Analyzing Mobile Application Usage: Generating Log Files from Mobile Screen Recordings

Philipp Krieter
Institute for Information Management Bremen GmbH, University of Bremen
Bremen, Germany
pkrieter@ifib.de

Andreas Breiter
Institute for Information Management Bremen GmbH, University of Bremen
Bremen, Germany
 abreiter@ifib.de

ABSTRACT
Logging mobile application usage on smartphones is limited to rather general system events unless one has access to the operating system’s or applications’ source code. In this paper, we present a method for analyzing mobile application usage in detail by generating log files based on mobile screen output. We are combining long-term log file analysis and short-term screen recording analysis by utilizing existing computer vision and machine learning methods. To validate the log results of our approach and implementation we collect 118 sample screen recordings of phone usage sessions and evaluate the resulting log files manually. Besides that, we explore the performance of our approach with different video quality parameters: frame rate and bit rate. We show that our method provides detailed data about application use and can work with low-quality video under certain circumstances.

Author Keywords
mobile application usage; log files; screen recordings; video analysis

ACM Classification Keywords
H.5.2. User Interfaces: Evaluation/methodology

INTRODUCTION
Several insights from research into the use of mobile applications are based on log files from data collection applications or frameworks, as analyzing log files can be very useful in order to capture user behavior over the long term or to track specific events or usability problems. [2, 23, 10, 3, 20, 19]. Most logs are limited to rather general system events [21] like which application is present or if the screen is turned on or off. These approaches are able to answer a variety of questions about the use of smartphones and applications, but cannot take into account what users actually do within applications [2]. A user that opens Whatsapp could just chat with another person, but could also be in an audio or video call, update his or her status, take a picture, video or voice message and send it to a group chat. This is making the analysis of indetail mobile application usage difficult, as most logs can not provide deep in-app insights of events or interactions inside applications without modifying the apps’ source code.

In contrast to log files, the analysis of mobile screen recordings provides a different view of user interaction. The strong focus on GUIs in HCI leads to the fact, that the interpretation of screen recordings enables understanding almost any action, behavior or task the user is pursuing while using a computer system as a GUI always represents the current state of a system, based on user and system events [1]. One widely used possibility resulting from this is the recording and analysis of screen videos to draw conclusions about how users interact with software or to evaluate usability or user experience. The analysis of screencasts can provide deep qualitative and quantitative insights into how humans use and interact with computer systems [17, 13, 4, 18]. Hence it is a time-consuming method: the analysis of screen recordings to find certain events or actions of interest is not suitable for long-term interpretation of hours, days or even years of screen recordings. This limits the analysis of screen recordings to relatively short time spans. Currently most user experience evaluation methods are focused on short-term evaluations [24]. Knowledge of how users interact with applications is essential for the development of software products. Therefore we combine two common existing approaches to generate data about user interaction: log files and screen recordings. The goal is to utilize the strengths of both methods to gain meaningful data for further analysis.

In this paper, we propose a method for analyzing mobile application usage in high detail by generating log files based on mobile screen recordings through applying computer vision and machine learning methods. This approach has some major advantages. The method is not dependent on implementing log commands in applications’ source code and is for that reason applicable to “all” application and systems events that occur on the user’s mobile screen, extending the logging capabilities from rather general system information to in-app interactions. It is possible to track multiple mobile applications using the same method. The definition of log events of interest can be done even after collecting data, as the video material can be reused to find different events. This makes the approach flexible and in the case that during the analysis it becomes obvious that other log data is actually needed to...
carry out a study or answer a question, there is no need to collect data again.

The main contribution of this paper is to introduce a method for data collection.

- We show how we can find events automatically in mobile screen recordings to create log files. In order to validate the proposed method, we perform test runs with mobile screen recordings of a sample size that can still be analyzed manually to compare and verify the results. Another question closely related to this is what kind of events are detectable using the proposed method and how these can be defined.

- In addition, we analyze video quality requirements based on bit rate and frame rate to prepare for future large-scale, long-term studies. As high-quality video recordings easily produce big file sizes, it is an issue especially in a mobile environment, under which video quality conditions the method can still produce acceptable results.

RELATED WORK
There are some approaches in the research community focusing on logging and analyzing human-computer interaction which are related to the proposed logging method.

The idea to use screenshots or screen recordings to get better insights has been a frequent topic in HCI research. Findings of a study exploring the possibilities of a tool for logging screenshots and interactions (in this case clicks, touches, selects etc.) show that this information helps to analyze human-computer interaction on mobile devices in a non-laboratory setup together with log data [14]. One of the benefits of this logging method highlighted by Kawalek et al. is that the sighting of the logs combined with screenshots and information about user interactions is more time efficient compared to the review of a screencast to find usability problems.

