FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Technology Adequate Commenting

On the Importance of Details

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Masterstudium Software Engineering/Internet Computing

eingereicht von

Alexander Bachinger, BSc
Matrikelnummer 0225468

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof Dipl.Ing. Dr.techn. Peter Purgathofer

Wien, 15. Oktober 2020

Alexander Bachinger	Peter Purgathofer

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

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Technology Adequate Commenting

On the Importance of Details

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering/Internet Computing

by

Alexander Bachinger, BSc
Registration Number 0225468

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof Dipl.Ing. Dr.techn. Peter Purgathofer

Vienna, 15th October, 2020

Alexander Bachinger	Peter Purgathofer

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

Erklärung zur Verfassung der Arbeit

Alexander Bachinger, BSc
1030 Wien

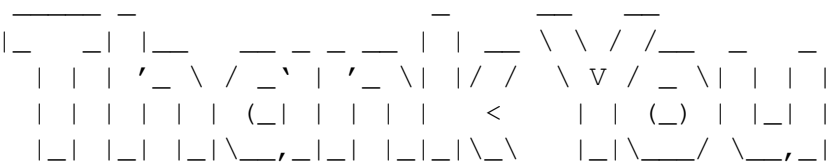
Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach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

Wien, 15. Oktober 2020

Alexander Bachinger

Acknowledgements

First of all, I would like to express my gratitude to my advisor Prof. Peter Purgathofer who patiently guided me through this process, for his genuine ideas and his helpful advice. I would also like to express my recognition to Prof. Geraldine Fitzpatrick who gave me insight into her scientific workflow and provided me with annotated documents that I could use as the foundation of my research. I am profoundly grateful to my parents, grandparents and aunt Susi for their unconditional support during my studies, this goes dedicated to all of you. Finally to my wife Debbie, whom I owe the greatest debt, as she encouraged me to start and keep writing this thesis, without her love and faith in me this work could not have been culminated.



Kurzfassung

Moderne Tablet Computer und ihre drahtlos angebundenen Eingabestifte, wie beispielsweise das iPad Pro und der Apple Pencil, bieten eine einzigartige Erfahrung für die Bedienung von Grafikprogrammen und zum Verfassen von Notizen. Obwohl sie eine ganzheitlichen Alternative zu Papier darstellen, bevorzugen viele Wissensarbeiterinnen und Wissensarbeiter ihre Dokumente in gedruckter Form zu annotieren. Ziel dieser Arbeit ist es einen neuen digitalen Workflow einzuführen, welcher in der Lage ist Wissensarbeiterinnen und Wissensarbeiter zu motivieren, ihre Dokumente nicht mehr wie bisher auf gedrucktem Papier zu annotieren, sondern das iPad als digitalen Ersatz zu verwenden. Um dies zu erreichen wurden annotierte Dokumente, die von Personen aus der Zielgruppe zur Verfügung gestellt worden sind, analysiert und die Anforderungen der wissenschaftlichen Vorgehensweise beim Annotieren von Dokumenten durch Anwenden der Observation Technique erhoben. Durch Anwenden des Research through Design Ansatzes, in Verbindung mit einer umfangreichen Literaturanalyse bestehender Arbeiten in den Fachbereichen User Interface Design und Wahrnehmungsforschung, wurden eine Auswahl an Applikationen für das iPad evaluiert und deren Vor- und Nachteile anhand etablierter Designmethoden gegenübergestellt. Basierend auf den Ergebnissen der Anforderungsanalyse und der evaluierung existierender Applikation wurde ein neuer Workflow erarbeitet und mittels *Business Process Model and Notation 2.0* beschrieben. Anschließend wurde ein Prototyp für das iPad und den Apple Pencil entworfen und implementiert, welcher als Grundlage für weiterführende Forschungsarbeiten dient. Die Herausforderungen während der Implementierungsphase und die dadurch eingegangenen Kompromisse wurden detailliert erläutert und weitere Ideen für zukünftige Versionen der Anwendung präsentiert.

Abstract

Modern tablet computers and wireless stylus pen accessories, such as the iPad Pro and the Apple Pencil, create a holistic experience by capturing input for drawing and note-taking. They offer useful alternatives to paper; nevertheless, it is still extremely common for knowledge workers to annotate upon printed documents instead. This thesis aims to introduce a new digital workflow, capable of motivating knowledge workers to annotate documents on an iPad as a substitute for paper. To achieve this, annotated documents provided by volunteers from the target group were analyzed and the requirements of the scientific markup workflow were elicited by applying the observation technique. By applying the research through design approach and conducting a literature review of the existing work in user interface design and cognitive psychology, a selection of iPad applications was evaluated and their advantages and disadvantages compared against good design practices. Based upon the results of the requirements elicitation and the application review, the workflow was created and described using *Business Process Model and Notation 2.0*. Subsequently, the prototype application for the iPad Pro and the Apple Pencil, which lays the foundation for further research, was designed and implemented. The challenges that arose during the implementation phase and the compromises that had to be made are explained in detail, and ideas for future versions of the application are presented.

Contents

CHAPTER 1

Introduction

Countless knowledge workers currently interact with printed versions of scientific documents rather than using digital alternatives. The scientific process that this encompasses includes many aspects commonly known as active reading (AR) and involves annotating, highlighting, and note-taking, among others [MBM07].

While paper is an established medium, it is also associated with certain disadvantages. Printed documents can entail a considerable amount of space and weight, especially when a large quantity of papers need to be transported between multiple locations. After they have served their purpose they then need to be archived or discarded, which produces unnecessary waste [Car].

Additionally, working with printed content has limitations with regard to AR requirements. Printed content is structured to increase readability, and the resulting space between paragraphs, borders and indents can be used to insert comments and annotations. However, due to the physically fixed nature of the content, it is impossible to insert more blank space except by adding supplements, such as adherable notes. Moreover, editing previously written annotations is very inconvenient due to the fact that some pens use permanent ink that cannot be erased. Nevertheless, over the years, people have become accustomed to these limitations because of the benefits that paper offers.

However, since the appearance of computers and tablets, paper is increasingly being replaced by digital alternatives [Car]. Screen-based media enable us to circumvent the physical limitations of printed documents. In this thesis, we focus specifically upon the *iPad Pro* and the *Apple Pencil*, and sometimes we refer to them as *iPad* and *pencil* for the sake of brevity.

With the introduction of the first iPhone in 2007, Steve Jobs, the former CEO of Apple, introduced a new era of handheld devices [Wikb]. One of their most outstanding features was a large multi-touch display, which rendered the physical keyboard obsolete [Pas11]. The additional space and the flexibility of placing buttons and other user interface (UI)

elements in any possible location on the screen enabled software developers to create natural mappings as described in "The Design of Everyday Things" [Nor02]. Serving as a continual reminder to users of how to operate the iPhone and its software [Pas11], the display allowed the use of an enormous variety of different applications (apps) and at the same time kept them easy to use and self-explanatory. Human interface guidelines [Appk] ensured that all the apps had a consistent appearance and quality across the platform. Apple's concept of an app market, the App Store [Appc], made it easier for developers to distribute apps than ever before. At the same time, buying and installing apps became straightforward for users. Consequently, over the years, smartphones have become personal assistants. The available apps range from web browsers, email, messaging, digital calendars, notes and reminders to navigation, music, social media platforms and games to name just a few. The *Smartphone Reflex* states that whatever your need, "there is an app for that" [DLD+13]. Everyday tasks have been replaced by using equivalent mobile apps. In addition, in 2010, Apple announced the first iPad [Wika], a tablet computer with a 9.7 inch display. The website *Lifewire* noted that Apple has sold over 360 million iPads since the release of the original iPad [Nat19].

(a) The first iPhone (2007)[1] (b) The first iPad (2010)[2]

Figure 1.1: The first iPhone and iPad

In 2015, the Apple Pencil [Appd] was introduced alongside the iPad Pro and soon the App Store was filled with related apps, ranging from simple note-taking to professional painting.

Nonetheless, even with the availability of these tools, annotating documents as part of the scientific process is still mostly performed upon the hard copy rather than upon a tablet. However, with the latest generation of tablet computers and digital pencils, it is finally possible to develop apps that are capable of delivering a holistic experience in regards to replacing pen and paper. These software solutions need to perform equally well, or even better, to gain public approbation over the "analog alternative".

[1] https://en.wikipedia.org/wiki/IPhone_(1st_generation)
[2] https://en.wikipedia.org/wiki/IPad_(1st_generation)

Accordingly, the goal of this thesis is to explore the possibilities that the *iPad Pro* offers in combination with the *Apple Pencil*. With this in mind, our research scope is based upon the following questions:

- How do knowledge workers interact with their documents?
- How do knowledge workers organize their documents?
- What are good design practices for user interfaces?

