

# Project Report

## Automatic Logging of Environmental Events to a Centralised Server and Statistical Overview of Logs

by

**Göran Gustafsson** (gogu0001@student.hv.se)

2015-01-12

## **Project Report**

1. Project Description	3
2. Specification of System	3
2.1. EventLogger	5
2.2. EventServer	6
2.3. EventParser	7
3. Requirements	9
3.1. Hardware Requirements	9
3.2. Software Requirements	10
3.3. Development Requirements	11
4. Changes in Project	12
5. Development Method	13
6. Testing and Validation	14
8. Conclusion	16
8.1. Future Development	17

# 1. Project Description

The goal of this project is to create a system consisting of several different software components that together satisfies the following feature requirements:

1. Automatically log information when environmental changes occur.
2. Send and store the logged information at a centralised location.
3. Present the logged information in a simple and user-friendly way.

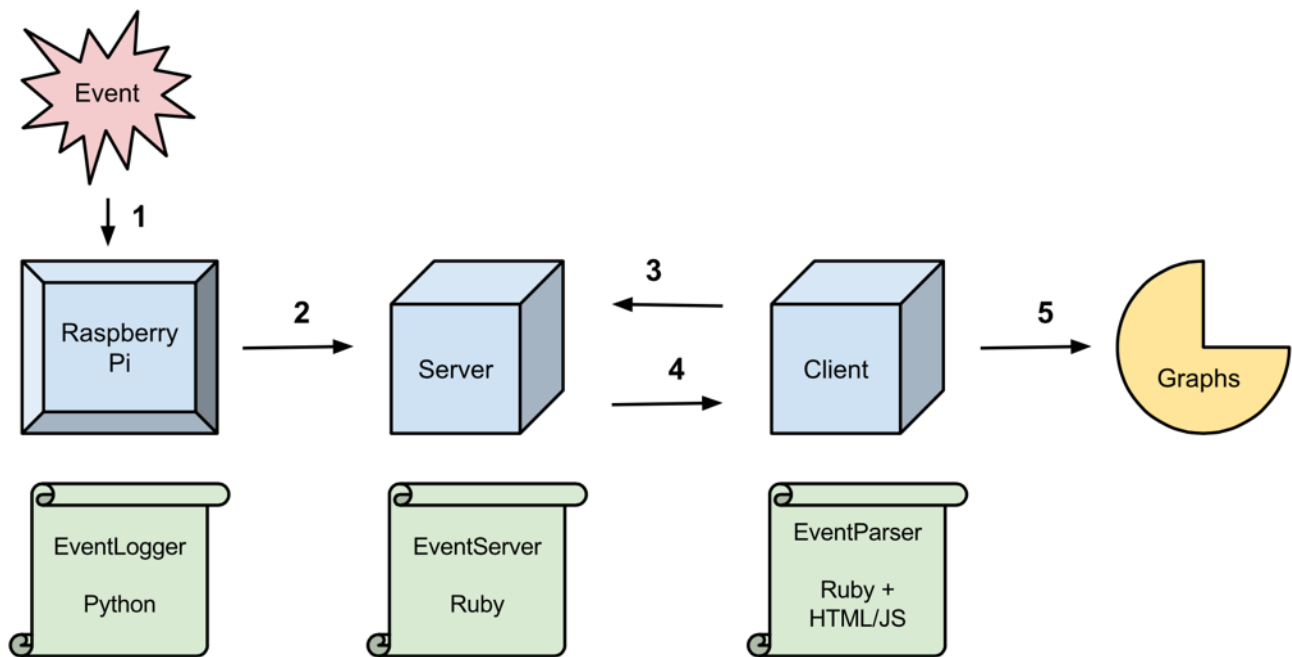
This project idea grew from a personal desire to be able to log information and turn said information into something that is easier to comprehend. I have tried using programs and services for logging before but most of the time you end up being forced to do manual logging, as in manually add information every single time something has happened. This is simply not good enough for me. The burden of the manual labour greatly outweighs the potential benefits of the information. Simply put, if the logging is not automatically performed then it has zero value.

The obstacle of manual logging has put me off on the data logging idea for a few years but I have never lost interest in it. Through this project I got the chance to once again revisit the idea and try to do something about the situation. The system created makes it possible and also easy for me to automatically log a wide range of different things, there are certain limitations of course but I am now one step closer to what I have wanted to have all these years.

