Preface

This document is the last of three reports. 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.

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, June 26, 2013

Niels Doekemeijer
Camiel Steenstra
## Contents

### Introduction

#### I Orientation

1 Assignment 5
   1.1 Client 5
   1.2 Problem definition 6
   1.3 Objectives 7
   1.4 Assignment definition 7
   1.5 Project status 7
   1.6 Deliverables 9
   1.7 Requirements 10

2 Approach 14
   2.1 Methodology 14
   2.2 Technology 14
   2.3 Proceedings 16
   2.4 Planning 16

3 Quality assurance 19
   3.1 Quality and testing 19
   3.2 Documentation 20
   3.3 Version control 20
   3.4 Risk factors 20

4 Framework selection 22
   4.1 Selection 22
   4.2 Comparison 25
   4.3 Choice 29

#### II Implementation

5 User registration 32
   5.1 Sign up 32
   5.2 Sign in 32
   5.3 Roles 33

6 Database 34
   6.1 Database design 34
   6.2 Floating point coordinates 35
Part

Introduction
OTEC

OTEC, short for Ocean Thermal Energy Conversion, is a renewable source of energy that takes advantage of the solar energy absorbed by the oceans. Electricity is generated by exchanging heat of the warm water ocean surface with the cold water from the deep ocean. Due to the needed temperature differential, OTEC is primarily viable in equatorial areas. A main advantage is that it is available day and night, something that is not evident for wind and solar energy, alternatives in the region.

Bluerise

Bluerise, started in 2009 and located in Delft, is a technology provider in the emerging OTEC market. It aims to advance and commercialize Ocean Thermal Energy. One of the services that Bluerise wants to provide is an easy assessment tool for potential customers. This tool should give a quick indication of whether the technology is viable in a certain area or place.

Project

This project, conducted as part of the TU Delft Computer Science Bachelor program, involves the design and implementation of the Bluerise OTEC assessment tool. The tool builds upon a prototype and should integrate with the Bluerise assessment toolkit.

Report

The development of this project has been carried out in roughly three stages. This report will discuss each of them. In the first stage, laid out in Part I, the problem definition was established and research was carried out about the use of third party tools. During the orientation, the main goal was to determine, as accurately as possible, the requirements that the final product should fulfill. Common meetings with the Bluerise staff were key. After orienting, development could commence. During this second stage, described in Part II, the developers faced newly emerged requirements and programming challenges. By frequently meeting with the Bluerise staff, the opinions of the company could be taken into account during development. In the final stage (Part III) a reflection was made on the overall process.
Part I

Orientation
Before working on the actual application, the team spent two weeks orienting. In these first two weeks, two reports were produced. The orientation phase ended with the project plan, containing a detailed description of the assignment and approach, and a toolkit research report, containing a comparison of possible frameworks with their pros and cons.

The problem definition is addressed in chapter 1, and requirements and objectives are laid out. Then chapter 2 focuses on the proposed approach, describing the methodology, technology, and planning. A description of quality assurance methods is made in chapter 3.

The framework comparison and selection of tools is addressed in chapter 4. A selection is made of plausible frameworks before testing a subset and picking the final framework.
1 Assignment

The OTEC Tool project aims to help Bluerise on different levels, eventually resulting in benefits for both Bluerise, their customers and Ocean Thermal Energy Conversion (OTEC) in general. First a description of Bluerise is given in section 1.1. Then the problems and objectives will be discussed in section 1.2 and section 1.3. The assignment, its current progress and the requirements will be given in section 1.4, section 1.5, section 1.6 and section 1.7.

1.1 Client

Bluerise is a company concerned with OTEC. Using OTEC, the difference in temperature at the ocean’s surface water and the deeper water can be used to generate electricity or to obtain cold water which can be used in various different ways. OTEC provides an interesting durable energy resource which is yet to be exploited. Currently, Bluerise is attempting to demonstrate this technique using pilot facilities demonstrating the wide spectrum of applications for which this technique can be used. These pilot projects are one way for Bluerise to increase the global interests in their innovative technology, however, assessing the usability of these techniques in certain areas is not easy for an untrained individual. Although the pilot projects can hint possible clients about the profitability that can be achieved, it is not easy to determine the real financial and technical implications that an OTEC installation could have on certain sites. Because unknown parameters like financial information, e.g. current energy prices, are not always available for every area, it is hard to determine whether certain areas could make profit when using OTEC. Concluding, there are two problems that are central to this project: informing clients and gaining insight in the market. Using an online tool, both Bluerise and its customers should be able to gain insight in OTEC possibilities.

OTEC

About 70 percent of all sunlight received by Earth falls on oceans. The energy from sunlight falling on oceans causes upper layers of water to increase in temperature. With OTEC, the energy involved in this process which would otherwise just go to waste, can be intercepted, giving rise to a clean source of energy. The interception is done using an exchange in heat between colder deep-sea water and warmer surface water. The technique can be used in different scenarios which will be shortly discussed, but first the general idea behind OTEC is explained.
Technique

OTECH can be applied in areas where the difference between cold reachable water and surface water is at least 20 degrees Celsius. Central in this system is a working fluid, which must be a fluid that has a boiling point beneath the surface temperature. The surface temperature is then used to boil this working fluid. The transition between the fluid state and gas state of this working fluid, the evaporation, can be used to propel a generator. Using the energy that was generated, cold water can be retrieved from deep-sea through long pipes using pumps. This cold water is then used to condense the working fluids gas, which is then again used in the evaporation stage. This cycle results in a positive energy bias which can be used for many different things.

Advantages of OTEC

OTECH provides a number of advantages compared to fossil fuels and traditional durable energy sources. First of all, there is no chemical waste involved in the process. Unlike other durable energy solutions like solar panels or wind turbines, OTECH can provide a highly reliable base load power with little fluctuation. The process can continue even when the sun is down. This aspect of the technology is even more appreciated on tropical islands, where power grids very often lack the ability to store large quantities of power.

1.2 Problem definition

The first and main application of OTECH is generating energy. Taking into account the total area in which this technique is applicable, OTECH could provide roughly 6000 times the global energy needs. The second application is Seawater Air Conditioning (SWAC). With this variant cold water is pumped to shore where it is then used for cooling air. This technique enables energy savings to up to 90% compared current air conditioning energy costs. Other applications include water desalination and agriculture irrigation.

Variants

There are two variants of OTECH installations: onshore and offshore. The onshore variant has its installation placed on shore. The installation requires pipes going to warm and cold areas of the ocean. These pipes follow the continental slope until a depth is reached where the water is cold enough to enable the OTECH cycle. The problem with this variant is determining whether it is feasible to install pipes going to cold water, this could be distanced too far off shore making the particular spot unsuitable. The other variant uses an installation that is placed offshore. This second variant uses pipes that go vertically. Here, only energy is transported to shore. Even though this last variant has less limitations considering distance to the shore, not every spot is suited.
1.3 Objectives

The web site should enable Bluerise to grant possible clients an assessment regarding the application of the technology and financial applicability. In this assessment, certain aspects of the calculations are presumed, and users should be able to adjust these assumptions to better represent their needs. These adjustments can be used by Bluerise, to determine areas of interest. The overall end goals of this product are to:

- provide an information source for potential customers;
- provide insight in customer needs;
- gather data about areas of interest;
- demonstrate possibilities of OTEC.

Besides these product objectives, the resulting system must adhere to high quality standards and must be easy to maintain. Using common software techniques, the project must yield a solid and robust base for further expansion by other developers.

1.4 Assignment definition

In this project, the goal is to create a solid and secure base for the OTEC assessment tool. Part of this project is determining the different tools that will be used by the final product. These choices will be motivated using system requirements and scientific research, resulting in the answer to what tools are best suited for this situation. The system will use a back end that will handle the geographic calculations. This back end, created using Generic Mapping Tools (GMT), is already implemented and will not be a main subject of this project, although possibly some optimization will take place when time allows it. The main part of this project is concerned with building upon a visual design of the OTEC online application. This design contains a usable layout and user work flows. More detail about this is given in section 1.5. Besides some extra work on this layout, a robust underlying system of the application needs to be created.

1.5 Project status

Last year, a group of designers studied the interface design for this application. The result of this study is the ManyHands report. This report contains the visual design of the application, together with the user work flow. The design report contains information about different pages and gives insight in functionality that should be present. After this report was finished, a prototype has been built containing some of the functionality. This section briefly discusses the content of these components.

1 gmt.soest.hawaii.edu: A collection of tools for manipulating geographic data sets.
ManyHands report

The tool described in the ManyHands report revolves around the idea that users step through different stages of project creation. The first step allows users to play with the different parameters in either onshore or offshore projects, this stage is called the practice stage. Once the user feels comfortable with the environment, he or she can progress to the next stage. In this stage, an area and different parameters concerning the area can be selected. This area is assessed by the system’s back end, which leads to the next step, the project page. The project page shows more detailed information about the area. From here, the same practice page can be accessed with the parameters set to the correct parameters. These so called projects can be saved to user accounts, which allows the users to view previously made projects. Two pages of this report are shown in Figure 1.1 and Figure 1.2.

The prototype

The prototype of the tool was used for presentations and as a proof of concept. Although this prototype suffices to give a good first impression to the possibilities of such a tool, it is not yet ready for a public release. There is still functionality that is not yet implemented, and current functionality implementations are not built with expansion, security, scalability and stability in mind. This project will implement the tool with these values in mind. The prototype has shown that the idea is viable, but now the system should be made for the public. No code of this prototype will be reused because the code lacks the proper structure and quality. In the next two chapters, the two components listed here are used to compose requirements.

Figure 1.1: Onshore page design
1.6 Deliverables

The resulting product can be divided in roughly five sections. Each of these sections will now be shortly discussed. The exact requirements will be listed in 1.7. Besides these five main components, there will be a homepage. This homepage contains information about the application, instructions and links to the five components listed below.

**Onshore/Offshore playground tool**

The onshore playground tool is a tool that can be used to get a feel for the different parameters that determine whether a particular site can be used for OTEC. In this tool, users can play with different parameters like water temperature, distance to shore, energy demands and other parameters that influence finance-ability. According to the parameters specified by the user, the tool will generate an approximate rating of the particular site regarding the financial situation and the geographical situation. Both the onshore and offshore tools offer similar functionality but differ in parameter types and layout.

**Project assessment tool**

The project assessment tool focuses on a single OTEC project. In this assessment, a user specified area is assessed by the back end. The results can then be further inspected and adjustments can be made to different project parameters. The tool enables a user to easily configure the onshore or offshore playground tool to represent the given location. Besides the practice pages representing this project, further detailed information can be inspected. The user is also able to generate a pdf report and either directly download this report or have it sent by mail.
User registration system

When registered, a user can save projects so they can be viewed later. The user system must be prepared to handle functionality that might be added when the system evolves. These possible future functionalities includes things like sharing projects between users.

