to test only the most popular frameworks. It is assumed that frequently used frameworks are mature and production ready. When there are more developers working on a framework, it is assumed to be better supported and maintained.

Using Ohloh[1] and HotFrameworks[2], a list of actively maintained and widely used frameworks can be composed. To rank these frameworks (listed in Table 3.1), a number of factors can be used. Popularity can be measured by popularity of the programming language, popularity of the framework (web traffic, number of users), number of contributers and the amount of documentation. Using these factors, the frameworks will be ranked and six frameworks will be selected for further testing.

Language Ranking

The first aspect on which frameworks differ is the platform on which they run. As for frameworks, popularity says something about the maturity of a programming language. The language ranking mark is based on the Tiobe ranking ([6], measuring the number of engineers, courses and third party vendors), the RedMonk ranking ([4], using GitHub[3] and StackOverflow[4]) and the Ubuntu Computer Language Benchmark [7].
The final language ranking is calculated by $103 - \frac{Tiobe+RedMonk+Ubuntu}{3}$.

[1] Ohloh.net: Public directory of Free and Open Source Software. Every project is analyzed regularly and statistics like Lines of Code, Active Contributors and Users are measured.[2]
[2] HotFrameworks.com: Relative ranking of web frameworks based on the number of GitHub watchers, site traffic and inbound links. The most popular framework is graded 100.[1]
[3] GitHub.com: Project hosting. A lot of frameworks are hosted here.
[4] StackOverflow.com: A question and answer site for programmers.

Web Traffic

Web traffic for frameworks can be measured using HotFrameworks and Google Trends[5]. This is a simple indication of popularity.
The final web traffic mark is calculated by $\frac{HotFramework+Trends}{2}$.

Community

The size of the community is a more important aspect of popularity. The number of users and the amount of documentation gives a good indication of the size of a community. For the amount of users, Ohloh and GitHub can be used as users can manually 'use' or 'star' a framework. As GitHub is bigger than Ohloh, the GitHub number has a higher weight.
The number of users is estimated by $\frac{Ohloh+3*GitHub}{4}$.

The level of documentation is hard to measure. The number of questions and answers is used to measure the amount of 'help'. This is done by counting the number of questions for a certain framework on Stackoverflow.

Code per Contributor

To get a superficial idea of the quality of a framework and the degree of maintenance that is happening, the number of active contributors can be checked. Because not all frameworks are equally big, the total lines of code is also taken into account. Ohloh provides both of these statistics, so the code per contributer can be calculated using $\frac{LinesofCode}{AciveContributors}$.

Ranking

To come to a final ranking, all frameworks are sorted from 0 to 18 for the five factors (the best framework is ranked 0) and the final score is calculated using $100 - 100 * (\frac{0.25*Language+0.25*Traffic+0.25*Users+Contributors+Help}{52.25})$. Emphasis is on the number of contributors and help. Only the initial values and final score are displayed in Table 3.1, the sorted values are left out.

[5]Google.com/trends: Compare multiple search terms with the Google search engine. Results are relative numbers between 0 and 100, with 100 being most found.

		Language Ranking ↑	Web Traffic ↑	Users ↑	Code per Contributor ↓	Help ↑	Score ↑
Ruby on Rails	Ruby	81	86	13918	206	112921	**92,3**
Django	Python	85	59	4864	592	50431	**85,6**
CodeIgniter	PHP	89	40	4224	1309	18660	**76,1**
Flask	Python	85	48	4331	154	1761	**69,9**
Symfony	PHP	89	46	4948	1684	9886	**68,9**
ASP.NET	C#	88	100	1320	7780	167501	**67**
Laravel	PHP	89	39	3049	149	1050	**64,6**
Sinatra	Ruby	81	34	3630	520	2452	**64,1**
CakePHP	PHP	89	50	3260	4414	13527	**64,1**
Zend Framework	PHP	89	46	3046	6156	14810	**59,8**
Yii	PHP	89	49	1995	3659	4767	**58,4**
Spring	Java	98	43	1565	14557	27306	**53,1**
Grails	Groovy	29	55	642	5840	6009	**51,2**
Kohana	PHP	89	21	953	4280	1614	**44**
Google Web Toolkit	Java	98	27	1304	10939	12757	**44**
Pylons	Python	85	21	120	1746	739	**37,3**
Lift	Scala	60	22	508	7441	854	**28,2**
Zope	Python	85	28	592	8864	446	**23**
Vaadin	Java	98	27	148	85185	903	**22,5**

Table 3.1: Framework comparison

↑ Higher is better | ↓ Lower is better

To limit the number of possible frameworks (as shown in Table 3.1), all scores below 60 will be dropped. Also, as Bluerise is already working with PHP, emphasis will be on PHP frameworks. The second Python and Ruby frameworks, Flask and Sinatra, will therefore not be tested. ASP.NET, running on C# and only supported for Windows platforms, will also be skipped. This leaves six frameworks to be tested more thoroughly.

