Faculty of Engineering, Computer Science and Psychology
Institute of Databases and Information Systems Ulm University

Design and Implementation of an Android Sleep Monitoring Framework

Submitted by
David Paul James Mohr

Verifier
Prof. Dr. Manfred Reichert

Supervisor
Dipl.-Inf. Marc Schickler

Bachelor thesis
2015

Abstract

Smartphones were originally mainly used for making phone calls and playing games, but as they become more powerful and are equipped with a wide variety of sensors new use cases become interesting. One of these use cases is sleep monitoring, which is interesting for many different research areas. The goal of this bachelor thesis is to develop a sleep monitoring framework for the Android platform which can be used easily by third party applications. The framework takes care of detecting sleep related events like snoring and movement as well as monitoring the ambient light during the night. Additionally, a demo application is developed to demonstrate the functionality of the framework and to highlight some best practices regarding Android background services as they are essential for monitoring sleep.

Contents

1 Introduction

As the amount of people owning a powerful mobile device grows, the urge to develop software which helps to handle the daily routine as well as improving it, increases as well. Almost every modern smartphone is equipped with a variety of sensors, ranging from cameras, light sensors and microphones to motion sensors and compasses. Thanks to the hardware and operating system developers it becomes increasingly easy to use those sensors in custom applications, thus helping users to better understand their body and helping them to live a more efficient and happy life.
Sleep is an essential part of our lives; when not being able to sleep well or long enough, we feel tired, are less productive and are likely to gain weight and as a result even develop major diseases like diabetes [16]. Moreover, we need the time at night to come to rest and to give our brains the possibility to process the things we experienced throughout the day. Additionally, problems or irregularities regarding sleep often originate from other diseases. Therefore, being able to monitor sleep without the need of a conventional sleep study could help in a wide variety of use cases.
Conventional sleep studies cost up to \$1000-\$5000 [20][15][19] whereas using a smartphone application only requires the user to download it once and remember to start and stop it in the evening and morning. As 80% of adults which are regularly online also own a smartphone, most participants already have their monitoring device at their fingertips [13]. Of course a smartphone application can't provide the detailed insights of an extensive sleep study but having sleep data of a wide variety of patients with very low cost is a huge benefit for sleep researchers.

1.1 Motivation

Creating a sleep monitoring application for Android^{TM} might seem easy at first sight but presents some difficulties at closer look. Simply reading audio data from the internal microphone or fetching light intensity is easy but interpreting it in the context of sleep monitoring is non-trivial. This thesis explores the possibilities of using Android device sensors for sleep monitoring and describes the development and usage of a sleep monitoring framework. The aim of the framework is to provide a high level API for accessing sleep related data and taking care of recording and saving the necessary data. Providing such a framework enables the development of a

wide variety of applications without them having to worry about audio and light interpretation, long running background services or CPU usage and power consumption. Therefore, the development of specific applications for specific use cases becomes only a matter of embedding the library and creating a user interface which might provide some additional use case specific functions. A simple implementation of the developed framework is also discussed, so the aforementioned mobile applications may use it as a starting point.
The framework is designed for very easy usage. Especially the storage and retrieval of the collected and interpreted data is very easy to adapt to third party applications, so they can use the framework while keeping their existing data storage components and being able to upload or export the data as needed.

1.2 Structure

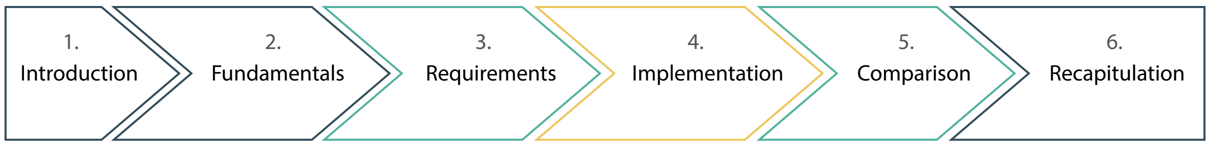


Figure 1: The structure of this thesis

Chapter 2 covers the fundamentals of light and audio interpretation from smartphone sensors. This chapter is still independent from the actual implementation and is intended as an overview of the selected methods and as preparation for chapter 4. In chapter 3 the target features of the framework and the demo application are described. In chapter 4 the actual implementation is discussed. Section 4.1 describes the functionality of the framework and explains the decisions made during development. Section 4.2 continues with the demo application and covers some best practices for using the monitoring framework. Chapter 5 discusses the achieved results in comparison with chapter 2. Finally, in chapter 6 the results are evaluated and some final thoughts are made on how to further improve the framework and where it could be used.

2 Fundamentals of Light and Audio Interpretation

In order to give insights into the reasons for choosing light and audio as markers for sleep state and quality, as well as how it is possible to extract sleep cycles out of an audio recording, both features are discussed in the following chapter.
Audio was chosen over direct device movement tracking because using the device's gyroscope for detecting movement would require the user to place the device into the bed. The better solution is to only use audio; then the device simply needs to be placed near the bed. Not having to place the device into the bed leads to a more natural sleep for many users. Additionally, most users won't have to change their habits as the smartphone is often used as an alarm clock, thus it is already placed near the bed.

2.1 Light Interpretation

The framework should not only record the sleep state but also other features which might affect the sleep quality. Therefore, the light intensity, measured in lux, is also recorded and interpreted. Light intensity is an important factor for sleep quality. Starting at about 100 lux, light affects the plasma melatonin concentration which is a widely used marker for the human circadian pacemaker [29, p. 695]. 100 lux already shifted the internal clock phase about 1 hour back, whereas a light intensity of about 1000 lux shifted the phase for -2 hours [29, p. 699]. Therefore, recording these light intensities is an important factor for determining sleep quality and comparing it to other nights.

2.2 Audio Interpretation

Audio interpretation is the core functionality of the monitoring framework. Compared to the light interpretation, much more work is involved here, because the only output Android provides is an array of numbers which indicate the audio amplitude. After receiving these values it is the task of the framework to interpret them as audio events.
On the one hand, the application should be able to detect snoring, which

is an indicator for deep sleep, and movement on the other hand. To understand why movement can be used to detect sleep cycles, human sleep stages and cycles are discussed first.
Human sleep consists of about 5 repeating cycles, each having 6 stages [3]. The stages are:

- Awake
- non-REM stages 1-4
- REM

REM stands for "rapid eye movement" and describes the phase in which dreams happen and the brain is relatively awake compared to the non-REM stages. While usually an EEG is used to determine sleep stages, smartphones obviously do not offer such technology. Luckily, it was discovered that a very close relationship between movement events and the REM stage exists [6]. The movement events seem to happen at the beginning and the end of each REM stage. Therefore, movement is perfect to detect the sleepcycles. Furthermore, it is important to distinguish snoring from movement as snoring is a very common, loud audio event which happens during sleep. Additionally, conventional snoring is unlikely to happen during the REM stage, so it is important to not confuse it with movement [4].
It was further found that three features, which can be computed out of the smartphone's recording, are sufficient to detect whether snoring, a movement or a miscellaneous noise occurred [9]. These features are the following:

- **RLH** The **r**atio of **l**ow frequency to **h**igh frequency
- **RMS** The **r**oot **m**ean **s**quare - The average loudness of the frame
- **VAR** The volume **var**iance of the frame

The recording is divided into 0.1 second intervals. Each interval will then be used to calculate all three features. These features can then be used to identify the event which occurred. Sample recordings are shown in figures 2 and 3. It was found that the RLH feature is very suitable to distinguish snoring from moving as movement consists of almost equal parts of high and low frequencies, whereas snoring consists mainly out of low band frequencies. Therefore, the rise of RLH can be used effectively to identify snoring [9]. Combined with the other features it is possible to extract movement events as well; this is further discussed in section 4.1.2.

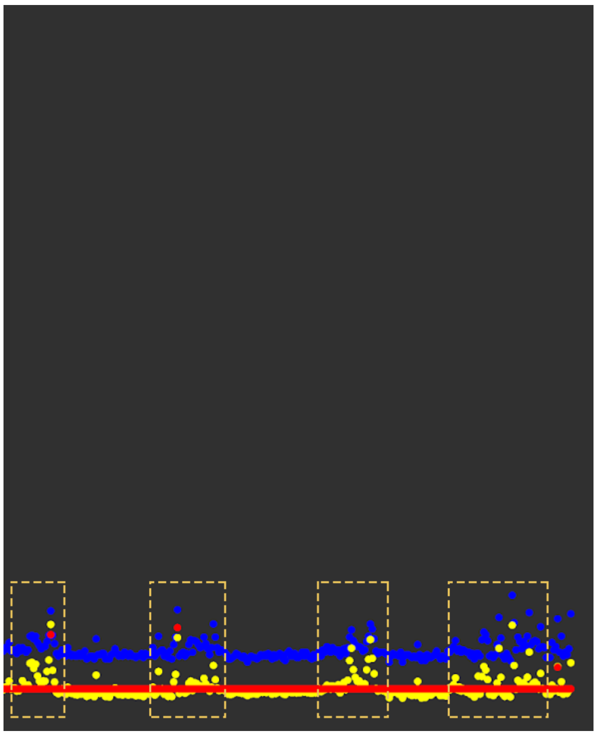


Figure 2: Feature extraction for movement events (RLH: red, RMS: blue, VAR: yellow)

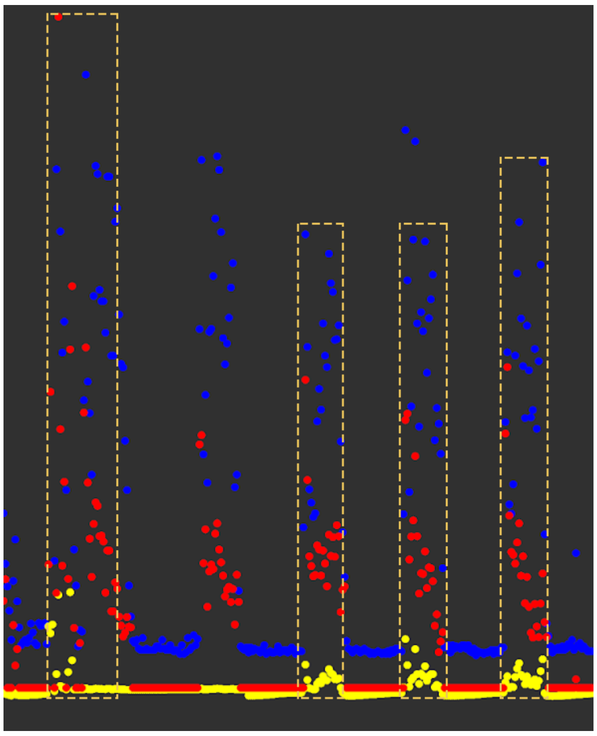


Figure 3: Feature extraction for snoring events (RLH: red, RMS: blue, VAR: yellow)

