



Berufsbildende Schule des Landkreises Ahrweiler

Höhere Berufsfachschule IT-Systeme

HBFIT15

2015

Entwicklungs eines mobilen Standortbestimmungsgerätes für WiFi-Netzwerke und  
Programmierung in C++

Klaus Müller

Erik Berreßem

23.02.2017

# Inhaltsverzeichnis

<b>1</b>	<b>Vorwort</b>	<b>1</b>
1.1	Projektidee .....	1
<b>2</b>	<b>Projekt</b>	<b>2</b>
2.1	Planung .....	2
2.2	Durchführung .....	3
<b>3</b>	<b>Problemstellung</b>	<b>9</b>
<b>4</b>	<b>Zielsetzung</b>	<b>10</b>
<b>5</b>	<b>Kritische Reflexion</b>	<b>11</b>
5.1	Was war gut? .....	11
5.2	Was hätte anders sein können oder müssen? .....	11
5.3	Sind die Ziele erreicht worden? .....	11
<b>6</b>	<b>Quellenangaben</b>	<b>12</b>
<b>7</b>	<b>Anhang</b>	<b>13</b>
7.1	Projektantrag .....	13
7.2	Projektstrukturplan .....	14
7.3	Projektablaufplan .....	14

# 1 Vorwort

„Was genau ist eigentlich eine WarKitty?“ – Diese Frage wurde mir während der Projektbearbeitung oft gestellt. Der Begriff „WarKitty“ ist ein Portmanteauwort, also ein Wort welches aus zwei Wörtern entstanden ist. Nämlich „War“ für Wardriving und „Kitty“ für Katze. Was eine Katze ist wissen die meisten dann auch, dies kann man nicht für Wardriving behaupten.

„Wardriving ist das systematische Suchen nach Wireless Local Area Networks mit Hilfe eines Fahrzeugs. [...] Einige Wardriver sehen die drei Anfangsbuchstaben dabei als Backronym für 'Wireless Access Revolution'.“<sup>1</sup>

Wardriving ist also ein Begriff der das suchen und dokumentieren von WLAN / WiFi Netzwerken beschreibt.

## 1.1 Projektidee

„Warum Katzen?“ – Das traditionelle Wardriving erfolgt entweder per Auto oder zu Fuß. Während man fährt / geht sammelt man Daten über WiFi Netzwerke in der Umgebung und verknüpft diese mit GPS Daten. Ich habe mir vorgenommen diesen Prozess noch weiter zu automatisieren. Was ich mir dachte war „Was wäre wenn man nur noch die Daten einsammeln müsste?“.

Für dieses Vorhaben überlegte ich mir verschiedene Realisierungsmöglichkeiten: Drohnen, Hunde, Katzen.

---

1 <https://de.wikipedia.org/wiki/Wardriving>, (Download: 16.02.2017).

Ich entschied mich letztendlich für die Katze da sie, im Gegensatz zum Hund, eine höhere Wahrscheinlichkeit hat zurückzukommen, wenn man sie frei laufen lässt und da das Projekt so leichter zu realisieren ist als bei einer Drohne (Kostenfaktor).

## 2 Projekt

### 2.1 Planung

Ich hatte nun also eine grobe Planung für mein Projekt. Was ich mir nun überlegen musste war wie ich es umsetzen wollte. Ich entschied mich dafür Arduino basierte Hardware, Git Versionsverwaltung durch GitHub und die PlatformIO IDE für Atom (im Gegensatz zur Arduino IDE) zu verwenden. Das sind nun viele neue Wörter, welche ich nun erklären werde.

**Arduino** – ist eine Open-Source Elektronik Plattform die auf einfach zu benutzbarer Hardware und Software basiert. Es gibt viele Erweiterungen, sogenannte „Shields“, wie zum Beispiel SD-Karten Leser, GPS-Empfänger, usw. Arduino und Arduino ähnliche Produkte sind außerdem relativ günstig. Ich entschied mich für das sogenannte „NodeMCU v1“, ein WiFi-Development Board mit einem mir bereits bekannten ESP8266 Mikrocontroller und den „NEO-6M/7M“ GPS-Empfänger von WaveShare.

