UNIVERSITY OF TURKU

Logging software upgrade

Master's Degree Programme in Information and Communication Technology

Author:
Ari Nevalainen

Supervisors:
D.Sc. Antti Hakkala
Professor Jouni Isoaho
Tuomas Ruohola (Vaisala Oyj)

July 2023

Master’s Degree Programme in Information and Communication Technology
Department of Computing, Faculty of Technology
University of Turku

Subject: Information Technology
Programme: Master’s Degree Programme in Information and Communication Technology
Author: Ari Nevalainen
Title: Logging software upgrade
Number of pages: 61 pages
Date: July 2023

Vaisala weather instruments testing requires data logging to produce information for decisions making. Data logging is increasingly becoming database focused. Database logging was not usable and available to all users easily, for which reason this thesis aimed to bring database logging into internally used logging software. VB.NET based software was to be updated bringing database logging available to all users.

The thesis and development of the software update used empirical research. Software principles were applied from theoretical literacy sources. Not all theoretical principles were applied to the internal software.

Most software acceptance requirements were met, and software was evaluated acceptable to use. Especially its productivity was remarkable. However, its development should continue to bring all evaluation subjects to fully satisfying level. Some erroneous issues can escalate in wider use.

The software was successfully updated, and it began a new era in its development lifecycle. Software was uploaded into GIT and should continue to be improved to better serve the company and its needs.

Keywords: Vaisala, weather instruments, testing, logging, InfluxDB, Grafana.

Table of contents

1 Introduction

1.1 Motivation

Vaisala was founded in 1936 by Vilho Väisälä. First sold products were radiosonde pieces. Since then, range of products and the size of company have expanded enormously. Today Vaisala has offices and employees in six continents of the world ("Observations for", 2022). Business areas are split into industrial measurements and weather and environment.

Vaisala's R&D and innovation center in Vantaa, Finland serves the development and testing of new and existing products. About 300 professionals work in this building meaning most of Vaisala's R&D personnel work there ("New R&D", 2022). Because of the nature of R&D work, the building and surroundings house many sensors from both business areas. Especially weather and environment instruments call for a dedicated outdoors area. This need is fulfilled by having a dedicated test field in Vantaa, wherein instruments are installed and monitored.

The need for this thesis rises from test field and its needs for effective results collection and their visualization. Test results have often been collected using plain text files, which main strength is their simplicity. However, when data amount rises and many sources of data need to be connected and visualized together, better solutions are needed to be more productive.

Database results collection has been in use at Vaisala test field for years, but its usage is complicated and for that reason dedicated into hands of just few. When other people want to add a new instrument into logging system, they need help from those people. If they are unavailable, instrument cannot be added to the database.

This is not a desired state in the realm of ever-increasing amount of data and its sources. Database logging should be available to all personnel who use the logging software and more easily than before. Instead of using multiple software to connect to the database like now, one software should be enough.

1.2 Research questions

This thesis answers following research questions:

Can existing data logger software be updated?
How to make the software meet the objective?

1.3 Objective

Main objective of this thesis is to design and create an update to an existing instrument data logging software called *Boson* enabling it to save data directly into the database. The resulting software should be:

1. Usable,
2. Available,
3. Simple or familiar,
4. Maintainable,
5. Reliable and
6. Profitable.

First and foremost, the software should be usable to log instrument data from sensors into database. This is the main development objective. Yet, this feature will not benefit anyone if it is unavailable.

Software is considered available when it is distributed in such way, that it can be found and downloaded internally in the company. Also, it should be usable in most used computers and their operating systems. They are commonly laptops running Windows 10.

Simple to use or familiarity take place when new features are attached so that if appropriate, they do not limit or change old functionality but add to it in a simple and/or familiar way. Familiarity manifests in that existing Boson users adapt it easily.

Maintainability is achieved by writing documented and clear program code and saving it into a version control system. Additionally, using modern and supported programming code is preferred when possible.

Reliability is measured by data availability and its integrity, so that it is not lost nor corrupted, but the same data that leaves the instrument, is collected by the database. This needs to be verified by testing.

Profitability means that software should produce more than it uses. It is most easily measured in saved time when compared to the alternative. Monetary benefit can be calculated from the saved time.

Four of the objectives deal with user experience and two its development and basis for it. However, without users, there is no reasoning for development and upkeep. This means user experience must be on such a level that the new software will be used.

The represented six criteria fulfilling software will save personnel's time not only when setting up and monitoring instrument data but also, in data processing and its transformation into graphical information. The main objective fulfilment of creating the new software and the six criteria evaluating it will be presented in this thesis.

1.4 Outline

The rest of the thesis is organized as follows:

Background chapter describes the theory of key terms and concepts. Weather instruments, their testing, logging, data visualization and software engineering are discussed. Current software with its shortcomings, users and facts are considered, as they direct the new software design.

Implementation chapter presents the requirements and the design of the new software. Where will the software be used and of what kind of modules it consists of. It ends in the tests, bugs and use of new software.

Evaluation chapter describes how software was tested especially against the criteria and whether is meets them or not. It shows the strengths and the shortcomings of the new software and in final subchapter sums the results in a score table.

Finally, in Conclusions chapter the findings are discussed. The found future development subjects are also considered. They did not see accomplishment in this development project. They were categorized into key and wanted features.

2 Background

2.1 Theory

2.1.1 Weather instruments

Weather instruments, also known as meteorological instruments, are hundreds of years old term. Santorio created a thermometer in year 1612, whereas Galileo had begun preparing it already in 1600 (Burt, 2012, p.8). Even multi-element automatic weather station (AWS) comes from that same century, being developed in England in 1678/1679. Other related remarkable events are the development of electronic automatic weather station in 1964 and first datalogger in 1975. (Burt, p.11). Vaisala was founded 39 years earlier in 1936 ("Observations for", 2022). Since then, Vaisala has produced measurement instruments from very low-level components up to weather stations and radars. The instruments are used around the planet earth and some even beyond (The Red Planet, 2021). One weather station application involving Vaisala is set up at Mount Everest (Wilkinson, 2019). Vaisala HMP155 humidity and temperature probe and PTB210 barometer were installed in 7945 m altitude above sea level and the data was sent near-real time using a satellite link (Matthews et al., 2022).

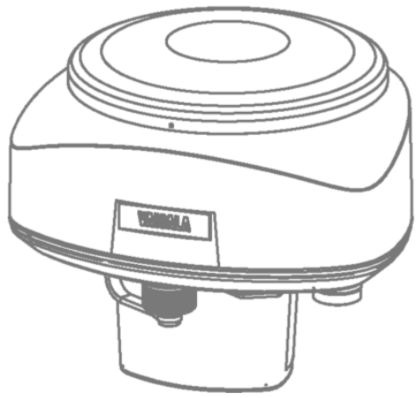


Figure 1. Vaisala RAINCAP on WXT531 ("WXT530 Series User Guide", 2023)

Vaisala weather instruments that measure air pressure, temperature, humidity, wind, and rain have a sensing element, e.g., a rain cap collecting rain drops (Figure 1). The movement or change in environment is changed into electrical signal that is digitized and processed by an algorithm. After processing, the sensor produces the output, that is commonly digital. Hardware interface is usually RS-232 or RS-485 standard serial port and the message forms it transmits are configurable. Configuration may allow user to pick certain parameters to be sent and skip the others or combine multiple into one message. Sensors can send multiple digital readings per second, minutes or less often depending on the setting. Moreover, if analog output option is used, sensor output is continuous, and the collecting system defines the used data acquisition interval. This process in many instruments is often similar as described by Vaisala RAINCAP® technology description and pictured below in Figure 2 ("Vaisala RAINCAP", 2023).

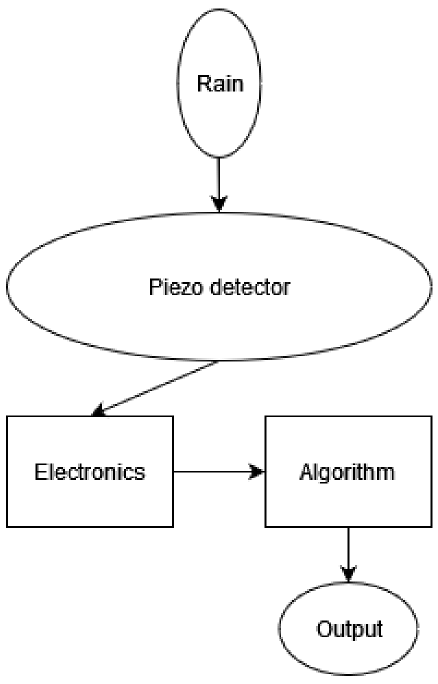


Figure 2. From rain event to output information

2.1.2 Instruments under test

Weather instruments experience same outdoor elements as the nature does. Whether it is extreme temperatures, lowest and highest altitudes, uncommon air pressure changes or record wind speeds, no environmental scene is the same as last time and there is always more than one variable in play. Outdoor environment changes every day of the year everywhere, yet weather instruments should not. For collecting reliable measurements systematically, instrument should remain the same as when it was produced. Yet this is more of a vision than reality. The reality is that instrument's mechanical parts decay, electronic parts can malfunction, lightning can strike unexpectedly, and many other scenarios can take place without warning.

Test field provides valuable real-world environment for instruments testing. It has space for hundreds of instruments for testing. Some tests require constant output monitoring whereas others use just power wiring, and some are not wired at all. It depends on what is being measured and examined. Example picture of a common looking test field in Netherlands holding Vaisala sensors is provided by Haij (Haij, 2010, p. 6). Similar kind of design and layout of weather instrument test fields around the globe is guided by World Meteorological Organization (WMO). Their publication called *Guide to instruments and methods of observation* gives abundant instruction on installation and use of instruments. Depending on instrument, they must e.g., not be too nearby to surfaces, shading or disturbed by vegetation. ("Guide to Instruments and Methods of Observation – Volume I", 2021, p.45).

Instruments are usually tested in one or more of the following:

Environmental test.
Electrical and electromagnetical interference (EMI).
Functional test ("Guide to Instruments and Methods of Observation – Volume V", 2018, p.52).

The list applies to Vaisala as well. Vaisala has facilities and capability to high variety of tests, that are run in new product development but also when customer returned instruments are evaluated. Returned instrument examination and level of it depends on device and test plan, but often it includes a visual and a functional test both in laboratory and in suitable environment. Suitable environment for testing e.g., a humidity and temperature probe is often a climate chamber that can increase temperature over probe's maximum operational temperature and decrease lower than its minimum temperature. For a wind sensor suitable test environment would be outdoors test field or a wind tunnel. These environmental tests often require best possible measurements. That can mean more precise, faster, more abundant amount of output data and sometimes testing over limits. The purpose of testing is foremost to measure if the instrument works like specified on a datasheet. Moreover, if a customer reported a measurement issue, it can ideally be repeated and measured in a controlled test environment.

In case of a new product the tests will be more plenty. They could even include the whole superset list mentioned by WMO:

> "high temperature, low temperature, temperature shock, temperature cycling, humidity, wind, rain, freezing rain, dust, sunshine (insolation), low pressure,

transportation vibration and transportation shock." ("Guide to Instruments and Methods of Observation – Volume V", 2018, p.53)

2.1.3 Data logging

Instruments measuring is further enhanced when it is possible to compare change in a parameter over time. For this use a measurement log with timestamps is required. This is called a data log. Systems creating data logs are called data loggers and they come in many forms. In many applications especially concerning safety, a real time data is desired as well (Campbell et al., 2013, p.1).

Concerning data correctness, sensor, data interface (e.g., cable and connectors) and data logger each can unwantedly alter the original measured parameter, such as wind speed. Also, some sensors have built in output values to indicate an error, which may show up as an uncommon spike in the data. For these reasons, the quality of the received data should be inspected. This process is called Quality Control (QC) (Campbell et al., 2013, p.2). In addition, Quality Assurance (QA) can be done simultaneously with data logging. Cambell et al. state that QA is meant to produce high quality data without a need for further actions. (Campbell et al., p.2). This means QA can consist of data parsing also.

Quality Control in their view could commonly be applied using six tests:

1. Sequential date and time data. No gaps or irregularities.
2. Data within a given reasonable range.
3. Persistence in the sense that some data should not remain the same constantly. Yet another data might.
4. Slopes and their rising and lowering times to see if they are expected for the given parameter or if they are erroneous.
5. Internal consistency test inspects if related parameters make sense. E.g., lowest wind speed cannot be more than the highest.
6. Spatial consistency involves comparing equal parameters between sensors and/or locations. (Campbell et al., 2013, p.5)

The tests would be used to flag the data. Flags could be available for both internal and external use. In addition, uncertainties could be given if seen appropriate. (Campbell et al., 2013, p. 11). Flags could be used to mark the QC failed data on the visualization (Campbell et al., p. 9). Similar methodology could be used to start and end a test. A logging system would know when it is supposed to function and when not.

Traditionally data loggers use special software, are costly and need power supply and/or PC (Fuentes et al., 2014, p.1). Need for a separate PC has been especially true for low budget Automated Weather Stations (AWS) (Burt, 2012, p.46). Yet, there have been project initiatives using e.g., Arduino microcontroller to make low cost, independent and open data loggers (Fuentes et al., p.1). Desktop computers, Raspberry Pi and data acquisition cards have also been used (Harrison, 2015, p.67).

Data loggers are highly crucial elements, as all data could be lost from all connected instruments due to its failure (Burt, 2012, p.36). Because of this, data should be periodically saved into a separate data backup (Burt, p.289).

Vaisala produces dedicated data loggers for sale that have extensive configurability, such as capability to be programmed using flowcharts and Python (Data management, 2023). QA is often integrated in the instruments themselves. Most have error codes and ability to flag data. This does not relieve a data logger from quality related responsibilities when best practises are looked for. E.g., configuration of a logger and QA flags concerning it is remarkably easier than customization of an instrument firmware to change its behaviour and output messages. Also, logger QA & QC cover the transmission medium and not just the instrument output. Once the data is ready to be used and preferably after passed quality tests, it is visualized.

2.1.4 Data visualization

In data visualization a comma separated data in a text file can be sufficient for seeing a clear change (e.g., a shorter line) in a small set of data, but usually more is needed. Colours, bars, and plots are among many ways to visualize the data and they require more than a simple text editor, such as a spreadsheets/statistics software (Burt, 2012, p.337).

Data visualization should help its users to comprehend the data (Chen et al., 2007, p.16). Depending on the graphic's use case, the appearance is chosen from the two:

1. Presentation graphics – summary of information for common viewers. Ideally one graphic.
2. Exploratory graphics – results review, e.g., for an analyst. Number can be more than hundreds of graphics. (Chen et al., p.14-15)

Many comprehensive software exists for data visualization. Yet some have coded visualization applications to application specific need such as high frequency data collection from multiple sources (Kelso et al., 2023).

At Vaisala several spreadsheet and statistics programs are extensively used for data visualization. Presentation form can be finetuned in that software. Test field and some other uses apply also Grafana analytics and monitoring tool for visualization.

2.1.5 Software engineering

Software engineering deals with software development and related activities. One of the first activities in it is defining the software requirements. Requirements Engineering (RE) should collect customer requirements analysing, modelling, and verifying them before creating the specifications (Tian et al., p.2). If RE fails, customer will not be satisfied.

Tom Glib wrote *Competitive Engineering* book involving Planguage language. He classified software requirements further as:

1. Vision
2. Functional
3. Performance
4. Qualities
5. Resources
6. Capacity
7. Resources
8. Design constraints
9. Condition constraints.