Furthermore we answer the following design related questions:

- How do existing applications work and what features do they offer?
- Can a tablet application improve interaction with and organization of scientific documents?
- What improvements would our app offer in comparison to others?
- What are the current technical possibilities of the Apple Pencil?

Consequently, *Chapter 2* describes the requirements elicitation process for the common scientific annotation workflow. A qualitative in-depth analysis of different scientific papers is performed and any remaining ambiguities are explained during an interview with the person who annotated the scientific papers.

In *Chapter 3*, background information about user experiences and established design practices has been gathered. Literature such as Dieter Rams' *Ten Principles for Good Design* [DJKM17], Donald A. Norman's *Design of Everyday Things* [Nor02], and other scientific papers about cognitive psychology [Pas11] are reviewed to create the foundation of our prototype's UI design.

Chapter 4 evaluates the actual technical possibilities of the Apple Pencil. Considering that there are many apps on the market that support the AR process, we review a hand-picked subset of them in detail and identify their unique features. We conclude this chapter by introducing our new workflow and describing it with a subset of the Business Process Model and Notation (BPMN) [Luca].

In *Chapter 5*, we explain the significant concepts behind our prototype and our approach to the AR process. We examine the advantages and disadvantages of paper compared to tablet computers with pencil support. Based upon our findings during the research phase, a UI mock-up is then described and illustrated.

Chapter 6 describes the implementation of our prototype app in detail. Expert programming, as part of the agile software development methodology, is applied during the design, prototyping and implementation phase of the project. We present the chosen software architecture, explain critical design decisions concerning the graphics application

programming interface (API) and how we implemented tools such as the pencil and text highlighter. Furthermore, we detail the difficulties that occurred during the software development phase.

Chapter 7 concludes the thesis by reporting the results of a small field test we carried out to gather feedback about the advantages and disadvantages of our prototype. Additionally, we present our own findings and ideas for future iterations.

CHAPTER 2

Understanding the Scientific Markup Workflow

2.1 Design Inquiry

To design a new form of a digital scientific markup workflow, our first objective was to understand how scientific documents are currently being annotated. As our target group, we focused mainly upon academics and scientists, but as mentioned in Chapter 1, we could easily have extended this focus to other professions. We performed a qualitative in-depth analysis of three different scientific papers given to us by a professional. After a thorough inquiry, we clarified the remaining ambiguities during a personal interview with that person.

During the evaluation we focused upon the following details and questions:

- How are scientific papers being annotated?
- Why are scientific papers begin annotated?
- Where are the notes written (notebook, paper, adhesive notes, or somewhere else)?
- What types of writing implements are used (pencil, pen, highlighter, or others)?
- What colors are used?
- What types of annotations are made and what types of drawings can be observed?
- What tools are used (laptop, tablet, folders, rulers, or something else)?
- Would it be beneficial if existing annotations could be moved?

- How are the notes managed (folder, covers, laptop, or other)?
- How are references accomplished?
- How are references sorted?
- Is there anything else that might be of interest?

To document our findings, we took photos with a digital camera to indicate the different types of annotations being made. The text passages have been blurred for privacy reasons in the resulting photos, since the sample papers had not been published at the time of our research. Chapter 2.2 describes these findings in detail.

2.2 Observations

We encountered several different reasons as to why academics annotate their papers but the main reason, according to our interview, is because they are proofreading somebody else's paper. Other reasons we observed were either for targeted research, for marking "especially interesting passages" or sometimes "just because it is interesting," as one person stated.

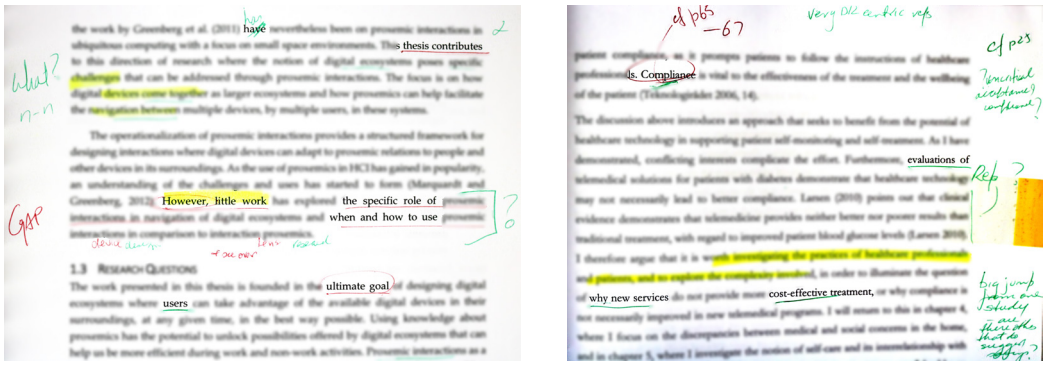
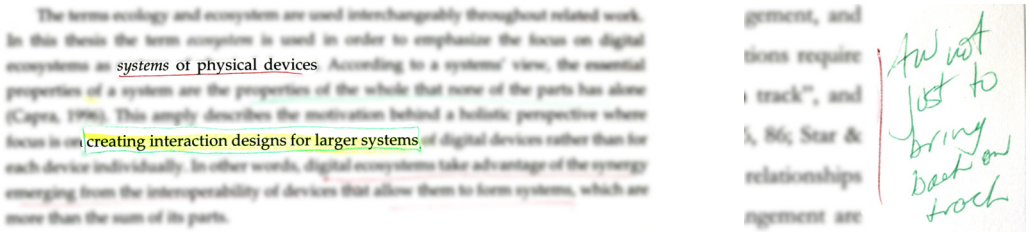


Figure 2.1: General Observations (blurred for privacy reasons)



(a) Highlights and Underlining (b) Notes askew

Figure 2.2: Observations - Highlights and Notes (blurred for privacy reasons)

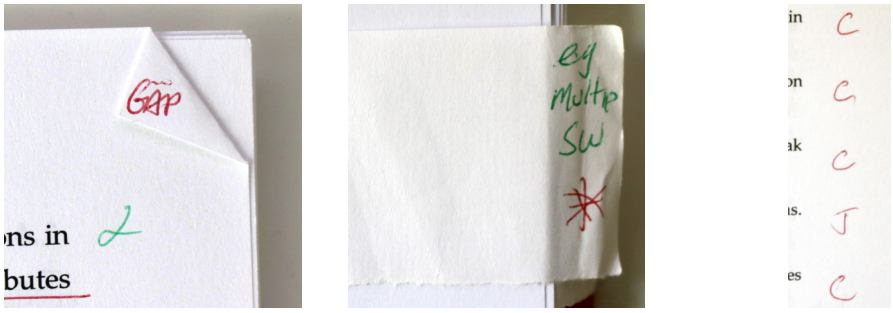
As Figure 2.1 shows, the notes were all written directly into empty spaces on the document itself. In rare cases, some of the annotations were written upon adhesive notes attached to the side of the document as can be observed in Figure 2.3b. Due to the limited white space at the margins of the document, multiple notes were crammed in and askew, especially in Figure 2.2b, which clearly indicates the limits of note-taking on printed documents. During the comparison of existing apps presented in Chapter 4, we discovered that many of the established apps, even the Acrobat Reader itself, limit the user's annotations to the size of the digital document. During the interview it was explicitly mentioned that annotating documents with the Adobe Reader "does not feel good."

Our observations demonstrated that two kinds of essential writing paraphernalia are in use:

First, a regular pen for writing notes in different colors; we encountered three main colors: blue, green and red. While the colors blue, red and green were being used to write text, red and green were mainly used for underlining text passages.

Second, in addition to the written notes, parts of the document were marked using a yellow highlighter. In this regard, we noticed that no other color than yellow was used for highlighting, while other text passages were underlined or circled using a green or red pen (Figure 2.1 and Figure 2.2a).

During the interview, we learned that the colors were chosen without reference to any system. It was stated that more colors would be used if they were available, but it was "easier to carry fewer pens along." It was mentioned that, if available, yellow was used to indicate repetitions and green to point out typographical errors. The same person usually used a multicolored pen and switched between green and red, but hardly ever used blue or black, to improve the readability and distinguishability of the annotations concerning the document's content. During the interview, we discovered another aspect with regard to choosing colors. Papers that were about to be given back to students were annotated mainly in green but red was hardly ever used. The reason for this was to avoid the negative connotations of the color red.