Figure 2 shows a 30 second interval of movement (audio recording from [5]), figure 3 shows a 30 second interval of snoring (audio recording from [7]). The framed parts are identified as movement and snoring events respectively. RLH (shown in red) is the perfect feature to distinguish snoring from movement as the RLH feature stays at zero for the complete movement interval whereas it kicks very high during snore events.
One further optimization is very essential for the framework. Audio characterization needs to be possible on a wide variety of devices and moreover, a wide variety of microphones. Additionally, Android sensors can be quite unreliable [23]. Therefore, the VAR feature must be normalized to ensure that the specific loudness of the microphone doesn't change the thresholds which will be used to interpret the audio frame:

$$normalized(x) = \frac{VAR_x - mean(VAR)}{std(VAR)} \tag{1}$$

This also has the effect that background noise will be filtered automatically. Noises like cars, trains, and microphone related static noise won't effect the normalized variance, so it can effectively be used to detect interesting frames. The normalization utilizes the last 100 frames. 100 frames turned out to be a good value as a longer period would have the problem of not being able to react to occurring external events like an air conditioning turning on. A shorter period would be problematic because the snore or movement event itself would have a too large influence, thus rendering the normalization useless.

3 Requirements Analysis

In this chapter the requirements for the framework and the demo application are discussed. Firstly, the functional requirements are listed and afterwards some important non-functional requirements.

3.1 Functional Requirements

The functional requirements describe the features of the framework and the demo application.

3.1.1 Framework

The following table lists all functional requirements for the framework.

Table 1: Functional requirements for the framework

	Requirement	**Explanation**
FR # 1	The framework should detect and record movement during sleep	Movement is the main indicator for light sleep phases.
FR # 2	The framework should record light intensity changes	Light intensity is an important factor for sleep quality.
FR # 3	The framework should detect and record snoring during sleep.	Snoring must be distinguished from movement as snoring is an indicator for deep sleep phases.
FR # 4	The framework should record the mean audio volume per frame.	The overall audio volume is a secondary factor for sleep quality.

Continued on next page

Table 1 – *Continued from previous page*

	Requirement	**Explanation**
FR # 5	The framework should offer a functionality to rate the sleep quality based on a recording.	This is a convenient feature for third party applications so they have a basic method of visualizing the recorded data.
FR # 6	The framework should offer a customizable data storage system.	To embed the framework into third party applications it is important to be adaptable to other storage systems.
FR # 7	The Framework should write the accumulated data to the output handler in regular intervals.	As it is not guaranteed that the background service is never terminated it is important to not only write the data at the end of the recording but in regular intervals so only little data is in danger to be lost at any time.

3.1.2 Demo Application

The following table lists all functional requirements for the demo application.

Table 2: Functional requirements for the demo application

	Requirement	**Explanation**
FR #8	The demo application should offer the user a detail view for each recording.	It is important that the user can see what data has been recorded and might be transferred to a third party organization. Also it is interesting for the user to evaluate the recorded nights.
FR #9	The demo application should offer a share functionality for the recordings.	Sending the recordings to study leaders is important. A share button should be used as this is a known concept for the user.

3.2 Non-Functional Requirements

The non-functional requirements mainly describe the relevant qualities of the framework and the demo application regarding efficiency and usability.

3.2.1 Framework

The following table lists the non-functional requirements for the framework.

Table 3: Non-functional requirements for the framework

	Requirement	**Explanation**
NFR #1	The framework should use the CPU as little as possible.	The goal should be to record 8 hours of sleep without having to connect the phone to the power supply.
NFR #2	The framework should handle all non-critical errors silently.	Simply skipping frames when an error ocurrs is the best solution, because some missing frames won't affect the overall sleep analysis. Stopping the application because of an error and notifying the user — like usual applications would — is not helpful in this case.
NFR #3	The framework should restart the recording after critical errors.	Recording as much as possible is the primary goal. As android background services are not guaranteed to be kept alive, restarting the recording service after a shutdown is essential.
NFR #4	The framework should be memory efficient.	In place algorithms should be used where possible as the available amount of memory is very limited on some devices and a lot of data needs to be processed.

3.2.2 Demo Application

The following table lists the non-functional requirements for the demo application.

Table 4: Non-functional requirements for the demo application

	Requirement	**Explanation**
NFR #5	The demo application should always inform the user about the current recording state	Showing the user when the application is recording and when not is crucial for a healthy relationship with the user.
NFR #6	The demo application should be easily usable in dark environments	Usually, the user interacts with the application during little environmental light. Therefore, the application should only use dark colors while still emphasizing the main start / stop button.
NFR #7	The demo application should respond fast to user input	As sleep monitoring applications are usually used by users to participate in studies, fast response times are very important so the time spent using the application is as short as possible.
NFR #8	The demo application should be usable on as many devices as possible	As preferably as much users as possible should be able to participate in a study, the application should not make unnecessary requirements to the Android version installed on the system or the availability of specific hardware features.

4 Implementation

Both, the framework and the demo application have been programmed in Java. It would be possible to use non-native technology, namely web technology, but using Java ensures best performance and sensor compatibility [25]. All code has been written with Android Studio. Android Studio is a very good choice because it has all the tools available to develop a library separated from an Android application while still using them in a single project. It also provides very convenient methods to create many different virtual devices and test the application on them. The application was mainly developed and tested on a real Nexus 6 device but a lot of different virtual Android devices have been used to test the application on the most important Android versions between API level 10 and 21.
Figure 4 shows how the framework, the demo application and the sensors work together when the application is recording. All hardware interaction as well as the interpretation of the audio signals is done by the framework and is then saved by the demo application to the device's storage. Due to the later discussed *OutputHandler* interface, the data could as well be saved directly to an external web service.

4.1 Sleep Monitoring Framework

Figure 5 presents the main module — the recording cycle. The figure shows the progress when the recording is started, how the accumulated data is queried and finally written to the *OutputHandler*.
At this point the overall concept of the recording cycle is described. The light and audio recorders are discussed in further detail in sections 4.1.1 and 4.1.2.
The cycle is started by creating a *Recorder* object and calling start() on it. The *Recorder* then initialises the light and audio recorders.
The *AudioRecorder* works by creating an *AudioRecord* object which then continuously writes the recorded data back to the *AudioRecorder* in 0.1 second intervals. The Audio recorder takes care of updating the *Feature-*

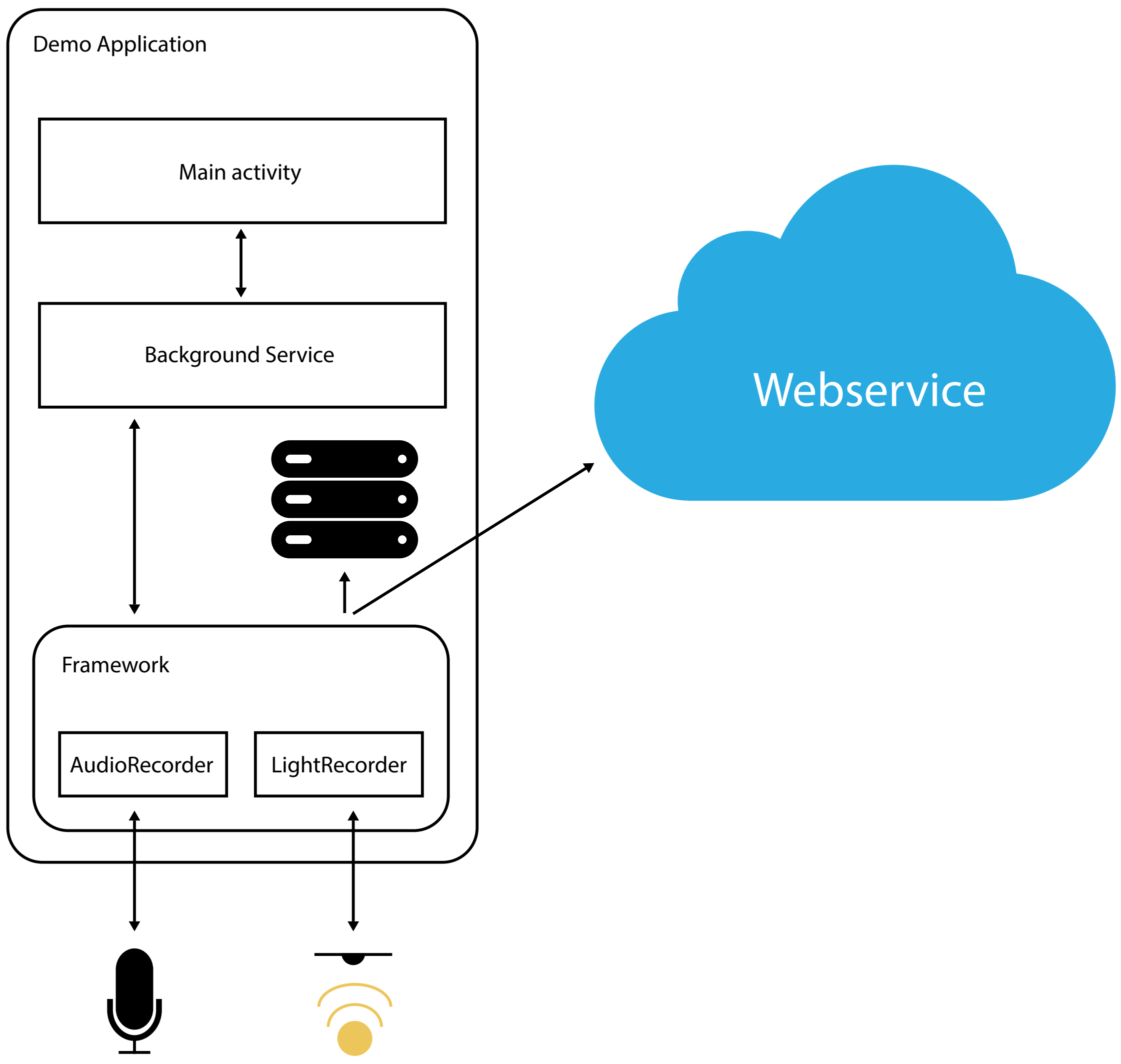


Figure 4: The architecture of the framework and the demo application

Extractor which in turn updates the *NoiseModel*. The *FeatureExtractor* and *NoiseModel* objects take care of extracting the audio events and are further discussed in section 4.1.2.
The *LightRecorder* simply hooks into the *LightSensor* from the device and stores the current light intensity in lux.
The actual cycle begins with the *Recorder* starting a 5 second interval, querying the data from the light and audio recorders every time. About every 15 minutes the data is handed over to the *OutputHandler* which takes care of persisting it. The entire cycle is started in a separate thread, otherwise the user interface would be frozen when the cycle is running. This is caused by the *AudioRecord's* methods which are blocking the CPU during recording.