## 2. Specification of System

Before going into detail about each software component that this system consists of it is best to start off with a short introduction to the system as a whole. It will be it easier to understand the different components when you have a vague idea about how the different pieces fit together first.

The easiest way to describe the whole system is to illustrate it with an image, detailed description of the usage flow can be found under it. The image below contains something that happens (Event), three hardware devices (Raspberry Pi, Server and Client), three software components (EventLogger, EventServer and EventParser) and an end goal (Graphs).



Normal usage flow of the system would look something like this:

Step	Description
1	Something in the physical environment changes the state of a sensor. For example if the sight between two points on a photosensor is blocked.
2	Information about the event is saved locally and also sent to a log server.
3	Client connects to the log server with a normal web browser without needing to specify anything other than the address and port number.
4	Server responds to the command that the web browser sends with presenting the whole log file.
5	The log file can then be parsed and the output is a web page containing simple statistical overview of logs.

An example of the information logged when events occur can be seen below.

Date	Time	Device ID	Optional Message
2014-12-14	18:45:05	1	Text
Raw form: "2014-12-14,18:45:05,1,Text"			

This information and the raw format of the information is what all of the software components work with (EventLogger, EventServer and EventParser). The field "Device ID" is a user chosen number that indicates which logging device the logged event comes from, several logging devices can be used simultaneously. The field "Optional Message" is used for logging of data that certain input device can supply, for example it might be used to store a temperature value from a thermometer.

## 2.1. EventLogger

The purpose of the EventLogger component is to check for status changes of input devices connected to GPIO pins at frequent intervals, log events locally and also send log messages over a network to the EventServer component. If connectivity issues are encountered for any reason then the log messages that was unsuccessfully sent will be not only be written down in the default local log file but also written down in a separate log file containing only unsent log messages. This is done to ensure that it is very unlikely that log messages will go missing and also make it easy for the user to manually merge them with the log file at the centralised location later on (EventServer).

EventLogger can send out the follow commands to EventServer:

Command	Description
LOG <MESSAGE>	Log message sent to the log server. The message itself uses the format specified earlier.
WARN <ID>	A warning message that tells the log server that there are unsent messages on the device. The device id of the current device is sent along with the message.

There are several settings for EventLogger that can be found and edited near the top of the source code. The variable names are descriptive and therefor I will not go into detail about them. Comments are added where it is appropriate to explain them further. See table below.

Settings Section
<pre>device_id = 1 # Only integers are allowed as id's. log_server = "172.20.10.2" log_port = 8080 log_file = "logfile.csv" log_file_unsent = "logfile_unsent.csv" check_interval = 0.5 # Wait between each check in seconds. tcp_timeout = 1 # When should TCP connections timeout?</pre>

The most user-demanding part of using this system is the event checking part of EventLogger. This code has to be customised by the user because there simply is no alternative. This system is developed with a focus on general use and the type of input device used are not dictated, neither are the rules for what a triggered event is. You can use whatever you want and check for whatever you want. With great power comes great responsibility. Having said that I took several measures for making it as easy as possible to get things working for new users.

To ease the creation of a custom made event check I clearly marked the location of where the custom code should exist, and the code I have used is still there and commented. See table below. The only thing a user needs to do is keep on using the event\_check() function name and run the log\_message() function when event is triggered, optional message can be saved by sending it to the function like this: log\_message("Text").

A very good short introduction to GPIO on the Raspberry Pi can be found here <http://www.raspberrypi.org/documentation/usage/gpio/> and documentation on using RPi.GPIO and Python can be found here <http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/>.