4 Comparison

In this chapter, the frameworks selected in chapter 3 will be compared. For each of these frameworks, the same web page will be implemented. After these tests in section 4.1, the framework will be rated and a list of pros and cons is given. Performance of the implementation will be tested in section 4.2.

4.1 Test Drive

To get an impression for each of the preselected frameworks, a simple test page will be implemented for every framework. The test page will contain a simple form in which users can submit a message. All the submitted messages will be listed beneath the input form.

For this page to work, a couple of things have to be set up. First of all, the framework has to be installed and a test environment has to be set up. After that, the application has to be configured to use a database for storing the messages. The page is responsible for handling user input and validating that the input is not empty. By implementing this test for the selected frameworks, insight is gained over how the frameworks operate.

All of the frameworks utilize the MVC design pattern. Input handling, database interaction and page generation are covered in this test. Working with the framework also gives the opportunity to evaluate file structure and code documentation. Performance and code quality is measured in in section 4.2. Actual code for the implementations can be found in Appendix A.

Next, a brief list of pros and cons will be given for each framework. Every framework will also be rated zero to five stars for syntax, features, documentation and overall usability. These ratings are subjective and the result of the developers' first impression with the frameworks.

Ruby on Rails

Version:	3.2.13 *(March 18 2013)*
License:	MIT
Platform:	Ruby >= v1.8.7

Syntax	★★★★☆
Features	★★★★☆
Documentation	★★★½☆
Overall	★★★★☆

Pro

- ✓ Very clean, readable syntax.
- ✓ Clear documentation.
- ✓ Console helpers and generators.
- ✓ Database generation and migration (don't lose data with upgrades).
- ✓ Framework and application files well separated.
- ✓ A lot of plugins (Gems) which can be easily added and managed.
- ✓ Used for web applications such as Twitter, Github, Groupon, Hulu and Soundcloud.

Con

- ✗ Difficult to set up for Windows; not all Gems work.
- ✗ Error reports not always clear. As a lot is left to the framework, it is not always clear where bugs originate.
- ✗ Although syntax is readable, it is not very strict and mistakes are easily made.

CodeIgniter

Version:	2.1.3 *(October 8, 2012)*
License:	BSD
Platform:	PHP >= 5.1.6

Syntax	★★☆☆☆
Features	★★☆☆☆
Documentation	★★½☆☆
Overall	★★☆☆☆

Pro

- ✓ Explicit, low level API with a lot of freedom.
- ✓ Database migrations.
- ✓ Very clear documentation.
- ✓ Framework and application files well separated.

Con

- ✗ Limited templating support. Templates are plain PHP by default.
- ✗ Unclear error reporting for templates.
- ✗ Low level database interaction. Models have to be manually implemented with trivial methods.

Django

Version:	1.5.1 *(March 28 2013)*
License:	BSD
Platform:	Python >= v2.6.5

Syntax	★★★★★
Features	★★★★☆
Documentation	★★★★☆
Overall	★★★★☆

Pro

- ✓ Very clean, readable syntax.
- ✓ Clear documentation.
- ✓ Console helpers and generators.
- ✓ Database generation.
- ✓ Clean, readable templates.
- ✓ Small and obvious file structure.
- ✓ Out of the box administration system for database management.
- ✓ Out of the box geographic framework to handle spatially enabled data.
- ✓ Used for web applications such as Disqus, Instagram, Pinterest and Rdio.

Con

- ✗ Difficult to set up with Apache and MySQL.

Laravel

Version:	3.2.14 *(March 21, 2013)*
License:	MIT
Platform:	PHP >= v5.3

Syntax	★★★☆☆
Features	★★★☆☆
Documentation	★★☆☆☆
Overall	★★☆☆☆

Pro

- ✓ Clean, readable syntax.
- ✓ Console helpers and generators.
- ✓ Database migrations.
- ✓ Clean, adaptable templates.
- ✓ Framework and application files well separated.

Con

- ✗ Lack of books and tutorials, but API documentation is available.
- ✗ No out of the box support for model validations. Validating models are a common operation and having it done automatically saves work and ensures valid models.

Symfony

Version:	2.3.2 *(March 24, 2013)*
License:	MIT
Platform:	PHP >= v5.3.2

Syntax	★☆☆☆☆
Features	★★★★☆
Documentation	★★☆☆☆
Overall	★★☆☆☆

Pro

- ✓ Console helpers and generators.
- ✓ Database generation and migration.
- ✓ Development toolbar. This page overlay offers insight in loading speed, memory usage and SQL statements that were executed.
- ✓ Clean, readable templates (based on Django template syntax).
- ✓ Out of the box extensive authentication system.
- ✓ Out of the box administration panel that offers functionality such as user mailing.