Community map tool

The community map is focused on displaying projects. With this tool, users can view areas of the map and look for projects that other users have made public. This part of the application provides a way for different stakeholders to determine if there is a general interest in OTEC in a specific area.

Back end system for gaining insight in gathered data

This feature is concerned with displaying gathered data so that valuable information can be discovered. This section will only be available to those with enough rights within the system (Bluerise employees). User interface is not important for this part of the application.

1.7 Requirements

This section contains the requirements to which the final product must adhere. These requirements form a basis on which the completeness of the final product can be tested. The requirements consist of functional requirements, constraints and non-functional requirements. The functional requirements consist of requirements that can be linked to specific functionality. These requirements can be tested by simply checking for the functionality in the final product. The constraints pose features to which the system must adhere. These constraints include boundaries on technologies that must be tested and supported by the application. Testing these constraints is done by running the applications on technologies that lie on these boundaries. Non-functional requirements are harder to check, but form directions in which the system must be developed.

The functional requirements of this project will be prioritized according to the MoSCoW priority model, in which each functionality is sorted in either of the following categories: Must have, Should have, Could have, Won’t have. The functionalities will be listed under the components in which the functionality can be accessed.

- Must have
  - Home page
    1. General information about the OTEC tool
    2. Instructions for use of this application
    3. Links to the tools
  - Onshore/Offshore practice tool
    4. Displaying project parameters
    5. Adjustable parameters
6. Adjustable energy/SWAC demands
7. Display evaluation in form of site/financial score
8. Instant project evaluation after adjusting parameters

- Project assessment tool
  9. Choosing a location using:
     a) Location name
     b) Location coordinates
     c) Picker with range selection
10. Choosing project type (Onshore/Offshore)
11. View available OTEC locations in the area
12. Onshore/Offshore views of the project with filled in parameters
13. See detailed information gathered by the back end
14. Generate PDF file containing an overview of the project

- User registration system
  15. Sign up
  16. Sign in/out
  17. View list of saved projects
  18. Open/edit previous projects

- Back end system
  19. Use Google Analytics with extended tracking

• Should have

- Onshore/Offshore practice tool
  20. Adjust assumptions
  21. Feedback on parameter ranges (visual indications)
  22. Display plant costs
  23. Display pipe costs

- Project assessment tool
  24. Choose locations for on and offshore installations
  25. Save personal project (with a given name)
  26. Clone other projects to user account
  27. Change initial project parameters
  28. Set project as public
  29. Get link to project for easy sharing

- User registration system
  30. Edit user settings
  31. Change project names

- Back end system
  32. Plot all projects on a map to determine areas of interest
  33. Plot user locations on a map
  34. Sent emails to users about assessments or other notifications
• Could have
  – Onshore/Offshore practice tool
    35. Extra visual feedback when changing parameters (e.g. change length of pipe on screen)
  – Project assessment tool
    36. See a list of users interested in your own projects
  – User registration system
    37. Ability to sort projects on screen
  – Community map tool
    38. Declare interest in public project
    39. Get into contact with creators or interested users of public projects
    40. Display map with public projects in an area
    41. Open public projects from the map
  – Backed system
    42. Displaying data gathered through data mining so that a quick assessment can be made about the users interests.

• Won’t have
  – Onshore/Offshore practice tool
    43. Support for extra OTEC purposes, such as desalination and irrigation
    44. Display emission savings
    45. Prompt for current energy usage data
  – Project assessment tool
    46. Use assumptions given by users for certain areas
  – User registration system
    47. Collaborated projects

Constraints
The next list contains a set of constraints, these constraints represent minimum conditions under which the system must operate.

1. The product must work on modern browsers (desktop as well as mobile)
   a) Chrome (version 16+)
   b) Microsoft Internet Explorer (version 8+)
   c) Mozilla Firefox (versions 10+)
   d) Safari (version 5+)
   e) Android standard browser (Android 4 and higher)
2. An overflow of users requesting heavy calculations should not cause the system to crash, the system must be able to handle at least 20 simultaneous requests. (This is aimed at the project assessment tool)

3. The interface should be fully viewable on resolutions of at least 1024 * 768

4. The interface must be operable on touch screens

Non-functional requirements
The final list consists of non-functional requirements that should be apparent in the final product. These final requirements can not be linked to specific functionality nor is testing easy, but they are worth mentioning.

1. Consistent interface
2. The interface should be based on the Many Hands report
3. Security; the system is secure and is prepared for standard web application digital attack attempts
4. Testability; the system has been tested extensively and this the results provide insight in product quality
5. Usability; the system is clear and provides enough feedback for untrained users to retrieve information they need
6. Maintainability, the system is structured and modular, providing good means for further maintaining and expending the code
This chapter describes the approach the team will take to successfully complete the project. Methodology and technology will be outlined in section 2.1 and section 2.2. The chapter ends with a description of the proceedings in section 2.3 and planning in section 2.4.

2.1 Methodology

Two developing phases can be distinguished in the project. The first phase consists of research and laying out the assignment, while the actual implementation of the assignment is done in the second phase.

The second phase of the project will be executed using an agile software development method called Scrum. Instead of planning every step of the development process in advance, Scrum tries to be agile and flexible. The Scrum methodology will not be strictly followed, because the team of developers and managers is relatively small. Relevant for this project is mainly the use of sprints and the aspect of always having a working product.

A sprint is a development effort for a short time period. The duration is fixed and will be one week in the case of this project. Each sprint starts with a meeting in which tasks are identified and rated for difficulty. The team sorts the tasks on priority and determines how much of these "sprint goals" are feasible for the coming sprint. The team then agrees on a division of tasks for each team member and development starts. Each day of development, the team comes together for the "daily scrum" to quickly discuss problems, progress and plans.

At the end of a sprint, there is a review meeting. There should always be some form of working product after a sprint. The idea is that you start with a working product, extend this product with new features and then end up with an improved working product. The product is reviewed with team members and supervisors. After the review, the cycle starts again with the composition of tasks for the next sprint. Each cycle ends with a short summary of the sprint with a report containing the progress and problems.

2.2 Technology

The OTEC tool for Bluerise is an application for the web. There are three major components with different functionalities. There is a component for the geographic computations, a component for serving data to the browser, and a user interface component.
Geographic computations

The technology for performing the geographic computations on the back end is provided by Bluerise. These computations are relatively time-consuming and may be optimized in the future. Currently, GMT is used to perform a lot of the calculations. No results from this back end are cached. Using PostGIS\textsuperscript{1} an extension for PostgreSQL\textsuperscript{2} some of the calculations could be optimized and cached. Bluerise is working on these improvements. Although not essential for the final product, the team can help with these improvements.

Server

The server is responsible for the serving the web pages. For the client it is a gateway to information and results of computations. The server is operated by Linux (CentOS\textsuperscript{3}) and dynamic pages are generated using PHP\textsuperscript{4}. PHP, a dynamic programming language, has an active development community with a lot of available tools and frameworks to aid development. As part of the assignment, the team will research different options and will pick the most suitable toolkit.

Client

The client, the user interface, is an interactive web page. It displays information and communicates with the server. No calculations are performed on the side of the client. Pages should behave properly on recent versions of the most used desktop and mobile browsers. Older browsers will not be tested; a list of compatible browsers is given in section 1.7.

The web pages are served as HTML, the main markup language for creating web pages. The appearance of the HTML file, which mainly focuses on text and structure, is done use Cascading Style Sheets (CSS). Both standards are supported by all web browsers and are maintained by the World Wide Web Consortium (W3C\textsuperscript{5}), the international standard organization for the World Wide Web.

To create interactive web pages, JavaScript (JS) code is added to the web pages. JavaScript is a programming language for client side scripting. Instead of letting the server generate every web page, the browser can directly modify the web page when, for example, a button is pressed. This lowers server load and results in faster response times for the user. JavaScript can communicate with the server using the AJAX standard, this way the web page can load new information in the background and adjust the page when information has arrived. When using AJAX, the server does not have to send entire web pages. Bypassing these redundancies will result in decreased server load.

Just as the server side, there are a lot of available tools and frameworks for the client side and the team will research the different options. When picking the most suitable option, emphasis is on security, ease of use and browser support.

\textsuperscript{1}PostGIS.net: A spatial database extender for PostgreSQL.
\textsuperscript{2}PostgreSQL.org: A powerful, open source object-relational database system.
\textsuperscript{3}CentOS.org: An enterprise-class Linux distribution.
\textsuperscript{4}php.net: A widely-used scripting language that is especially suited for web development.
\textsuperscript{5}w3.org: An international community which develops web standards.
2.3 Proceedings

The project runs for 11 weeks. In this time, multiple reports, a presentation and the final product should be produced. The team consists of two Computer Science students, one project supervisor (Bluerise) and one project adviser (TU Delft).

Work space is provided at Bluerise and the TU Delft, but there is no fixed workplace and there are no fixed working hours. Bluerise also provides a testing server. Work is shared using a private Git repository hosted on GitHub.6

Meetings with the project supervisor will be at least once a week to discuss progress. Meetings with the project adviser will be at least once a week in the early stages of the project, but may be reduced to once every two weeks in later stages of development. Both adviser and supervisor are easily accessible by bringing a visit, calling or sending an email.

2.4 Planning

This project plans to run over a 11 week period (from April 22th until July 5th 2013), ending with a final product presentation in the last week. In the meantime, several deadlines have been set by the team:

- **Project plan**: Assignment definition and approach. *(26-04-2013)*
- **Orientation report**: Toolset research. *(03-05-2013)*
- **SIG Code evaluation**: Evaluation of the code quality. *(14-06-2013)*
- **Final report**: Evaluation of the project. *(21-06-2013)*
- **Presentation**: Product demonstration. *(01-07-2013 — 05-07-2013)*

Two phases of development can be distinguished: an orientation phase and a development phase. For the first phase, assignment definition and orientation, two weeks have been allocated. This phase ends with the completion of the project plan and the orientation report. Development starts on May 6th and consists of 6 Scrum iterations. After that, there will be two weeks to add finishing touches to the product and prepare the final report and presentation.

---

6GitHub.com: Project hosting for collaborative development.
Milestones

For the development phase, there is no strict planning. Following the Scrum methodology, goals will be set at the beginning of each week. The following list of milestones, ordered by priority, will be used as reference:

1. **First page:** Software is installed and prepared. Home page is working.

2. **Onshore/offshore playground:** The playground, with adjustable parameters, is working and can evaluate parameters.

