Informatics

Software Komponenten zur Implementierung von Multimedia Applikationen für Android

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Lukas Naske, BSc
Matrikelnummer 01426015

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dr.techn. Hilda Tellioğlu
Mitwirkung: Mag.rer.soc.oec. Dr.rer.soc.oec. Roman Ganhör

Wien, 22. August 2020

Lukas Naske   Hilda Tellioğlu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Informatics

Software components supporting the implementation of multimedia applications for Android

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Lukas Naske, BSc
Registration Number 01426015

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dr.techn. Hilda Tellioğlu
Assistance: Mag.rer.soc.oec. Dr.rer.soc.oec. Roman Ganhör

Vienna, 22nd August, 2020

Lukas Naske	Hilda Tellioğlu

Erklärung zur Verfassung der Arbeit

Lukas Naske, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 22. August 2020

Lukas Naske

Danksagung

Ich möchte mich bei Hilda Tellioğlu für die Betreuung dieser Diplomarbeit bedanken. Besonderer Dank geht an Roman Ganhör für viele großartige Diskussionen und Hilfestellungen sowie für das umfangreiche Feedback, ohne dem diese Diplomarbeit nicht entstehen hätte können. Außerdem möchte ich mich bei meinen Eltern und bei meiner Freundin Lea bedanken, die mich nicht nur während dem Schreibprozess dieser Diplomarbeit sondern während meines gesamten Studiums immer unterstützt haben.

Auch allen, die die MicroDO Software-Bibliothek getested haben und die Probleme der Software-Bibliothek aufgzeigt haben, gebührt außerordentlicher Dank.

Acknowledgements

I want to thank Hilda Tellioğlu for being the advisor for this thesis. Special thanks go to Roman Ganhör for many hours of discussion and support and the amazing feedback while reviewing this thesis. Also, I want to thank my parents and my girlfriend Lea for the support not only during the process of writing this thesis but during all of my studies.

In addition, I want to thank everyone who participated in testing the MicroDO software library and provide valuable feedback.

Kurzfassung

Moderne Smartphones werden mit jeder Generation an neuen Geräten leistungsfähiger. Deshalb wird die Bearbeitung von Multimediainhalten, wie zum Beispiel das Bearbeiten von Videos, direkt auf den Smartphones immer relevanter. Die Designprinzipien von Desktop-Applikationen sind nicht immer auf den Kontext von mobilen Smarpthone Applikationen übertragbar. Das kann vor allem auf die unterschiedlichen Eingabe- und Ausgabemodalitäten von Smartphones und Desktop-Computer zurückgeführt werden, wie Gestensteuerung statt Keyboard und kleiner tragbarer Bildschirm statt großen stationären Monitor. Trotzdem versuchen zum Beispiel Videobearbeitungsprogramme auf Smartphones bekannte Konzepte aus Desktop-Videobearbeitungssoftware zu replizieren. Im Rahmen dieser Diplomarbeit wurden mit qualitativen Methoden aus dem Bereich des User-Centered-Designs neue User-Interaktionen und Design-Elemente für die Arbeit mit Multimediadateien am Smartphone identifiziert und abstrahiert. Dann wurden die abstrahierten User-Interaktionen und Design-Elemente in einer Software-Bibliothek mit dem Namen MicroDO zusammengefasst und unter Beachtung moderner Software-Qualitätsstandards für die Smartphone-Platform Android implementiert. Im Anschluß daran haben erfahrene Programmiererinnen und Programmierer vorbereitete Beispiele unter Verwendung der MicroDo Software-Bibliothek implementiert. Während der Implementierung wurden die Programmiererinnen und Programmierer beobachtet und danach wurden Semi-Strukturierte Interviews mit ihnen durchgeführt und ausgewertet. Dieser Implementierungstest zeigte, dass die MicroDO Software-Bibliothek eine modulare und flexible Software-Bibilothek ist, welche sich gut in bestehende Android-Entwicklungsumgebungen einfügt und hilfreich für die Implementierung von innovativen Multimedia-Applikationen sein kann.

Abstract

The performance of smartphones increases with each generation. Because of this, editing multimedia content directly on smartphone devices gets more and more relevant. The design principles of desktop and mobile applications differ in many cases because smartphones have a smaller screen and use touch-gestures for input instead of using mouse and keyboard. But, for example, mobile video editing applications on smartphones often rely on the same metaphors and design decisions of their desktop counterparts. Within this thesis new user interaction and user interface designs were identified and described using qualitative methods from the research field of user-centered-design. Then a software library called MicroDO was implemented, which supports the implementation of the identified user interactions and user interface designs on the Android operating system. To evaluate the implementation, experienced Android developers were asked to implement small tasks using the MicroDO software library and describe any issues they found within the software library. In addition, semi-structured interviews were held with the developers after they finished implementing the tasks. The feedback of the developers and the analysis of the semi-structured interviews showed, that the MicroDO software library is a flexible and modular software library that can support the implementation of innovative multimedia applications on Android.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Background 1
1.2 Problem definition 2
1.3 Expected Results 3

2 State of the art **5**
2.1 User Interaction on Smartphones 5
2.2 Development on Android 6
2.3 Sorting 10
2.4 Video Editing 11
2.5 New ideas for Video editing 14
2.6 Componentization of Software 20

3 Methodology **21**
3.1 Thematic Analysis 21
3.2 Requirements Engineering 22
3.3 Agile Software Development 23
3.4 UserX Story 25
3.5 Semi-Structured Interviews 27
3.6 API Usability Test 28

4 Implementation **31**
4.1 Requirements Analysis 31
4.2 UserX Stories 37
4.3 Implementation of the MicroDO library 46

5 Evaluation **57**
5.1 Design of the MicroDO API usability test 57

5.2 Conducting the API usability test 61
5.3 Feedback Analysis 66

6 Results **73**
6.1 Results of the API usability test 73
6.2 Categorisation of the Usability Issues 77

7 Conclusion **81**
7.1 Summary 81
7.2 Critical Reflection 82
7.3 Future Work 84

List of Figures **85**

List of Tables **87**

Bibliography **89**

Attachements **A1**
Usability Test Introduction A1
Usability Test Scenario 1 . . . . A3
Usability Test Scenario 2 . . . . A6
Usability Test Scenario 3 . . . . A9
Usability Test Scenario 4 . . . . A12
Usability Test Scenario 5 . . . . A15

CHAPTER 1

Introduction

This chapter introduces the topic of this thesis. The background and motivation for this thesis are described. This is followed by the definition of the problem space and a discussion of the results expected from completing this thesis. Also, the research questions this thesis is built upon are stated.

1.1 Background

The performance of smartphones increases with each generation. Using these mobile devices for consumption and creation of multimedia content has become a familiar use case for users. With the increasing power of these devices, editing the created content right on the device gets more and more relevant. But editing multimedia content, like for example videos, right on the device still does not provide the experience users expect and because of this, it is rarely done, especially in a professional context.

There exist multiple social media applications like Instagram, Snapchat and Facebook where videos that have been filmed on mobile phones are posted regularly. Companies monetize these videos and a whole market has been created around videos for mobile consumption. But instead of being able to quickly edit videos directly on the device, they are transferred onto a PC and then edited and posted from there. So, creating a more enjoyable user experience could improve the workflow.

Software for editing multimedia content like videos is mainly developed for PCs. Because of this user interactions for the editing process of e.g. videos have been designed with mouse and keyboard in mind. Shortcuts using the keyboard are also used to improve productivity. This leads to problems as user interaction design concepts for mouse and keyboard are not always applicable in the mobile context. Because of this, the well-known user interactions from PC video editing cannot be mapped easily. As Schoeffmann et al. (2015) state, the usage of tablets and smartphones increased very rapidly over the last few years. They also emphasize that mobile devices enable new user interactions through the integrated touchscreens and the integrated sensors like for example motion. All of this leads to the necessity of new user interactions with the context of mobile phone usage in mind.

While mobile devices provide new forms of input in comparison to common desktop computers, the screen size is far more limited on smartphones in comparison to desktop computers. This leads to problems with e.g. showing both control elements and the content to edit on the same screen. As a result, user interface designers have to make compromises.

Another challenge that still exists is the limited power and storage compared to modern desktop computers. But the gain in performance over the last years shows a trend that smartphones will grow even stronger and these limitations might be able to be overcome in the future. Cobârzan et al. (2014) showed that mobile devices are capable of taking over image processing tasks from desktop computers, even though they still lack some of the performance. Still, it is necessary to create user interfaces that respond fast and give understandable feedback such that the user can easily comprehend the current state of the editing process.

1.2 Problem definition

In a user study, Puikkonen et al. (2009) investigated when and why videos are created and edited. They found out, that taking videos was a major use case for smartphone users. However, users rarely edited the created videos right on their mobile devices. While there exists research on how to improve the user experience when editing multimedia content on smartphones, implementing prototypes is still a challenge. Especially with regards to adapting old user interfaces to new improved technologies to compare prototypes with each other. All of this points to a software library that supports the development of prototypes for new user interaction and user interface ideas as a valuable achievement.

There exist prototypes from prior research, but a real working implementation is still missing for many of them to be able to re-evaluate the proposed user interactions with real users. The different implementations are also hard to compare, as they are programmed as proof-of-concept and not as reusable components. Another aspect that makes comparison difficult, is the different hardware standards that were available during the implementation of the proofs-of-concept.

Of course, researchers already use software libraries when implementing novel interaction ideas, however, software libraries are often targeted at technical aspects and not at aspects concerning user-interaction. We found, that a software library that focuses on mobile multimedia user-interaction could be supportive of the advances in this research area.

Google supplies developers with a fully functioning media player[1], the editing of videos is not supported by this player by default. There exist multiple apps for video editing in the play store, like “InShot” [2]. But software libraries to include the editing of videos or other multimedia content in new apps are scarce and often no longer supported. For example “Media for Mobile”[3] has been created to support multiple multimedia editing processes but has not been updated for the last two years. Because of this the development of freely available components to support the creation of prototypes for new user interactions on Android devices can benefit researches and developers of new multimedia editing applications.

1.3 Expected Results

The goal of this thesis is to find similarities in existing prototypes for newly proposed user interactions for the arrangement and ordering of multimedia items on smartphones, find abstract descriptions for those similarities, group them accordingly and implement them as standalone software components. The implemented components will be usable to create prototypes for new user interactions for video editing on smartphones. This will help researches to not only implement new ideas for user interactions on smartphones but make the created prototypes easier to compare.

Each of the implemented software components will represent a small part of a user interface or a user interaction and will be designed with reusability in mind. By using the created components in combination, it should be possible to create new prototypes for user interactions on Android[4] devices. Even though the focus of this thesis lies on the arrangement and sorting of multimedia items, some of the created components might be usable in different contexts.

The software will be created for Android smartphones using the programming language Kotlin[5]. Even though it would be possible to use Java[6] to implement Android applications, Kotlin is the current standard for implementing new software for Android smartphones.

The implementation will be guided using modern quality assurance standards. Since it is one of the goals of the thesis to make the software usable by other developers, the documentation of the software is important. To integrate the implemented software components into other applications, detailed descriptions are required on how to use the provided software. Advice on when to use each of the software components for user interaction will be part of this description. In addition, since the software modules should

[1] `https://developer.android.com/guide/topics/media/mediaplayer` (Accessed: 18.08.2019)
[2] `play.google.com/store/apps/details?id=com.camerasideas.instashot` (Accessed: 09.06.2020)
[3] `https://github.com/INDExOS/media-for-mobile` (Accessed: 09.06.2020)
[4] `https://www.android.com/` (Accessed: 13.08.2019)
[5] `https://kotlinlang.org/` (Accessed: 13.08.2019)
[6] `https://www.java.com/` (Accessed: 13.08.2019)

be extendable and maintainable possible in the future, documentation of the code using KDoc[7] style comments in the code base will be performed.

Another important aspect of this thesis is to gather valuable feedback from developers regarding the usability of the created software components with regards to using them to create new prototypes for the rearrangement and ordering process of videos on smartphones. Because of this, a usability test will be designed and developers will be asked to implement given user interfaces. The results of the usability test will be analyzed and will provide the basis for future improvements of the created software components.

In conclusion, the thesis will focus on answering the following research questions:

RQ1: What is missing in current software-frameworks to better support the implementation of media-intensive user-interfaces on Android?

RQ2: How can these missing features in current software-frameworks be identified and defined?

RQ3: How can the software for these missing features be designed to be flexible and modular?

[7]`https://kotlinlang.org/docs/reference/kotlin-doc.html` (Accessed: 13.08.2019)

CHAPTER 2

State of the art

The following chapter gives a brief overview of the main topics of this thesis. First, a short introduction to the history of smartphones and the standards of user interaction on smartphones is given. Then the principles of developing software applications for Android devices are described and the sorting elements in general and on mobile devices are discussed. This is followed by a short introduction into the topic of video editing and descriptions of research done in the field of new ideas for video editing on mobile devices. Also, a quick overview of the principles of componentization of software applications in general, is given at the end of this chapter.

2.1 User Interaction on Smartphones

Before smartphones with touchscreens and a large number of applications that can fulfill a variety of use cases were introduced, mobile phones mainly consisted only of number buttons and a few additional buttons for navigation through lists. With the technology for mobile internet connections improving, mobile phones also started to evolve to adapt to new functionality available through internet connectivity. In 2007 the first iPhone[1] was introduced by Apple, which was revolutionary for using a touchscreen instead of hardware buttons and supported web browsing via the Safari browser and downloading audio and video content via the iTunes store, which is a web-based dedicated content and application store. While the functionality of the first iPhone was still limited, it was the start of a still growing market of new smartphone devices. There were multiple attempts to provide the same experience as Apple's iPhone by other manufacturers. But, only with the introduction of the Android operating system by Google a variety of manufacturers were able to establish a market of devices and custom software applications that was competitive to the Apple ecosystem. With each iteration of new smartphones being

[1]`https://www.apple.com/iphone/` (Accessed: 17.08.2020)

developed, smartphones get higher performance, new sensors and functionality and better and more cameras. (Cecere et al., 2015; Han and Cho, 2016)

The input on a smartphone is mostly carried out through its touchscreen. By using one or multiple fingers different forms of input can be performed. A list of the gestures that are commonly performed on smartphones to interact with user interfaces and suggestions by Saffer (2009) on when to use those gestures can be seen in Table 2.1.

Next to the gestures mentioned in the table above, modern touchscreens also allow the detection of “free-form gestures”(Wörndl et al., 2013), which means that any kind of form could be used to perform interactions, such as squares, circles or also letters. In addition, most smartphones have a few hardware buttons on the case to perform simple actions such as locking the smartphone and changing the volume of the speakers among others. (Wörndl et al., 2013)

Smartphones are also often equipped with sensors, that allow the detection of the current orientation of the smartphone. Through using the information from the sensors, users can interact with a user interface by e.g. shaking, rotating or tilting the smartphone. Some smartphones also support the detection of squeezes of the smartphone to trigger functionality. (Wörndl et al., 2013)

2.2 Development on Android

Android is an operating system mainly used for mobile systems. Development on Android started in 2003 by Android Inc., which was purchased by Google in 2005[2]. According to StatCounter (2020) the market share of Android in the context of mobile operating systems is around 70%. Android was created as an open-source project and is used by many smartphone manufacturers. Even though there exist projects and devices that use a desktop version of Android known as Chrome OS[3] on notebooks and desktop computers, it is mainly used and designed for usage on smartphones and tablets. There also exists support for integrated systems in cars[4], smart TVs [5] and wearable devices like smartwatches which use a dedicated version called Wear OS[6].

The programming language Android was primarily developed in was Java. However, lately the programming language Kotlin is the recommended language for Android development. Kotlin is a rather new programming language with its first release in 2016. Kotlin and Java are fully compatible, which allows developers to use both programming languages within the same software application. This is possible because the Kotlin compiler also compiles to Java byte-code by default. The benefits of Kotlin are its compatibility with many platforms such as Android and website development. The compatibility allows for

[2] https://en.wikipedia.org/wiki/Android_(operating_system) (Accessed: 17.08.2020)
[3] https://developer.android.com/chrome-os (Accessed: 17.08.2020)
[4] https://www.android.com/auto/ (Accessed: 17.08.2020)
[5] https://www.android.com/tv/ (Accessed: 17.08.2020)
[6] https://wearos.google.com/ (Accessed: 17.08.2020)

Gesture	**Defintion**	**Recommondation of use**
"TAP"	"The tip or pad of the finger touches the surface briefly (<100 milliseconds). A double tap performs this gesture twice rapidly, with a <75-millisecond pause in between the two contacts."	Use a tap to start functionality or select something
"DRAG/ SLIDE"	"The tip or pad of the finger moves over the surface without losing contact with the surface. Use for drag-and-drop and scrolling."	To move something on the screen or to scroll through a list
"FLICK ("FLING")"	"Flick can be done in two ways. In the first way, the finger is crooked to start, and then the tip of the finger or part of the finger pad brushes the surface briefly (<75 milliseconds) as the finger uncurls. In the second way, the finger is straighter and the movement is nearly reversed, with the finger drawing closer to the body and the fingertip or part of the finger pad brushing the surface. Both of these are also called Fling."	To move something on the screen or to scroll through a list in a fast manner
"'NUDGE"	"The pad of a straight (index) finger slides briefly (<2 seconds) forward."	To move something.
"PINCH"	"Two fingers (typically the thumb and index finger, although it can be two fingers from either hand or even two fingers on two different hands on multitouch surfaces) move closer together."	To decrease the size of something or to reduce the zoom level
"SPREAD"	"Two fingers (typically the thumb and index finger, although it can be two fingers from either hand or even two fingers on two different hands on multitouch surfaces) move farther apart."	To increase the size of something or to increase the zoom level
"HOLD" ("Press")	"The tip or pad of the finger is pressed onto the surface for an extended period of time."	To select something or to perform continuous scrolling

Table 2.1: Gestures used on touchscreens. The gesture names and definitions under quotation marks are taken from Saffer (2009, Appendix A). The recommendations of when to use the gestures are rephrased from Saffer (2009, Chapter 3).

the usage of many existing libraries. The design goal of Kotlin was to reduce the amount of code compared to Java to achieve the same results. (Kotlin Foundation, 2020)

Multiple challenges arise when developing software applications for Android. One challenge arises due to the fact that a wide variety of different devices from different manufacturers use the Android operation system, which leads to differences in performance and the underlying hardware. Also, there exists a wide range of screen sizes and pixel densities of the touchscreen. Next to the differences in hardware, many different versions of Android are used. Older devices often do not get updates to newer versions of the mobile operating system. To overcome the mentioned problems, Knych and Baliga (2014) suggest various methods to ensure that the developed applications can be tested and how these tests should be structured to make locating errors easier. The main focus of the research was to improve the automated testing of Android applications. By using an automated test suite that is run directly on Android devices, a variety of devices with different versions of the Android operating system can be tested in quick succession without the need for manual interaction with the mobile devices.

Another challenge that occurs when developing Android applications in comparison to desktop applications is the importance of application state handling. While desktop applications get started and remain started until they are closed, an Android application can be in various states. These states represent that the application is starting or being stopped, but also if the application is currently not in the foreground because a separate application was opened over the currently running one. This change of states is also referred to as the “Activity Lifecycle”(Google, 2020), as the software component responsible for handling the state changes and visualizing the application is referred to as an `Activity`. This “Activity Lifecycle” can be seen in Figure 2.1.

On application startup, the main activity gets created, and the `onCreate()` callback is called by the system. Within the `onCreate()` callback anything that has to be done only once while the application is not completely removed from the memory should be done. For example, the XML file defining how the application looks like can be loaded within this method. Then the `onStart()` callback is called, which is also called if the application was not completely removed from memory but only minimised by the user. Within the `onStart()` callback, all resources that have to be available for application use have to be allocated and the data to be displayed should be loaded to let the user interact with the application. After all, resources are allocated, the `onResume()` callback is called, which represents that the application entered the state of running and the user can interact with the application. Within the `onResume()` callback, resources, that should and can only be available while the app is running in the foreground, should be allocated. An example of such a resource would be accessing the Camera of the Android smartphone. When the application is currently not in the foreground anymore, the `onPause()` callback is called. Here all resources that were allocated within `onResume()` should be freed again. As an example, if access to the Camera was requested in `onResume()`, the Camera access should be stopped in the `onPause()` method. If the app was only paused because of a notification or the usage of a different application in multi-window mode, the `onResume()` method is called again and the application keeps running. In case the application was closed and is no longer visible, the `onStop()` callback is called, which represents that the application was closed. Within this callback, all allocated resources within `onStart()` should be freed again and the state of the application should be stored if necessary. If the application was only minimised and opened again, the `onRestart()` method is called which is followed by the `onCreate()` callback again. The `onRestart()` method can be used if the app should behave differently on application restart then on application creation. In case the application was destroyed either by the user or the system, the `onDestroy()` callback is called and all allocated resources should be freed again if they were not freed before. The lifecycle begins from the `onCreate()` callback. In some cases it can happen, that the application process is killed without calling `onPause()` or `onStop()`, which is why the application state should be saved whenever possible if changes were made to ensure that no information gets lost. The challenges the "Activity Lifecycle" bring, are that developers have to very careful when to allocate resources and how to ensure that no information gets lost, especially as applications can be killed without all lifecycle methods being called. In addition, when e.g. the user changes the orientation of the phone and the application supports different orientations, the application gets destroyed and the lifecycle is run through again. This often leads to issues as resources get allocated multiple times or resources that were expected to be already loaded are in fact not loaded anymore. (Google, 2020)

By conducting qualitative research with developers of mobile applications, Francese et al. (2017) found that testing is one of the big challenges of developing apps for Android. But they also identified that UX design is a key part of developing successful applications. While they were also able to identify that agile workflows are the standard development methodology for mobile applications, they were not able to state one approach to be the most fitting one.

To create software that is usable by other developers, it is important to rely on modern quality standards. Documentation is a vital part of these quality standards, as it describes how components work, without having to understand the actual source code. In some cases, the creation of documentation takes up to 20% of the development time. This documentation can be done both inside the source code using comments as well as outside of the source code by using any kind of external written text. Comments within the source code are mostly focused on the technical description of the software components and can be used directly during development. The comments can be left out if a good naming scheme for classes, methods and variables is used. Some styles of commenting also make it possible to create external files that can be published as websites for example. This makes it possible to access the documentation without access to the source code. The external documentation for a software framework can be done in various forms but is often done in so-called "Readme" files. The Readme files include a common overview of the functionality in a less technical description and often also describe how to get started with using the software framework. (London, 2003)

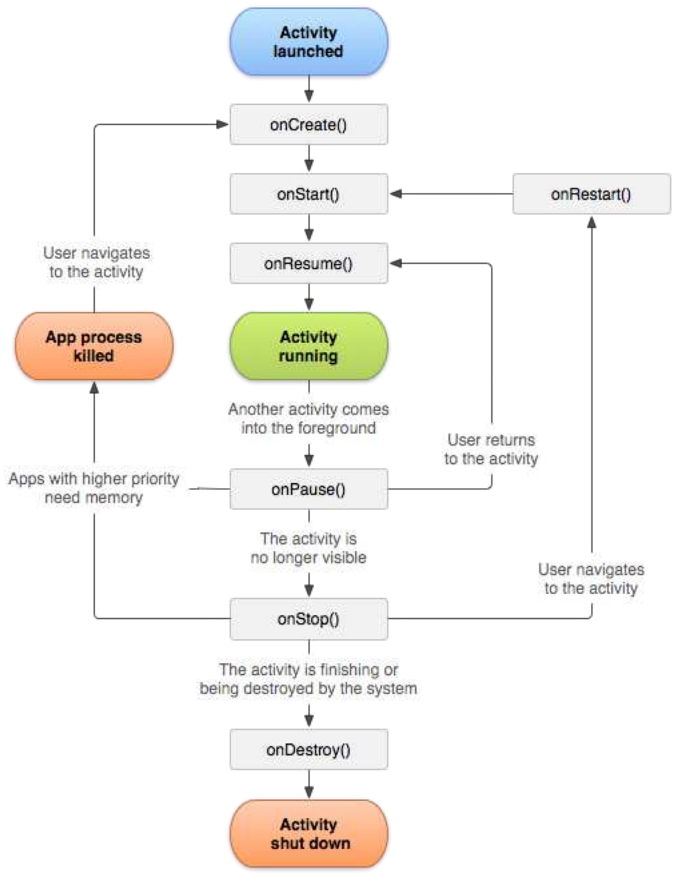

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 2.1: The activity lifecycle of an Android applicaiton. (Google, 2020)

2.3 Sorting

When using desktop computers or mobile phones the users often acquire a big amount of data files. Similar to the real world, it is in many cases wanted to bring those files into some kind of order and to categorise similar things together. This process of bringing order into a set of files or the data within a file can be seen as sorting the files or their content. Sorting can be done automatically based on the metadata of files, for example the creation date or the name of the file, or through detection specific features within the content of the file and putting these features into relation with each other. For example, when sorting pictures, algorithms can detect specific colors or people within the pictures to categorise and sort the image files. The automatic detection of similarities and differences of files is constantly improved by applying new improved algorithms or through using machine learning. (Drucker et al., 2011; Darwaish et al., 2014)

Because of the limitations of automatic sorting, users are often required to sort and categorise the files on their devices manually. Especially on mobile devices, the limitation of the screen size can lead to issues when sorting and categorising large amounts of files. For example, users often use their smartphones to take a lot of photos and through storage expansions or cloud storage usage the amount of photos that can be accessed on a single device constantly grows larger. The data to sort is often displayed in lists but they can mostly only be scrolled horizontally or vertically. One solution to overcome the limitation of screen size would be to allow the users to scroll both vertically and horizontally and sort the files based on different parameters on the 2 dimensions. Another solution as proposed by Ganhör and Güldenpfennig (2015) was to split the manual sorting process into separate steps and to provide proper forms of user interaction for those steps. The authors proposed that sorting consists of "browsing, selecting and filing"(Ganhör and Güldenpfennig, 2015, p. 346). They implemented a novel user interface that allowed the user to browse through images on different levels of detail and then select the exact ones for the filing process. But sorting images is not only limited to be done in lists, as Ott et al. (2012) show in their research where multidimensional interfaces are proposed and tested. (Panizzi and Marzo, 2014)

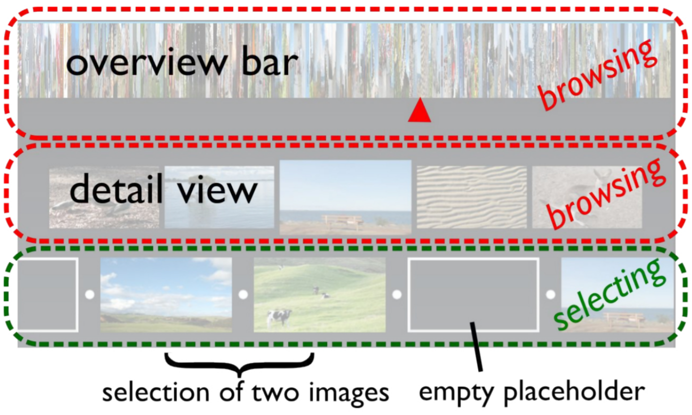


Figure 2.2: The interface proposed by Ganhör and Güldenpfennig (2015, p. 346). The top two rows allow for different detail levels of browing, while the bottom row is used to select images to be filed back into the center browsing row.

2.4 Video Editing

According to the "Merriam-Webster.com" online dictionary, editing with regards to a video or a film is defined as

> "to assemble (something, such as a moving picture or tape recording) by cutting and rearranging" (Merriam-Webster.com, 2020)

The result of video editing is a new video, which is either the final product or part of further video editing processes. Within the definition, it was already mentioned, that an integral part of the video editing process is the rearrangement of videos and sections of those videos to create the wanted result. To start the arrangement process the videos that should be edited have to be selected. This selection process can result in a single video or multiple videos to be used. During the selection process, the videos might already be sorted and ordered for the arrangement process. Depending on the number of videos used for editing, the videos might also be grouped to ease finding the correct scene during the whole video editing process. A schematic representation of the video editing process can be seen in Figure 2.3

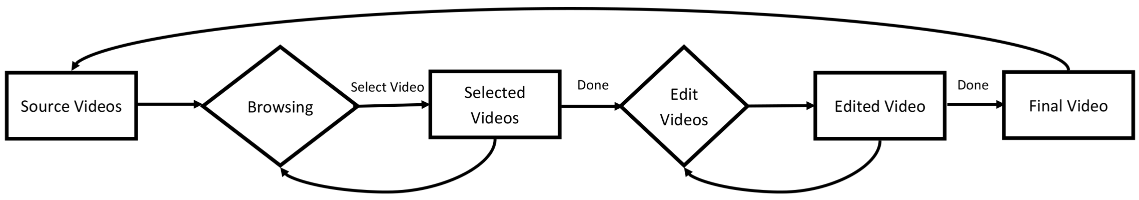


Figure 2.3: The video editing process. The source videos are first browsed through and selected and then edited. The final video can be the source for another video editing process.