Chang et al. [9] presented "Sikuli Test", an approach to test desktop GUIs automatically using computer vision methods. The GUI tester first defines scripted GUI events using images and adds how to act on these events. After that, the script acts "like a robotic tester". Although the approach does not aim to generate log files from screen recordings, there are some things in common with our method, like the definition of events and detection of these using computer vision.

The project "InspectorWidget" [12] proposes an automatic screencast annotating system using computer vision techniques. The user can (visually) program annotations and after that automatically annotate screen recordings and visualize the results or export them for further analysis. Template matching and optical character recognition (OCR) are used to recognize events. They present a test scenario for a usability check. The project focuses on desktop devices and was developed for Linux, Windows and Mac. Mobile devices like smartphones and tablet computers are not supported. Although the project’s event detection is not tested or optimized to work with mobile screen recordings, the process of detection is related to the pursued approach presented in this paper.

In addition to approaches, which tend to focus on automation, there are also several research projects that focus on the manual analysis of videos in order to investigate mobile usage behavior.

To understand end-user mobile device usage in detail and context Brown et al. collected screen recordings, audio, GPS location and application launches of iPhone users and let the participants annotate the video material in form of small diary entries for later analysis by the researchers [5]. The purpose here is to investigate specific situations "framed by the concept of the occasioned nature" of the use of mobile devices. Though the focus is especially to not automate the analysis of the collected screen recordings, the work shows how helpful video analysis can be for understanding the details of how users interact and use mobile devices. Based on this study McMillian et al. combined multiple sources of data to analyze mobile device usage in the field and high detail. The so-called in vivo method combines logs of system data like GPS position, active application and additionally video recording of the screen and the phone’s camera and audio of the surrounding noise and voices [21]. They highlight the fact of differences between lab studies and real-use scenarios and the importance to analyze the multiple and combined use of smartphones.

Based on this previous related work we want to develop an approach that combines the advantages of log files and of analyzing screen recordings manually. The related approaches working with desktop screens and computer vision mention the high computational workload in general and especially of template matching. We address this in our approach by using an alternative to compare image parts in fixed regions, exploiting characteristics of mobile screen structure and by video preprocessing. In the next section we describe our approach in detail.

DETECTING EVENTS IN MOBILE SCREEN RECORDINGS

Figure 1. Overview: from mobile screen recordings to log file analysis.
In view of the fact that the use of computer vision methods and machine learning models for text recognition on every frame of a video causes computational effort, it is important to keep this workload low. There are several differences between analyzing mobile and desktop screen recordings which leads to possible optimizations for an efficient detection of events on a mobile platform. Desktop systems typically display multiple applications simultaneously in scalable and moving windows. Most mobile platforms support to display only one application at a time in full screen what reduces the cost to check where and which application or activity is present. Another major difference is that a mobile screen has a way more fixed structure. A lot of applications follow a fixed scheme. The application bar at the top or bottom e.g. usually uniquely identifies the present application and does not change. On this basis, our approach reduces the computing power required by exploiting the characteristics of the mobile platform.

**Video Preprocessing**

We interpret each video file frame by frame. As a first step, the video is slightly preprocessed. Each video frame is compared to its predecessor to calculate similarity. This speeds up the whole process a lot as there are many frames with no visible difference and therefore do not have to be processed again. The found events from the previous frame just get copied for the next log-entry. The similarity is determined by two simple and quick tests. Two consecutive frames are subtracted from each other, resulting in zero values for identical pixels or very small values for small changes. The maximum value and the average difference compared against a threshold are used to decide how different two frames are.

**Event Detection**

Every frame is checked against a list of previously defined events. A frame can contain multiple events or no events, but the log file contains an entry for each frame. An Event contains a log message and several conditions and rules based on a GUI state. In order to define an event, a screenshot of the GUI state of interest is marked with bounding boxes representing regions of interest (illustrated in figure 2). We distinguish between fixed position areas and areas that can occur anywhere on the screen. Additionally, the occurrence of text strings can be searched in areas or in the whole screen using OCR. Besides that, a text string from a screen area can be attached dynamically to the log entry.

In order to save computational time, we start by checking the event’s condition that requires the lowest computational power: first comparison of fixed areas, second searching for image parts that are not at a fixed position by using template matching and afterward optical character recognition. By this order, we can exclude events early. The recognition of events works frame by frame, which makes multi-processing easy and speeds up the process considerably: we compute each frame on its own core, which facilitates the scalability of the process.

In view of the fact that the use of computer vision methods and machine learning models for text recognition on every frame of a video causes computational effort, it is important to keep this workload low. There are several differences between analyzing mobile and desktop screen recordings which leads to possible optimizations for an efficient detection of events on a mobile platform. Desktop systems typically display multiple applications simultaneously in scalable and moving windows. Most mobile platforms support to display only one application at a time in full screen what reduces the cost to check where and which application or activity is present. Another major difference is that a mobile screen has a way more fixed structure. A lot of applications follow a fixed scheme. The application bar at the top or bottom e.g. usually uniquely identifies the present application and does not change. On this basis, our approach reduces the computing power required by exploiting the characteristics of the mobile platform.