3. **Area selection:** The user can search for an area and view/adjust this area with an interactive map. The background can perform the geographic calculations for this area.

4. **Project view:** The detailed results of the calculations are presented and specialized onshore/offshore views are available.

5. **Prototype:** The system offers at least the functionality of the old prototype. This covers all above milestones.

6. **User registration:** Users can register, sign in, edit their profile and save projects.

7. **Project sharing:** Users can view detailed information of published projects, clone these projects to their own log. PDF files can be generated for projects.

8. **Assumption adjustment:** Users can adjust assumptions for calculations. New assumptions are stored for statistics.

9. **System overview:** Administrators can view detailed information about how the system is used. Activity, areas of interest and custom assumptions become clear.

10. **Community map:** Users can publish their projects and view locations of published projects in a community map.
Schedule

An overview of chronological events is displayed in Table 2.1. This schedule is not fixed and some dates may shift later on in the project. However, it does give a general idea of the planning and structure.

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22-04-2013</td>
<td>Start project plan</td>
</tr>
<tr>
<td>1</td>
<td>24-04-2013</td>
<td>Project plan draft</td>
</tr>
<tr>
<td>1</td>
<td>26-04-2013</td>
<td>Finalize project plan</td>
</tr>
<tr>
<td>1</td>
<td>26-04-2013</td>
<td>Start orientation phase</td>
</tr>
<tr>
<td>2</td>
<td>03-05-2013</td>
<td>Finalize orientation report</td>
</tr>
<tr>
<td>2</td>
<td>03-05-2013</td>
<td>Start implementation phase</td>
</tr>
<tr>
<td>3</td>
<td>06-05-2013</td>
<td>Start Scrum 1, finalize milestone 1</td>
</tr>
<tr>
<td>3</td>
<td>10-05-2013</td>
<td>End Scrum 1</td>
</tr>
<tr>
<td>4</td>
<td>13-05-2013</td>
<td>Start Scrum 2</td>
</tr>
<tr>
<td>4</td>
<td>17-05-2013</td>
<td>End Scrum 2, finalize milestone 5</td>
</tr>
<tr>
<td>5</td>
<td>20-05-2013</td>
<td>Start Scrum 3</td>
</tr>
<tr>
<td>5</td>
<td>24-05-2013</td>
<td>End Scrum 3, finalize milestones 6, 7</td>
</tr>
<tr>
<td>6</td>
<td>27-05-2013</td>
<td>Start Scrum 4</td>
</tr>
<tr>
<td>6</td>
<td>31-05-2013</td>
<td>End Scrum 4, finalize milestone 8</td>
</tr>
<tr>
<td>7</td>
<td>03-06-2013</td>
<td>Start Scrum 5</td>
</tr>
<tr>
<td>7</td>
<td>07-06-2013</td>
<td>End Scrum 5, finalize milestone 9</td>
</tr>
<tr>
<td>8</td>
<td>10-06-2013</td>
<td>Start Scrum 6</td>
</tr>
<tr>
<td>8</td>
<td>14-06-2013</td>
<td>End Scrum 6, finalize milestone 10</td>
</tr>
<tr>
<td>8</td>
<td>14-06-2013</td>
<td>Deadline SIG code evaluation</td>
</tr>
<tr>
<td>9</td>
<td>24-06-2013</td>
<td>Finalize final report</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Applying finishing touches and SIG feedback</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Prepare presentation</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>Presentation</td>
</tr>
</tbody>
</table>

Table 2.1: Project schedule
3 Quality assurance

Quality of software is something that must be ensured and proven by the developers. Since the OTEC web tool will include a user registration system, it is especially important that the security of the system is high and modules act as they are supposed to. Using different tools and techniques, the project team can increase quality and security. This section will discuss the different efforts that will be made in order to reach these objectives.

3.1 Quality and testing

Quality of software is not easy to assess. It is impossible to test every possible input or to test whether the system is fully secured. However, it is possible to increase quality of code by using proven techniques. Before these techniques are discussed, some more detail is given about what quality of code means.

Quality

Quality of code can be tested in two different ways. First of all, the quality of code can be tested towards functional requirements. In this case, the code is tested against what the code is supposed to do. A high rating on this quality dimension is necessary, low ratings suggest that the system does not function as expected. The second dimension is the structural quality of code. While code can function as it should, the code can still have low quality. The structure of the code determines how well the system can be maintained. Ordered structures enable developers previously unknown with the system to quickly get the picture regarding the system structure. Since an objective in this project is creating a robust solid base for further development, it is apparent that both of these qualities should clearly be present in the resulting product.

Testing

The functional quality will be tested by using unit tests. Web development provides a platform to a wide variety of devices. Although this property of web development is desirable, this also comes with a downside. Many possible configurations of devices make it even harder to test whether the application is working properly. Because of this difficulty, the testing will not only contain unit tests, but also regular behavior tests on different setups. Finally, behavior tests are conducted by the client.
Evaluation

To gain more insight into the quality of the code that this project team will produce, the code will be evaluated by the Software Improvement Group. This group specializes in code quality and uses different tools to assess complexity and structure of code. The SIG evaluation will provide the team with pieces of code that are open for improvement, which can then be incorporated in the final product.

3.2 Documentation

Since the code is likely to be used in further development, readability is of the essence. To ensure that future developers understand what the code does, documentation will be maintained throughout the project in the form of comments. Beside this documentation in code, underlying motivations of different choices within the system will be highlighted and explained. Future developers can consult this report or possibly contact the developers.

3.3 Version control

To further ensure structure in the work flow a version control system will be used. This system allows the team to work on code at the same time, without interference and while keeping structure at a high level. Git, the used software, offers functionality like merging and branching code. This helps the process of Scrum, in which each feature is implemented by a single member or a small group of team members. Using branches, the team members can work without affecting the other members’ work. Besides enabling a version environment, GitHub also provides a ticketing system. This system allows the developers to create tickets for tasks and problems concerning the application. By keeping the division of labor inside this system it will be less likely that two team members will accidentally work on the same task.

3.4 Risk factors

In this section, situations that may negatively influence the projects progress will be discussed. In order to get this project finished in time, a tight schedule needs to be followed. There is potential risk that some unforeseen event will disturb the progress, causing delays and other harms to this project.

Third party technology

In this project a wide variety of different tools will be used. Each of these third party tools introduce risks that can harm the security of the system. To ensure that these risks are decreased as much as possible, these third party software systems need to be kept as up to date as possible. Besides using up to date software, the selected tools should adhere to high security standards and should be widely used. The project team must assure that the code is maintainable and organized. The resulting product must emphasize that third party software should be up to date.
Requirements

When composing system requirements, there is a risk of omitting important system specifications. To ensure that the project supervision and the project team are on the same level regarding the resulting product, a list of requirements is given in [1.7]. This list consists of easy to verify checkpoints that can be used when evaluating the final system. By creating a list that is checked by both parties, comprehensiveness is assured as much as possible. Besides the initial efforts to ensure solid requirements, frequent meetings will be arranged with the client. In these meetings, project supervision can address issues in the current design to better fit the requirements.

Software development skills

Development teams always face the danger of incompetent team members. To ensure that every team member efforts will suffice to the project’s demands, tasks are split up. By splitting up tasks, each team member can be held accountable if something does not pass the quality control. Furthermore, a TU Delft associated supervisor will guide the project. This supervisor will ensure that the teams’ progress is satisfactory and will raise the alarms if this is not the case. Besides lacking software development skills, there is a possibility that the relatively small team has taken a project that is too big to handle. The specifications under Must have are going to end up in the final system no matter what. Other categories will be implemented when time allows it.

Continuous changes

As mentioned in the previous subsection, the requirements may not be comprehensive. During meetings, changes can be suggested that could alter the system architecture. During the project’s runtime, Bluerise will also discuss the OTEC tool with another company, Deltares. During this collaboration, more requirements might emerge that could alter the system’s back end. To prevent that changes cause other parts of the systems to fail, preventative measures will be taken. One such measure is using standard software engineering patterns to keep code modular. When choosing a framework, the requirement of modular code is also taken under consideration. This modularity and quality of code is further assured by feedback provided by the Software Improvement Group (SIG). This feedback will yield helpful commentary with which the development team can further improve the system’s maintainability.

Political risks

In software development projects, there is a risk of the two involved parties (the developers and the client) to collide. When development does not go according to the client’s vision, the development team should undertake actions to lead the project in the right direction. The client can help this process by giving honest opinions and feedback during meetings. In the end, the client can decide whether or not they will actually use the system or discard the project team’s efforts.
4 Framework selection

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 features. 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.

Important features for the ideal framework are described in Appendix A. The most popular frameworks are determined in section 4.1 and an initial selection of frameworks will be made. The preselected frameworks will be compared in section 4.2. Finally, the best framework will be recommended in section 4.3.

4.1 Selection

The needed features for this project 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 and HotFrameworks a list of actively maintained and widely used frameworks can be composed. To rank these frameworks (listed in Table 4.1), a number of factors can be used. Popularity can be measured by popularity of the programming language, popularity of the framework (web

---

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 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.
traffic, number of users), number of contributors 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 and StackOverflow\(^4\)) and the Ubuntu Computer Language Benchmark \([7]\). The final language ranking is calculated by \(103 - \frac{\text{Tiobe} + \text{RedMonk} + \text{Ubuntu}}{3}\).

**Web traffic**

Web traffic for frameworks can be measured using HotFrameworks and Google Trends\(^4\). This is a simple indication of popularity. The final web traffic mark is calculated by \(\frac{\text{HotFramework} + \text{Trends}}{2}\).

**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 \(\frac{\text{Ohloh} + 3\times\text{GitHub}}{4}\).

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 contributor can be calculated using \(\frac{\text{Lines of Code}}{\text{Active Contributors}}\).

\(^3\)StackOverflow.com: A question and answer site for programmers.