It is seen especially important that:

1. Important stakeholders should be identified and involved in RE to make requirements complete and satisfying. (Glib & Brodie, 2005, p.38).
2. Requirements are separated from the methods to meet them.
3. Key requirements are identified and focused on. They are the prime goal. (Glib & Brodie, p.39).
4. Success and failure are measured numerically. Moreover, survival level can be specified, describing the level on which existence is still possible. Potential and wanted requirements can be collected, to wait for the resources for their implementation. (Glib & Brodie, p.40).

Stakeholders view is more crucial in commercial software, but not as much with internal software, that is not sold. It is expected that internal software has less regulations and rules governing it, smaller user base and straighter feedback from that internal group of users. This means internal software can be more easily updated and changed as needed. However, requirements, methods, success, and failures need to be accounted, as they aim to make the software measurable. Measurable software can be more comprehensibly improved.

Other important principles in the software development are:

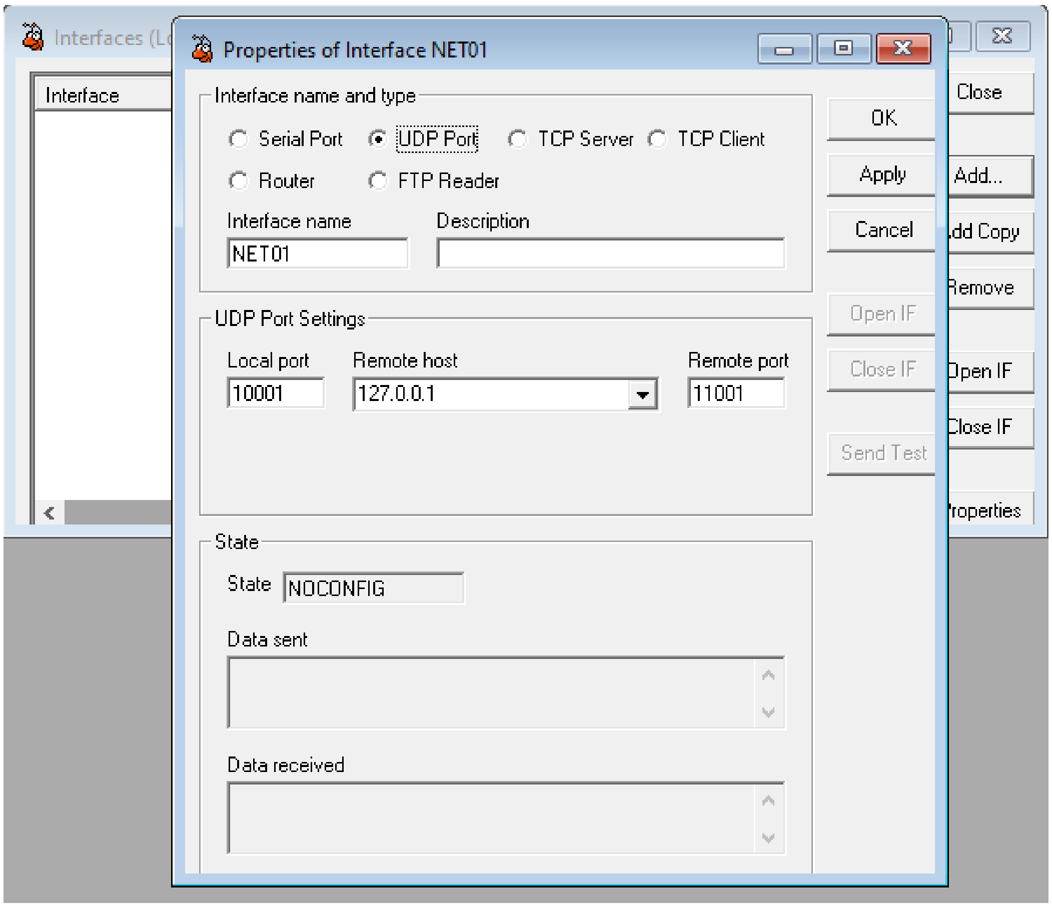
1. Formal specifications, that are created based on software requirements. (Tchier et al., 2015, p.2).
2. Modular design hiding individual modules revealing only their specifications. This is opposite to having all code within one module.
3. Verification-based programming meaning thoughtful programming instead of testing what works and what does not.
4. Goal oriented testing should be precise and systematic defining following:
5. Test goal,
   a. assumptions of what is the tested object and about its correctness,
   b. test deciding the result,
   c. test data selection criteria,
   d. condition when testing is finished,
   e. how results are analysed and
   f. that software can be created after all preceding subjects are successful. (Tchier et al., p.2).

The principles deal with software design and its implementation and lastly its testing. Many important principles can be fulfilled by using automated tests in the programming environment. To be able to do so, software should be designed so that it can be tested automatically. One requirement for this is the mentioned modular design.

2.2 Current software

Boson is used for both logging (and simulation) of sensors. It is often used when a reliable logging environment is set up for long running test with multiple instruments. Instruments can be polled for replies, or they can just be monitored as they send messages automatically. The received messages can be parsed using simple start and stop tag. Results can be saved on a text file to a local disk or e.g., to a OneDrive folder.

Boson runs on Windows and instruments are connected to it using serial ports or over network using TCP/UDP (Figure 3). Traditionally instruments output the data using serial ports, but in Boson the TCP/UDP option is applied increasingly using port servers in between instrument and logging software.



Properties of Interface NET01

Interface name and type

Serial Port [✓ UDP Port] TCP Server TCP Client Router FTP Reader

Interface name: NET01

Description:

UDP Port Settings

Local port: 10001

Remote host: 127.0.0.1

Remote port: 11001

State

State: NOCONFIG

Data sent

Data received

Figure 3. Boson interface configuration

There are two ways to capture and visualize data:

1. Local file logging: Save data from Boson into files and visualize them in common spreadsheets/statistics software.
2. Remote database logging: Setup the data transfer into InfluxDB database and visualize it in Grafana visualization application.

Yet, depending on the use case there may be a need to go through results more precisely. In such a case the *local file logging* method may be preferred. It produces easily usable unprocessed results. Even then Grafana visualization can be useful for monitoring the test, as the methods are not exclusive.

2.2.1 Local file logging

Text file logs show raw results from the instruments in multiple formats, that include CSV. Often, they need to be parsed into clean CSV format. This is usually the most time demanding operation because data amount can be so plenty. Then CSV-files can be visualized, and multiple calculations done in spreadsheet or other suitable software. Regularly results with millions of values cannot be visualized with that software because they freeze or inform that memory runs out or data is too plenty. This often leaves a few options:

1. Hardware and software environment updates.
2. Developing scripts for data handling and visualization.
3. Handling less data at once.
4. Waiting.

Hardware and software environment updates include adding RAM-memory to computer and changing Excel to 64-bit version if 32-bit was in use. Yet, company computers are under centralized maintenance and deviation from the standard is not recommended or possible.

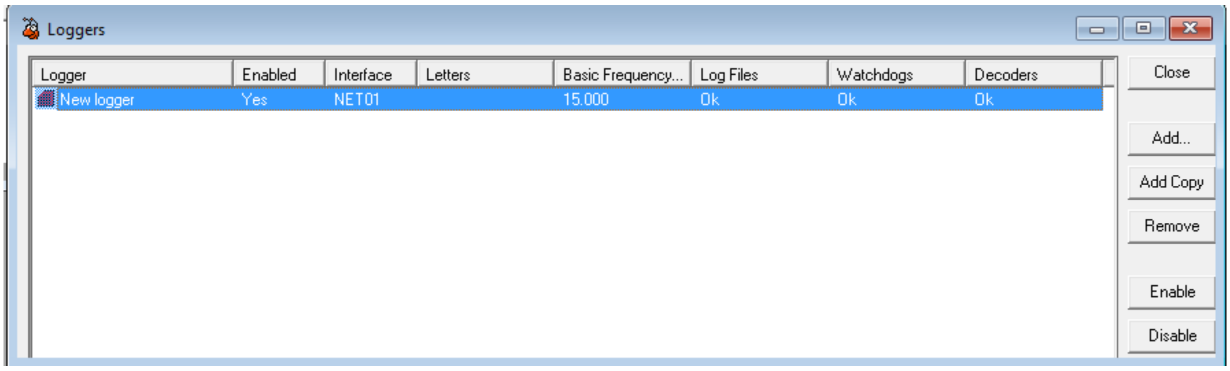
However, for example, Python scripts with Pandas and NumPy libraries are highly capable and fast compared to spreadsheets/statistics software, but they demand customization and programming skills. They are very effective when used to parse data for other software to use.

Handling less data at once is preferred but it does not help in creating the big picture. For example, plots may need to be split into multiple, each holding e.g., 100 000-time domain values. Also, in some cases there is no knowledge of what should be captured and then it may be best practice to collect extra data for later analysis, for the moment when it is known what to look at and what to ignore.

Waiting for software to finish their processing is a very time demanding operation. It also takes place in several stages. There is delay first for data to open, then to plot it, scaling the plot etc. It is also questionable how such waiting fits into today’s world where processing capability exists but is not fully applied. This freezing and delaying is common with existing processing software and during such the pause keeps all software instances from taking commands. Nor does it say how long it will take. Unlike in Python scripts, wherein progress can be shown to the user.

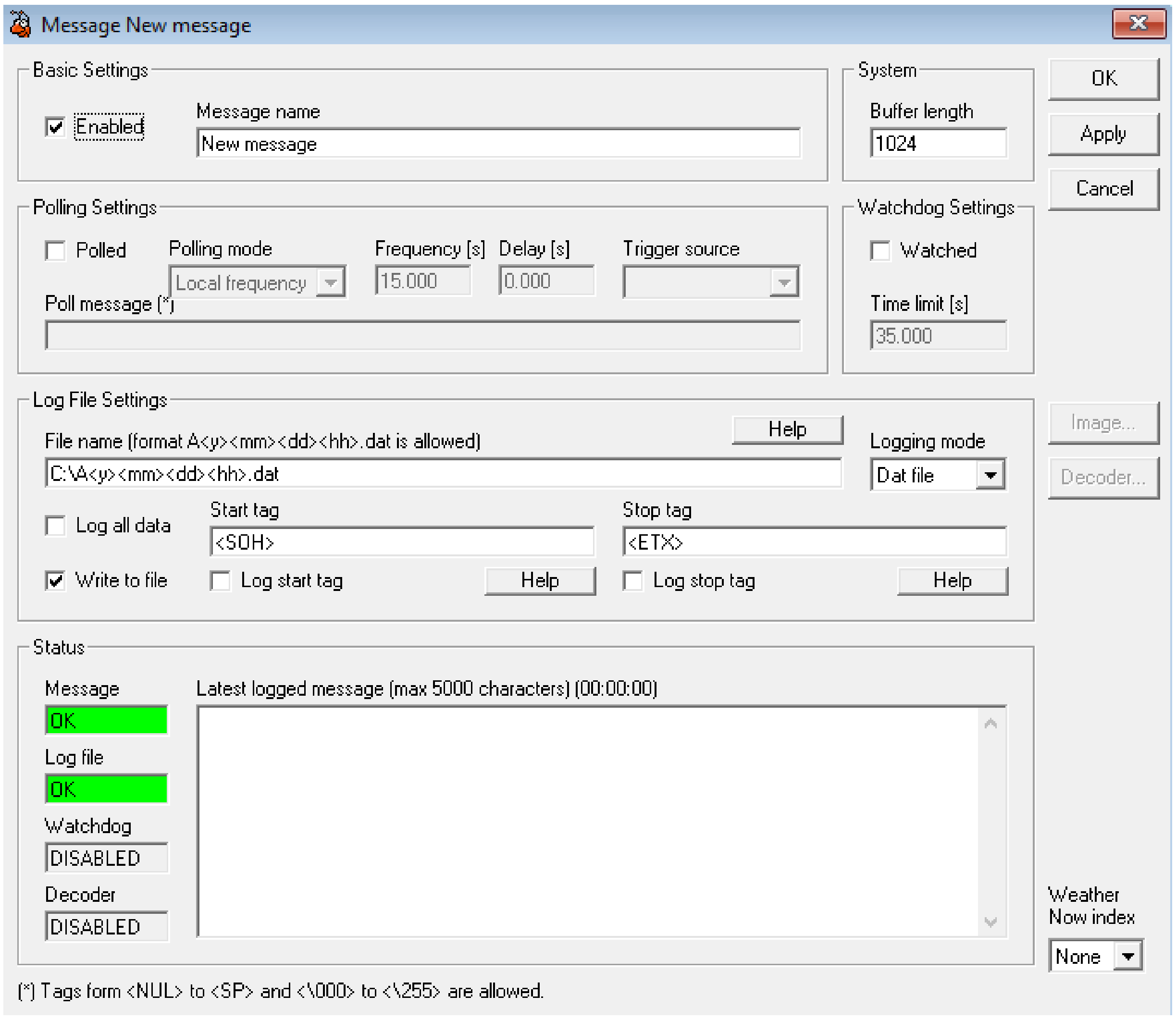
Following Figures 4 and 5 show the configuration of logger, file path, start and end tags for parser that is done after Interface has been configured. After a logger is set up and the message configured, data is logged to a file on disk.

Polled option is only used if instrument is not automatically sending messages. Instruments can commonly be configured to send automatic messages, but sometimes there is a need for more than one message type retrieval. For such a case, polling is useful.



Logger	Enabled	Interface	Letters	Basic Frequency...	Log Files	Watchdogs	Decoders
New logger	Yes	NET01		15.000	Ok	Ok	Ok

Figure 4. Boson Loggers menu


Message New message

Basic Settings

[✓] Enabled

Message name: New message

System

Buffer length: 1024

Polling Settings

[ ] Polled

Polling mode: Local frequency

Frequency [s]: 15.000

Delay [s]: 0.000

Trigger source:

Poll message (*):

Watchdog Settings

[ ] Watched

Time limit [s]: 35.000

Log File Settings

File name (format A<y><mm><dd><hh>.dat is allowed): C:\A<y><mm><dd><hh>.dat

Logging mode: Dat file

[ ] Log all data

Start tag: <SOH>

Stop tag: <ETX>

[✓] Write to file

[ ] Log start tag

[ ] Log stop tag

Status

Message: OK

Log file: OK

Watchdog: DISABLED

Decoder: DISABLED

Latest logged message (max 5000 characters) (00:00:00)

Weather Now index: None

(*) Tags form <NUL> to <SP> and <\000> to <\255> are allowed.

Figure 5. Boson Message menu

2.2.2 Remote database logging

The method to transfer data from Boson to InfluxDB and visualize it in Grafana uses *local file logging*. Files are then monitored by a Python script that uses a user modifiable parser and InfluxDB library which in turn send the parsed data into the database. Python script needs to be modified when a new sensor or message needs to be sent to the database. This process is pictured in Figure 6.