**Git** – „[...] ist eine freie Software zur verteilten Versionsverwaltung von Dateien, die durch Linus Torvalds initiiert wurde.“<sup>2</sup> Git hat grob zusammengefasst zwei sehr wichtige Funktionen: Das Verwalten verschiedener Versionen einer Datei

---

2 <https://de.wikipedia.org/wiki/Git>, (Download: 16.02.2017).

und die Möglichkeit sogenannte „Branches“, also Abzweigungen eines Projektes, zu erstellen die auch wieder zusammengeführt werden können. Versionsverwaltung wird von mit später nochmal genauer Erklärt.

**GitHub** – ist ein Webbasierter Hosting-Dienst für Git-Projekte<sup>3</sup>

**Atom** – ist ein von GitHub entwickelter Open-Source Texteditor der sehr erweiterbar ist. So gibt es zum Beispiel Syntaxhervorhebung für so gut wie alle Programmiersprachen.

**PlatformIO IDE** – „[...] ist eine mächtige Alternative zu der recht spärlich ausgestatteten Arduino IDE und ist gerade für größere Projekte eine sinnvolle Ergänzung. Durch die zahlreich unterstützten Plattformen ist zudem kein „Umgewöhnen“ in der Entwicklungsumgebung mehr nötig, wenn man oft die Entwicklungsplattform wechselt.“<sup>4</sup>

## 2.2 Durchführung

Man kann meine Durchführung in grob drei Abschnitte aufteilen: Informationsbeschaffung, Coding und Testing, wobei diese Abschnitte nicht linear sondern anachronistisch abliefen.

### Informationsbeschaffung

Da es bei meinen Projekt viel für mich neues Wissen zu erlangen hieß war das erste

---

3 Vgl. <https://de.wikipedia.org/wiki/Git>, (Download: 16.02.2017).

4 <https://alexbloggt.com/platformio-vorgestellt>, (Download: 16.02.2017).

was ich während der Durchführung tat mich zu informieren.

NMEA sentences, Beacon frames, Versionsverwaltung – All dies und noch mehr war neu für mich und es dauerte ein bisschen bis ich diese Begriffe, nicht nur durch Internetrecherchen sondern besonders auch durch trial-and-error, verstand. Wenn man nur eine relativ kurze Zeit von sechs Wochen für das Projekt hat ist es hilfreich gut zu Planen wie man sich informiert und über was man sich informiert, damit man nicht unnötig Zeit „Verschwendet“ dich über dinge zu informieren die man nicht wirklich in der Umsetzung des Projekts braucht.

„**NMEA 0183** (meist einfach NMEA genannt) ist ein Standard [...] für die Kommunikation zwischen GPS-Empfänger und PCs sowie mobilen Endgeräten [...]“<sup>5</sup>  
NMEA ist eine Abkürzung für **N**ational **M**arine **E**lectronics **A**ssociation, welche ihn entwickelt hat. So sieht ein NMEA-Datensatz aus:

```
$GPVTG,,T,,M,1.012,N,1.874,K,A*2B
$GPGGA,165719.00,50.562662,N,07.264540,E,1,09,0.88,76.3,M,46.9,M,,*67
$GPGSA,A,3,12,19,17,32,15,25,10,14,24,,,,,1.50,0.88,1.21*09
$GPGSV,3,1,11,02,08,123,,06,16,088,,10,06,268,24,12,79,264,35*76
$GPGSV,3,2,11,14,15,321,28,15,14,179,18,17,20,041,32,19,35,053,26*78
$GPGSV,3,3,11,24,70,127,25,25,35,252,24,32,34,304,19*40
$GPGLL,5033.76060,N,00715.87459,E,165719.00,A,A*62
$GPRMC,165720.00,A,5033.76089,N,00715.87455,E,0.644,,180217,,,,A*71
```

Ein NMEA-Datensatz besteht aus mehreren NMEA sentences, wobei das erste Wort eines Satzes den Datentyp angibt, der bestimmt wie der Rest des Satzes interpretiert werden soll. Diese sentences sind vollgepackt mit Informationen; so vieler, dass ich nicht auf alles eingehen kann. In folgenden werde ich zwei sentences dieses NMEA-Datensatzes auflösen.