\(^4\)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.
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 \(100 - 100 \times (0.25 \times \text{Language} + 0.25 \times \text{Traffic} + 0.25 \times \text{Users} + \text{Contributors} + \text{Help})\). Emphasis is on the number of contributors and help. Only the initial values and final score are displayed in Table 4.1 the sorted values are left out.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Language</th>
<th>Web Traffic</th>
<th>Users</th>
<th>Code per Contributor</th>
<th>Help</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ruby on Rails</td>
<td>Ruby</td>
<td>81</td>
<td>86</td>
<td>13918</td>
<td>206</td>
<td>112921</td>
</tr>
<tr>
<td>Django</td>
<td>Python</td>
<td>85</td>
<td>59</td>
<td>4864</td>
<td>592</td>
<td>50431</td>
</tr>
<tr>
<td>CodeIgniter</td>
<td>PHP</td>
<td>89</td>
<td>40</td>
<td>4224</td>
<td>1309</td>
<td>18660</td>
</tr>
<tr>
<td>Flask</td>
<td>Python</td>
<td>85</td>
<td>48</td>
<td>4331</td>
<td>154</td>
<td>1761</td>
</tr>
<tr>
<td>Symfony</td>
<td>PHP</td>
<td>89</td>
<td>46</td>
<td>4948</td>
<td>1684</td>
<td>9886</td>
</tr>
<tr>
<td>ASP.NET</td>
<td>C#</td>
<td>88</td>
<td>100</td>
<td>1320</td>
<td>7780</td>
<td>167501</td>
</tr>
<tr>
<td>Laravel</td>
<td>PHP</td>
<td>89</td>
<td>39</td>
<td>3049</td>
<td>149</td>
<td>1050</td>
</tr>
<tr>
<td>Sinatra</td>
<td>Ruby</td>
<td>81</td>
<td>34</td>
<td>3630</td>
<td>520</td>
<td>2452</td>
</tr>
<tr>
<td>CakePHP</td>
<td>PHP</td>
<td>89</td>
<td>50</td>
<td>3260</td>
<td>4414</td>
<td>13527</td>
</tr>
<tr>
<td>Zend Framework</td>
<td>PHP</td>
<td>89</td>
<td>46</td>
<td>3046</td>
<td>6136</td>
<td>14810</td>
</tr>
<tr>
<td>Yii</td>
<td>PHP</td>
<td>89</td>
<td>49</td>
<td>1995</td>
<td>3659</td>
<td>4767</td>
</tr>
<tr>
<td>Spring</td>
<td>Java</td>
<td>98</td>
<td>43</td>
<td>1565</td>
<td>14557</td>
<td>27306</td>
</tr>
<tr>
<td>Grails</td>
<td>Groovy</td>
<td>29</td>
<td>55</td>
<td>642</td>
<td>5840</td>
<td>6009</td>
</tr>
<tr>
<td>Kohana</td>
<td>PHP</td>
<td>89</td>
<td>21</td>
<td>953</td>
<td>4280</td>
<td>1614</td>
</tr>
<tr>
<td>Google Web Toolkit</td>
<td>Java</td>
<td>98</td>
<td>27</td>
<td>1304</td>
<td>10939</td>
<td>12757</td>
</tr>
<tr>
<td>Pylons</td>
<td>Python</td>
<td>85</td>
<td>21</td>
<td>120</td>
<td>1746</td>
<td>739</td>
</tr>
<tr>
<td>Lift</td>
<td>Scala</td>
<td>60</td>
<td>22</td>
<td>508</td>
<td>7441</td>
<td>854</td>
</tr>
<tr>
<td>Zope</td>
<td>Python</td>
<td>85</td>
<td>28</td>
<td>592</td>
<td>8864</td>
<td>446</td>
</tr>
<tr>
<td>Vaadin</td>
<td>Java</td>
<td>98</td>
<td>27</td>
<td>148</td>
<td>85185</td>
<td>903</td>
</tr>
</tbody>
</table>

Table 4.1: Framework comparison

↑ Higher is better | ↓ Lower is better

To limit the number of possible frameworks (as shown in Table 4.1), 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.
4.2 Comparison

To get an impression for each of the frameworks selected in section 4.1, 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 Model-View-Controller design pattern (more on this in Appendix A). 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 Appendix B. Actual code for the implementations can be found in Appendix C.

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 | ★★★★☆
| 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 | ★☆☆☆☆
| 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

<table>
<thead>
<tr>
<th>Version:</th>
<th>1.5.1 (March 28 2013)</th>
</tr>
</thead>
<tbody>
<tr>
<td>License:</td>
<td>BSD</td>
</tr>
<tr>
<td>Platform:</td>
<td>Python &gt;= v2.6.5</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Version:</th>
<th>3.2.14 (March 21, 2013)</th>
</tr>
</thead>
<tbody>
<tr>
<td>License:</td>
<td>MIT</td>
</tr>
<tr>
<td>Platform:</td>
<td>PHP &gt;= v5.3</td>
</tr>
</tbody>
</table>

**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) | Syntax | ★☆☆☆☆ |
| License: MIT | Features | ★★★★☆ |
| Platform: PHP ≥ v5.3.2 | 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) | Syntax | ★★★★☆ |
| License: MIT | Features | ★★★★☆ |
| Platform: PHP ≥ v5.2.8 | 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.
4.3 Choice

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.

After discussing the options with Bluerise, the team has decided to pick CakePHP for the implementation of the OTEC tool.
Part II

Implementation
Implementation

This part of the report is about the implementation phase of the project. Weekly progress is described in the iteration reports. These can be found in Appendix D. In the following chapters, a few design decisions will be further explained.

First, chapter 5 elaborates on the user registration system. Database design decisions are described in chapter 7. To make sure the server does not overload, a queue system has been designed. The assessment queue is described in chapter 7. Finally, some comments on code quality will be given in chapter 8, including a section discussing feedback from the Software Improvement Group.
5 User registration

A core feature of the application is the registration and login system. A lot of other functionalities rely on the correct and secure working of it. Due to the active community for CakePHP, a lot of plugins are available. However, a quick assessment made it clear that there are no user plugins available for CakePHP which offer fitting functionality and proper security. For this reason we have decided to implement the functionality ourselves.

5.1 Sign up

To make sure users can be contacted by Bluerise, new users must sign up with a valid email address. After sign up, the account must be activated through email. To achieve this, the server generates a random token and sends this to the user via email. The account can only be activated using this token, so the user must have a valid email address to receive the token and activate his/her account.

Resetting a password works in a similar way. The application assumes a valid email address, so a token to request a new password will be sent via email. After validating this token (done by clicking a link), the user will be sent a new password via email. The user is not sent a new password right away, because this would allow users to reset passwords for others.

There are no plain passwords stored in the database. Only a hashed representation of the password is stored. This hash can be used to validate the password, but cannot be reversed to uncover the original password. The Blowfish hashing algorithm is used with a random salt. Adding a salt ensures that even if passwords were similar, their hash will appear different, rendering a brute force attack to recover passwords almost impossible.

5.2 Sign in

Sign in is part of the base template and is available everywhere through a pull down menu, as shown in Figure 5.1. It was the desire of Bluerise that people could also sign in using third party accounts. When users use a third party account, for example a Google or Facebook account, they do not have to register and they are able to log in with one click.

To implement this feature, HybridAuth\(^1\) was used. This library abstracts the connection between the application and the social identity providers. Hybri-

\(^1\)GitHub.com/hybridauth/hybridauth: Open source social sign on PHP Library
dAuth supports a large number of providers, but the OTEC tool can only use providers that are able to provide an email address.

The application lets HybridAuth connect to the social provider and retrieve the email. If there is no user in the database with that email address, a new user is created. The user will be logged in as if with a password; there is no extra social information stored.

![User drop down menu for signing in, available throughout the site.](image)

### 5.3 Roles

To allow for different kinds of users, each user is assigned a role. This role is basically a clearance level, meaning that a user with a higher role has access to all the lower features, plus some extra or extended features.

When a user registers, it is automatically assigned the 'registered' role. Other roles have to be assigned manually by someone who has the proper qualification.
6 Database

An important part of an application is the permanent storage. The storage device used in this application is MySQL, a relational database system. Within a relational database, tables are used to store data. This data can be queried using SQL (Structured Query Language). In this application, a number of choices have been made regarding the database design. These choices have to do with performance and CakePHP limitations. In section 6.1, a design decision regarding data storage is discussed. This is followed by section 6.2 in which a flaw is discussed that was present using floating point values to store geographical coordinates. A visualization of the database tables and their relations can be found in Appendix E.

6.1 Database design

An important design decision in this application is the fact data storages are not connected to a user, but to a location. This way, data calculated by the back end can be reused for different users. Although chances of two independent users choosing exactly the same sites or projects are slim, a couple of measures were taken within the application in order to increase reusability. Two of these measures will now be discussed, followed by the benefits these measures offer.

Project bounds

In the application, users have the opportunity to assess areas and create sites in these areas. Users can select project bounds using a Google Maps interface, making the assessment area highly adjustable. For this reason, it is unlikely that two users choose exactly the same project bounds. Therefore, project bounds are snapped to a wider area. This causes small modifications in project bounds to snap to the same area, making it more likely generated data is reused.

Site locations

The application contains functionality for free movement of site locations. This functionality however is not accessible for the average user. These users will only receive predefined site recommendations. These recommendations follow from back end calculations. This will result in less frequent occurrences of sites that have not been assessed before.

Benefits

The system implemented for reuse of data has a number of benefits for both users and Bluerise. First of all, users who pick areas that have already been
assessed before, experience significant faster loading times. In such a case, no back end assessment has to be done. This is the case for both projects and sites. Secondly, administrators within the application have the ability to clone projects to other users. This allows Bluerise employees to hand out example projects to clients. Because data can be reused, cloning can be done instantly without having to run an assessment. Last but not least, reusing data saves a lot of computation time and storage space.

6.2 Floating point coordinates

Coordinates are numbers with a high precision in which a small deviation can be noticeable. For this reason, coordinates should not be stored in MySQL Float or Double fields, as these fields are approximate values. The coordinates should be stored in Decimal fields, in which no floating point is present and data is stored exactly the way it is entered.

An issue that came to light after the database structure had been implemented, was the fact that the CakePHP ORM cannot handle decimal fields in the database. This meant that decimal fields were silently converted to float fields. The bug was quickly found during development. In order to fix it, problematic models that store coordinates were adjusted to use a mechanism that converts coordinates to integer values. The input is then stored in an Integer MySQL field. The mechanism also converts the integers back to decimal values when the data is retrieved. This mechanism abstracts the data conversion and allows for precise storing of coordinate data regardless of CakePHP’s inability to handle decimal fields.
7 Assessment queue

Assessments for an area of site are made on the server using a toolkit provided by Bluerise. These assessments are resource consuming and running more than one assessment at once can jeopardize the server’s stability.

7.1 Command queue

Every assessment is divided into several console commands. These commands are stored in the database. After processing, the results are stored as strings in the database. This allows for caching and prevents the same command from running more than needed. The commands, and their results, are location based. This means that multiple users can use the cached results.

The queue works according to the First In, First Out (FIFO) principle. This means that the first queued task gets processed first.

7.2 Stability

Having the assessment scripts run in a separate environment improves server stability. Now, the server does not have to wait for the assessment to finish and it does not have to keep open the connection to the user.