### Event Check Section

```
gpio_pin = 5
previous_state = 1 # HIGH is the default start state used.

GPIO.setmode(GPIO.BCM) # Use Broadcom board layout instead of BOARD.
GPIO.setup(gpio_pin, GPIO.IN) # Set pin _gpio_pin_ as input type.

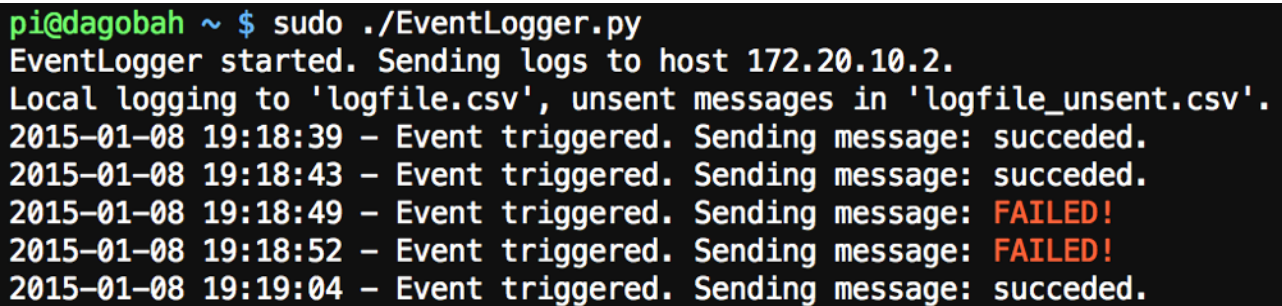
def event_check():
    global gpio_pin
    global previous_state

    current_state = GPIO.input(gpio_pin) # Save current state.
    log_message = "-" # Optional message sent with the log message.

    # Check if current state is HIGH and previous was LOW (triggered).
    if current_state == 1 and previous_state == 0:
        log_event(log_message)

    previous_state = current_state # Save current state for next run.
```

Once the event check function is finished it is easy to use EventLogger. Just run the command without any arguments. Note that it must (unfortunately) run as the user root because we need direct access to the memory device “/dev/mem” for RPi.GPIO. See screenshot below.



```
pi@dagobah ~ $ sudo ./EventLogger.py
EventLogger started. Sending logs to host 172.20.10.2.
Local logging to 'logfile.csv', unsent messages in 'logfile_unsent.csv'.
2015-01-08 19:18:39 - Event triggered. Sending message: succeeded.
2015-01-08 19:18:43 - Event triggered. Sending message: succeeded.
2015-01-08 19:18:49 - Event triggered. Sending message: FAILED!
2015-01-08 19:18:52 - Event triggered. Sending message: FAILED!
2015-01-08 19:19:04 - Event triggered. Sending message: succeeded.
```

The best and most convenient way to keep the program running is to execute it inside of a terminal multiplexor program like “tmux” or “screen”.

## 2.2. EventServer

The purpose of the EventServer component is to listen for incoming commands. Specifically log messages or warning messages from EventLogger, or requests for the whole log file from clients. This is basically the component that takes care of the centralised logging (log server) and it also acts as a web server to make it very easy for users to retrieve log files from client machines.

EventServer can handle the following incoming commands:

Command(s)	Description
LOG <MESSAGE>	Write down the message. The message comes directly after “LOG”.
WARN <ID>	Message that indicates that there are unsent messages on a device with the id <ID>. Print out warning about it to the user.
GET / HTTP* GET /index.html HTTP*	Present the whole log file using the HTTP protocol so it can not only be downloaded through a browser but also viewed.
Everything else	Completely ignored and connection closes as soon as possible.

EventServer will answer the allowed incoming commands the following ways:

Command	Description
ACK	Response sent out when receiving LOG or WARN commands.
HTTP/1.1 200 OK	When an acceptable GET command is retrieved (see table above) we answer with HTTP 200 and present the whole log file.
HTTP/1.1 404 Not Found	When an acceptable GET command is retrieved but the log file does NOT exist we answer with HTTP 404.

There are two settings for EventServer that can be found and edited near the top of the source code. The variable names are descriptive and therefore I will not go into detail about them. See table below.

Settings Section
<pre>log_port = 8080 log_file = "logfile.csv"</pre>

To start EventServer you just run the command without any arguments. See screenshot below.

```
Coruscant ~/Projects/Event Log System $ ./EventServer.rb
EventServer started on port 8080. Logging to 'logfile.csv'.
2015-01-08 19:19:04 - Warning received! Unreported message(s) exist on 1.
2015-01-08 19:19:04 - LOG 2015-01-08,19:19:04,1,None
2015-01-08 19:19:11 - LOG 2015-01-08,19:19:11,1,None
```

Just as with EventLogger the best and most convenient way to keep the program running is to execute it inside of a terminal multiplexor program like “tmux” or “screen”.

## 2.3. EventParser

The purpose of the EventParser component is to parse the log file(s) and extract interesting information and create a log report. The log report it creates is a web page containing statistics and various interactive bar graphs of all logged events.