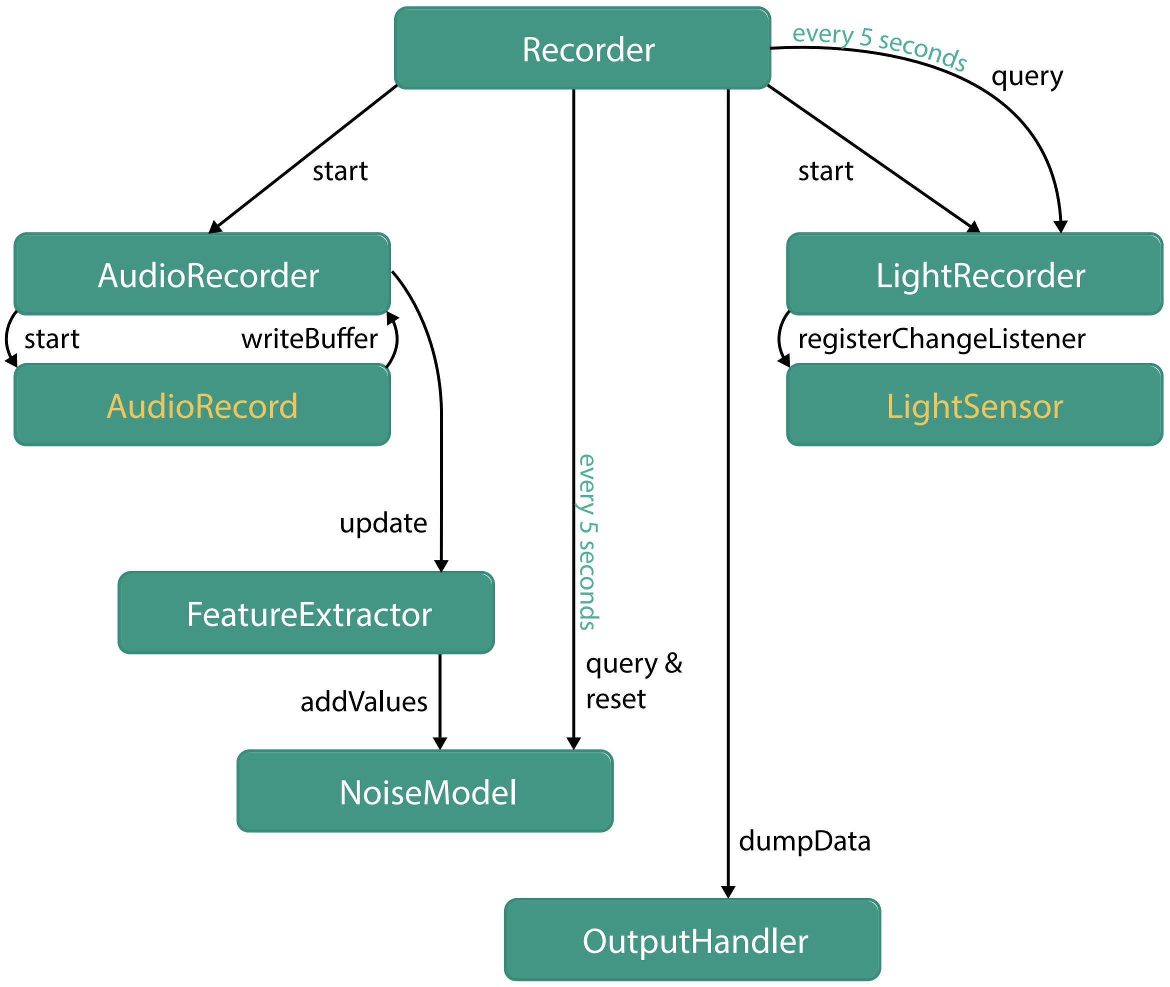


Figure 5: The structure of the frameworks recording cycle. White class names are implemented by the framework, coloured ones are system classes

4.1.1 Light Component

Fetching the light intensity is very easy with Android. The *LightRecorder* object only has to setup a *SensorEventListener* which is then called by the Android system every time the light intensity changes. The *Sensor-Event* already holds the light intensity in lux, which is very convenient for further interpretation. To be able to continuously receive *SensorEvents* it is necessary to create a wake lock. Wake locks are used in Android to keep specific parts of the device running until the wake lock is released. There are different wake locks available; they are discussed in detail because it is not trivial to decide which one to use. Table 5 presents the relevant wakelock types and shows the different features.

All wake locks seem to guarantee that sensor events will be delivered as

Wake Lock	**CPU State**	**Screen**
PARTIAL_WAKE_LOCK	On*	Off
SCREEN_DIM_WAKELOCK	On	Dim
SCREEN_BRIGHT_WAKELOCK	On	Full

Table 5: Some of the wake locks which are available in Android. Table taken from [21]

long as the wake lock is held. The * at PARTIAL_WAKE_LOCK indicates that the CPU is truly always on and the user has no possibility to change that. The other two wake lock types can be interrupted by the user by pressing the power button. Thereby, the screen turns off and the CPU is not further guaranteed to be on. As the screen is not needed for the framework and might even interfere with the light intensity measurement, the obvious choice is to use a PARTIAL_WAKE_LOCK. This is also the best choice regarding power consumption as the screen is one of the main power consumers.
However, some devices have one major drawback regarding PARTIAL-_WAKE_LOCK. They stop some sensors while the device's screen is off, even when the lock is held [17]. This seems to be a problem with some device's drivers which link the screen with these sensors. As soon as the screen is turned off, the sensor is stopped as well. A workaround for some devices is to reregister the *SensorEventListener* after the screen was turned off [28]. This behavior is implemented in the framework but does not resolve the issue for every device. Currently the only possible workaround for every device would be to implement a feature which tries to discover if the light sensor doesn't report new values and then acquires a SCREEN_DIM_WAKELOCK. This would guarantee that the light sensor keeps working (as long as the user doesn't manually turn the screen off) but would also drain a lot more battery.
Another general issue with most smartphone light sensors is the way the light intensity measurement works. Basically the light sensor only detects light which falls orthogonally onto the device. That means the light sensor can't give an exact representation of the room light, but only of the light which is falling directly onto the light sensor. A simple test has been made to demonstrate the issue:

The device was placed into a completely dark bedroom. A light source

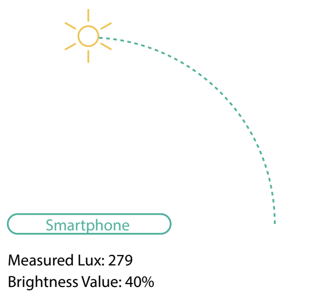


Figure 6: Light source orthogonally to the device

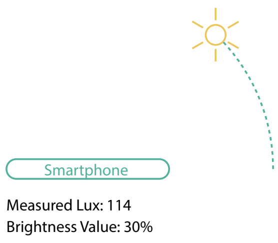


Figure 7: Angle of 45 degrees

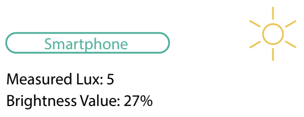


Figure 8: Angle of 5 degrees

was then placed at different angles and the lux value from the device was noted. Furthermore, an image has been taken from which the perceived brightness value was extracted. When the light source is orthogonally to the device as seen in figure 6, a lux value of 279 was measured. This is the base case. The smaller the angle gets, the larger the discrepancy between measured brightness and perceived brightness gets. The difference in perceived brightness is only 10% between figure 7 and figure 8, but the measured brightness of figure 8 compared with figure 7 is 95% lower. This can be explained by the fact that the light sensor is effectively measuring the brightness of the ceiling and not the environmental light. As the ceiling is always illuminated as well when there is a light source in the room, the lux value can still be used. It is, however, important to be aware of this issue.

4.1.2 Audio Component

```
new AudioRecord(
    MediaRecorder.AudioSource.MIC,
    16000,
    AudioFormat.CHANNEL_IN_MONO,
    AudioFormat.ENCODING_PCM_16BIT,
    1600
);
```

Listing 1: Initialisation of an AudioRecord object

This initialisation code is discussed in detail because the parameters are very important for the further processing of the data and for minimal power consumption and best device compatibility.

- Line 2 specifies which microphone to use. MIC is used because it is available on all platforms [2].

- Line 3 is the sampling rate in Hz. This corresponds to line 6, as all audio data is processed in 0.1 second frames. 16000Hz is used as no voice audio needs to be recorded and the highest relevant frequency won't be above 8000Hz. Therefore, the sampling rate of 16000 holds true to the Nyquist rate. Although, 44100Hz is available on all devices and 16000Hz is not guaranteed to be available, 16000Hz is used because of the reduced CPU usage.

- Line 4 specifies the configuration of the audio channels. CHANNEL_IN_MONO is guaranteed to work on all devices and as stereo recording isn't needed anyways, this setting is perfect [2].

- Line 5 specifies the audio data format. ENCODING_PCM_16BIT is supported by all devices and therefore used [2].