On the server side, the queue is implemented using a Cron job. A Cron job is a command that gets executed at a fixed interval. In this case, the server will check for a queued task every 3 seconds. When an assessment is already running, it will not begin another one.
8 Code quality

Code quality was an important aspect during implementation. With the techniques described in chapter 2 the team tried to uphold a maintainable and secure code base. First, this chapter will discuss the external quality assessment made by the Software Improvement Group in section 8.1. Then, section 8.2 will elaborate on the used testing methods.

8.1 Software Improvement Group feedback

As part of the quality evaluation, the Software Improvement Group (SIG) has evaluated the application’s code. SIG noted that the quality of the software is above average. The feedback provided will be shortly discussed. An explanation will be given as to why the particular comments were made and how the team used this feedback to improve the code.

Voor Module Coupling wordt er gekeken naar het percentage van de code wat relatief vaak wordt aangeroepen. Normaal gesproken zorgt code die vaak aangeroepen wordt voor een minder stabiel systeem omdat veranderingen binnen dit type code kan leiden tot aanpassingen op veel verschillende plaatsen. Binnen dit systeem valt met name op dat er binnen de JavaScript enkele grote bestanden zijn, voorbeelden zijn 'base.js' en 'index.js'. In beide gevallen lijkt het erop dat er verschillende type functionaliteit wordt geïmplementeerd in het bestand. Zo is er in 'base.js' code te vinden voor zowel de 'onshore' als de 'offshore' functionaliteit. Daarnaast valt op dat de er verschillende bestanden zijn met dezelfde naam ('manage.js', 'index.js'), dit maakt het voor toekomstige ontwikkelaars moeilijker om de juiste functionaliteit terug te vinden. Het is aan te raden om zowel de naamgeving als de locatie van de afzonderlijke functies kritisch te bekijken.

This piece of feedback concerns Module Coupling, in which an assessment is done of code stability. A comment is made on the size of some JavaScript files. The comments concern files 'base.js' and 'assessment/index.js'. We agree with the comments provided that these files contain too big of an assortment of logic. In order to reduce file size and increase readability the contents of these files will be split over more appropriate files. We do want to clear up that the 'base.js' file contains calculations for assessing scenarios, these calculations are given by Bluerise, which is the reason why it was put in a single file. Secondly, the calculations in these files are likely to be removed from JavaScript altogether in the future.
The second comment about Module Coupling concerns file names. Currently, JavaScript files have the name of their corresponding controller action and are located in a directory with the name of the controller. So for example, the action manage for the projects controller has its own JavaScript file. This file is placed in the 'js/projects' folder and has the name named 'manage.js'. The comment made by SIG suggests this naming convention could be confusion for future developers and that file names should be unique over the entire system. We partly agree with this comment and in order to make file names more meaningful, the files names will now include the controller name. In the example given above the full path will now be 'js/projects/projects\manage.js'. This will result in a slight redundancy regarding the path and file name, which is the main reason for files not being named like this in the first place. However, this convention will result in a much clearer file structure in which ambiguities are ruled out.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de 'update'-methode in de 'ProfilesController', zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Aparte blokken van statements, zoals hier het daadwerkelijk aanpassen van de user, zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

The next piece of feedback concerns the size of functions. The given example is indeed too long. A fix is checking the entire code for long functions and refactoring them into smaller ones. Although we do not disagree with this comment, we also believe that function size is something that was put on high priority within the system and overall we are pleased with function size and the resulting readability. The comment will be taken into account and we are pleased these remarks are the only comments that SIG has offered.

8.2 Testing

During development, there was much emphasis on testing. Important code segments were thoroughly tested with unit tests. Although the interface was properly tested by hand by multiple people, time did not allow to write specific user interface tests. Future tests could focus more on this aspect of testing, perhaps even with automated software (such as Crawljax\(^1\)). Another aspect that could be looked into is performance testing.

As can be seen in figures 8.1 and 8.2 important code segments are well tested. The test coverage in percentage in Figure 8.1 shows the percentage of code that has been executed after all the tests have been executed. With a

\(^1\)Crawljax.com: a web crawler for automatically crawling and analyzing dynamic Ajax-based Web applications.
coverage of almost 90%, the ProjectStorage model has the lowest coverage. An important part of the application, the user system, has a 99% model coverage and a 91% controller coverage. This means that most of the functionality is tested, except for some edge cases. The application is tested above average, as SIG noted, but there is still room for improvement.

When looking at Figure 8.2, the green color stands out when compared to the red. This is a good sign, as it confirms the percentage coverage report. The three files that could receive some extra test attention are the AssessmentsController, UsersController and ProjectStorage files. At the time of writing the entire test suite included 335 test methods and 1105 assertions.
Part III

Reflection
Reflection

This part of the report will reflect on the project. A reflection will be made on task distribution, collaboration and the overall process in Chapter 9. The chosen framework, as described in Chapter 4 will be reviewed in Chapter 10, again listing the pros and cons.

Finally, the initially established requirements will be revisited in Chapter 11. For each requirement, it will be made clear whether it is implemented or why it is lacking from the resulting system.
9 Process

During this project the development methodology Scrum was used. In this chapter, first a reflection is made on the overall Scrum process. After that, there will be a short discussion on the way tasks were divided and managed. This will be followed by a section about the collaboration while writing code and the efforts made to increase code awareness and keep quality high using code reviewing. Finally, this chapter will end with a section about the weekly meetings with Bluerise.

9.1 Agile development

Since this project was carried out by a small development team, not all Scrum elements could be used as it was meant to be used. For some elements the team size was beneficial, while for others it was not. One beneficial thing was the communication. During development, frequent small meetings took place. Since every developer was present at each of these meetings, no information was left out for other developers. The small team size allowed us to quickly discuss important decisions. Another advantage was task management. The division of labor was an easy thing to do since in most cases each of the developers already tended to certain tasks. On other tasks, where both developers were interested, the division proceeded peacefully.

One downside of the small team was the fact that some issues were rather big. Combined with iterations of just one week, this caused the developers to only be able to focus on a small amount of tasks. This is just a logical consequence of the fact that a small team has less combined productivity than a bigger team, but fast progress can be motivating while slow progress can be discouraging. Finally, it could be said that a team of two is a little too small to utilize Scrum.

9.2 Task management

Dividing tasks requires no additional tools, but managing these tasks in a structured way can be a big improvement to the team's productivity. The GitHub issuing system has been used extensively for this reason. At the start of every iteration, the team had a meeting where the issues were made for the main components that would be implemented. During development, additional issues were added when newly emerged requirements came up, as well as issues for bugs or malfunctions. This system also allows for milestones to be made, on which issues can be adhered. This provided a structured overview of what had to be done in each iteration. It also provided a good historical overview afterwards.
9.3 Collaborated coding

During development, tasks were split up between the developers. This means every developer produced their own piece of code which in some cases would result in a stand alone piece of code that would not be reused by the other developer. To ensure every aspect of the system was rechecked by every developer, code reviewing was used. This was done using GitHub’s pull request system. In this system, one is able to see every code change made in a certain branch. When checking a pull request, the developer is able to see everything the other developer has produced and is able to add comments if something is not clear.

9.4 Meetings

In the beginning of every iteration a meeting was planned with Bluerise. During these meetings the developers first showed the features implemented in the previous iteration. After that, there was a discussion about the planning for the next iteration. This provided both parties a clear view of what was done and what had to be done. These meetings also allowed for discussions about problems or decisions that came up during the previous iteration.
A significant part of the orientation phase was finding the best framework. Different aspects of multiple frameworks were examined to determine what was the right choice. In this section a reflection will be made on this choice. First the pros and cons will be discussed, followed by a short discussion on whether another framework would have been better.

10.1 Advantages

Overall the CakePHP framework performed well. The documentation was clear in most situations. A very useful feature was migrations, which allowed for versioning the database structure. Another good feature was the testing support CakePHP offers. This allowed the developers to test quickly without too much additional configuration.

One downside that came to light in the first assessment about CakePHP, the limited template support, actually turned out to be an advantage. Templates are written in plain PHP with some helper functions, this allowed templates to do exactly what they are supposed to do without any limitations provided by the template engine. Other than that, most of the code is structured and readable and CakePHP offers a lot of functionality for different occasions.

10.2 Disadvantages

Although most of the times we were pleased with the CakePHP framework, there also were some noticeable downsides. First of all, the file system CakePHP forces developers to use is not very clean. Although efforts are made to split library code from application code, the application folder still contains a lot of sub folders. These sub folders contain plugin code, testing code, source code and assets. When looking at this folder it is not immediately clear what code is developed in house and what code is merely used. We would have preferred a file system in which developed code was split from other components.

As mentioned in section 10.1 documentation was clear most of the time. However, in situations where documentation was not clear, looking at CakePHP source code did not clear up uncertainties either. The CakePHP source has very large functions which makes it unreadable.

A major downside of CakePHP is the ORM. ORM stands for object relational mapping. Most ORMs allow developers to use database objects as if they were
class objects in code (see Appendix A). CakePHP has a different concept behind ORM. In this system, every database object has a model object which in fact is a database library for that class. Finding items from the database using this model library results in an array with data. When the data is changed, the array needs to be changed and passed to the library object for saving. In our opinion, this is not what an ORM should do. This way of coding causes a lot of boilerplate code, is not intuitive and does not make for readable code.

The final downside of CakePHP is its speed. Our application does not use too much CakePHP functionality so one would assume that CakePHP would not cause too much overhead. However, the opposite appeared to be the case.

10.3 Remarks

While developing using the CakePHP framework we were pretty pleased with the chosen framework. Most of the time we could easily do what we wanted within the frameworks bounds. From the four inspected PHP frameworks, after developing in CakePHP, we would still make the same choice. One thing that did cause some slight irritations while developing was the PHP programming language. Therefore, a non PHP framework is preferred when programming language is not an issue.
11 Requirements

As mentioned in section 1.7, the requirements form a base to what the system must adhere to. Some of these requirements turned out to be unfeasible, while a lot of new requirements emerged. In this chapter the implementation of these requirements will be discussed, pointing out why some have been left out and why others have been added. First we will give an overview in section 11.1 of which requirements are implemented and which are not. This is followed by an explanation in section 11.2 regarding unimplemented requirements. Then in section 11.3 a short overview of newly added requirements is given. Finally in section 11.4 the efforts to fulfill the constraints and non functional requirements are discussed.