**Video Preprocessing**

We interpret each video file frame by frame. As a first step, the video is slightly preprocessed. Each video frame is compared to its predecessor to calculate similarity. This speeds up the whole process a lot as there are many frames with no visible difference and therefore do not have to be processed again. The found events from the previous frame just get copied for the next log-entry. The similarity is determined by two simple and quick tests. Two consecutive frames are subtracted from each other, resulting in zero values for identical pixels or very small values for small changes. The maximum value and the average difference compared against a threshold are used to decide how different two frames are.

**Event Detection**

Every frame is checked against a list of previously defined events. A frame can contain multiple events or no events, but the log file contains an entry for each frame. An Event contains a log message and several conditions and rules based on a GUI state. In order to define an event, a screenshot of the GUI state of interest is marked with bounding boxes representing regions of interest (illustrated in figure 2). We distinguish between fixed position areas and areas that can occur anywhere on the screen. Additionally, the occurrence of text strings can be searched in areas or in the whole screen using OCR. Besides that, a text string from a screen area can be attached dynamically to the log entry.

In order to save computational time, we start by checking the event’s condition that requires the lowest computational power: first comparison of fixed areas, second searching for image parts that are not at a fixed position by using template matching and afterward optical character recognition. By this order, we can exclude events early. The recognition of events works frame by frame, which makes multi-processing easy and speeds up the process considerably: we compute each frame on its own core, which facilitates the scalability of the process.

**Comparing Fixed Areas**

To compare whether a fixed area from an event definition appears at the same position in the current frame we use a perceptual hash function [6]. We take the hash of this fixed area from the current frame and calculate the hamming distance to the hash from the event definition to decide whether the areas are similar. Due to the already mentioned fixed structure of many mobile applications, this approach is beneficial for checking the similarity of important fixed-position areas to increase the detection speed compared to existing approaches for desktop screen recordings, where more expensive template matching techniques are used for the entire video image or parts of it. Additionally, perceptual hashes are quite robust in terms of different scaling of images or small changes and artifacts from video or image compression. We show an example in figure 2, in which we check whether the keyboard is opened in the current screen. Interface elements may appear on a changing background (e.g. Android home button). Perceptual hash comparisons or template matching do not work reliably in this situation. In this case, we modified the detection process slightly. We use OpenCV’s [22] Canny edge detection [7] for elements on a changing backgrounds and use perceptual hashing afterwards. By using the edged image instead of the original image it is possible to identify icons or shapes on changing backgrounds.

**Searching for an Image Part**

To search for partial interface elements or image parts that do not appear at a fixed position we use OpenCV’s template matching algorithm. Searching for a part of an image in another image can be costly in computing time compared to
matching fixed areas of two images. Therefore, template matching is executed in the second step, which means that many events are already excluded. An example of using this is to recognize the moving button to take an incoming call on an Android phone.

**Searching for Text**
As a final step, we use the open source library tesseract [11] for OCR to compare if a string specified in an event definition is found in a certain region of the screen or to fetch a string from a defined position on the screen to add it to the event’s log message. This is helpful to distinguish a lot of events or to save the name of a chat partner for example.

**EXPERIMENTAL SETUP**
In order to evaluate the method, we have carried out a series of test runs. Our approach is twofold: to begin with, we compared the found events by hand with the corresponding video material to validate the results and track down false log entries. Our intention was to identify problematic situations, events and the limits of our event detection.

Besides that, we did certain test runs with the same video material but changes in frame rate and video quality. Table 1 lists the variations of video parameters we used for exploring bit rate and frame rate limits. All video variations use the same codec, h.264. The event list and event definitions did not change during the test phase. Afterwards, we evaluated whether log entries were lost and which events were not recognized correctly or were problematic.

**Collecting Screen Recording Data**
In order to test the proposed method, we collected screen recordings on an Android phone (Google Pixel 2, Android 8 Oreo). We collected 118 recordings of a total of 68 minutes of video material during 24 hours, all in portrait mode. As these recordings may contain very personal data, we decided to use a working telephone of one of the authors to collect test material instead of external participants. The intention was to use a set of videos for testing purposes, which represents approximately a realistic average usage time per day [2]. No special instructions or tasks were followed. These sample recordings were used for testing and validating the log results of the proposed method. At this video length, it is still possible with reasonable effort and expense to find certain events and create a "manual log file" for validating and evaluating the developed method and its setup variations. The original recording was made in high quality, full resolution (1080p) and frame rate to encode later variations of the video material in lower bit and frame rates. This makes it possible to carry out different test runs with the same material but with different video parameters in order to ensure that the results are comparable.