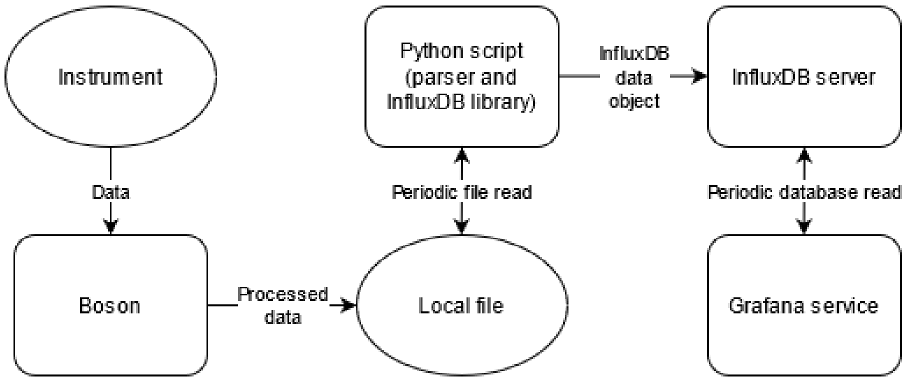


Figure 6. Boson database logging process

2.2.3 Processing and visualization

After log files are complete, they are commonly parsed. In the parsing process corrupted lines can be removed and commas added like wanted. In Figure 7 a raw (unprocessed) data log from Boson is presented with extra spaces and line feeds.

```
2022-12-21 12:28:54, $00.1,125.9

2022-12-21 12:28:54, $00.5,137.9

2022-12-21 12:28:54, $00.3,147.2

2022-12-21 12:28:54, $00.4,155.7

2022-12-21 12:28:55, $00.4,163.9
```

Figure 7. Unprocessed data log

Sometimes columns can be removed altogether, if they are not related or the data would be too large to handle in common spreadsheets/statistics software. Often the minimum level of processing is to add headers and remove extra spaces and lines. Figure 8 presents Figure 7 data after processing.

```
Time,Wind speed,Wind direction
2022-12-21 12:28:54,00.1,125.9,
2022-12-21 12:28:54,00.5,137.9,
2022-12-21 12:28:54,00.3,147.2,
2022-12-21 12:28:54,00.4,155.7,
2022-12-21 12:28:55,00.4,163.9,
```

Figure 8. Processed data log

Spreadsheets/statistics software is used to understand the data better. Plots are commonly used to see changes over time in different parameters. Yet, many other methods are also used, like conditional formatting. Also, XY-plots can be applied to see how one parameter changes compared to second.

2.3 Future needs and users

Because of the complexity of database collection, growing need for more productivity, efficient data handling and its synthesis, more than text-based logging and Python based database collection were desired. Moreover, database logging, its visualization and understanding should be easy, so enabling it for all common Boson users. Figure 9 represents the need for a new flow path of instrument data from instrument to Boson and to Grafana visualization.

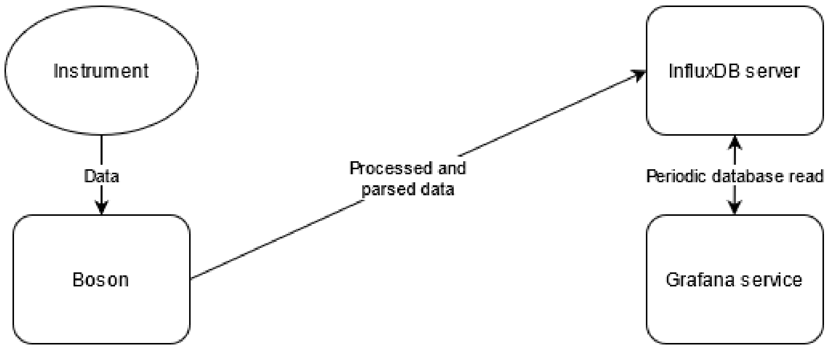


Figure 9. Vision of Boson database logging process

Boson is designed for only internal use and mainly used at R&D and innovation centre and its nearby test field. Users vary from technical personnel to scientists. Currently users cannot rely solely on Boson for database connectivity, yet objectives satisfying and database logging process according to the vision in Figure 9 would make Grafana's effective visualization tools available for all its users.

2.4 Facts

Currently used software in version control system (VCS) GIT was Visual Basic based. It used .NET version 2.5.2 dated 2007. The executable in common use was even older. The old .NET version set limits for available features and especially plugins. Plugin availability limitations could have forced development of new features from scratch. For this reason, the adaptation of a more up-to-date .NET framework became of interest.

Another limitation was that source code of Boson was so large and coded in Visual Basic. This hindered rewriting and effective debugging of the code.

Lastly, the .NET software version in GIT was not yet ready and fully functional. Some features were never implemented unlike in the old version, that was in use. It needed to be fixed just to run it.

3 Implementation

3.1 New software

All theory and practices about Software Engineering (Chapter 2.1.5) were not necessarily applied to new resulting software. The software is not recreated totally, and it is not to be sold and it's available only in the company internally. Due to these limitations, it is not seen necessary to apply overly time or cost heavy processes. The applied processes and methods should suitably fit for their purpose.

The basis for new Boson called BosonZP.NET was a VB.NET version program that was not usable. It was an unfinished port from Visual Basic into .NET framework 2.5.2. As platform change could introduce more bugs, it was compiled in the final .NET framework version 4.8, wherein all relevant bugs and problems were to be solved, and development continue.

3.2 Wanted requirements

The key requirements come from six criteria that are presented in Objective chapter and the logging use cases (the software must be usable in these cases). Wanted requirements are not evaluated in this thesis, but presented mostly as a future development goal, as their fulfilment will take more time. Some of them were met in the new software.

Functional and data requirements

Software must be able to log instrument data and send it into a database. Data must not be altered or corrupted unintentionally.

Data should be sent to the defined IP address at a defined interval, that is given in seconds. Headers should be given to the instrument data or dummy headers used instead otherwise. The resulting message should be represented to user's knowledge.

Performance requirements

There should be no longer than 3 second response time without providing a separate explanation. Separate explanation being for example popup window with text.

Software should not freeze when used in its purpose.

Accuracy and capacity requirements

Results in the database must describe real output of instrument, like received by Boson.

Reliability and availability requirements

Software must not fail without user interaction. User initiated failures must be prioritized for fixing queue based on their appearance likelihood. E.g., very rare configuration could cause a failure but not be feasible to debug and solve.

Data should not be lost. Duplicated data storage path must be provided, meaning the second one survives if first one fails. It must be possible to target separate storage locations.

Software availability should be 24 hours a day, 365 days per year. Database functionality should be available when network and database server permit.

Maintainability and supportability requirements

Software needs to be debug able and maintainable by non-original developers.

Testability should be designed and applied for most relevant functionality. These are e.g., main logging functions, windows, buttons, and parser.

Software and its development are to be supported on a need basis. A crucial bug is considered a need, as well as a group of less severe ones collected over time. A user guide and/or instruction must be created for the new functionality.

Preceding requirements assume Visual Basic code remains capable of being developed and debugged.

Adaptability requirements

Software is expected to run in Windows 10.

Security, usability, and understandability requirements

Software should not pose unaccepted risk to IT security.
Software must be used only internally in Vaisala. It should be free to use without limitations.
Software should be easy to use and familiar to old users. Only English language is to be used.
Symbols and words should be understandable by users.

Look and feel requirements

Software must keep old Boson software look and feel but be distinguishable from it. User that distinguishes the two software know what software functionality to expect, but still feel familiar with what they see.

Legal requirements

Software must not violate any software licenses knowingly. This means used software module licenses should be read and obeyed, e.g., have licenses available to read.

3.3 Methods

3.3.1 Use cases

Usually Boson is only set up for longer than a few hours test. For e.g., 15 minutes logging there is more suitable internal software available. Also, for interacting with the instrument, Boson is not preferred. Yet it excels in its logging purposes. Example use cases are following:

- Short < 8-hour tests

  In the wind tunnel or in a laboratory test on table it may be beneficial to apply Boson especially if there are multiple instruments to be logged simultaneously. Boson permits maximum of 16 serial ports and much more by applying port servers that are limited by IP-address space.

  In wind tunnel test instrument is installed on a mount, and a wind sequence is run while logging the device under test (DUT) and reference (REF) sensor. The results are logged into separate files but can be connected afterwards using timestamps.

- Medium < 7-day tests

  Climate chamber and running multiple test cycles experience even more benefit from Boson, as it is easy to set up the logging into OneDrive or other cloud storage, that is linked to Windows file system and remotely viewable.

  Climate chambers are metallic boxes with a door. They commonly have adjustable temperature and humidity. Instruments are left inside commonly from two hours to up to one week or more. Instruments are connected to computer using wires going through inlet hole and a seal. There must usually also be a REF sensor for comparison.

- Long or continuing > 7-day tests

  Test field with months and years long runtime between updates relies crucially on Boson for connecting to instruments. Not only does it allow many connected devices, but it also makes their upkeep easier.

  When e.g., a software update is released for an instrument, there may be a need to install one instrument with new software to test field. Then it needs to be wired into computer running Boson or to port server connected to it.

3.3.2 Design

Programming language

C# -language was selected to be used for new functionality, when possible, because of its familiarity to the developer, company employees and maintainability. Moreover, in .NET Framework's three available programming languages C# is preferred over F# due to C# having roots in C and C++ (".NET Programming", 2022). This would further increase maintainability over F# and VB as C and C++ developers would have an easier time to adapt into Boson development using C#.

Current architecture

High-level software architecture of Boson is simple. Everything is in same project with dependencies on some external libraries. All code is in VB.NET and it uses object-oriented software architecture, where classes are used to hold functionality (Oussalah, 2014). Figure 10 displays a simple presentation of current architecture.

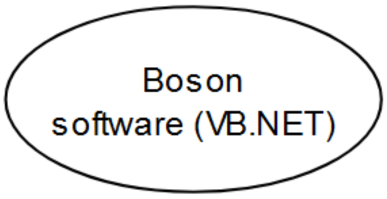


Figure 10. Boson architecture (external dependencies hidden)

Proposed architecture

New features are to be introduced in Meson dynamic link library (DLL) that should then be programmed in C#. DLL was to be used to create a separation between old and new code (Boson and Meson) and to be able to use another programming language than Visual Basic (Figure 11). Not only could Meson be used in Boson but elsewhere also, because it was to be a separate module written in a common language.

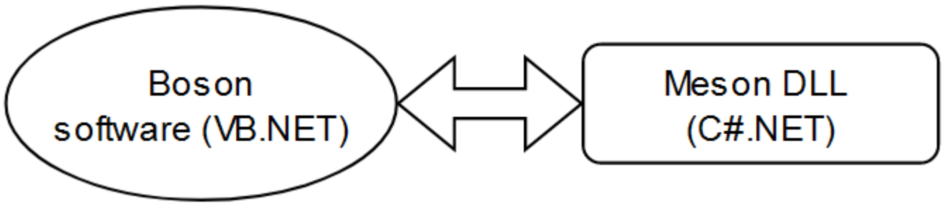


Figure 11. New software structure

Having Boson and Meson separated means they need means to communicate. Interface would need to be written using VB in Boson to call functions that were to be written with C# in Meson.

New architecture would be a component-based software architecture, which at component level would still be object-oriented in Meson also. The whole architecture is pictured in Figure 12.

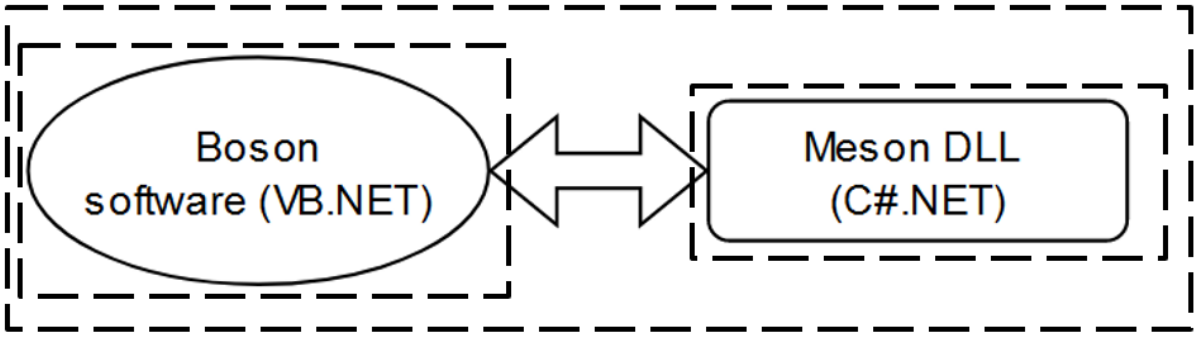


Figure 12. New proposed software architecture

3.4 Software elements

3.4.1 Boson

To reach the desired features using old Boson VB.NET code, the target .NET framework needed to be updated into a fresher one. Old .NET framework versions are not as supported by plugins and do not offer the latest features. Old target framework version was 2.5.2 and this was to be updated into 4.8. There were some 11 years and 20 distributions in between (".NET Framework", 2023).

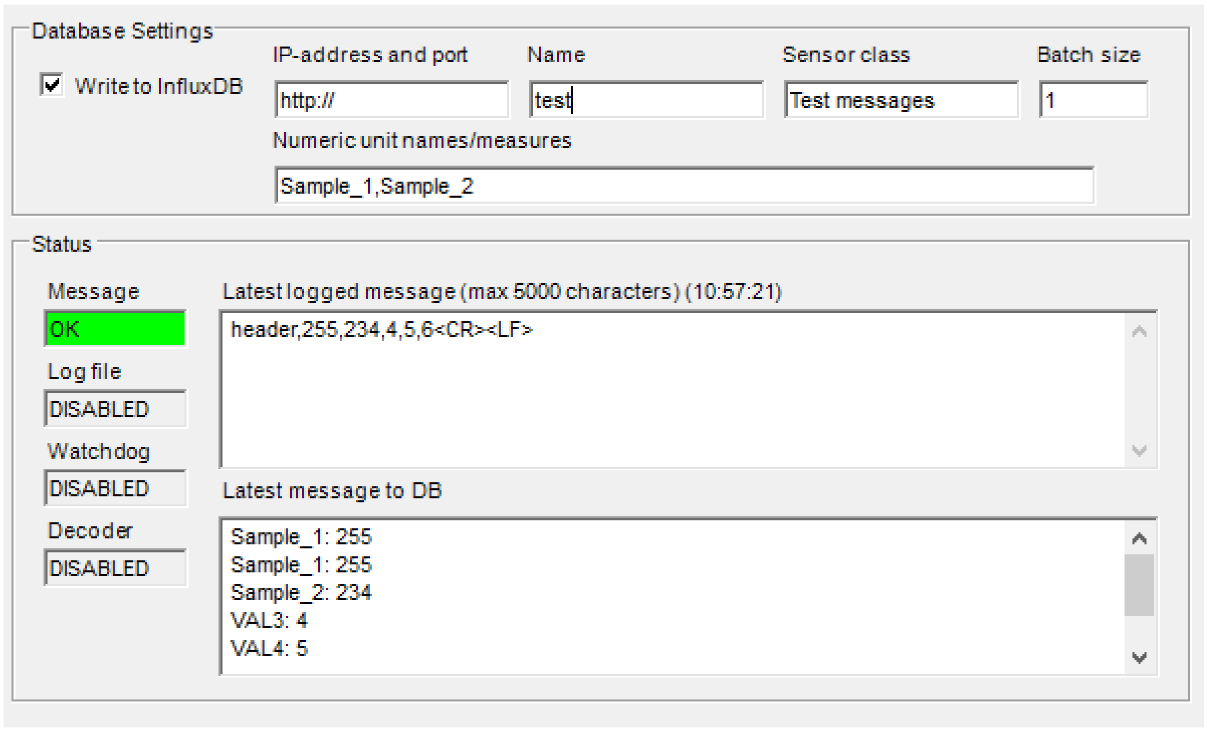
Old development environment is unknown, but it was likely Visual Studio 2005 or similar. Using such is not feasible when newer options are available. For this reason, Visual Studio 2019 was used in the beginning and Visual Studio 2022 adapted later during the development.