Video editing software is well established on desktop computers, with a variety of commercial and free software tools. Some well known programs are “Adobe Premiere”[7], “Apple Final Cut Pro X”[8] or “Apple iMovie”[9]. All of the mentioned use a timeline metaphor in their user interface to arrange the videos on. The videos are arranged by adding them to and removing them from the timeline. Parts of single video files can be cut out of videos. Over time this process results in the final video, which in turn can be used in a different video editing process for further modifications.

As an example, within “Adobe Premiere 2020“ the videos to be edited have to be imported into the project before they can be edited. For this, the “Media Browser”, which gives the user access to the PC file system, can be used or the videos can be added via drag and drop from the desktop file system. The imported videos can then be sorted into “Bins” that function like folders in the desktop file system and as such support the grouping of videos. The videos can be viewed as list elements with names or - as can be seen in Figure 2.4 - as elements previewing the content of the video files. Additional functionality is available in the “Assembly View” of “Adobe Premiere”, which can be accessed in the large preview of the selected video that can be seen on the right of Figure 2.4. For example, videos can already be added into a “Sequence”, which represents the timeline in “Adobe Premiere”.

[7]https://www.adobe.com/at/products/premiere.html (Accessed: 07.05.202)
[8]https://www.apple.com/final-cut-pro/ (Accessed: 07.05.202)
[9]https://www.apple.com/imovie/ (Accessed: 07.05.202)

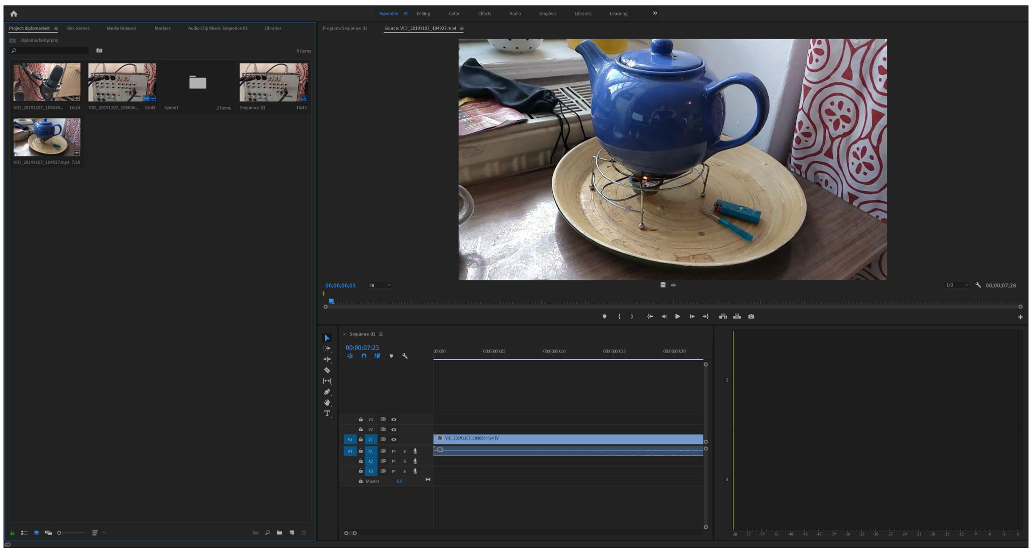


Figure 2.4: Assembly view in Adobe Premiere 2020 (Created on 14.11.2019)

In contrast to desktop video editing applications, mobile video editing applications apply a more simple approach to the selection and rearrangement process of videos. The top three mentioned apps when searching for “Video Editor” on the “Google Play Store”[10] “YouCut - Video Editor”[11], “InShot”[12] and “VideoShow”[13] all have limited functionality with regards to the arrangement and sorting of videos. When videos are selected, they are arranged on the timeline in the order they were tapped on. This order can be seen at the bottom of the screen on the left in Figure 2.5. The order can be modified on the selection screen by long clicking on an element and dragging it into a different position. On the timeline itself, the videos can be rearranged and additional functionality is available to edit the video on the editing screen of the application - which can be seen in on the right of Figure 2.5. The sorting and arrangement functionalities do not allow the user to sort and group the files without taking immediate effect on the timeline itself. As a result, a user would have to use a different application or access the file system directly to do be able to do this process if wanted. The functionality for selecting, sorting and rearranging the videos is similar in all three mobile video editing applications.

[10]`https://play.google.com/store` (Accessed: 14.11.2019)

[11]`https://play.google.com/store/apps/details?id=com.camerasideas.trimmer` (Accessed: 14.11.2019)

[12]`https://https://play.google.com/store/apps/details?id=com.camerasideas.instashot` (Accessed: 14.11.2019)

[13]`https://play.google.com/store/apps/details?id=com.xvideostudio.videoeditor` (Accessed: 14.11.2019)

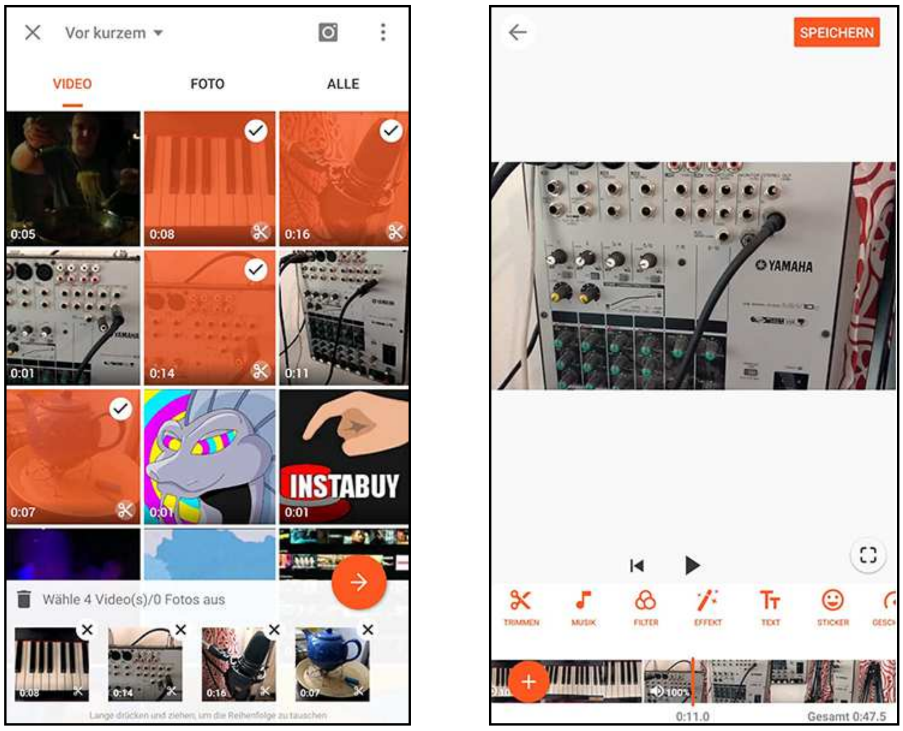


Figure 2.5: Screenshots of selection and editing view of "You Cut - Video Editor" (Created on 14.11.2019)

2.5 New ideas for Video editing

With the introduction of modern smartphones research on how to bring better user experience to video editing on the smartphone was conducted as well. Even before modern smartphones with touchscreens were part of everyday usage, Jokela et al. (2007) investigated the possibilities of video editing on mobile phones. The mobile phone used in their experiment had a small "2.2-inch 65,000-color graphical display"(Jokela et al., 2007, p. 345), low pixel density of "176x208 pixels"(Jokela et al., 2007, p. 345) and did not have touch input on the screen. But as it already had a camera integrated that could record videos, the need for video editing on the device was identified. All input had to be done via the keyboard on the device, which only contained numbers, navigation buttons for direction and a few extra buttons for selection and cancellation. Regardless of these restrictions Jokela et al. (2007) succeeded in providing a prototype to edit videos on the mobile phone. The video editing application consisted of 8 different user interface screens for selecting videos, previewing videos, editing the videos and a special view for cutting a video. An overview of the screens can be seen in Figure 2.6. The separation of the cut and edit view was done to reduce the complexity of the edit view on the small screen. The videos were arranged on the edit view using the timeline metaphor. The edit view supported the arrangement of videos, adding transitions between videos, adding effects to videos and also allowed to mute the original audio of the videos and to add audio from audio files.

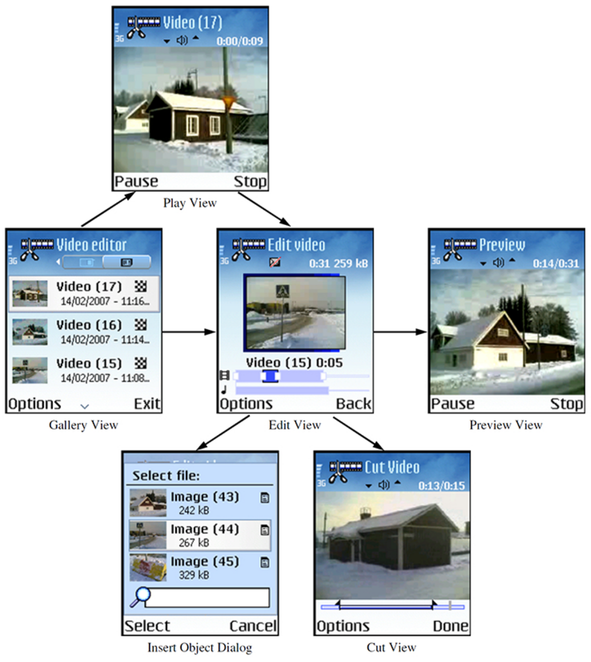
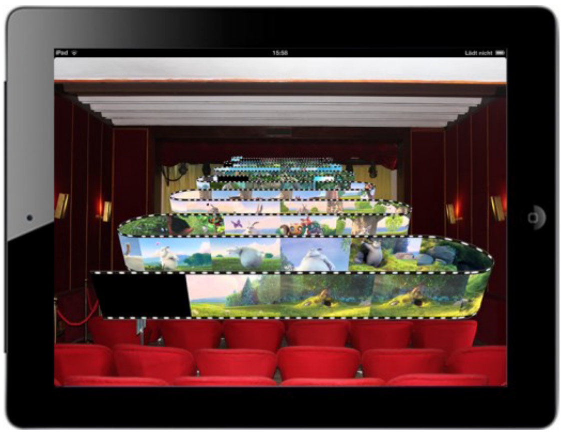
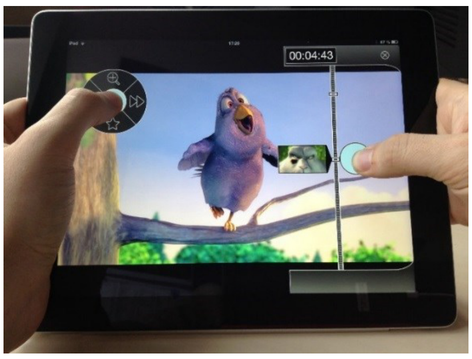


Figure 2.6: Overview of the screens of the video editing application by Jokela et al. (2007, p. 348)

Image browsing is the process of searching through a set of images to find the wanted one. Video browsing, in contrast, is the process of looking for a specific section or frame within a video. Hudelist (2013) focused on the possibilities to improve image and video browsing on modern smartphones with touchscreens. Sorting and grouping the videos automatically or manually was identified as a valuable improvement to image browsing on smartphones. It was shown that using 2D or 3D visualizations can improve the usability of software applications significantly. For example, the files of the images or videos to browse through should be displayed on a rendered globe, cylinder or in a grid. Combining automatic sorting and advanced visualization techniques lead to even higher improvements in usability. With regards to video browsing, an approach to browse a video on the visualization of a 3D filmstrip was given and later further investigated, which can be seen in Figure 2.7a. This film strip showed sections of the video to browse through and was scrollable using touch input. (Hudelist et al., 2013a)

In a later approach Hudelist et al. (2013b) implemented a video browsing interface called “ThumbBrowser” (Hudelist et al., 2013b, p. 405). This user interface was used in the landscape mode of smartphones and enabled the control of the seekbar vertically on the side. In most applications, the seekbar is displayed horizontally similar to a timeline. In addition to the vertical seekbar, a radial menu is used to provide further functionality. The radial menu allowed for bookmarking timeframes within the video and fast-forwarding and fast-reverting of the currently played video. There is also the option on the radial menu to zoom in on the seekbar to make it easier to find an exact frame using the vertical seekbar on the right. The positioning of the controls on both sides of the horizontally held smartphone made it possible to have interactions with content using both hands at the same time while holding the device. The "ThumbBrowser" user interface is depicted in Figure 2.7b.


(a) The 3D Filmstrip for video browsing by Hudelist et al. (2013a, p. 299).



(b) The "Thumbbrowser" user interface. (Hudelist et al., 2013b, p. 348)

Figure 2.7: The new ideas for video browsing by Hudelist et al. (2013a,b)

There exists more research on improving video browsing on mobile devices by Ganhör (2012). With "Propane" (Ganhör, 2012, p. 1) a user interface was designed to improve finding an exact frame in a video using the touchscreen on a smartphone. This interaction can be performed using only the thumbs in landscape mode of the smartphone, similar to the approach of Hudelist et al. (2013b). While other video editing applications use a seekbar to browse through a video, "Propane" lets the user browse a video by touching the side areas of the screen. When the left area of the screen is touched the currently played video gets played in reverse, when the right area is touched the video gets played forward. The user can slide up and down the areas on the sides to increase or decrease the playback speed. Depending on where the user begins the touch, the playback speed can be increased or decreased by more or less granular steps. As an example, if the user starts on the top of the side areas, the playback speed can be decreased in multiple steps while it cannot be increased at all. The opposite applies if the user starts the touch gesture on the bottom of the side areas, where the playback speed can then only be increased but not decreased. If the user starts the playback in the center of the side areas, the playback can be increased by sliding up and decreased by sliding down, but only in less detailed steps. The idea of starting the touch gesture in the center can be seen in Figure 2.8. By giving the user the possibility to browse through a video frame by frame or in faster than normal playback speed, the user interaction of finding an exact frame in a video was improved over other applications.

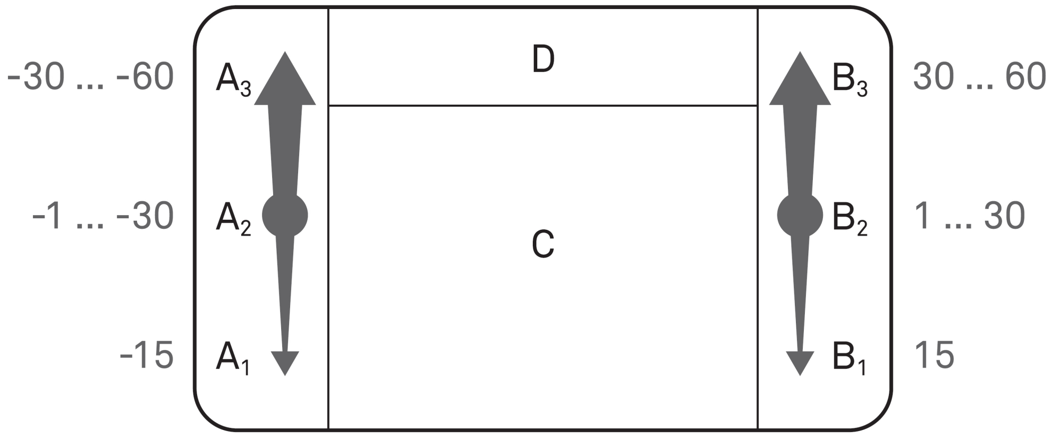
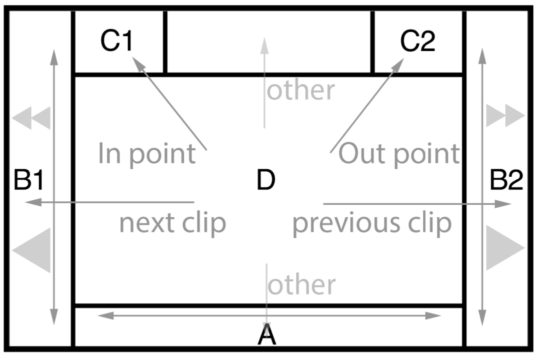


Figure 2.8: The progressive scrolling view of “Propane”. (Ganhör, 2012, p. 4)

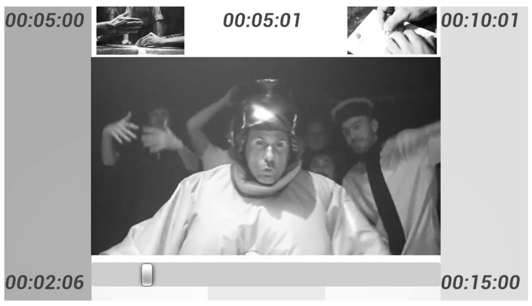
In a second paper by Ganhör (2014) another user interface was designed with the focus on trimming a video instead of only browsing them. The process of trimming a video is mainly based on setting the In-Point, the frame at which the video should start playing, and the Out-Point, the frame at which the video should stop playing, for a video. It was also mentioned that the previously implemented user interaction of video browsing is tightly coupled to video trimming, as the appropriate timestamps for setting the In-Point and Out-Point first have to be found. Because of this the user interface from “Propane” was reused and extended by adding areas for setting the In-Point and the Out-Point using swipe gestures. The new user interface that was designed within this research was called “Muvee” and can be seen in Figure 2.9. A user study was conducted using “Muvee” and compared it to a different existing mobile video editing application called “iMovie”[14]. The user study showed that the new ideas for video browsing and video trimming suggested within the paper are suitable for the context of video editing on a smartphone. “Muvee” does not implement user interactions that are known from desktop video editing applications and does not fully comply with “Fitts’ Law”(Ganhör, 2014, p.230), as the gestures necessary to fulfill tasks require to move the finger over long distances. The user study conducted within the research suggested that the user interactions used within “Muvee” are suitable for the context of mobile video editing. (Fitts, 1954)

The before mentioned user interfaces for video editing were designed for standalone applications primarily designed for smartphones and tablets. Dai et al. (2017) tried to implement a video editing user interface in a web browser. Web applications can be accessed both on mobile devices and on desktop computers. If well designed, the user

[14]https://www.apple.com/imovie/(Accessed: 07.08.2020)



(a) The "Muvee" user interface with the possible swipe gestures. (Ganhör, 2014, p. 232).



(b) The prototype of "Muvee". (Ganhör, 2014, p. 232)

Figure 2.9: The "Muvee" user interface. (Ganhör, 2014, p. 232)

interface of websites also adapts to the device used to access the website, and as such should also be usable on smartphones. Dai et al. (2017) investigated how to add so-called hotspots to videos using an HTML5 based video editor. Hotspots can be defined by a user to track objects or people in a video and display information when hovering over or clicking on these defined hotspots. A problem identified for this operation was the performance of the browser, as web browsers are not designed to do perform such a task in a timely manner. To overcome the limitations in performance, Dai et al. (2017) implemented a workflow where the processing was performed remotely and not in the browser itself. While their solution is not focused on the editing of videos on mobile devices, it is still important to consider as smartphones are also equipped with modern web browsers. Designing web pages responsively, which means that they can be used on both desktop workstations and mobile devices, has become a standard and as such video editing in a web browser could also be usable using smartphones.

With the limited performance of mobile phones in mind, Yu and Liao (2013) designed a solution to move video editing into the cloud. Watching video content that is streamed to a device is a very common use case and well optimised. But the researches identified that the process of transforming a video, searching for specific time frames and switching between multiple videos to edit requires a more versatile approach than just watching content at the given playback speed. They also mentioned, that having the editing process stored remotely makes it easier to resume work on a different device or could be used to enable multiple editors to work concurrently on the same project. Another way to overcome the restrictions of hardware performance on the devices is to optimize the used formats. By using different more efficient algorithms or more powerful devices, all of the video transformations could be done on the device itself. (Islam et al., 2006)

Zhang et al. (2014) designed new user interactions to explore videos by focusing on the content of the video instead of relying on the time aspect of multimedia content. Their approach was to create a graph like representation called "VideoGraph"(Zhang et al., 2014, p. 1123). The idea behind "VideoGraph" was to first etract the scenes of a video automatically. Then the scenes are put into relation with each other and a graph is built based on the relation. The final step was to find a fitting form of representing the single scenes. Figure 2.10 shows different forms of extracting the scenes, building the graph and displaying the scenes. The left images display the scenes that identified, which are represented by the different color bars on top of the scenes. The center images represent the graph that is generated, where each color corresponds to a scene. The right image then shows different forms of representing the graph. A single scene can be represented by a single frame or multiple frames. While still limited on many levels this approach can help video editors to find specific parts of a video easier.

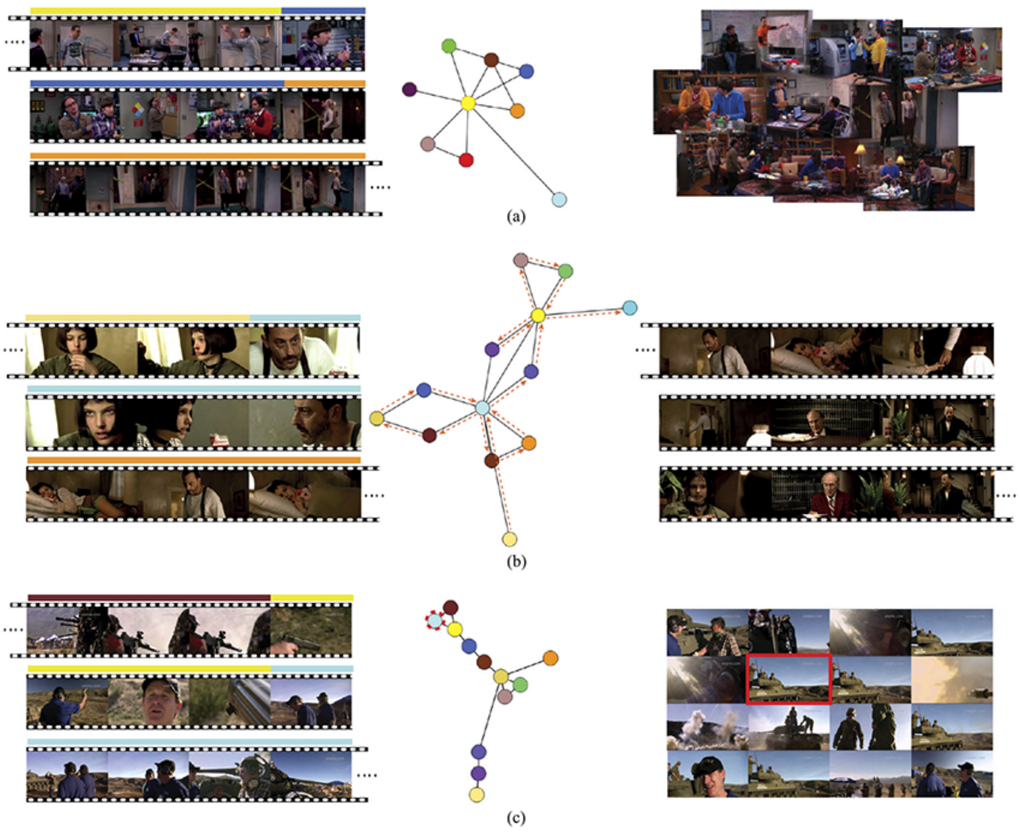


Figure 2.10: The "VideoGraph" depending on the used graph algorithm. (Zhang et al., 2014, p. 1129)

Most automated editing algorithms focus only on the image and audio of the recorded video. Taylor and Qureshi (2016) designed a framework that tries to automate video editing using the sensors found on a smartphone to gather information on how to edit the video. The accelerometer, magnetometer and gyroscope readings within others were used to collect data for the editing process. For example, data from the accelerometer can be used to identify parts of the video where the camera was not held stable and as such are likely to be removed. In addition to collecting data from the sensors while filming the video, the video gets processed afterwards. For this for example the brightness of the video was analysed as well as face detection algorithms used to find parts of the video with people in them. Also, the sections of the video that are not in focus are identified. All of this information is sorted into streams of data which can then be used during the video editing process to quickly remove parts of the videos with unwanted attributes, like being out of focus, or select parts of wanted content, like having the proper brightness levels.

2.6 Componentization of Software

In object-oriented programming, it is common practice to split the application into small components. This leads to better reusability of the same code within the same application as well as within other applications. While there exist many software components to create user interfaces, these software components for user interfaces often only cover basic functionality and still require a lot of programming to support complex user interactions. An example of a predefined software component for user interfaces on Android would be the `VideoView`, that is provided by Google. This `VideoView` can play the videos, but the user interface elements to provide controls for the user to interact with the video have to be implemented by the developer using e.g. button components. Karuzaki and Savidis (2015) implemented a library that eases the composition of UI elements in Java, which should also be usable on Android, called “YETI” (Karuzaki and Savidis, 2015, p. 1). They use a tree-like description of tasks to model how components interact and how they are contained in each other. This way the composition of the UI components can be taken over by the “YETI” library. Even though the “YETI” tool automates the composition, it is still possible to add other UI components. An important aspect they wanted to keep, was the modifiability of the components using the programming language. This should help the developer to keep control over the user interface and make the tool more flexible. While “YETI” is a promising approach to user interface reusability, it is only a research project and not used in production environments.

A problem that Paulheim and Erdogan (2010) identified is the heterogeneity of created software components. Over the years many new tools and programming languages were created and UI components developed in one language are often not be usable with UI components from another language. Even though it might be possible to display them in the same application the communication can be impossible. To overcome this problem Paulheim and Erdogan (2010) implemented an event bus that enables the communication between the components in a unified way. For example, they use proxy implementations to enable the drag and drop behaviour of objects between components that otherwise were not capable of communicating with each other.

CHAPTER 3

Methodology

This chapter describes the scientific methods used in this thesis. This includes an explanation of how to conduct a thematic analysis. Then requirements engineering, agile software development and the concept of "UserX Stories" are described. This is followed by a description of how to conduct an API usability test.

3.1 Thematic Analysis

Thematic analysis is a qualitative methodical approach to analyse a dataset. Thematic analysis is widely used in qualitative research and acknowledged as a scientific method. The benefit of a thematic analysis is its flexibility, as there are no restrictions on what the result of the thematic analysis should be. This benefit of flexibility is also a big disadvantage, as researchers who are using thematic analysis for the first time miss a clear guideline on how to conduct a thematic analysis. Through conducting a thematic analysis on a dataset, the "themes" of the dataset can be identified. "A theme captures something important about the data in relation to the research question and represents some level of patterned response or meaning within the data set"(Braun and Clarke, 2006, p. 10). The dataset itself can be collected in various forms, which include audio recordings, video recordings, text and also notes taken during the data collection. (Boyatzis, 1998; Braun and Clarke, 2006; Nowell et al., 2017)

To identify the themes, six phases are applied to reduce the content of the dataset to the essential parts and to find general descriptions of what can be deducted from the interviews. The six phases to apply are as follows.

The goal of phase one is to familiarize oneself with the data set. To accomplish this the dataset has to be read at least once. If the dataset consists of verbal interviews, it is suggested to transcribe the data as a first step. The first phase is vital even if the person who analyses the dataset took part in the collection of the data to get a complete overview of the data before starting the analysis. (Braun and Clarke, 2006)

In phase two, codes are assigned to sections of the interviews. “Codes identify a feature of the data (semantic content or latent) that appears interesting to the analyst”(Braun and Clarke, 2006, p. 18). These codes are the start of generalizing the statements and identifying common ideas within different interviews. The codes can, for example, be collected as notes within the dataset. The coding should be done for every part of the dataset with the same precision to find all relevant parts of the interviews. This can require the person conducting the thematic analysis to go over the dataset multiple times.