---

5 [https://de.wikipedia.org/wiki/NMEA\\_0183](https://de.wikipedia.org/wiki/NMEA_0183), (Download: 18.02.2017).

Jeder sentence beginnt mit einem „\$“ um die sentences voneinander abzugrenzen. Das „GP“ gibt an von welchem Gerät die Informationen kommen. Anstelle von „GP“ für GPS-Empfänger könnte an dieser Stelle noch andere Kürzel stehen. „LC“ für Loran-C Empfänger (ein älteres Positionsbestimmungssystem), „OM“ für Omega Navigations Empfänger (ein älteres Radionavigationssystem; ausser Betrieb) und „II“ für Integrated Instrumentation (z.B. Autopiloten; AutoHelm Seataalk System) sind die gängigsten. Danach folgen die Informationen. Am Ende eines jeden sentence steht noch die Prüfsumme des Satzes, die sich aus den ASCII-Werten aller Zeichen berechnet und in hexadezimal dargestellt wird. Das Endgerät welches die Datensätze empfängt berechnet dann nochmal die Prüfsumme und vergleicht diese mit der vom GPS-Gerät berechneten. Unterscheiden diese sich, dann ist der Datensatz fehlerhaft.

\$GP`VTG`,`,``T`,`,``M`,`1.012`,`,``N`,`1.874`,`,``K`,`A`\*2B

<code>VTG</code>	-	<b>Geschwindigkeit und Richtung</b>	
<code>,T</code>	-	Kurs	[NULL grad]
<code>,M</code>	-	Kurs	[NULL mag]
<code>1.012,N</code>	-	Geschwindigkeit	[1,012 knoten]
<code>1.874,K</code>	-	Geschwindigkeit	[1,874 km/h]
<code>A</code>	-	Art der Bestimmung	[A = Autonom]

\$GP`GGA`,`165719.00`,`50.562662`,`,``N`,`07.264540`,`,``E`,`1`,`09`,`0.88`,`76.3`,`,``M`,`46.9`,`,``M`,`,`\*67

<code>GGA</code>	-	<b>Zeit, Position und Qualität</b>	
<code>165719.00</code>	-	Uhrzeit	[16:57:19 Uhr]
<code>50.562662,N</code>	-	Breitengrad	[50° 33' 45.5832" N]
<code>07.264540,E</code>	-	Längengrad	[7° 15' 52.344" E]
<code>1</code>	-	Qualität der Messung	[1 = GPS]
<code>09</code>	-	Anzahl der Satelliten	
<code>0.88</code>	-	HDOP (horizontale Genauigkeit)	
<code>76.3,M</code>	-	Höhe über den Meer	[76,3 meter]
<code>46.9,M</code>	-	Höhe Geoid minus Ellipsoid	[46,9 m.]

**Beacon Frames** enthalten alle Informationen über ein Netzwerk welches es aussendet. Sie werden periodisch ausgesendet um die Präsenz eines WiFi-Netzwerkes anzukündigen. Beacon frames bestehen aus einem Header, welcher die Quell MAC-Adresse enthält, einem Body und einer Blockprüfzeichenfolge (BPF) die den Frame terminiert. Die BPF funktioniert genau wie die Prüfsummen von NMEA-Datensätzen. Der Body enthält einen Zeitstempel, der zur Synchronisation der lokalen Uhren der Empfänger dient, den Beacon Intervall, der angibt in welchem Intervall die Frames gesendet werden (meist 100ms), sowie Informationen zur Leistungsfähigkeit. In diesem Feld wird die Art des Netzwerkes, zum Beispiel Ad-hoc oder Infrastruktur Netzwerk, mitgeteilt. Abgesehen von diesen Informationen kündigt es die Unterstützung für Polling, sowie Verschlüsselungs Details an. Weitere Felder im Body sind die SSID, die Unterstützten Raten, eine „Traffic indication map“, sowie „Frequency-hopping“, „Direct-Sequence“, „Contention-Free“ und „IBSS“-Parameter.