3.4.2 Meson library

InfluxDB database plugin was downloaded from addon manager in Visual Studio. Its licenses required visibility for which new button was to be created in GUI. This button should launch *licenses.txt* file listing all licenses requiring it.

InfluxDB and Grafana were installed on the development PC. This allowed local testing without need for external server. This option would remain available for future use also, when Grafana is used locally and there is no need to send the data to a server on network.

InfluxDB plugin tests began within Meson using simple example code test in C#, but evolved into systematically reliable method, that sent consistent results into database. Part of this development was the parser, that was needed to make a simple and effective data to plot method. Majority of data would be numerical, so a simple method was used. C# code would search through all received strings one at a time looking for numbers. When it would find an integer, it would continue until it found a decimal separator or similar symbol that represents a change in a number, e.g., change from a whole number into fractional part. It would then form the number accordingly like purposed by the symbology. Users could pick headers for each separate number or allow the software to pick itself. Yet, simple headers would be used by default, in series and they would not be describing. This process is shown in Figure 13, wherein named quantities are sent into database along with unnamed numbers.


Database Settings

☑ Write to InfluxDB

IP-address and port	Name	Sensor class	Batch size
http://	test	Test messages	1

Numeric unit names/measures: Sample_1,Sample_2

Status

Message: OK

Log file: DISABLED

Watchdog: DISABLED

Decoder: DISABLED

Latest logged message (max 5000 characters) (10:57:21)

```
header,255,234,4,5,6<CR><LF>
```

Latest message to DB

```
Sample_1: 255
Sample_1: 255
Sample_2: 234
VAL3: 4
VAL4: 5
```

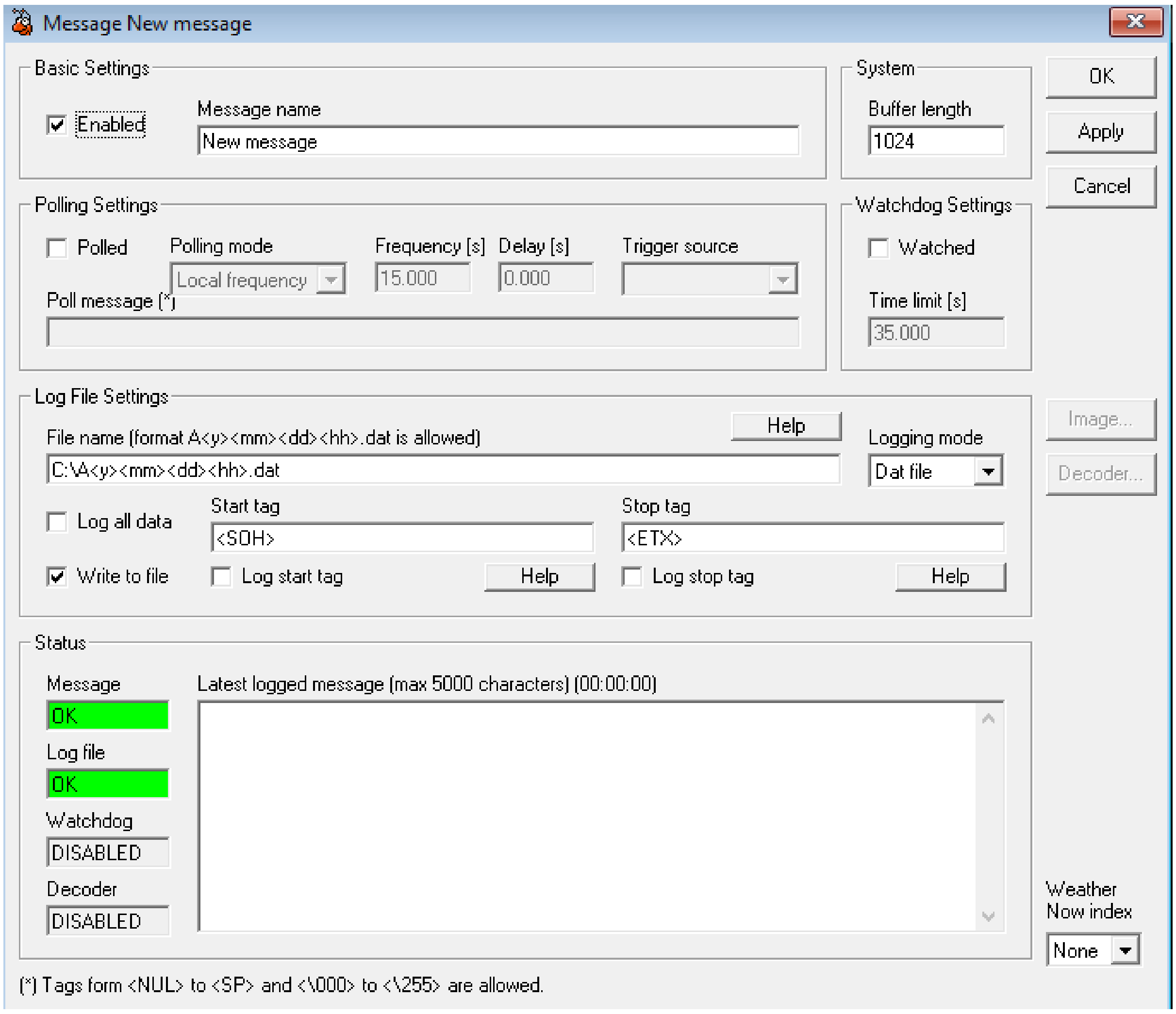
Figure 13. Parser and unit naming

Batch size was final larger change. During debugging a small static batch size was in use to receive instant feedback in Grafana, but for normal use, InfluxDB instructions recommended using a large batch size to decrease network traffic overhead (“Optimize writes”, n.d.). Interface code in VB concluding of < 10 lines was used to collect a buffer of batch size of received messages. When the buffer became full, it was sent to Meson.

3.4.3 Dialogs

In Boson code the GUI was changed first. New buttons and dialogs were put in place. Their coding style and naming was adapted from old code. This enabled them to mimic old and known behaviour and programming style. E.g., input boxes could be frozen, and text be painted red.

Most changes took place in Message configuration. In old configuration (Figure 14) only log files were configured, but database settings needed a placeholder also.



Message New message

Basic Settings
- [✓] Enabled
- Message name: New message

System
- Buffer length: 1024

Polling Settings
- [ ] Polled
- Polling mode: Local frequency
- Frequency [s]: 15.000
- Delay [s]: 0.000
- Trigger source:
- Poll message (*):

Watchdog Settings
- [ ] Watched
- Time limit [s]: 35.000

Log File Settings
- File name (format A<y><mm><dd><hh>.dat is allowed): C:\A<y><mm><dd><hh>.dat
- Logging mode: Dat file
- [ ] Log all data
- [✓] Write to file
- Start tag: <SOH>
- [ ] Log start tag
- Stop tag: <ETX>
- [ ] Log stop tag

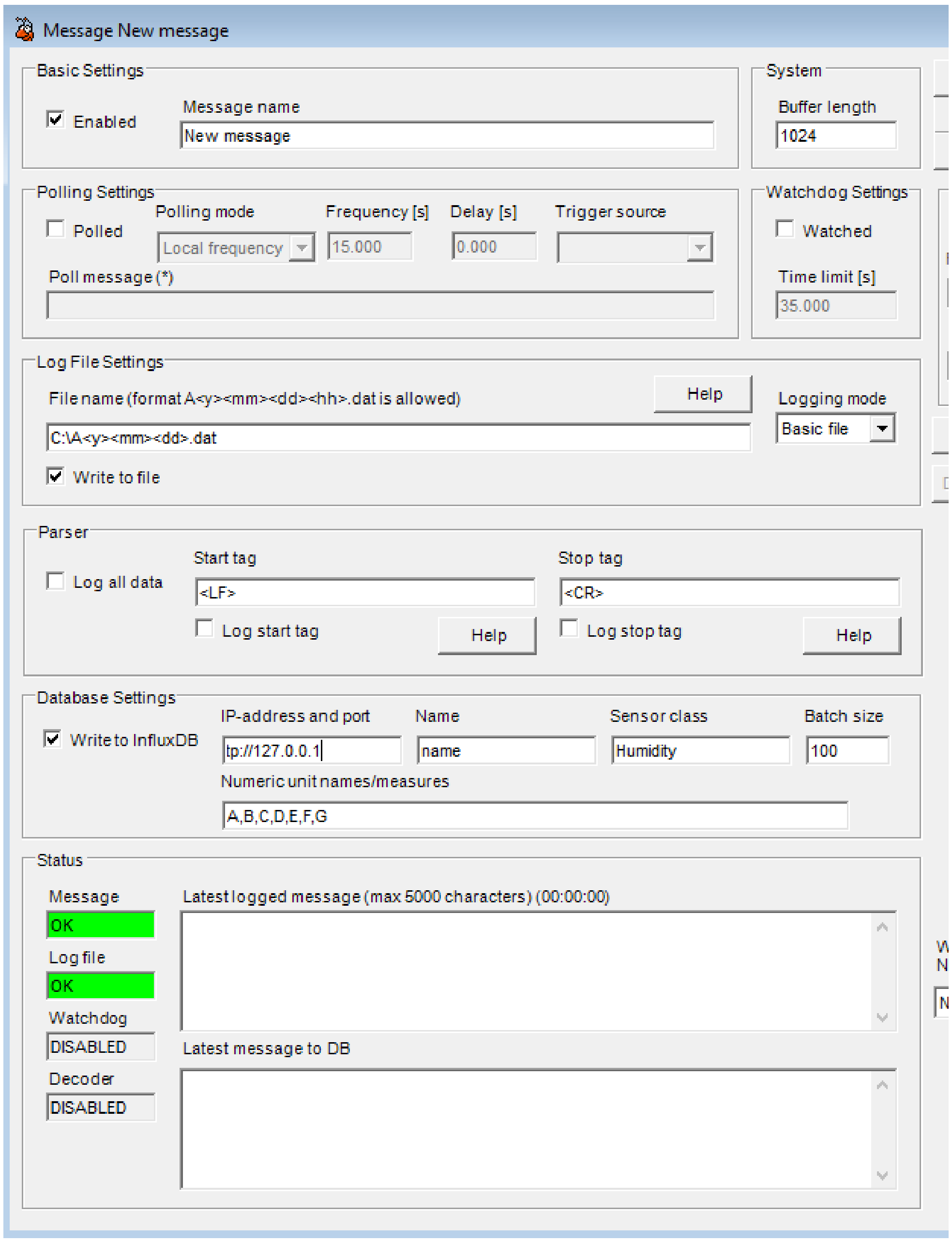
Status
- Message: OK
- Log file: OK
- Watchdog: DISABLED
- Decoder: DISABLED
- Latest logged message (max 5000 characters) (00:00:00)

Weather Now index: None

(*) Tags form <NUL> to <SP> and <\000> to <\255> are allowed.

Figure 14. Boson Message configuration

Database settings were embedded within the same Message configuration window (Figure 15). Both file and database logging were optional and could be used like desired. If both were enabled and successful, there would be two copies of same data.


Message New message

Basic Settings
- [✓] Enabled
- Message name: New message

System
- Buffer length: 1024

Polling Settings
- [ ] Polled
- Polling mode: Local frequency
- Frequency [s]: 15.000
- Delay [s]: 0.000
- Trigger source:
- Poll message (*):

Watchdog Settings
- [ ] Watched
- Time limit [s]: 35.000

Log File Settings
- File name (format A<y><mm><dd><hh>.dat is allowed): C:\A<y><mm><dd>.dat
- [✓] Write to file
- Logging mode: Basic file

Parser
- [ ] Log all data
- Start tag: <LF>
- Stop tag: <CR>
- [ ] Log start tag
- [ ] Log stop tag

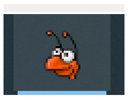
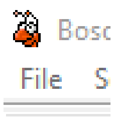
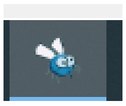
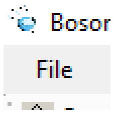
Database Settings
- [✓] Write to InfluxDB
- IP-address and port: tp://127.0.0.1
- Name: name
- Sensor class: Humidity
- Batch size: 100
- Numeric unit names/measures: A,B,C,D,E,F,G

Status
- Message: OK
- Log file: OK
- Watchdog: DISABLED
- Decoder: DISABLED
- Latest logged message (max 5000 characters) (00:00:00)
- Latest message to DB

Figure 15. BosonZP.NET Message configuration

Moreover, to be able to tell the difference between old Boson and BosonZP.NET, their icons were changed. Old software mascot was a red bug, and BosonZP.NET introduced a blue bug (Table 1).

Table 1. Icon changes
Icon changes between old and new Boson.

Software implementation	Taskbar view	Title bar view
Boson		
New software		

3.4.4 Functions

Meson library

Meson DLL library consisted of database related functions. The most important are:

- UnixTime.Time, providing correct Unix time stamps for data that is sent into database.
- InfluxDB.Send, processing the data and sending it to database.
- Check.Send, inspects the parameters given to *Send*.
- Parser.PlainNumbers, parses and processes the data given to *Send*.

Meson tests

To test Meson functions, a new test project was created under same solution. Initially only one test method was used, which was called *PlainNumbersTest*. Four different message examples from different devices were given to it as an input, and it was expected to give a correct output to each. E.g. input string *"NO2 (ppm): 0.004"* would have expected output of an array of double type with two numbers, 2 and 0.004.

Boson software

New functions were created to support new dialog functionality. These were mostly based on existing templates, as old dialogs were used as a basis for new functionality. Most crucial new code were the lines used to send data to Meson's *InfluxDB.Send* function.

3.5 Initial test

3.5.1 Compilation and debugging

Implementation began after research into Boson's code and how it is structured. It appeared to be over tens of source code files. Debugging was not possible without changes. Compilation did not work out of the box, likely because of development environment, target framework changes and incomplete .NET based Boson software version. After compilation was able to run and finish, the program did not run. One solution was to remove language options and use English as default. Fixing multiple languages support was possible but not seen feasible, as upkeeping one language also makes more sense in internal use.

Debugging revealed that not all functionality was there, as some functions were empty or incomplete. E.g., buttons and dialogs could be in place but have no function. Fortunately, most crucial logging functions were enabled.

3.5.2 Boson corrections

Boson code or installation was in many cases corrected. Either a bug needed to be fixed or something was missing and needed to be improved. Several issues are listed below:

- Boson uses bitmap image files, and it does not run without them. They are used or e.g., icons. They needed to be installed in the same folder where the executable resides.
- Terminal tool is used for serial communication. It was not working. There was a reported issue that data is not visible in the terminal. This was fixed by writing new code for the interface dialogs.
- Boson was traditionally capable to search weather from internet. At some point during the years and development this functionality was lost. It was corrected and solved some other more minor issues at same time.

3.5.3 Use

In October 2022 BosonZP.NET was adapted to use for the test field in testing purpose. It started collection of equal data as old database collection system and sent it into database to be plotted together with new. These were to be monitored since then.

Later the software was shared to some co-workers and installed into multiple test computers for test use. It appeared some computers were able to run the new software, but some were not. Application simply might not start and creating an irrelevant log file. This could indicate a problem with

- dependencies, as .NET applications require more background software than for example Microsoft Foundation (MFC) software ("Application deployment", 2023),
- configuration, because some computers with pre-installed Boson were able to run the new software,
- security settings and/or user permission, or/and
- compatibility between operating system and .NET framework in use.