(a) Dog Ears as Markers (b) Adhesive Notes with Labels (c) Journal or Conference

Figure 2.3: Observations - Markers and Adhesive Notes

An interesting observation was that the pages were sometimes marked using dog ears as shown in Figure 2.3a. These were used to enable the reader to more rapidly find the annotated pages. The word "GAP" was used to indicate a scientific gap, and what the thesis contributed to filling this gap. In addition, the bibliography section was annotated using a "C" for conference paper or "J" for journal to obtain an overview of the balance of the references (Figure 2.3c).

We noticed that adhesive notes with labels were used to indicate different chapters but rarely had notes or text written on them as can be seen in Figures 2.3b and 2.4. These adhesive notes were used to indicate different chapters or interesting passages of the document. The colors were chosen randomly according to our participant. The exception was one note describing the purpose of a pile of books as part of document management.

Figure 2.4: Adhesive Notes as Page Indicators

No other tools were used: no rulers, folders, or anything else. The documents were all bound, hence, no single sheets of paper were loose.

2.2.1 A word about organizing documents in real life

During the interview, we also focused upon how documents were being managed and multiple documents were being maintained. The conclusion was that there was little to no system behind the management. Documents that were no longer being used were thrown into the paper trash or shredded if they contained sensitive content. Documents that were going to be handed back were placed in a pile on the desk, and papers were placed into a plastic sleeve to keep them together. Eventually, a document pile is cleaned up, thrown away or returned to the author as feedback. Documents were, however, not being sorted, labeled, or indexed in any other way.

2.3 Conclusions

The most important observations arising from Chapter 2.2 can be summarized as follows:

- The main reason for markup is proofreading.
- Pens and highlighters are the most important writing paraphernalia; a pencil was only used once to write text.
- Only a few basic colors, such as black, blue, red, and green are necessary. However, more colors would be used if they were available.

- Only a yellow highlighter was used, although multiple colors might be preferred if available.
- Annotations are written into the free spaces of the documents. They are sometimes written askew if there is too little space left.
- Adhesive notes are mainly used as page references but contain no comments. Sometimes, dog ears are used as page references.
- Adhesive notes are used to describe the purpose of a pile of documents.
- Documents are piled upon the desk and in shelves.
- Sometimes annotated documents are returned to the author.
- Documents that are no longer used are thrown away or shredded after a period of time.

Based on these conclusions, we developed ideas and designed the digital workflow that is described in Chapter 4. First, however, we take a brief detour into user experience and good design practices in the following chapter.

CHAPTER 3

On the Importance of Details in User Experience

In this chapter, our focus is upon describing the key traits of good design and exploring the principles that can be used to identify the quality of a design. We examine how cognitive psychology [Pas11] plays an important role in user experience and how it affects product design. We introduce the "Ten Principles for Good Design" by Dieter Rams [DJKM17], a former designer of Braun consumer products and cross reference these principles with several examples of real-world products and mobile UIs. Although we do present examples from different product categories, we focus upon user experiences in UI design, especially of iOS apps. However, before delving into the importance of details in the user experience, the term “user experience” must be defined.

3.1 What is User Experience?

According to the International Organization for Standardization ISO 9241-210, Ergonomics of human-system interaction, user experience (UX) is defined as "a person’s perceptions and responses that result from the use or anticipated use of a product, system or service." This includes all the users’ emotions, beliefs, preferences, perceptions, physical and psychological responses, behaviors, and accomplishments that occur before, during and after use [ISO]. However, this definition is not the only one currently in existence. The website *ALL ABOUT UX* lists a collection of no less than 27(!) different definitions for the term "user experience," from different authors, companies, websites and other sources [RLP+], which include the following.

"All the aspects of how people use an interactive product:
the way it feels in their hands,
how well they understand how it works,
how they feel about it while they're using it,
how well it serves their purposes,
and how well it fits into the entire context in which they are using it."
-Alben (1996) [RLP+]

"The design of user interaction with a system, product or service
considering the usability, the enjoyment and the fit to the way users think."
-UXmatters [RLP+]

"An activity of encounter by a computer user
with the auditory and visual presentation of a collection of computer programs.
It is important to note that this includes only what the user perceives
and not all that is presented."
-Microsoft [RLP+]

A term that has such a large variety of definitions is not only hard to grasp, there is also no general recipe or formula when it comes to designing an app following its principles. Design and UX go hand-in-hand with cognitive psychology, which is a field of psychology that studies how people perceive, remember, think, speak, and solve problems [FR09]. To be able to create well-designed products, one must first understand how the user's mind works and the product is perceived, recognizing that UX cannot be measured. It is a feeling [Rin14] that may depend upon many aspects, including the user's previous experiences with other products. A product cannot define a user's experience; it is defined by the user's own behavior, attitude, and emotions [Rin14]. Therefore, a product must be designed *for* the person that is going to use it. People from different target groups might perceive a product in different ways and their UXs may vary. When designing an app for senior citizens, focusing upon high contrast and better readability with larger font sizes might be desirable, while an app for children may have a more playful UI. An interface that is being used by scientists might be more difficult to operate without special knowledge but also provide more necessary functionalities.

Many calculator apps, for instance, have different modes. There is a basic mode with the primary functions such as addition, subtraction, multiplication, division and perhaps buttons for percentage calculations that are sufficient for everyday use and financial calculations. There may also be a scientific mode with more complex mathematical operations, such as trigonometric operations. Additionally, a mode for programmers is available that supports converting numbers between binary, decimal and hexadecimal representations. All these modes are designed to best suit the needs of different target groups.

3.2 An Excursion into Cognitive Psychology

To anticipate how a person perceives a product can be difficult, if not almost impossible; nevertheless, many insights are available into how the human brain responds to different types of stimulus and how to optimize a UI to create a positive impact upon its user. Konstantinos Paschalidis presents three aspects of cognitive psychology that are important to UI design: learning, attention, and perception [Pas11].

3.2.1 Learning

"For the things we have to learn before we can do them, we learn by doing them."
—Aristotle

Every app has a learning curve. Some apps are simple and easy to understand, others are more complex and contain many features that may not be clear at first. Modern apps are mostly self-explanatory and provide the means to make using them as easy as possible. If they do not do so, most users will quickly uninstall them. The experience a user has with an app in the first few minutes, perhaps even seconds, defines its success or failure. A well-designed app needs to be centered around its users to provide a good UX to increase its chances of success. Donald A. Norman describes two fundamental principles of designing for people: First, provide a good *conceptual model* and second, *make things visible* [Nor02].

Provide a Conceptual Model

A conceptual model forms in the user's mind while using an app. If the app is designed well, it enables the user to mentally simulate an action and predict its outcome. In the long term, it allows the user to identify how a product can be operated, thereby, making a product self-explanatory. A conceptual model for an app needs to combine the knowledge that a user has already acquired from previous experiences with the knowledge provided by the app itself. This information does not need to be precise; it only needs to be sufficient. Norman explains this with the "One Cent Coin" experiment, conducted by Nickerson and Adams, in which students were presented with 15 pictures of different one cent coins [NAB79]. The outcome of this experiment was that less than half of the American students who participated in the experiment were able to select the actual correct one cent coin. However, with regard to using them, this does not matter since they only need to be distinguishable from other coins [Nor02].

Accordingly, in iOS, the placement of buttons with different styles and locations is a significant achievement that creates a dynamic mapping, which constantly reminds the user how to operate the app. Consequently, one does not need to learn by memorizing button combinations and reading manuals.

However, if an app fails to deliver enough information to perform a desired task, or even worse, if it provides the wrong information, the resulting conceptual model may be misleading and prevent the user from obtaining the desired outcome of an action. This

might be a frustrating experience and lead to attrition. On the other hand, if an app provides too much information at once, people may become overwhelmed, which has a negative impact upon the learning experience.

A conceptual model consists of *affordances*, *constraints* and *mappings* [Nor02].

The iPad and its UI provide these three aspects. With regard to the affordances, the iPad consists of a touch display, a prominently placed home button, a power button and two volume buttons that are less visible. If the touch display is enabled, it allows additional types of interactions, among these are tapping and swiping with one or more fingers. The availability of these actions depends upon the context that is visible on the display. There are also different ways to *constrain* actions; for instance, buttons can be disabled to indicate that the action they offer is currently not possible. Other UI elements may be completely hidden to provide a cleaner interface and to avoid overwhelming the user with a flood of possible actions. Each action also requires a *mapping* that allows the user to anticipate the outcome. The back button indicates that the user will return to the previous view, the app icon indicates the app that will start and other buttons have a text or a symbol suggesting their function. Mappings should be natural and not misleading. The back button, for example, is visualized using an arrow that points to the left, indicating a backward direction. If the arrow pointed to the right, this would be confusing and possibly even prevent people from tapping it.