Con

- ✗ Counterintuitive syntax. Model classes are overly expressive, error prone and need manual implementation of trivial methods.
- ✗ Separation of logic that should belong in the same file (such as model validation).
- ✗ Unclear file structure. Application and framework code are mixed.
- ✗ Generally unclear error messages. As code logic is separated, it is not always clear what is causing a failure.
- ✗ Overly expressive code. Most lines of code needed for the test application by far.

CakePHP

Version:	2.3.4 *(April 28 2013)*
License:	MIT
Platform:	PHP >= v5.2.8

Syntax	★★★★☆
Features	★★★☆☆
Documentation	★★★★★
Overall	★★★★☆

Pro

- ✓ Clean, readable syntax.
- ✓ Very clear documentation.
- ✓ Console helpers and generators.
- ✓ Database migrations.
- ✓ Framework and application files well separated.

Con

- ✗ Limited templating support, although helpers are available. Templates are plain PHP, but helpers can be used to generate forms or include other templates.

4.2 Performance

Most frameworks perform different tasks in order to serve requested pages. Because these tasks require some form of computation, a framework usually causes time overhead. In this section, the relative speed of each framework is illustrated using data from the TechEmpower web framework benchmark[5]. Another important feature of a framework is the quality of code. A quality review is given for every PHP framework using a PHP inspection tool.

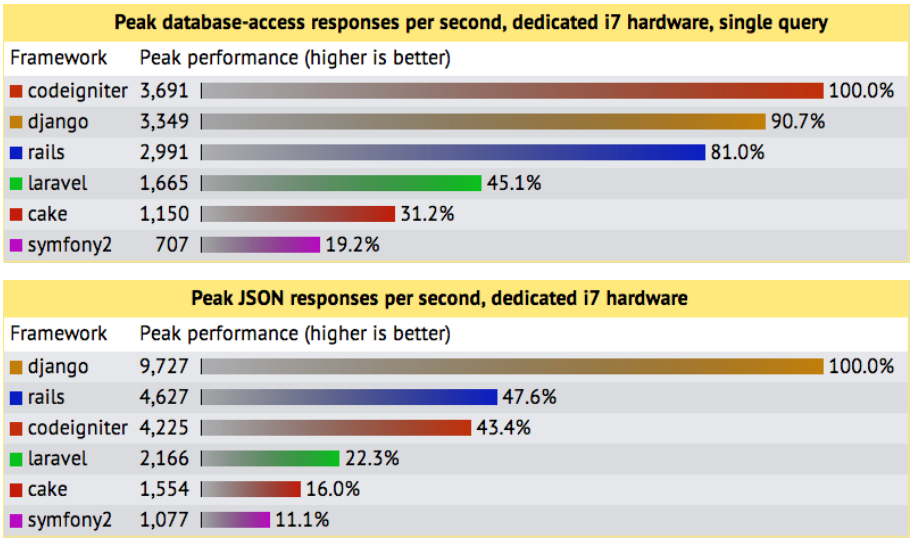

Figure 4.1: TechEmpower Framework comparison[5].

Web Frameworks Benchmark

The benchmark of frameworks focuses on two different tasks, querying a database and serializing JSON. The results shown in Figure 4.1 depict the performance of each framework, where a higher score is better. The results show that some frameworks perform notably better. The difference in performance can also be linked to the number of features each framework has to offer. For example, CodeIgniter has a high score, but the number of features that CodeIgniter offers is less than the other frameworks. Symfony provides much more functionality out of the box, but does not perform well on these tests. The results of these tests will be taken into account in the final decision, but the fact that not every framework does the same amount of legwork should not be forgotten.

Code Quality

To be able to provide a solid base for secure web applications, a framework's code must adhere to high quality. Quality of code is hard to determine, but several signs can be taken into account to get a picture of the overall health of code. Most quality tools use their own signs. This makes it hard to compare different programming languages. For this reason, only the four PHP frameworks will be examined. The tool used to inspect code is PHP Depend[1], this tool produces two charts containing different information. For each PHP framework, a small review based on the results is given. The actual results with detailed explanations can be found in Appendix B.

CakePHP

The results indicate that CakePHP is fairly complex. This means it is harder to maintain CakePHP framework code, which may indicate that it is hard to maintain a high quality when the framework evolves.

CodeIgniter

The results show that CodeIgniter is reasonably complex, but dependencies are high and unstable. Changes in certain packages could easily cause errors in other depending packages. Maintaining CodeIgniter should be doable, but the quality is not ideal.