After the initialisation it is now possible to call the *read*() method on the *AudioRecord* object which fills a buffer of *short* values with the recorded audio. This buffer is then passed to the *FeatureExtractor* as seen in figure 5. The *FeatureExtractor* calculates each of the features RLH, RMS and VAR as follows:

RMS
As the values from the buffer can be negative and positive the root mean square is used to get a marker for the audio volume:

$$RMS = \sqrt{\sum_{i=1}^{b.length} b_i^2} \qquad (2)$$

VAR
The variance is used as an indicator whether an event occurred or not. In

the following formula *mean* is the average value of all buffer elements.

$$VAR = \frac{\sum_{i=1}^{b.length}(b_i - mean)^2}{b.length} \tag{3}$$

RLH
The ratio of low to high frequency bands is the most difficult feature to calculate, because low and high frequency bands have to be calculated first. Many different algorithms exist for calculating these bands, the following ones are used in the framework:

Low frequency

$$b_i^l = b_{i-1}^l + \alpha \cdot (b_i - b_{i-1}^l) \tag{4}$$

With $b_0^l = 0$, $\alpha = 0.25$

High frequency

$$b_i^h = \alpha \cdot (b_{i-1}^h + b_i - b_{i-1}) \tag{5}$$

With $b_0^h = 0$, $\alpha = 0.25$

Figure 9: Formulas for calculating low and high frequency bands. Both formulas are taken from [9].

The RLH feature is then calculated by dividing the RMS value of the low frequency band by the RMS value of the high frequency band:

$$RLH = \frac{RMS(b^l)}{RMS(b^h)} \tag{6}$$

These features are then fed into the NoiseModel which stores the last 100 VAR values and the current RMS and RLH values as discussed in section 2.2. The *NoiseModel* decides if an event occurred based on the values of VAR, RLH and RMS. Figure 10 shows the decision values. For $RLH > 10$ and $VAR > 2$ snoring is detected. For $VAR > 0.5$, $RMS > 15$ and $RLH < 10$ movement is detected. Every other value combination is considered to be noise. These values have been selected by tests with a Nexus 6 and a Nexus 7. Other microphones might report different values for the three features, but the thresholds seem to be chosen good enough that the detection works for most phones. The largest differences between devices occurred at the RMS feature but as it is only used for the movement event and all values above 15 are accepted the differences seem not to matter too much.

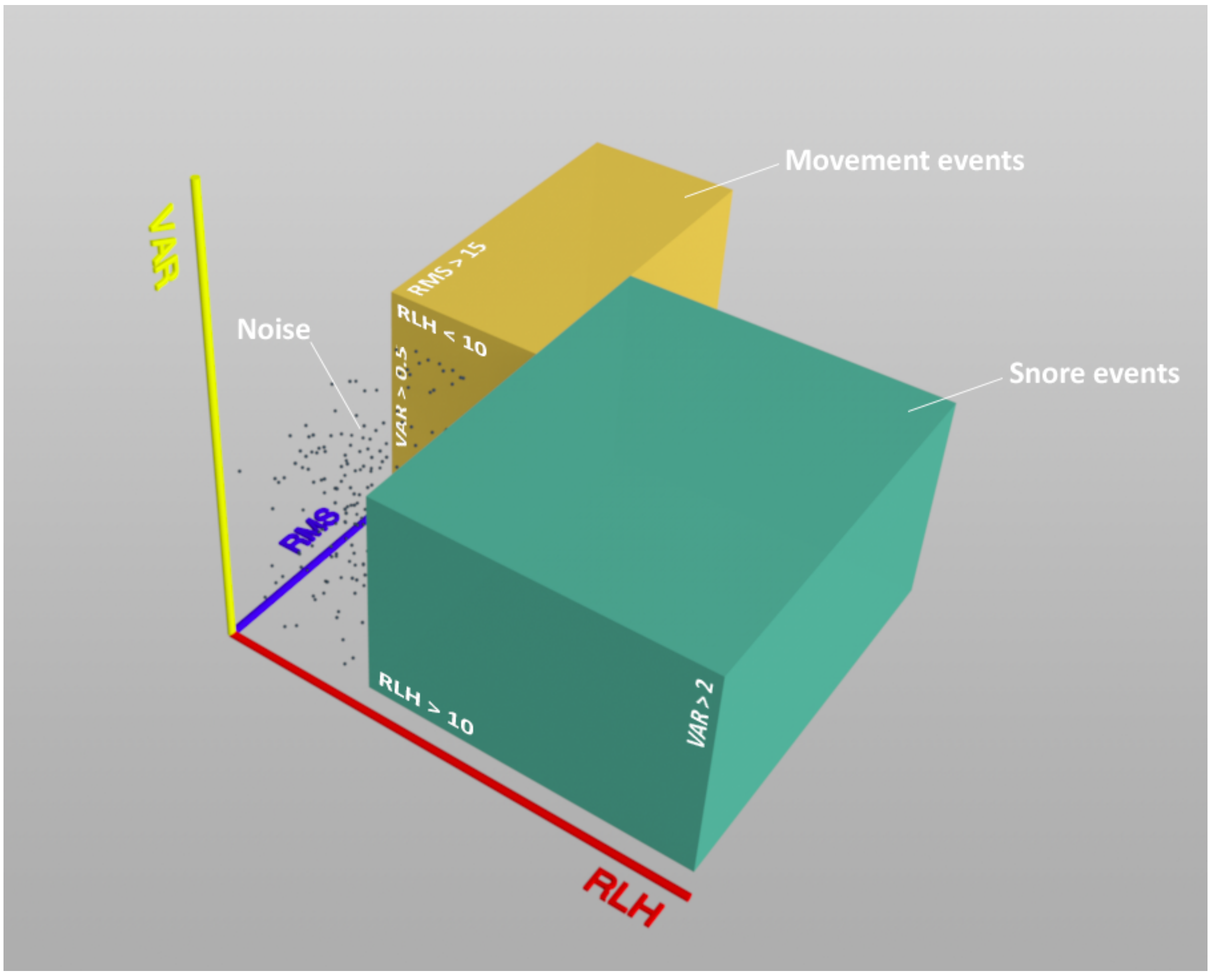


Figure 10: Visualisation of the decision values for the NoiseModel

4.1.3 Output Handling

In order to integrate the framework nicely into third party applications, it follows the SOLID principles. The SOLID principles are a set of best practices regarding object oriented software design which were introduced by Robert C. Martin in the early 2000s [14]. The five principles are:

- Single responsibility principle: A class should only have a single reason to change
- Open / closed principle: A class should be open for extension but closed for modification. That means extending the class should be easy without actually changing it's code.
- Liskov substitution principle: Objects in a program should be re-

placeable by subclasses of these objects without the program behaving different. In practice this means that code should never check for a specific implementation but always assume the interface.

- Interface segregation principle: Many interfaces are better then one large general one.
- Dependency inversion principle: High level classes should not depend on lower level classes; Both should depend on interfaces. Also abstractions shouldn't depend upon specific implementations.

The most important one of these principles for the framework is the dependency inversion principle as it allows external code to hook into the storage mechanism. The Recorder class expects an implementation of the OutputHandler interface:

```java
public interface OutputHandler {
    void saveData(String data, String identifier);
}
```

Listing 2: The OutputHandler interface

The implementation provided by the final application simply implements the saveData method which accepts the recorded data and an identifier for the recording. The identifier is basically the unix timestamp from the start of the recording. By using the interface and not a concrete implementation it is possible to implement custom OutputHandlers which might transfer the recording directly onto a server or store it in a database on the device.

4.1.4 Data Structure

A custom data structure for storing the recorded events has been developed as existing formats have a lot of overhead. A lot of data is accumulated over the night so an efficient storage format is important. Figure 11 shows the final data structure.

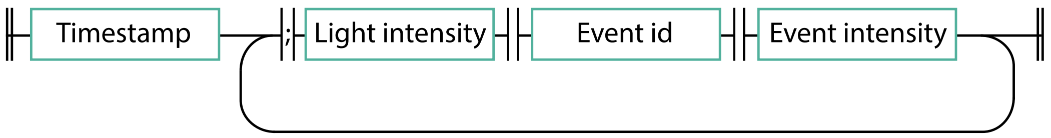


Figure 11: The data structure which is used for the recorded data

Firstly, the unix timestamp of the start of the recording is stored. The unix timestamp represents the amount of seconds which passed since the first January of 1970. Leap seconds are not counted.
The initial timestamp is followed by n integer triples. Each integer is divided by a space and each triple by a semicolon. The three integers define the light intensity in lux, the event id and the event intensity. The possible event ids are:

- **0** No event occurred
- **1** Snoring
- **2** Movement

The event intensity specifies the amount of frames in which the event occurred. One frame is 0.1 seconds long and each triple represents a 5 second interval. This results in a maximum intensity of 50.
This storage format is very space efficient. For an example night of 8 hours, 5760 triples are recorded. The lux intensity ranges from about $5-15$ (dark room) to < 1000 (morning light). Most of the time the room should be quite dark. Therefore, an average character length of two can be assumed. The event id is obviously one character long and the event intensity should average at one character as most triples have no event and therefore no intensity. Two spaces and one semicolon are needed to separate the values and the triples, so we have an average length of

$$2B + 1B + 1B + 3B = 7B$$

per triple. The size of the recording for the sample night of 8 hours should therefore be around

$$10B + 7B \cdot 5760 = 40330B \approx 40kB$$

Storing the same data in a minimal JSON format would require at least 2.3 times more space.

```
12 2 5 // One triple in the custom-build format
{l:12,e:2,i:5} // One entity in JSON format
```

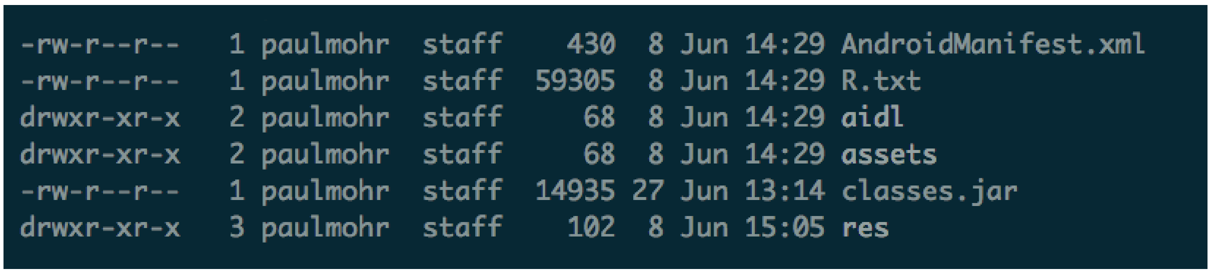
Listing 3: Comparison between the custom-build storage format and a minimal JSON

4.2 Sample Application

Besides the framework, a sample application was developed which utilises the features offered by the framework and also demonstrates some best practices for building a sleep monitoring application. Moreover, the sample application was used to validate the assumptions made during the development of the framework and could also serve as a starting point for third party applications using the framework. Therefore, the framework and the sample application have been open sourced and are available via GitHub: https://github.com/Sopamo/sleepminder.
The demo application is an easy to use sleep tracker. The user can start recording by tapping on the main button. The device then starts to record and interpret the audio and light information. The user is informed by an ongoing notification that the recording is active. This is very important, as the user should always be aware of the current recording status. Otherwise a bad feeling of being unknowingly monitored might arise. The user can stop the recording at any time by pressing the main button again. Furthermore, the application shows a list of all past recording which may be viewed by the user. The application then shows details of the light intensity during the night, the sleep states and an approximation of the sleep quality. Those details will be discussed in section 4.2.5.

