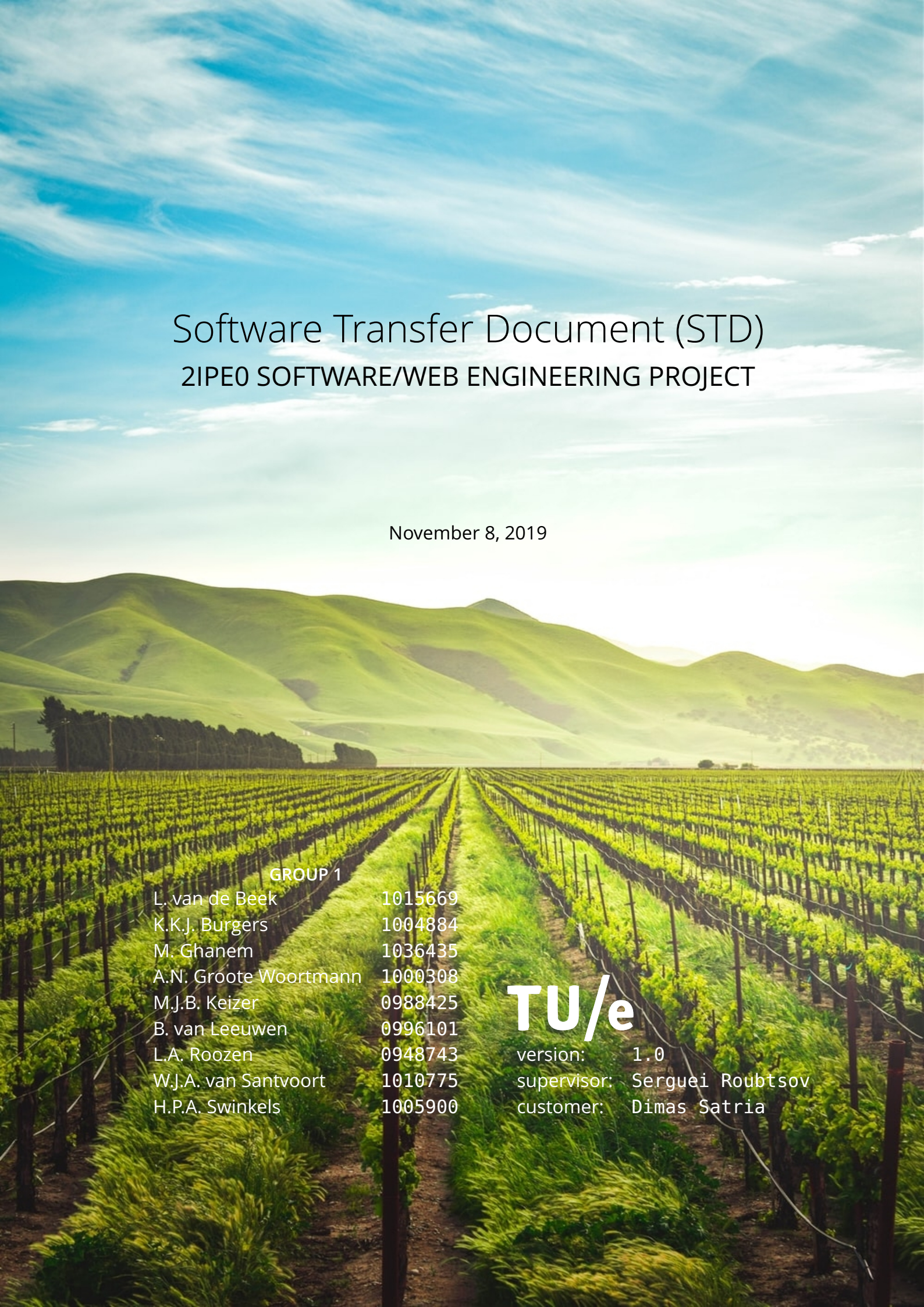

# Software Transfer Document (STD)

## 2IPE0 SOFTWARE/WEB ENGINEERING PROJECT

November 8, 2019


**GROUP 1**

| | |
|---|---|
| L. van de Beek | 1015669 |
| K.K.J. Burgers | 1004884 |
| M. Ghanem | 1036435 |
| A.N. Groote Woortmann | 1000308 |
| M.J.B. Keizer | 0988425 |
| B. van Leeuwen | 0996101 |
| L.A. Roozen | 0948743 |
| W.J.A. van Santvoort | 1010775 |
| H.P.A. Swinkels | 1005900 |