Phase three is used to find the themes within the codes. Codes are grouped under a theme when they have a similar meaning or represent a similar idea that can be expressed through a theme. Themes can relate to only small parts of the dataset, but themes can also be used to group other themes under a single theme. To achieve the grouping a mindmap can be used where codes are grouped into themes and subthemes. It is also possible to find a theme for the whole dataset. (Braun and Clarke, 2006; Attride-Stirling, 2001)

Within phase four the themes are revisited to refine the themes and eliminate, merge or split themes if necessary. If the data for a theme is not enough to support it as a standalone theme the theme might be eliminated. Some themes might be merged into a single theme if they represent similar ideas and the level of detail accomplished through separate themes is not required. It could also happen that a single theme has to be split into multiple themes if a higher level of detail for a theme is wanted. Each theme should be meaningful with regards to the whole dataset while being specific enough to capture the results from the dataset properly. The mindmap from phase three can be used and refined to generate the final themes. The refinement process of the themes can require multiple iterations over the themes, the codes and the dataset. (Braun and Clarke, 2006)

Phase five defines and describes the found themes in written form. With these definitions and descriptions, a final report about the found themes within the dataset and their relation to the research questions is written as the sixth and last phase of the thematic analysis. (Braun and Clarke, 2006)

3.2 Requirements Engineering

In software development, the features a system has to fulfill are often described as requirements. The field of research related to identifying and describing the requirements of a system is referred to as Requirements Engineering. The process of identifying the requirements of a system is called requirements analysis. To find the correct requirements a lot of different factors have to be taken into account. On the one hand, the technical requirements have to be defined. This means it needs to analysed, which platforms the software should be build for, which needs with regards to performance exist and any other factor that affects the technical design of the resulting system. On the other hand, the needs of the customer and the users have to be taken into account and defined as requirements. According to the "IEEE Recommended Practice for Software Requirements Specifications"(IEEE, 1998, p. iii) defining requirements helps to:

- "Establish the basis for agreement between the customers and the suppliers on what the software product is to do"
- "Reduce the development effort"
- "Provide a basis for estimating costs and schedules"
- "Provide a baseline for validation and verification"
- "Facilitate transfer"
- "Serve as a basis for enhancement"

Requirements can also be seen as goals or ideas. By referring to requirements as goals, the positive aspect of achieving part of the system when fulfilling a requirement is emphasized. The concept of calling the requirements ideas should take away the negative stereotypes that are attached to the word requirement. Ideas also help to encapsulate that requirements are subject to change and evolve with the system. (Nuseibeh and Easterbrook, 2000; van Lamsweerde, 2001; Mohanani et al., 2014)

When it comes to identifying requirements regarding the usability and user interaction of a system, more than just the system alone has to be taken into consideration. The needs of the user and the context the user uses the system in affect not only the requirements itself but also how the requirements should be written down. One form of describing requirements in a user-centric way is the format of user stories which is discussed later in Section 3.4. Buhne et al. (2004) also describe, how describing requirements from different points of view can help to keep track on how a requirement affects different parts of the system and the development process. (Harrison and Barnard, 1993; Cohn, 2004)

3.3 Agile Software Development

In software development projects requirements are often unclear or change during the development. Changing and unclear requirements make the planning of software projects difficult and often lead to discrepancies between what the development team delivers and what the customer expects. To mitigate the challenge of unclarity in software projects, agile software development strategies are often applied. The agile software development strategies are mainly based on the agile manifesto by Beck et al. (2001):

"**Individuals and interactions** over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan" (Beck et al., 2001)

The manifesto should encourage software developers to work in iterative software development cycles to create working software quickly, to welcome changes in the requirements and to react to them effectively and to heavily incorporate the customer in the development process. (Radinger and Goeschka, 2003)

Some well known agile software development strategies are “Kanban” (Sugimori et al., 1977, p. 553), “Scrum”(Schwaber and Beedle, 2001) and “Extreme Programming” (Beck, 1998, p. 1). The three mentioned development strategies are similar with regards to developing software in an iterative process but distinguish by the approaches taken to work effectively on the software development. Scrum is most well known for its planning of each short iteration, which is referred to as a sprint. Each sprint the development team plans what work should and can be achieved and on the exact date of the end of the sprint the iteration has to be done or the work is concluded to be not done. By reflecting on the amount of work a team was able to achieve during a sprint the estimations should get better over time. In extreme programming, sprints are also used but a lot shorter and the work done during a sprint can change based on the prioritisation. Following is a more detailed description of the Kanban software development process. Kanban was first introduced in the automobile industry and has been adopted by multiple other areas such as software development. When using Kanban the development is done in iterative cycles with the goal to release small parts of the software fast and rework the software if the requirements change. When a task has to be completed during development a ticket is created, which contains all the information necessary to complete the task. All tickets are collected in a backlog and prioritised. Based on this prioritisation the tickets are chosen for development one after the other. The most important artifact of the Kanban development cycle is the Kanban board. The Kanban board itself is split into columns, where each column represents a different state a ticket can be in. The states of the tickets are often similar to “In Development”, “In Testing”, “In Review” and others depending on the development flow of the project the tickets were created for. The tickets are getting added to and removed from the columns on the Kanban board based on the ticket’s current development state. To ensure that a ticket gets completely finished and will not remain in an almost finished state, like for example in the “In Review” state, a work in progress limit is applied to each of the columns. This work in progress limit is the maximum number of tickets that can be in the state represented by the column at the same time. The work in progress limits should force developers to move tickets to the end before other tickets can be further developed. This encourages the developers to react to anything that blocks finishing a ticket in a fast manner. Also, the work in progress limits should encourage developers to help out other developers to finish their tickets. (Rautiainen, 2004)

The Kanban board can be a real physical board with physical tickets being added to and removed from it. But there also exist a lot of software solutions for representing the Kanban board. This allows for integration with other systems like source code repositories. When using the Kanban board in integration with a source code repository, the id of the ticket the source code was produced for could be directly mentioned and linked. As an example Github[1] is a website that offers free and paid plans to use their services as a source code repository using Git[2] as source code management and version control framework. Github also offers the option to create so-called “Projects” for source code repositories. These projects are used to create tickets for tasks that have to be completed as “Issues”. If wanted, Github also provides the option to create a Kanban board where the tickets can be moved from one state to the other. Github does not support the enforcement of work in progress limits on the columns of the Kanban board. Anyways, the work in progress limits can be added to the names of the columns and the adherence to the work in progress limit of a column lies in the responsibility of the software development team.

3.4 UserX Story

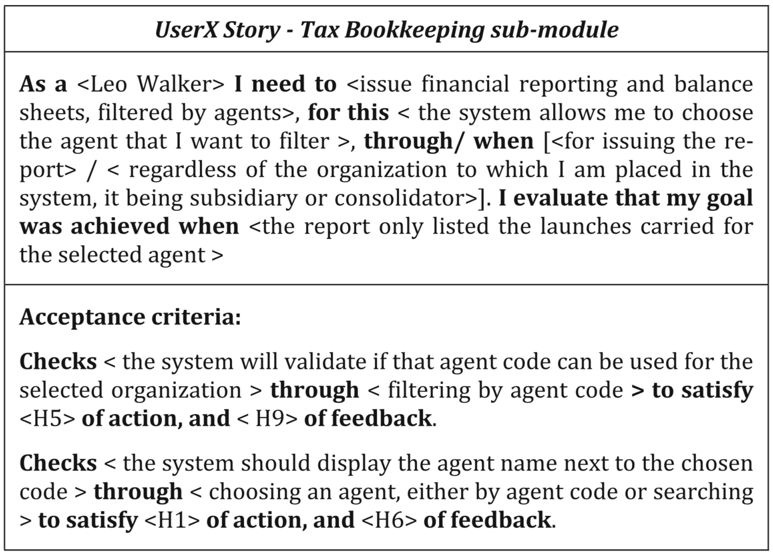
In agile software development, it is common to define the features to be implemented as so-called “User Stories”. User stories focus on describing what is needed by the user to accomplish a task without defining the technical details. Each user story should start by defining the kind of role the user that should accomplish the described task has, which is often done in the form of “As a <user-role>”. This way different kinds of users of the software can be differentiated. An example of this would be to differentiate between customers and administrators by starting the user stories with “As a customer” or “As an administrator”. The user story is then further developed by adding the task to be covered after implementing the user story. An example of a full user story would be “As a customer, I can log in on the website”. It is important to keep the user stories focused on tasks completed by the user and not to describe technical parts of the software application. An example of a poorly written user story would be “As a developer, I write the application using Java”. This poor example of a user story does not describe what a user of the software wants to accomplish by using the software, but only a technical detail about the implementation. Technical requirements can be added in a later description of the user story within the whole documentation, but should not be part of the actual user story. Lucassen et al. (2015) describe the qualities of a good user story to be “atomic”, “minimal”, “well-formed”, “conceptually sound”, “conflict-free”, “problem oriented”, “unambiguous”, “complete”, “explicit [in] dependencies”, “full sentence”, “independent”, “scaleable”, “uniform” and “unique” (Lucassen et al., 2015, pp. 127-129). (Cohn, 2004; Zeaaraoui et al., 2013)

A user story also includes the criteria to accept the user story as completed. This allows for testing an application based on the user stories. In addition to the acceptance criteria, any kind of further information can be added to a description of the user story. This can include meeting notes about the functionality of the software, requirements regarding the technical implementation or any kind of diagram to support the implementation. Any kind of information that supports either the development or the testing of the

[1] `https://github.com/` (Accessed: 01.06.2020)
[2] `https://git-scm.com/` (Accessed: 01.06.2020)

functionality described within the user story is valuable and should be added. A user story can always be updated if the requirements change or parts of the user story are vague or ambiguous and need further refinement. (Cohn, 2004)

A "UserX Story"(Choma et al., 2016, p. 131) is a special type of user story, which incorporates the user interaction as an essential part of the story. Similar to a user story, a UserX Story describes the task that has to be accomplished and the role the user has. However, the UserX Story adds a description of how the task is achieved through user interaction.



UserX Story - Tax Bookkeeping sub-module

As a <Leo Walker> **I need to** <issue financial reporting and balance sheets, filtered by agents>, **for this** < the system allows me to choose the agent that I want to filter >, **through/ when** [<for issuing the report> / < regardless of the organization to which I am placed in the system, it being subsidiary or consolidator>]. **I evaluate that my goal was achieved when** <the report only listed the launches carried for the selected agent >

Acceptance criteria:

Checks < the system will validate if that agent code can be used for the selected organization > **through** < filtering by agent code **> to satisfy** <H5> **of action, and** < H9> **of feedback**.

Checks < the system should display the agent name next to the chosen code > **through** < choosing an agent, either by agent code or searching > **to satisfy** <H1> **of action, and** <H6> **of feedback**.

Figure 3.1: Example of a UserX Stories written by a participant of the evaluation done by Choma et al. (2016, p. 138).

Thus, the UserX Story is split into five well-defined parts:

1. The description of the user's role, which starts similar to a user story with "**As a <Persona>**"(Choma et al., 2016, p. 136). "Personas" are a way of describing the role of a user by also taking the goals and motivation of the users into account and should encourage to really think about the users who will accomplish the described task. (Cooper et al., 2007)

2. The description of the goal that has to be achieved by completing the UserX Story. The goal of the UserX Story is described by adding "**I want/need <goal>**"(Choma et al., 2016, p. 136) to the story.

3. The third part of the UserX Story is the description of the interaction of the user with the application. This is achieved by adding "**for this <interaction>**"(Choma et al., 2016, p. 136) to the UserX Story.

4. In addition to the interaction, the context of the user interaction can be described by adding “**through/when [<task> /<context>]**”(Choma et al., 2016, p. 136) to the UserX story.

5. The last part of the UserX Story is the description of how the UserX Story can be evaluated. This is done by adding “**I evaluate that my goal was achieved when <feedback>**”(Choma et al., 2016, p. 136) to the UserX Story.

An example for a UserX Story as proposed by Choma et al. (2016) is shown in Figure 3.1. Another difference to user stories is the description of the acceptance criteria. User stories do not require a special format for acceptance criteria. The acceptance criteria of UserX Stories should include the “Nielsen’s Heuristics”(Nielsen, 1995), which are commonly used usability guidelines, that should be fulfilled by completing the UserX Story. UserX Stories help to take user interaction, user interface design and usability into account from the beginning of the development cycle. (Cohn, 2004)

3.5 Semi-Structured Interviews

There exist open, structured and semi-structured interviews. Open interviews do not follow a pre-planned interview structure while structured interviews follow a pre-planned structured very closely. The method of semi-structured interviews is common in qualitative research and combines the approaches of open and structured interviews. The idea behind semi-structured interviews is to prepare a set of questions but to only use them as a guideline and not as a predefined structure of the interview. By prioritising part of the questions the goal of the interview can still be achieved, while other questions can be omitted if the time frame set for the interview does not allow asking more questions. Also, it is important to follow up on answers of the interviewees by asking “How” or “Why”, especially if the questions are asked as closed questions. Through the open design of the interview, the interview can feel more like a conversation and as such might reveal unexpected outcomes. The acceptance of change in the interview process also relates to agile development and the prepared questions and their order should be iterated over regularly in between interviews. Especially after the first interview, it should be assessed whether the questions and their order have to be adapter to answer the research questions. (Leech, 2002; Adams, 2015)

The interviews can be recorded or notes can be taken during the interview to analyse the interviews. It can also be helpful to take notes even if the interview is recorded in case something the interviewee said that should be followed up later in the interview. After conducting the interviews, the process of analysing and extracting the results of the interviews begins. There exist many methods for analysing interviews. The scientific method of conducting a thematic analysis on the interviews was described before in Section 3.1. Burnard (1991) propose a more detailed method for analysing semi-structured interviews. A 14 step guide is provided on how to analyse the interviews which results in a report to be written in the end. (Adams, 2015)

3.6 API Usability Test

Usability tests are performed to identify problems within products and to get feedback from real users. Because of this usability tests are often used and researched in the field of human-computer interaction (HCI). Commonly, usability tests are performed on products that include some form of a graphic user interface (GUI) that the participants have to interact with. There are different forms of gathering information during the usability test, for example the "think aloud (TA) protocol" (Alhadreti and Mayhew, 2018, p. 1) can be used. When using the "TA protocol" the participants of the usability test are asked to say anything that comes to their mind, which includes also what they are thinking about while using the software and how they experience the used software. By having the participants express everything they think about, it should be possible to get a deeper understanding of how the users experience the software. If it is not possible or fitting to have the participants of the usability test say everything aloud during the usability test, t (Ericsson and Simon, 1984)

An application programming interface (API) is used by developers to access a different software component within their own software application. There are different forms of APIs. An API can be published by providing the compiled source code to be used directly within a software application. Another form of accessing an API is by calling published software application remotely via e.g. REST. While an API does most often not provide a form of a graphical user interface, its usability when used by developers can still be tested. Grill et al. (2012) suggest in their paper a way to conduct a usability test for their API, which was a software library to support researchers to "develop contextual study setups"[169](Grill et al., 2012). The authors combined a variety of different API usability test methods to test the API. The combination of different methodologies was done to minimise the disadvantages of other API usability test designs while getting the advantages of the replicated methods. Disadvantages of other API usability evaluation methods were their high cost, which made them only suitable for projects with high financial backing, or the requirement of special knowledge by the participants of the API usability test. Also, usability tests that rely solely on the evaluation through experts, which is easier done than tests with real users, do not reflect on the usage of the system by real users. For this reason, a combination of evaluating the API by experts but also have real users test the API on a smaller scale was designed. (Bore and Bore, 2005; Farooq et al., 2010)

The API usability test proposed by Grill et al. (2012) was done in 5 phases, which can be seen in Figure 3.2. In the first phase, the "Planning"(Grill et al., 2012, p. 168) phase, experts in the field and developers with experience in programming are recruited to take part in the API usability test. Programming tasks have to be designed in this primary phase, which the recruited developers should complete. Also, heuristics are chosen, based on which the API is evaluated.

Within phase two to four, the usability of the API is evaluated. The experts identify usability issues by doing a heuristic evaluation based on the heuristics chosen in phase

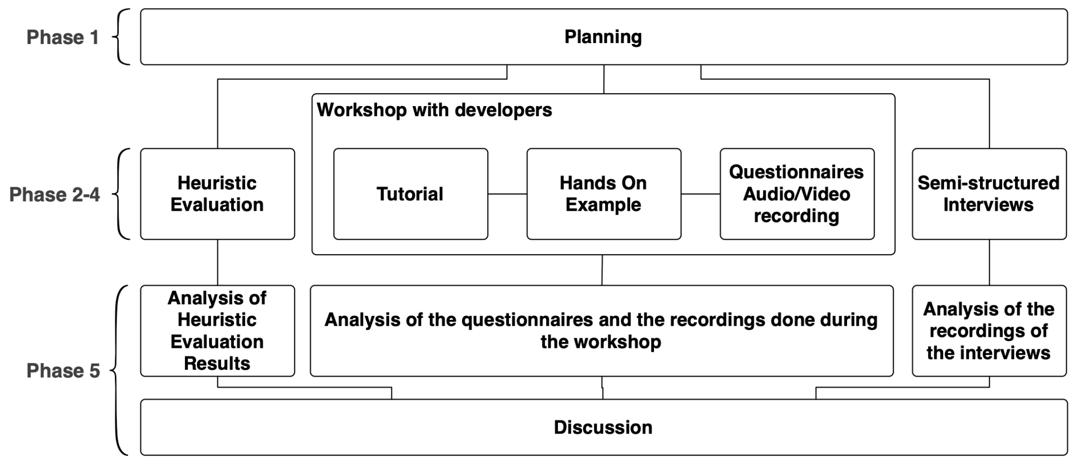

Phase 1
Planning
Workshop with developers
Phase 2-4
Heuristic Evaluation
Tutorial
Hands On Example
Questionnaires Audio/Video recording
Semi-structured Interviews
Phase 5
Analysis of Heuristic Evaluation Results
Analysis of the questionnaires and the recordings done during the workshop
Analysis of the recordings of the interviews
Discussion

Fig. 1. Methodological Approach

Figure 3.2: Visualisation of the different stages of the API usability test. (Grill et al., 2012, p. 168)

one. This is done to find and categorise issues based on standardized heuristics. Each expert covers only a small subset of the chosen heuristics to find issues for this exact heuristic in as many places as possible. The heuristics chosen for the heuristic evaluation in the case of Grill et al. (2012) can be seen in Figure 3.3.

In phase three the developers get to use the API to complete a set of tasks. This so-called "Workshop with Developers"(Grill et al., 2012, p. 167) starts with an introduction to the API and the collection of demographic data about the developers. This is followed by a set of small tasks to be implemented by the developers using the API that is tested through the API usability test. During the implementation, the developers are asked to fill out a questionnaire to note the found usability issues and voice and screen of the developers are recorded. After the workshop, the developers are asked to take part in a short interview no longer than 30 minutes to discuss the usability of the API, which is phase four of the API usability test. The interview takes place shortly after the API usability test to make sure the developers remember the found usability issues.

In phase five the collected data from the heuristic evaluation, the workshop with developers and the interviews are analysed and the found usability issues are documented. The found usability issues during the heuristic evaluation are categorised based on the heuristics they were identified for. The usability issues found by the developers during the workshop are also categorised based on the chosen heuristics. This allows for a comparison of the usability issues found by the experts and those found by the developers during the API usability test and during the interviews. The results of the API usability test are mainly qualitative, but if the number of developers and experts who take part in the API usability test is high enough, also quantitative conclusions can be made.

Name	Description
Complexity	An API should not be too complex. Complexity and flexibility should be balanced. Use abstraction.
Naming	Names should be self-documenting and used consistently.
Caller's perspective	Make the code readable, e.g. makeTV(Color) is better than makeTV(true).
Documentation	Provide documentation and examples.
Consistency and Conventions	Design consistent APIs (order of parameters, call semantics) and obey conventions (get/set mehods).
Conceptual correctness	Help programmers to use an API properly by using correct elements.
Method parameters and return type	Do not use many parameters. Return values should indicate result of the method. Use exceptions when exceptional processing is demanded.
Parametrized constructor	Always provide default constructor and setters rather than constructor with multiple parameters.
Factory pattern	Use factory pattern only when inevitable.
Data types	Choose correct data types. Do not force users to use casting. Avoid using strings if better type exists.
Concurrency	Anticipate concurrent access in mind.
Error handling and Exceptions	Define class members as public only when necessary. Exceptions should be handled near from where it occurred. Error message should convey sufficient information.
Leftovers for client code	Make the user type as few code as possible.
Multiple ways to do one	Do not provide multiple ways to achieve one thing.
Long chain of References	Do not use long complex inheritance hierarchies.
Implementation vs. Interface	Interface dependencies should be preferred as they are more flexible.

Figure 3.3: Heuristics chosen by Grill et al. (2012, p. 171)

CHAPTER 4

Implementation

This chapter focuses on the implementation done for this thesis. First, it is described how the requirements were analysed using thematic analysis and defined using UserX Stories. This is followed by a description of the implemented software library which was named MicroDO and what challenges were faced during the development.

4.1 Requirements Analysis

To identify what is missing in current software frameworks to better support the implementation of new media-intensive user interactions, it was first necessary to identify new ideas for user interactions and user interfaces. This was achieved by conducting a thematic analysis on 144 student submissions for the course "Interface & Interaction Design" on the topic of sorting of multimedia elements on mobile phones. The goal of the exercise of the students was to reflect on how they organise the photos on their smartphones and how the sorting process could be improved. For this, a set of sketches had to be drawn and discussed. By knowing what kind of new ideas for user interactions and user interfaces exist, it was possible to identify how the implementation of the new ideas could be supported.

The student submissions were analysed by conducting a thematic analysis as described in Section 3.1. The submissions were first read to get an overlook of the dataset. In phase 2 of the thematic analysis 25 codes have been identified, which can be seen in Table 4.1. The codes were found by repeatedly reading through the student submissions and noting what kind of ideas for sorting and rearranging of multimedia files on smartphones the students mentioned.

The codes were then analysed to identify the themes. The themes show which ideas for user interaction and user interfaces exist and as such build the basis to answer RQ1. The codes were analysed and checked if the proposed user interaction or user interface

Autosort	List-To-List
Sidescrolling	Folders
Multiselect	Multimove
Select and Move	Parking
Preview	Tagging
3D Visualization	Swipe Select
Corner Buttons	Selected State Animation
Favorite	Order By Selection
Lassoselect	Radial Menu
Alternating List	Map Visualization
Circular List	From Preview To List
Non Linear List	Swap 2 Elements
Zoomable	

Table 4.1: Codes found during thematic analysis

element can be easily implemented using the Android framework or how it could be supported. A mind map has been created to visualize the initially identified themes of user interactions and user interface elements that were identified to benefit from a new software library supporting their implementation. The themes have also been grouped into subthemes in a mindmap to get a better overview, which can be seen in Figure 4.1.

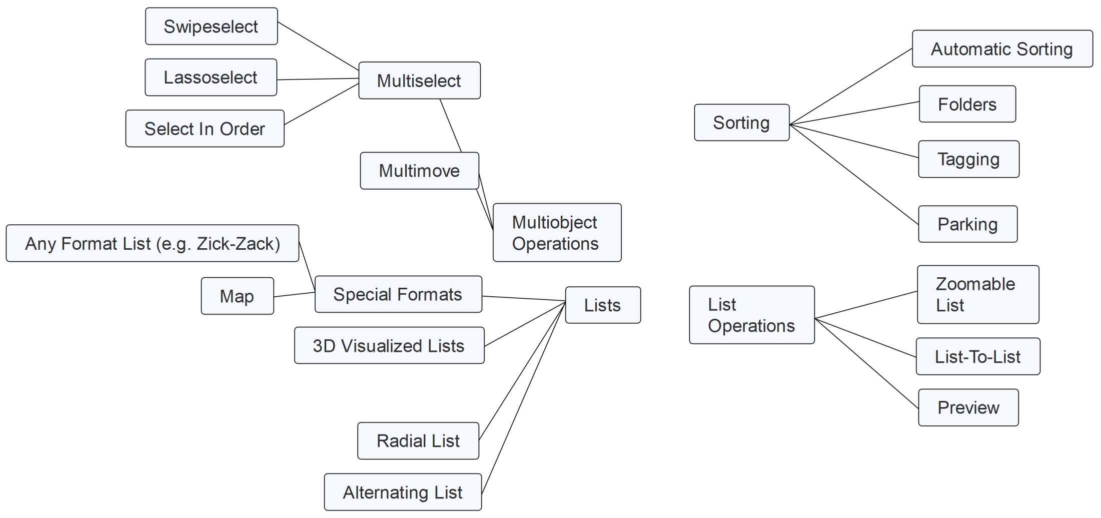


Figure 4.1: Mindmap of themes and subthemes after phase 3 of the thematic analysis of the student submissions.

The themes were then discussed with an expert in the field. This discussion showed that the themes of “Lassoselect”, “Select In Order”, “Any Format List”, “Map”, “Special Formats”, “3D Visualized Lists”, “Radial List”, “Automatic Sorting”, “Folders”, “Zoomable List” and “Preview” should be dropped. In addition to the remaining themes of “Swipeselect”, “Multimove”, “Alternating List”, “Tagging” and “List-To-List” the themes of “Corner Gestures” and “Side Tags” were added. The theme “Preview” has been removed, as a component for this theme already exists as it can be represented by the “VideoView”[1] component provided within the Android framework. The integration of a preview with a list to preview videos from would couple the components very tightly and as such make the software component harder to be reused in various ways. Also, the themes regarding the “Radial List” and the “3D Visualized List” were dropped as they serve very specific implementations. The “Special Formats Lists” theme lacked data to support a single implementation to be chosen and was, consequently, also dropped. The theme “Lassoselect” was dropped in favor of supporting “Swipeselect”. The theme of “Select In Order” was also dropped as the “Swipeselect” theme covered a more challenging use case, and it was decided to cover only one form of element selection for this thesis. The theme of folders was removed because the storage of modern smartphones is still organised the same way as PC storage systems, which already are folders. Also, the theme of automatic sorting was removed since the progress of sorting elements via algorithms is not user interaction based and as such is not as relevant for answering the research questions of this thesis.

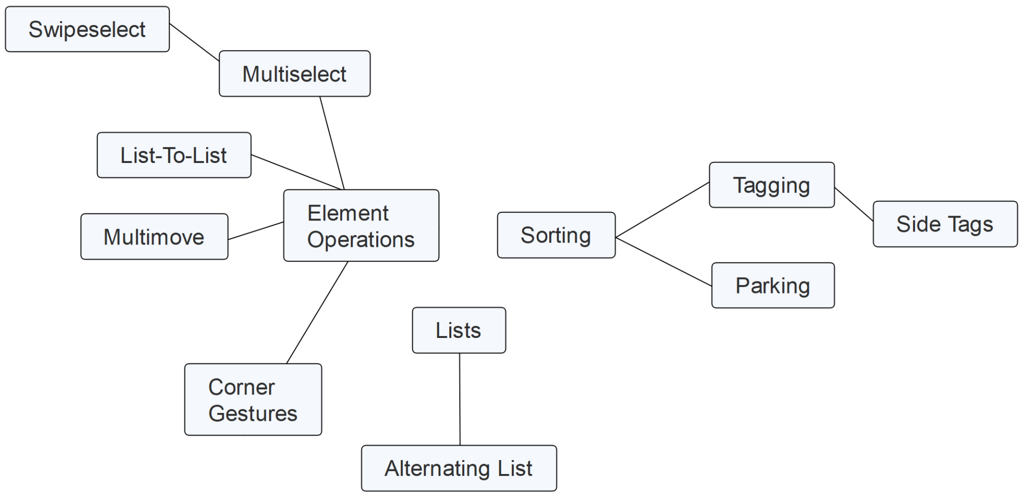


Figure 4.2: Mindmap of themes and grouped into subthemes after phase 4 of the thematic analysis.