For collection, we developed a simple open-source Android application to record the screen [15]. The application runs as a foreground service and is unnoticeable by the user, except through a notification in the notification center that informs about the recording. Battery, CPU and memory consumption are low and not noticeable on a Google Pixel 2 during the recording. The application splits the videos into usage sessions as defined by van Berkel et al. [23] and only records if the screen is active. The application also recognizes the change of orientation between portrait and landscape and changes the video orientation accordingly. All files are stored as MP4 files and contain the start timestamp in their name.

| description               | frame rate (FPS) | avg. bit rate (KBit/s) | file size (MB) |
|---------------------------|------------------|------------------------|----------------|
| original quality          | 20 (avg.)        | 4418                   | 2191           |
| 10 FPS, low quality       | 10 (fixed)       | 227                    | 113            |
| 5 FPS, low quality        | 5 (fixed)        | 203                    | 101            |
| 5 FPS, very low quality   | 5 (fixed)        | 51                     | 25             |

Table 1: This table specifies the originally recorded video (first row) followed by variations of the same video material but different bit and frame rate settings. The bit rate is the average bit rate.

The originally recorded video material has a variable frame rate according to changes and light situation of 20 FPS on average. By lowering the bit and frame rate we reduced the file size from originally 2191 MB to 25 MB. The indicated bit rate is the average bitrate of all 118 videos of one video quality (Table 1).

**Eventlist for Testing**
The exact definition of events is critical for reliable log results. We defined the test events based on Android screenshots, as our screen recordings for evaluation come from an Android device, but by adjusting the event definitions, the approach would also work for recordings from other devices (e.g. iPhones). When defining an event, the following question is decisive: Which features make the event or action differentiable in the GUI state? We defined a list of 30 sample events representing a series of user interface (UI) events and specific situations when using certain applications. The intention is to test a series of events that present a number of difficulties for the detection: Distinguishability from other similar events, or no events at all, and very short or fine-grained events that could be difficult to capture if the frame rate and quality of the collected video material are low. The events vary and range from very short, fast interactions such as triggering the camera in a Whatsapp-chat to more general interactions such as the presence of the keyboard or smiley keyboard. By "short" events we mean fast interactions of less than one second, such as pressing the "k" key on the keyboard or using the Android home button. Another example is to register the opening of a chat in several chat applications and attaching the names of the chat partners to the log entries (similar to figure 2). A complete list of all events can be found in Table 2 and 3. The variable "x" stands for a value that is retrieved from the event screen, e.g. the name of an incoming caller or the number of new chats in the Whatsapp chat application. Besides that, the tables indicate which techniques the event definitions utilize. The UI events in Table 3 are slightly different to the app-specific events, as they can occur in parallel with other events. E.g. the home button can be pressed in any application or the keyboard is used concurrently with many other events in applications.
# Event description | perceptual hashes | edge detection | template matching | OCR |
---|---|---|---|---|
1. SMS chat opened with chat partner x | ✓ | | | |
2. SMS message/chatlist opened | ✓ | | | |
3. SS starting a new conversation screen opened | ✓ | | | |
4. Whatsapp, chat list opened | ✓ | | | |
5. Whatsapp, chat list opened, the list is scrolled down | ✓ | | | |
6. Whatsapp, number of chats with new messages x | ✓ | | | |
7. Whatsapp, chat opened, chat partner x | ✓ | | | |
8. Whatsapp, using rear camera in a chat | ✓ | | | |
9. Whatsapp, button for taking a picture pressed | ✓ | | | |
10. Instagram opened | ✓ | | | |
11. Instagram feed opened | ✓ | | | |
12. Instagram feed opened, there are new unread messages | ✓ | | | |
13. Instagram direct chat opened with chat partner x | ✓ | | | |
14. Pinterest opened, chat with chat partner x | ✓ | | | |
15. BBC news opened, viewing top stories | ✓ | | | |
16. home screen present | ✓ | | | |
17. Android all apps menu present | ✓ | | | |
18. calculator opened | ✓ | ✓ | | |
19. contacts opened, viewing list | ✓ | | | |
20. contacts opened, editing a contact | ✓ | | | |
21. calling, trying to call contact x | ✓ | | | |
22. calling, in call with contact x | ✓ | | | |
23. calling, hanging up contact x | ✓ | | | |
24. lock screen, incoming call from contact x | ✓ | ✓ | | |
25. lockscreen present | ✓ | ✓ | | |
26. lockscreen present, there are notifications | ✓ | ✓ | ✓ | |

Table 2. This table shows app-specific or task-specific events that are used for test runs and which techniques the individual events use, marked by checkmarks. The "x" indicates that a text or numeric variable from the video material is attached to the log entry.

| # | Event description | perceptual hashes | edge detection | template matching | OCR |
---|---|---|---|---|---|
27. keyboard is opened | ✓ | | | |
28. smiley keyboard is opened | ✓ | | | |
29. the "k" key is pressed | ✓ | | | |
30. the home button is pressed | ✓ | ✓ | | |

Table 3. This table lists general UI events that can occur in parallel with other events.