4.2.1 Using an Android framework

While one would assume that using an Android framework is the same as using any .jar file while developing Java applications there are indeed some differences. Android frameworks are not distributed as .jar files but as Android specific .aar files. An .aar file is actually a simple zip file which contains the files seen in figure 12 [1].



```
-rw-r--r--   1 paulmohr  staff    430  8 Jun 14:29 AndroidManifest.xml
-rw-r--r--   1 paulmohr  staff  59305  8 Jun 14:29 R.txt
drwxr-xr-x   2 paulmohr  staff     68  8 Jun 14:29 aidl
drwxr-xr-x   2 paulmohr  staff     68  8 Jun 14:29 assets
-rw-r--r--   1 paulmohr  staff  14935 27 Jun 13:14 classes.jar
drwxr-xr-x   3 paulmohr  staff    102  8 Jun 15:05 res
```

Figure 12: The contents of an .aar file

The differences to a simple .jar file are obvious. The .aar file contains Android specific files and folders like the AndroidManifest.xml file which can be used to define necessary permissions which are needed by the library. These permissions will then be merged into the application's AndroidManifest.xml file.
The .aar file can simply be imported into the IDE. For the widely used IDE Android Studio this is fairly easy. The .aar file is included into the project and then added to the build.gradle file as seen in figure 4.

```
...

dependencies {

    ...

    compile project(':sleepminder.lib')

}
```

Listing 4: Adding the sleepminder library

4.2.2 Demo Application Layout

The layout of the demo application is centered around two goals. The application should be very fast to use and it should be usable in dark environments. As it is used mainly in the evening and morning this is a convenient feature for the user. The main screen of the application is presented in figure 16. The primary action is the start / stop button. By tapping on it the user can start the recording. Below the button is a list of past recordings which can be viewed by tapping on them. The used colours can be seen in figure 13. For large areas like the list background and the header the dark shades are used. Therefore the user is not blinded when using the app. When the user starts / stops the recording a popup informs the user about the success of the action. Both, the floating action button and the popup are features of the Android Design Support Library [11]. This library is used to bring the latest layout features to older Android devices as well. Otherwise, the minimum Android version would have to be 5.0 (API level 21) or a separate layout would have to be made for pre Lollipop devices.

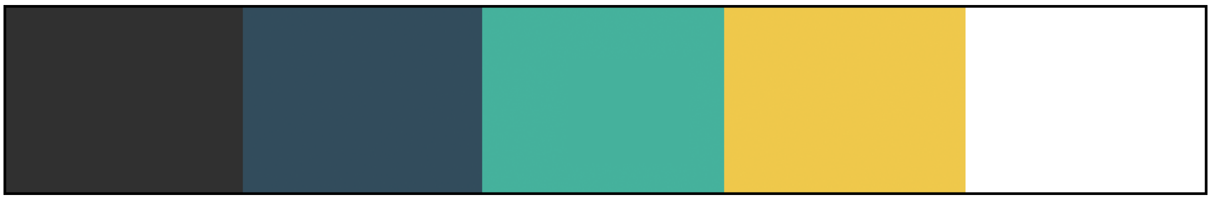


Figure 13: The colour scheme of the demo application

4.2.3 Android Background Service

Even though, it would be possible to start the recording in the main application, there is a much better method which ensures that the recording is not interrupted. The Android *Service* class is made for exactly this reason. "A Service is an application component that can perform long-running operations in the background and does not provide a user interface" [27]. The Service is started and controlled by an activity and may provide a notification to the user about the current state of the service. The notification which is shown in the sample application can be seen in figure 14.

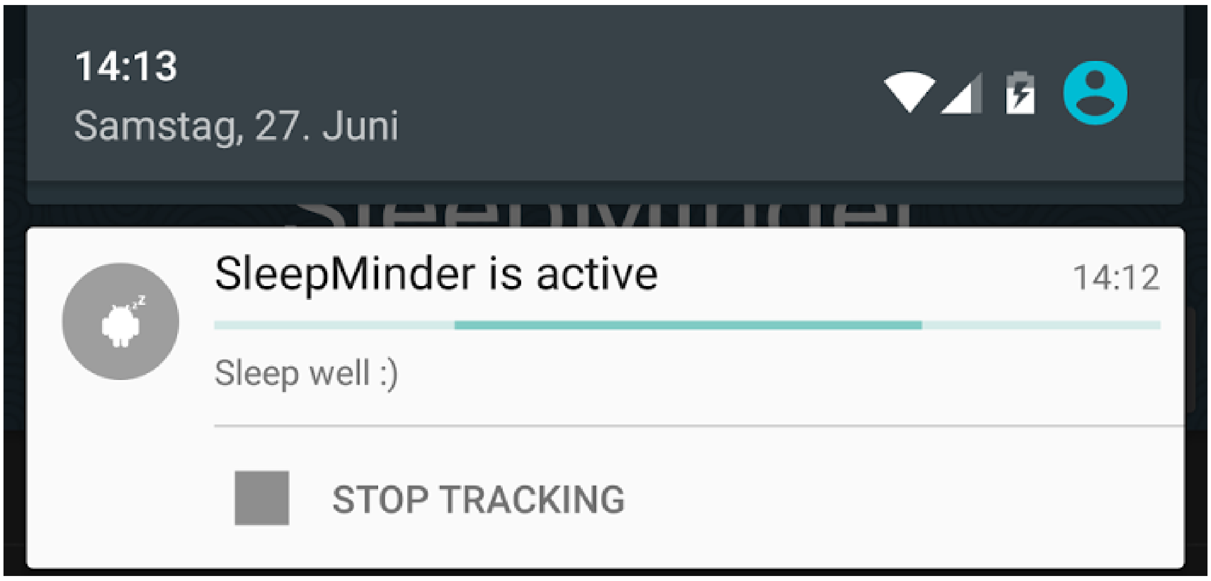


Figure 14: The notification of the demo application in the notification bar

In order to create a Service some configuration steps are necessary, so the service behaves as required.

Declare the Service
Like an activity, the service must be declared in the AndroidManifest.xml file. No further configuration is needed at this place.

Start the Service
The *startService* method must be called on an Android context to start a service. This results in the Android system calling the *onStartCommand* method of the implemented Service class. The return value of this method is very important as it tells the Android system how to handle the service. Three main return values are available, they are listed in table 6.

Flag	**Description**
START_NOT_STICKY	Suitable for services which need to do some less important work in the background. The Android system might kill the service due to memory pressure and will not start it again automatically.
START_STICKY	If the service should get stopped, it will be restarted by the system until the *stopSelf* method is called.
START_REDELIVER_INTENT	Behaves like START_STICKY but the initial intend will be redelivered.

Table 6: Some of the return values for the onStartCommand method [26].

In the demo application the START_STICKY flag is returned, because keeping the service alive is the number one priority. Redelivering the intent is not necessary in this case as the intend doesn't hold any relevant information, apart from the service itself.

Enable foreground mode
To start the service in foreground mode, the *startForeground* method needs to be called in the *onStartCommand* method of the service. This also creates a notification which is automatically shown and hidden on

service creation / destruction. Normally, services are started in the background which means that they may be killed by the Android system to free up memory. When *startForeground* is used, the system knows that the service is essential for the user. A notification needs to be provided so the user knows about the current state of the service. As seen in figure 14, the notification can provide custom actions which are useful for the user. The notification is even shown on the lock screen which is perfect for a sleep monitoring application as the user is remembered of the active recording when they first pick up the phone in the morning. They can then directly stop the recording. The notification on the lock screen can be seen in figure 15.