TU/e


version: 1.0
supervisor: Serguei Roubtsov
customer: Dimas Satria

**Abstract**

This document is the Software Transfer Document (STD) for CloudFarmer, developed by CloudFarmers as part of the Software Engineering Project in the Technical University of Eindhoven. This STD describes a detailed procedure to build and install CloudFarmer in order for the product to successfully transfer to the client. This document complies with the ESA software standards [1]

# Contents

# I Document Status Sheet

## I.I General

| | | |
|---|---|---|
| *Document title:* | Software Transfer Document (STD) | |
| *Document identifier:* | STD/1.0 | |
| *Authors:* | L. van de Beek | 1015669 |
| | K.K.J. Burgers | 1004884 |
| | M. Ghanem | 1036435 |
| | A.N. Groote Woortmann | 1000308 |
| | M.J.B. Keizer | 0988425 |
| | B. van Leeuwen | 0996101 |
| | L.A. Roozen | 0948743 |
| | W.J.A. van Santvoort | 1010775 |
| | H.P.A. Swinkels | 1005900 |
| *Document status:* | Version 1.0 | |

## I.II Document History

| Version | Date | Authors | Reason |
|---|---|---|---|
| 0.1 | 24-10-2019 | Léon | Initial version. |
| 0.2 | 29-10-2019 | Léon | Build procedure added |
| 0.3 | 1-11-2019 | Léon | Installation instruction added |
| 0.4 | 2-11-2019 | Léon | Added instruction to install datalinking service and the last 3 sections |
| 1.0 | 2-11-2019 | Léon | Implemented feedback from team-members |

# II Document Change Records

| Version | Date | Section | Reason |
|---|---|---|---|
| 0.1 | 24-10-2019 | All | Initial version. |

| | | | |
|---|---|---|---|
| 0.4 | 2-11-2019 | All | Draft version. |
| 1.0 | 2-11-2019 | All | Final version. |

# 1 Introduction

## 1.1 Purpose

This document contains detailed steps on how to run the CloudFarmer application. By following the steps in this document anyone with an intermediate knowledge of computers should be able to download, build and serve the application, given that he/she has access to the CloudFarmer repository.

## 1.2 Scope

CloudFarmers is a team of Computer Science Bachelor students working on a Software Engineering Project for the TU/e and Dimas Satria. Dimas Satria is a representative of *Praktijkcentrum Voor Precisie Landbouw*, a centre founded to speed up the adoption of precision agriculture (PA) in the Netherlands.

Precision agriculture is a farming management concept based on observing, measuring and responding to inter and intra-field variability in crops. The goal of precision agriculture research is to define a decision support system (DSS) for the whole farm management with the goal of optimising returns on inputs while preserving resources.

The goal of the CloudFarmers is to develop a web application that gives users of the application insights into the information of specific fields. These users might be farmers, researchers or even just curious people. This information will be accessible through an easy to use user interface that displays data, gathered from a number of different data sources, in an orderly manner.

## 1.3 List of definitions

### 1.3.1 Definitions

| Name | Definition |
|---|---|
| Active farm | An active farm is the farm to which the map, history, live, field, edit data and farm settings view are currently referencing. |
| Active field | An active field is the field to which the field view is currently referencing. |
| Active crop field | An active crop field is the crop field to which the field view is currently referencing. |

| | |
|---|---|
| Available farm | A farm of which the user is allowed to see the information. |
| CloudFarm | Database server for the application. |
| CloudFarmer | Application as defined by the project group and the customer. |
| CloudFarmers | The project group tasked to create the aforementioned application. |
| Crop | The virtual representation of a physical crop. |
| Crop field | The virtual representation of a physical crop field. A field can contain multiple crop fields. |
| Crop field information/ the information of a crop field | All the information about a specific crop field; location, crop type, crop year, crop season, area. |
| Customer | The person who assigned the task to create the application to the project group. In this case Dimas Satria. |
| Dacom | The company that provided us with their crop database and their API for extracting information from this database. |
| Data sources | List of all the sources of data that are known by the application. |
| Equipment | The sensors in a field. |
| Farm | The virtual representation of a physical farm. A user is allowed to have more than one virtual farm. |
| *Farmer* (Role) | A person who works on the physical farm. |
| *Farm admin* (Role) | A user with unrestricted access to all data associated with a specific farm. (E.g farm information/permissions, field information/permissions ... etc) |
| Farm information/ the information of a farm | All the information about a specific farm; ID number, name, address, postal code, country, email, phone number, website. |
| Farm permissions | A set of rules that determine which users are able to read, write or delete farm information. |
| Field | The virtual representation of a physical field. A farm can have multiple virtual fields. |