RESULTS

Using the original video material, our approach produced a log file containing one or more entries for all 79790 frames. We transferred this frame-by-frame log file into an a different format to facilitate analysis. In this final log format an event is defined as a contiguous group of the same results for consecutive frames. E.g. if the event "keyboard opened" was found in the last 100 consecutive frames, we take this as an event with a starting time and of a certain length of time. This results in a list of log entries in which each entry represents an event with a starting point and a duration in chronological order, what makes comparisons with the corresponding video material easier. In addition, we have stored each frame that contained an event together with the log message to make validation of the results more feasible. The processing of the original video material on an Intel i7 CPU took about 10 hours, i.e. an average of 133 frames per minute.

Our approach did not miss any events when processing the originally recorded high-quality videos but produced 2790 log entries with faulty log messages. Most of these entries occurred from OCR when used on frames from a screen transition when switching or closing an application. Taking a closer look at the frame-per-frame log files shows that in some cases of switching applications problems occur with detecting certain events. In these situations, we still find an open chat application with a certain person for example, but when OCR tries to get the name from the chat header, the started transition into another GUI state results in a messed up text string. This produced faulty log messages for events that utilize OCR to retrieve text from frames. Almost all other faulty interpreted frames resulted from the "calculator opened" event which was based on comparing perceptual hash values of the lower screen area with edge detection applied to the image before which led to the unexpected effect that other frames got interpreted randomly as calculator screens. All faulty frames to-
gether resulted in a total of 17 (1.3%) false positive events. As a false positive we consider the finding of an event which contains a correct log message, but which did not appear at this position in the video. The calculator application was present one time in one of the 118 test videos but was found 12 times. Besides that, the number of new messages in chats in WhatsApp was recognized (as 1 and 3) although there were no new messages. The remaining false positives are distributed between the events "Instagram, feed opened" and "SMS chat opened". In the latter case, some frames were interpreted as SMS chat, although they were screens for starting a new SMS conversation.

All other events were detected as intended and are part of a detailed log file of 931 entries (without faulty OCR log messages and false positives) representing a detailed overview of phone usage and in-app events. In total, 40 unique events occurred in the final log file. Short or very temporary events like touching the home button (40 times) or using the k-key on the keyboard (24 times) were detected reliably. The correctly recognized chat events led to log entries of chats with 11 different people in 4 applications (WhatsApp, SMS, Instagram direct message and Pinterest). The smartphones rear camera was used two times inside of WhatsApp to take pictures and send them to two different people. Phone call events were recognized correctly (one outgoing and one incoming call to the same person).

As problematic we found using OCR in GUI transition states leading to unpredictable text results. All these events are conditioned by previous checks of perceptual hashes of fixed areas of the frame to identify the event which are more robust to changes: when comparing perceptual hashes, a transition frame may still contain a WhatsApp header, but OCR no longer delivers stable results. Another problem encountered in the results is that edge detection does not work reliably in all situations and leads to several false positives.

**Phone and App Usage Sessions**

To graphically display the log results in a descriptive way we visualized the log file as phone usage sessions containing application usage sessions in reference to van Berkel et al [23]. An example of a short part from the log files is shown in Figure 3 representing one of the 118 recorded phone usage sessions. The phone usage session, in this case, has a duration of 128 seconds and contains the use of two different applications and the presence of the Android home screen. The application session illustrates the detailed use of in-app events and additionally parallel UI events (table 3), like the opened keyboard in a second line. In this case, the use of the chat application WhatsApp with two different persons and viewing and scrolling down the list of chats is shown. The application session in this example has a duration of 28 seconds.

**Results with Lower Video Qualities**

Following the test run with the original video material, we carried out further tests with the same material, but in poorer quality and at a lower frame rate as described in Table 1. This led to differences in the resulting log files and in missing the occurrence of events.