Laravel

According to the results, Laravel's quality is fairly high, although dependencies are not ideal. There are some unstable packages that might cause problems when changes are made. Maintaining Laravel should not cause too many problems.

Symfony

Symfony has high ratings on structure of code, but the score on a more global level indicates many problems. There are a lot of dependent packages that are not structured to assist in maintaining backward compatibility. According to the tests, Symfony is not very stable and therefore hard to maintain.

[1] pdepend.org: PHP Depend is a tool for generating quality assessment charts on PHP software.

5 Conclusion

Choosing *the best* framework is very hard, perhaps even impossible. Frameworks offer a fast way to set up an application, but the framework that fits a project best will always be a tailor made framework. Besides the framework being of high quality, the quality of the application code also plays a big role in security and maintainability.

This research has highlighted some of the features that are desired in a good framework. It turns out that most of the big frameworks offer functionality, but only few offer usability. Because usability is a subjective measure, *the best* framework is also subjective. For the most suitable framework, developers can try out several frameworks. That is what happened for the Bluerise OTEC Tool.

After testing the frameworks, it became clear that Symfony is not a good contestant for this project. Although it offers a lot of features, usability and performance are severely lacking. CodeIgniter, having decent performance but bad usability, is also not the framework of choice for this project.

The four remaining frameworks can be divided into PHP and non-PHP frameworks. The non-PHP frameworks, Django and Ruby on Rails, generally perform better, but require the extra step of changing programming languages. Currently, Bluerise utilizes PHP as programming language for web development, so picking a PHP framework would reduce installing and learning time. However, picking a non-PHP framework might proof to be more rewarding in the long run.

This research shows that Django would be the framework of choice. It offers a good set of features, while maintaining a clean syntax and good performance. When looking only at PHP frameworks, Laravel appears to be a good all-round framework. It offers a decent set of features, good performance and a maintainable code base, but its documentation is sub par. CakePHP stands out in usability, syntax and documentation, but its code base is relatively complex and harder to maintain. Overall, CakePHP would be the PHP framework of choice.

A Framework Implementations

These are the implementations for the frameworks as described in chapter 4.

Ruby on Rails

Rails Model

```
class Message < ActiveRecord::Base
  attr_accessible :message

  validates :message, :presence => true, :allow_blank => false
end
```

Rails Controller

```
class MessagesController < ApplicationController
  def index
    @message = Message.new(params[:message])
    @message = Message.new if params.has_key?(:message) && @message.save

    @messages = Message.all
    render :index
  end

  def create
    index
  end
end
```

Rails Template Main

```
<!DOCTYPE html>
<html>
<head>
  <title>Bluerise</title>
  <%= stylesheet_link_tag "application", :media => "all" %>
  <%= csrf_meta_tags %>
</head>
<body>
  <div id='main'>
    <div class='border' id='header'>Rails Test</div>
    <%= yield %>
  </div>
</body>
</html>
```

Rails Template View

```
<%= form_for(@message) do |f| %>
  <%= f.text_field :message %>
  <%= f.submit "Save" %>
<% end %>

<div id='messages'>
  <% @messages.each do |message| %>
    <div class="message">
      <%= message.message %>
    </div>
  <% end %>
</div>
```

Django

Django Model

```
from django.db import models

class Message(models.Model):
   message = models.CharField(max_length=255)
   def __unicode__(self):
      return self.message
```

Django Controller

```
from django.shortcuts import render

from messages.models import Message

# Create your views here.
def index(request):
   try:
      message = request.GET['message']
      m = Message(message=message)
      m.save()
   except:
      pass

   messages = Message.objects
   context = {'messages' : messages}
   return render(request, 'messages/index.html', context)
```

Django Template

```
{% load staticfiles %}
<!doctype html>
<html lang=en>
   <head>
      <meta charset=utf-8>
      <link rel="stylesheet" type="text/css" href="{% static 'messages/style.css' %}" />
      <title>Messages</title>
   </head>
   <body>
      <div id='main'>
         <div class='border' id='header'>Django Test</div>
         <form method="get">
            <input type='text' name='message'>
            <input type='submit' value='Save'>
         </form>
         <div id='messages'>
            {% for message in messages %}
            <div class='message'>
               {{message.message}}
            </div>
            {% endfor %}
         </div>
      </div>
   </body>
</div>
```

CodeIgniter

CodeIgniter Model

```
class Message_model extends CI_Model {

   public function __construct()
   {
      $this->load->database();
   }

   public function get_messages()
   {
      $query = $this->db->get("messages");
      return $query->result_array();
   }

   public function set_message()
   {
      $data = array(
         "message" => $this->input->post('message'),
      );

      return $this->db->insert("messages", $data);
   }
}
```