Make Actions Visible

The second key to designing for people is *visibility*. Possible actions need to be visible or the user may not know that they exist. If one imagines all the hotkeys on the macOS, most are familiar with *CMD + C* and *CMD + V*, but how many people know that with *CMD + CTRL + SPACEBAR* an emoji keyboard appears? These hotkeys are certainly recorded somewhere, but most people simply do not know that the function even exists. Consequently, they do not look for it in the first place.

A good application makes its actions visible in form of buttons, icons, scrollbars, and other UI elements. Icons and buttons can be touched, and scrollbars indicate that sliding a finger over the display lets the user scroll over the content. There are many details that can indicate possible actions. Scrollbars for example cover a part of the screen which might be considered ugly. Therefore, a good implementation flashes the scrollbar just long enough for the user to be perceived and then hides them away. The same is valid for icons that might not appear as buttons in the first place. Sometimes they pulsate a bit, to indicate that they are an interactive element and can be tapped. A common practice is to use little animations to get the user's attention. It is the details that matter and guide the user to the desired outcome.

However, new versions of iOS, especially iPadOS, work counter to the concept of *visibility*. Apple have introduced many hidden gestures that enable more practical features, starting with multitasking, undo/redo, dragging and dropping between apps and many others. These are gestures that need to be learned at the outset. Since they are not necessary for

the everyday use of the devices, they do not have a negative impact on the UX, rather quite the opposite. Once people discover them, they have a positive impact upon the already existing UX. The three most important gestures, *Go Home*, *Switch Between Recent Apps*, or *Quickly Access Controls* for using the iPhone, are explained during the initial setup. They are simple to learn and easy to remember.

Improve the Learning Experience

The way the human brain processes information needs to be considered when designing an app in order to improve the learning experience. Considering these aspects makes it easier for people to understand how an app works. For instance, memorizing icons or shapes is known to be easier than learning alphanumeric text when there are no more than 49 UI elements involved [DD09].

Transference

People have expectations based upon their previous experiences with other UIs. This leads them to expect that the new UI will behave in the same or at least a similar way [DD09]. On iOS, every app needs to meet certain design guidelines provided by Apple [Appk] or it will not be allowed to be published in the App Store. These requirements define an expected style and behavior so that users can transfer their experiences to the new app and familiarize themselves with new apps more quickly. The navigation bar is a good example of *transference*. It has a similar appearance in every app; it is placed upon the top of the screen, the back button is placed on the left side, a title in the center and one or more actions on the right. These guidelines enforce the idea that apps do not use elements counter to their purpose, which would confuse people. Imagine a text passage that is underlined and colored blue but does not contain a hyperlink. Most people would touch it and expect the browser to open and would be confused when this did not occur.

Mental Images

Similar to transference, people become acquainted with the appearance of certain UI elements. This mental image is then transferred across different apps along with its associated function. Years ago, when data was stored on floppy discs, their icons were used to symbolize the save button. Even though floppy discs are now completely outdated, some apps still make use of these icons, as the represented functionality of saving a document is still in people's minds. There are many other examples such as the refresh button of a web browser or the paperclip icon that symbolizes an attachment in email programs or messengers. Icons that are customary for certain actions should not be used in a misleading way so that users can rely upon the mental image they already have. The concept of mental images is not only limited to simple icons or buttons, it can also be applied to the whole layout of a UI [DD09]. This is especially important when creating new apps whereby their design takes advantage of the previous knowledge and experiences of a user.

Motor System Limitations

Humans are not good at carrying out more than one task at a time, which can easily overload their cognitive capacity. The human brain is simply not good at multitasking. For this reason, UIs should avoid bombarding the brain with simultaneous information. To this end, if the interface needs to provide written information, it should not be placed above a background image that also contains text. Figure 3.1 shows an example of bad readability that has been taken from the website of a hotel, in which the background pattern interferes with the overlying text. The color of the pattern is almost identical to the color of the text and as a result the contrast between them is very low. The same principle is valid for audio feedback. If an app reads text using a text-to-speech engine, for example, it should not play music in the background at the same time since the user cannot pay equal attention to both sounds [DD09]. Apple's iOS enables the programmer to automatically lower the volume of music while its text-to-speech engine is reading text. Following the completion of the text, the volume of the music is increased again to its regular level.

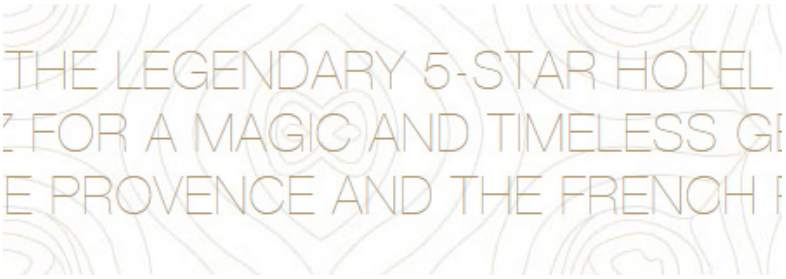


Figure 3.1: Example of Bad Readability

Memory Limitations

A good UI design practice is to limit the information that is simultaneously visible to lower the perpetual intake. In different experiments, the capacity of short-term memory appears to be 7 ± 2 items [DD09]. With that in mind, complex tasks should be broken down into smaller tasks. A long registration form with multiple input fields, for example, should be divided into multiple steps. The user should only initially be asked the questions that are necessary to complete the registration and answer other important questions at a later point. Additionally, a navigation menu, for example, should not offer too many entries at once; instead the navigation required should be reduced as much as possible. Removing complexity from a UI improves the learning experience and helps the user to perform the desired actions more rapidly. Figure 3.2 shows a program called *Bulk Rename Utility*. On the website, it states that it is "an easy to use file rename program (a.k.a. file renamer)" [Sof]. It definitely has all the necessary functionalities, and once the user has learned how to use it, it can be a very helpful tool. However, a user may be discouraged by its somewhat congested appearance.

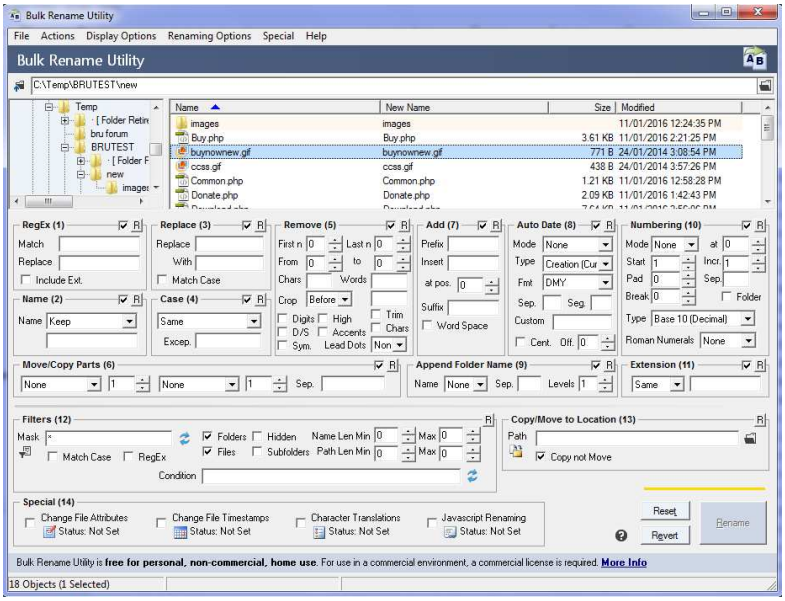


Figure 3.2: Example of an Overloaded UI[3]

3.2.2 Attention

"Attention is the ability we have to discriminate and to focus only on that which we want to perceive."
-Miguel Angel Ruiz

A study conducted by Microsoft in 2015 showed that in 2013, the average human attention span was approximately eight seconds. This is four seconds less than it was in 2000, when the average attention span was still 12 seconds [CI15]. These results are particularly important for advertising and marketing, but they also indicate that it is becoming increasingly difficult to obtain a person's attention. Instagram stories are designed with this attention span in mind. Every story is limited in length, which is seven seconds for images and 15 seconds at most for videos. Delivering information via Instagram stories needs to be executed in the simplest way possible: an image, short text and nothing more. Despite these limitations, it is still possible to add multiple stories in a row. Nonetheless, capturing the user's attention can be achieved in different ways by exploiting the peculiarities of the human brain. A well-designed app takes this into consideration and presents its elements so that important UI elements are larger and brighter than less important ones. Another way of focusing the user's attention is to use animations such as shaking or blinking UI elements. In this way, the human brain can easily archive information depending upon its significance [Pas11]. Apple's iOS animates incoming notifications by having them appear on the top border of the screen for a few seconds before hiding them again. This not only captures the user's attention, it also

[3] https://www.bulkrenameutility.co.uk/

provides a hint upon where to find them – by sliding one's finger down from the top edge of the screen.