| | |
|---|---|
| Field information/ the information of a field | All the information about a specific field; location, field name, area, size in hectares, soil type. |
| Field permissions | A set of rules that determine which users are able to read, write or delete field information. |
| *General user* (Role) | A user that is a member of a farm without permissions. |
| Member | A user is a member of a farm, if they have a role assigned to them on that farm. |
| Observation | Metadata. |
| Observation Data | A general class of data referring to Dacom's data and/or WolkyTolky data and/or data input by the user. |
| Physical crop field | A physical crop field is a physical field that contains at least one type of crop. |
| Physical farm | An area of land that is devoted to agricultural processes, it is the basic facility for food. |
| Physical field | A physical field is a patch of dirt, clay, or some type of rock that is owned by a farmer, on which they are able to grow crops. |
| Props | A collection of data variables. Most of the time use abbreviate which variables are sent to a function. |
| *Researcher* (Role) | A person who carries out academic or scientific research. |
| Selected farm | A farm that the user has selected. |
| Slug | A set of attributes of an equipment model : brand name, model, model year, series, version. |
| User | A person that interacts with the application. |
| User role | A role that can be assigned to a user within a farm which defines all the field permissions and farm permissions of the user. These different roles are defined in the definitions and have the tag (role) at the end of them. |
| Views | The map view, history view, live view, fields view, settings view and personal settings view. |
| Widgets | A component of an interface that enables a user to perform a function or access a service. |

### 1.3.2 Abbreviations and Acronyms

| Acronym | Meaning |
|---|---|
| CSV | Comma-Separated Values (file format) |
| DIPPA | Data Integration Platform for Precision Agriculture |
| ESA | European Space Agency |
| DOM | Document Object Model |
| ID | Identity |
| JSON | JavaScript Object Notation |
| JWT | JSON Web Tokens |
| OAuth | Open authorization |
| PDEng | Professional Doctorate Engineering |
| PNG | Portable Network Graphics (file format) |
| SEP | Software Engineering Project |
| TU/e | Eindhoven University of Technology |
| URD | User Requirements Document |
| XML | Extensible Markup Language |

## 1.4 List of references


[1] *E.B. for software Standardisation and Control*. ESA software standards, 1991.
[2] *Acceptance Test Plan*. Technische Universiteit Eindhoven, 2019.
[3] *Software Design Document*. Technische Universiteit Eindhoven, 2019.
[4] *Software User Manual*. Technische Universiteit Eindhoven, 2019.
[5] *Unit Test Plan*. Technische Universiteit Eindhoven, 2019.
[6] *User Requirements Document*. Technische Universiteit Eindhoven, 2019.

# 2 Build procedure

This chapter describes the steps needed in order to build the front-end of the application. This means that all Typescript, Javascript and CSS files used in the project will be compiled into a few Javascript files that are hardly readable, but optimised for efficiency and disk size.

To build the front-end, one needs to have the following software installed:

- Node 12+ To download and for more information visit: `https://nodejs.org/`
- Git To download and for more information visit: `https://git-scm.com/`

After the software has been installed, open a command prompt and create a folder to place the project. Navigate to this folder and execute the following steps:

1. Clone the GitLab repository by executing:

   ```
   git clone HTTPS://$REPOSITORY-URL$
   ```

2. After cloning the repository move into the CloudFarmer folder and install all the needed Node packages by executing:

   ```
   npm install
   ```

3. To create the static build files run the following command:

   ```
   npm run build
   ```

4. Afterwards, a folder called 'build' is created inside the directory of CloudFarmer. It is possible to host these static files anywhere, an example of this is given in section 3.1.2 using the NPM serve package.

# 3 Installation procedure

This section describes how to install the datalinking service and the front-end of CloudFarmer. By following these steps someone with intermediate knowledge of computers in general should be able to run both. For the front-end a procedure to run a development and a production variant are given.

## 3.1 Front-end

To run the front-end, the minimum requirements of the Development machine described in the section 'Feasibility and Resource Estimates' of the SDD [3] should be met.

### 3.1.1 Development

In a command prompt inside the CloudFarmer directory, execute the following commands in order to run a development server of the front-end without the need to build the files first.

1. Make sure all dependencies are installed by running the following command:

   ```
   npm install
   ```

2. Run the development server by executing:

   ```
   npm start
   ```

   This will automatically open the default browser and navigate to the localhost address that the development server is listening on. When changes are made to the code and saved, the server will automatically recompile and reload the browser.