CodeIgniter Controller

```
class Messages extends CI_Controller {

   public function index()
   {
      $this->load->helper('html');
      $this->load->helper('form');
      $this->load->library('form_validation');
      $this->load->model("message_model");

      $this->form_validation->set_rules('message', 'Message', 'required|min_length[1]|max_length[255]|xss_clean');

      if (!($this->form_validation->run() === FALSE))
      {
         $this->message_model->set_message();
      }
      $data["messages"] = $this->message_model->get_messages();
      $this->load->view('base', $data);
   }
}
```

CodeIgniter Template

```
<!DOCTYPE html>
<html lang=en>
   <head>
      <meta charset=utf-8>
      <?php echo link_tag('assets/css/style.css'); ?>
      <title>Messages</title>
   </head>
   <body>
      <div id='main'>
         <div class='border' id='header'>CodeIgniter Test</div>
         <?php echo form_open('messages'); ?>
            <input type='text' name='message'>
            <input type='submit' value='Save'>
         </form>
         <div id='messages'>
            <?php foreach ($messages as $message): ?>
            <div class='message'><?php echo $message["message"]; ?></div>
            <?php endforeach ?>
         </div>
      </div>
   </body>
</html>
```

Symfony

Symfony Model

```php
namespace Bin3\TestBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="messages")
 * Message
 */
class Message
{
    /**
       * @ORM\Id
       * @ORM\Column(type="integer")
       * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
       * @ORM\Column(type="string", length=200)
     */
    private $message;

    /**
     * Get id
     *
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * Set message
     *
     * @param string $message
     * @return Message
     */
    public function setMessage($message)
    {
        $this->message = $message;

        return $this;
    }

    /**
     * Get message
     *
     * @return string
     */
    public function getMessage()
    {
        return $this->message;
    }
}
```

Symfony Controller

```
namespace Bin3\TestBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;
use Bin3\TestBundle\Entity\Message;

class TestController extends Controller
{
    public function indexAction()
    {
         $request = Request::createFromGlobals();
         if (strlen($request->query->get('message')) > 0) {
            $message = new Message();
            $message->setMessage($request->query->get('message'));

            $em = $this->getDoctrine()->getManager();
            $em->persist($message);
            $em->flush();
         }

         $repository = $this->getDoctrine()
            ->getRepository("Bin3TestBundle:Message");
         $messages = $repository->findAll();

         return $this->render('Bin3TestBundle:Test:index.html.twig', array("messages" => $messages));
    }
}
```

Symfony Template Main

```
{# app/Resources/views/base.html.twig #}
<!DOCTYPE html>
<html lang=en>
   <head>
      <meta charset=utf-8>
      <link rel=StyleSheet href="style.css" type="text/css" media=screen>
      {% block stylesheets %}{% endblock %}
      <title>{% block title %}Bla!{% endblock %}</title>
   </head>
   <body>
      <div id='main'>
      {% block main %}{% endblock %}
      </div>
   </body>
</html>
```

Symfony Template View

```
{% extends '::base.html.twig' %}

{% block stylesheets %}
{{ parent() }}
<link rel=StyleSheet href="{{ asset('bundles/test/css/style.css') }}" type="text/css" media=screen>
{% endblock %}

{% block main %}
<div class='border' id='header'>Symfony Test</div>
<form method="get">
   <input type='text' name='message'>
   <input type='submit' value='Save'>
</form>
<div id='messages'>
   {% for message in messages %}
   <div class='message'>
      <p>{{ message.message }}</p>
      <a class='hover-link'></a>
   </div>
   {% endfor %}
</div>
{% endblock %}
```

Laravel

Laravel Model

```
class Message extends Eloquent
{
   public static $rules = array(
      'message' => 'required|min:1',
   );

   public static function validate($data)
   {
      $v = Validator::make($data, self::$rules);
      return $v->passes();
   }
}
```

Laravel Controller

```
class Messages_Controller extends Base_Controller
{
   public $restful = true;

   public function get_index() {
      $messages = Message::all();
      return View::make('messages')
         ->with('messages', $messages);
   }

   public function post_index() {
      $new_message = array(
         'message' => Input::get('message')
      );

      if (Message::validate($new_message)) {
         $message = new Message($new_message);
         $message->save();
      }

      return $this->get_index();
   }
}
```

Laravel Template Main

```
<!doctype html>
<html lang=en>
   <head>
      <meta charset=utf-8>
      <title>@yield('title')</title>

      @section('head')
      <link rel=StyleSheet href="css/style.css" type="text/css" media=screen>
      @yield_section
   </head>
   <body>
      <div id='main'>
         <div class='border' id='header'>Laravel test</div>
         @yield('main')
      </div>
   </body>
</html>
```