Left to Right Theory

In the Western world, information is approached and interpreted from left to right and top to bottom [ÅC05]. This should be considered when designing an app layout. Important content should be positioned in the top-left area and less important elements moved to the bottom-right corner. However, it is mandatory to consider the display size of mobile devices. The top-left area is the most difficult area to reach when interacting with the device using only one's right hand, a practice that is extremely common. Figure 3.3 shows the different zones, starting with the iPhone 4S, which had a 3.5-inch display, up to the iPhone 6 Plus, which shipped with a 5.5-inch display [App18] . These zones have become even more dramatic in the newer phones, as the iPhone 11 Pro Max already has a 6.5-inch display built in. Needless to say, those who operate the devices with their left hands need to be considered as well.

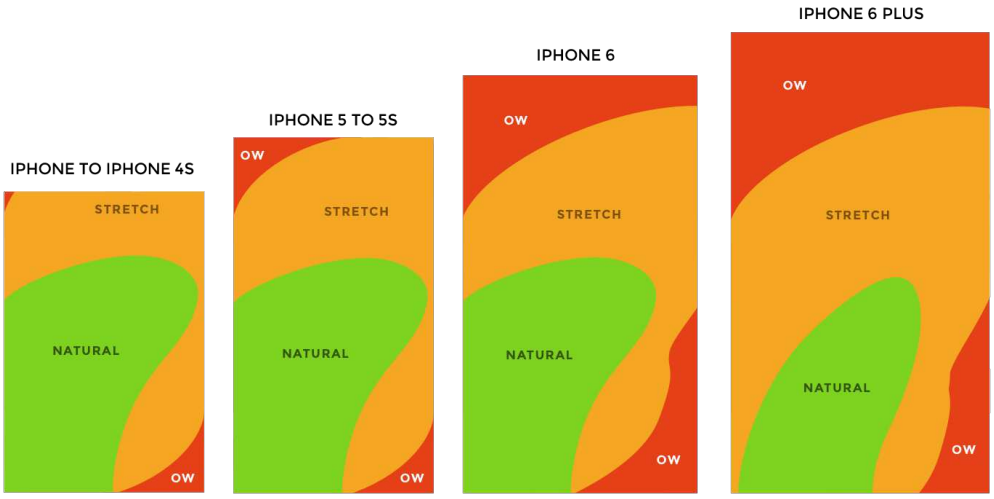


Figure 3.3: Thumb Zones[4]

For this reason, it is good practice to place the interactive elements in the most reachable places. This is especially true for the menu button that is commonly placed in the top-left corner.

Figure 3.4: Evolution of the Hamburger Menu[4]

In this regard, modern layouts tend to replace the so-called Hamburger Menu button (Figure 3.4) with a tab bar on the bottom [Abr14]. Tab bars are not only easier to reach when placed on the bottom, they also do not allow too many tabs, which reduces the navigation complexity. In addition, they follow the visibility principle mentioned in Section 3.2.1 compared to a menu button that hides the actual content unless it is tapped. For tablets, the thumb zones are a less critical factor in UI design since most tablets are never operated single-handedly. They either lie on a table, are placed on a stand, or are held with one hand while being operated with the other hand.

Blue Peripheral Vision

Acquiring and directing a person's attention towards certain areas of a UI is an important aspect of good design. The nature of the human eye suggests that the color blue is best suited to capturing a person's attention. There are two types of photoreceptors in the retina: rods and cones. While rods are more numerous and sensitive, there are three different type of cones that are able to detect the colors red, green and blue. Blue cones are mostly found on the outer edges of the retina and though they constitute only 2% of all the cones, they show the highest sensitivity [Nav]. It would exceed the scope of this thesis to provide more detail regarding the human eye and the processing of colors and images inside the visual cortex of the human brain, but the blue peripheral vision theory suggests the existence of a selective "blue amplifier" in the visual processing of the brain. This is supported by the fact that the higher sensitivity of the blue cones, compared to others, would not compensate for their significantly lower numbers [Nav]. For this reason, blue is the best choice whenever it is necessary to draw the user's attention towards a certain area of the UI. In addition, other colors are known to have particular psychological effects as well. While red signals a warning, green appears to be calming and safe. These colors appear in our everyday lives, in traffic lights, for example, or stop signs [Pas11]. Apple's UIAlertController, which can be used for popup dialogs, enforces these color schemes by using blue as the default action button. For destructive actions, the buttons appear in red as red is perceived as a warning signal. Other buttons simply remain in gray.

Another example are the default color schemes used for the Philips Hue smart lights,[6] in which blue is assigned to a scheme called *Concentrate* (Figure 3.5).

Simon Effect

The so-called Simon effect demonstrates that people tend to respond faster and more accurately to a stimulus when it occurs close to the area of the response. This is because there is an innate tendency in humans to respond in the direction of the source of the stimulus [DD09]. This behavior is also related to proximity and similarity as can be seen

[4] https://uxplanet.org/the-ultimate-guide-to-the-hamburger-menu-and-its-alternatives-e2da8dc7f1db

[5] https://play.google.com/store/apps/details?id=com.philips.lighting.hue2

[6] https://www.philips-hue.com/

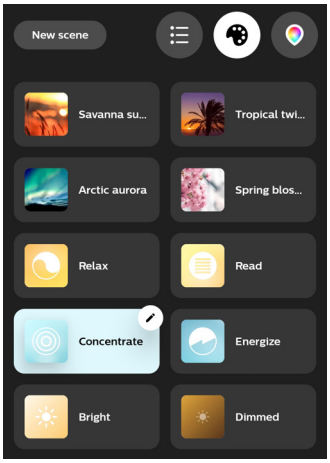


Figure 3.5: Philips Hue color schemes in the Hue Android App[5]

in the following section with regard to perception (3.2.3). Accordingly, in an app, UI elements that logically belong together should be placed closely together. They should be designed to provide congruent conditions in the context of the Simon effect [DD09]. They should also be placed in such a way that they represent a natural mapping [Nor02]. As an example, a button that indicates left should be placed on the left side of a button that indicates right. Imagine if the arrow keys on a physical keyboard were mixed up. Input errors would be an inevitable result. The Simon effect can be used to guide the user to the desired outcome. When a button is touched, its color changes to a darker variant, hence, the user finds the information they have requested on the same spot that the acquirement takes place [Pas11]. On iOS devices, deleting apps can be performed in the following way. On the home screen, one tap and holding a finger down on an app icon triggers a mode in which apps can be deleted. This mode is visualized by showing a little "X" icon on the top-left corner of every app icon. In addition, every icon shakes as if it is nervous or scared. The little "X" button is the stimulus placed close to the action as previously mentioned.

Automatic Processing

Some processes are executed consciously and intentionally, others are not under conscious control. Paschalidis shows an example of an image containing the text "Do not read this message." It is clearly impossible to follow the instructions of the image [Pas11]. In UI design, notifications are a good example of automatic processing. A notification appears and grabs the user's attention. Most of the time the short message that a notification contains is automatically read before it disappears. Even when the notification disappears from the screen, it is still listed in the mobile device's notification center. The user encounters the remaining notifications again before they can be hidden permanently.

3.2.3 Perception

"The more I see, the less I know for sure."
—John Lennon

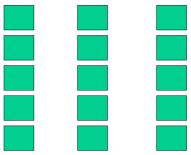
"Perception (from the Latin *perceptio*) is the organization, identification, and interpretation of sensory information in order to represent and understand the presented information, or the environment" [SGW11]. A well-designed UI helps with this process by providing information in such a way that a person can perceive it in a timely and correct manner [DD09]. Visual feedback should be clearly distinguishable from the background. Images and icons need to be understandable and follow the rules of transference (Chapter 3.2.1) and mental images (Chapter 3.2.1). The same principles are valid for audible feedback, where background music should not interfere with spoken text.

People that use assistive technologies often have different requirements, such as senior citizens for example, who may use larger fonts and need higher contrast to improve their experience when reading content. The Web Content Accessibility Guidelines (WCAG) [Ini] provide a set of rules that should be considered when designing a UI but providing more detail on this issue would exceed the scope of this thesis.

Proximity

The proximity rule in Figure 3.6 indicates that UI elements that are placed closely together are perceived as being related or associated and that our brain processes the information as a whole [Wit03].

(a) Proximity suggests two statements, not one



(b) Proximity suggests three columns, not five rows

Figure 3.6: Rule of Proximity [Wit03]

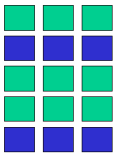
Similarity

The similarity rule in Figure 3.7 indicates that UI elements that have a similar appearance are perceived as being related or associated.