„Eine **Versionsverwaltung** ist ein System, das zur Erfassung von Änderungen an Dokumenten oder Dateien verwendet wird.“<sup>6</sup> Die Hauptaufgaben einer Versionsverwaltung sind Protokollierungen der Änderungen, Wiederherstellung von alten Ständen einzelner Dateien, Archivierung der einzelnen Stände eines Projektes, Koordinierung des gemeinsamen Zugriffs von mehreren Entwicklern auf die Dateien und Gleichzeitige Entwicklung mehrerer Entwicklungszweige (engl. Branches) eines Projektes<sup>6</sup>. Versionsverwaltung ist besonders dann wichtig, wenn man an einem größeren Projekt und / oder mit mehreren Entwicklern arbeitet. Eines der bekanntesten Versionsverwaltungsprogramme ist Git, „[...] eine freie Software zur verteilten Versionsverwaltung von Dateien, die durch Linus Torvalds (den Initiator sowie die treibende Kraft hinter dem Linux-Kernel) initiiert wurde.“<sup>7</sup> „GitHub ist ein webbasierter Online-Dienst, der Software-Entwicklungsprojekte auf seinen Servern bereitstellt [...]“<sup>8</sup>, welchen ich verwende. Wenn man ein neues Git Repository initialisieren möchte kann man das im Terminal machen indem man zum Ordner navigiert, den man initialisieren möchte und dann mithilfe von „git init“ das Git Repository erstellt. Änderungen lassen sich mit „git add <dateiname>“ oder mit

6 Vgl. <https://de.wikipedia.org/wiki/Versionsverwaltung>, (Download: 21.02.2017).

7 <https://de.wikipedia.org/wiki/Git>, (Download: 21.02.2017).

8 <https://de.wikipedia.org/wiki/GitHub>, (Download: 21.02.2017).

Wildcards wie „`git add *`“ dem Index hinzufügen und mit „`git commit -m`“ `<Commit-Nachricht>`“, wobei man `<Commit-Nachricht>` mit einer Nachricht über die Änderung ersetzt, bestätigen. Nun sind die Änderungen im Index der noch verändert werden kann. Um die Änderung festzumachen muss man dann noch „`git push origin <Branch>`“, wobei `<Branch>` ersetzt wird durch den Zweig auf den man pushen möchte (z.B. master, Dev), ausführen. All dies lässt sich auch mit der GitHub GUI erledigen die von GitHub entwickelt wurde und eine grafische Oberfläche für Git bietet.

## Coding

Um meinen Code zu Organisieren und leicht benutzbar zu machen hab ich ihn in eine Library verpackt und public Methoden geschrieben, die private Methoden ausführen. Ich hab eine C++ Library geschrieben, die mit „`#include <path/to/WarKitty.h>`“ importiert werden kann. Wenn man die Library dann importiert hat ist es ein leichtes sie zu bedienen:

Es gibt drei Konstruktoren:

```
„WarKitty( bool verbal, int update_rate )“,
„WarKitty( bool verbal )“ und
„WarKitty( int update_rate )“
```

Der erste Konstruktor nimmt zwei Argumente ein, eine boolesche Variable namens „verbal“ und eine integer Variable namens „update\_rate“. Wenn man für „verbal“ ein true angibt werden über die Serielle Schnittstelle Debug Logs mitgeteilt. Die „update\_rate“ bestimmt wie lange zwischen den updates des WiFi-Loggers gewartet werden soll. Falls man nur „verbal“ oder „update\_rate“ angeben möchte geht das dank der zwei anderen Konstruktoren auch, der nicht angegebene Wert fällt dann zum default zurück. Bei „verbal“ ist der default false und bei „update\_rate“ ist er 0. Ich hab außerdem noch einen unbenutzten Dekonstruktor bereitgestellt, einfach weil ich dachte, dass ich ihn nutzen würde was aber nicht der Fall war.