These common issues were analysed further starting from the most obvious. Security settings, user permissions and operating system are not supposed to be an issue, because using administrator rights and all computers running Windows 10 should be sufficiently acceptable.

Compatibility and dependencies concerning .NET framework and related software were not expected to be a problem because a release version was compiled in Visual Studio. It should consist of needed files.

Configuration subject and related search for old Boson installation packet revealed that it was installed using a *setup.exe*. This indicated the difference in two installations of old Boson and that of BosonZP.NET. BosonZP.NET was distributed from Visual Studio output directories without an installer. Earlier finding of BosonZP.NET opening and running correctly in those computers that can run old Boson also supported this assumption. It could be inspected using few methods:

1. Install BosonZP.NET files inside old Boson *setup.exe* installation folder and see if it runs as normal user or as administrator. It is expected to work.

   Table 2. 1st installation method

   First installation method worked only as administrator.

User	Result
Normal	Fail
Administrator	Pass

2. Install BosonZP.NET to a fresh Windows 10 with no Boson installation history and see if it works. It is not expected to work.

   Table 3. 2nd installation method

   Second installation method was not functional at all.

User	Result

Normal	Fail
Administrator	Fail

3. Try to fix the previous result by installing old Boson using *Setup.exe* and then bringing BosonZP.NET files to the main installation directory holding Boson.exe. This is expected to work at least as Administrator.

Table 4. 3rd installation method

Last installation method was functional with all access rights.

User	Result
Normal	Pass
Administrator	Pass

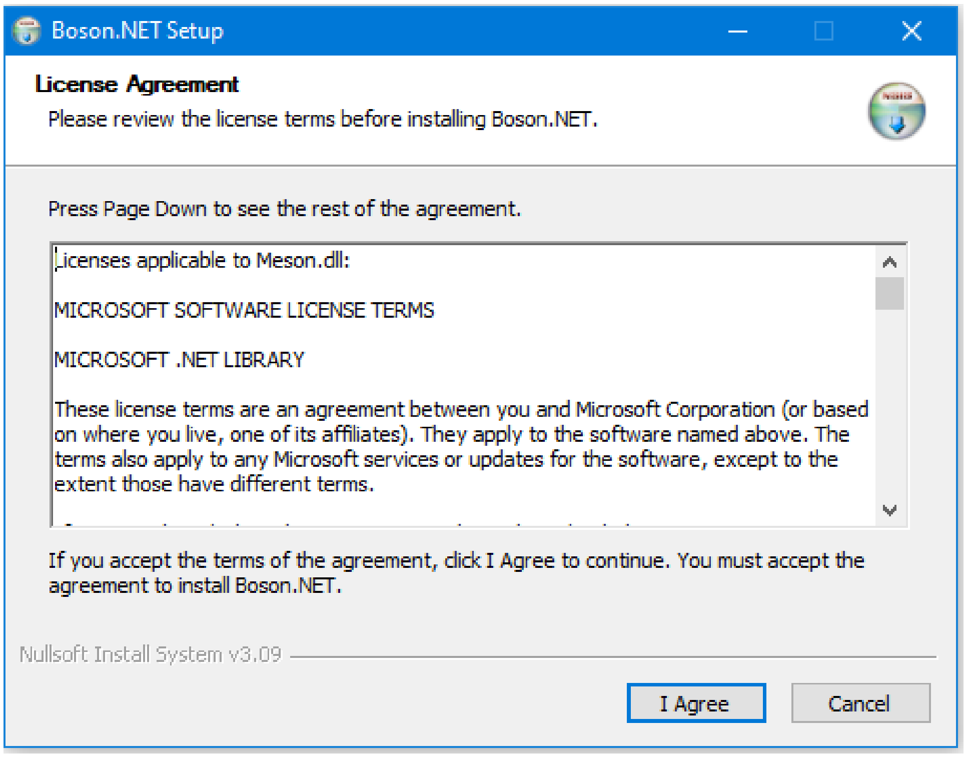
Further inspection of old Boson's *SETUP.lst* indicated that the *setup.exe* tries to find certain files from *WinSysPathSysFile* that points to *C:/Windows/system32/* . If they are not found, they are installed from a *.CAB* file, that also holds icons, images, and other misc. files.

From this process it seems a fully functional step-by-step installation instruction could be created:

1. Install old Boson using *setup.exe* and keep newest files if asked.
2. Open directory (e.g. C:\Program Files (x86)\BosonZP) holding Bozon.exe
3. Extract BosonZP.NET files to the directory replacing if needed.

The method overwrites old Boson files possibly invalidating its use. If old Boson needs to be used as well, BosonZP folder (2.) should be copied, renamed, and used instead (3.). The described installation method is not user friendly and not even on the original standard level.

Installation method was later improved by using NSIS application. Included example script was modified to distribute BosonZP.NET release files and legacy Boson DLL files using InstallLib macro ("Appendix B", n.d.). The NSIS script produced installer software for a user. Licenses used in Meson were listed in the installer (Figure 16). Old Boson licensing is unknown.


Boson.NET Setup

License Agreement

Please review the license terms before installing Boson.NET.

Press Page Down to see the rest of the agreement.

Licenses applicable to Meson.dll:

MICROSOFT SOFTWARE LICENSE TERMS

MICROSOFT .NET LIBRARY

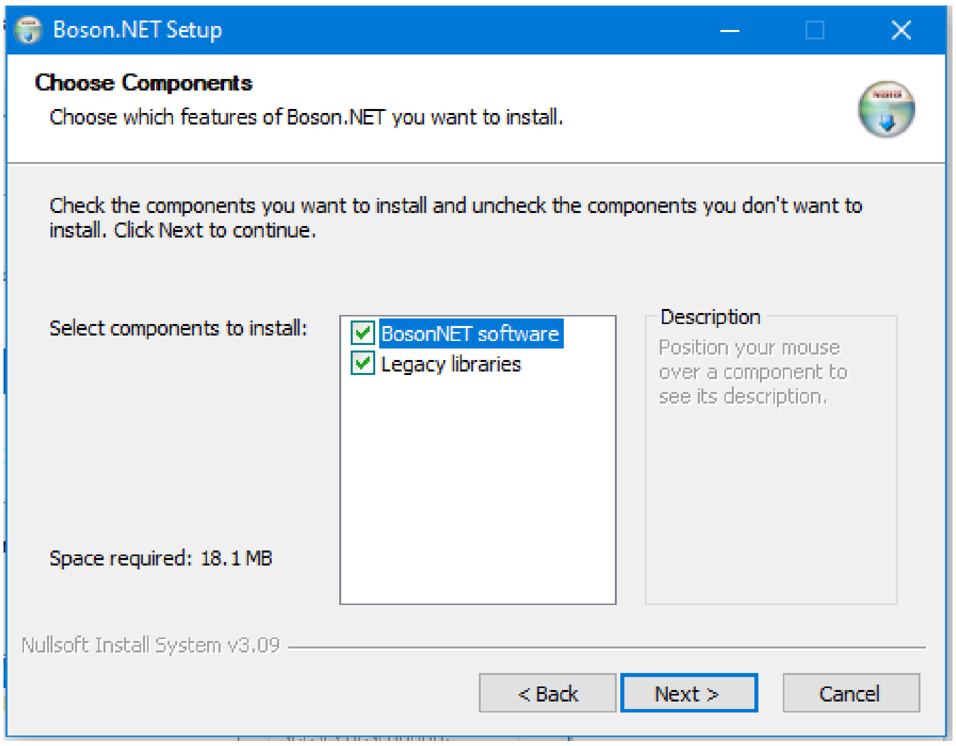
These license terms are an agreement between you and Microsoft Corporation (or based on where you live, one of its affiliates). They apply to the software named above. The terms also apply to any Microsoft services or updates for the software, except to the extent those have different terms.

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Boson.NET.

Nullsoft Install System v3.09

Figure 16. BosonZP.NET installer and licenses

Installation of BosonZP.NET and legacy Boson files were set optional (Figure 17). For most users the installation of both would be ideal.


Boson.NET Setup

Choose Components

Choose which features of Boson.NET you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:
- [✓] BosonNET software
- [✓] Legacy libraries

Description

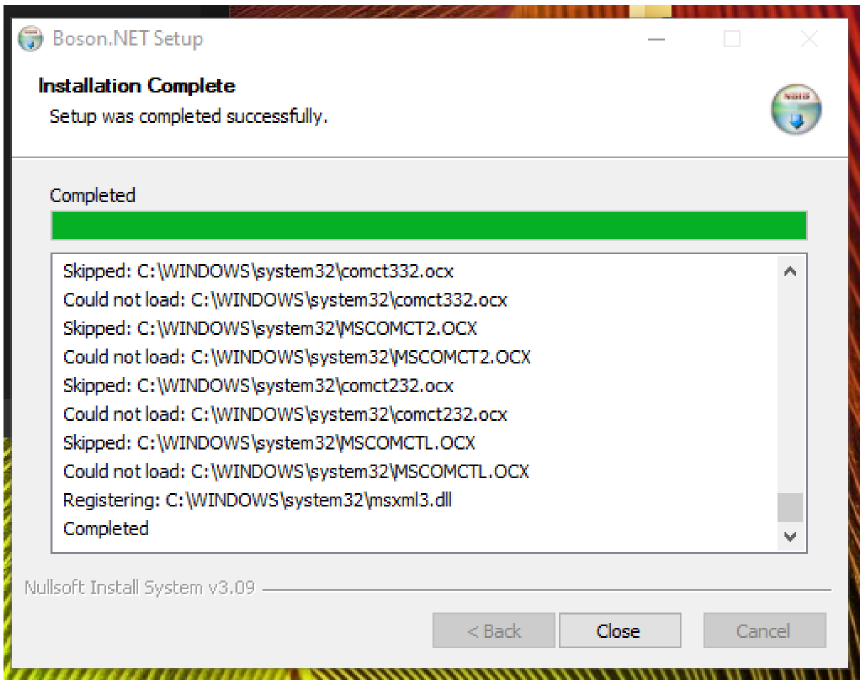
Position your mouse over a component to see its description.

Space required: 18.1 MB

Nullsoft Install System v3.09

Figure 17. BosonZP.NET installation

Test of the setup on a fresh test laptop revealed that installation of legacy libraries requires administrator rights. If setup was started as normal user, the setup would complain *"Could not load"* for several DLL (Figure 18).



Installation Complete

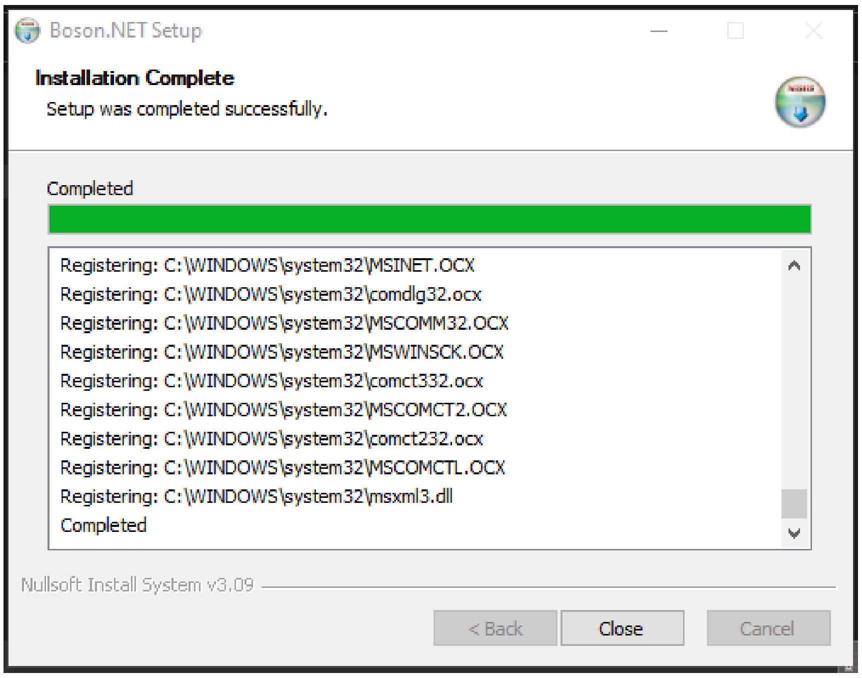
Setup was completed successfully.

Completed

```
Skipped: C:\WINDOWS\system32\comct332.ocx
Could not load: C:\WINDOWS\system32\comct332.ocx
Skipped: C:\WINDOWS\system32\MSCOMCT2.OCX
Could not load: C:\WINDOWS\system32\MSCOMCT2.OCX
Skipped: C:\WINDOWS\system32\comct232.ocx
Could not load: C:\WINDOWS\system32\comct232.ocx
Skipped: C:\WINDOWS\system32\MSCOMCTL.OCX
Could not load: C:\WINDOWS\system32\MSCOMCTL.OCX
Registering: C:\WINDOWS\system32\msxml3.dll
Completed
```

Figure 18. Failed DLL installation

Yet, when it was run as administrator, each DLL file installation succeeded with text *"Registering"* (Figure 19). Then the software was also capable to open and run.



Installation Complete

Setup was completed successfully.

Completed

```
Registering: C:\WINDOWS\system32\MSINET.OCX
Registering: C:\WINDOWS\system32\comdlg32.ocx
Registering: C:\WINDOWS\system32\MSCOMM32.OCX
Registering: C:\WINDOWS\system32\MSWINSCK.OCX
Registering: C:\WINDOWS\system32\comct332.ocx
Registering: C:\WINDOWS\system32\MSCOMCT2.OCX
Registering: C:\WINDOWS\system32\comct232.ocx
Registering: C:\WINDOWS\system32\MSCOMCTL.OCX
Registering: C:\WINDOWS\system32\msxml3.dll
Completed
```

Figure 19. Successful DLL installation

Administrator installation was forced by updating installation script setting *RequestExecutionLevel* from *user* to *admin* (“Reference/RequestExecutionLevel”, 2013). After this the installer would not run as normal user. As administrator, installation of both BosonNET and legacy libraries would succeed and not be dependable on old Boson *setup.exe* any longer.

4 Evaluation

Software developers use half of their time to fix bugs. Less time is used for developing new features. Bug fixing consists of both finding the reason for wrong behaviour and correcting it. Correcting update of source code is known as bug-fixing change (BFC). (Rodríguez-Pérez et al., 2020, p.2)

To make functional software and to decrease the number and severity of bugs, software is evaluated and tested. During the testing, known bugs and out of specification issues are solved. Part of these concerning Boson have been reported inside a Jira project and were supposedly already solved in the .NET platform change. Others are expected to be found in testing.

4.1 Test plan and results

White-box testing is used to test the software internally, knowing its functions, expectations etc. Simulation features are left outside the testing. Tests targeting Meson use automated tests (Visual Studio Unit Testing). Other mainly dialog targeting tests need manual work.

Unit tests (Table 5) target functionality of software code at function level. Unit testing is done along the development. When new code and function is written, it should be tested to verify its function. Units to be tested are following:

- Parser, which extracts number data from data strings. Some data strings can have number embedded into text as part of header, which may cause confusion in the end results and is not needed.
- Database sender, that transfers data packets to InfluxDB. It is in Meson and called only from Boson.
- New dialogs and buttons in Boson. They need to mimic the functionality of their peers. E.g., background of input dialog should change into red in case of invalid value, just like indicated by similar older input dialog. This means old code is reused.