### 3.1.2 Production

As described in section 2 Build Procedure it is possible to build the files into a production optimized build. In this section a way to easily serve these files on the local machine with an NPM package named serve are described. To do this execute the following steps:

1. Execute the steps named in section 2. There should now be a folder named 'Build' with the static files.
2. Install the NPM package 'serve' by executing:

   ```
   npm install -g serve
   ```

3. After installation is completed simply run the following command to serve the files from the build folder:

```
serve -s build
```

4. The console will now show on which port the static files are being served. Go to that address in the browser and the site will show up.

## 3.2 Data linking service

Installing the datalinking service is done in order to enable the use of WolkyTolky equipment in the front-end. It can be easily expanded to be able to use other kinds of equipment. To run the datalinking service the minimum requirements of a *Development machine* are needed, which can be found in the section 'Feasibility and Resource Estimates' of the SDD [3]. Follow these steps to install the data linking service locally:

- Download and install PostgreSQL using the instructions of the desired operating system from: https://www.postgresql.org/download/
- In the initial set up, keep the local port at the default 5432 and use the password: admin
  When using a different user, password or database name make sure to change the connection details in the file microservice.ts
- Either by accessing psql in the terminal on Linux or opening the pgadmin 4 application on Windows, enter the following query to create the needed tables in the default database postgres:

```
CREATE TABLE EquipmentInformation (
    id serial PRIMARY KEY,
    farm_id integer NOT NULL,
    field_id integer NOT NULL,
    cropfield_id integer,
    equipment_id integer NOT NULL,
    savedata boolean NOT NULL
);
CREATE TABLE dataLastSaved (
    id serial PRIMARY KEY,
    equipment_id integer NOT NULL,
    last_ran varchar(25) NOT NULL
);
CREATE TABLE WolkyTolkyEquipment (
    id serial PRIMARY KEY,
    equipment_id integer NOT NULL,
    station_id integer NOT NULL,
    api_key varchar(200) NOT NULL
);
```

- After the database has been succesfully set up with the correct tables install all dependencies of the datalinking service by running the following command inside of a command prompt inside microservice/datalinkingservice:

```
npm install
```

- Now start the datalinking service by going into the folder /microservice/datalinkingservice/ and running the following command:

```
npm run dev
```

  The command prompt will now show if the server succesfully started and will display any queries being requested by the datalinkingservice.

- Finally, in order for the front-end to use the local instance of the datalinking service instead of the one the SEP group hosted on TU/e change the dataLinkUrl inside of the file cloudfarmer.ts from:

```
export const dataLinkUrl = 'https://cloudfarm.win.tue.nl/api/v1/data';
```

  To:

```
export const dataLinkUrl = 'http://localhost:3001/api/v1/data';
```

  After this change save the file and run the build procedure of the front-end. When running the development server of the front-end, saving will trigger a restart and the browser will reload automatically.

# 4 Configuration item list

This chapter describes all items that CloudFarmers will deliver. All documents will be handed in in PDF format. Addionally, the source code of the project has been made availible to the client via a shared repository located on the TU/e gitlab servers.

## 4.1 Documentation

- User Requirements Document [6]
- Software Design Document [3]
- Software Transfer Document (this document)
- Software User Manual [4]

## 4.2 Test Plans

The following test plans will be delivered:

- Acceptance Test Plan [2]
- Unit Test Plan [5]

## 4.3 Software

The delivered software consists of the following parts:

- CloudFarmer Front-end and data controllers
- Datalinking service
- Used PostgreSQL Database scheme

# 5 Acceptance test report summary

The first and only acceptance test was performed on 23-10-2019 from 14:00 until 17:00. The client, two team members and a manager were present. Before the test procedure commenced, the client signed the Acceptance Test Plan for conforming with the requirements formed in the User Requirements Document. All tests passed according to their criteria. However, some small artifacts were found which were fixed on the spot, these are documented in section 7 Software Change Requests. All test results were positive, therefore the client deemed the acceptance test successful and signed the document again.

# 6 Software Problem Reports

No software problems were reported during the transfer period. Thus this chapter has been left empty.

# 7 Software Change Requests

The client encountered the following three issues during the acceptance test:

- In the personal settings view, pressing the ’enter’ button would trigger the page to reload without saving the changes in the data.
- When adding an account to a farm in the farm settings view, this account would by default get the role of a general user.
- Some names used for navigation items, which were agreed upon, and changed during the test.

The issues were resolved immediately and the client agreed to continue with the acceptance test afterwards.

# 8 Software Modification Reports

In agreement with the customer the team added unit tests and code comments after the first acceptance test of 23-10-2019. No functionalities were changed since this would violate the outcome of the acceptance test.