EventParser reads and insert several external files into the log report web page it creates. All of the files are expected to be under the folder “EventParser-files” at the same location as EventParser itself. See table below.

File	Description
Chart.min.js	JavaScript library used for drawing up bar graphs.
External.css	Added for making it easy to add user-defined CSS settings.
External.js	Added for making it easy to add user-defined JavaScript code.

There are several settings for EventParser that can be found and edited near the top of the source code. The variable names are descriptive and therefor I will not go into detail about them. Comments are added to explain them further. See table below.

Settings Section
<pre>bar_graphs_latest = 1 # Show graphs for latest 24 hours and 31 days? bar_graphs_color1 = "#88c157" # Color for latest 24 hours and 31 days. bar_graphs_color2 = "#4086cc" # Color for everything else.</pre>

To start EventParser you just run the command with two arguments, the input file (log file) followed with the output file (log report). See screenshot below.

```
Coruscant ~/Projects/Event Log System $ ./EventParser.rb logfile.csv output.html
Parsing 'logfile.csv' and writing to 'output.html'...
Coruscant ~/Projects/Event Log System $
```

The log report is way to big to be included in this document but the first part of the log report can be seen in the screenshot below so you can get an idea of what the log report is all about. The full report contains short summarisation and bar graphs for the following:

- Latest 24 hours. (Optional)
- Latest 31 days. (Optional)
- Events per hour.
- Events per weekdays.
- Events per day of month.
- Events per month.



# Summarisation of log file 'logfile.csv'

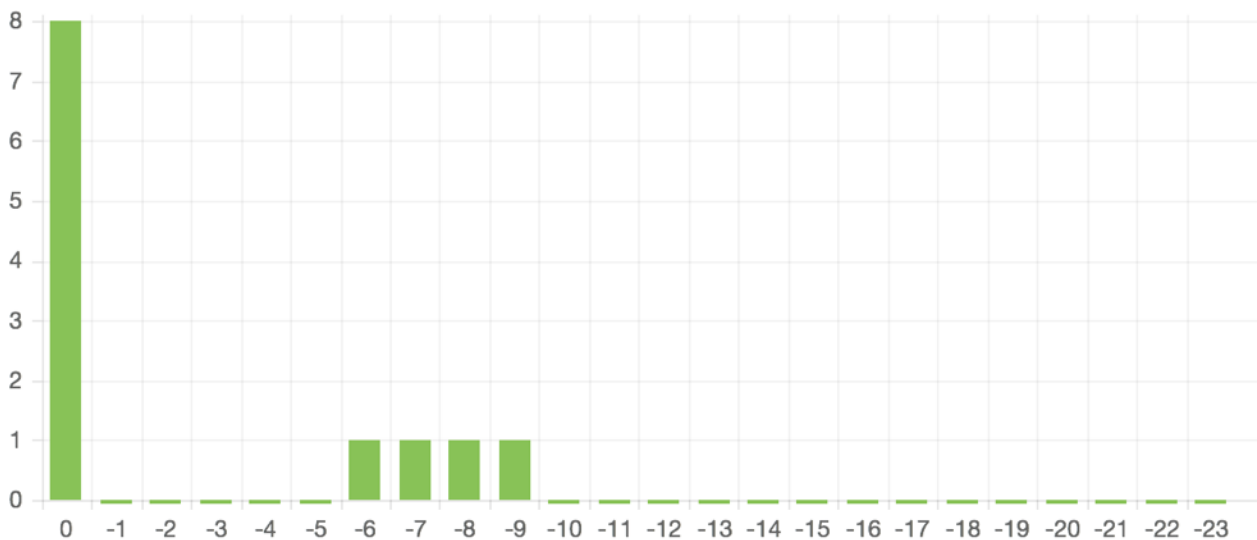
Log file parsed at **2015-01-08 19:13:08**