3.3 What is Good Design?

"Good design is under subsequent development, like technology and culture," stated Dieter Rams, perhaps the most famous German industrial designer. He was chief design officer at Braun for 40 years and well-known for many products, such as the Braun SK 4

(a) Similarity suggests two statements, not one



(b) Second and fifth row perceived as distinct rows

Figure 3.7: Rule of Similarity [Wit03]

record player and the Braun ET66 calculator. The Phonosuper SK 4 was the first record player with a cover made of acrylic glass, a material that had just entered the market, hence, the nickname "Schneewitchensarg" (English: Snow White's Coffin)

(a) Braun Phonosuper SK 4 "Schneewitchensarg"[7]

(b) Braun ET66 Calculator by Dietrich Lubs for Braun[8]

Figure 3.8: Products designed by Dieter Rams

Dieter Rams' conception of design inspired people such as Steve Jobs and Jonathan Ive, former chief design officer at Apple [Isa11], and his principles can be observed in many aspects of Apple products. According to Rams, design is always under subsequent development, like technology and culture. In the course of his career, he began formulating his ideas concerning good design into rules, which were published in the book *Ten Principles For Good Design* [DJKM17]. These principles are as follows:

1. **Good design is innovative**
   Good design cannot be derivative; it must push our understanding of a product forward [Sot19]. The first iPhone revolutionized the way mobile devices were being operated. Its display, as mentioned in Chapter 1, is capable of handling multi-touch finger gestures as input. The large display rendered a physical keyboard unnecessary, which was replaced by a software keyboard that is displayed only when needed. The area that was previously occupied by the physical keyboard then

[7] https://de.m.wikipedia.org/wiki/Braun_KM_3

[8] https://commons.wikimedia.org/wiki/File:Dieter_Rams_i_Dieter_Lubs-_Calculator_Braun_ET66-1987.dhub.jpg

became available for a larger display, allowing more sophisticated UIs and larger buttons that were more comfortable for tapping with a finger. With every new iPhone release, the focus has been upon providing progressively more space for the screen, thereby, displacing other components such as the home button, fingerprint sensor and camera.

(a) The first iPhone (2007)[9] (b) The iPhone 11 (2019)[10]

Figure 3.9: Twelve years of product design iteration

Figure 3.9 shows 12 years of product design iteration, starting with the first iPhone culminating in the iPhone 11, released in 2019. In the iPhone 11, the home button has been completely removed and replaced by gestures. The screen almost fills the whole body, leaving a notch for the camera and speaker. Authentication is performed by face recognition, therefore, rendering a fingerprint sensor obsolete.

2. **Good design makes a product useful**
"A good design is not only functional but also serves a psychological purpose," claims Dieter Rams.

Figure 3.10: Tesla, Nest Thermostat, and a MacBook[11]

[9] https://en.wikipedia.org/wiki/IPhone_(1st_generation)
[10] https://en.wikipedia.org/wiki/IPhone_11
[11] https://www.empathy.co/blog/dieter-rams-10-principles-of-good-design-in-a-digital-world/

Products, such as those shown in Figure 3.10, are recognized for elements beyond their functions. A Tesla is one car among many others, but it is still perceived as the uppermost reference in cases of environmental conscience. A MacBook Pro does not necessarily offer more functionality and performance than cheaper alternatives. Nevertheless, people project the image of a MacBook, that is, quality and creativity, onto themselves. This image adds to their own user identity and self-perception. Properly designed UIs can achieve a similar effect. They need to be straightforward, self-explanatory and memorable [Sot19].

3. **Good design is aesthetic**
   Even though the perception of a product's aesthetic is very subjective and influenced by social and cultural factors [Bar], there are best practices involved in designing a notable UX. These factors include values such as colors, consistency, proportions, and typography. They define the visual language of a UI and are vital to delivering an enjoyable product [Sot19]. Dieter Rams suggests that one should "cut away the fat of an object until all that's left is its function" [Wal19].

4. **Good design makes a product understandable**
   To quote Martin Leblanc, "A user interface is like a joke. If you have to explain it, it's not that good." In this regard, many older products used to ship with large manuals, but nowadays few people are willing to spend the time to read extensive instructions. A good product needs to work straight out of the box and its user needs to be able to learn its operations by using them. Modern smart phones are comprised of a vast array of different functions. In addition to making a simple phone call, there is messaging, internet browsing, calendars, and millions of apps available for downloading [Sta]. The iPhone's self-explanatory UI plays an important role in its success story. Its physical buttons have been reduced to a minimum, leaving only the power button, volume buttons, a button to mute the phone and the home button, including the fingerprint sensor, as seen in Figure 3.11.

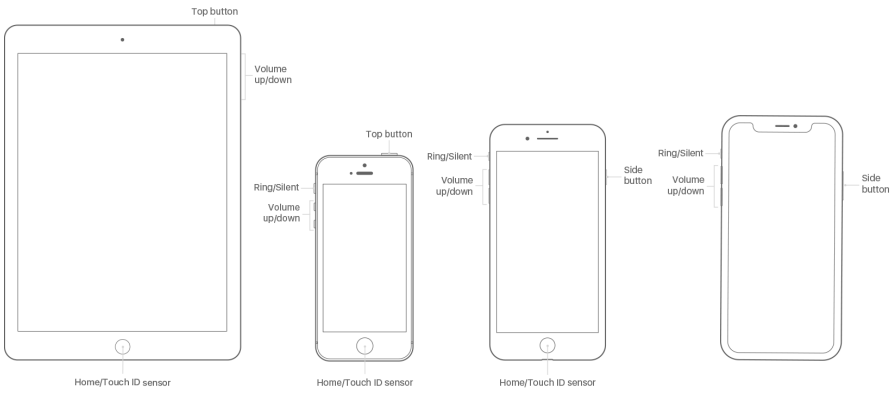


Figure 3.11: Buttons of different iPhone and iPad Models[12]

On newer iPhone models, starting with the iPhone X, the home button has been completely removed and its functionality has been replaced by gestures and a facial recognition system called *Face ID*. The iPhone's UI is fully self-explanatory, except for the newly added gestures, which contradict the concept of visibility [Nor02]. This issue is addressed in the iPhone's initial setup process, which explains and teaches these essential gestures.

5. **Good design is unobtrusive**
   The Leatherman Tool is an example of obtrusive design. It is over-designed and contains too many functions, therefore, people hardly use them [Wal19]. Although it is well designed and encompasses many tools, most of them are never needed, and many of them are not usable due to their small size. The Leatherman Super Tool 200 (Figure 3.12) was produced until 2005 and contains pliers, two different knifes, an awl, a Philips-tip screwdriver, three differently sized slot screwdrivers, a saw blade, a ruler in inches and centimeters, a can opener, a file, and a lanyard ring [Lea].

Figure 3.12: Leatherman Super Tool 200[13]

Figure 3.13: Leatherman Super Tool 200 - Ruler[13]

Other tools such as the ruler, shown in Figure 3.13, are uneven and have no sharp borders, therefore, measuring is imprecise. There are also issues with the file and the wooden saw. They are very small, therefore, using them is quite uncomfortable. Still, the Leatherman Tool is quite popular and receives good reviews. The reason

[12] https://support.apple.com/en-us/HT203017
[13] https://www.leatherman.com/super-tool-200-86.html

for this is that the tools were not actually designed for use in a workshop that has a workbench and enough space to store professional tools. The Leatherman Tool was designed to be portable, unlike a toolbox which is heavy and unwieldy. In UI design, unobtrusiveness can be achieved, by including only the necessary functionalities. An app, which is meant to solve a certain problem, should not distract the user by offering unnecessary features.

6. **Good design is honest**
   Good design should not promise more that the product can deliver. It should not suggest features that are not available or deceive the person that buys and uses the product in any way. Dishonest design can often be found on product packaging in the food industry. A good example is the images of the food that is presented in advertisements compared to the real products. This is most obvious in the burgers depicted in advertisements for well-known fast food restaurants compared to how they actually look when purchased. Some people call this marketing. Another example is the approach used to conceal increasing production costs. It is a common strategy of manufacturers to not completely fill the packaging, and people become used to this. However, it can easily lead to a scandal if, rather than increasing the product's price in the store, the content of the packaging is reduced while the package size remains the same, as was the case with Toblerone in Figure 3.14 [War16].