Ich hab außerdem noch drei public Methoden geschrieben:

```
„bool update( int MODE )“,
„void gps( String latitude, String longitude, TinyGPSTime time,
          TinyGPSDate date )“ und
„bool reset( void )“
```

Die update Methode nimmt ein integer Argument namens „MODE“ an. Je nachdem ob man eine zwei oder eine drei überreicht ändert sich die Funktionsweise dieser Methode. Anstatt einer Zahl kann man auch die von mir definierten Konstanten SCAN und VIEW überreichen. Überreicht man eine nicht gültige Zahl, also eine Zahl die weder eine zwei oder eine drei ist, so gibt die Methode ein false wieder ohne etwas zu machen. Ich hab die update Methode außerdem so geschrieben, dass sie den gewählten Modus automatisch initialisiert, egal ob sie das erste mal aufgerufen wird oder ob der Modus gewechselt wird. Sie gibt ein true bei Erfolg und ein false bei Misserfolg wieder.

Wählt man den Scan Modus wird erst mal gescheckt ob er initialisiert werden muss, falls dies der Fall ist wird das Filesystem und WiFi so initialisiert, dass das scannen beginnen kann, ist dies nicht nötig weil es bereits initialisiert wurde wird dieser Schritt übersprungen. Das scannen an sich erfolgt indem ich mithilfe der „ESP8266WiFi“ Library die Beacon Frames der Netzwerke lese und diese, zusammen mit den GPS-Daten, in einen JSON-Objekt in einer Datei im Filesystem speicher.

Ich hab das JSON-Datenformat aus dem Grunde genommen, weil es kompakt aber auch einfach lesbar ist. So sieht ein JSON-Objekt aus:

```
{"Name": "Georg", "Alter": 47, "Verheiratet": false, "Beruf":
null, "Kinder": [{"Name": "Lukas", "Alter": 19, "Schulabschluss":
"Realschule"}, {"Name": "Lisa", "Alter": 14, "Schulabschluss":
null}]}
```

Um diese JSON-Objekte zu generieren verwende ich eine Library namens „ArduinoJson“.

Wird ein WiFi-Netzwerk gefunden, das bereits im JSON-Objekt ist, also ein Netzwerk

das schon mal gefunden wurde, so werden die Daten geupdated.

Die gps Methode nimmt insgesamt vier Argumente an. „latitude“ und „longitude“, welche beide Strings sind, „time“, welches vom Datentyp ein TinyGPSTime Objekt ist und „date“, welches ein TinyGPSDate Object ist. Ich habe deshalb eine separate Methode für das abfangen der GPS Daten gemacht weil es den Nutzer so ermöglicht das GPS-Serial selber zu bestimmen.

Diese TinyGPS Datentypen stammen aus einer GPS Library namens „TinyGPS++“ die dazu da ist NMEA sentences zu interpretieren.

Man würde die Methode dann in etwa so benutzen:

```
// Zum einlesen der Daten
while ( gps_serial.available() ) gps.encode( gps_serial.read() );

warkitty.gps( (String)gps.location.lat(),
              (String)gps.location.lng(), gps.time, gps.date );
```

Die reset Methode resetet einfach nur die Datei in der die gesammelten Daten gespeichert werden. Sie gibt, genau wie die update Methode, ein true bei Erfolg und ein false bei Misserfolg wieder.

### 3 Problemstellung

Beim Projekt gab es auch mehrere Probleme. Ein Problem war, dass ich das SD-Karten Modul nicht benutzen konnte, da der Mikrocontroller nur eine Software Serielle Schnittstelle zuließ, die aber schon durch das GPS-Modul benutzt wurde. Die Hardware Serielle Schnittstelle konnte ich auch nicht benutzen, da sie bereits vom USB-Port benutzt wird und das Empfangen / Senden von Daten zu einen Feedback loop führte.

Ein weiteres Problem war, dass ich mehrfach Hardware schaden erlitt, wie zum Beispiel das GPS-Modul und der Mikrocontroller die mir beide durchbrannten; sowie

mein Laptop, dessen Festplatte kaputt ging, was zu einen Datenverlust des Codes führte, den ich an diesen Tag schrieb.