It was identified by the expert that the theme of “Multimove” and “Parking” are very relevant for the research within the field despite not being mentioned by the students as often. The theme of “Tagging” is mostly focused on adding the functionality to add meta information in form of tags to a multimedia file. There are various ways to visualize the tags on the multimedia elements in the user interface. As no form of visualization was heavily supported by the data of the submissions, a variant suggested during the

[1] `https://developer.android.com/reference/android/widget/VideoView` (Accessed: 06.06.2020)

discussion with the expert was chosen. It is called "Side Tags" and shows up to 4 tags visualized via colors on the four sides of a multimedia element. The main focus of this theme will lie in enabling the usage of tags for video files at all. With the support of the "Corner Gestures", which came up a lot during the discussion with the expert and was mentioned and coded as "Corner Buttons" but not identified as theme initially, a high number of use cases can be covered.

At the end of phase four, the reduction of the themes led to the final seven themes that were found by the thematic analysis and the discussion with the expert. A mind map showing the found themes can be seen in Figure 4.2. The choice of themes supports a variety of use cases by having a new form of visualizing elements in a list, a way to interact with elements, ways select elements and drag and drop them and a way to add more information to elements.

To clarify what exactly is meant by the names of the themes, phase five of the thematic analysis requires the themes to be defined and described. This can be seen in the following enumeration. Each theme is also accompanied by a sketch of the user interface element or user interaction to be supported by the theme.

T1 **Alternating List**: The Alternating List theme represents a layout for a list, that supports arranging elements in an alternating fashion. Instead of aligning all elements in the list on the same height, the elements get displayed having an offset in height. This can be done both in vertical or horizontal directions. The Android framework supports different orientations and alignment methods for lists by default, but none match the design requested within the Alternating List theme. Also in open-source libraries, no support for such a list layout was found.

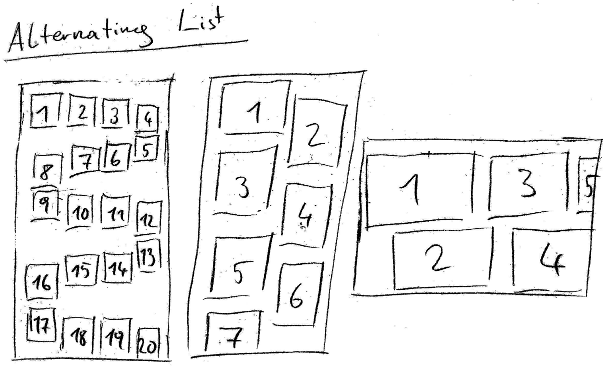


Figure 4.3: Sketch of the Alternating List theme

T2 **List-To-List:** This theme represents an operation on a list element, with which it should be possible to drag elements from one list to another. The removal and insertion of elements into the lists should be properly animated. It should also be possible to use this operation to rearrange elements within the same list. Android supports the drag and drop of user interface elements, but it requires a lot of code and is not trivial to implement. As such supporting and easing this behaviour on lists is beneficial.

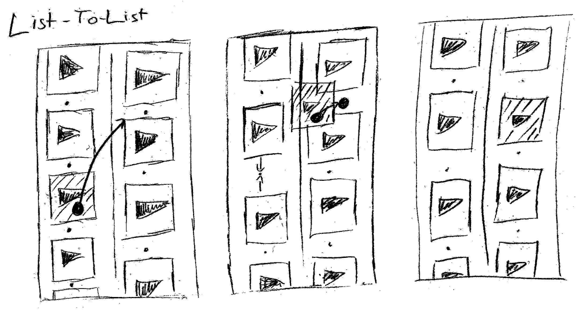


Figure 4.4: Sketch of the List-To-List theme

T3 **Parking:** The theme of Parking represents a user interface element, where elements can be placed during the rearrangement process without forcing them to stay within a list-like form. Which can be seen as parking the element within a user interface element. In addition, it should not be restricted to what type of element can be parked in the area, but a copy of the original representation should be used to visualize the element. As for the List-To-List theme already mentioned, the drag and drop behaviour for user interface elements is supported by Android. But a component just accepting any kind of user interface element to be parked and later moved to a different location does not exist at the moment.

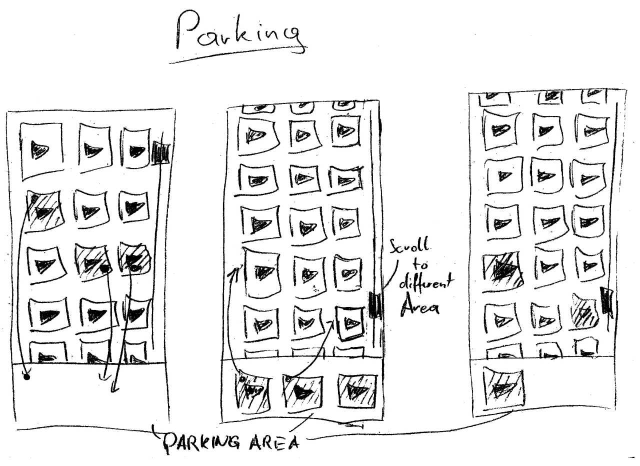


Figure 4.5: Sketch of the Parking theme

T4 **Tagging:** The Tagging theme should support the storage of meta information within the multimedia files. The meta information can be anything that can be represented by a character string. Media recording applications often support the addition of meta information like GPS location or date and time of the recording. The Tagging theme should enable the user to add any kind of meta information in as key value pairs. Adding the data directly to the files enables the user to share the data with other users without the necessity to use the same application to edit the videos, as long as a standardized key for storing the meta information is used. To visualize the tags added to a multimedia file in the user interface, the subtheme of Side Tags was found. For each tag, one of the sides of the displayed component is colored differently. Each side or color should represent a different tag and as such supports the display of up to four tags at the same time.

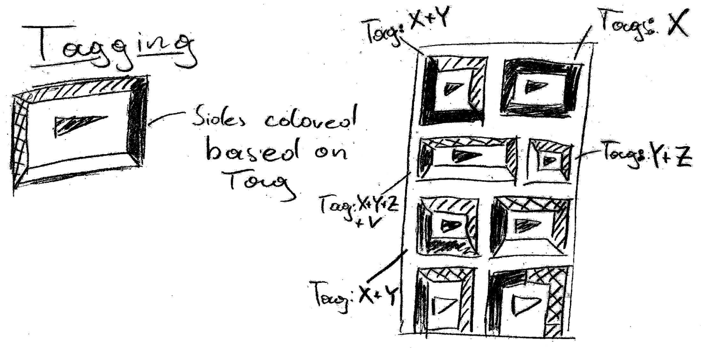


Figure 4.6: Sketch of the Side Tags theme

T5 **Multimove:** The theme of multimove should enable the movement of multiple selected elements at once. It should be visualized that the elements get grouped together and moved as a virtual single element.

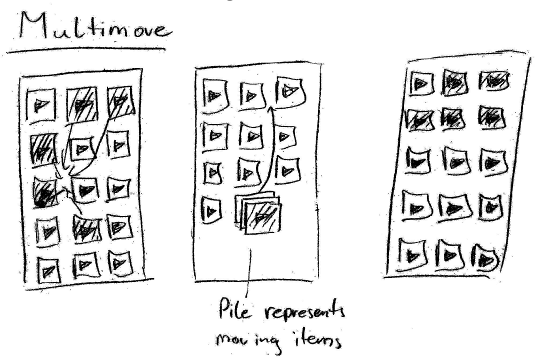


Figure 4.7: Sketch of the multimove theme

T6 **Corner Gestures:** The Corner Gestures theme enables a user to trigger functions by swiping to the corners of a user interface element. A swipe to one of the corners could trigger a state change, for example swiping to the top left corner starts the selection mode in a list, or trigger operations, for example swiping to the bottom right deletes the element the swipe has been performed on. The operations themselves are not predefined and can be defined based on the use case the Corner Gestures feature is used for. This theme supports detecting swipes to the corners as well as providing a default user interface to use for visualizing the actions performed at the corners.

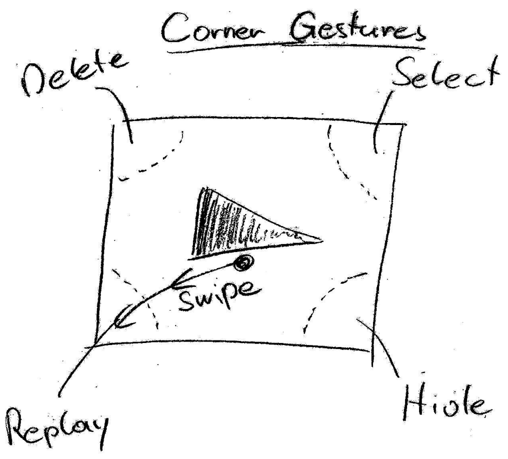


Figure 4.8: Sketch of the corner gestures theme

T7 **Swipeselect:** Swipeselect enables the user to select elements by swiping over them. This can be combined with the process of clicking single items to enable fast and precise selection of multiple elements. Not only should this theme support the developers in creating the swipe to select gesture but also a user interface element showing that an element counts as selected should be provided.

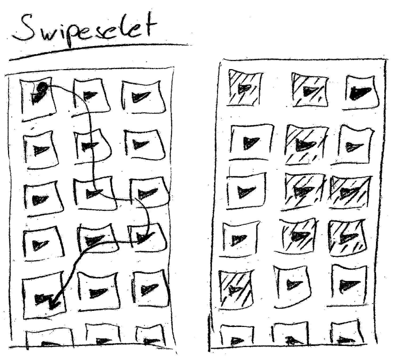


Figure 4.9: Sketch of the swipeselect theme

4.2 UserX Stories

The last phase of the thematic analysis is writing a report about the found themes. As the themes should be the basis of the implementation to be done within this thesis, the themes were transformed into the requirements for the implementation instead of writing a formal report. UserX Stories, as described in Section 3.4, were created for each theme identified within the thematic analysis. The format proposed by Choma et al. (2016) has been slightly tailored to fit the needs of the development done for this thesis. The format of the acceptance criteria has been loosened, as one of the goals of the thesis is to define abstract software components that can be reused in various different ways. This would not have been possible if such a concrete description of acceptance criteria was given in the format of a “Nielsen’s Heuristic”(Nielsen, 1995). Since acceptance criteria are helpful in testing if the wanted behaviour has been fulfilled, a simple enumeration of the wanted behaviour was written instead. In addition, the sketches from the themes were transformed into more detailed mockups to clarify how the user interactions and user interface elements should look after the implementation.

Each of the following UserX Stories represents one theme. The story of the UserX Story captures the user interaction and the acceptance criteria describe in short what should be possible when using the implemented user interface element. This built the basis for the implementation and answer both the research question on what is missing current software frameworks to better support the implementation of media-intensive user interfaces (RQ2) on Android and how the missing elements in current software frameworks can be defined (RQ3).

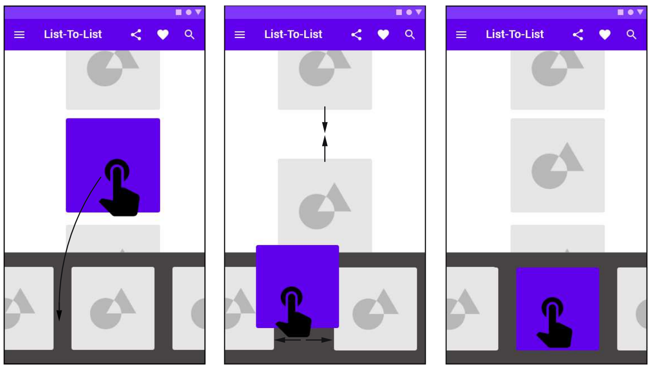
UXS1: List-To-List
As a person who edits videos, **I want to** move elements between lists, **for this the system allows me to** drag and drop elements between lists, **when** an element is dragged it is no longer part of the source list and on drop it gets part of the target list. **I evaluate that my goal was achieved when** the element is no longer part of the source list and the element is part of the target list.
Acceptance Criteria: • On element removal the source list animates the removal • On element insertion the target list animates the insertion • The element is not part of the source list after the movement is finished • The element is part of the target list after the movement is finished
Mockup:  Figure 4.10: UX flow of an element being removed from the list (left side), the animation happening (center) and added to the list (right side)

Table 4.2: UserX Story 1 - List-To-List

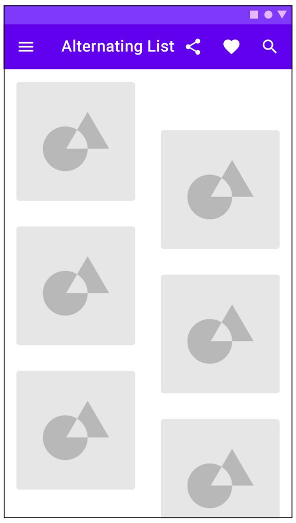
UXS2: Alternating List
As a person who edits videos, **I want to** see my elements in a list that alternates the offset per row, **for this the system** visualizes the elements alternating on the left or right side of the list and offsets them. **I evaluate that my goal was achieved when** I can see the elements in a list in an alternating manner.
Acceptance Criteria: • The elements get inserted with an offset per row if the list is vertical • The elements get inserted with an offset per column if the list is horizontal
Mockup:  Figure 4.11: Mockup of a vertical alternating list

Table 4.3: UserX Story 2 - Alternating List

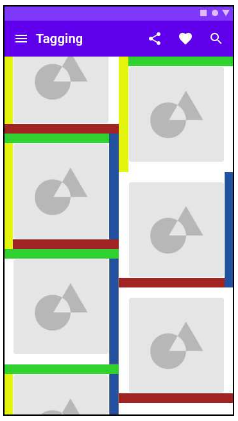
UXS3: Tagging
As a person who edits videos, **I want to** tag my multimedia files with custom values, **for this the system allows me to** select an element and assign a custom value to it, **when** the element is loaded at a later time or on a different device this custom value can still be read. **I evaluate that my goal was achieved when** I can read previously stored values for an element and see those tags for any video file.
Acceptance Criteria: • I can add tags to any .mp4 file • The tags can be visualized on my media files by showing a colored border on the element in the UI • Up to 4 tags get assigned custom colors to represent them in the UI
Mockup:  Figure 4.12: Mockup of a tagging User Interface. The sides of the lists elements get colored based on the assigned tags.

Table 4.4: UserX Story 3 - Tagging

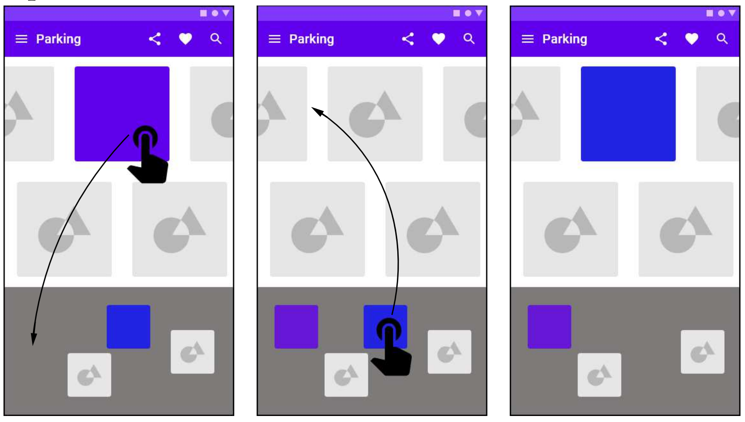
UXS4: Parking
As a person who edits videos, **I want to** temporarily park video files in a predefined area, **for this the system allows me to** drag and drop the video into a predefined are, **when** I drop the element it gets displayed similar to its original representation. **I evaluate that my goal was achieved when** I can drop any video file into an area.
Acceptance Criteria: • A video can be dropped inside the parking area where it stays at least until the app is closed • A video can be taken out of the parking area to be inserted somewhere else
Mockup:    Figure 4.13: Mockup of the parking area. The elements can be placed anywhere inside the gray area at the bottom and dragged out to a different location again.

Table 4.5: UserX Story 4 - Parking

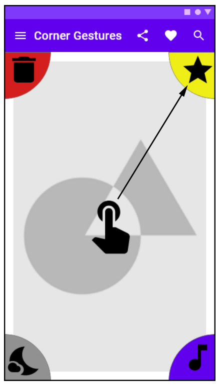
UXS5: Corner Gestures
As a person who edits videos, **I want to** trigger functionality with the currently shown video file, **for this the system allows to** me swipe to the corners of a shown video, **when** I reach one of the corners the predefined functionality for the specific corner is triggered. **I evaluate that my goal was achieved when** I am able to start functionality on each of the 4 corners of the element.
Acceptance Criteria: • If a swipe gesture is done towards one of the corners, a functionality that can be freely defined is triggered • For each corner can trigger different functionality • Icons can be used to represent the functionality triggered by a corner • The minimum distance to a corner to trigger the assigned functionality can be modified
Mockup:  Figure 4.14: Mockup of the Corner Gestures.

Table 4.6: UserX Story 5 - Corner Gestures

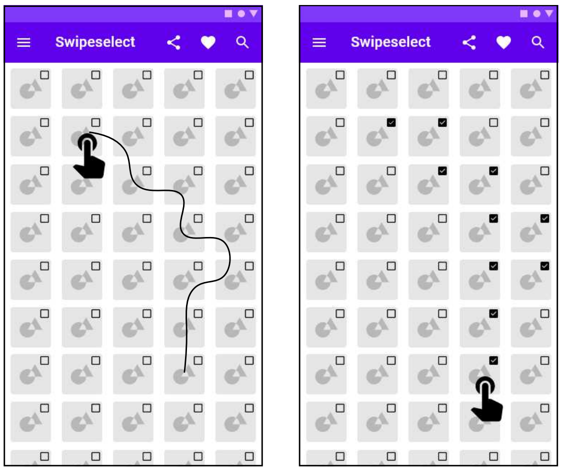
UXS6: Swipeselect
As a person who edits videos, **I want to** select multiple elements at once, **for this the system allows me to** swipe over the elements that I want to select, **when** an element is selected it is visualized on the element. **I evaluate that my goal was achieved when** I can select multiple elements within a list by swiping over them.
Acceptance Criteria: • Elements that were swiped over are set to be selected • A checkbox is set to be checked when an element is set to be selected
Mockup:  Figure 4.15: Mockup of the Swipeselect gesture. All elements swiped over (left image) are set as selected (right image).

Table 4.7: UserX Story 6 - Swipeselect

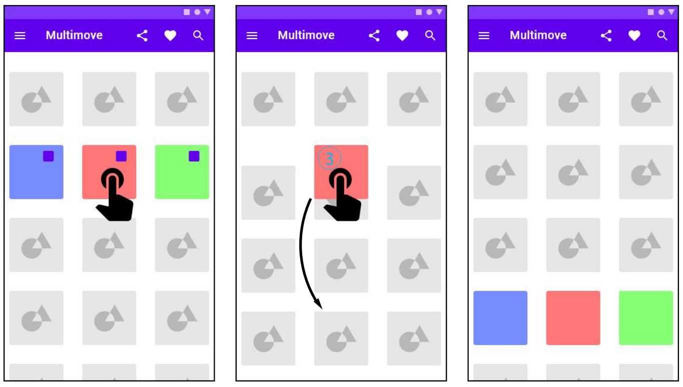
UXS7: Multimove
As a person who edits videos, **I want to** move multiple elements at once, **for this the system allows me to** drag multiple previously selected elements to a new location, **when** the move is happening it is shown how many elements are currently being moved. **I evaluate that my goal was achieved when** all elements that were supposed to be moved have been moved to the new location.
Acceptance Criteria: • If one one or more elements are selected and one is dragged all selected items move into a single item that moves along the finger during the drag and drop gesture • When the dragged element is dropped, the previously selected elements are inserted before the element they are dropped on • The format of this insertion depends on the UI element the drop has been performed on
Mockup:  Figure 4.16: Mockup of the multimove gesture

Table 4.8: UserX Story 7 - MultiMove

4.3 Implementation of the MicroDO library

After the definition of the UserX Stories, the development of the software library was started. The name chosen for the software library and their software components was "MicroDO". "Micro" stands for the size of the software components, which should be small and reusable in different contexts. "DO" stands for supporting the implementation of user interactions. The definition of a MicroDO software component for the development was as follows:

> A "MicroDO" is a software component that eases the implementation of user interactions on Android devices. With the combination of multiple "MicroDOs" complex user interactions and user interfaces can be represented.

In total 42 classes and interfaces were implemented for the MicroDO software library. These classes are part of eleven software components which represent seven MicroDOs. In addition to the MicroDO library itself, a demo application was implemented for showcasing the MicroDOs. This demo application was implemented in a separate software module to keep it completely separated from the actual MicroDO library source code. As such it is possible to create a build of the library without containing the source code of the demo module to keep the size of the final library small.

To avoid confusion within the descriptions of the MicroDO implementation, short low-level descriptions of commonly used classes within the Android framework are given below. For more detailed descriptions of the classes and paradigms defined within the Android development framework, please refer to the Android API reference[2].

- `Activity`: An `Activity` is the starting point for an Android application. An `Activity` gives the developer access to callbacks to react to lifecycle changes within the application. This also includes the setup and destruction of the user interface. In many cases, a single `Activity` object is created for each Window of a user interface, but also multiple activities can be used within multi-window applications or when floating windows are shown to the user.

- `View`: A `View` is the top-level class that represents a user interface element, similar to how `Object` is the top-level class for any Java class. The `View` class provides all basic functionality necessary to display user interface elements on the screen. All other user interface elements within the Android framework either extend the `View` class or a subtype of it.

- `RecyclerView`: The `RecyclerView` can display bigger datasets in a performant way. The `RecyclerView` uses an `Adapter` to store the data and detect changes within the dataset and a `LayoutManager` to handle how the data is presented

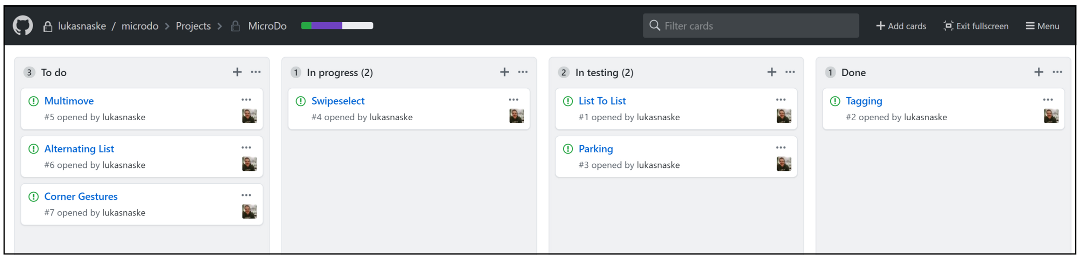
[2]`https://developer.android.com/reference` (Accessed: 03.06.2020)

and rendered. To handle big data sets efficiently, a `RecyclerView` re-uses `View` objects instead of creating new `View` objects repeatedly. The `Adapter` attached to the `RecyclerView` is responsible to handle the management of `View` object such that the `View` objects are re-used efficiently.

- Layouts: Layouts are root elements of user interfaces where `View` objects can be arranged in. Each layout is a `View` itself and as such the layouts can be stacked. The layouts are normally specified within XML files, but can also be modified from within the source code. To display the XML files, they are "inflated" within the `Activity` context to create the `View` objects. Android provides a selection of common basic layouts to arrange the user interface elements. Commonly used layouts are:
  - `ConstraintLayout`: The `ConstraintLayout` lets the developer assign constraints to the user interface elements based on other user interface elements. For example, it can be defined that a `View` should always be 10 pixels from the bottom of the screen, or that a `Button` should always be directly below another `View`.
  - `FrameLayout`: The `FrameLayout` is used as a wrapper around a single child `View` to use within another layout. It also supports adding multiple children that are overlapped.
  - `RelativeLayout`: The `RelativeLayout` allows the developer to display user interface elements relative to each other. This is quite similar to the `ConstraintLayout` but it uses slightly different paradigms to align the `View` objects.
- Listeners: Android supports by default a lot of listeners that help the developers to detect different kinds of user inputs. These listeners are interfaces. Classes that implement the interface can be attached to any kind of `View`. Commonly used listeners are:
  - `OnClickListener`: Detects a single short tap on the `View` the listener is attached to.
  - `OnLongClickListener`: Detects a single long tap on the `View` the listener is attached to.
  - `OnTouchListener`: Detects any kind of touch on the `View` the listener is attached to. The different kinds of events, like starting the touch, moving the finger and lifting it back up are sent to the listener.
  - `OnDragListener`: Detects that an element is currently dragged over or dropped on the `View` the listener is attached to. The events sent to the `OnDragListener` contain information about the dropped object as well as the exact location.

The implementation was done in Kotlin, using Android Studio[3] as integrated development environment (IDE) and Gradle[4] as build and dependency management tool. For source code management and version control, a Git repository on Github was used. Github was also used to handle ticket management. As mentioned in Section 3.3 Github also provides a solution for the Kanban board of the project which was also used. The implementation was done in increments. For this, each feature was first implemented rudimentarily in a first iteration and then refined in incremental steps. In addition, after finishing a MicroDO, already finished MicroDOs were revisited and new learnings were applied.

The Kanban board, which was used within the Github platform, can be seen in Figure 4.17. The work in progress limits set to each of the columns of the Kanban board was two, which means in both the "In Development" and "In Testing" columns only two tickets could be at the same time. The limit for the "In Development" column was chosen, to move already finished tickets back into the development column when new learnings needed to be applied, despite still working on a new ticket. This was essential to apply an iterative development approach and was necessary to make it possible to have interactions between the software components and keep all software at the same level of quality. The "In Testing" column also got the work in progress limit of two tickets, as this phase was mainly used to implement and polish the demo use cases to test the functionality. Having two MicroDOs in the columns at the same time, allowed to combine different MicroDOs into a single demo use case while sticking to the Kanban development process. By using this approach the limits of the columns on the Kanban board were never broken. When a column was full it enforced the focus on nearly finished tickets and as such sped up the development process overall.



3 To do	1 In progress (2)	2 In testing (2)	1 Done
Multimove #5 opened by lukasnaske	Swipeselect #4 opened by lukasnaske	List To List #1 opened by lukasnaske	Tagging #2 opened by lukasnaske
Alternating List #6 opened by lukasnaske		Parking #3 opened by lukasnaske	
Corner Gestures #7 opened by lukasnaske			

Figure 4.17: The Kanban board used on Github during development

Following is a list of the implemented MicroDO components and short descriptions of their usage within Android development. A set of common interfaces and classes was defined to ensure interoperability and are not specifically mentioned here. For more detailed information please refer to the documentation in the source code, which can be found on Github[5].

[3]`https://developer.android.com/studio` (Accessed: 31.07.2020)
[4]`https://gradle.org/` (Accessed: 06.06.2020)
[5]`https://github.com/lukasnaske/microdo` (Accessed: 17.08.2020)

- `AlternatingLayoutManager`: This layout manager can be used for a `RecyclerView` to get the staggered design from UserX Story 2 - the Alternating List. The `RecyclerView` concept is widely used and changing the way the elements of the list are displayed by simply attaching a different `LayoutManager` was favored over other list implementations. The `AlternatingLayoutManager` supports both vertical and horizontal item alignment and can be configured using the `AlternatingLayoutConfigLookup` class if wanted. The design is heavily based upon a blog post by Emre Babur on baddotech.badoo.com[6] but extended to support both vertical and horizontal alignment.

- `CornerGesturesOnTouchListener`: This `OnTouchListener` allows the functionality of the Corner Gestures MicroDO. It can be attached to any `View` and reports back to a class implementing the `CornerGesturesListener` interface which corner has been swiped to. It was also implemented to report if one of the corners is currently being hovered over, which means that a swipe is happening, but the touch did not yet end. To specify at which point each corner should be activated, a percentage can be given. The percentage is then used to calculate the activation distance by taking the percentage of the shorter side of the view the `CornerGesturesOnTouchListener` is attached to.

- `CornerGestureLayout`: This class is an extension of the `FrameLayout` from the Android framework. It can be used to show small corner buttons with icons in them around a view to give a visual representation of the corner gestures. To ease updating the visibility and the color of the corner buttons, the `CornerGestureHelper` was implemented. The `CornerGestureLayout` can be seen in Figure 4.18b.