**152** logged events from **2** device(s)

- **97** events from device **1**
- **55** events from device **2**

Date ranges from **2013-01-15** to **2015-01-08**

## 1. Latest 24 hours



## 3. Requirements

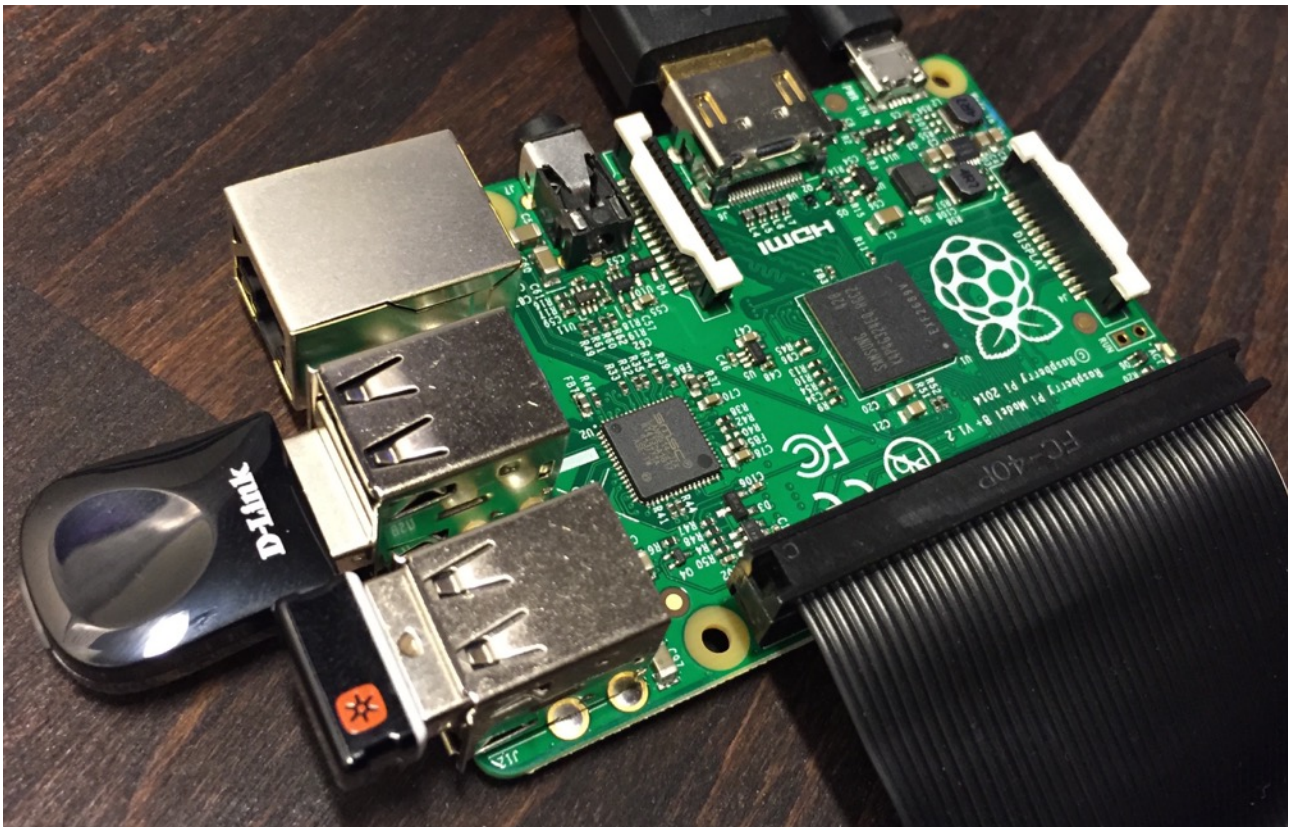
This section covers all of the hardware and software requirements for both developing and running the whole system. Hardware and software specifications for running the system, and hardware and software specifications related only to the development phase of the project.

### 3.1. Hardware Requirements

The following table contains information about all the hardware used both while developing and running the system.

Hardware	Optional?	Purpose
Raspberry Pi B+	No	Small, cheap computer with GPIO that is used for running EventLogger.
5V USB power adapter	No	Power source for the Raspberry Pi.
Micro-USB cable	No	Used with the USB power adapter.
2GB+ MicroSD card	No	Storage for Raspbian and log files.
Sensor with digital signal	No	Any type of sensor that can detect environmental changes. What kind is dictated by your own goals.
D-link DWA-131	Yes	Wi-Fi dongle that works well with the Raspberry Pi. Optional but very convenient and cheap.
Computer with network capabilities	No	Needed for development and running everything except for EventLogger.

Below is a picture of the Raspberry Pi used during the development phase of this project.



### 3.2. Software Requirements

The following table contains information about all the different software, programming languages and libraries used or required to run the system.