Die Lieferung der Hardware war auch problematisch, da sie lange dauerte.

## 4 Zielsetzung

Mein Ziel war es eine leicht zu bedienende Wardriving Library in C++ zu schreiben, die von allen ESP8266 Plattformen genutzt werden kann.

Siehe „Projektidee“:

„Warum Katzen?“ – Das traditionelle Wardriving erfolgt entweder per Auto oder zu Fuß. Während man fährt / geht sammelt man Daten über WiFi Netzwerke in der Umgebung und verknüpft diese mit GPS Daten. Ich habe mir vorgenommen diesen Prozess noch weiter zu automatisieren. Was ich mir dachte war „Was wäre wenn man nur noch die Daten einsammeln müsste?“.

Für dieses Vorhaben überlegte ich mir verschiedene Realisierungsmöglichkeiten: Drohnen, Hunde, Katzen.

Ich entschied mich letztendlich für die Katze da sie, im Gegensatz zum Hund, eine höhere Wahrscheinlichkeit hat zurückzukommen, wenn man sie frei laufen lässt und da das Projekt so leichter zu realisieren ist als bei einer Drohne (Kostenfaktor).

## **5 Kritische Reflexion**

Insgesamt bin ich zufrieden mit meinem Projekt. Ich hätte nur etwas mehr Zeit gebraucht.

### **5.1 Was war gut?**

Ich hab vieles dazu gelernt, was mir auch sehr Spaß gemacht hat.

### **5.2 Was hätte anders sein können oder müssen?**

Wie bereits erwähnt war mir die Zeit am Ende zu knapp. Ich hätte auch eine zweite Person gebrauchen können, die mir etwas an Arbeit abnimmt.

### **5.3 Sind die Ziele erreicht worden?**

Im großen und ganzen ja.

## 6 Quellenangaben

Verwendete Librarys:

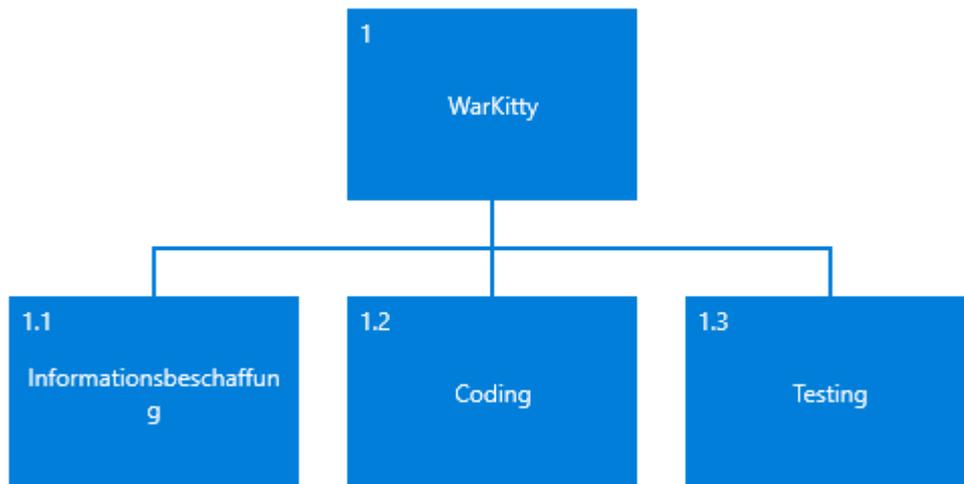
- ArduinoJson  
<https://github.com/bblanchon/ArduinoJson/>
- ESP8266WiFi  
<https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi>
- FS  
<https://github.com/esp8266/Arduino/blob/master/doc/filesystem.md>
- SoftwareSerial  
<https://www.arduino.cc/en/Reference/SoftwareSerial>
- TinyGPS++  
<http://arduiniiana.org/libraries/tinygpsplus/>