- `ListToListOnDragListener`: To fulfill the List-To-List MicroDO, the process of enabling the Android drag and drop behaviour had to be made easier. For this the `ListToListOnDragListener` was implemented, which is a generic class that accepts a `MicroDoDragEvent` and adds it to the `View` the dragged element was dropped on. If the view the element is dropped on is a `RecyclerView`, the element dropped is added to the `Adapter` of the `RecyclerView`. Either the `View` itself or the `Adapter` of the `RecyclerView` has to implement the `ListToListDropable` interface, which defines methods for adding the element. To start the drag and drop behaviour for any user interface element, the `ListToListDragInitializer` can be used. In addition, as the List-To-List MicroDO is mainly focused on Lists, the abstract class `ListToListAdapter` was implemented as an extension of the `RecyclerView.Adapter` class. The `ListToListAdapter` class is abstract as it only handles the behaviour of adding or removing data for the drag and drop behaviour. A class extending the `ListToListAdapter` then does not have to handle the drag and

[6]`https://badootech.badoo.com/a-custom-layoutmanager-case-beeline-on-android-d8d31526596b` (Accessed: 01.03.2020)

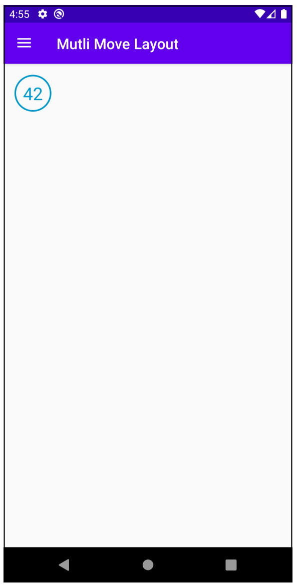
drop behaviour and only has to focus on the normal responsibilities of an `RecyclerView.Adatper`. The `ListToListAdapter` can also automatically attach the `ListToListOnLongClickListener` to all `View` objects created for the `RecyclerView`, which starts the drag automatically on a long click on one of the items within the `RecyclerView`.

- `MultiMoveListToListOnDragListener`: To support moving multiple elements for the Multimove MicroDO, an extension of the List-To-List MicroDO was implemented. The `MultiMoveListToListOnDragListener` extends the before mentioned `ListToListOnDragListener` and replicates the behaviour for all elements that were moved. Since the Multimove MicroDO relies on more than a single view being dragged, the `MutliMoveDragInitializer` was implemented, which takes all elements that should be moved and starts the drag and drop behaviour. A `View` object has to be given to the `MutliMoveDragInitializer` that is used as representation during the drag and drop behaviour.

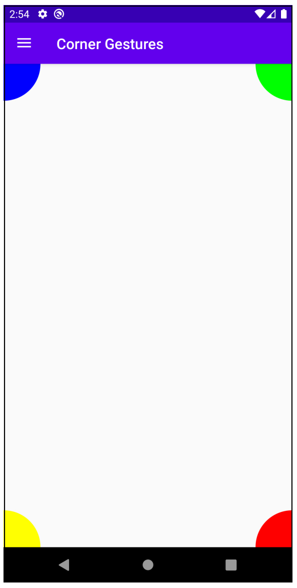
- `MultimoveLayout`: To visualize that more than a single element is dragged, a layout was designed that shows the number of currently dragged elements on any of the views being dragged as a number. Because of this the `MultimoveLayout` was created as an extension of the Android `FrameLayout`. The `MultimoveLayout` adds a `TextView` to the `View` it surrounds. This `TextView` can be used to show the number of elements that are currently being dragged. The `MultiMoveDragInitializer` checks if the `MultimoveLayout` has been applied to the `View` that should be used as representation during drag and drop and automatically shows the number of dragged elements in the `TextView` of the `MultimoveLayout`. In addition the `MultiMoveHelper` was implemented to ease updating the visibility of the `TextView` and the shown number. The `MultimoveLayout` can be seen in Figure 4.18a.

- `ParkingLayout`: The `ParkingLayout` is an extension of the `RelativeLayout`, which accepts any kind of `View` object dropped on it to be stored and later moved out of the `ParkingLayout` again. It attaches the `ParkingOnDragListener`, which accepts a `MicroDoDragEvent` to construct a copy of the dropped view and display it. It can also be defined in the layout XML if and how much the dropped view should be scaled to make it possible to reduce the size of the dropped views. By default the `ParkingOnLongClickListener` is attached to any `View` dropped in the `ParkingLayout`. The `ParkingOnLongClickListener` starts the drag and drop behaviour on long click on the dropped `View` and add the original event into the drop behaviour. If a different kind of interaction is wanted to start the drag and drop behaviour, the `ParkingLayout` can be extended and the `ParkingDragInitializer` can be used to start the drag and drop behaviour.

- `SwipeSelectListener`: To enable the Swipeselect MicroDO, it was necessary to detect all elements that were swiped over within a `View`. For this the `Swipe-`



(a) The `MultiMoveLayout` showing the number 42.



(b) The `CornerGestureLayout` with colors assigned to the corners.

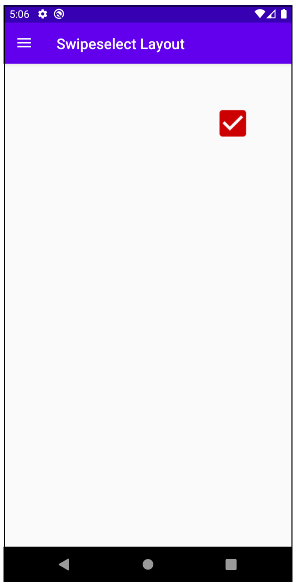
Figure 4.18: The (a) `MultiMoveLayout` and (b) `CornerGestureLayout` as seen in a fullscreen Android application.

`SelectListener` was implemented that is both an `OnTouchListener` and an `OnClickListener`. This way it can be used when swiping over elements as well as when the elements are only tapped. The `SwipeSelectListener` accepts an `SwipeSelectCallback`, which gets notified of the elements that are selected. The `SwipeSelectListener` also keeps track of elements that have already been swiped over to not report an element that has been swiped over twice. The `SwipeSelectListener` allows also for multiple touches to be seen as a single swipe, so that also when the user wants to select the items in multiple swipes, the listener still only reports each element swiped over once. This behaviour can be triggered by setting a parameter to either `true` or `false`, which then either automatically resets the cache of already swiped over elements or requires the developer to manually reset it.

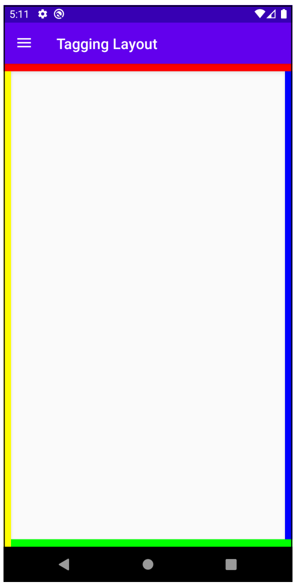
- `SwipeSelectLayout`: To have an indicator that a `View` was selected, an extension of the `FrameLayout` was implemented that puts a checkbox on the top right corner of the view. When using the `SwipeSelectHelper` this checkbox can be easily set to be checked or not. The `SwipeSelectLayout` can be seen in Figure 4.19a.
- `VideoTaggingService`: To store any kind of information with a video file, a service was implemented that can store any kind of key-value pair with the file. For

this the free to use software library "JCodec" [7] was used. This library provides the functionality to store the information within the file structure of an MP4 file. The implemented `VideoTaggingService` encapsulates and eases the functionality of the "JCodec" library and lets the developer use objects called `Tag` to store and read the values.

- `SideTagsLayout`: To provide an indicator that a view has been tagged with specific values, an extension of the `ConstraintLayout` has been implemented. The `SideTagsLayout` shows a border around the encapsulated view. Each of the four sides can be assigned with a different color to indicate that the encapsulated element has been tagged by a specific value, where each color represents a tag. By using the `TaggingHelper` the color for each of the border sides can be easily changed and the visibility updated. The `SideTagsLayout` can be seen in Figure 4.19b.



(a) The `SwipeSelectLayout` with the checkbox checked.



(b) The `SideTagsLayout` with all sides having different colors assigned.

Figure 4.19: The (a) `SwipeSelectLayout` and (b) `SideTagsLayout` as seen in a fullscreen Android application.

During the development of the MicroDO software library, there were a few challenges to tackle. The necessity to create a software library that can be reused in multiple contexts in a flexible and modular way required to focus on potential special cases. To create software components, which are reusable in different contexts, while providing useful

[7] `https://github.com/jcodec/jcodec` (Accessed: 08.05.2020)

functionality, the concept of generic classes was used a lot. Generic classes allow the programmer to use the same software component to handle a variety of different data types. While generic classes provide great benefits, there are also great limitations within the Java and Kotlin programming languages. When making no restrictions on which kind of data types can be used for with the generic class, only the most common methods of the base class `Object` can be used on objects of the generic type. This only results in issues when operations on the actual objects of the datatype were required. In cases where more operations were required, interfaces were implemented, that the users of the MicroDO software library have to implement in their own classes.

In addition there exist some limitations with regards to subtyping when using generic classes. Assume `Mp4Video` is a subtype of `Video`. Then `SwipeSelectListener<MP4Video>` is not a subtype of `SwipeSelectListener<Video>`. But, assuming `CustomSwipeSelectListener` is a subtype of `SwipeSelectListener`, then `CustomSwipeSelectListener<Video>` is a subtype of `SwipeSelectListener<Video>`. This behaviour is less intuitive than handling subtyping for non generic objects and has to be considered within implementations of the same generic class with different types and subtypes.

The subtyping of generic types had to be considered when implementing the `ListToListOnDragListener`. In cases the `View` object, which the listener is attached to, cannot accept the datatype that is dropped, the `ListToListOnDragListener` should reject the drop and the drag and drop should be reverted. For this reason, the `View` objects that attach a `ListToListOnDragListener` have to implement the generic `ListToListDroppable` interface. Based on the generic datatype chosen for the `ListToListDroppable`, the `ListToListOnDragListener` can decide if the `View` can handle the dropped element or not. If the developers using the `ListToListOnDragListener` do not focus on the issues with subtyping of generic classes this could lead to the drag and drop behaviour not working as intended. Also, there is no way to enforce the dependency of the `ListToListOnDragListener` to have the `View` object implement the `ListToListDroppable` interface. But the abstract class `ListToListAdapter`, which is a custom extension of the `RecyclerView.Adapter` implemented in the MicroDO library, attaches a `ListToListOnDragListener` of the correct type automatically to the `RecyclerView`. It is also mentioned in the documentation that this relation of the `ListToListOnDragListener` and `ListToListDroppable` is required when using it outside of the context of `RecyclerView` instances with the `ListToListAdapter`.

In Kotlin all classes are non-extendable by default, which means custom subtypes can normally not be created. To make it possible to extend the classes within the MicroDO library and to update as minimal code as possible to cover a custom use case, the classes and methods in the MicroDO library are marked as `open` to make them extendable. The usage of `private` methods and parameters was also avoided. While it is normally seen as bad practice to not encapsulate the code within the class, it was more valuable providing the possibility to override only parts of the implementation while reusing the main functionality without having to copy and paste the code from private methods.

Doing the implementation for the Android framework did not always result in the expected behaviour. Android supports attaching listeners to cover the most important interactions like clicking, long clicking and touch gestures. But it can easily happen that the listeners attached to a `View` do not receive events even though it is expected. As an example, when the child of a layout has an `OnClickListener` implemented, the parent `View` object of the layout does not get any touch events when the child with the attached listener is touched in any way, even though the touch gesture might be a swipe that the `OnClickListener` does not get notified of. To prevent this, the child `View` objects may only have an `OnTouchListener` attached, which can distinguish between a click event and a swipe and will ignore events that should not be handled. By ignoring the event correctly, the parent `OnTouchListener` gets correctly notified of the events. Also, it is not possible to attach multiple listeners of the same type without adding a custom implementation for this use case. This process of events not being propagated to other listeners despite not being handled by the current listener is also referred to as swallowing events. The listeners implemented in the MicroDO library all try to avoid swallowing events.

Another challenge that had to be tackled was the inability to create a simple copy of a `View` object. In most cases, a `View` is inflated from the layout definition in an XML file and then updated with the data to display. For the Parking MicroDO, it was necessary to accept and display `View` objects without knowing anything about the used XML file or the data that is displayed within. In a first attempt the `View` object that was dropped into the `ParkingLayout` was removed from its parent layout and simply added to the `ParkingLayout`. But this resulted in issues when dropping a `View` that was previously attached to a `RecyclerView`. As the `RecyclerView` reuses created `View` objects, this led to a different element appearing on the `ParkingLayout` than the `View` that was dropped. This happened because the `RecyclerView` already updated the data within the reused `View` object to show a different list element. To overcome this issue, the `ViewCopy` class was created. This class accepts the `View` and draws the user interface elements onto a `Canvas` object with the same size as the `View`. This drawing onto a `Canvas` object is also done when creating a representation of the `View` for the drag and drop behaviour within the Android framework. The `Canvas` that the `View` was drawn on, is then drawn onto the `ParkingLayout` resulting in a copy of the visualization of the original `View` that should be added to the `ParkingLayout`. The only limitation was that any behaviour or listeners of the `View` object get lost. When using the `ParkingLayout` in an implementation where it is known what kind of objects are dropped into the parking area, it might be beneficial to create an extension of the `ParkingLayout` that recreates the original `View` instead of creating the `ViewCopy` object. But this was not possible to do without limiting the usability of the `ParkingLayout` and making it less versatile.

When looking back at the themes that were discarded during phase four of the thematic analysis in Section 4.1, the theme of "Folders" can also be implemented using code provided by the List-To-List MicroDO. The interface `ListToListDropable` is used to check if the `ListToListOnDragListener` can interact with the `View` the `ListToListOnDragListener` is attached to. If the `View` implements the given interface, the `addItem()` method is called on the `ListToListDropable` object. As the `ListToListDropable` interface can be implemented by any extension of a `View` this allows for a folder style `View` to function easily with drag and drop. All that would have to be added is a user interface element representing the folder and the code to store the elements dropped into the folder and what to do with them. This is a good example of the reusability of the code within the MicroDO library.

CHAPTER 5

Evaluation

This chapter describes how the MicroDO software library was evaluated by conducting an API usability test. It is described how the API usability test was set up and how the tasks for the participants were designed. Then the conduction of the API usability test is explained in detail. This is followed by an analysis of the feedback gathered through the API usability test by conducting a thematic analysis.

5.1 Design of the MicroDO API usability test

To evaluate the usability of the MicroDO software library, an API usability test was conducted. The design of the API usability test was based on the API usability test as described by Grill et al. (2012). The authors also include a heuristic evaluation in their API usability test. However, this was the first iteration of the MicroDO software library and we were more interested in qualitative feedback from developers than in heuristic evaluation. Therefore we focused on the "Workshop with Developers"(Grill et al., 2012, p. 167) and the heuristic evaluation was omitted.

The API usability test was designed to have developers implement a small application using the MicroDO software library. Each application to be implemented was described in the form of a scenario, which consisted of small tasks which built upon each other. Each task was designed to be implemented using one or more MicroDOs from the MicroDO software library. At the beginning of an API usability test, demographic data from the participants was gathered using a questionnaire. Each participant took part in the API usability test one after the other to make it possible to use the "think aloud (TA) protocol" (Alhadreti and Mayhew, 2018, p. 1). The participants were encouraged to say anything that came to their mind during the implementation. The conversation and development process were recorded. Because of the possibility for the participants to state anything that came to their mind, it was not necessary for the participants to take any additional notes in written form as Grill et al. (2012) suggest. Right after the implementation section of the API usability test, a semi-structured interview was held with every participant of the API usability test. Through the semi-structured interview, further feedback of the participants was collected.

The scenarios were designed with four goals in mind.

1. All of the software components implemented in the MicroDO software library should be used by at least one of the participants. To achieve this goal, each scenario was designed to cover only a subset of the MicroDOs. Also, the order in which MicroDOs had to be used was swapped. This way the risk of scenarios not being finished and the MicroDOs not being used was mitigated.

2. The implementation done in the scenarios should cover realistic new use cases for the MicroDO software library. Because of this goal, the scenarios were discussed with the expert of the field who also took part in the discussion of the themes of the thematic analysis conducted in Section 4.1. Through the discussion relevant and new use cases could be found to be covered in the scenarios. Also, to simulate the context of a real software project, the tasks were designed to build upon each other and allowed for improving the implementation in iterations using the MicroDO software library.

3. The modularity and extendability of the MicroDO software library should be tested, as they were core goals of the implementation. To achieve testing of the modularity, the design of some scenarios was adapted to require the usage of multiple MicroDOs and standard Android components in combination. The extendability was tested by designing some scenarios in a way that the MicroDO software library did not support by default and required the participants to extend the existing code.

4. An experienced Android developer can pass the API usability test in no more than two hours. This was a goal, as the developers should stay motivated during the API usability test. Also, it was necessary to recruit multiple developers, which would have been significantly harder if the API usability test duration was not limited. To achieve the goal of limiting the time the participation in the API usability takes, multiple steps were taken. First, the scenarios were adapted to remove parts that did not require the usage of the MicroDO software library and were not essential for creating a realistic application. For example, it was decided not to require navigation between screens to achieve a single task. Secondly, the set up of the project was prepared beforehand and code that did not require the use of the MicroDO software library was pre-implemented. This included the basic user interface, the set up of the `Activity` and the `Fragment` components and the navigation between tasks. Also, the loading of the data to display from the device was implemented to avoid issues in performance. As a third step, each task should be possible to be implemented without finishing the previous task if the participants could not achieve the goal in a limited time frame. As mentioned before the tasks were designed to build upon each other. To make it possible to skip tasks or parts

of the tasks, the prerequisites for each task were implemented beforehand. These prerequisites also included the usage of the MicroDOs from the previous tasks, which made it possible to discuss different solutions for a task in case the developers took a different approach than the prepared implementation suggested. As each task was started in order, the developers did not see the pre-implemented sections until they finished or decided to skip the previous task.

To define the scenarios, the format of the UserX Stories as described in Section 3.4 was used again. The format of the UseX Stories was chosen as it allowed for a clear description of what was expected to be implemented while putting a focus on the required user interaction. The UserX Stories handed to the participants can be seen in the Attachments 2 to 6. Following is a description of the five scenarios that were designed:

Scenario 1: This scenario covered the MicroDOs Corner Gestures, Alternating List, Tagging and Multimove. The first subtask was designed to tag a single video that was displayed with values assigned to the corners using the corner gestures. In the second task, a list of the videos should be displayed using the `SideTagsLayout` showing the tags assigned to each video within the previous task. Using the corner gestures again, the videos should be filtered based on the assigned tags when swiped to a corner. Within the final subtask, it should then be possible to long click on any of the videos that were filtered before and drag all of the currently displayed videos into another list below using the Multimove MicroDO. The UserX Stories that were handed to the participant who had to implement scenario one can be found in Attachment 2.

Scenario 2: Within the second scenario the MicroDOs for Alternating List, Swipeselect, Corner Gestures, Tagging, Parking and Multimove were covered. The first subtask required the participant to display the videos in a list using the Alternating List layout and enabling the Swipeselect feature on the elements of the list. In the second subtask, it was then required to add Corner Gestures to the list. With the corner gestures, all previously selected videos should be tagged with a tag that was defined for the corner gesture. The third subtask should then enable the user to long click on any of the selected videos to drag and drop them into a Parking area below using the Multimove feature. The UserX Stories that were handed to the participant who had to implement Scenario 2 can be found in Attachment 3.

Scenario 3: In this scenario, the MicroDOs for Alternating List, List-To-List and Parking were covered. In the first subtask, two lists had to be displayed using the phone in the landscape orientation. The top list should be displayed using the Alternating List layout, while the bottom list should use a simple linear layout. Then the List-To-List feature should be enabled on both lists and videos should be possible to be drag and dropped from either list. To see if the MicroDOs are extendable, the default behaviour of long clicking the elements to start the drag and drop behaviour should be replaced by using a vertical swipe to start the drag and drop behaviour. In the second task, two parking areas should be added below the lists. In subtask three the swipe to drag behaviour implemented for the list elements should also be enabled on the elements dropped into the parking area. While this scenario covers fewer MicroDOs, the complexity was a lot higher as extensions of the classes had to be implemented. The UserX Stories that were handed to the participant who had to implement Scenario 3 can be found in Attachment 4.

Scenario 4: The fourth scenario covered the Alternating List, Tagging, List-To-List and Parking MicroDOs. The first subtask should display a list using the Alternating List layout. When clicking on any element of the list, a textbox should appear. Text written into this textbox should be stored within the video files using the Tagging MicroDO. In the second subtask, a parking area should be added on the left of the list to park any video. This also required the usage of the List-To-List MicroDO to start the drag and drop behaviour from the list. The third subtask focuses on replacing the parking area with a secondary list where the elements should be possible to be dragged and dropped to as well. The UserX Stories that were handed to the participant who had to implement Scenario 4 can be found in Attachment 5.

Scenario 5: This scenario required the usage of the MicroDOs for Swipeselect, Multimove and List-To-List. In the first subtask, a list should be displayed that scrolls horizontally. Using Swipeselect it should be possible to select the elements within the list. In the secondary subtask, it should be made possible to drag and drop the selected videos using the Multimove MicroDO. This should make it possible to drop the selected videos into so-called "Buckets". These "Buckets" function similar to folders on desktop computers and collect all videos dropped into the "Buckets". Each of these Buckets is represented by an icon and should display the number of videos that were dropped into it. In the third subtask, the videos, that were dropped into the buckets, should be displayed in the list below when a bucket is clicked. The select and drag and drop functionality added to the list in the previous tasks should still be usable when displaying the videos from a bucket. Through this, it should be possible to move videos in between buckets. While this scenario only covers three MicroDOs, it used the MicroDOs in a more complex manner that required reusing code from the List-To-List MicroDO to enable the dropping of elements into a user interface element that is not a list. The UserX Stories that were handed to the participant who had to implement Scenario 5 can be found in Attachement 6.

The scenarios for the API usability test cover all of the MicroDO software components. Some of the scenarios require the usage of more MicroDOs by using only the code provided within the MicroDO software library. Other scenarios encourage the extension of the functionality or the usage of the MicroDO components for use cases that were not initially expected. This should ensure to find usability issues in different areas. Also, it supported verifying that research questions RQ 3 on how to design the software framework in a way to cover a variety of use cases, was answered by making the design decisions for the MicroDO software library.

To conduct the semi-structured interview after the development section of the API usability test was done, a set of questions was prepared to guide the conversation. The questions were adapted based on the conversation flow. In addition to questions about the usage of the MicroDO library, the participants were also asked for feedback about the API usability test itself. During the interviews, the audio and screen were still recorded, as this allowed the participant to show directly if something would have been expected to work differently. The questions prepared in advance were the following:

- Is there any important feature missing from the library that you would have liked to see?
- Is there any feature that you did not need and would have rather implemented from scratch that was implemented within the MicroDO framework?
- Do you have experience with similar libraries like the MicroDO library? If yes, how does the MicroDO library compare to those?
- How is the MicroDO library different to other libraries in general that you have used before?
- How could you imagine using the MicroDO library for other tasks?
- How did you feel during this API usability test? (stressed, observed, controlled, stupid, annoyed, comfortable)
- Is there anything that you would have expected to be different when agreeing to take part in this API usability test?
- Do you have any other feedback that you would like to share regarding the MicroDO library or the usability test?

5.2 Conducting the API usability test

For the MicroDO API usability test, five experienced Android developers were asked to take part, so that each of the designed scenarios was implemented by one of the participants. The participants were recruited by asking already known software developers if they had acquaintances who fit the participant subscription of being experienced Android developers. The developers found through this referral were asked to also refer more developers until the required number of participants for the API usability test was recruited. This process is also known as snowball sampling. (Naderifar et al., 2017)

At the start of the usability test, a document was sent to the participants which consisted of an introduction and the scenario to be implemented by the participant. The introduction included a summary of the context of the MicroDO API usability test, a questionnaire to collect the demographic data and a short background story which should set the context of implementing a real software project. The summary text was not necessary to be read by the participants but was covered while introducing the participants to the MicroDO software library and the purpose of the API usability test. The questionnaire to collect the demographic data of the participants can be seen in Table 5.1. After explaining the context of the MicrDO software library and filling out the questionnaire, the background story was read to the participants that should describe a context in which the library should be used. The introductory part of the API usability test description including the summary, the questionnaire and the background story can be seen in Attachment 1. The introductory part was added to each of the scenario specifications, which consisted of the task descriptions in the form of UserX Stories and can be seen in Attachments 2 to 6.

Age:			
Years of experience with Android:			
<1	1 - 3	3 - 6	>6
Years of experience with Kotlin:			
<1	1 - 3	3 - 6	>6
Years of experience with Programming:			
<1	1 - 5	5 - 10	>10
Experience with Video Editing:			
Yes		No	

Table 5.1: Questionnaire handed to participants

The participants had to be provided with the MicroDO software library before the first release of the MicroDO software library to the public. For this, the participants got access to the original Git source code repository. A separate software module for the API usability test was added to the source code and the MicroDO library module was linked to the API usability test module. Within the API usability test module, the setup of the application was done and the subtasks were prepared for each of the scenarios. The participants were asked to only implement code in the API usability test module to keep the code separated from the MicroDO software library itself. This should ensure the usage of the MicroDO library was as close to a real software project without having the MicroDO software library released. Having the module within the official repository made it possible for the participants to check both the library source code as well as the source code for the demo application within the Android Studio IDE. It allowed the participants to check the comments within the source code without leaving the Android

Studio IDE or having to decompile the source code.

For the analysis after completion of the API usability tests, the conversation with the participants as well as their screen, where the implementation was done, got recorded. The recording was vital despite the MicroDO API usability test being held one after the other, as this removed the necessity to write down all remarks by the participants during the test and staying responsive to questions at any time.

The API usability test was conducted during the COVID-19 pandemic[1]. Because of this, it was not allowed by the government of Austria to meet with the participants who agreed to take part in the API usability test in person. This lead to the API usability test being conducted remotely using online video chat tools that support screen sharing. The free web application “Let’s meet”[2] by Uninett[3] was used for four of the API usability tests. The benefits of using “Let’s meet” over other applications were that it did not need any local installation, it was free to use and had no limit on meeting duration. In general, it turned out that "Let’s meet" was a viable tool with minor issues which did not hamper the over-all process. However, one participant asked to use a different web-based video chat tool (“Google Meet”[4]), as he did not want to use a tool he was not familiar with. There was no impact on the API usability test by using a different tool, as “Google Meet” provided a similar feature set. The only feature both of the video chat platforms were missing was some kind of pointing device on the shared screen. The possibility to show the participants during the introduction where some of the described code can be found or where to click during the setup process would have been very helpful as it was often hard to describe exactly where the necessary button or feature of Android Studio was located.

It was not possible to provide the participants with a notebook and smartphone already setup to start the implementation right away. Because of this, the personal notebooks of the participants had to be set up before the beginning of the API usability test. The Android Studio project had to be shared with the participants, the correct Android Studio version and Android Software Development Kit (SDK) needed to be installed and the necessary dependencies to other software libraries had to be downloaded. In addition, an emulator or physical Android device had to be set up with the videos to display in the API usability test in the correct location. The downloads and installations were done by the participants beforehand. Parts of the setup, e.g. copying the video files to use during the API usability test to the correct location on the Android device or emulator, were more convenient to be done together at the beginning of the API usability test. Overall this increased the time necessary before starting the implementation section of the API usability test.

[1]`https://www.who.int/emergencies/diseases/novel-coronavirus-2019` (Accessed: 08.06.2020)
[2]`https://letsmeet.no` (Accessed: 14.05.2020)
[3]`https://www.uninett.no/en/about-uninett` (Accessed: 14.05.2020)
[4]`https://meet.google.com/` (Accessed: 23.05.2020)

Three of the participants asked to do the implementation in Java instead of Kotlin as they were more used to write applications using the Java programming language. As Kotlin provides full interoperability with Java, this was possible. In addition, by testing the usability with both programming languages, it showed if there are usability issues that only arise when using the MicroDO software library within one of the programming languages. The set up of the API usability tests was adapted to be written in Java in these cases to make all the code easily understandable for the participants.