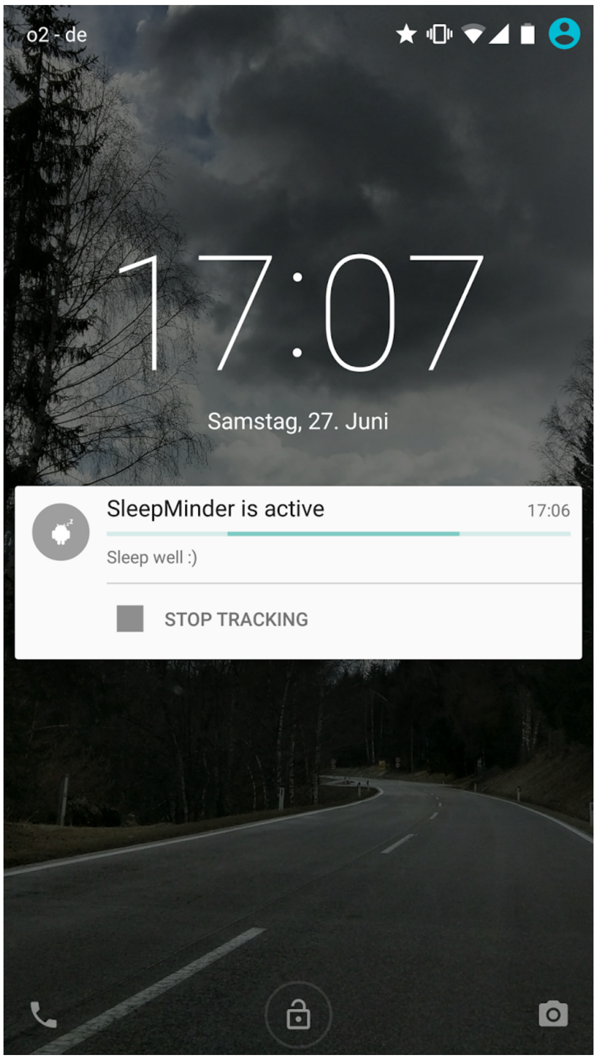
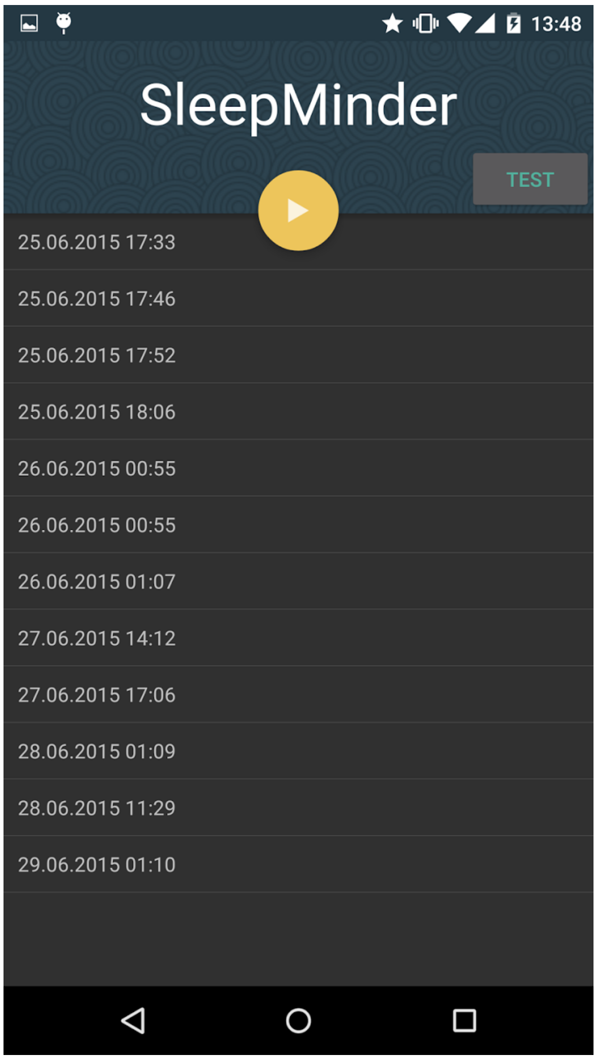


Figure 15: The notification of the demo application on the lock screen so the user sees it after waking up.



25.06.2015 17:33
25.06.2015 17:46
25.06.2015 17:52
25.06.2015 18:06
26.06.2015 00:55
26.06.2015 00:55
26.06.2015 01:07
27.06.2015 14:12
27.06.2015 17:06
28.06.2015 01:09
28.06.2015 11:29
29.06.2015 01:10

Figure 16: The main view of the demo application. The primary button transforms from a start to a stop button.

4.2.4 Data storage

As discussed in section 4.1.3 a custom *StorageHandler* must be implemented. The demo application stores the recordings in the external storage of the Android device. An alternate solution would be the internal storage which is guaranteed to be available at all times. However, using the internal storage also has some drawbacks. Many users prefer to store as much data as possible on the external storage because the internal storage is usually quite limited. In contrast, the external storage is usually a lot larger and as the recordings do use a good amount of space when the application is monitoring sleep every day, the probability of running out of space is a lot smaller when using the external storage. Furthermore, sharing files from the internal storage is relatively difficult compared with sharing files from the external storage. Another difference between external and internal storage is privacy. It is a lot more difficult to access data stored on the internal storage — even with physical access to the device. Therefore, the internal storage is usually used to store very sensitive information. As the recordings do not save any specific data like sound samples they are not considered very sensitive. Hence, the demo application uses the external storage to persist the recordings.

4.2.5 Data visualization & interpretation

In figure 17 and 18 the detail view of a recording can be seen. The course of the night is displayed at the top. The time span is shown above the graph, the bars indicate movement. The night is separated into 30 minute intervals. For each interval all movement events of the 5 second intervals are accumulated. If more than one movement event occurred, the interval is considered a light sleep phase. The height of the bars indicate the intensity of the movement. This is quite a large abstraction but worked in practice. Further optimisations could be made to detect the REM / non-REM phases more accurately.
Figure 17 shows a pretty normal night. 5-6 sleep phases occurred; usually a single cycle lasts about 100 minutes [22]. Therefore, about 5 phases are normal for a sleep duration of 8 hours.
Both screenshots are made on a Nexus 6 device which suffers from the light sensor problems described in section 4.1.1. The light sensor did report a change in the light intensity at about 7:30 because another application requested a SCREEN_DIM_WAKE_LOCK. On a device which keeps the

light sensor active, even when the screen is turned off, the ascent of the light intensity would be more slowly.
Below the main graph, four additional indicators are shown:

Sleep stages
This pie chart shows the amount of time spent in light sleep versus the amount of time spent in deep sleep. It was found that for a sleep duration of about 8 hours about 90 minutes in the deep sleep phase (also called slow-wave sleep) are normal [12]. The graph shows a deep sleep percentage of 33% which indicates good sleep.

Light quality
The light intensities are divided into three groups: Night, dawn and day. Lux values up to 20 are considered as night, values up to 100 as dawn and everything above as daylight. Starting at about 100 lux, light is considered to interfere with the sleep cycle and therefore to affect the sleep in a negative way [29]. Due to the Nexus 6 suffering from the light sensor bug, the light quality pie chart shows 0% dawn light intensity. A perfect night should consist mainly of light below 20 lux and only gets brighter while waking up.

Sleep quality
The sleep quality diagram shows the distribution of audio events. Repeated snoring would be visible here.

Overall quality indicator
Based on the feedback of some test users a simple sleep rating system has been developed. A smiley face indicates the estimated overall sleep quality. There are three different faces: "Good", "Not too bad" and "Bad". Three different indicators are calculated to estimate the quality. Each indicator gets a 1, 0 or -1 depending on how good the sleep is. 1 represents good sleep quality, whereas -1 indicates bad quality. The average is then calculated and the rounded value determines the estimated overall quality.
The first indicator is light quality. If at least one hour of sleep was during daytime light the quality is rated "bad". If daytime light combined with dawnlight is at least 1.5 hours long, the quality is rated as "not too bad". Everything else is rated "good".
The second indicator is the amount of sleep cycles. As discussed earlier about 5 cycles are normal. More than 10 cycles or less than 4 are considered "not too bad", everything else is rated "good".
The third indicator is sleep duration. Sleep duration is difficult to rate

based on a single night. Short sleepers may be perfectly fine with only 5.5 hours of sleep while long sleepers may need 8.5 hours [8]. As the reasons for the amount of sleep an individual needs, are not clear yet, the optimal duration would have to be determined by a questionnaire where the user would have to state the needed duration to be well-rested as well as the family background — families have shown to have a similar need for sleep duration [10]. Such a questionnaire could be implemented in the application as well, but would exceed the scope of this thesis [24]. Therefore, the application simply rates more than 7 hours of sleep as "good", more than 5.5 hours as "not too bad" and everything below as "bad".

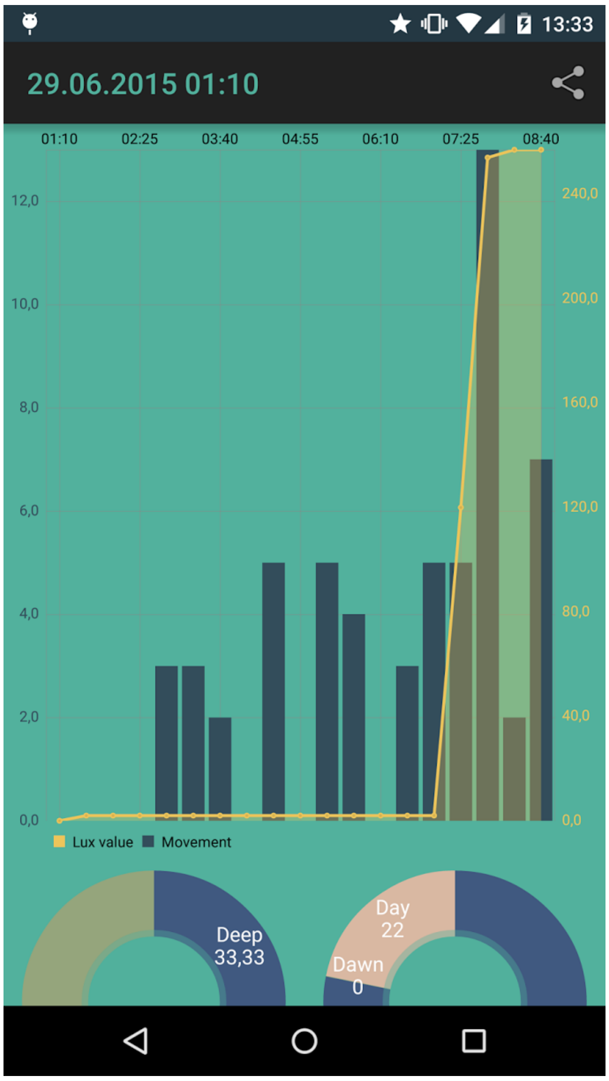


Figure 17: The detail view of a recording shows the movement events and light intensities over the course of the night