Figure 3.14: Toblerone hidden price increment[14]

7. **Good design is long lasting**
   If a design is really good, it rarely becomes outdated, like a timeless suit. If we look at 20-year-old pictures of people wearing suits and compare them with pictures of people wearing clothes that were considered fashionable at that time, the difference

[14] https://www.test.de/Mogelpackung-Toblerone-auch-in-Deutschland-mit-weniger-Zacken-5115737-0/

between fashionable design and long-lasting design is evident. The iPhone offers this kind of enduring design. The different iPhone models still seem very similar, with only minor changes in appearance. The display has been enlarged, the borders between the display and the frame have been reduced, and the phones overall have become thinner. Nevertheless, they all appear remarkably similar. It should also be remembered that for a mobile phone, optical design is not everything. Consider the battery life of a mobile phone. How many recharge cycles will it last? How complicated is it to replace the battery? Is it even possible to replace the battery? In addition to hardware maintainability, software support, especially security updates, is also an important criterion. The design aspect lies within the hardware. Is it capable of running newer software that is more computing intensive? How long does the manufacturer support the software?

For the media control unit (MCU) of their electric car, Tesla relied upon an embedded multi-media controller flash chip (eMMC), rather than removable storage [Nar19]. This type of memory has a limited amount of write cycles and once the limit is reached, the eMMC can wear out and crash the MCU, effectively rendering the car inoperable [Ruf19]. Usually, this kind of memory uses a technique called "wear leveling" to distribute write operations across its different memory addresses. This strategy is used in all modern memory chips and in solid-state drives. Unfortunately, this method only works well when there is enough free space available, which is not the case for the Tesla MCU because its firmware utilizes nearly 100% of the available space [Atw19]. Consequently, repairing the module can be quite expensive [Ruf19]. This is a problem that could have been easily avoided by using either larger memory modules or removable storage for easier replacement.

8. **Good design is thorough down to the last detail**
   Many products offer good design but then fail in terms of the small details. An example of this occurred with the charging of the first Apple Pencil. It can be charged either by using a tiny adapter, which can be easily lost, as seen in Figure 3.15b, or by plugging it into the iPad in a way that it could easily break the plug, as seen in Figure 3.15a. The newer version, the Apple Pencil 2, solves this problem on the newer iPad models. It can be attached to the tablet's side using magnets and charged wirelessly (Figure 3.16a).

9. **Good design is environmentally friendly**
   Unlike other products in our throwaway society, well-designed products are built in such a way that they can be used or reused for a long time. Compare soda water in plastic bottles, which are thrown away following consumption and likely to end up in the ocean, with the SodaStream [Gmb] or similar products that support refill cylinders. This principle aligns with Principle 7, which states that good design should be long lasting. In this regard, unfortunately, most mobile phones are designed to be replaced after only one or two years. The recharge lifecycle of

[15] https://support.apple.com/en-us/HT205236

(a) **BAD DESIGN**: Charging with iPad. The connector can be easily broken.

(b) **BAD**: Charging with Adapter. Adapter can be easily lost.[15]

Figure 3.15: Charging the Old Apple Pencil

the lithium-ion battery is limited, and most batteries are glued into the device and can only be replaced by a professional technician using the right equipment. Suppliers of hardware components stop delivering drivers for new operating system versions and the necessary security updates are no longer available [Cun]. This is unfortunate considering that mobile phones are manufactured using rare earth elements that are mined in a non-environmentally friendly way [Wikf].

10. **Good design is as little design as possible**
"Less, but better – because it concentrates on the essential aspects, and the products are not burdened with non-essentials. Back to purity, back to simplicity" [Ram]. Dieter Rams last principle of good design leads directly to the ten Laws of Simplicity composed by John Maeda [Mae06].

3.4 Ten Laws of Simplicity

"Simplicity is not the absence of clutter, that's a consequence of simplicity.
Simplicity is somehow essentially describing
the purpose and place of an object and product.
The absence of clutter is just a clutter-free product.
That's not simple."
-Jonathan Ive

In his book *The Laws of Simplicity*, John Maeda, MIT professor and graphic designer, explains the ten steps to achieving a simple product [Mae06].

(a) **GOOD DESIGN**: Magnets align and hold the pencil. It is charged wirelessly.[15]

Figure 3.16: Charging the new Apple Pencil

- **Law 1/Reduce**
  The simplest way to achieve simplicity is through thoughtful reduction. Simplification through reduction is often a complex task. It requires skill to identify the design aspects that can be reduced and those that need to stay in place to guide the user to its function [Hat]. This principle can be observed in Figure 3.17, which shows the *Siri Remote* for *AppleTV* and the *Amazon Fire TV Remote*.

(a) Siri Remote[16] (b) Fire TV Remote[17]

Figure 3.17: Modern Remote Controls

[16] https://www.apple.com/shop/product/MQGD2LL/A/siri-remote
[17] https://www.amazon.de/Alexa-Sprachfernbedienung-Fire-TV-Tasten-An-Aus-

These controls have been reduced to a minimum; all the unnecessary buttons have been removed. While the Siri Remote offers a touch surface for navigation, the Fire TV Remote relies upon a round button that allows navigation in all four directions. In addition to navigation, both remotes offer buttons for the most important functions such as volume, play and pause. In addition, each remote includes a microphone to send voice commands to the company's digital assistant. In UI design, visible information should be limited to no more than 7 ± 2 elements as stated in Chapter 3.2.1.

- **Law 2/Organize**
  Organization makes a system of many appear fewer.
  Organizing content creates a better oversight of what is relevant. Accordingly, 365 days of the year can be grouped into weeks, 52 weeks can be grouped into months and 12 months can be grouped into a year [Hat]. Another example of organization is a well-designed file manager. It makes a difference when thousands of files are organized into a folder structure with tags and labels and a good search functionality, rather than being listed inside one single view. A well-structured and reduced UI improves a user's perceptions of the UI. This kind of UI also helps the user to perform certain tasks quicker, which leads to the next point.

- **Law 3/Time**
  Savings in time feel like simplicity.

  "Time is a precious thing. Never waste it."
  -Gene Wilder

  A good UI helps to save time; it should only present the functionalities that are feasible, and each action should lead directly to the user's desired outcome. It does not make sense to show both login and logout buttons together, for instance. Depending upon whether the user is currently logged in, only one of these buttons should be visible. Nevertheless, certain actions, such as downloading information from the network or performing long running calculations, may take a considerable amount of time. Jacob Nielsen considers it good practice to offer visible feedback for tasks that take longer than one second to indicate that the app is busy. For tasks that take longer than 10 seconds, an indication of the remaining time, or the current progress, should be provided [Nie93]. A well-designed graphical progress display can provide the illusion that a process is more rapidly completed than it actually is [Mae06]. John Maeda describes a tradeoff between the quantitatively fast versus the qualitatively fast:

  How can you make the wait shorter? — *How can you make the wait more tolerable?*

  A reduction in the time users spend waiting can be an increase in the time that they can use for something more beneficial. However, if reducing time is not possible,

Lautstaerke/dp/B07PZR3P7W/

then a well designed app needs to at least take extra care of the user [Mae06]. The more rapidly a task can be completed, the simpler it seems to the user. For this reason, it is beneficial to omit unnecessary steps that pose a distraction. Currently, all websites are forced to ask visitors to provide their consent for the use of tracking cookies [Com]. This behavior is important to respecting the user's privacy, however, it is a distraction immediately before entering the website itself, especially on mobile devices where the screen size is limited. Despite this, many websites do not cease their interference with this requirement but continue to ask visitors for additional information. Some pages immediately ask the visitor to sign up for their newsletters, while others continuously ask for permission to send push notifications. These are some of the many distractions that jeopardize people's UXs. Reduced time is also a reason why websites presenting product and price comparisons have become increasingly popular. Rather than the user having to check each shop individually, using one website that aggregates information from several online shops and presents it in a compact way saves time and effort.

- **Law 4/Learn**
  Knowledge makes everything simpler.
  Many apps ship with complex functionalities that cannot be reduced, instead learning how these functionalities work must be made easy for the user. Consequently, the app must provide all the knowledge necessary to complete a task, especially knowledge that the user cannot be assumed to have or to have gained through previous experiences with similar products. A well-designed UI enables only actions that can be executed successfully. It needs to make all possible actions visible to the user and it is mandatory that these actions do not result in a fault state, especially not an irreversible fault state for the app itself. Well-designed UIs also need to offer a certain level of operational consistency between different actions so that the acquired knowledge can be applied to other actions and their outcome can be predicted by the user. John Maeda summarizes his approach as follows:

  BASICS are the beginning.
  REPEAT yourself often.
  AVOID creating desperation.
  INSPIRE with examples.
  NEVER forget to repeat yourself.