The tutorial phase of the API usability test was started after the project was fully set up and running. The demographic data was collected as a first step and can be seen in Table 5.2. The names of the participants were anonymised, the here mentioned names were chosen in alphabetical order for citation purposes. All of the five participants were male and were actively working as software developers. The participants used their computers due to the necessity of the API usability tests being held remotely. This led to two participants using Windows, two participants using MacOS and one of the participants using Ubuntu. Two of the participants decided to run the API usability test application on their own physical Android devices and showed the results into the camera after each task, the other three participants used an emulator to run the Android application in.

Name	Age	Experience in Android in Years	Experience in Kotlin in Years	Experience in Programming in Years	Experience in Video Editing
Adam	26	1 -3	1 - 3	5 - 10	No
Bart	28	1 - 3	< 1	> 10	Yes
Cesar	31	3 - 6	1 - 3	> 10	No
Dennis	24	1 - 3	< 1	5 - 10	Yes
Erik	28	1 - 3	< 1	> 10	Yes

Table 5.2: Demographic data of the participants collected during the API usability test

After collecting the demographic data of the participants, a short introduction to the MicroDOs was given to the participants. The Readme files were also shown to the participants. Each Readme file contained a description of the MicroDO it was created for in a less technical way. The descriptions also included screenshots of the available user interface elements as well as code examples on how to use the software components. A separate Readme file existed for every MicroDO and the Readme file in the root folder of the source code repository had the other Readme files linked. These links could be used like hyperlinks on the Github website to navigate to the correct Readme file without navigating the file structure.

The set up was done in 10 minutes in general. Only for one API usability test, the files of Android Studio or the Android SDK were corrupted and the source code could not be compiled successfully. As the participant had a tight schedule, it was decided to reinstall Android Studio and the necessary Android SDK again, while the API usability test already started. Because of this, the first task was done in a way that the participant did not write the code himself, but dictated what code should be written while the screen was shared to him. He still had full access to the documentation on his own computer. After finishing the first task the reinstallment of Android Studio was done and the build was successful. At this point, the participant could take over on his own PC with task two, as the parts from task one, that were needed for the second task, were pre-implemented in the setup. In all other cases, the execution of the API usability tests did not run into any significant issues.

The duration of the API usability tests ranged from 1.5 hours to 2.5 hours. The implementation of Scenario 1 lasted the longest with 2.5 hours. This was probably due to the fact, that the user interface for the second subtask was not based on the user interface of the primary task. In addition, the participant was quite nervous about having someone watching him implement the tasks. Also, the Corner Gestures MicroDO would have had to be used twice, which is a MicroDO that takes a considerable amount of time and implementation effort to assign functionality to each corner. As the API usability test ran already for over one hour, Scenario 1 was slightly altered during the API usability test to reduce the implementation time. As such the filtering process via the Corner Gesture MicroDO and the Multimove functionality were not implemented. Instead, the participant used the List-To-List MicroDO to move single elements from the list at the top to the list at the bottom of the screen.

The participants were able to fulfill the tasks with only little help. The most significant issue where support was needed was when it came to identifying the functionality of the Alternating List by reading its name. None of the participants was able to identify the feature of displaying a list in a staggered manner by the name of the Alternating List MicroDO. All of the participants first tried to find the documentation to create the staggered layout by checking the documentation of the List-To-List MicroDO. After the participants were pointed at the documentation of the Alternating List MicroDO, they were capable to use the software component on their own without any issues. The participants mentioned that the name of the Alternating List MicroDO was not intuitive and hard to derive meaning from the name itself. There were other occasions where the participants that took part in the API usability test first checked the documentation for a different MicroDO, but they were quickly able to identify the correct MicroDO on their own in those cases.

Help was also provided to the participants when either the compilation of the API usability test code failed or the application crashed on startup. The reasons for those failures were mostly related to typing errors, where syntactic symbols were missing, or due to the fact that methods were implemented but not used. All of those errors were only pointed out if the participants seemed to have problems identifying the issue quickly due to being nervous or not seeing the notifications from the Android Studio IDE.

5.3 Feedback Analysis

To analyse the API usability test a thematic analysis was conducted on the collected material from the API usability test. For the first phase of the thematic analysis, which is the introduction to the dataset to be analysed, it is suggested to transcribe any collected audio material. But, an essential part of the MicroDO library API usability test was not only what the participants of the API usability test said, but also what they programmed and how they interacted with the provided documentation and source code. A full transcription of the interactions and the code written would have proposed significant effort, while not being able to retain the same level of detail as the video and audio recordings did. Instead of doing a full transcription, the recordings of the API usability test were watched and notes were taken with timestamps when anything was said about the library or the participants encountered new parts of the library. In a second step, the videos were watched again and the previously taken notes were iterated over. Also, the timestamps for the questions of the semi-structured interviews were written down and the answers of the participants during the semi-structured interviews were completely transcribed. (Denham and Onwuegbuzie, 2013)

The notes with the timestamps were then revisited in phase two of the thematic analysis to apply the codes. Each of the noted timestamps was watched again and the sections were coded. By not being limited to a written transcription but having the video recording of the screen of the participant, the handling with documentation and the usage of the library within Android Studio was also possible to be coded. The codes were assigned to sections of the recordings and were documented with essential statements by the participants and descriptions of how the participants interacted with the software library. In some cases also screenshots or sections of the produced code were copied to document the assigned code. The answers to the semi-structured interviews, which were transcribed, were also coded within this process. The result of coding the API usability test were 55 codes which can be seen in Table 5.3. The codes have been sorted into different subcategories for better readability.

With the codes extracted from the API usability test, the first iteration of the themes was created in phase three of the thematic analysis. The mindmap created for the themes can be seen in Figure 5.1. The themes for the analysis of the API usability tests were grouped under parent themes to create an overview of the origin of the theme.

The themes of the API usability test were revisited in phase four of the thematic analysis to verify the found themes. This led to the themes of “Copy & Paste from Readme”, “Defaults of Kotlin not working Java” and “Java code missing in Readme” being removed, as they can be represented by the single theme of “External documentation extendable”. The themes “Swipe To Unselect” and “Searchable Tags” were combined into a single theme called “Cover more features”. None of the feature requests had enough data supporting a single feature as a standalone theme. As such it is more suitable to cover the request for more features under a general theme called “Cover more features”. The themes of “Methods and variables named understandably “ and “Most classes and features named

Features recognised	**Features misunderstood**
Corner Gestures recognised	Differences of Corner Gestures and Tagging
Swipeselect recognised	List-To-List and Alternating List confused
Multimove recognised	Mutimove expected by List-To-List
Parking recognised	Parking confused with Tagging
List-To-List recognised	Swipeselect for touch to move
Tagging recognised	AlternatingLayout misunderstood
Multimove layout recognised	Not sure if applies to Android Guidelines
Parkinglayout recognised	Tagging layout not expected
Drag initializer recognised	Parameters of helpers misunderstood
Documentation	**Positive remarks**
Copy & Past from Readme	Works on first try
Different tag objects for each tag	Easy to use
Helper classes not expected	Small amount of code
Methods recognised by name	Usable in future projects
Readme taken for information	Similarity to other libraries
Check source code	Little “black magic”
Generics not expected	Few imports
Helper initialisation not understood	Documentation good for new users
How does this work internally?	Everything is extendable
Java code example missing	
Source code well commented	**Feature Requests**
Readme ok but extendable	Preview of videos from API usability tests as MicroDO
	Searchable tags
API usability test	Features unreleated to arrangment and sorting
Uncomfortable being watched	Better Swipeselect integration with lists
Comfortable	Swipe to unselect
UI test expected	TaggingLayout naming not fitting
Very tailored use cases	Helpers also handle features
As expected	Configurable Tagging Layout
Worked well in emulator	Always show UI element in editor
	List-To-List unnecessary

Table 5.3: Codes extracted from the API usability test during phase two of the thematic analysis.

understandably“ were combined into a single theme called “Good naming strategies” as it was not necessary to name each well-named part of the code by itself. The one feature being specifically called out for not being named in a way to understand the feature was the Alternating List. Because of this, a theme is dedicated to renaming the Alternating List feature was kept with “Alternating List not named understandably”. The themes for issues with inconsistencies within the API were slightly renamed. The

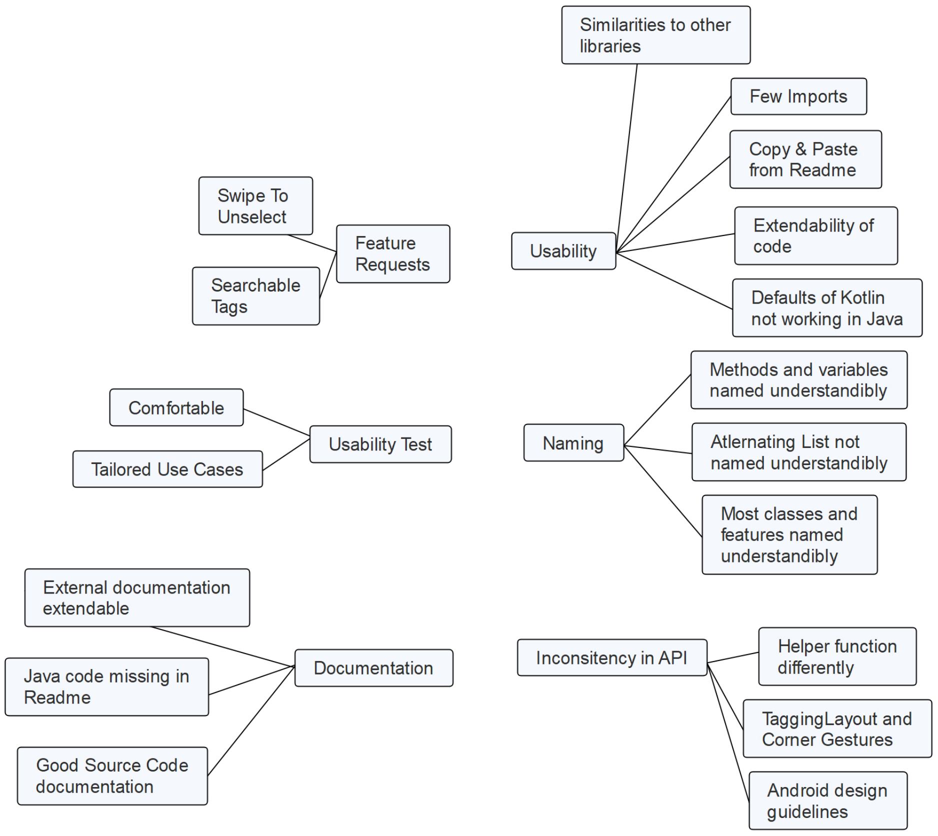

Similarities to other libraries
Few Imports
Copy & Paste from Readme
Swipe To Unselect
Feature Requests
Searchable Tags
Usability
Extendability of code
Defaults of Kotlin not working in Java
Methods and variables named understandibly
Comfortable
Usability Test
Tailored Use Cases
Naming
Atlernating List not named understandibly
Most classes and features named understandibly
External documentation extendable
Java code missing in Readme
Documentation
Good Source Code documentation
Inconsistency in API
Helper function differently
TaggingLayout and Corner Gestures
Android design guidelines

Figure 5.1: Mindmap of themes and subthemes for the API usability test analysis after phase 3 of the thematic analysis.

theme for the inconsistency of the `SideTagsLayout` and the Corner Gestures was removed as the issue was mentioned by only one of the participants in the API usability test. This participant had to specifically use the features Corner Gestures and Tagging in combination. He mentioned, that it was confusing, that the corners for the Corner Gesture feature were identified by "top left", "top right", "bottom left" and "bottom right", but the sides of the `SideTagsLayout` were identified by "top", "left", "right", "bottom". The ideas behind the "Corner Gestures" and the "SideTagsLayout" are in fact quite different and were not intended to be consistent with each other. As the issue was only mentioned by one participant, the theme was discarded. The mindmap of the final themes for the API usability test analysis can be seen in Figure 5.2.

With phase five of the thematic analysis, the found themes are defined and described. Following are descriptions of the found themes, which relate to the usability of the MicroDO software library and were grouped under the theme "Usability".

- **Extendability of code:** This theme relates to the fact, that the code of the MicroDO library is fully extendable. The interfaces can be implemented by custom

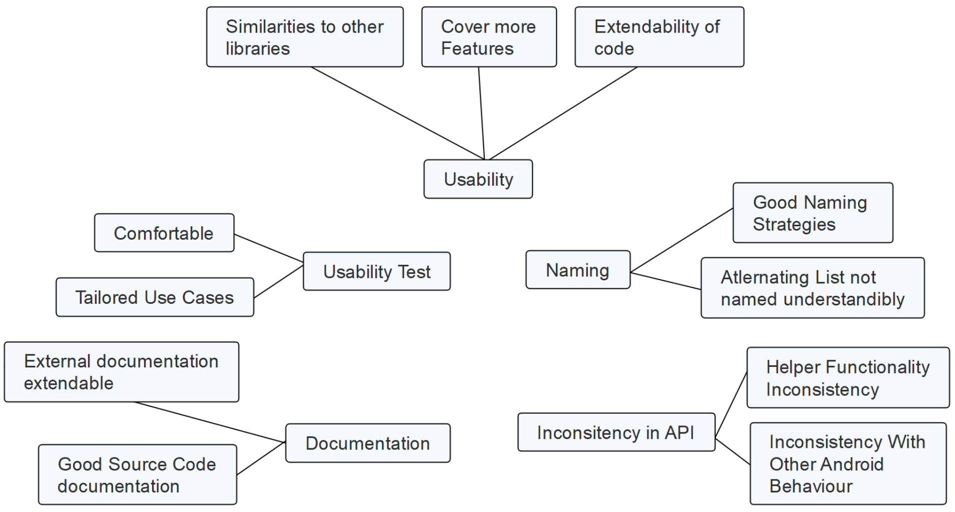

Similarities to other libraries
Cover more Features
Extendability of code
Usability
Comfortable
Usability Test
Tailored Use Cases
Naming
Good Naming Strategies
Atlernating List not named understandibly
External documentation extendable
Documentation
Good Source Code documentation
Inconsitency in API
Helper Functionality Inconsistency
Inconsistency With Other Android Behaviour

Figure 5.2: Mindmap of themes and subthemes for the API usability test analysis after phase 4 of the thematic analysis.

classes and used with the MicroDO implementations. But also all classes of the MicroDO software library can be extended. Also, all methods are not defined as private methods and can be overwritten by an extension of the class. This allows participants that use the MicroDO library to reuse the existing code and only add new behaviour or replace unwanted behaviour. It has been mentioned in the API usability tests by participant Dennis that this possibility is not given by many other libraries, which normally leads to the necessity of finding workarounds. In addition, Dennis also mentioned that he liked the whole library's "modular" design. As such the goal of creating a modular, flexible and reusable software framework has been fulfilled.

- **Cover more features:** The feature set of the MicroDO library is currently limited to user interactions and user interface components with regards to sorting media-items. During the API usability test, it was suggested to provide further functionality within the already covered use cases. For example, participant Eric mentioned that it would be a valuable feature to be able to search for tags using the Tagging MicroDO. In addition, there were features suggested outside of the context of sorting multimedia items. As an example, participant Cesar mentioned that he would like a list component, that automatically changes the width of the elements that get displayed based on the length of the video. Through this, a representation of the timeline metaphor should be easier to implement. All of the participants mentioned that the MicroDO software library could be used in the future for software projects with suitable requirements. Because of this, covering a bigger variety of use cases for video editing applications can benefit the usability and relevance of the MicroDO library in the future.

- **Similarities to other libraries:** All of the participants of the API usability test mentioned that the MicroDO library has similarities in usage to other software libraries with regards to quality and usability of the source code and documentation. The similarities were mentioned as being rather positive, but also feedback on how to improve the MicroDO library was given. As mentioned before, the possibility to extend every class of the library fully was mentioned as something the MicroDO library does better than many other libraries. But, the documentation outside of the source code of the MicroDO library could be better in comparison to other software libraries, which is discussed in the "External documentation extendable" theme.

Next to themes relating to the usability of the MicroDO software, there were also themes found which specifically relate to the of the MicroDO software library. These themes were grouped under a common theme "Documentation" and are described below.

- **External documentation extendable:** The documentation outside of the source code for the MicroDO library is entirely done within Readme files. The Readme files that were created for the MicroDO library describe the software components, how to use them and provide code examples in Kotlin that can be used by copying the code and pasting the code. The only changes that are necessary outside of the copy & paste action are adding the correct `import` statements and potentially changing variable names to fit the code it is copied into. But, it was mentioned by 4 of the 5 participants that the external documentation could be extended. For example, the code examples could be given in both Kotlin and Java to remove the necessity for Java participants to change the code to be in Java. Also, some features of Kotlin, like default values for parameters within methods or constructors, are not available in Java. In addition, while the source code documentation covered all methods and parameters, the participants of the API usability test asked for the source code documentation to be also available in the Readme files. They mentioned that the Readme files are the primary source for information for them. The source code documentation is only checked when looking for implementation-specific information.
- **Good source code documentation:** Of the five participants, that took part in the API usability test, three took a look at the source code documentation and comments within the classes. All three of them mentioned that the source code of the MicroDO library was well documented and as mentioned in the "External documentation extendable" theme, they would only ask for parts of the source code documentation to be added to the external documentation.

Also, themes were found that describe inconsistencies within the MicroDO software library. Following are description of these themes, which were grouped under the theme "Inconsistency in API".

- **Helper functionality inconsistency:** Of the eleven software components that were implemented, four contain so-called helper class that supports the participant in updating user interface components from the MicroDO software library. The participants of the API usability test mentioned, that after using multiple of these helper classes, there exist inconsistencies between the different helpers. This means that the way the helpers are initialized and used is not in line with the same paradigms for all helpers. It was suggested to choose one design for all of the helpers.

- **Inconsistencies with other Android behaviour:** Android has quite detailed design guidelines[5] that define how the user interface and user interactions should be designed. By applying those design guidelines it should be easier for users to use different Android applications, as all Android applications should use similar user interface and user interaction design approaches. Participant Bart mentioned during the API usability test, that he was not sure if the user interactions supported by using the MicroDO library would comply with the Android design guidelines. Participant Anton brought to attention, that the way the layout components function within the MicroDO library is different from how the layout components function within the Android framework. When using the graphical layout editor, it is normally clear to the participant that the element added last is on top of other components added earlier. Also, it is normally the case, that when using a layout to wrap a component, the child components are on top of the parent component. But in the case of the layouts implemented within the MicroDO library, the parent components add more user interface components that are added over the child components. The participant of the API usability test mentioned that this could lead to confusion.

Some of the found themes relate to the naming used for methods and classes in the MicroDO software library. These themes were grouped under the theme “Naming” and following are descriptions of them.

- **Alternating List not named understandably:** All four of the participants who had to use the Alternating List MicroDO within the API usability test did not recognise the Alternating List MicroDO to provide the wanted functionality. They all were confused about what was meant by the name of the Alternating List MicroDO. All of the participants rather checked the documentation of the List-To-List MicroDO to find information on how to create the staggered layout provided by the Alternating List. It does not appear to be an issue with the naming of the List-To-List MicroDO, as all of the participants recognised the List-To-List MicroDO to provide the functionality to move elements from one list to another. The participants mentioned that a different name would be better for the Alternating List MicroDO, but they could not think of any examples for a

[5] `https://developer.android.com/design` (Accessed: 20.08.2020)

better name. The only reason the participants gave as to why they first checked the List-To-List MicroDO, was that the List-To-List MicroDO had “list” in its name. The naming for the Alternating List MicroDO should be revisited before releasing the MicroDO software library to avoid the feature being overlooked. It could also help to provide descriptions and screenshots of the MicroDOs on the parent Readme file to give the users of the MicroDO library an idea of what the MicroDOs feature sets are.

- **Good naming strategies:** Overall the participants had little to no issues finding the correct classes and MicroDOs to use. The names for most of the MicroDOs were chosen well, except for the Alternating List MicroDO. The participants were also easily able to find the classes and methods to use by using the automatic suggestion feature built within the Android Studio IDE. The participants were able to understand the functionalities of almost all classes, methods and parameters by reading the name without requiring further investigation by reading the documentation.

In addition to the themes that relate to the source code and quality of the MicroDO software library, there also exist themes that relate to the design of the API usability test. These themes are described below.

- **Tailored Use Cases:** All of the 5 participants who took part in the API usability test mentioned in the end that they could imagine using the MicroDO library in a future project. But they also mentioned that the use cases provided during the API usability tests were tailored to make sure the MicroDO library fulfills the needs to implement the tasks. As such, they said that it would have to be very clearly checked whether the requirements of future projects could be fully covered by the MicroDO library.
- **Comfortable:** 4 of the 5 participants of the API usability test mentioned that they felt quite comfortable with the set up of the API usability test. Only one participant would have preferred to implement the scenario without having the screen recorded and somebody watching him implementing the scenario. He felt it was more similar to an exam situation when somebody is watching him implement a feature. On the other hand, the participant that would have preferred to implement the scenario on his own was glad that the time to implement the single steps was restricted, as he would have otherwise spend a lot more time on layout changes and would not have focused only on the MicroDO parts of the API usability test. This theme shows that the setup and design of the API usability test were successful in creating a positive environment for the participants. Through this it allowed the participants to focus on discussing issues within the implementation of the MicroDO software library instead of criticizing the API usability test itself.

CHAPTER 6

Results

This chapter describes the results of the conducted API usability test. First, the themes from the thematic analysis are revisited and discussed. This is followed by a categorisation of the found usability issues.

6.1 Results of the API usability test

The themes from the thematic analysis were rated as *positive*, *rather positive*, *rather negative* and '*negative* with regards to the usability and quality of a software development project. The four themes "Good Naming Strategies", "Extendability of code", "Similarities to other libraries" and "Good source code documentation" were rated *positiv*. The theme "Cover more Features" was rated as *rather positive* and the theme "External documentation extendable" as *rather negative*. The themes "Helper Functionality Inconsistency", "Inconsistencies with other Android behaviour", "Alternating List not named understandably" were rated as *negative*. In addition, the two themes that describe the API usability test process were rated as well. The "Comfortable" theme was rather as *positive* and the "Tailored Use Cases" was rated as *rather negative*. In the following paragraphs, first the *positive* and *rather positive* rated themes are discussed and the positive and negative learnings for the themes are pointed out. This is followed by a discussion of the *negative* and *rather negative* rated themes and potential solutions are considered.

6.1.1 Positive rated themes

The naming chosen throughout the implementation of the MicroDO library was fitting, which was summarized in the theme of "Good Naming Strategies" and was rated as *positive*. The participants of the API usability test had little issues when looking for the functionality required to fulfill the tasks of the API usability test. The parameters of the methods were also understandable. Only when using the MicroDO library with the Java programming language, some confusion occurred. When using Kotlin, default values can be provided for parameters of constructors or methods, making it optional to change the value assigned to a parameter. The default values for parameters were used for constructor and method parameters when a parameter was optional. But, when using constructors and methods which provide default values for parameters with Java, the values have to be assigned by the participant. To find which value is assigned by default to a parameter the participants would have to check the documentation or the source code. A solution for the issue with default parameters would be to overload the methods, which means to provide the same method with different parameters and set default values for the parameters this way.

The theme of "Extendability of code" encapsulates that the MicroDO implementation does not restrict participants to create extensions of the created classes and methods. The participants liked the fact, that the library allows for custom classes being implemented by extending the existing implementations. Because of this, the theme was rated as *positive*. There exist no restrictions on access to methods and variables in the code, which allowed the participants to only overwrite the behaviour they wanted to be changed. The only issue the extendability of the MicroDO software library provides is that the encapsulation of the code is worsened, as everything can be accessed and changed. But, this was done intentionally to provide a flexible library that can be used in various ways.

The participants also mentioned that there were quite a few similarities to other well-known libraries the participants have used before, which is represented by the theme "Similarities to other libraries". The similarities mentioned were from a positive nature with regards to the quality of the source code and the documentation. The participants liked that, similar to other software libraries, code examples were given in the documentation that were usable directly in the code. Also, the documentation within the source code was done for all classes and methods and described how methods and classes are intended to be used, which is also mentioned as a *positive* theme called "Good source code documentation". An important point that was mentioned with regards to what is often better in other libraries, was the extend of the external documentation in the Readme files. Well known open source libraries often do not only provide examples and descriptions of the use cases for the described software components but also describe the methods and parameters that can be used. This description of the methods and parameters was only done in the source code and was missing in the Readme files for the MicroDO software components.

The fact that the participants requested the implementation of more features, also outside of the sorting and rearrangement process of video editing, shows the need for such a library. This fact was captured in the theme of "Cover more Features" and was rated as *rather positive*. The theme was not rated as *positive*, as for example one participant expected the Tagging MicroDO to also support searching for tags. This shows that was functionality missing that was expected to exist. But, it was also mentioned that the library would benefit from even more features that were e.g. provided in the API usability test. An example would be the feature to play a video on a tap and only show a thumbnail while the video is not being played, which was provided for ease of implementation in the API usability test, could be extracted to a new MicroDO.

With regards to the set up of the API usability test, the theme “Comfortable” was extracted from the feedback of the participants, which is rated as *positive*. The participants generally preferred to implement the tasks during the usability test and to have a limited time frame in which the tasks had to be implemented. It was also mentioned, that the already existing code made it easier to start using the MicroDO software library right from the beginning and removed time-consuming development tasks. Only one of the participants mentioned that he would have preferred to implement the scenarios on his own and then discuss the found results afterwards. But he also pointed out, that he liked the fact that he did have a set time frame for the scenarios. He said that he would have spent a lot of time perfecting the code and implementing not required functionality instead of going on to the next task if he would have implemented the scenario on his own.

6.1.2 Negative rated themes

The documentation of the MicroDO library can be improved. While the source code documentation was mentioned to be well done and sufficient, the external documentation in form of Readme files was mentioned to be too extendable. This is encapsulated in the theme “External documentation extendable”, which was rated as *rather negative*. The Readme files contain code examples showing how to use the library, which was perceived positively. The copy & paste allowed for a quick first look at the functionality of the software components before adapting the code to fit the needs of the implementation. An issue with the code within the Reamde files that was mentioned, was that the code examples were only given in Kotlin. When using the code examples in a Java-based project, some code had to be adapted. As such it was mentioned to be a good idea to add the Java variant of the code to the Readme files as well. As mentioned before, to bring the external documentation to the standards of other well-known open-source software libraries, the methods and parameters should also be described in detail in the Readme files. Despite the “Good Naming Strategies”, the theme of “Alternating List not named understandably” describes that the naming for the Alternating List theme was not chosen properly. None of the participants that had to use the Alternating List MicroDO, identified this MicroDO to fulfill the needs of the task description. The participants were rather checking the documentation of the List-To-List MicroDO. When asked why this was the case they said that they did not understand what was meant by the Alternating List and List-To-List was the only other MicroDO referring to a list. None of the participants could suggest a better name but mentioned that it could help to include an image showing the alternating list on the primary Reamde file. The primary Readme file is displayed first when navigating to the MicroDO Github page and contains links to the Readme files for the single MicroDOs. A better name for the Alternating List MicroDO should be considered and the MicroDO API should be updated with the new name.

An issue with inconsistencies within the MicroDO software library itself was identified by the participants during the API usability test as well, which the theme “Helper Functionality Inconsistency” represents. The inconsistencies within the MicroDO library were related to the inconsistencies within the helper classes that were created to support the control of the user interface elements that were implemented within the MicroDO library. The inconsistencies were related to how the helper classes were used. For example, the `MultiMoveHelper` uses static methods, where the `View` object to update has to be given in the method. In comparison to the `MultiMoveHepler`, the `CornerGestureHelper` does not use static methods but an object is created and the `View` object to update with the helper has to be given in the constructor. Also the order of parameters was not chosen consistently and not every helper covered the same functionality. This issue came up when multiple of the helper classes were used within the same scenario for the API usability test. The inconsistencies within the MicroDO library should be resolved by choosing one of the design decisions made for the helper classes and applying the design to all helper classes. In addition an interface should be designed to ensure all helpers provide similar functionality.