The results with lower video qualities are shown in Table 4. The table lists the results for missed events and false positive rate per video variation. Lowering the quality and frame rate did not necessarily result in a higher false positive rate, which stayed low between 0.7% and 2.0% for all video qualities. The undetected events are not distributed evenly over the entire event list. Some events performed significantly worse, others remained at a high level. The findings from the results of the original video occur in the lower quality version as well. Most false positives resulted from misinterpreting frames as calculator frames, too, exactly as for the original video material. The same is true for using edge detection on image parts before comparing their perceptual hash values, although this negative effect is much stronger for some events. The home button event, for example, was not missed in the original material, although the event definition uses edge detection as the button can occur on different backgrounds. But all log files of the three lower quality versions of the video material show significant loss of the number of found home button events. The test run using the lowest bit rate videos resulted in zero home button events.

| video material | missed events | false positives |
|----------------|---------------|-----------------|
| 10 FPS, low quality | 18.2% | 1.0% |
| 5 FPS, low quality | 24.8% | 0.7% |
| 5 FPS, very low quality | 30.4% | 2.0% |

**Table 4.** This table lists the results for missed events and false positive rate of the test runs with low quality video material.
Figure 4 shows the performance of some selected events per video quality. We have selected a number of events as examples, which are stable, which are for the most part stable, and events that have produced very poor results. The diagram indicates the proximity of the number of found events in relation to the number of events found in the original video material in percent. This is not the same number as the number of missed events. The expectation is that despite the reduction of video quality and frame rate, the same log file will be created. However, it is possible that the number for an event may vary without missing it. For example, the Instagram direct message chat event was found 2 times in the original material, but 7 times in the lowest quality material. The second occurrence of the event is approx. 5 seconds long in the original videos, but is divided into 6 very short events, which add up to approx. 5 seconds when using the 5 FPS videos with a very low bit rate to generate the log file. Some frames within the event are wrongly recognized due to quality losses. This means that the event was generally recognized, but the number of events in the log file is not as expected.

Figure 4. This diagram shows how close the results for different video qualities are to the results with the original material in percent. For example, for the "home button pressed" event only a little more than 40% of events are found at 10 FPS, at 5 FPS about 30% and at 5 FPS with very low video quality 0%.

On average for the complete event list, the difference to the number of found events in the original video material is 14% for the 10 FPS material, 19% for the 5 FPS videos and 33% for the 5 FPS videos with very low bit rate. The averages were derived from the results for all 40 unique log file events. Many events have stable results in all three test runs, but some of them are very different from the results with the original video material. The selected events in figure 4 show exemplary how single events scored. The first two events (Instagram, incoming call) did not show any differences compared to the original log file, this was true for 8 events in total. The next two events (smiley keyboard, k-key pressed) are slightly different, but still over 80%, which holds true for a group of 14 events. When the k-key is pressed, the animation lasts about 0.3 seconds, long enough to be recorded at a frame rate of 5 FPS. The last three events are examples of events that were very far from the results of the original video material in at least one case. The lock screen event only reached 80% for the 10 FPS videos and had a difference of more than 60% for the lowest quality videos. The last two events did not occur at all in the log files from the video material in lowest quality. The home button event is based on only a part of the button animation whose presence is too short to be reliably captured at low frame rates and bit rates.

Video Preprocessing Results

Table 5 lists the number of skipped frames by comparing the similarity of consecutive frames during preprocessing the videos before the event detection process starts. All test runs have benefited considerably from video preprocessing. Running through the original and low quality 10 FPS and 5 FPS material more than half of the frames were skipped. We expected a higher percentage of redundant frames at higher frame rates, which is the case for 10 FPS and 5 FPS, but not for the original video material at an average of 20 FPS. This might result from the adaptive frame rate of the original screen recordings which produces less redundant frames. Using the lowest quality resulted in skipping slightly less than half of the frames. The difference of 10% between the two 5 FPS versions might result from apparent differences in the stronger compression of video frames and related artifacts.

This preprocessing step saved a considerable amount of computational time, as checking each frame against the event list is the most costly step in the whole process.

| video material         | skipped frames |
|------------------------|----------------|
| original quality       | 54%            |
| 10 FPS, low quality    | 64%            |
| 5 FPS, low quality     | 56%            |
| 5 FPS, very low quality| 46%            |

Table 5. This table shows how many frames are skipped by preprocessing the video material.

DISCUSSION

Most previous work on mobile applications usage that works with log files relies on log events that do not provide insights about what users do inside applications, limiting research in this direction. We present an approach that is capable of providing log files in a different level of detail about what users do in their applications. It is not necessary to implement log commands in the source code of any application beforehand. This makes the approach flexible and the definition of log events can be done after collecting video data.
Our approach combines two methods of different characteristics. Log file generation in general, is an exact long-term instrument to document events in computer systems. Analyzing interactions between users and applications by manually viewing screen recordings is a short-term method. Using computer vision and machine learning methods to create log files is less exact in its outcomes as these techniques rely on probabilities instead of exact methods. Although finding an event like using a certain chat application in mobile screen recordings is different to finding scenes containing a male lion in a nature documentary for example. Mobile screens have a fixed structure and therefore make the use of rather simple computer vision methods like template matching or perceptual hashes possible, in contrast to more advanced methods that are necessary to classify a lion in wildlife videos. Current related work uses manual sighting to analyze video material or template matching and OCR on videos from the desktop platform. By exploiting the fixed structure of mobile applications our approach saves computation time compared to desktop approaches. The determination of the similarity of certain fixed image areas using perceptual hashes is sufficient to identify events. Our approach struggles when detecting UI elements on changing backgrounds. We use edge detection to extract the foreground, but how we use this does not produce reliable results in our evaluation. Template matching was barely used in our event list as most areas of interest of our example events were at fixed positions. This stresses the difference between mobile and desktop UIs. In our approach template matching produces better results with the high-quality videos compared to our test runs with low video quality. Using OCR on transition frames from switching between GUI states are causing the largest amount of corrupted log messages. This needs to be addressed from a perspective of how these situations can be identified to prevent using OCR on these frames as tesseract delivers accurate results when used on clear video images.