Laravel Template View

```
@layout('template')

@section('title')
Messages
@endsection

@section('main')
{{ Form::open() }}
{{ Form::text('message') }}
{{ Form::submit('Save') }}
{{ Form::close() }}

<div id='messages'>
   @foreach ($messages as $message)
   <div class='message'>
      {{ $message->message }}
   </div>
   @endforeach
</div>
@endsection
```

CakePHP

CakePHP Model

```
class Message extends AppModel {
    public $validate = array(
        'message' => array(
            'rule' => 'notEmpty'
        )
    );
}
```

CakePHP Controller

```
class MessagesController extends AppController {
    public $helpers = array('Form', 'Session');

    public function index() {
        if ($this->request->is('post')) {
            $this->Message->create();
            if (!$this->Message->save($this->request->data)) {
                $this->Session->setFlash('Unable to add your message.');
            }
        }

        $this->set('messages', $this->Message->find('all'));
    }
}
```

CakePHP Template Main

```
<!doctype html>
<html lang=en>
   <head>
      <title><?php echo $title_for_layout?></title>
      <?php
      echo $this->Html->css('default');
      echo $this->fetch('meta');
      echo $this->fetch('css');
      echo $this->fetch('script');
      ?>
   </head>
   <body>
      <div id='main'>
         <div class='border' id='header'>CakePHP test</div>
         <?php echo $this->fetch('content'); ?>
      </div>
   </body>
</html>
```

CakePHP Template View

```
<?php
echo $this->Form->create('Message');
echo $this->Form->text('message');
echo $this->Form->end('Save');
?>

<div id='messages'>
   <?php foreach($messages as $message): ?>
   <div class='message'>
      <?php echo h($message['Message']['message']); ?>
   </div>
   <?php endforeach; ?>
</div>
```

B Framework Quality

These are the detailed results of the quality assessment of the frameworks as done in section 4.2.

Overview Pyramid

The first chart generated by PHP Depend is the Overview Pyramid. This pyramid shows different values that are present in the system. These values contain: Number Of Packages (NOP), Number of Classes (NOC), Number Of Methods (NOM), Lines of Code (LOC), Cyclomatic Complexity (CYCLO), Distinct function calls (CALLS), Type references (FANOUT), Average Number of Derived Classes (ANDC) and Average Hierarchy Height (AHH). Some of these values can be set against each other, giving interesting ratios. For example, dividing the number of methods by the number of lines of code results in the average number of lines per method. A high number indicates that methods are complicated while methods should just perform a simple task. Each of these ratios are colored according to average results, green indicates a good value.

Abstraction Instability Chart

The second chart generated by PHP Depend is the Abstraction Instability Chart. This chart contains information about the packages within a system and their dependencies. Two measurements are used for each package: instability and abstractness. Instability takes into account the ratio between other packages being dependent on the subject package and the subject package being dependent on other packages. In this ratio, 1.0 means that a package has no incoming dependencies but is itself dependent on other packages. The other end of this ratio, 0.0, means that the package is not dependent on other packages but other packages are dependent on this subject package. The second ratio is abstractness, this indicates the ratio between abstract classes and implemented classes. Abstract classes are classes that describe certain functionalities but do not perform the functionality itself. By using abstractions, the implementation can be decoupled from the initial idea for what a class should do. The main philosophy in this chart is that packages that have a high number of incoming dependencies should use more abstraction to ensure decoupled code. This way, changes to the implementation of one package will be less likely to effect the behavior of others.

CakePHP

The Overview Pyramid B.1 shows that CakePHP is fairly complex. This means it is harder to maintain CakePHP framework code, which may indicate that it is hard to keep high quality when the framework evolves. The Abstraction Instability chart B.2 shows that CakePHP combines unstable code with a low level of abstraction, which also indicates that the maintainability is not very high.