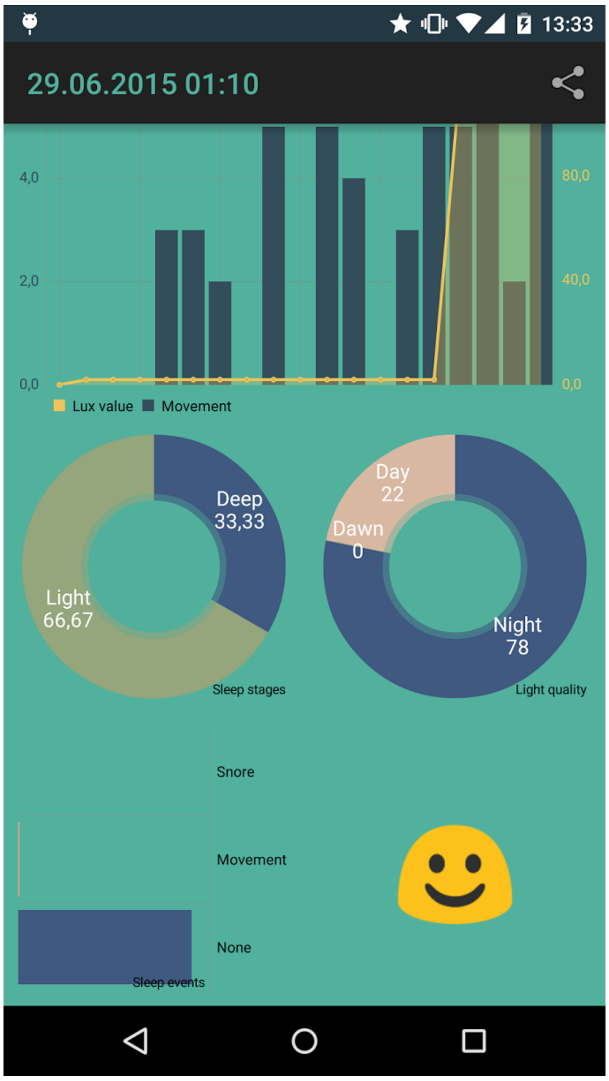


Figure 18: The bottom part of a recording shows some statistics and the sleep quality

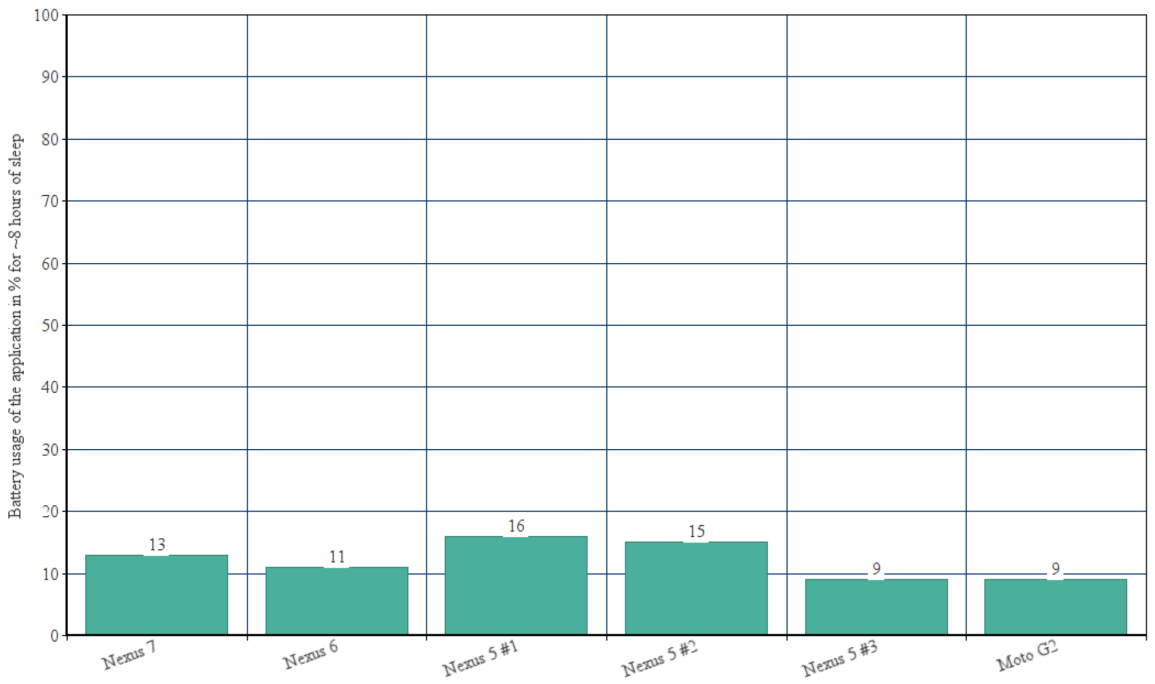


Figure 19: The power consumption in % for 8 hours of recording for various devices

4.2.6 Power consumption

The user should be able to monitor their sleep without having to connect the smartphone to the power supply. Therefore, one of the secondary goals of the framework and the demo application is to consume as little power as possible. Personal experience showed, that other sleep monitoring applications for Android use about 50-80% of the device's battery. A small study with six devices has been made to test the power consumption of the demo application. Each device monitored about 8 hours of sleep and was charged very close to 100% before starting the recording. Figure 19 presents the results of the study. Power consumption is between 9% and 16% for all devices. This allows the user to record their sleep without having to worry about power supply.

5 Requirements Comparison

The requirements from section 3 are compared with the results of the thesis. A simple rating scheme from ++ if the requirement is met perfectly, to −− if the requirement could not be implemented, is used.

5.1 Functional Requirements

The essential functional requirements for both the framework and the application could be implemented. Some issues with the devices' light sensors and differences in the microphone volumes occurred, but none of them had a significant impact on the main functionality.

5.1.1 Framework

Regarding the functional requirements for the framework, not being able to record light intensities on some phones was the main issue. This issue could be resolved partially but still exist on some devices. Apart from that, no major problems remained.

Table 7: Functional requirements comparison for the framework

	Requirement	**Rating**	**Comparison**
FR #1	The framework should detect and record movement during sleep.	++	Movement is detected and recorded.
FR #2	The framework should record light intensity changes.	o	Light intensity is recorded on most devices. As some devices are unable to record the light intensity while the screen is off, this feature does not work on all devices.

Continued on next page

Table 7 – *Continued from previous page*

	Requirement	**Rating**	**Comparison**
FR #3	The framework should detect and record snoring during sleep.	++	Snoring is detected and differentiated from movement. As snoring might also trigger movement events, a threshold is introduced to separate movement frames from snoring frames.
FR #4	The framework should record the mean audio volume per frame.	−−	Due to lack of time and problems with the accuracy of the different smartphone microphones this feature has not been implemented.
FR #5	The framework should offer a functionality to rate the sleep quality based on a recording.	+	The framework and the demo application extract sleep quality and cycles as far as the scope of the thesis allows.
FR #6	The framework should offer a customizable data storage system.	++	A very flexible storage system was implemented.
FR #7	The Framework should write the accumulated data to the output handler in regular intervals.	++	A sensitive value of about 15 minutes is used to guarantee very litte data loss at worst.

5.1.2 Demo Application

All functional requirements for the demo application could be implemented.

Table 8: Functional requirements comparison for the demo application

	Requirement	Rating	Comparison
FR #8	The demo application should offer the user a detail view for each recording.	++	A detail view which shows all collected data was implemented. The detail view even offers some interpretation of the recorded events.
FR #9	The demo application should offer a share functionality for the recordings.	++	A share button is present on the detail view which lets the user choose how to share the text file with the recording information.

5.2 Non-Functional Requirements

Most of the non-functional requirements could be met perfectly. The framework and demo application are already very fast and have a small power consumption. The optimisations which could be made mainly focus on low-end devices.

5.2.1 Framework

Apart from some minor performance optimisations the non-functional requirements for the framework could be fulfilled. Especially the power consumption is even better than initially assumed. Therefore, some of the calculations which are currently made in the demo application when the user views the recording could already be made during the night after the device calculated the three audio features. The framework could offer a functionality to call application specific code every time a 0.1 second frame was calculated. With this additional feature the application could receive a completely analysed recording at the end of the night.

Table 9: Non-Functional requirements comparison for the framework

	Requirement	**Rating**	**Comparison**
NFR #1	The framework should use the CPU as little as possible.	+	The framework needs only about 10% - 15% of the battery for 8 hours of recording. Therefore, the goal of being able to record a night without having to connect the phone to power supply was reached. Some optimisations could be implemented to further reduce the CPU usage.
NFR #2	The framework should handle all non-critical errors silently.	++	All critical methods are wrapped in try ... catch blocks to be able to handle runtime problems like OutOfMemory or BufferOverflow exceptions.
NFR #3	The framework should restart the recording after critical errors.	++	Not the framework, but the demo application starts the service in foreground mode with the START_STICKY flag which ensures the restart of the recording if it should crash.
NFR #4	The framework should be memory efficient.	+	Some memory problems occurred during development and the framework was optimized accordingly. Some of the algorithms could be rewritten to in place variants which would improve the memory efficiency even further.

5.2.2 Demo Application

The non-functional requirements for the demo application could be fulfilled as well. The demo application runs on most devices running Android API level 10 or higher. With the possibility to test on older or more exotic devices like smart watches, the application could be made available on even more devices. According to the Google Play developer console 8078 out of the 9746 registered devices are supported. In consideration of the large Android device and operating system fragmentation, 82% device support is a good value. Furthermore, most of the unsupported devices are TVs or digital cameras which run Android as operating system. These are irrelevant for this use case anyways.

Table 10: Non functional requirements comparison for the demo application

	Requirement	**Rating**	**Comparison**
NFR #5	The demo application should always inform the user about the current recording state.	++	A temporary popup is shown to the user to inform about the successful start / finish of the recording. Additionally, a permanent notification is shown in the notification bar.
NFR #6	The demo application should be easy to use in dark environments.	++	All background colours are quite dark. The main action is highlighted with the primary colour of the colour scheme. The detail view also uses adjusted colours for all graphs to be not too bright.