11.1 Fulfillment

- Must have
  - Home page
    - General information about the OTEC tool
    - Instructions for use of this application
    - Links to the tools
  - Onshore/Offshore practice tool
    - Displaying project parameters
    - Adjustable parameters
    - Adjustable energy/SWAC demands
    - Display evaluation in form of site/financial score
    - Instant project evaluation after adjusting parameters
  - Project assessment tool
    - Choosing a location using:
      - Location name
      - Location coordinates
      - Picker with range selection
    - Choosing project type (Onshore/Offshore)
    - View available OTEC locations in the area
    - Onshore/Offshore views of the project with filled in parameters
    - See detailed information gathered by the back end
    - Generate PDF file containing an overview of the project
  - User registration system
• Sign up
• Sign in/out
• View list of saved projects
• Open/edit previous projects

– Back end system

• Use Google Analytics with extended tracking

• Should have

– Onshore/Offshore practice tool

✓ Adjust assumptions
✓ Feedback on parameter ranges (visual indications)
✓ Display plant costs
✓ Display pipe costs

– Project assessment tool

✓ Choose locations for on and offshore installations
✓ Save personal project (with a given name)
✓ Clone other projects to user account
✓ Change initial project parameters
✓ Set project as public
✓ Get link to project for easy sharing

– User registration system

✓ Edit user settings
✓ Change project names

– Back end system

✓ Plot all projects on a map to determine areas of interest
✓ Plot user locations on a map
✓ Sent emails to users about assessments or other notifications

• Could have

– Onshore/Offshore practice tool

✓ Extra visual feedback when changing parameters (e.g. change length of pipe on screen)

– Project assessment tool

✓ See a list of users interested in your own projects

– User registration system

✓ Ability to sort projects on screen

✗ Community map tool

✓ Declare interest in public project
✓ Get into contact with creators or interested users of public projects
✓ Display map with public projects in an area

47
- Open public projects from the map
- Backed system
  - Displaying data gathered through data mining so that a quick
    assessment can be made about the users interests.
- Won’t have
  - Onshore/Offshore practice tool
    - Support for extra OTEC purposes, such as desalination and
      irrigation
    - Display emission savings
    - Prompt for current energy usage data
  - Project assessment tool
    - Use assumptions given by users for certain areas
  - User registration system
    - Collaborated projects

Constraints

✓ The product must work on modern browsers (desktop as well as mobile)
  ✓ Chrome (version 16+)
  ✓ Microsoft Internet Explorer (version 8+)
  ✓ Mozilla Firefox (versions 10+)
  ✓ Safari (version 5+)
  ✓ Android standard browser (Android 4 and higher)

✓ An overflow of users requesting heavy calculations should not cause the
  system to crash, the system must be able to handle at least 20 simulta-
  neous requests. (This is aimed at the project assessment tool)

✓ The interface should be fully viewable on resolutions of at least 1024 *
  768

✓ The interface must be operable on touch screens

Non-functional requirements

✓ Consistent interface
✓ The interface should be based on the Many Hands report
✓ Security; the system is secure and is prepared for standard web applica-
  tion digital attack attempts
✓ Testability; the system has been tested extensively and this the results
  provide insight in product quality
✓ Usability; the system is clear and provides enough feedback for untrained
  users to retrieve information they need
✓ Maintainability, the system is structured and modular, providing good
  means for further maintaining and expanding the code
11.2 Unfulfilled

In this section a motivation behind the absence of several requirements will be given. Most of these requirements turned out to be less urgent than was initially planned. Some of these requirements could be added in the future while others just do not fit in the resulting system.

Insight in user activity

One functionality that did not get the attention that was planned is a comprehensive back end for inspecting what users are doing. Since there were more basic user functionalities than was initially planned, this part of the application was put on a lower priority. In the future this could become more interesting, but first the application should have a fully functional user flow. In the meanwhile user activity can be monitored using Google Analytics, which is configured to record detailed user activity. Under this back end section also falls data mining on collected data.

Community map tool

A feature that was highlighted in the ManyHands report was the community map tool. This section of the tool would allow users to collectively work on projects. During meetings with Bluerise we quickly discovered that this aspect of the application would not be desired by users. It was more important to give a good assessment than to include overly social aspects.

11.3 Unforeseen

In this section a couple of aspects of the application which were not originally planned are discussed. These requirements came up during scrum meetings. Some of these requirements concerned modifications to older requirements, while completely new requirements also emerged.

Queuing system

Soon after implementing the base of the application it became clear that it would not be feasible for multiple users to do an assessment at the same time. Since the calculations required substantial computer power, the stability of the server could be in danger when users were allowed to simultaneously assess multiple areas. For this reason a queuing system has been developed. This system handles computer intensive tasks in manageable amounts by putting tasks in a queue. A description of the implementation can be found in chapter 7.

Login using alternative sites

Today there are numerous web applications that require a login. For some people creating a new account for every application is too much of a hassle. To not scare off users, the login system has been modified to be able to accept accounts for well know services; Yahoo, Google, Facebook and Microsoft Live. A description of the implementation can be found in chapter 5.
Templates

Part of the application design was covered in the Many Hands report. After rethinking about what should really be important within the application, it turned out that there were numerous pages that were not designed yet. This forced the developers to create new templates that were intuitive and self-explanatory.

User roles

In the progress of developing the application it became clear that user control was important. Different user roles should be possible where each of these roles would have certain rights within the system. Although not every role is currently present yet, the base has been formed for extended user control. The role system is capable of limiting the actions of certain accounts which enables Bluerise to control what parts of the applications are available for each role. A description of the implementation can be found in chapter 7.

Site/scenario creation and snapping

The initial idea for the interface regarding site creation has evolved over the course of the project. This resulted in a managing work flow that is also carried out in scenarios. An additional functionality that has been implemented within the site creation page is snapping of coordinates. This functionality causes created sites to instantly snap to data points.

11.4 Non-functional fulfillment

In section 1.7 a list of constraints and functionalities has been given. In this section the efforts for adhering to these constraints and functionalities are discussed. First, an explanation will be given regarding the compatibility of the tool on different platforms. After that, the ways in which the development team has tried to produce a product on which Bluerise can build upon in the future will be elaborated. Finally, some common security leaks and our attempts to secure these leaks are discussed.

Compatibility

For compatibility it is important to adhere to web standards. One important factor for reaching overall browser compatibility is having clean HTML documents. These documents must have the right structure and contain no syntax errors. When documents are correct it will be less likely that different browsers display the pages differently. For this reason, all pages are validated in the official W3 validator[1] before they are submitted to the release version. Manual testing on different browsers also took place to find and solve browser specific flaws.

[1]validator.w3.org: W3C HTML Validator
Maintainability

The potential for this system is great and can have a positive impact on Bluerise’s market share. This means it will be unlikely the developers of this project are the only developers that will work with the source code. During the project it was very important that code was structured, readable and maintainable. Documentation has been added to every file so that future developers can easily continue working on the application. Code quality has been kept high by frequent testing using various tools.

Safety

Common flaws in web applications are an easy target for malicious users. By using a web framework, some of these attacks can be prevented without any additional coding. For example, SQL injection prevention is handled by CakePHP. Other flaws like cross site scripting are handled by using standard PHP prevention functions on user generated data. Error reporting has been turned off on the server. Flaws that concern unintended file access are handled by the CakePHP routing system.
12 Conclusion

During this project, the development team has used various software development techniques in order to create a solid base for an online OTEC assessment tool. In the first phase, an orientation has been done to fully capture the requirements of the application. After the problem was defined, research was done on what development framework to use for implementation. The chosen framework was CakePHP due to good documentation and decent quality. During development, the Scrum development approach was used for an agile development process.

The framework selection turned out well and most of the initially planned features have been implemented. The result is a working, usable and maintainable web application for OTEC assessments. Now that there is a steady base, further work can be done to extend functionality. A few aspects that can be improved or extended will be listed:

**Administrator interface**
Administrators have a lot of rights in the system, but do not always have access to these rights. For example, an administrator should be able to add and edit users through a decent interface.

**Extra visualization**
To improve insight in area and site viability, more visualization methods can be added. Ideally, the maps should be dynamic and interactive. Higher user ranks should be able to view more information than lower ranked users.

**Project sharing**
To add a more social aspect to the tool, the feature of sharing projects could be added. Allowing an unregistered user to view an assessment makes the tool more accessible. An added bonus for this is that the assessments will also be available via search engines.

**Data mining**
With all the data available on user entered assumptions and assessed areas, Bluerise could improve their own services. The available data can also be used to improve recommendations for new areas and sites.

Overall, we would deem it a successful project with a bright future ahead. Bluerise will put the tool into service and will continue the development of the tool.
Part IV

Appendix
A Framework 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 section 1.7.

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 patterns 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 provide access to other 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, MVC 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 an 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.
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. Another important feature of a framework is the quality of code. A quality review is given for every PHP framework using a PHP quality assessment tool.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Peak database access responses per second, dedicated i7 hardware, single query</th>
</tr>
</thead>
<tbody>
<tr>
<td>codeigniter</td>
<td>3,691</td>
</tr>
<tr>
<td>django</td>
<td>3,349</td>
</tr>
<tr>
<td>rails</td>
<td>2,991</td>
</tr>
<tr>
<td>laravel</td>
<td>1,665</td>
</tr>
<tr>
<td>cake</td>
<td>1,150</td>
</tr>
<tr>
<td>symfony2</td>
<td>707</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Framework</th>
<th>Peak JSON responses per second, dedicated i7 hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>django</td>
<td>9,727</td>
</tr>
<tr>
<td>rails</td>
<td>4,627</td>
</tr>
<tr>
<td>codeigniter</td>
<td>4,225</td>
</tr>
<tr>
<td>laravel</td>
<td>2,166</td>
</tr>
<tr>
<td>cake</td>
<td>1,554</td>
</tr>
<tr>
<td>symfony2</td>
<td>1,077</td>
</tr>
</tbody>
</table>

Figure B.1: TechEmpower Framework comparison.

**Web frameworks benchmark**

The benchmark of frameworks focuses on two different tasks, querying a database and serializing JSON. The results shown in Figure B.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 assessment tools use their own measurement methods. 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.

**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.

\(^1\)pdepend.org: PHP Depend is a tool for generating quality assessment charts on PHP software.
CakePHP

The Overview Pyramid [B.2] 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.3] shows that CakePHP combines unstable code with a low level of abstraction, which also indicates that the maintainability is not very high.

Figure B.2: CakePHP Overview Pyramid, generated by PHP Depend

Figure B.3: CakePHP Complexity Chart, generated by PHP Depend
CodeIgniter

In the Overview Pyramid B.4, CodeIgniter scores reasonable, although methods are somewhat complex and classes are too big. According to the Abstraction Instability chart B.5, 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.4: CodeIgniter Overview Pyramid, generated by PHP Depend

Figure B.5: CodeIgniter Complexity Chart, generated by PHP Depend
Laravel

According to the Pyramid Overview B.6, 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.7 shows that packages are somewhat unstable, while little abstractions are used. Maintaining Laravel should not cause too many problems.