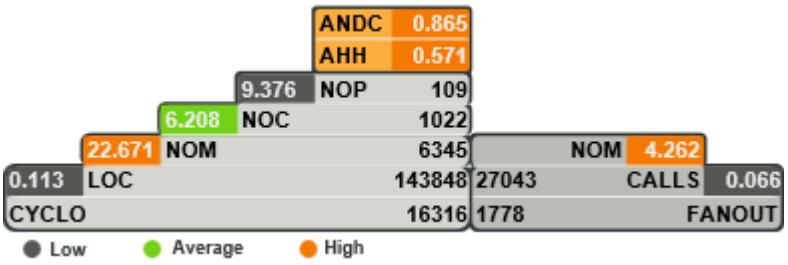

Figure B.1: CakePHP Overview Pyramid, generated by PHP Depend

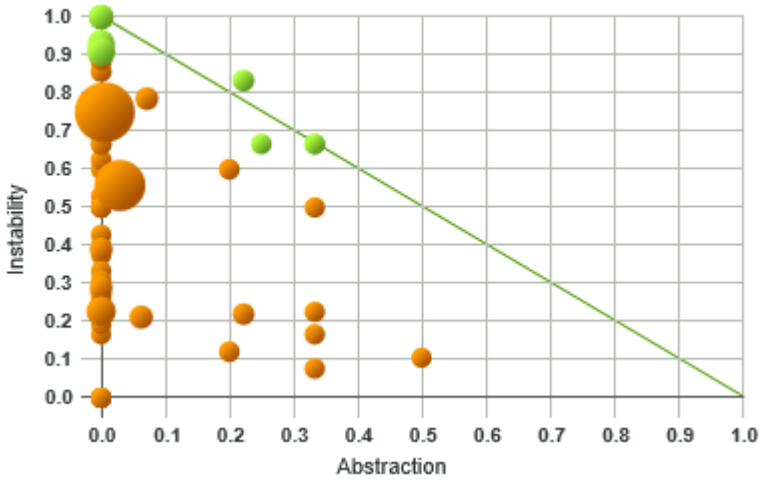

Figure B.2: CakePHP Complexity Chart, generated by PHP Depend

CodeIgniter

In the Overview Pyramid B.3 CodeIgniter scores reasonable, although methods are somewhat complex and classes are too big. According to the Abstraction Instability chart B.4, CodeIgniter uses no abstraction and fairly unstable packages. Changing these unstable packages could easily cause errors in the depending packages. Maintaining CodeIgniter should be doable, but the quality is not ideal.

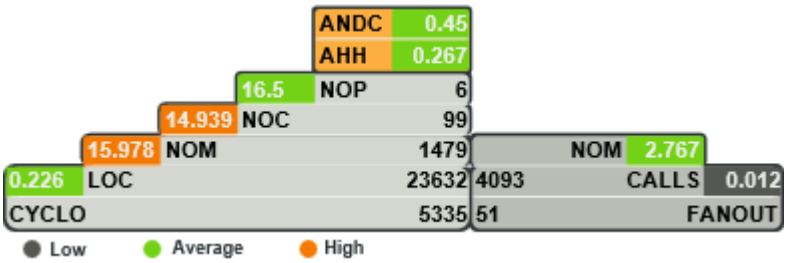

Figure B.3: CodeIgniter Overview Pyramid, generated by PHP Depend

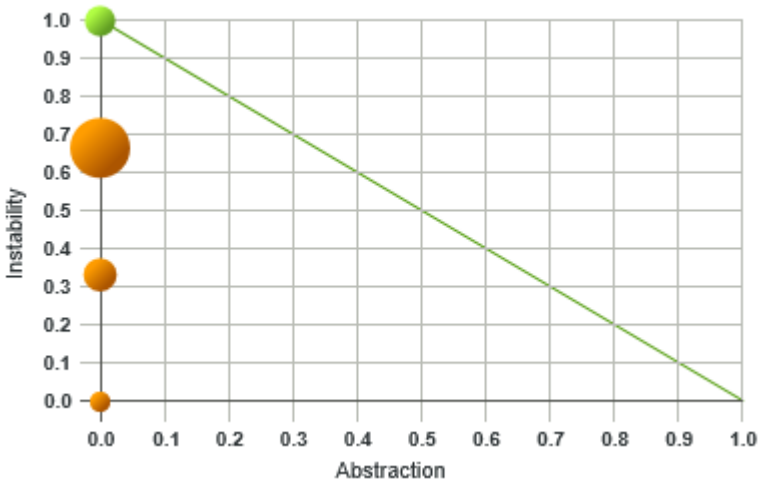

Figure B.4: CodeIgniter Complexity Chart, generated by PHP Depend

Laravel

According to the Pyramid Overview B.5, Laravel's quality is fairly high. Most of the indicators fall within the desired range, although the complexity of inheritance is a little too high. The Abstraction Instability charts B.6 shows that packages are somewhat unstable, while little abstractions are used. Maintaining Laravel should not cause too many problems.