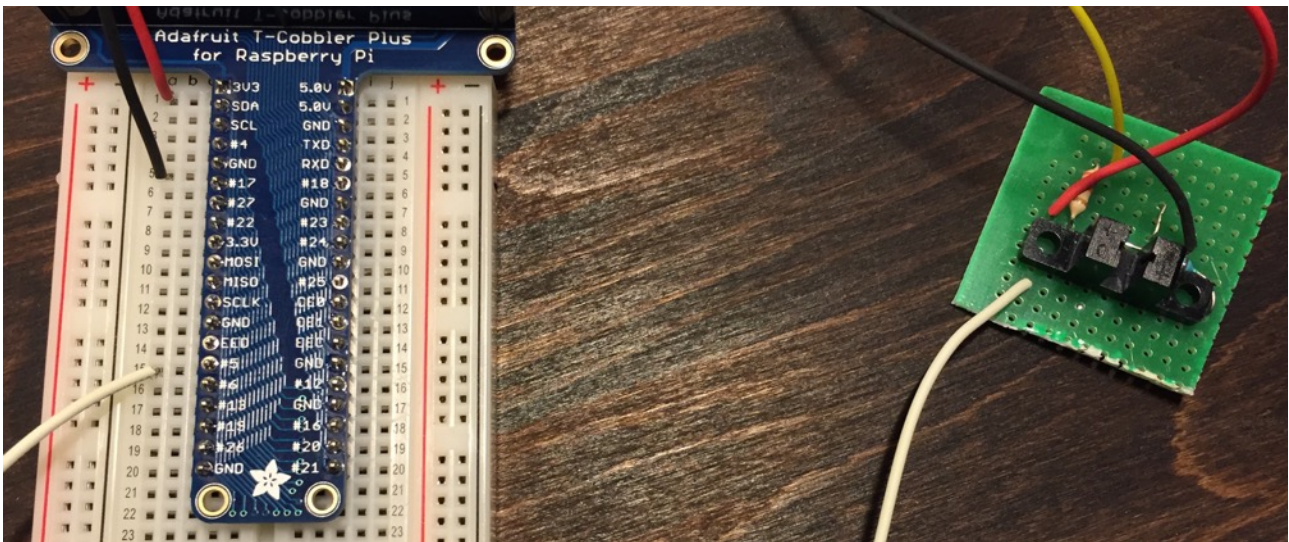
<b>Component</b>	<b>Software</b>	<b>Reason(s)</b>
Raspberry Pi	Raspbian	A very good Linux distribution specific for the Raspberry Pi. RPi.GPIO is installed by default.
EventLogger	Python 3 RPi.GPIO	The program is coded using the Python 3 language. RPi.GPIO is used for easy access to the GPIO pins on the Raspberry Pi.
EventServer	Ruby 2	The program is coded using the Ruby 2 language. No real reason other than because Ruby is really pleasant to work with.
EventParser	Ruby 2 Chart.js	The program is coded using the Ruby 2 language and the log report it creates uses a combination of HTML, CSS and JavaScript/Chart.js to create a good looking report with interactive bar graphs.
Computer with network capabilities	Unix-like operating system and modern web browser	EventServer and EventLogger is developed for and used on only Unix-like operating systems. A modern web browser is needed to view the log reports, JavaScript and canvas support needed.

### 3.3. Development Requirements

The following table contains information about all the different hardware and software used during the development phase. This in combination with everything specified earlier.

<b>Hardware / Software</b>	<b>Purpose</b>
Adafruit T-Cobbler Plus for Raspberry Pi + breadboard	Makes it convenient to use input devices. You get labels and easier access to pins. See picture below.
Soldering iron, cables, soldering lead etc	Needed for putting cables on to different input devices and putting together the T-Cobbler.
netcat	Used for sending and receiving text through TCP between programs. Mainly used when working on one component while another component was not completely finished.

Below is a picture of the breadboard, T-Cobbler and photosensor used during the development phase of this project.



## 4. Changes in Project

During the development of the various system components in this project there has been two major changes since after the project proposal and project plan was created. These changes took place because it was impossible to finish the project using the initial plan. All of the problems that I have encountered took up a lot of unnecessary time but the end results are of much higher quality simply because of these changes. Detailed description of the changes can be found below.