Continued on next page

Table 10 – *Continued from previous page*

	Requirement	**Rating**	**Comparison**
NFR #7	The demo application should respond fast to user input.	+	Starting and stopping the recording happens instantly. These two actions are the most used ones and are therefore most important. The detail view has to process a lot of information which makes it a bit slow on less powerful devices ($\sim$ 500ms loading time). Some precalculations could be made to interpret the recordings while the user is not using the phone.
NFR #8	The demo application should be usable on as many devices as possible.	+	The demo application uses some features of the API level 21 (Android version 5.0). The layout has been adjusted for older API levels but is not looking identical. Also a recording parameter is used which is not guaranteed to be available for all devices but ensures less CPU usage. The application could be improved by checking for the available recording parameters first. Nevertheless, a good compatibility could be achieved.

6 Recapitulation

In order to complete this thesis an overview of the achieved features is given as well as an outlook on what could be improved regarding the framework. The goal to develop a framework for the Android platform which can be easily used in other applications has been achieved. The framework provides a very clean interface to start a recording and extract relevant information out of it. The framework's goal to be user friendly by not having to place the smartphone into the bed during the night was a tough one to achieve but could finally be implemented. Simply using the motion sensor would be a lot easier — the main difficulty of this thesis turned out to be the interpretation of the audio data. Extracting audio events like movement or snoring out of an array of values ranging from -10 to 10 proved to be quite a challenge. Nevertheless a solid working algorithm could be implemented.

6.1 Improvements

Whereas the framework is already working as expected it could still be improved in both energy efficiency and accuracy.

6.1.1 Event extraction

Even though a normalisation algorithm has been applied, the feature extraction is still dependent on the device's microphone in some parts. Especially the interpretation part of the framework which maps the extracted features VAR, RLH and RMS to specific audio events could be improved by executing a larger study with a wide variety of devices to evaluate which thresholds do not work equally on all devices and have to be adjusted.

6.1.2 Efficiency

Currently, the framework calculates all features for every 0.1 second frame individually. Even though the power consumption turned out to be good enough to be able to record a full night without having to charge the phone, a more optimised algorithm would use even less energy. By detecting events only via a change in the VAR feature and deciding based on the value of VAR whether or not the other features need to be calculated some computing time could be saved. An even more advanced approach would be to reduce the sampling interval by not interpreting every 0.1 second frame when a sleep phase is detected. As sleep stages usually last at least half an hour only every 10th (or even 100th) frame could be calculated. The skipped frames could still be recorded and be calculated afterwards when an audio event occurred. This would possibly result in a fairly large reduction in power consumption but would also be quite complex to implement.

6.1.3 Co sleeping

Due to the framework detecting movement events via audio, it might not correctly detect movements when the user is sleeping in one bed with their partner. Even tough it was found that about 1/3 of the movement events are shared by both partners, there are still a lot of movements which the microphone might pick up from the partner sleeping next to the user [18]. A possible solution would be that both partners record their sleep with their smartphones placed on their side of the bed. In the morning both smartphones could exchange the recorded data and try to assign each movement event to the correct user. A good indicator for deciding which user to assign the event to would be the intensity or the volume of the event.

6.2 Closing Statement

The demo application and the framework are a starting point for Android applications which want to collect audio events during human sleep. Although, there are existing Android applications which monitor sleep, none of them are open source or even offer an encapsulated framework. The initial use case for the framework was to develop an application to help diagnosing the correct tinnitus variant. In the final tinnitus app this framework should be used next to other modules. Therefore, the focus to develop an easy to integrate framework was important. Of course the framework can also be used in other projects. The framework and the demo application are available on Github: https://github.com/Sopamo/sleepminder.

Next to the application and the framework, a complete documentation has been created to enable other developers to use and modify the framework as needed. The documentation is created as Javadoc in the codebase. Additionally, the most important parts are described and explained in detail in this thesis.

7 Acknowledgement

Firstly, I would like to thank my advisor Marc Schickler for his support and for giving me the possibility to challenge myself. I would also like to thank Kiara Freitag and Fabian Henkel for listening to my struggles with audio interpretation and motivating me to keep going. Likewise, a lot of thanks to my family for supporting me during the course of my bachelor.

Additional thanks go to the Android team for developing such a great mobile platform and especially to the Android Studio team which built a fantastic IDE for Android development.

References

[1] *AAR Format*. URL: http://tools.android.com/tech-docs/new-build-system/aar-format (Last accessed: 2nd July 2015).

[2] *AudioFormat | Android Developers*. 2015. URL: http://developer.android.com/reference/android/media/AudioFormat.html (Last accessed: 2nd July 2015).

[3] Alexander A Borbély et al. “Sleep deprivation: effect on sleep stages and EEG power density in man”. In: *Electroencephalography and clinical neurophysiology* 51.5 (1981), pp. 483–493.

[4] Sharon J Borrow. *The Stages Of Snoring*. URL: http://www.britishsnoring.co.uk/stages_of_snoring.php (Last accessed: 2nd July 2015).

[5] Bsmacbride. *Bed Foley.wav*. 2010. URL: https://www.freesound.org/people/bsmacbride/sounds/108515/ (Last accessed: 2nd July 2015).

[6] William Dement and Nathaniel Kleitman. “Cyclic variations in EEG during sleep and their relation to eye movements, body motility, and dreaming”. In: *Electroencephalography and clinical neurophysiology* 9.4 (1957), pp. 673–690.

[7] Ermine. *snore_okm2.flac*. 2006. URL: https://www.freesound.org/people/ermine/sounds/27403/ (Last accessed: 2nd July 2015).

[8] Michele Ferrara and Luigi De Gennaro. “How much sleep do we need?” In: *Sleep medicine reviews* 5.2 (2001), pp. 155–179.

[9] Tian Hao, Guoliang Xing and Gang Zhou. “iSleep: Unobtrusive Sleep Quality Monitoring using Smartphones”. In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM. 2013, p. 4.

[10] Hyun Hor and Mehdi Tafti. “How much sleep do we need?” In: *Science* 325.5942 (2009), pp. 825–826.

[11] Ian Lake. *Android Design Support Library*. May 2015. URL: http://android-developers.blogspot.de/2015/05/android-design-support-library.html (Last accessed: 2nd July 2015).

[12] Hans-Peter Landolt et al. “Caffeine attenuates waking and sleep electroencephalographic markers of sleep homeostasis in humans”. In: *Neuropsychopharmacology* 29.10 (2004), pp. 1933–1939.

[13] Ingrid Lunden. *80% Of All Online Adults Now Own A Smartphone, Less Than 10% Use Wearables.* 2015. URL: `http://techcrunch.com/2015/01/12/80-of-all-online-adults-now-own-a-smartphone-less-than-10-use-wearables/` (Last accessed: 2nd July 2015).

[14] Robert Cecil Martin. *The Principles of OOD.* URL: `http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod` (Last accessed: 2nd July 2015).

[15] Sherry Mazzocchi. *How Much Does a Sleep Study Cost? Well, $600 or $5,070.* 2013. URL: `http://clearhealthcosts.com/blog/2013/04/how-much-does-a-sleep-study-cost-well-600-or-5070/` (Last accessed: 2nd July 2015).

[16] Bruce S McEwen. "Sleep deprivation as a neurobiologic and physiologic stressor: allostasis and allostatic load". In: *Metabolism* 55 (2006), S20–S23.

[17] *OnSensorChanged() is no longer called in standby mode since last Firmware upgrade.* 2009. URL: `https://code.google.com/p/android/issues/detail?id=3708` (Last accessed: 2nd July 2015).

[18] FRANCESCA P Pankhurst and JA Horne. "The influence of bed partners on movement during sleep." In: *Sleep* 17.4 (1994), pp. 308–315.

[19] Brandon Peters. *How Much Do Sleep Studies Cost?* 2014. URL: `http://sleepdisorders.about.com/od/sleepdisorderevaluation/fl/How-Much-Do-Sleep-Studies-Cost.htm` (Last accessed: 2nd July 2015).

[20] Kevin Phillips. *How Much Does a Sleep Study Cost? (Rates, Fees, & Discounts).* 2014. URL: `http://www.alaskasleep.com/blog/costs-sleep-studies-rates-fees-discounts` (Last accessed: 2nd July 2015).

[21] *Power Manager | Android Developers.* 2015. URL: `http://developer.android.com/reference/android/os/PowerManager.html` (Last accessed: 2nd July 2015).

[22] Howard P Roffwarg, Joseph N Muzio and William C Dement. "Ontogenetic development of the human sleep-dream cycle." In: *Science* (1966).

[23] Marc Schickler et al. "An Engine Enabling Location-based Mobile Augmented Reality Applications". In: *Web Information Systems and Technologies - 10th International Conference, WEBIST 2014, Barcelona, Spain, April 3-5, 2014, Revised Selected Papers*. LNBIP. Springer, 2015. URL: `http://dbis.eprints.uni-ulm.de/1137/`.

[24] Johannes Schobel et al. "Process-Driven Data Collection with Smart Mobile Devices". In: *Web Information Systems and Technologies - 10th International Conference, WEBIST 2014, Barcelona, Spain, Revised Selected Papers*. LNBIP. Springer, 2015. URL: `http://dbis.eprints.uni-ulm.de/1136/`.

[25] Johannes Schobel et al. "Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned". In: *9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps*. May 2013, pp. 509–518. URL: `http://dbis.eprints.uni-ulm.de/918/`.

[26] *Service | Android Developers*. URL: `http://developer.android.com/reference/android/app/Service.html` (Last accessed: 2nd July 2015).

[27] *Services | Android Developers*. URL: `http://developer.android.com/guide/components/services.html` (Last accessed: 2nd July 2015).

[28] Jameson Williams. *Getting Android Sensor Events While The Screen is Off*. 2012. URL: `http://nosemaj.org/android-persistent-sensors` (Last accessed: 2nd July 2015).

[29] Jamie M Zeitzer et al. "Sensitivity of the human circadian pacemaker to nocturnal light: melatonin phase resetting and suppression". In: *The Journal of physiology* 526.3 (2000), pp. 695–702.

List of Figures

List of Tables

Statutory Declaration

Hereby I declare that I have authored this thesis with the topic:

“Design and Implementation of an Android Sleep Monitoring Framework”

independently. I have not used other than the declared resources. I have marked all material which has been quoted either literally or by content from the used sources.

Further I declare that I performed all my scientific work following the principles of good scientific practice after the directive of the current “Satzung der Universität Ulm zur Sicherung guter wissenschaftlicher Praxis”.

Ulm,

David Paul James Mohr