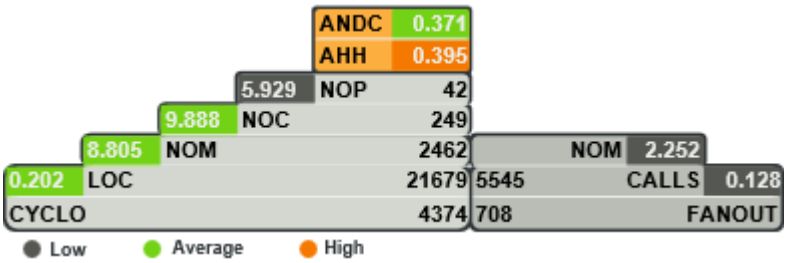

Figure B.5: Laravel Overview Pyramid, generated by PHP Depend

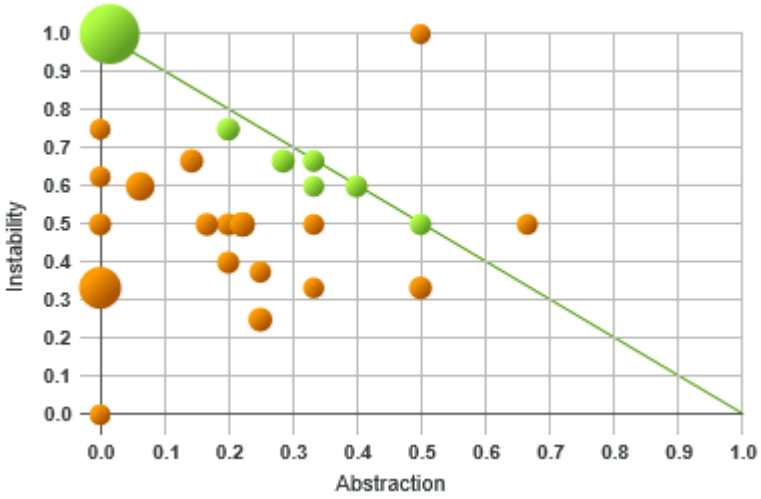

Figure B.6: Laravel Complexity Chart, generated by PHP Depend

Symfony

The Pyramid Overview B.7 for Symfony indicates that the quality is comparable to Laravel's. However, the Complexity Chart B.8 indicates that Symfony has a high number of packages and most of these packages do not fall within the desired range. This means that Symfony is not very stable and therefore hard to maintain.

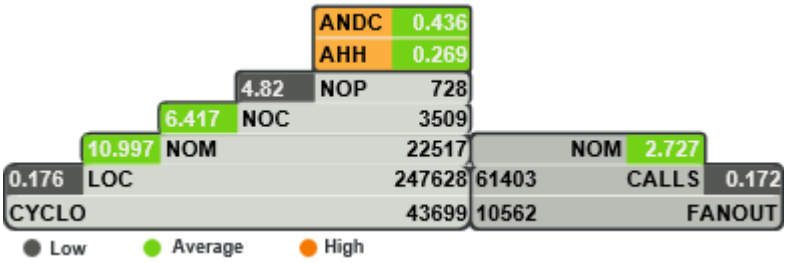

Figure B.7: Symfony Overview Pyramid, generated by PHP Depend

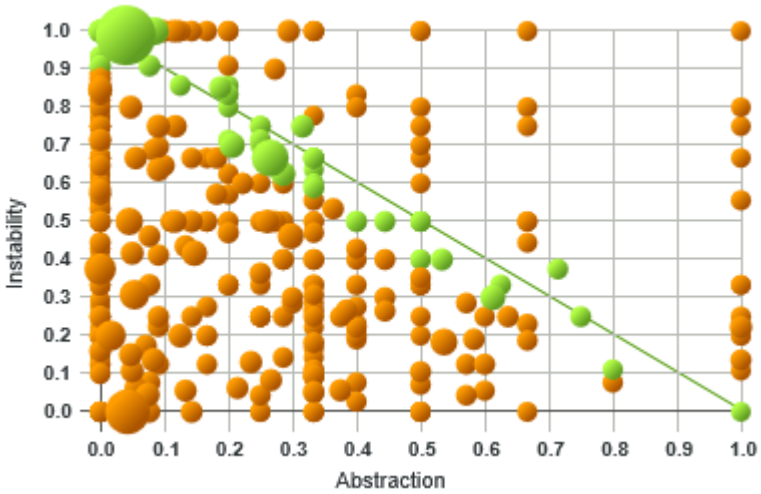

Figure B.8: Symfony Complexity Chart, generated by PHP Depend

Bibliography

[1] HotFrameworks. Web framework popularity rankings, May 2013. URL http://hotframeworks.com/rankings.

[2] Ohloh. Open source web frameworks sorted on number of users, May 2013. URL https://www.ohloh.net/p?query=web+frameworks&sort=users.

[3] Matt Raible. Comparing jvm web frameworks, November 2010. URL http://www.slideshare.net/mraible/comparing-jvm-web-frameworks.

[4] RedMonk. Programming language rankings: January, 2013. URL http://redmonk.com/sogrady/category/programming-languages/.

[5] TechEmpower. Benchmark on different web frameworks (round 4), May 2013. URL http://www.techempower.com/benchmarks/#section=data-r4.

[6] TIOBE. Programming community index for april, 2013. URL http://www.tiobe.com/tpci.htm.

[7] Ubuntu. The computer language benchmarks game, May 2013. URL http://benchmarksgame.alioth.debian.org/.