Figure B.6: Laravel Overview Pyramid, generated by PHP Depend

Figure B.7: Laravel Complexity Chart, generated by PHP Depend
Symfony

The Pyramid Overview [B.8] for Symfony indicates that the quality is comparable to Laravel’s. However, the Complexity Chart [B.9] 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.8: Symfony Overview Pyramid, generated by PHP Depend

![Symfony Overview Pyramid](image)

Figure B.9: Symfony Complexity Chart, generated by PHP Depend

![Symfony Complexity Chart](image)
### C Framework implementations

#### Ruby on Rails

**Rails Model**

```ruby
class Message < ActiveRecord::Base
  attr_accessible :message
  validates :message, :presence => true, :allow_blank => false
end
```

**Rails Controller**

```ruby
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**

```html
<!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**

```html
<% 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>
```

---

61
Django Model

```python
from django.db import models

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

Django Controller

```python
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

```html
{% 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>
</html>
```
**CodeIgniter**

```php
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**

```php
class Messages extends CI_Controller {
    public function index()
    {
        $this->load->helper('html');
        $this->load->helper('form');
        $this->load->library('form_validation');
        $this->load->model("message_model");
        if (!($this->form_validation->run() === FALSE))
        {
            $this->message_model->set_message();
        }
        $data["messages"] = $this->message_model->get_messages();
        $this->load->view('base', $data);
    }
}
```

**CodeIgniter Template**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="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>
```
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;
}
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));
    }
}

<!-- 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>

<!-- app/Resources/views/Test/index.html.twig -->
{% 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 Model

```php
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

```php
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

```html
<!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')
    @endsection
</head>
<body>
    <div id='main'>
        <div class='border' id='header'>Laravel test</div>
        @yield('main')
    </div>
</body>
</html>
```

Laravel Template View

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

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'));
    }
}

<!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>

<?php
$messages = $this->Message->find('all');
foreach($messages as $message):
    <div class='message'>
        <?php echo h($message['Message']['message']); ?>
    </div>
endforeach;
?>
D Implementation iterations

As described in section 2.4, product development is split into six Scrum sprints. Before each sprint, the team decides what issues to resolve in this iteration. These issues depend on input from the supervisors, the project plan and possible new problems the team has run into.

In this part of the report, a summary is given for every sprint. The summary includes a list of resolved issues, design decisions and comments on the iteration.
D.1 Iteration 1 / 06 May - 13 May

In this iteration a few basic tasks have to be executed that enable further project specific programming. First of all, the environment has to be set up so that the application can be tested. After the environment is ready, the framework has to be configured to work on the environment. When the environment is completely ready for application specific programming tasks, the first thing that needs to be done is creating the layout in HTML. Besides templates, some basic functionality will be implemented.

<table>
<thead>
<tr>
<th>Task</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic git pull on the server</td>
<td>Niels</td>
</tr>
<tr>
<td>HTML templates</td>
<td>Camiel</td>
</tr>
<tr>
<td>Onshore/offshore page</td>
<td>Niels</td>
</tr>
<tr>
<td>Area selection</td>
<td>Camiel</td>
</tr>
</tbody>
</table>

LessCSS

In order to keep the CSS files structured and to minimize redundancy, the team decided to use an additional library called LessCSS\[1\] This library provides an extended CSS abstraction layer in which more possibilities for programming exist. By using this library the CSS files can contain a minimum of redundant information, which makes it easier to alter things like font style and layout color schemes.

Client side calculations

In the prototype many calculations were performed in shell scripts. To keep calculations within the framework code, the decision has been made to move these calculations to the client. Besides a more central code architecture, this decision makes it possible to greatly increase performance on the playground page. Because no server interaction is needed to recalculate a project’s score the response time of the practice page is considerably better. Another option in this decision is to move the calculations to the PHP back end, this way the calculations are kept away from the user. Either way we advice that calculations that don’t require any interaction with a tool like GMT are done within PHP or JavaScript, not by shell commands.

HTML quality

The quality of the HTML code has impact on a number of different things. When HTML is not correct, search engines will index the site less frequently or may not index properly. Other then that, incorrect HTML will cause the layout to differ more greatly on different platforms, which will most likely cause flaws in the interface that reduce usability. It is clear that high quality is very important and to ensure this quality, different tools can be used. The World Wide Web Consortium (W3C) offers a HTML5 tool to validate HTML pages\[2\]

To ensure quality all HTML templates will be tested to pass this validation.

\[1\]LessCSS.org: LESS extends CSS with dynamic behavior such as variables and mix-ins
\[2\]validator.w3.org: W3C HTML Validator
JavaScript quality

JavaScript is a powerful language, but the highly dynamic language also imposes many risks for low quality software. Because some browsers interpret the same JavaScript code differently, it is important to use JavaScript in the correct way. JSLint is a tool that analyzes JavaScript code to look for common coding errors that won’t necessarily cause runtime errors directly. By using this tool on the JavaScript part of the application, potential bug causing code can be detected without testing for functionality.\(^3\)

Progress

This week, 12 issues were resolved in 31 commits.

Figure D.1: Daily commits for first iteration

\(^3\)JSLint.com: The JavaScript code quality tool
D.2 Iteration 2 / 13 May - 20 May

After the first iteration the basic layout from the ManyHands report has been implemented with a few basic functionalities. Now that the environment has been set up, the programming on more advanced features can begin.

The major components in this iteration are the user interaction and the permanent storage. Because the layout of the ManyHands report is not fully qualified for the user requirements some templates will have to be reproduced. Besides reproducing certain templates, the overall flow of the application must be re-analyzed to better fit the requirements.

<table>
<thead>
<tr>
<th>Task</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database structure</td>
<td>Niels/Camiel</td>
</tr>
<tr>
<td>Work flow</td>
<td>Niels/Camiel</td>
</tr>
<tr>
<td>Processing page template</td>
<td>Niels/Camiel</td>
</tr>
<tr>
<td>User database/model</td>
<td>Niels</td>
</tr>
<tr>
<td>User functionalities (Register/Sign in/Sign out)</td>
<td>Niels</td>
</tr>
<tr>
<td>User related templates (Register/Forgot password)</td>
<td>Niels</td>
</tr>
<tr>
<td>Sign in/Sign out via AJAX</td>
<td>Niels</td>
</tr>
<tr>
<td>Install phpMyAdmin on the server</td>
<td>Niels</td>
</tr>
<tr>
<td>Sign in toolbar dropdown template</td>
<td>Camiel</td>
</tr>
<tr>
<td>My Projects template</td>
<td>Camiel</td>
</tr>
<tr>
<td>Project template</td>
<td>Camiel</td>
</tr>
<tr>
<td>Site template</td>
<td>Camiel</td>
</tr>
<tr>
<td>Case template</td>
<td>Camiel</td>
</tr>
<tr>
<td>Create initial database tables for models</td>
<td>Camiel</td>
</tr>
<tr>
<td>Setup PHP testing</td>
<td>Camiel</td>
</tr>
</tbody>
</table>

Database structure

Before the application work flow can be established, there must be a clear view of what elements are present in the system. After talking to the Bluerise staff it became clear that the major components in the systems were going to be: users, projects, sites and scenarios. For the last three of these components some data is needed that is calculated by the GMT back end. A short description of each of these elements will now be given:

Users

A user is a person who is logged in. A user can manage projects, sites and scenarios and can have access to additional functionalities not present for the unregistered user. The most important attribute for a user is its email, which will be used for notifying site events (such as calculation reports) or distribute other Bluerise related emails.

Projects

A project represents an area populated with several sites. Thus, the minimal representation of a project is a set of coordinates. A user can name a site and generate a PDF report of the entire project. The GMT back end calculates data for a project. Because the assessment takes a reasonable amount of time and
because the generated data takes up substantial space, reusing data between users is essential. Because of this, the choice has been made to separate project data and user project configuration. This means there will be two project objects in the system: projects and project storages. The project object holds data that can be linked to a user, this object has an additional pointer to a project storage object. The storage object holds information about the project area and the location of the generated data.

**Sites**

For sites, a similar setup has been created as for projects. In the current back end situation, site data is retrieved from a project. This is however something that is likely to change in the future. When changed, data is linked to sites instead of projects and thus the storage should also be unlinked from a particular user. A user can name a site and when the site parameters are changed, a new site storage object is created (or reused if it already existed).

**Scenarios**

The final component in this application are scenarios. The main goal of the OTEC tool is to evaluate sites. In such an evaluation, assumptions have to be made regarding component prices such as pipes. Because Bluerise does not have full insight in these prices in different areas of the world, a default value will be used initially. Certain users will have more correct information than Bluerise can offer, therefore the user can adjust these assumptions in a scenario. Another way in which these scenarios could be used is to play out different realizations in the future. A user can create a scenario in which the prices of oil are high to simulate the future in which fossil fuels will be depleted.

**Assumptions fall through model**

Because some assumptions are not necessarily linked to one scenario, the user will also have the ability to add assumptions that will be used for every future evaluation. When a user wants to evaluate a site, the following steps are executed. First, the site specific assumptions are checked, when an assumption is set by the user, this assumption is used. When no assumptions are found, the user’s global assumptions are checked. If after this still no assumptions are found, the global Bluerise assumptions are used.

**Work flow**

After the model structure discussed in the previous section had been established, a work flow that fits this model can be created. In this flow, it is important that interface elements are self explanatory and consistent. The basic work flow is: First the user creates a project by selecting an area. Then the user creates a site by choosing a location based on the visual data that is retrieved from the GMT back end. After the site location has been established, scenarios can be made regarding this site. In this work flow there is a tree structure. Projects can have any amount of sites, sites can have any amount of cases. For projects there will be an overview page where information regarding the project and a list of sites is shown. For sites, there will be a similar
overview but this time with a list of scenarios. The scenario’s can be inspected by using the playground tool as discussed in the previous part of this report. Finally, a registered user can view a list of all previously made projects.

User implementation

A core feature of the application is the registration and login system. A lot of other functionalities rely on the correct and secure working of it. Instead of using a plugin to extend the framework[4] we have decided to implement the functionality ourselves. A quick look around made it clear that there are no good user plugins available for CakePHP which offer fitting functionality and proper security.

This iteration, basic functionality like sign up, sign in, sign out and password reset were implemented. This entailed the design for proper templates for the web application and emails. To make sure users can be contacted by Bluerise, the users must sign up with a valid email address. After sign up, the account must be activated through email. A lost password can also be reset using email. For security and mailing, CakePHP was a great help.

PHP testing

As mentioned in the project plan report it is important to keep code quality high. For this reason, all PHP code should be thoroughly tested. To facilitate proper testing, a testing environment has been setup. This testing environment uses PHPUnit[5] and XDebug[6]. In this environment unit tests can be executed and line coverage can be inspected. The first tool gives the ability to test whether units behave as intended while the latter tool gives insight in what parts of the code are tested.