Our test runs indicate that the proposed method can deliver detailed log files. For long-term use on mobile devices the file size of video data should be low. For this reason, we test different video qualities with smaller file sizes. Missing an event does not necessarily correlate with a low frame rate or bit rate. However lowering the framerate lowers the number of occurrences of short events on video frames and therefore the chance of detecting them. Comparing the results for very short events like pressing the k-key shows, that even with low quality and frame rate a detection is possible. Besides keeping the file size low on the mobile site it is necessary to optimize the recognition process for using our approach at larger scale. The video preprocessing showed that all tested video variations contained a lot of frames that are redundant for our recognition. This leads to the question of whether this step should be implemented earlier: already when recording the screen of a mobile device.

Privacy
In contrast to very transparent self-reporting methods or logging of more general mobile system events, screen recordings or screenshots can contain very sensitive and personal information. Our approach possibly analysis everything that occurs on the user’s screen. As a side-effect, this implies the need to develop an effective privacy concept that ensures that users can stay in control of which events they share with who. Following the Privacy by Design [8] approach, this needs to be addressed right from the beginning. Although in the current stage of development the room for abuse scenarios is small, it is important to take this into account and to make sure that participants screen data stays private in further studies.

Limitations
Our log files show that videos of 5 FPS are sufficient for a range of our event definitions and can work for short interactions. But to use this bit and frame rate in a large scale study, more test runs and further development are needed. Besides that, we use video recordings from using the phone in portrait mode. Changing the video orientation to landscape would not work with the event definition list used for our test runs. We worked with absolute numbers defining the bounding boxes for events. For landscape videos the same event list would have to be checked and adjusted accordingly to the changed GUI and aspect ratio. Working with relative values instead might solve this to some degree. One strength and weakness of our approach is that almost everything that happens on the screen can be tracked. However, events that can be triggered by the user via hardware buttons, such as the volume or skipping of a music track, cannot be detected.

CONCLUSIONS
In this paper, we present a concept, implementation and evaluation of a method to analyze mobile application usage by automatically generating log files based on mobile screen recordings. We showed that exploiting characteristics of mobile screen recordings is possible. The following list sums up our key contributions.

- The proposed logging approach can provide fine-grained log files on in-app events and general interactions independent of modifying the source code of any application.
- In contrast to similar desktop approaches it is possible to exploit the structured GUI of a mobile device: many elements appear at fixed positions of the screen. We compare the similarity of fixed areas of the screen with fixed areas of the event definition to successfully identify a wide range of events. Our video preprocessing eliminates redundant frames and thereby speeds up the recognition process.
- We show that using perceptual hashes, template matching and OCR are appropriate to identify events in mobile screens to create log files. Recognizing elements on changing backgrounds using edge detection does not work reliably in our approach.
- We compare the log results of different video qualities. A wide range of events is detectable successfully even in low bit and frame rate videos. Resulting log files are missing several of our example events depending on the specific event and its definition.

The source code and event list of this project is available open source [16].
REFERENCES

1. Pekka Aho, Teemu Kanstren, Tomi Räty, and Juha Röning. 2014. Advances in Computers. Academic Press.

2. Matthias Böhmer, Brent Hecht, Johannes Schöning, Antonio Krüger, and Gernot Bauer. 2011. Falling Asleep with Angry Birds, Facebook and Kindle: A Large Scale Study on Mobile Application Usage. In Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '11). ACM, New York, NY, USA, 47–56. DOI: http://dx.doi.org/10.1145/2037373.2037383

3. Florence Balagtas-Fernandez and Heinrich Hussmann. 2009. A Methodology and Framework to Simplify Usability Analysis of Mobile Applications. In Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering (ASE '09). IEEE Computer Society, Washington, DC, USA, 520–524. DOI: http://dx.doi.org/10.1109/ASE.2009.12