Table 5. Unit test evaluation

Subject	Software module	Description	Pass criteria	Result
Parser	Meson	Parser should correctly extract all numbers from data string.	All numbers in string should be extracted separately if anything else than .,- exist in them.	Pass
Database sender	Meson	Sender should send data to database.	All input parameters are used, data is sent and received correctly by database.	Pass
Dialogs and buttons	Boson	New elements serve database functionality.	Elements need to be functional and correct in their behaviour.	Pass

Integration test (Table 6) aims to verify the functionality of software interfaces. The integration test targets the interface between Boson and Meson. Minimum level being its functionality while performance is also of interest. Integration testing is timed after all unit testing is in satisfactory level. Satisfactory level is seen as the requirement to be able to start integration testing.

Table 6. Integration test evaluation

Subject	Software module	Description	Pass criteria	Result
Boson-Meson interface	Boson, Meson	Data flows between the modules	Data and parameters that leave one module, should arrive in the other intact.	Pass

System test (Table 7) verifies the whole software entity. In Boson and Meson system test they are tested together in their normal function. They should receive data and transmit it to database correctly. System testing is finally done to verify new software with all its pieces. It is followed by acceptance testing.

Table 7. System test evaluation

Subject		Pass criteria	Result
Software	Opens	Opens without extra messages or unexpected delay.	Pass
	Loads configuration	Logger and database related configuration are loaded correctly	Pass
	Saves configuration	Logger and database related configuration are saved correctly	Pass
	Closes	Closes without extra messages or unexpected delay.	Pass
Database	Can be disabled and enabled	Change applies and can be seen outside software	Pass
	Settings can be changed (IP, batch size, parser headers, interval)	Change applies and can be seen outside software	Pass
	Data is sent correctly	Change applies and can be seen outside software	Pass
	Data is received correctly	Change applies and can be seen outside software	Pass
	Works simultaneously with local log file	Change applies and can be seen outside software	Pass
Visualization	Received database data is visualized.	Visualization must reflect the real output of instrument.	Pass

Number of passed tests was 13 out of 13 tests that were run. These tests can be used as a standard test when software has been changed substantially in the future.

4.2 Acceptance testing

Evaluation and acceptance testing was done against the multiple criteria presented in Chapter 1.3. The criteria are *Software update* and following:

1. Usability
2. Available
3. Simple or familiar
4. Maintainable
5. Reliable

6. Profitable

Criteria were further specified to present them unambiguous in Table 8 below. The test results are further discussed in related subchapters.

Table 8. Acceptance testing criteria

Subject	Pass criteria	Result
Software update	Software update 0.0 was designed 0.1 was implemented 0.2 can send data to database	Pass
Usable	1.1 Configuration of database related functions are - Changeable - Loadable - Saveable - Frozen or greyed out when not used - Sane 1.2 Database destined data is shown in textbox	Pass Pass
Available	Software is available when 2.1 found in internal network by author 2.2 found in internal network by another employee 2.3 it is functional in Windows 10 2.4 it can be startup by both persons (see above 2.1, 2.2) and generally buttons and windows can be used	 Fail Fail Pass Pass
Simple or familiar	Software is simple and/or familiar if 3.1 it can be adapted by a legacy Boson user without help from outside the logger window (self-explaining) 3.2 it looks and acts the same as legacy Boson concerning logging	Pass Pass
Maintainability	Evaluation passes when 4.1 code is commented and/or documented on function level 4.2 code is understandable by a new program code reviewer 4.3 code is saved into version control system 4.4 chosen programming language(s) use is reasonably grounded	 Pass N/A Pass Pass
Reliability	5.1 Database data integrity is intact when data in textbox (1.6) is equal to database data 5.2 Database data is not lost in data transfer.	Pass N/A

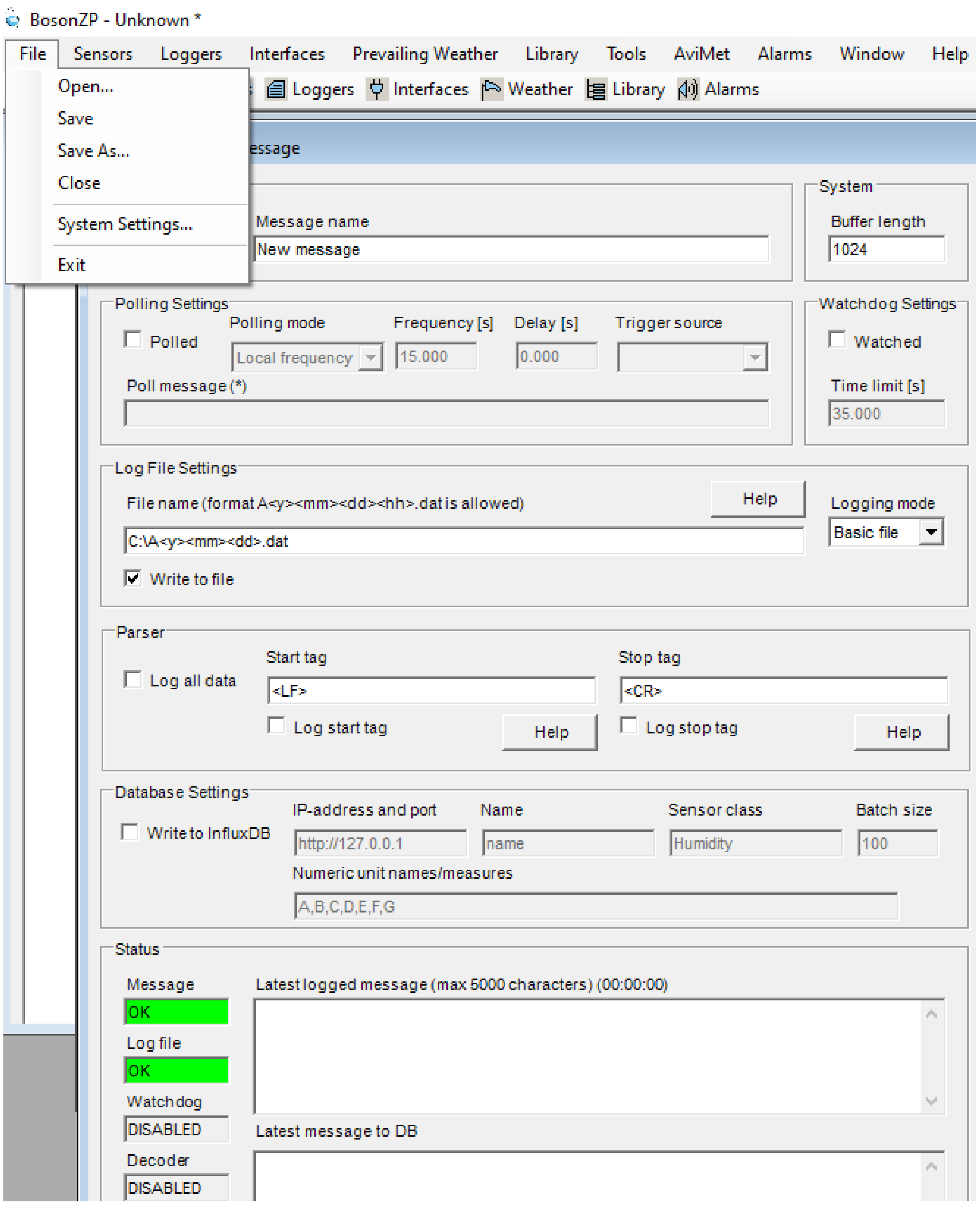
Subject	Pass criteria	Result
Profitability	Software is profitable when it saves time or other resources in data collection and/or analysis when compared with older methods.	Pass

4.2.1 Software update

Software update was designed and implemented like required by the main objective. The final software was able to send data into database like planned. C#.NET was used for database functionality and new .NET framework version was adapted for use. The resulting new software is programmed with VB.NET and C#.NET and consists of the main executable (VB), addon library (C#) and other files and libraries that needed to be included in the installer.

4.2.2 Usability

Usability tests consisted of clicking menus and buttons and seeing that the behaviour is correct, and it can be used in its purpose. Usability is concern since it can alienate users if not done well (Rajanen, 2021, p.1). Usability tests 1.1-1.6 were successfully done in BosonZP.NET (Figure 20).



System
Buffer length: 1024

Message name: New message

Polling Settings
☐ Polled
Polling mode: Local frequency
Frequency [s]: 15.000
Delay [s]: 0.000
Trigger source:
Poll message (*):

Watchdog Settings
☐ Watched
Time limit [s]: 35.000

Log File Settings
File name (format A<y><mm><dd><hh>.dat is allowed): C:\A<y><mm><dd>.dat
☑ Write to file
Logging mode: Basic file

Parser
☐ Log all data
Start tag: <LF>
☐ Log start tag
Stop tag: <CR>
☐ Log stop tag

Database Settings
☐ Write to InfluxDB
IP-address and port: http://127.0.0.1
Name: name
Sensor class: Humidity
Batch size: 100
Numeric unit names/measures: A,B,C,D,E,F,G

Status
Message: OK
Log file: OK
Watchdog: DISABLED
Decoder: DISABLED
Latest logged message (max 5000 characters) (00:00:00)
Latest message to DB

Figure 20. Dialog usability tests

4.2.3 Availability

For software to be available for installation, it needs to be shared and be installable. If installation is not very simple and familiar (self-explaining), instructions are needed.

Software installer was not available in internal network correctly in time, but it was tested functional and startup able in Windows 10 in multiple computers and by many users. Installer will become available and will be meanwhile shared using other methods.

4.2.4 Simplicity

Software needs to be simple and/or familiar. These properties drive the acceptance of a new software (Rajanen, 2021, p.3). Surely errors and learning endeavour will be less with also increased customer happiness (Rajanen, p.3).

Old Boson user tested BosonZP.NET and was able to set it for logging and database transfer correctly. One second interval data was plotted in Grafana for over 3 hours.

Updated Boson looked the same as old software, except the logger and interface windows. There should be no obstacles for old Boson users if they apply old logging practises. New options need to be introduced and documented, after which they are expected to become a standard also. One of these is the batch size, that is a new term in Boson. It can leave impression of unfunctional database transfer if it is not understood correctly. Traditionally in file logging the *latest logged message* box (Figure 21) updated with fresh data after each data retrieval interval period, yet in database transfer due to batch size, the *latest message to DB* box will update only after *batch size* number of messages have been collected, sent, and received from a function in Meson.

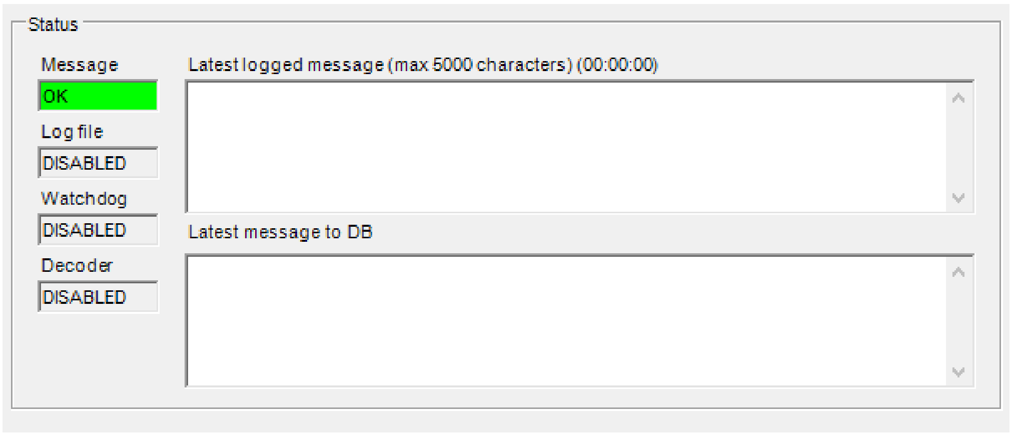

Status
Message: OK
Log file: DISABLED
Watchdog: DISABLED
Decoder: DISABLED
Latest logged message (max 5000 characters) (00:00:00)
Latest message to DB

Figure 21. Message status information

4.2.5 Maintainability

New code in C#.NET DLL was commented and function documented. Function documentation provides a detailed instruction about their use, easing it ("How to: Insert", 2023).

The code was not reviewed by software developer unlike planned. The short notice and priority of the review especially in summertime was not sufficient. However, this can be done later.

Vaisala internal GIT was used to store the source code of BosonZP.NET and related material. This allows code review and further development by other people as well. Foremost, the source code is safe also outside the developer's computer.

4.2.6 Reliability

Database data integrity was inspected in a short test. Numbers were sent in ASCII through RealTerm terminal software and Python script separately. Purpose was to see if textbox data is equal to the data shown in Grafana. RealTerm test using batch size 1 caused worst data availability and increasing it to 10 improved the availability. Inconsistency root cause was not solved, but suspect was on network issues, as similar behaviour was unseen at Test field data collection and in wind tunnel tests. VPN, WLAN were greatest differentiator in the network at the time of this test, whereas Test field data collection does not use VPN and applies LAN extensively.

Reliability concerning *data not being lost* was also evaluated for a longer time. Data availability and its integrity were evaluated for 82 days comparing BosonZP.NET readings to old Boson readings. BosonZP.NET used *batch size* of 15. There were 1968 lines of hourly air pressure readings. They were plotted with their difference (Figure 22).

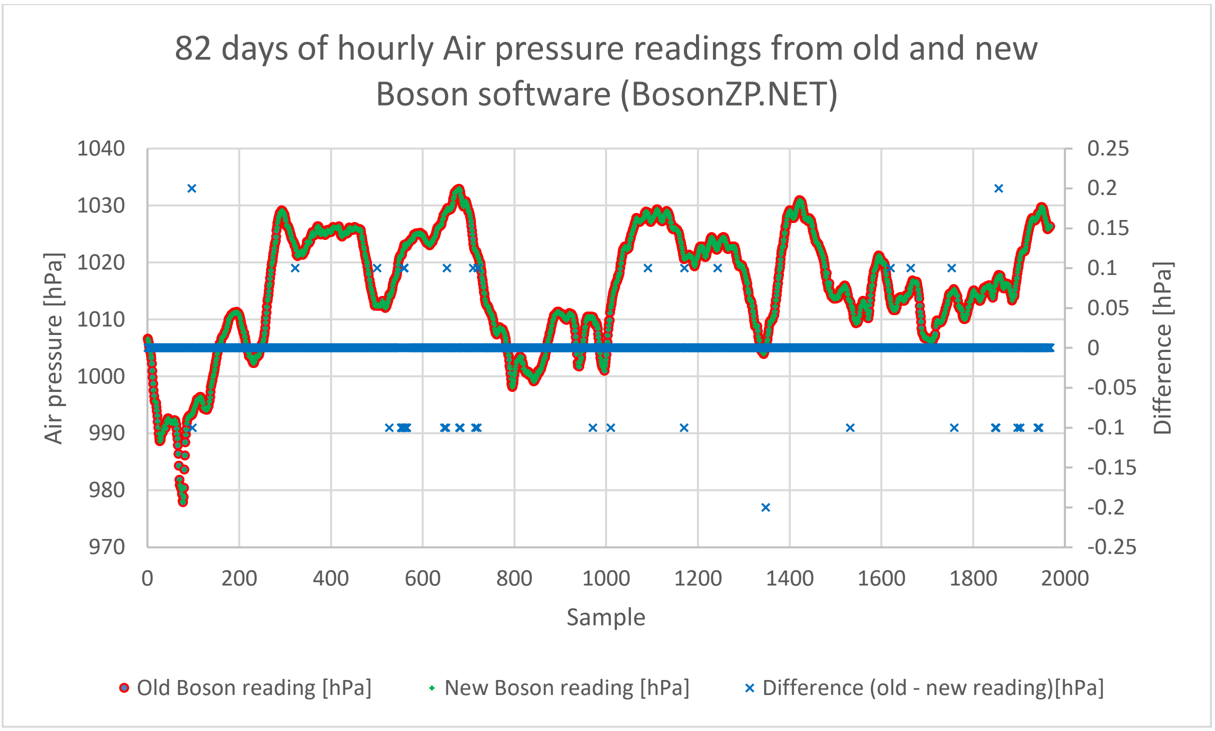


Figure 22. 82 days of Boson readings

Old and new Boson readings were very alike. Difference of maximum of 0.2 hPa was spotted only few times whereas 0.1 hPa difference was more common.