Progress

This week, 23 issues were resolved in 64 commits.

![Daily commits for second iteration](image)

Figure D.2: Daily commits for second iteration

[5]phpunit.de: PHPUnit testing tool

73
D.3 Iteration 3 / 20 May - 27 May

The second iteration revolved around the core of the application. Different templates were linked and created, and the database layout was set up. This iteration revolves around linking the database with the templates and creating additional templates and user interaction elements. The main template revolved around the interaction model concerning creating sites.

<table>
<thead>
<tr>
<th>Task</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link models to templates</td>
<td>Niels</td>
</tr>
<tr>
<td>Database model relations</td>
<td>Niels</td>
</tr>
<tr>
<td>Login using Google, Yahoo, Live, Facebook</td>
<td>Niels</td>
</tr>
<tr>
<td>Finish site view template</td>
<td>Camiel</td>
</tr>
<tr>
<td>Site selection template</td>
<td>Camiel</td>
</tr>
<tr>
<td>JavaScript testing</td>
<td>Camiel</td>
</tr>
</tbody>
</table>

Link models to templates

Before this iteration templates were created using stub data. This stub data did not conform to cakePHP’s ORM model, which means that templates were not yet ready for real database objects. Besides rewriting some of the templates to use the correct ORM model, the controllers were created that handle database interaction.

Database model relations

The different models present in the system interact with each other in certain ways. Some objects can “own” other objects, such as in the case of users and projects. Some objects belong to other objects, such as sites that belong to a project. This issue revolved around setting up these relations so that they can be used when retrieving data from the database.

Login using Google/Yahoo/Live/Facebook

An important aspect of an application is a personal login. Nowadays there is an overflow of application for which an account is needed and a lot of users will preferably not create an additional account. A good answer for this dilemma is the ability to login using existing channels. Therefore the functionality has been build to login using accounts for common existing sites.

Site selection template

One element from the Many Hands report concerning the user flow that was not fully up to date with the requirements was the site selection template. On this page users will have the ability to create new sites and manage existing sites. The requirements combined with some interface prototypes resulted in a managing page in which a user can see a list of all sites that belong to a project. The user can also see all sites on a Google Maps element. The user has the ability to add new sites and delete and alter existing sites.
JavaScript unit testing

While PHP code is also submitted to unit testing, this iteration JavaScript took an increasingly big role in the code base. JavaScript bugs could be regarded as even more harmful to user experience than server side code. For this reason, some time in this iteration was spent on setting up unit testing in JavaScript. JavaScript tests were conducted using QUnit. This tool provides a clean testing API with extensive debugging capabilities.

Progress

This week, 23 issues were resolved in 71 commits.

Figure D.3: Daily commits for third iteration

---

1. QUnitJS.com: QUnit JavaScript testing
The third iteration made a start at connecting all elements within the system. Since this connection was substantially big, one iteration was not enough to implement every database connection. The fourth iteration included the implementation of the back end assessment tool connection, the back end connection for managing projects and the final templates. Key features are reading project data and queuing assessment tasks.

This iteration included a big milestone, the connection with the real backbone scripts of the application. Setting up a system that is prepared for multiple users while doing CPU heavy calculations required some additional work. Other activities concerned extending the managing page with a connection to the database and reading generated project data.

<table>
<thead>
<tr>
<th>Task</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>back end connection for project managing</td>
<td>Camiel</td>
</tr>
<tr>
<td>Scenario templates</td>
<td>Camiel</td>
</tr>
<tr>
<td>Breadcrumbs and savable headers</td>
<td>Camiel</td>
</tr>
<tr>
<td>Assessment processing controller</td>
<td>Niels</td>
</tr>
<tr>
<td>Queue for back end assessment</td>
<td>Niels</td>
</tr>
</tbody>
</table>

**back end connection for project managing**

The previous iteration yielded templates for project managing. In this iteration the connection with the database via AJAX has been set up. Another function that has been added is the snapping of created sites to data points. These data points consist of depth/temperature data and coastal points. The snapping ensures assessments are made on locations where actual gathered data is present. This allows the tool to give a solid trace of information, which increases the credibility.

**Scenario templates**

To keep the user flow consistent, a similar managing page for sites has been created. This managing page has the same structure as the project managing page. By using a similar approach to site management, the same back end code for saving could be used.

**Breadcrumbs and savable headers**

Another returning element in the interface was the ability to change a name. This functionality was paired with the fact that some elements have so called child elements; projects have sites and sites have scenarios. In each of these object pages it is desired to return to the parent object. This requirement resulted in a breadcrumb header which allows for navigating to parent objects. Besides navigation, this header allows the user to change the name of an object, saving it or deleting it.
Assessment processing controller

The assessment processing plays an important role in the OTEC tool. Assessing a project is done by scripts provided by Bluerise. The assessment controller handles the creation of assessments. The controller also handles the notification of the progress of the assessment, this is done via AJAX.

Queue for back end assessment

The scripts provided by Bluerise run GMT, which can be quite CPU and memory intensive. It is important that a user cannot overload the system. For this reason a queuing system has been set up. This system allows for tasks to be added to a queue, which will be processed by a single Cron job. A Cron job is a single process that is called at a fixed time interval. This means that within this system only one assessment will be handled at a time. While this could increase waiting time experienced by users, this system will not be able to overload by doing multiple calculations at a time.

Progress

This week, 8 (big) issues were resolved in 72 commits.

Figure D.4: Daily commits for fourth iteration
After the fourth iteration, the server was able to make assessments, but the results could not be visualized yet. The fifth iteration builds on this and connects the assessment results to the application. Because the first SIG code evaluation was nearing, there was also an emphasis on tests and code enhancements. After this iteration, the application could successfully perform assessments and display the correct results.

With this iteration, a lot of previously prepared work came together to end up with most features properly working. Assessments are queued and executed and the results are properly displayed in the application. A lot of work also happened on the back end. Code enhancements to increase maintainability and security took place and basic access management for users was also implemented. Finally error page templates have been designed and implemented.

<table>
<thead>
<tr>
<th>Task</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cron job to process queued tasks</td>
<td>Niels</td>
</tr>
<tr>
<td>Read assessment results</td>
<td>Niels</td>
</tr>
<tr>
<td>Ability for users to edit profile</td>
<td>Niels</td>
</tr>
<tr>
<td>Access management</td>
<td>Niels</td>
</tr>
<tr>
<td>Link assumptions for assessments</td>
<td>Niels</td>
</tr>
<tr>
<td>Code enhancements</td>
<td>Camiel</td>
</tr>
<tr>
<td>ProjectStorage storage fix</td>
<td>Camiel</td>
</tr>
<tr>
<td>Escape user inputs</td>
<td>Camiel</td>
</tr>
<tr>
<td>Testing</td>
<td>Camiel</td>
</tr>
<tr>
<td>Error page templates</td>
<td>Camiel</td>
</tr>
</tbody>
</table>

**Display correct assessment results**

For the correct assessment results to be displayed inside the application, a few issues had to be taken care of. First, the cron job to process the queue had to be installed on the server. Then, the results had to be read from the storage folder into PHP and served to the client. To make sure the scenario assessments also were correct, these evaluations had to be coupled with the assumptions stored in the database.

**Access management**

Not every visitor should have access to all features. That is why users can register and can be assigned different roles. The user’s role specifies to which extent access can be granted to certain features. Roles are stacked, which means that a registered user gets the same rights as an unregistered user, plus some extras, and a Bluerise account gets all the features from the roles beneath him, plus some extras.

**Code enhancements**

As the first SIG code evaluation is nearing, some code enhancements took place this iteration. There was a floating point accuracy bug fixed that caused coordinates stored in the database to be corrupt. Testing also brought some bugs to light.
Escape user input

There is a high security risk when dealing with user input. To make sure user input is displayed and stored correctly, the input is escaped. This prevents XSS attacks and makes sure HTML code cannot be inserted into the templates.

Progress

This week, 19 issues were resolved in 68 commits.

Figure D.5: Daily commits for fifth iteration
D.6 Iteration 6 / 10 Jun - 17 Jun

With the last iteration yielding in a properly working application, this iteration mainly around small tweaks and fixes. During this iteration, code was sent to SIG for the first code evaluation. The last iteration made a start to improve code quality and prepare for this evaluation, this iteration continues that work.

Next to a decent number of fixes and tweaks (such as size and number restrictions), this iteration introduces a new version of the assessment tool and adds site recommendations. Code was read through to make sure comments are up to date before sending it in for a quality evaluation from SIG. Google Analytics was connected to have a good overview of active users and usage.

<table>
<thead>
<tr>
<th>Task</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read site recommendations</td>
<td>Camiel</td>
</tr>
<tr>
<td>Table view for projects</td>
<td>Camiel</td>
</tr>
<tr>
<td>Area size restrictions</td>
<td>Camiel</td>
</tr>
<tr>
<td>Number of assessments restrictions</td>
<td>Camiel</td>
</tr>
<tr>
<td>User experience updates</td>
<td>Camiel/Niels</td>
</tr>
<tr>
<td>Update assessment scripts</td>
<td>Niels</td>
</tr>
<tr>
<td>Update code comments</td>
<td>Niels</td>
</tr>
<tr>
<td>Google Analytics</td>
<td>Niels</td>
</tr>
</tbody>
</table>

User experience updates

After the last iteration some minor user experience issues came afloat. This iteration automatically focuses input fields when available (login menu, registration page, assessment page, etc). Other improvements include: a table view is available for projects, input fields where only numbers are valid ignore other keys, and there are special messages for views when there is no data available.

Area size restrictions

As started in the previous iteration, the user role specifies to which degree a user has access to a certain functionality. The maximum size of an assessment area is restricted per user role in this iteration. Similarly, the number of saved assessments is restricted.

Assessment tool update

A newer version of the assessment tool for the back end was released and linked to the application. This new version of the tool required changes to the way the queue is managed, but adds functionality by suggesting good sites for the area. These recommendations are visualized after an assessment.
Google Analytics

To track the website visitors and the usage of the website, Google Analytics was added. The tool displays the number of users currently visiting and their locations. Extra functionality was added to track the pre-assessment and user login activity.

Progress

This week, 37 issues were resolved in 83 commits.

Figure D.6: Daily commits for last iteration

Figure D.7 displays the commits times for all the commits (not just this iteration). It gives a good representation of the working hours and effort put into this project.

Figure D.7: Hourly commits for whole development period
E Database relations

The relation schema in Figure E.1 visualizes the database tables and their relations. Design decisions are described in chapter 6.
Bibliography