<b>Planned on Using</b>	Arduino Uno with Wi-Fi shield
<b>Ended Up Using</b>	Raspberry Pi B+ with Wi-Fi dongle
<b>Description</b>	The hardware which EventLogger runs on.
<b>Reasons for Change</b>	<ol style="list-style-type: none"> <li>1. MicroSD card on the Wi-Fi shield could not be used at the same time that the Wi-Fi functionality is used. There is one shared channel for data. Manual change of what uses the channel is possible but I did not try it out before encounter problem #2.</li> <li>2. The Arduino is unreliable and crashes at somewhat random times. The cause is most likely linked to the Wi-Fi driver which is known to have problems and the very limited amount of memory that the device has.</li> </ol>
<b>Positive Consequences</b>	<ol style="list-style-type: none"> <li>1. The Raspberry Pi has an underlying operating system that solves Wi-Fi connectivity problems that we would otherwise need to handle in the Arduino program manually.</li> <li>2. Any programming language that works on Linux can be used.</li> <li>3. The device can be used for several other things at the same time.</li> </ol>
<b>Negative Consequences</b>	The Raspberry Pi only have digital input/output. Input devices using analog signals cannot be used directly.

<b>Planned on Using</b>	MetricsGraphics.js
<b>Ended Up Using</b>	Chart.js
<b>Description</b>	The JavaScript library used for drawing up bar graphs in the log report web page created by EventParser.
<b>Reasons for Change</b>	<ol style="list-style-type: none"> <li>1. It almost worked but the bar graphs did not look 100% correct and there were several JavaScript errors indicated in the web browser that said it was trying to draw up boxes with negative width's.</li> <li>2. Could not find a solution to the problem. Bug in the library.</li> </ol>
<b>Positive Consequences</b>	<ol style="list-style-type: none"> <li>1. We get rid of three big JavaScript libraries. MetricGraphics.js relies on D3.js and that in turn relies on jQuery.</li> <li>2. Future potential problems will be much easier to debug because Chart.js is a small library.</li> <li>3. Bar graphs looks much better and no changes are necessary.</li> </ol>
<b>Negative Consequences</b>	None.

## 5. Development Method

When developing this system I did not start with looking up different methods other people, companies, organisations etc use while developing projects. Not because I do not think that it can give you valuable ideas (it can be quite the opposite) but rather because the methods I used for this project came natural, they kind of picked themselves. I ended up using an incremental and iterative development methodology and this is the main reasons why:

First, I frequently use this method while doing pretty much anything. Everything from school work to projects at home. It has served me well before and it feels like a good fit for projects of this size.

Secondly, this project is of a very experimental nature. The system covers areas of technologies that I have no prior knowledge to and the system is already divided up in several different components.

When I say incremental and iterative i mean dividing up something whole into several smaller parts, creating half-working parts rather early on and then constantly rotating my focus on the several parts. Basically do small improvements on all of the different components at an even pace. The nature of my system makes this a methodology a perfect fit because the system already have several components that are separate but still rely on each other to different extents, I often needed one working component to develop another component. It was also a very good fit because of the experimental nature, there were a lot of different things that could either go completely wrong or change during the development phase. Creating rough versions of each component early on made it easier to notice potential showstoppers and good/bad implementation ideas.

Components was prioritised differently, some was more important, some was need early and some was more inclined to encounter severe problems than others. These two factors made me prioritise the different components and focus on them in the following order:



1. Hardware.
2. EventLogger.
3. EventServer.
4. EventParser.
5. Documentation.

Because of the development methodology used and some big changes during the development phase I ended up working on the different parts the following order:

	Hardware	EventLogger	EventServer	EventParser	Documentation
1	X				
2		X			
3			X		
4		X	X		
5				X	
6	X	X			
7				X	
8		X	X	X	X

The table above is roughly consistent with the one provided in the project plan but not completely. Mainly step 6 and 7 was added because of the severe problems encountered with the Arduino Uno and MetricGraphics.js. Because of this and the deadline several different parts was still worked on simultaneously during the documentation phase of this project.

## 6. Testing and Validation

There are two different aspect of the testing and validation that I am going to cover in this section. First, does the finished system do what I initially planned? Does it really do everything I said that it would in the project proposal and project plan? Secondly, is the quality of the various system components satisfactory high? Is it reliable and actually usable in the real world?