4. Carol M. Barnum. 2010. Usability Testing Essentials: Ready, Set...Test! Elsevier.

5. Barry Brown, Moira McGregor, and Donald McMillan. 2014. 100 Days of iPhone Use: Understanding the Details of Mobile Device Use. In Proceedings of the 16th International Conference on Human-computer Interaction with Mobile Devices & Services (MobileHCI '14). ACM, New York, NY, USA, 223–232. DOI: http://dx.doi.org/10.1145/2628363.2628377

6. Johannes Buchner. 2018. imagehash: A Python Perceptual Image Hashing Module. (Jan. 2018). https://github.com/JohannesBuchner/imagehash

7. J. Canny. 1986. A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-8, 6 (Nov. 1986), 679–698. DOI: http://dx.doi.org/10.1109/TPAMI.1986.4767851

8. Ann Cavoukian. 2011. Privacy by Design - The 7 Foundational Principles. (2011).

9. Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller. 2010. GUI Testing Using Computer Vision. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI ’10). ACM, New York, NY, USA, 1535–1544. DOI: http://dx.doi.org/10.1145/1753326.1753555

10. Trinh Minh Tri Do, Jan Blom, and Daniel Gatica-Perez. 2011. Smartphone Usage in the Wild: A Large-scale Analysis of Applications and Context. In Proceedings of the 13th International Conference on Multimodal Interfaces (ICMI '11). ACM, New York, NY, USA, 353–360. DOI: http://dx.doi.org/10.1145/2070481.2070550

11. Tesseract Open Source OCR Engine. 2018. (2018). https://github.com/tesseract-ocr

12. Christian Frisson, Sylvain Malacria, Gilles Baillly, and Thierry Dutoit. 2016. InspectorWidget: A System to Analyze Users Behaviors in Their Applications. In Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA ’16). ACM, New York, NY, USA, 1548–1554. DOI: http://dx.doi.org/10.1145/2851581.2892388

13. Maaike van den Haak, Menno De Jong, and Peter Jan Schellens. 2003. Retrospective vs. concurrent think-aloud protocols: Testing the usability of an online library catalogue. Behaviour & Information Technology 22, 5 (Sept. 2003), 339–351. DOI: http://dx.doi.org/10.1080/0144929031000

14. Jürgen Kawalek, Annegret Stark, and Marcel Riebeck. 2008. A New Approach to Analyze Human-mobile Computer Interaction. J. Usability Studies 3, 2 (Feb. 2008), 90–98. http://dx.doi.org/10.1145/2835562.2835566

15. Philipp Krieter. 2018a. Android background screen recorder. (2018). https://github.com/pkrieter/android-background-screen-recorder/

16. Philipp Krieter. 2018b. Mobile screen recordings to log file. (2018). https://github.com/pkrieter/mobile-screen-recordings-to-log-file/

17. Andre W. Kushniruk and Vinma L. Patel. 2004. Cognitive and usability engineering methods for the evaluation of clinical information systems. Journal of Biomedical Informatics 37, 1 (Feb. 2004), 56–76. DOI: http://dx.doi.org/10.1016/j.jbi.2004.01.003

18. Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. 2017. Research Methods in Human-Computer Interaction. Morgan Kaufmann. Google-Books-ID: hbkxDQAAQBAJ.

19. Florian Lethner and Clemens Holzmann. 2012a. Automated and Unsupervised User Interaction Logging As Basis for Usability Evaluation of Mobile Applications. In Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia (MoMM ’12). ACM, New York, NY, USA, 118–127. DOI: http://dx.doi.org/10.1145/2428955.2428983

20. F. Lettner and C. Holzmann. 2012b. Sensing mobile phone interaction in the field. In 2012 IEEE International Conference on Pervasive Computing and Communications Workshops. 877–882. DOI: http://dx.doi.org/10.1109/PerComW.2012.6197635

21. Donald McMillan, Moira McGregor, and Barry Brown. 2015. From in the Wild to in Vivo: Video Analysis of Mobile Device Use. In Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI ’15). ACM, New York, NY, USA, 494–503. DOI: http://dx.doi.org/10.1145/2785830.2785883

22. OpenCV library. 2018. (2018). https://opencv.org/
23. Niels van Berkel, Chu Luo, Theodoros Anagnostopoulos, Denzil Ferreira, Jorge Goncalves, Simo Hosio, and Vassilis Kostakos. 2016. A Systematic Assessment of Smartphone Usage Gaps. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI ’16). ACM, New York, NY, USA, 4711–4721. DOI:
http://dx.doi.org/10.1145/2858036.2858348

24. Arnold P. O. S. Vermeeren, Effie Lai-Chong Law, Virpi Roto, Marianna Obrist, Jettie Hoonhout, and Kaisa Väänänen-Vainio-Mattila. 2010. User Experience Evaluation Methods: Current State and Development Needs. In Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries (NordiCHI ’10). ACM, New York, NY, USA, 521–530. DOI:http://dx.doi.org/10.1145/1868914.1868973