The densest time in readings with 0.1 hPa difference was examined further in Grafana and Excel. Data export from that chosen 4-hour period provided 5 second interval data. This dataset indicated that BosonZP.NET data is timestamped for earlier time than in the old Boson. This takes place constantly. One hour data is presented in Figure 23.

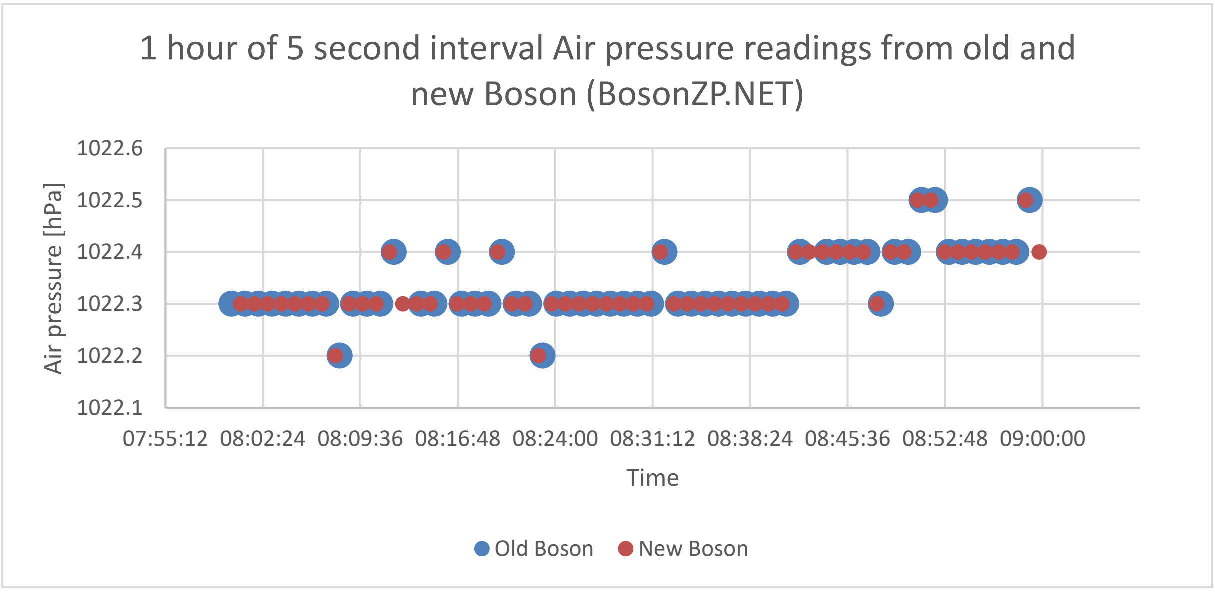


Figure 23. 1 hour data set with timestamp difference

Data availability appeared to be greater when BosonZP.NET was used. Two more 4-hour data sets from later time were inspected and all three are presented in Table 9.

Table 9. 4-hour data availability

Data set	Data availability improvement [%]
1st set at 14.4 8:00 – 12:00	1.67 %
2nd set at 14.4 14:00 - 18:00	-0.42 %
3rd set at 15.4 4:00 – 8:00	3.33 %

BosonZP.NET was also tested in test field for over 9 months of data collection with WXT-530 series weather transmitters, past 3 months with WMT700 wind sensors and in shorter than day lasting tests. Not every sample from every test was examined, but many data feeds were monitored daily. The monitored and randomly sampled data was all reasonable and no reliability issues were spotted.

These tests suggested that BosonZP.NET may be more reliable than old Boson with its database collection system, when used in ideal network environment, at e.g., test field, yet in other purposes its data may be unexpectedly lost when using small (< 10) batch sizes. Anyhow, batch size of 5000 was recommended by InfluxDB developers as the most optimal setting considering network usage ("Optimize writes", n.d.). When no instant data is needed, it makes sense, but using 1 second interval data would mean over 83 minutes delay before seeing the data in Grafana. That would not work for wind tunnel and other relatively shorts tests where instant data is desired.

4.2.7 Profitability

Profitability was evaluated on multiple categories (see Table 10). They were *Type testing*, *Common testing*, and *Other testing*. Type testing is equal to conformance testing and is used to determine if product complies with specifications. This is done after major software and/or hardware changes. Common testing means all other forms of testing. Often this is related to component or feature change or search for a reported issue. Other testing deals with other kind of tests or features that do not fit into other two categories. One of this kind is remote monitoring as it was introduced along with the new software in connection with Boson.

Table 10. Profitability subjects

Category	Subject	Description
Type testing	Data collection	Test is executed and data collected often following a standard.
	Sensor monitoring	Sensor data can be monitored against set limits in real time. Limits can be set in Grafana.
	Report	Report and documented proof are required. Format is predefined.
Common testing	Data collection	Test engineer decides the test and data collection more freely.
	Sensor monitoring	Monitoring can be sufficient with no need to investigate data logs. Testing may stop sometimes immediately when conditions are met.
	Report	Report is desired, but not required. Information and results are main interest.
Other testing	Remote monitoring and other features	Server based logging and visualization releases user to monitor measurements remotely.

Case study – Test field type test

Recently a type test for a sensor was setup to the test field so to be monitored in real outdoors environment for two weeks. Another sensor with a different software was installed nearby. Software performance was monitored for meeting certain conditions and especially data availability. Results were sent many times in a minute into InfluxDB and plotted in Grafana. Additionally, they were saved to local disk daily, making a log per day per sensor.

Old method of evaluation would have been to collect, parse, combine, calculate, plot, and analyse the data files. Sometimes this is still needed and preferred, e.g., in case of complex output. For this case, there would have been 28 data files with approximately 40 MB of size each. Even after filtering it heavily, each file would have been 5 – 10 MB with minimal required data of interest. Yet, this could have hidden some related and helpful data. Calculating the numerical data availability of just one measured quantity from both units and two weeks would have taken 2 hours or more time using conventional methods.

Yet using the BosonZP.NET provided means to monitor the data constantly on a dedicated display and online in Grafana. This brought the benefit of being able to spot anomalies on short notice or even immediately using Grafana alarms. Moreover, inspecting the 2 weeks of availability data visually was a task of 10 minutes. Numerical inspection is more precise than visual, but former is not always required.

Case study – Common tests in wind tunnel

Wind tunnel is used to test devices under blowing wind. The test units are set to a mount and wind would blow from a same direction. Test units could be turned and tilted so to test response to greater variety of changes. Both the units and the wind tunnel results would be collected and numerically monitored during the test.

Boson and Grafana were tested in common test scenario where new instrument firmware were iterated 1-2 times a day to see their data output in wind tunnel. Previously numerical results would have needed to be monitored visually constantly or after the test from a log. Yet, new software allowed visual monitoring with which data logs were not needed. Software was not final, so rapid testing and results were desired. New method allowed:

- Multitasking and less strenuous work, because there was no need to watch numbers constantly,
- immediate visual feedback on watched visualized numerical quantities and,
- connecting different quantities and understanding the interconnection between them.

In type testing scenario, where standardized test was run, test limits were set in Grafana, and data was configured into standard form. This meant Boson and Grafana were able to send, monitor and evaluate 1 second data from every second, so providing instant feedback if type tested unit was keeping with standard and specification or not. Traditionally, including this case also, numerical test results needed to be handled for the test report. However, unlike the report, instant monitoring is capable to provide rapid response in the quality control process. Figure 24 shows example of static horizontal lines that can be set to represent the specification limits. The lines will remain on the time series plot throughout the test unless hidden separately.

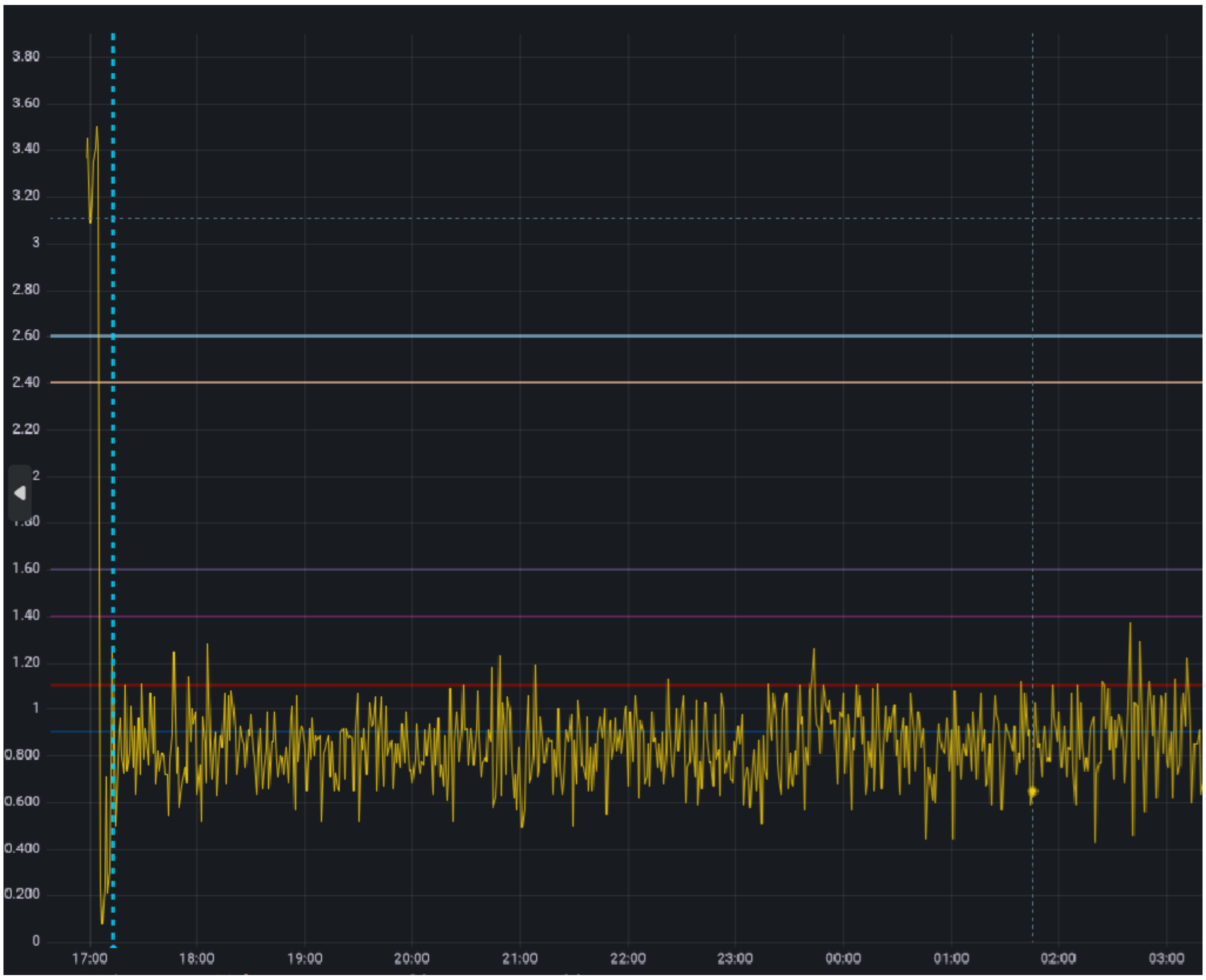


Figure 24. Wind speed (yellow) and some static lines

Yet due to implementation's design, it is possible (yet unseen in normal use), that not all data log data is correctly delivered and monitored in Grafana (Chapter 4.2.6). Not only is delivery receipt missing, but Grafana also experiences visual bugs changing the appearance and hiding some information (Figure 25). These restrictions should be noted in the current implementation. Its data availability can be roughly estimated from database by seeing if every second data is available. However, sometimes this can be misleading because at times the instruments do not send data expectedly.

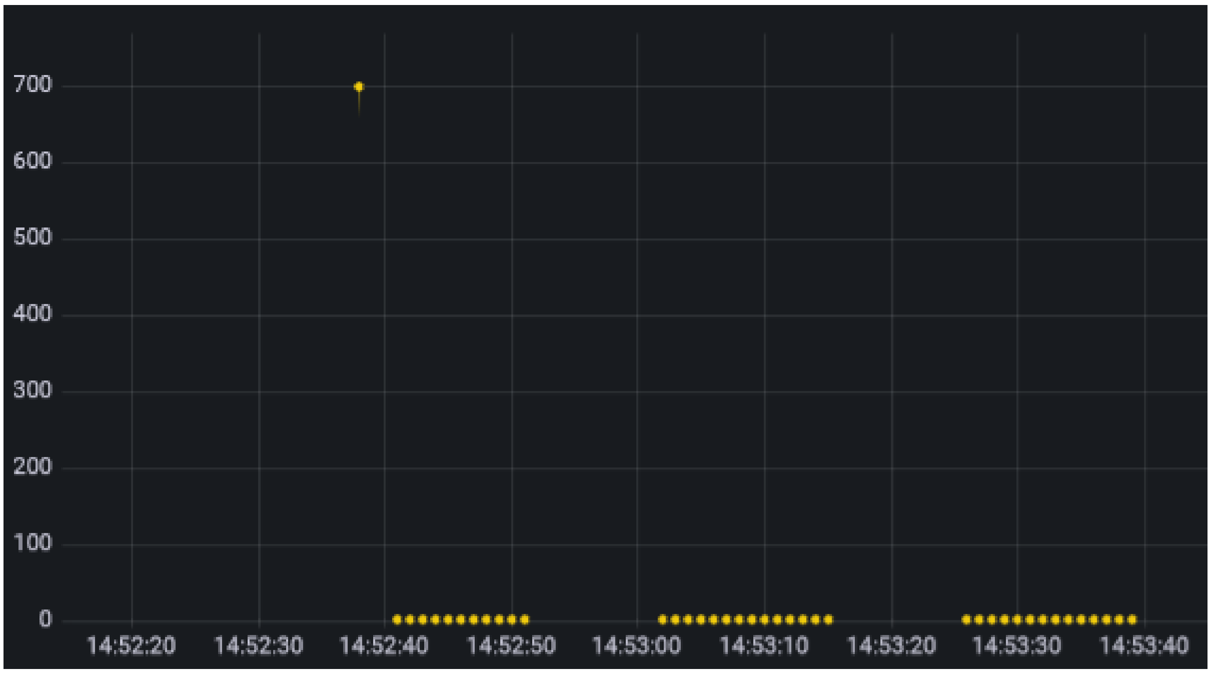


Figure 25. Grafana bug hiding many one second interval data points

Time saving in common test could be calculated from testing 10 firmware versions, each taking 45 minutes of time. After the test is finished, log files need to be connected by their timestamps, headers must be set properly, and all this examined in spreadsheets/statistics software. Then error is calculated and plotted along with manually set limits in same graph. These after-test steps can take 20-40 minutes of time. Total time for one firmware version test and results would be 65-85 minutes.