To answer the first question, does the finished system do what I initially planned, I started off with looking through the old documents one more time to make sure that I do not miss any information about what I have said. The three major key functionalities and how to achieve them all can be seen summarised below.

Feature	How to Achieve It
1. Automatically log information when environmental changes occur.	Through hardware and software. Arduino, Wi-Fi connectivity, MicroSD storage, any kind of input device and EventLogger.
2. Store log information at a central location over a network.	Arduino system and software. The system will send log information over the network to another computer that is running software that writes down incoming messages.
3. Turn the logged information into simple and easy to understand statistics.	Through one piece of software. The software parse the log file(s) and creates a log report web page containing graphs etc.

The system in its current state does indeed have the major features 1, 2 and 3. The specification of how to achieve functionality 1, 2 and 3 is almost but not quite accurate because of some necessary changes performed during the project. Specifically the Arduino device is no longer used but since the Raspberry Pi is equal in all important aspects i conclude that the planned functionality and planned way of achieving said functionality is still accurate.

To answer the second question, is the quality of the various system components satisfactory high, I did four things. 1. Used the different system components intensely during the development phase which gave me a sense of how usable it is and it also frequently brought various bugs to my attention. 2. Wrote down a list of likely potential problems of the various system components and solved them. 3. Identified the most critical aspects of the system and took extra measures to solve them in a good way. 4. Printed out the source code and walked through each line one last time.

The table below contains a list of some potential problems that I think users are somewhat likely to encounter. It is hard to predict such things without missing smaller problems that only arise during certain unanticipated scenarios but it is worth it to keep on trying to think ahead none the less.

Component	Potential Problems
Hardware	<ul style="list-style-type: none"> <li>• Network connectivity can fault.</li> </ul>
EventLogger	<ul style="list-style-type: none"> <li>• Cannot connect to log server.</li> <li>• Connection is possible but no reply is sent back.</li> <li>• Slow response that hold up execution.</li> <li>• Program not running as user root.</li> </ul>
EventServer	<ul style="list-style-type: none"> <li>• Invalid commands are received.</li> <li>• Log file does not exist when HTTP GET is received.</li> </ul>
EventParser	<ul style="list-style-type: none"> <li>• Wrong amount of arguments used during execution.</li> <li>• Filename supplied as input argument does not exist.</li> <li>• The log file read is empty.</li> <li>• The log file contains lines with wrong formatting.</li> <li>• The log file contains fields with wrong kind of data.</li> <li>• External CSS file does not exist at expected location.</li> <li>• External JavaScript files does not exist at expected location.</li> </ul>

Most of the problems mentioned above are easily fixed (and they have been) by just adding various checks throughout the code and then handle errors gracefully. However, there is one big potential problem area that one can easily identify and that is network communication. Since reliability is so important for this system I added several initially unplanned features to EventLogger and EventServer to solve the problems. Specifically:

1. Extra logging of messages that failed to send to a separate log file in EventLogger.
2. Print warning message of unsuccessfully sent messages in EventLogger.
3. Added warning command in EventServer and made EventLogger use it to indicate missed logs.
4. Print warning message when warning commands are received. in EventServer.

These measures solve the potential problems to my satisfaction. This in combination with the fact that Raspbian can take care of getting networks up and running again after connectivity issues makes the system very reliable and actually usable.

All of the steps I have taken to test and validate the system components makes me feel confident that it does what it should and it does that in a good way.

## 8. Conclusion

The performed testing and validation of the system created confirms that it does what it was meant to do and it does so in a satisfactory way. This in combination with the fact that I have created something that is (at least to me) actually usable makes me conclude that this project has been a great success. The end result is not exactly what was initially planned but it is equal in all important aspects and even slightly better thanks to some unplanned functionality.

Even though I consider this project a success there were times when things looked quite bleak. Fortunately I was able to handle the problems encountered during the development phase but the whole project could just as easily have failed completely. Thanks to this I might think twice before



doing another experimental project. But then again, living close to the edge can be a much more rewarding experience and this is where I thrive.

## **8.1. Future Development**

This system is quite functional and reliable in its current state but it is still in a very early stage of development. I will continue using this at home and it will most likely evolve into something even better rather soon. Functionality wise I got no future plans at the moment but software rarely reaches the status of “finished”.

The source code of the system can be found on GitHub at this address:

<https://github.com/ggustafsson/Event-Log-System>