There were also potential inconsistencies with the Android design guidelines identified, which is represented by the “Inconsistencies with other Android behaviour” theme. Participant Bart mentioned that the implementation of new user interactions and design ideas might not be in compliance with the Android design guidelines. It has to be considered what is more important - either to support new forms of user interactions or to fully comply with the Android design guidelines. While one of the goals of this thesis was to support the implementation of new user interactions, the compliance with the Android design guidelines was not a goal. As such, we deemed creating new possibilities for user interaction more important than complying with the Android design guidelines completely. Another inconsistency of the MicroDO library was found when comparing the behaviour of MicroDO user interface components with other user interface components within the Android framework. The custom layouts provided by the MicroDO library automatically add user interface elements that the participants do not have to define. But the order the user interface elements are added is different from the standard Android behaviour, which is known to put elements defined inside of a user interface element to be on the top. For example the `CornerGestureLayout` adds the corner buttons on top of the content put inside of the `CornerGestureLayout`. While this bevahiour is inconsistent, it was a goal of the implementation of the MicroDO library to ease the development of specific features. Requiring the participants to for example add each corner button manually would result in the participants not getting a real benefit from using the MicroDO software implementation. Anyways, if future feedback on the library suggests that this behaviour is more confusing than helpful the implementation should be changed to comply with the Android standards.

The theme of “Tailored Use Cases” encapsulates the issue that the scenarios chosen for the API usability test were designed to suggest the usage of the MicroDO software library.

In a real project, the use cases could be quite different and might result in the MicroDO software library not covering the required features. Because of this, the theme "Tailored Use Cases" and was rated as *rather negative*. Despite the use cases being tailored to be used with the MicroDO library, the API usability test covered a variety of use cases, mitigating the effect of the laboratory setting. The API usability test itself was also seen as well designed by the participants of the API usability test. Except for one participant, all participants liked the fact that each participant implemented a scenario one after the other. One participant would have preferred to implement the scenario on his own and only discuss the implementation and the found usability issues in the semi-structured interview afterwards.

6.2 Categorisation of the Usability Issues

The API usability test was built upon the API usability test described by Grill et al. (2012). The authors used a set of heuristics to categorise the usability issues found within their API which can be seen in Figure 3.3. In addition, the authors also categorised the usability issues found by the participants using those heuristics. While the heuristic evaluation by experts was not done for the MicroDO API usability test in favor of a more extensive API usability test by actual participants, it is still of interest how the found usability issues can be categorised within the heuristics chosen by Grill et al. (2012).

- **Complexity:** There were no comments by the participants take took part in the API usability test about the complexity of the MicroDO API. The participants of the API usability test also seemed to have no issues when it came to understanding how the software components work and how to use the software components within the implementation.
- **Naming:** The naming of the MicroDO software library was deemed to be appropriate during the API usability test. Only one of the MicroDOs was named in a way, that the participants that took part in the API usability test were not able to understand what the MicroDO should be used for. This was the case for the Alternating List MicroDO and as mentioned before the Alternating List MicroDO has to be renamed.
- **Caller's perspective:** There were no issues mentioned during the API usability test regarding method naming and usage within the library.
- **Documentation:** The source code documentation was done well in the MicroDO library according to the results of the API usability test. But the external documentation needs to be improved and provide more examples and more documentation about the methods and parameters.
- **Consistency and Conventions:** Regarding this heuristic, there were two main issues found within the MicroDO library. First, the helper classes to ease the control

of the user interface elements provided by the MicroDO library had inconsistencies. Second, the participants that took part in the API usability test mentioned that they were unsure if the MicroDO library was in line with the Android design guidelines and other behaviour known from using the Android software framework.

- **Parameterized constructor:** The MicroDO library heavily uses parameterized constructors for required parameters. For optional parameters, there are default values provided within the Kotlin programming language to overcome this issue. When using the MicroDO library with the Java programming language, the default values do not work and as such providing a default constructor and setter methods would be preferable for using the library with Java.
- **Factory pattern:** The MicroDO library does not use the factory pattern in any case. As such the MicroDO library is completely in line with this heuristic.
- **Data types:** There were no mentions of issues with the datatypes and the returns types of methods mentioned during the API usability test.
- **Concurrency:** There is no parallelisation done within the MicroDO library which could lead to the necessity of concurrency handling.
- **Error handling and exceptions:** There were no mentions of bad error handling within the MicroDO software library during the API usability test.
- **Leftover for client code:** All participants that took part in the API usability test mentioned, that the library was "easy to use" and "with very little code". As such the necessity for client code is rather small to use the MicroDO library - which was also a goal of the implementation.
- **Multiple ways to do one:** The participants mentioned nothing with regards to having multiple ways to achieve the same goal during the API usability tests. There was also nothing designed within the MicroDO library to support multiple ways to complete a task.
- **Long chain of references:** No issues were found during the API usability test regarding this heuristic.
- **Implementation vs. interfaces:** There was nothing mentioned during the API usability test concerning this heuristic.

Overall this means that there were seven usability issues identified with regards to the heuristics chosen by Grill et al. (2012). Most of the issues found are possible to be resolved with an update of the MicroDO software library. The Alternating List MicroDO components should be renamed to get an understanding of the functionality of the software component from its name. The constructor and methods that use default parameters should be considered to be rewritten using overloaded methods and constructors. Improvements in the documentation are necessary. The helper classes should be updated to use the same design paradigms to avoid inconsistencies. The inconsistencies with the Android design guidelines should be checked and it should be considered if the MicroDO software library needs to be updated to be in line with the Android design guidelines. In addition, it should be considered to update the MicroDO layout components to behave similarly to the Android layout components, but as mentioned before this might result in different usability issues. And as a last improvement, the documentation in the Readme files of the MicroDO software library has to be extended and updated to be more in line with regards to content and quality with other software libraries.

CHAPTER 7

Conclusion

This chapter summarizes this thesis and answers the research questions. A summary of the research done and its results are given and a conclusion is made how the research questions were answered. This is followed by a critical reflection about the work done and the results that were collected. In the end potential future work on the topics discussed in this thesis is described.

7.1 Summary

The goal of this thesis was to support researchers in creating media-intensive software applications and prototypes. A thematic analysis was conducted on student submission on the topic of new user interactions for sorting photos on smartphones. The thematic analysis resulted in seven themes, which represent features for the rearrangement and sorting process of multimedia items. Each of the themes was identified to benefit from a software library supporting the implementation of Android applications that are built upon using those features.

The first research question RQ1 was "What is missing in current software-frameworks to better support the implementation of media-intensive user-interfaces on Android". The themes extracted by conducting the thematic analysis provide an answer to this question. This shows that thematic analysis can be suitable to identify what is missing in current software frameworks to support the implementation of media-intensive user-interfaces on Android.

The requirements for implementing the software components were defined by writing UserX Stories based on the found themes. With the UserX Stories, a software library called MicroDO was implemented, with the goal to ease the implementation of the features defined in the UserX Stories. Eleven software components were implemented based on the defined UserX Stories. Each software component was called a MicroDO.

UserX Stories successfully defined the requirements for the software components needed to implement the missing features in current software-frameworks.

The second research question RQ2 was "How can these missing features in current software-frameworks be identified and defined?". By first conducting the thematic analysis on the student submission to identify the missing features and then defining the missing features using UserX Stories an answer for RQ2 could be found.

The MicroDO software library was then evaluated by conducting an API usability test. With the API usability test, usability issues within the MicroDO software library were found. A variety of different use cases were possible to be implemented during the API usability test and all of the participants who took part in the API usability test mentioned that they could consider using the MicroDO software library in future software projects with fitting requirements. The participants also suggested to cover more features and extend the functionality of the existing MicroDOs to cover more use cases, which shows that demand for such a software library exists. But, the API usability test was conducted in a laboratory setting as the scenarios were designed to ensure that the MicroDO software library can cover the designed use cases. To mitigate this effect, each scenario covered a different use case. However, in a real software project, the results could still be quite different.

The third research question RQ3 was "How can the software for these missing features be designed to be flexible and modular?". The design decisions of using small and generic software components, as well as making the whole source code fully extendable, provides an answer to research question RQ3. The participants of the API usability test were able to use multiple software components of the MicroDO software library in combination and together with default Android software components. Also, the participants were able to provide new functionality by extending existing classes from the MicroDO software library or using the provided classes in different contexts. As such the goal of easing the implementation of Android applications for a set of user interactions and user interfaces was achieved. But, it was also identified that there are aspects of the MicroDO software library that would benefit from further improvements. The name of the Alternating List MicroDO should be changed as the name does not reflect the functionality contained in the software component. Also, the external documentation should be extended as the participants of the API usability test rather used the external documentation than the source code documentation. For example, a description of the methods and parameters should be added, which is also done in other open-source software libraries. Also, the helper classes which were designed to make it easier to update the implemented user interface components from within the code should be redesigned to use similar design principals.

7.2 Critical Reflection

When looking back at the research, there was a lot learned in between conducting the first thematic analysis on the student submissions and conducting the second thematic analysis on the results of the API usability test. The issue that conducting a thematic analysis for the first time can be difficult was also mentioned by (Nowell et al., 2017) as a big disadvantage of conducting a thematic analysis. In future research, other qualitative methods like qualitative content analysis by Mayring (1985) could be more fitting to identify missing software components. The second thematic analysis was a lot easier to conduct.

Also, when doing similar research in the future, the source for identifying the missing software components could be chosen differently. While the positive side of the student submissions was the larger number of different people providing ideas, the negative side was that the students providing the ideas were not experts in the field of user interaction and user interface design and also not Android participants. As such the students were not the target user group for MicroDO software library. In future studies on similar topics, a discussion with experts int the field would be considered to identify the missing software components.

With regards to the implementation of the MicroDO library, a lot was learned about the Android framework and the Kotlin programming language during the development. Despite iterating over all components regularly and applying new learnings, the software library would potentially be quite different if it would be implemented from the beginning again.

Also, the effort of creating different scenarios for each of the participants who took part in the API usability test was underestimated. Not only did it require to create multiple descriptions and mock-ups but also the base software project for the API usability tests had to be adapted for every scenario. For future API usability tests, it would be beneficial to create smaller tasks and provide them to the participants in different orders to ensure that each software component was used during the API usability test.

Another aspect of the API usability test that was not expected was the necessity to conduct the API usability test remotely due to the COVID-19 pandemic. Initially, the API usability test was designed to be conducted together with the participants in the same location. As the measures to reduce the spread of the COVID-19 virus were announced the design of the API usability test had to be unexpectedly adopted. The positive side of conducting the API usability test remotely was, that the library was tested on different devices and as such, it was ensured that the software library will be usable by anyone. Also, there was no need for the participants of the API usability test to travel anywhere, which reduced the time needed to take part in the API usability test. But there were also challenges that had to be tackled. There existed issues with internet connectivity which reduced the resolution of the shared screen during the video call or shortly interrupted the call. Also, despite the participants installing all required software beforehand, there were still parts of the set up like copying the videos used during the API usability test to the correct folder on the Android device used, which required time at the beginning of the API usability test. If the API usability test would have been conducted in the same location, the participants would have been provided with a fully setup notebook and Android device to remove the factor of setting up the project from the API usability test.

In addition, while the video call platform “Let’s Meet” performed quite well and was usable without any installation, a more powerful platform could be beneficial for future remote API usability tests. Features that could help to reduce the negative effects of conducting the API usability test would be the possibility to point somewhere on the participant’s screen or to be able to control the PC of the participant for a short time to complete the setup. Personally conducting the API usability test on location instead of remotely would be preferred for future API usability tests.

7.3 Future Work

In future studies, more use cases can be supported by extending the MicroDO software library. The use cases could also be chosen outside of the field of rearrangement and sorting of multimedia items. By increasing the scope of covered features the MicroDO library could be used in a variety of projects and the feature set implemented with this thesis cannot cover all possible use cases right now. The participants of the API usability test mentioned, they could consider using the MicroDO software library in a future software project if the requirements were met. Also, the currently existing code can be reworked with further learnings to improve performance and usability.

While the MicroDO library is focused on supporting the implementation of user interaction and user interfaces, it could still be interesting to add implementations for supporting automation. One idea would be to add algorithms to automatically sort or categorise photos and videos. The automatic sorting was also identified as a theme during the thematic analysis of the student submissions but dropped in favor of supporting user interface and user interaction based themes. Supporting user interactions with automated algorithms was identified to be able to improve the usability of a software application. The literature on this topic provides ideas on how to support user interaction with automation. (Hudelist, 2013; Zhang et al., 2014; Taylor and Qureshi, 2016)

Another field for further research is the implementation of more user interfaces and new user interactions using the MicroDO library. With the MicroDO software library, it should be easier to implement and compare prototypes for new user interactions and user interfaces for the sorting and arrangement of multimedia items. Through comparison and hands-on usability tests, the new ideas for user interactions and user interfaces can be verified and analysed to bring better user experience to smartphone users in the future.

List of Figures

2.1 The activity lifecycle of an Android applicaiton. (Google, 2020) 10
2.2 The interface proposed by Ganhör and Güldenpfennig (2015, p. 346). The top two rows allow for different detail levels of browing, while the bottom row is used to select images to be filed back into the center browsing row. . . . 11
2.3 The video editing process. The source videos are first browsed through and selected and then edited. The final video can be the source for another video editing process. 12
2.4 Assembly view in Adobe Premiere 2020 (Created on 14.11.2019) 13
2.5 Screenshots of selection and editing view of “You Cut - Video Editor” (Created on 14.11.2019) 14
2.6 Overview of the screens of the video editing application by Jokela et al. (2007, p. 348) 15
2.7 The new ideas for video browsing by Hudelist et al. (2013a,b) 16
2.8 The progressive scrolling view of “Propane”. (Ganhör, 2012, p. 4) 17
2.9 The “Muvee” user interface. (Ganhör, 2014, p. 232) 18
2.10 The “VideoGraph” depending on the used graph algorithm. (Zhang et al., 2014, p. 1129) 19

3.1 Example of a UserX Stories written by a participant of the evaluation done by Choma et al. (2016, p. 138). 26
3.2 Visualisation of the different stages of the API usability test. (Grill et al., 2012, p. 168) 29
3.3 Heuristics chosen by Grill et al. (2012, p. 171) 30

4.1 Mindmap of themes and subthemes after phase 3 of the thematic analysis of the student submissions. 32
4.2 Mindmap of themes and grouped into subthemes after phase 4 of the thematic analysis. 33
4.3 Sketch of the Alternating List theme 34
4.4 Sketch of the List-To-List theme 35
4.5 Sketch of the Parking theme 35
4.6 Sketch of the Side Tags theme 36
4.7 Sketch of the multimove theme 36

4.8 Sketch of the corner gestures theme 37
4.9 Sketch of the swipeselect theme 37
4.10 UX flow of an element being removed from the list (left side), the animation happening (center) and added to the list (right side) 39
4.11 Mockup of a vertical alternating list 40
4.12 Mockup of a tagging User Interface. The sides of the lists elements get colored based on the assigned tags. 41
4.13 Mockup of the parking area. The elements can be placed anywhere inside the gray area at the bottom and dragged out to a different location again. . . 42
4.14 Mockup of the Corner Gestures. 43
4.15 Mockup of the Swipeselect gesture. All elements swiped over (left image) are set as selected (right image). 44
4.16 Mockup of the multimove gesture 45
4.17 The Kanban board used on Github during development 48
4.18 The (a) `MultiMoveLayout` and (b) `CornerGestureLayout` as seen in a fullscreen Android application. 51
4.19 The (a) `SwipeSelectLayout` and (b) `SideTagsLayout` as seen in a fullscreen Android application. 52

5.1 Mindmap of themes and subthemes for the API usability test analysis after phase 3 of the thematic analysis. 68
5.2 Mindmap of themes and subthemes for the API usability test analysis after phase 4 of the thematic analysis. 69

List of Tables

2.1 Gestures used on touchscreens. The gesture names and definitions under quotation marks are taken from Saffer (2009, Appendix A). The recommendations of when to use the gestures are rephrased from Saffer (2009, Chapter 3). 7

4.1 Codes found during thematic analysis 32
4.2 UserX Story 1 - List-To-List 39
4.3 UserX Story 2 - Alternating List 40
4.4 UserX Story 3 - Tagging 41
4.5 UserX Story 4 - Parking 42
4.6 UserX Story 5 - Corner Gestures 43
4.7 UserX Story 6 - Swipeselect 44
4.8 UserX Story 7 - MultiMove 45

5.1 Questionnaire handed to participants 62
5.2 Demographic data of the participants collected during the API usability test 64
5.3 Codes extracted from the API usability test during phase two of the thematic analysis. 67

Bibliography

Adams, W. C. (2015). *Conducting Semi-Structured Interviews*, chapter 19, pages 492–505. John Wiley Sons, Ltd.

Alhadreti, O. and Mayhew, P. (2018). Rethinking thinking aloud: A comparison of three think-aloud protocols. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–12, New York, NY, USA. Association for Computing Machinery.

Attride-Stirling, J. (2001). Thematic networks: an analytic tool for qualitative research. *Qualitative Research*, 1(3):385–405.

Beck, K. (1998). Extreme programming: A humanistic discipline of software development. In Astesiano, E., editor, *Fundamental Approaches to Software Engineering*, pages 1–6, Berlin, Heidelberg. Springer Berlin Heidelberg.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for agile software development. `http://www.agilemanifesto.org/`. Accessed: 01.06.2020.

Bore, C. and Bore, S. (2005). Profiling software api usability for consumer electronics. In *2005 Digest of Technical Papers. International Conference on Consumer Electronics, 2005. ICCE.*, pages 155–156.

Boyatzis, R. E. (1998). *Transforming qualitative information: Thematic analysis and code development.* Sage Publications, Inc.

Braun, V. and Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology*, 3:77–101.

Buhne, S., Halmans, G., Pohl, K., Weber, M., Kleinwechter, H., and Wierczoch, T. (2004). Defining requirements at different levels of abstraction. In *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.*, pages 346–347.

Burnard, P. (1991). A method of analysing interview transcripts in qualitative research. *Nurse education today*, 11(6):461—466.

Cecere, G., Corrocher, N., and Battaglia, R. D. (2015). Innovation and competition in the smartphone industry: Is there a dominant design? *Telecommunications Policy*, 39(3):162 – 175. New empirical approaches to telecommunications economics: Opportunities and challenges Mobile phone data and geographic modelling.

Choma, J., Zaina, L. A. M., and Beraldo, D. (2016). Userx story: Incorporating ux aspects into user stories elaboration. In Kurosu, M., editor, *Human-Computer Interaction. Theory, Design, Development and Practice*, pages 131–140, Cham. Springer International Publishing.

Cobârzan, C., Hudelist, M. A., and Del Fabro, M. (2014). Content-based video browsing with collaborating mobile clients. In Gurrin, C., Hopfgartner, F., Hurst, W., Johansen, H., Lee, H., and O'Connor, N., editors, *MultiMedia Modeling*, pages 402–406, Cham. Springer International Publishing.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison-Wesley signature series. Addison-Wesley.

Cooper, A., Reimann, R., and Cronin, D. (2007). *About face 3 : the essentials of interaction design*. Wiley Pub., Indianapolis, IN, 3rd ed.. edition.

Dai, B., Zhang, Y., Cai, D., and Wang, A. (2017). Html5-based interactive media editing system. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 1185–1189.

Darwaish, S. F., Moradian, E., Rahmani, T., and Knauer, M. (2014). Biometric identification on android smartphones. *Procedia Computer Science*, 35:832 – 841. Knowledge-Based and Intelligent Information Engineering Systems 18th Annual Conference, KES-2014 Gdynia, Poland, September 2014 Proceedings.

Denham, M. A. and Onwuegbuzie, A. J. (2013). Beyond words: Using nonverbal communication data in research to enhance thick description and interpretation. *International Journal of Qualitative Methods*, 12(1):670–696.

Drucker, S. M., Fisher, D., and Basu, S. (2011). Helping users sort faster with adaptive machine learning recommendations. In Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P., and Winckler, M., editors, *Human-Computer Interaction – INTERACT 2011*, pages 187–203, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ericsson, K. A. and Simon, H. A. (1984). *Protocol analysis: Verbal reports as data*. The MIT Press.

Farooq, U., Welicki, L., and Zirkler, D. (2010). Api usability peer reviews: A method for evaluating the usability of application programming interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 2327–2336, New York, NY, USA. Association for Computing Machinery.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental PSychology*, 74:381–391.

Francese, R., Gravino, C., Risi, M., Scanniello, G., and Tortora, G. (2017). Mobile app development and management: Results from a qualitative investigation. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, MOBILESoft '17, pages 133–143, Piscataway, NJ, USA. IEEE Press.

Ganhör, R. (2012). Propane: Fast and precise video browsing on mobile phones. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, MUM '12, pages 20:1–20:8, New York, NY, USA. ACM.

Ganhör, R. and Güldenpfennig, F. (2015). Insert: Efficient sorting of images on mobile devices. In *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction*, OzCHI '15, page 343–351, New York, NY, USA. Association for Computing Machinery.

Ganhör, R. (2014). Muvee: An alternative approach to mobile video trimming. In *2014 IEEE International Symposium on Multimedia*, pages 229–236.

Google (2020). Understand the activity lifecycle. `https://developer.android.com/guide/components/activities/activity-lifecycle`. Accessed: 16.08.2020.

Grill, T., Polacek, O., and Tscheligi, M. (2012). Methods towards api usability: A structural analysis of usability problem categories. In Winckler, M., Forbrig, P., and Bernhaupt, R., editors, *Human-Centered Software Engineering*, pages 164–180, Berlin, Heidelberg. Springer Berlin Heidelberg.

Han, Q. and Cho, D. (2016). Characterizing the technological evolution of smartphones: Insights from performance benchmarks. In *Proceedings of the 18th Annual International Conference on Electronic Commerce: E-Commerce in Smart Connected World*, ICEC '16, New York, NY, USA. Association for Computing Machinery.

Harrison, M. and Barnard, P. (1993). On defining requirements for interaction. In *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 50–54.

Hudelist, M. A. (2013). Next generation image and video browsing on mobile devices. In *Proceedings of the 3rd ACM Conference on International Conference on Multimedia Retrieval*, ICMR '13, pages 333–336, New York, NY, USA. ACM.

Hudelist, M. A., Schoeffmann, K., and Boeszoermenyi, L. (2013a). Mobile video browsing with a 3d filmstrip. In *Proceedings of the 3rd ACM Conference on International Conference on Multimedia Retrieval*, ICMR '13, pages 299–300, New York, NY, USA. ACM.

Hudelist, M. A., Schoeffmann, K., and Boeszoermenyi, L. (2013b). Mobile video browsing with the thumbbrowser. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, pages 405–406, New York, NY, USA. ACM.

IEEE (1998). Ieee recommended practice for software requirements specifications. *IEEE Std 830-1998*, pages 1–40.

Islam, A., Chebil, F., and Hourunranta, A. (2006). Efficient algorithms for editing h.263 and mpeg-4 videos on mobile terminals. In *2006 International Conference on Image Processing*, pages 3181–3184.

Jokela, T., Karukka, M., and Mäkelä, K. (2007). Mobile video editor: Design and evaluation. In Jacko, J. A., editor, *Human-Computer Interaction. Interaction Platforms and Techniques*, pages 344–353, Berlin, Heidelberg. Springer Berlin Heidelberg.

Karuzaki, E. and Savidis, A. (2015). Yeti: Yet another automatic interface composer. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '15, pages 12–21, New York, NY, USA. ACM.

Knych, T. W. and Baliga, A. (2014). Android application development and testability. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*, MOBILESoft 2014, pages 37–40, New York, NY, USA. ACM.

Kotlin Foundation (2020). Kotlin. `https://kotlinlang.org//`. Accessed: 07.08.2020.

Leech, B. L. (2002). Asking questions: Techniques for semistructured interviews. *PS: Political Science and Politics*, 35(4):665–668.

London, K. R. (2003). *Documentation*, page 602–608. John Wiley and Sons Ltd., GBR.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E. M., and Brinkkemper, S. (2015). Forging high-quality user stories: Towards a discipline for agile requirements. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 126–135.

Mayring, P. (1985). Qualitative inhaltsanalyse. In Jüttemann, G., editor, *Qualitative Forschung in der Psychologie. Grundfragen, Verfahrensweisen, Anwendungsfelder*, pages 187–211. Weinheim: Beltz.

Merriam-Webster.com (2020). "edit". `https://www.merriam-webster.com/dictionary/editing`. Accessed: 2020-03-06.

Mohanani, R., Ralph, P., and Shreeve, B. (2014). Requirements fixation. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 895–906, New York, NY, USA. Association for Computing Machinery.

Naderifar, M., Goli, H., and Ghaljaie, F. (2017). Snowball sampling: A purposeful method of sampling in qualitative research.

Nielsen, J. (1995). 10 usability heuristics for user interface design. `https://www.nngroup.com/articles/ten-usability-heuristics/`. Accessed: 02.06.2020.

Nowell, L. S., Norris, J. M., White, D. E., and Moules, N. J. (2017). Thematic analysis: Striving to meet the trustworthiness criteria. *International Journal of Qualitative Methods*, 16(1):1609406917733847.

Nuseibeh, B. and Easterbrook, S. (2000). Requirements engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, page 35–46, New York, NY, USA. Association for Computing Machinery.

Ott, C., Hebecker, R., and Wakes, S. (2012). Picture the space: Three concepts for management and presentation of personal digital photographs. In *Proceedings of the 13th International Conference of the NZ Chapter of the ACM's Special Interest Group on Human-Computer Interaction*, CHINZ '12, page 1–8, New York, NY, USA. Association for Computing Machinery.

Panizzi, E. and Marzo, G. (2014). Multidimensional sort of lists in mobile devices. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, AVI '14, page 375–376, New York, NY, USA. Association for Computing Machinery.

Paulheim, H. and Erdogan, A. (2010). Seamless integration of heterogeneous ui components. In *Proceedings of the 2Nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '10, pages 303–308, New York, NY, USA. ACM.

Puikkonen, A., Häkkilä, J., Ballagas, R., and Mäntyjärvi, J. (2009). Practices in creating videos with mobile phones. In *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '09, pages 3:1–3:10, New York, NY, USA. ACM.

Radinger, W. and Goeschka, K. M. (2003). Agile software development for component based software engineering. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA '03, page 300–301, New York, NY, USA. Association for Computing Machinery.

Rautiainen, K. (2004). Chapter 12: Kanban for software development. In Heikkilä, V., Rautiainen, K., and Vähäniitty, J., editors, *Towards Agile Product and Portfolio Management*, chapter 12, pages 184–192. Espoo: Aalto University.

Saffer, D. (2009). *Designing gestural interfaces.* O'Reilly, Sebastopol, first edition.. edition.

Schoeffmann, K., Hudelist, M. A., and Huber, J. (2015). Video interaction tools: A survey of recent work. *ACM Comput. Surv.*, 48(1).

Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall PTR, USA, 1st edition.

StatCounter (2020). Mobile operating system market share worldwide. `https://gs.statcounter.com/os-market-share/mobile/worldwide`. Accessed: 2020-05-31.

Sugimori, Y., Kusunoki, K., Cho, F., and Uchikawa, S. (1977). Toyota production system and kanban system materialization of just-in-time and respect-for-human system. *International Journal of Production Research*, 15(6):553–564.

Taylor, W. and Qureshi, F. Z. (2016). Automatic video editing for sensor-rich videos. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9.

van Lamsweerde, A. (2001). Goal-oriented requirements engineering: a guided tour. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, pages 249–262.

Wörndl, W., Weicker, J., and Lamche, B. (2013). Selecting gestural user interaction patterns for recommender applications on smartphones. In Chen, L., de Gemmis, M., Felfernig, A., Lops, P., Ricci, F., Semeraro, G., and Willemsen, M. C., editors, *Proceedings of the 3rd Workshop on Human Decision Making in Recommender Systems in conjunction with the 7th ACM Conference on Recommender Systems (RecSys 2013), Hong Kong, China, October 12, 2013*, volume 1050 of *CEUR Workshop Proceedings*, pages 17–20. CEUR-WS.org.

Yu, L. and Liao, X. (2013). Cloud based mobile video editing system. In *2013 IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*, pages 378–382.

Zeaaraoui, A., Bougroun, Z., Belkasmi, M. G., and Bouchentouf, T. (2013). User stories template for object-oriented applications. In *Third International Conference on Innovative Computing Technology (INTECH 2013)*, pages 407–410.

Zhang, L., Xu, Q.-K., Nie, L.-Z., and Huang, H. (2014). Videograph: a non-linear video representation for efficient exploration. *The Visual Computer*, 30(10):1123–1132.