With database enabled Boson and Grafana, results against specification are visualized instantly for the whole test time. The after-test steps do not take place. Moreover, if anomaly is detected during monitoring, test can be interrupted.

From 10 firmware versions test with no interruptions, it can be estimated that saved time would be 200-400 minutes, that is over 3-6 hours. If interruptions were considered, time saving would be even more.

Time saving in type testing is ambiguous, as it is powerful in its visual and instant design, but not unquestionably reliable due to lack of delivery receipts. Yet, in type testing, reports are anyhow created (and needed) using numerical results from locally saved log files.

Case study – Field testing with remote monitoring

A company engineer was sent to remote location to test an instrument against a reference. Few days of testing produced large amounts of log files. The log files analysis required more than 8 hours of time from the analyst using conventional spreadsheets/statistics methods like mentioned in previous case study.

If new software had been in use, there would have been multiple benefits:

- Likely no need to plot and analyse the log files by hand, as the results would be visualized in Grafana both during the test and then on.
- Real time data feed into server allows multiple analysts to monitor data remotely, guiding or redirecting the testing process. This has potential to avoid making a new trip, because there is no perfect test plan that could adapt into every change and situation.

Case study – Remote monitoring and other features

Old Boson permitted local file logging, but new software introduced remote/local database logging. This permits user to monitor results from distance. Foremost this is useful in terms of safety. Some laboratories for example can have limited exposure time, meaning user should not stay there longer than a certain time to avoid damage.

Moreover, database use allows multiple data source connection and visualization, correlations can be looked for, specification limits set, analog output measurement values converted to correct unit and quantity.

Summary

If a test engineer operates tests and analyses the results for a whole week, the new software has potential to save 6-18 hours of work time, and its indirect total saved time and equivalent monetary saving can expand from hundreds to some thousand euros if no larger effects were at stake and 1-2 people were involved in the process.

If an engineer is sent to remote location for testing, then the trip cost and many other variables are added to the cost. It can range from 1000 € to multiple tens of thousands of euros cost. If factory or customer relationship was at stake, even more saving could be expected.

4.2.8 Result

The acceptance testing numerical evaluation was accomplished by having score scale from 0 to 10 for each criteria subject (Table 11). Each subject was considered by their subtasks and their fulfilments. Score 10 represents meeting the requirement fully.

Table 11. Software evaluation scores

Subject	Scores
Software update	10
Usability	10
Availability	10
Simplicity	10
Maintainability	8
Reliability	8
Profitability	10

Maintainability objectives were achieved except for the review. For this reason, 8/10 scores were given, as 3 of 4 objectives were met.

Reliability evaluation is not straight forward. In the purposed and planned use at the test field, reliability was not yet found to be questionable. However, in other uses its fragility was found as some data could not be found in Grafana, which also introduces buggy view at times. This could be summarized so that test field data collection reliability issue manifestation is possible but unseen.

The scoring makes it visible where the development is most needed in. Reliability and maintainability need more work to be on an acceptable level in all uses, not just at test field.

5 Conclusion

The BosonZP.NET software was originally designed to test field use. It was planned, that it will replace 14 years old Boson with database logging addition and send data directly to the database without need for other external solutions. Its development with simultaneous testing was done for the designed use. As it was being tested it proved to be so useful, that its use expanded into wind tunnel laboratory as well. Test field and wind tunnel use indicated its rapidness in having visualized results from instrument data.

However, because of limited view in possible use cases, the initial plan and tests, the software was not created fully reliable for other than the designed use. Reliability issues seemed to rise where the use case was away from the original design. These issues appeared during testing, before distributing the software internally. Even so, it was decided to make software shareable, because problems seemed to be limited into rare case, and solvable by using larger batch size.

The used database and Grafana visualization features were not novel at Vaisala test field, but the ease of their use was unseen. Earlier Python script use is not needed any longer. BosonZP.NET brings visualization of measurements with short latency to all. This takes away the time needed in producing understandable results.

The visualization in Grafana can relieve BosonZP.NET users from spreadsheets and statistical software dependency. When Grafana visualization is sufficient, there is no need for other software use. Also, in that case dependency is on external software developers. If Grafana is kept up to date, the benefits of new features are applied to BosonZP.NET data as well with negligible time use and cost.

On a larger company scale, it should be noted that BozonZP.NET is a software tool that is not shared outside the company. It serves foremost in instrument logging, testing and simulation. It does not go through same process as customer destined software does. This means the remarks of the software tool made in this thesis apply to similar kind of tools but not necessarily to other software, that has more disciplined process. Given this, the thesis highlights several things that concern software tools.

Firstly, important software should be kept up to date. Software should follow the trend of technology advancements when it is beneficial and productive. Logging should commonly focus more on Grafana, as it has become a standard for now. And on some subjects, e.g., machine learning and artificial intelligence should be increasingly studied if they are profitable and bring more productivity. If these and other potential gamechangers are not applied in time and in proper cases, company competitivity is at stake.

Secondly, maintainability is needed to keep the software serviceable. While software does not change over time, the needs around it do and software should adapt to reflect that fact, or it may come obsolete. In that case, new software is needed to fill the gap and it can prove to be more costly than upkeeping the old one.

Thirdly, early software engineering is crucial for successful software design. It is more fluid in case of new software, but in old software like in Boson, there may be need to test the capability of being programmed, compiled, and executed before making further planning.

Lastly, sufficient resources in time and schedule are mandatory for continuing development. Developing software once in two weeks or once a month is not efficient compared to dedicated sprint iteration where the work will be done until the end.

Many features were planned for BosonZP.NET software's future. They were split into key and wanted features to form a priority order. Key features are seen as next steps in the development after making it public internally. They are also listed in GIT.

Quality Assurance and Control

It is highly important to inspect that all data sent to database is either acknowledged or sent back to the sender. This is because it was found that the data is not always delivered like expected. Moreover, data line availability should be confirmed. That is, no e.g., 1 second data lines should be missing, if data send interval is 1 second. QC could be extended past these few mentioned subjects by applying analysis practises mentioned earlier (Campbell et al., 2013).

QC could be applied in Meson using C# producing test results and flags into the database. Both these and other data transmissions should be inspected for existence and integrity in the database. This means the check method needs to be applied widely, as a standard.

Modbus

During the development time a need for new feature was found. Modbus RTU is in use in newest instruments of Vaisala. The protocol is simple and ready-made examples of its usage exist (Herath et al., 2020). There proved to be some C# library available for it, however simplest code of one function was found to be enough in brief test. It was integrated into Meson, but not used as Boson needed to adapt changes also.

The following wanted features are seen important to do at some point if no key features surpass them. Wanted features would make the software more robust, maintainable, and usable.

Automatic updates

Automatic updates without auto-reboot would be beneficial to increase odds of keeping most distributions of the software up to date. E.g., Microsoft saw their update installation rate increasing from 5 % to 90 % due to automatic update adaption. (Vaniea & Rashidi, 2016, p. 2). Microsoft in fact provides auto-update method to Visual Studio .NET applications that is called ClickOnce ("ClickOnce security", 2023). The more automatic update is, the less user needs to be convinced about it. Yet, if user interaction is needed, user should be informed about new changes. This increases their willingness to install the updates. Moreover, new version must not break existing functionality of the device or software e.g., by consuming resources or breaking the executable. Functional way to mitigate this risk is to save the previous software as backup, that user can revert to. (Vaniea & Rashidi, 2016, p. 10).

The requirements

There were many requirements that were not used to criticize the software yet. They should be applied if software was to be used as a standard logging program. However, over specifying and too many requirements in the initiative development would not be helpful, delaying the project that was under time constraint and of which future was not certain.

Documentation

New features and functionality should be documented, especially concerning new users. Even if familiarity and easiness of use criteria were met, new users might struggle. For this reason, practical instructions should be provided.

Maintenance

Maintenance deals with multiple areas. Documentation, programming language, automatic updates are related. Most of all, automatic updates would be highly beneficial, because they enable constant progress and distribution of the software and its latest version. Also, code availability and description of development environment. Possibly in active development phase a virtual machine holding the development environment within, to serve multiple developers would make development swifter.

References

.NET Framework versions and dependencies. (2023). Retrieved June 10, 2023, from https://learn.microsoft.com/en-us/dotnet/framework/migration-guide/versions-and-dependencies

.NET Programming Languages. (2023). Retrieved June 10, 2023, from https://dotnet.microsoft.com/en-us/languages

Appendix B: DLL/TLB Library Setup. Retrieved July 28, 2023, from https://nsis.sourceforge.io/Docs/AppendixB.html

Application deployment prerequisites (Windows desktop). (2023). Retrieved July 3, 2023, from https://learn.microsoft.com/en-us/visualstudio/deployment/application-deployment-prerequisites?view=vs-2022

Burt, S. (2012). *The weather observer's handbook*. Cambridge University Press.

Campbell, J. L., Rustad, L. E., Porter, J. H., Taylor, J. R., Dereszynski, E. W., Shanley, J. B., … Boose, E. R. (2013). Quantity is Nothing without Quality: Automated QA/QC for Streaming Environmental Sensor Data. *Bioscience, 63(7), 574–585*. https://doi.org/10.1525/bio.2013.63.7.10

Chen, C., Härdle, W., & Unwin, A. (2007). *Handbook of data visualization*. Berlin, Springer.

ClickOnce security and deployment. (2023). Retrieved June 10, 2023, from https://docs.microsoft.com/en-us/visualstudio/deployment/clickonce-security-and-deployment?view=vs-2022

Data management unit DMU801. 2023. Retrieved June 17, 2023 from https://www.vaisala.com/en/products/data-management-unit-dmu801

de Haij, Marijn & Wauben, Wiel. (2010). *Investigations into the improvement of automated precipitation type observations at KNMI*. Retrieved July 29, 2023, from https://www.researchgate.net/profile/Marijn-De-Haij/publication/267838173_Investigations_into_the_improvement_of_automated_precipitation_type_observations_at_KNMI/links/605316a1458515e834521820/Investigations-into-the-improvement-of-automated-precipitation-type-observations-at-KNMI.pdf?origin=publication_detail

Fuentes, M., Vivar, M., Burgos, J. M., Aguilera, J., & Vacas, J. A. (2014). Design of an accurate, low-cost autonomous data logger for PV system monitoring using ArduinoTM that complies with IEC standards. *Solar Energy Materials and Solar Cells, 130, 529–543*. https://doi.org/10.1016/j.solmat.2014.08.008

Gilb, T., & Brodie, L. (2005). *Competitive Engineering - A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Oxford: Elsevier. https://doi.org/10.1016/B978-075066507-0/50005-2

Guide to Instruments and Methods of Observation: Volume I – Measurement of Meteorological Variables. (2021). World Meteorological Organization. Retrieved July 29, 2023 from https://library.wmo.int/doc_num.php?explnum_id=11612

Guide to Instruments and Methods of Observation: Volume V – Quality Assurance and Management of Observing systems. (2018). World Meteorological Organization. Retrieved July 29, 2023, from https://library.wmo.int/doc_num.php?explnum_id=9869#page=60

Harrison, R. G. (2015). *Meteorological measurement and instrumentation*. Chichester, England: Wiley Blackwell.

Herath, Ariyathunge, Priyankara. (2020). Development of a Data Acquisition and Monitoring System Based on MODBUS RTU Communication Protocol. *International Journal of Innovative Science and Research Technology. 5.* https://doi.org/10.38124/IJISRT20JUN479

How to: Insert XML comments for documentation generation. (2023). Retrieved July 20, 2023, from https://learn.microsoft.com/en-us/visualstudio/ide/reference/generate-xml-documentation-comments?view=vs-2022

HUMICAP Humidity and Temperature Probe HMP155. 2021. Retrieved June 11, 2023, from https://docs.vaisala.com/v/u/B210752EN-J/en-US

Kelso, J. E., Saulnier, W., Fritz, K. M., Nadeau, T., & Topping, B. (2023). The stream intermittency visualization dashboard: A web application for high-frequency logger data and daily flow observations. *Hydrological Processes, 37(2).* https://doi.org/10.1002/hyp.14809

Matthews, T., Perry, B., Khadka, A., Sherpa, T. G., Shrestha, D., Aryal, D., Tuldahar, S., Thapa, N., Pradhananga, N., Athans, P., Sherpa, D. Y., Guy, H., Seimon, A., Elmore, A., Li, K., & Alexiev, N. (2022). *Weather Observations Reach the Summit of Mount Everest. Bulletin of the American Meteorological Society, 103(12), E2827-E2835.* https://doi.org/10.1175/BAMS-D-22-0120.1

New R&D and innovation center. (2023). Retrieved June 10, 2023, from https://www.vaisala.com/en/vaisala-company/innovation-and-rd/rd-and-innovation-center

Observations for a better world. (2023). Retrieved June 10, 2023, from https://www.vaisala.com/en/vaisala-company/vaisala-brief

Optimize writes to InfluxDB. Retrieved July 29, 2023, from https://docs.influxdata.com/influxdb/v2.0/write-data/best-practices/optimize-writes/

Oussalah, M. C. (Ed.). (2014). *Software architecture 1*. John Wiley & Sons, Incorporated.

Rajanen, M. (2021). *Benefits of Usability and User Experience in Automated Driving*. Retrieved from https://ceur-ws.org/Vol-3028/D1-06-ESAAMM_2021_paper_9.pdf

Reference/RequestExecutionLevel. 2013. Retrieved July 28, 2023, from https://nsis.sourceforge.io/Reference/RequestExecutionLevel

Rodríguez-Pérez, G., Robles, G., Serebrenik, A. et al. (2020). How bugs are born: a model to identify how bugs are introduced in software components. *Empir Software Eng 25, 1294–1340*. https://doi.org/10.1007/s10664-019-09781-y

Tchier, F., Rabai, L. B. A., & Mili, A. (2015). Putting engineering into software engineering: Upholding software engineering principles in the classroom. *Computers in Human Behavior, 48, 245–254*. https://doi.org/10.1016/j.chb.2015.01.054

The Red Planet is in sight – FMI and Vaisala's scientific measurement instruments onboard Mars rover are ready for action. 2021. Retrieved June 17, 2023 from https://www.vaisala.com/en/press-releases/2021-02/red-planet-sight-fmi-and-vaisalas-scientific-measurement-instruments-onboard-mars-rover-are-ready-action

Tian, J., Yin, J., & Xiao, L. (2022). Software Requirements Engineer's Ability Assessment Method Based on Empirical Software Engineering. *Wireless Communications and Mobile Computing, 2022, 1–10.* https://doi.org/10.1155/2022/3617140

Vaisala RAINCAP® Technology. Retrieved July 25, 2023, from https://www.vaisala.com/en/vaisala-raincapr-technology

Vaniea, K., & Rashidi, Y. (2016). Tales of Software Updates: The process of updating software. *In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16). Association for Computing Machinery, New York, NY, USA, 3215–3226*. https://doi.org/10.1145/2858036.2858303

Wilkinson, F. (2019). *Adventure | perpetual planet.* Retrieved July 25, 2023, from https://www.nationalgeographic.com/adventure/article/mount-everest-highest-weather-station-perpetual-planet

WXT530 Series User Guide M211840EN-G. Retrieved 29, 2023, from https://docs.vaisala.com/r/M211840EN-G/en-US/GUID-C59284C5-DC79-41ED-B024-E7D0FD694D32