## 7 Anhang

### 7.1 Projektantrag

Betreuer:	Klaus Müller
Bearbeiter:	<a href="#">Erik Berreßem</a>
Projektziel:	<p>Wardiving ----- Wardriving ist das systematische Suchen nach Wireless Local Area Networks mit Hilfe eines Fahrzeugs. Der Begriff leitet sich von Wardialing ab, einer Methode, mittels Durchprobieren vieler Telefonnummern offene Modem-Zugänge zu finden.</p> <p>Katzen ----- Die Hauskatze ist eine Unterart der Wildkatze und deren Haustierform. Sie ist ein Fleischfresser und zählt zu den beliebtesten Heimtieren.</p> <p>Mein Ziel ist es ein Katzenhalsband zu entwickeln mit dem man wardriven kann, d.h. während die Katze umher streunt sammelt Sie mit der Hardware im Halsband Daten über die WiFi Netzwerke der Nachbarschaft</p>
Ressourcen:	<p>1 PC mit installierter Arduino IDE ( Mein Linux Netbook )</p> <p>1 Mikrocontroller mit WiFi Modul ( <a href="https://www.amazon.de/gp/product/B018E741G4/ref=oh_aui_detailpage_o00_s01?ie=UTF8&amp;psc=1">https://www.amazon.de/gp/product/B018E741G4/ref=oh_aui_detailpage_o00_s01?ie=UTF8&amp;psc=1</a> )</p> <p>1 GPS Modul ( <a href="https://www.amazon.de/gp/product/B00S4RLICU/ref=oh_aui_detailpage_o04_s00?ie=UTF8&amp;psc=1">https://www.amazon.de/gp/product/B00S4RLICU/ref=oh_aui_detailpage_o04_s00?ie=UTF8&amp;psc=1</a> )</p> <p>1 Micro SD Card Modul ( <a href="https://www.amazon.de/gp/product/B00QIKJP5W/ref=oh_aui_detailpage_o01_s00?ie=UTF8&amp;psc=1">https://www.amazon.de/gp/product/B00QIKJP5W/ref=oh_aui_detailpage_o01_s00?ie=UTF8&amp;psc=1</a> )</p> <p>1 Breadboard ( <a href="https://www.amazon.de/gp/product/B00QV7NXCS/ref=oh_aui_detailpage_o00_s00?ie=UTF8&amp;psc=1">https://www.amazon.de/gp/product/B00QV7NXCS/ref=oh_aui_detailpage_o00_s00?ie=UTF8&amp;psc=1</a> )</p>
Restriktionen:	<p>Es wird nicht versucht das Passwort zu knacken</p> <p>Es werden nicht mehr Informationen gesammelt als die, die das Beacon Frame enthält ( d.h. es werden zum Beispiel keine Informationen gesammelt über die Geräte, die mit dem WiFi Netzwerk verbunden sind )</p> <p>Es wird sich aus Sicherheit und Gesetzlichen Gründen nicht mit WiFi Netzwerken verbunden</p> <p>Ich werde, zum besseren Verständnis, einen Prototypen abgeben ( kein Halsband )</p>
Zeitplanung:	<p>Meilensteine ----- Code fertig</p> <p>Prototyp (Hardware) fertig</p> <p>Bericht fertig</p>
Status:	genehmigt
Log:	<p>05.12.2016 14:55:48 [hbfit1502] Projekt angelegt.</p> <p>10.01.2017 20:43:45 [hbfit1502] Projektantrag gestellt.</p> <p>10.01.2017 20:47:08 [muellk] Projekt genehmigt.</p>

## 7.2 Projektstrukturplan



## 7.3 Projektablaufplan

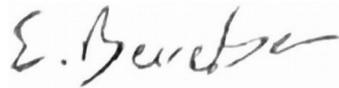
Phase	12.01.	22.02.
Informationsbeschaffung		
Coding		
Testing		

**Erklärung**

Ich versichere, dass ich die vorliegende Projektarbeit in allen Teilen selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe sowie dass alle wörtlichen und sinngemäßen Übernahmen aus anderen Quellen als solche kenntlich gemacht wurden.

23.02.2017 Remagen

\_\_\_\_\_  
(Ort, Datum)



\_\_\_\_\_  
(Unterschrift)