Attachements

Usability Test Introduction

Welcome to the MicroDO usability test!

The background of this usability test is the master thesis I am currently working on. The topic of this thesis is “Software components supporting the implementation of multimedia applications for Android” with special focus on the rearrangement process of video editing on mobile devices. Within this thesis a small software library called MicroDO has been implemented. This software library should ease the implementation of new prototypes for video editing applications on Android.

The goal of this usability test is to find out if the provided MicroDO API supports the implementation of video editing applications as intended and what could be improved upon further development of the library. During this usability test you will be asked to implement simple user interfaces for Android using Kotlin within the context of the rearrangement of Videos. For this you will be asked to use the MicroDO library wherever possible.

If you are ok with it, the implementation process and everything you say will be recorded to evaluate the test afterwards. All footage and audio will be deleted after the assessment. To ensure your privacy, all comments will be anonymized within the written thesis.

Feel free to say anything (good and bad) that crosses your mind during the development as any remark can be useful to improve the quality of the MicroDO library. Any provided mockups should only be used as a reference and it is not required to have everything look exactly as displayed on them. You are not required to fully implement all of the user stories handed to you and any parts can be dismissed or simplified if the effort is higher than expected. Overall the implementation time should not be longer than 1.5 hours.

If you have questions feel free to ask them, anytime. Also repeatedly. It is ok.

Thank you for your participation.

Data

Age:			
Years of Experience with Android:			
< 1	1 - 3	3 - 6	> 6
Years of Experience with Kotlin			
< 1	1 - 3	3 - 6	> 6
Years of Experience with Programming in General			
< 1	1 - 5	5 - 10	> 10
Experience with video editing			
	yes	no	

Background Story:

You have been asked to implement a prototype for a new video editing Android application. It was suggested to use the new MicroDO library to implement this prototype. An Android Studio project with the MicroDO library already included has been handed to you. In addition a set of user stories was provided to show what is expected for this prototype.

Usability Test Scenario 1

Scenario 1

1.1 Tag Videos

Story:

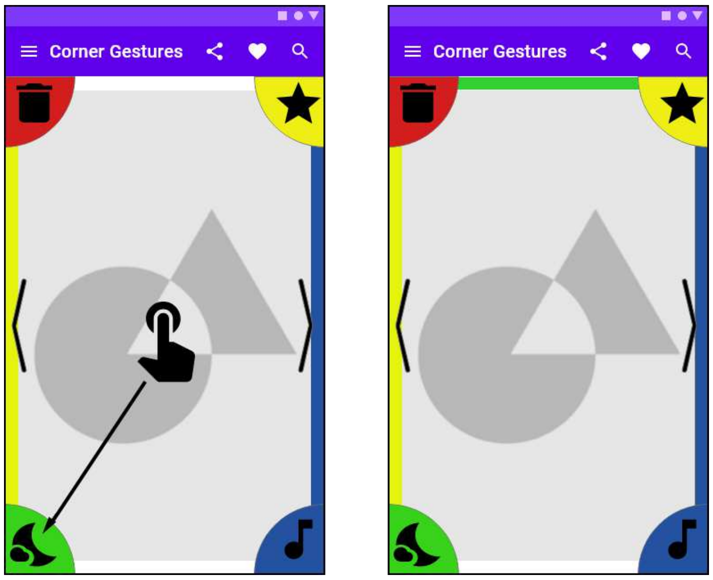
As a person who edits videos,
I want to tag video files with custom meta data,
for this the system allows me to swipe towards one of the corners of a displayed video,
when a corner is reached, the video gets tagged with the tag assigned to the respective corner.

I evaluate that my goal was achieved when the video shows the newly assigned tag.

Acceptance Criteria

- The video can be swapped by click on the arrows to the left or the right.
- When swiped to the top right, the displayed video gets tagged as “favorite”
- When swiped to the top left, the displayed video gets tagged as “deleteable”
- When swiped to the bottom left, the displayed video gets tagged as “nighttime”
- When swiped to the bottom right, the displayed video gets tagged as “music”
- When a tag is already assigned, it gets removed

Mockup



1.2 Filter Tagged Videos

Story:

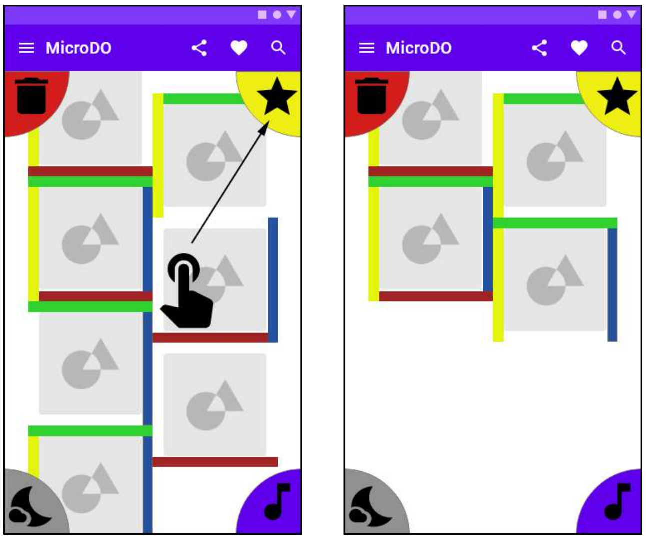
As a person who edits videos,
I want to filter my videos by tag,
for this the system allows me to swipe to a corner of my list,
when a corner is selected the list is filtered by having the tag assigned to the corner.

I evaluate that my goal was achieved when only the files with the assigned tag are shown.

Acceptance Criteria

- When swiped to the top right, only the file tagged as “favorite” are shown
- When swiped to the bottom left, only the files tagged as “night” are shown
- When swiped to the bottom right, only the files tagged as “music” are shown
- When swiped to the top left, only files tagged as “deletable” are shown
- When swiped to the same corner a second time, all files are shown

Mockup



1.3 Move All Filtered Items Into Different List

Story:

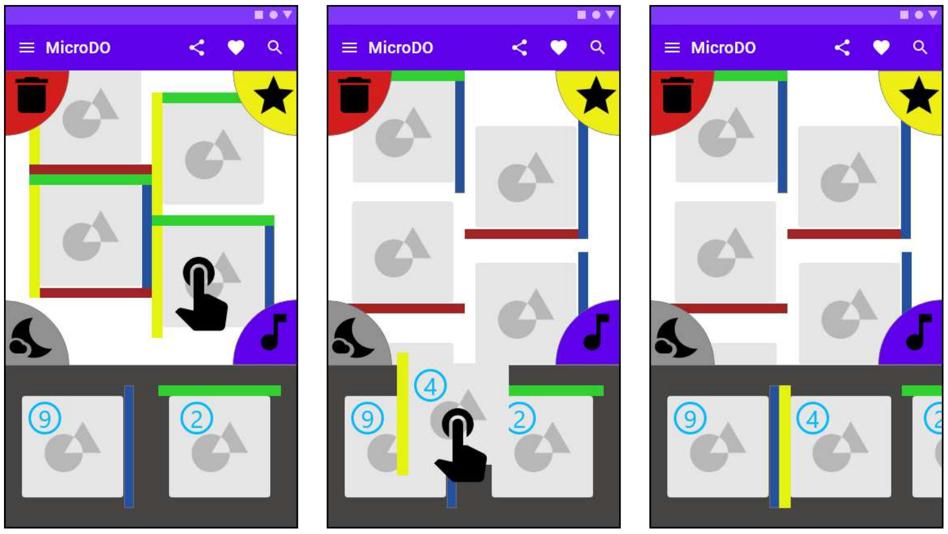
As a person who edits videos,
I want to move all previously filtered files into another List,
for this the system allows me to drag all elements filtered before into another list,
when the videos are dropped, they are part of the new list and no longer part of the original one.

I evaluate that my goal was achieved, when all previously filtered videos are no longer part of the original list, but part of the target list.

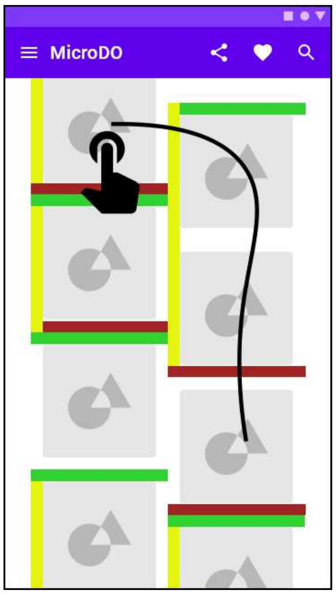
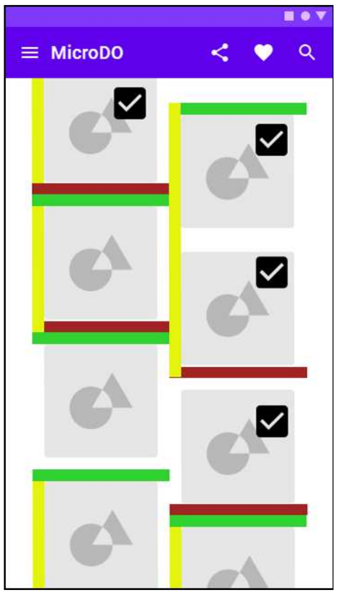
Acceptance Criteria

- When long clicked on one of the views that is currently shown, all currently shown elements are removed and become draggable
- The draggable object shows the long clicked element and the number of dragged elements on it
- When dropped into the list below, the elements get inserted as a single element, still showing how many elements have been dragged
- When dropped anywhere else, the elements get inserted back into the list
- After a successful drop, all elements are shown again

Mockup



Scenario 2

2.1 Select multiple videos by swiping
Story: **As a** person who edits videos, **I want to** select multiple videos within a list, **for this the system allows me to** long click on the list and then swipe over the wanted videos, **when** an element is selected a checkmark appears. **I evaluate that my goal was achieved when** all Elements I swiped over are marked as selected.
Acceptance Criteria ● When swiped over the list after a long click, all swiped over elements are marked as selected ● When swiped over without a long click, nothing happens ● When swiped over an element multiple times, it stays selected
Mockup  

2.2 Tag all selected elements via corners

Story:

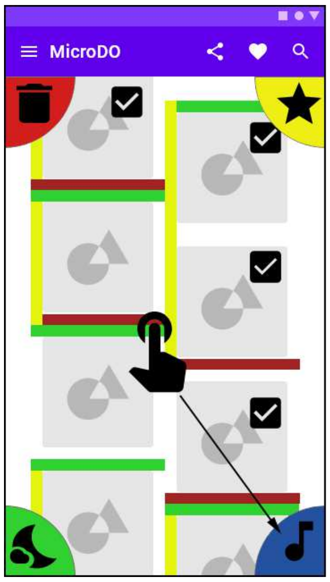
As a person who edits videos,
I want to tag multiple videos at once,
for this the system allows me to swipe towards one of the corners of the list,
when a corner is reached, the selected videos get tagged with the tag assigned to the respective corner.

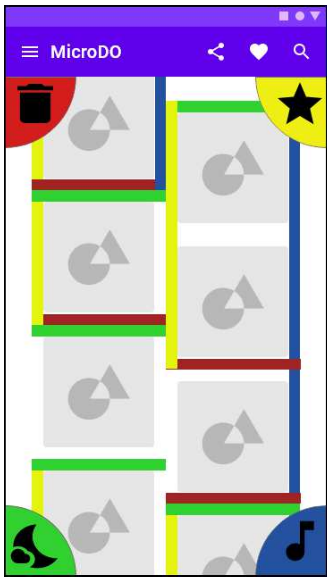
I evaluate that my goal was achieved when all selected elements show the newly assigned tag.

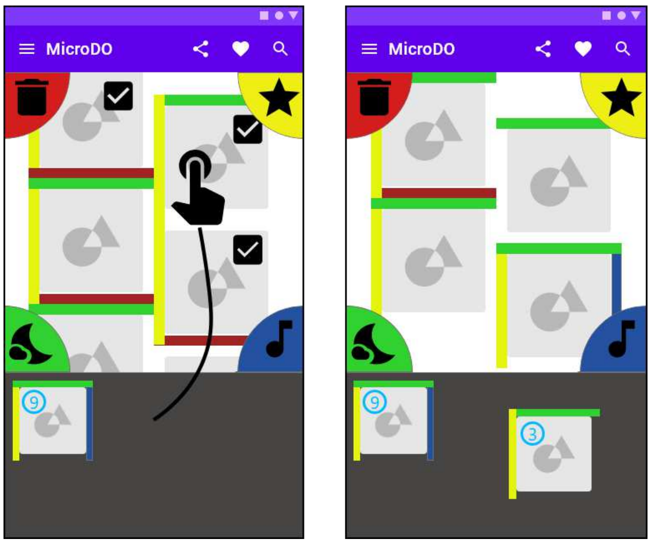
Acceptance Criteria

- When swiped to the top right, the selected elements get tagged as “favorite”
- When swiped to the top left, the selected elements get tagged as “deleteable”
- When swiped to the bottom left, the selected elements get tagged as “nighttime”
- When swiped to the bottom right, the selected elements get tagged as “music”
- When no elements are selected, swiping to any corner does nothing
- After a tag, all elements get marked as unselected

Mockup





2.3 Drag all selected videos into parking area
Story: **As a** person who edits videos, **I want to** park all selected videos for later usage, **for this the system allows me to** drag all videos into the parking area below, **when** the drag starts, the selected elements are removed and remain as a single element within the parking area. **I evaluate that my goal was achieved, when** the previously selected elements are no longer in the list and the element visualizing them can be found in the parking area.
Acceptance Criteria ● When a selected element is long clicked, drag and drop is started ● The elements are removed from the list on drag ● The dragging object shows the long click element and the number of elements currently being dragged ● When dropped in the parking area, the elements removes where dropped and is only scaled down slightly
Mockup 

Usability Test Scenario 3

Scenario 3

3.1 Move videos into the timeline

Story:

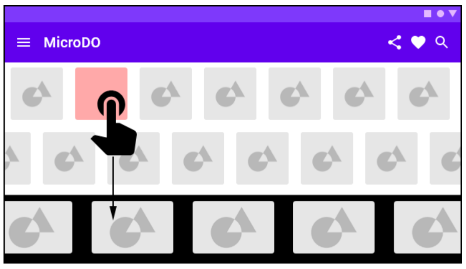
As a person who edits videos,
I want to move videos into the timeline,
for this the system allows me to drag and drop videos from a list showing my videos in a list representing the timeline,
when a video is dragged into the timeline, it is no longer part of the original list.

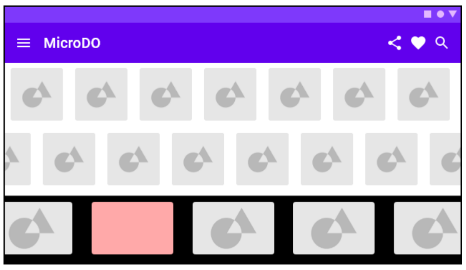
I evaluate that my goal was achieved when the video is no longer in the original list but in the timeline.

Acceptance Criteria

- When a video is swiped over, drag and drop is started
- When the video is dropped over one of the lists, it is inserted into that list at the position it has been dragged to

Mockup





3.2 Show parking area

Story:

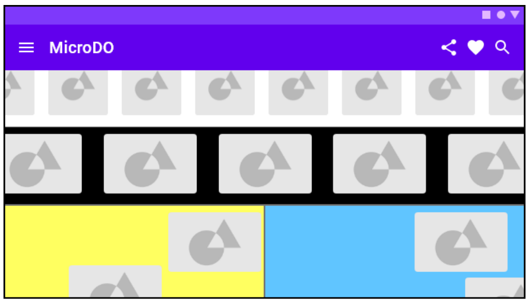
As a person who edits videos,
I want to switch from the list to the parking areas below the timeline,
for the system allows me to scroll the screen vertically,
when scrolled down 2 parking areas are revealed.

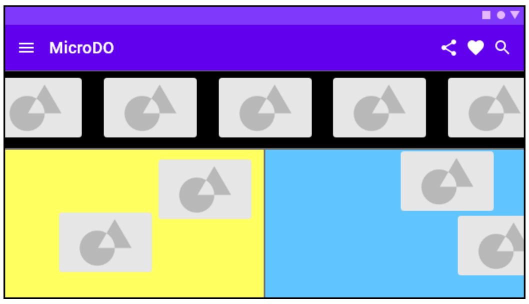
I evaluate that my goal was achieved, when both the parking areas are visible below the timeline.

Acceptance Criteria

- Vertical scrolling is enabled
- Below the timeline two parking areas get visible

Mockup





3.3 Park videos in the parking area

Story:

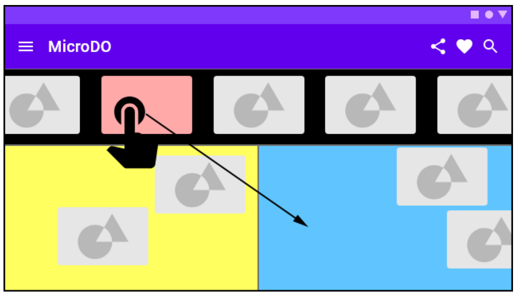
As a person who edits videos,
I want to move videos from the parking area into the timeline above or the other way around,
for this the system allows me to drag and drop videos between the parking areas and the timeline,
when the drag starts the video gets removed from the source and is inserted into the target on drop.

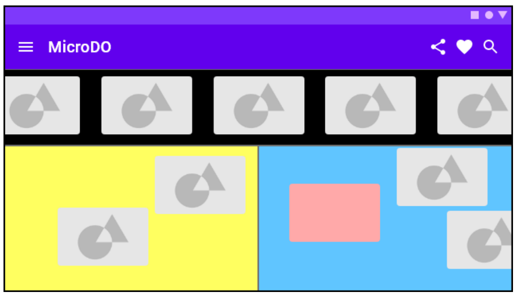
I evaluate that my goal was achieved, when a video can be dragged out of the parking area and dropped into the timeline below.

Acceptance Criteria

- When a video is swiped over, drag and drop is started
- When the video is dropped over one of the parking areas or on the timeline, it is inserted into that target

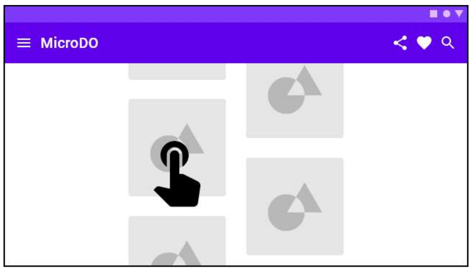
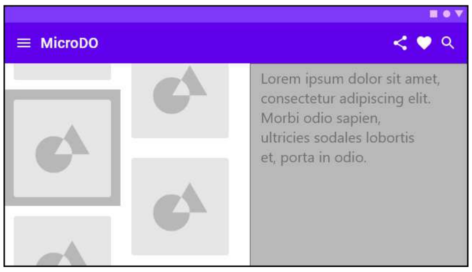
Mockup





Usability Test Scenario 4

Scenario 4

4.1 Store any text
Story: **As a** person who edits videos, **I want to** store custom text with my videos for the editing process, **for this the system allows me to** store any text with a video I click on, **when** I click on the video again the text is stored. **I evaluate that my goal was achieved, when** I have the written text stored with the video even after restart of the app.
Acceptance Criteria ● When clicked on a video in the list, a text box appears on the side, showing the stored text for the video ● If no text is stored, an empty text box is shown ● When clicked on the video again or a different video, the text is stored as a tag
Mockup       Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi odio sapien, ultricies sodales lobortis et, porta in odio.

4.2 Park videos in parking area

Story:

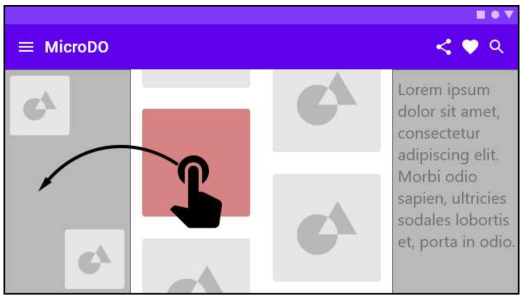
As a person who edits videos,
I want to store items for later use,
for this the system allows me to drag a video from the list into a parking area and back,
when a video has been dragged it is no longer part of the original source, but part of the target.

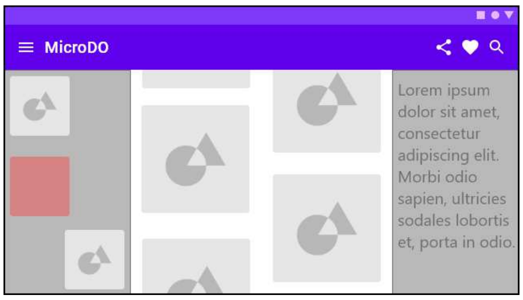
I evaluate that my goal was achieved, when the dragged video is no longer part of the original source, but part of the target.

Acceptance Criteria

- When long clicked on a video, drag and drop starts and the video is removed from the original source
- When dropped within the parking area or list, it gets inserted at the current position

Mockup


MicroDO
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi odio sapien, ultricies sodales lobortis et, porta in odio.


MicroDO
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi odio sapien, ultricies sodales lobortis et, porta in odio.

4.3 Add videos to timeline

Story:

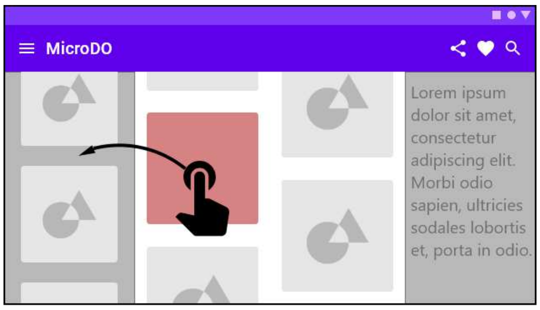
As a person who edits videos,
I want to store items for later use,
for this the system allows me to drag a video from the list into the timeline on the right,
when a video has been dragged it is no longer part of the original source, but part of the target.

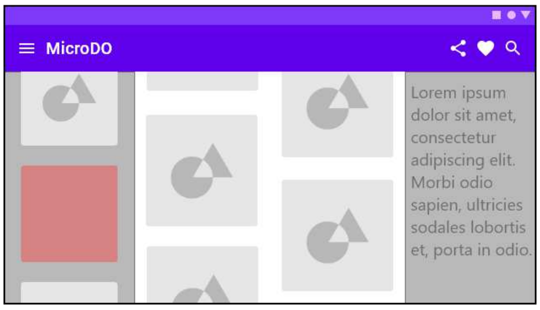
I evaluate that my goal was achieved, when the dragged video is no longer part of the original source, but part of the target.

Acceptance Criteria

- When long clicked on a video, drag and drop starts and the video is removed from the original source
- When dropped within the timeline or list, it gets inserted at the current position

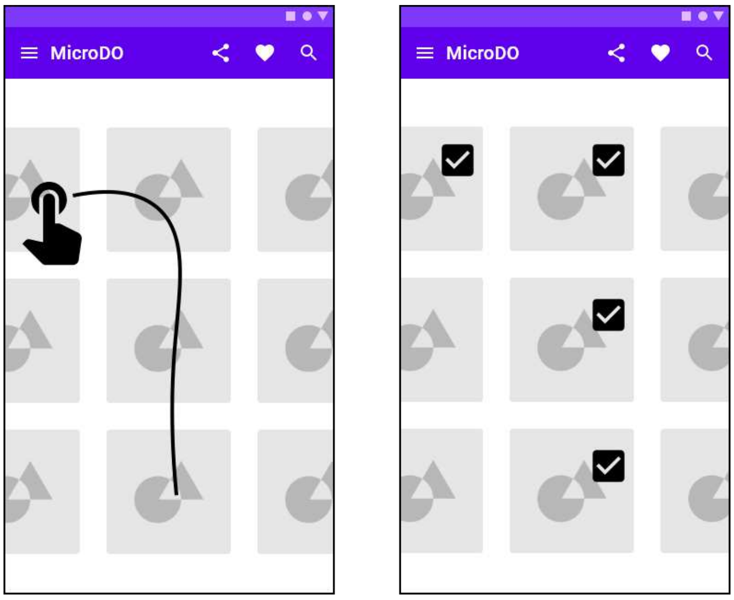
Mockup


MicroDO
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi odio sapien, ultricies sodales lobortis et, porta in odio.


MicroDO
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi odio sapien, ultricies sodales lobortis et, porta in odio.

Usability Test Scenario 5

Scenario 5

5.1 Select multiple videos by swiping
Story: **As a** person who edits videos, **I want to** select multiple videos within a list, **for this the system allows me to** swipe over the wanted elements or click on them, **when** an element is selected a checkmark shows this. **I evaluate that my goal was achieved when** all elements I swiped over are marked as selected.
Acceptance Criteria ● When swiped over the list after a long click, all swiped over elements are marked as selected ● When swiped over without a long click, nothing happens ● When swiped over an element multiple times, it stays selected
Mockup 

5.2 Drag videos into buckets

Story:

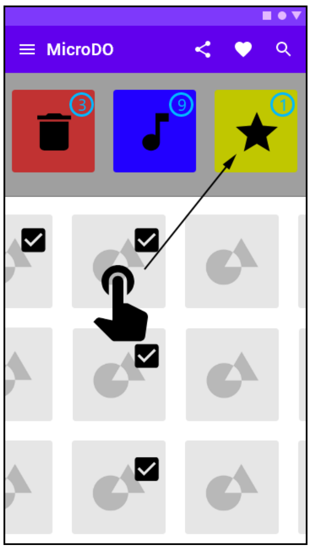
As a person who edits videos,
I want to quickly sort elements into buckets,
for this the system allows me to drag all selected videos into buckets on the top,
when items are added to buckets, the number showing the amount of elements included increases

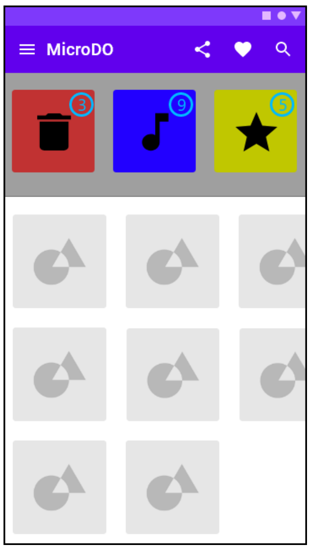
I evaluate that my goal was achieved when all moved videos are no longer part of the list and the number on the buckets show the correct number of included items.

Acceptance Criteria

- When long clicked on any selected video, the drag and drop starts and the items are removed from the list
- When the items are dragged they are no longer part of the original list
- When dropped on a bucket, its number of items increases

Mockup





5.3 Show videos of buckets

Story:

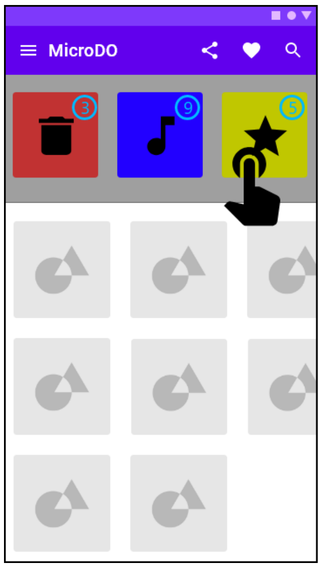
As a person who edits videos,
I want to check what items I have inside my buckets,
for this the system allows me to click on any of the buckets,
when clicked on one of them, the items of this bucket are shown in the list below.

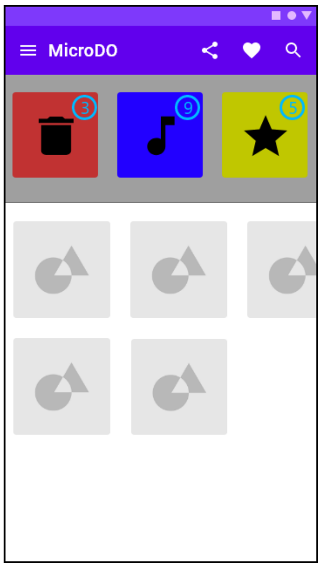
I evaluate that my goal was achieved, when the correct videos from the selected bucket are shown in the list.

Acceptance Criteria

- When a clicked on a bucket, the items added to this bucket are shown

Mockup


MicroDO
3
9
5


MicroDO
3
9
5