- **Law 5/Differences**
  Simplicity and complexity need each other.
  In Law 1, we cited the *Siri Remote* as an example of thoughtful reduction. The simplicity of this remote is even more conspicuous when it is compared with the other, more complex remotes available on the market. This insight can be transferred directly to UI design. If there are apps with complex UIs on the market, providing an alternative that has a simple, clean, and reduced interface that still offers the same functionalities renders it outstanding. People will recognize that its value exceeds those of others.

- **Law 6/Context**
  What lies in the periphery of simplicity is definitely not peripheral.
  Complicated features can capture all the designer's attention. Focusing upon those details is important for their successful realization but such focus is not always a good thing because too little attention may be given to the surrounding details. Maeda repeats some advice given to him by his teacher. A laser beam will highlight a spot, but a light bulb will illuminate everything around it [Mae06]. For UIs, it is important that they provide a holistic experience. Every detail should be given a similar amount of attention and focus, and no component should be neglected.

- **Law 7/Emotion**
  More emotions are better than less.
  Products need to trigger a positive emotional response, and sometimes reducing a product to its minimum may not be sufficient to achieve this. This contradicts the first law of simplicity to a certain degree by claiming that sometimes adding a reasonable level of complexity is necessary. Some people may even consider simplicity to be ugly; they desire colors and other decorative essentials. Smart phone cases provide a perfect example of this. Since mobile phones have become thinner and more fragile over the years, people have tended to use covers for protection but also for decoration. There is an enormous variety of smart phone cases available in different colors, designs, and styles. Clearly some people are missing emotional stimulus from their well-designed product and tend to fill this void with playful accessories such as the *Hello Kitty iPhone case* shown in Figure 3.18.

Figure 3.18: Hello Kitty iPhone case[18]

- **Law 8/Trust**
  In simplicity we trust.
  Software apps should help to make a person's life easier. Therefore, it is important to offer a certain level of comfort, and one aspect of this comfort lies in knowing that no mistakes can be made. Certainly, this level of convenience is not easily

[18] https://www.skinit.com

achieved in a complex app. However, knowing that mistakes can be corrected and decisions can be reversed can have the same calming effect [Mae06]. Users can risk experimenting with the app; they can explore the possibilities a program offers without being afraid of any negative consequences. The ability to "undo" recently performed actions has become essential in modern apps and people have learned to place their trust in this functionality. An alternative aspect of this law is that the more a system knows about its user, the more comfort it can offer. Google, for instance, combines the information it collects about a user to provide them with additional services. Incoming emails for hotel bookings or flights are scanned and the appropriate dates are automatically added to the user's calendar, including the hotel's address and any other information that can be extracted from the aforementioned email. Finally, we must trust that Google has extracted the correct information. Needless to say, however, this level of comfort comes at the cost of privacy. When designing a UI, existing knowledge about the user can be utilized to achieve a better experience at the risk of making mistakes.

- **Law 9/Failure**
  Some things can never be made simple.
  Simplifying a complex task may not always be possible because certain problems cannot be solved in an easy way. If a UI cannot be further reduced, then embracing its complexity is a valid option. John Maeda notes that there is always the so-called *return on failure*, which is learning from one's mistakes if simplification fails [Mae06].

- **Law 10/The one**
  Simplicity is about subtracting the obvious and adding the meaningful.
  This law stands for itself; however, Maeda adds three extra keys: *Away*, *Open* and *Power*.

  1. **Away**
     More appears like less by simply moving it far, far away.
     The simplicity of a product can be achieved by moving its complexity away from the person using it. Progressive web applications [Wike] provide a good example of this. By delivering apps through the web, there is no need to download them, install them locally or keep them up-to-date. They can be started by simply typing their URLs into the web browser and can create a UX similar to that of native applications.
  2. **Open**
     Openness simplifies complexity.
     Instead of spending time and resources solving complex problems, it can be beneficial to integrate solutions from third party members or integrate open source libraries, which are usually available without requiring any additional expense. Additionally, many companies, such as Google, provide APIs for a vast number of their services, such as its Translator or Google Maps, to cite just two examples. It is obviously more feasible to integrate an existing

service such as Google Maps, rather than recreating something similar from the beginning. Making a problem accessible to other people by opening it up can help to solve it in an efficient way. In this regard, crowdsourcing helped to solve a molecular puzzle that had stumped scientists for years within ten days [Boy11].

3. **Power**
   Use less, gain more.
   Decreasing an app's power consumption plays an important role in providing a good UX. One of the most critical resources of a mobile phone is its battery endurance. Mobile operating systems have become more critical and restrictive concerning power insensitive apps over the years. They limit active background tasks to the absolute minimum and inform users of software that drains the battery. Consequently, users are likely to change to alternative programs that have less impact on the battery life. A well-designed app can reduce its power consumption in different ways: First, by supporting the user to perform their tasks quickly and efficiently thus spending less time within the app, and second, by following best practices during the software implementation itself.

3.5 Picking a Color Theme

An attractive color theme is vital for an app's success. According to Apple's Human Interface Guidelines, color is "a great way to impart vitality, provide visual continuity, communicate status information, give feedback in response to user actions, and help people visualize data" [Appg]. Several aspects need to be considered when picking a color theme. The chosen colors need to indicate which elements are interactive and which are not. Picking the same color for both of these elements leads to confusion and should be avoided. Furthermore, the legibility of the UI elements, especially the text is extremely important and can be achieved by ensuring a high contrast ratio between the background and the content. The "Web Content Accessibility Guidelines 2.0" [Ini] level AA requires a contrast ratio of at least 4.5:1 for normal text and 3:1 for large text. The Apple guidelines recommend a contrast of 4.5:1 for fonts up to 17 points and 3:1 for larger fonts of 18 points and more [Apph]. The stricter level AAA requires an even higher contrast of at least 7:1 for normal text and 4.5:1 for large text [Ini]. To this end, there are many online tools, such as the *WebAIM: Contrast Checker*[19] or the website *contrast ratio*[20], that can help with this matter. In addition to the contrast ratio, it is also important to consider the needs of visually impaired people. Color-blind people, for example, have different requirements as shown in Figure 3.19.

We can also distinguish between the primary and the secondary colors. The primary color is the most visible color in the app's UI. It usually represents the brand and can be

[19] https://webaim.org/resources/contrastchecker/
[20] https://contrast-ratio.com/
[21] https://developer.apple.com/design/human-interface-guidelines/accessibility/overview/color-and-contrast/

(a) Perceived without color blindness (b) Perceived with red-green color blindness

Figure 3.19: Example of color blind perception[21]

used to accent certain UI elements. Dark and light variations can be used to distinguish between different types of UI elements. The secondary color is used as an alternative to the primary color. It can be applied to accent UI elements and is suitable for interactive elements such as buttons, selection controls, progress bars and sliders [Azh18]. Both colors need to differentiate themselves from the background and other surface colors. The background color is usually white or very light gray for a light theme. Other surface colors refer to cards or other components that are placed onto the background [Azh18]. New versions of mobile operating systems allow the user to choose between a light and a dark mode. As the name indicates, the dark mode entails black or a very dark gray as the background color. If the app supports both modes, then a suitable color theme needs to be provided for each one. Consequently, it might be necessary to provide different colors for each theme. There are many free tools available that can help designers with this task. One example is the website *coolors.co*.[22] It generates random color palettes, each consisting of five different colors that work together. If the palette does not completely fit the app's needs, single colors can be "locked," and the remaining colors can be randomized again until the desired palette is complete. The website emphcolormind.io [23] works in the same way.

3.5.1 Conclusion

"Perfection is achieved, not when there is nothing more to add,
but when there is nothing left to take away."
— Antoine de Saint-Exupéry

All the relevant principles and laws have been enumerated. Nonetheless, during the

[22] https://coolors.co
[23] http://colormind.io

design process, the order in which these items occur is meaningless. It is more important to understand them as a repository of principles that need to be considered concurrently. Rules that seem to be contradictory have to be weighed against each other to find the perfect balance. To achieve a good UI, two important guidelines are described in Chapter 3.2.1, *Learning*: First, provide a good conceptual model, and second, make the app's actions visible [Nor02]. Both help a user to understand how to operate an app by making its UI self-explanatory. Hidden functionalities need to be either explained at the right moment or to be anticipatable, as described in Chapter 3.2.1, *Transference*. Another equally essential practice is to simplify the app through thoughtful reduction, as described in the first law, *Reduce*, in *The Laws of Simplicity* in Chapter 3.4. Unnecessary UI elements should be decluttered and the load of simultaneously presented information should be minimized to reduce the cognitive load for the reasons described in *Memory Limitations* in Chapter 3.2.1. By following all these insights and best practices, the major objective is to design an app around the capabilities and limitations of the persons who will ultimately use it.

CHAPTER 4

Designing an App-Based Workflow

Prior to designing the workflow, we conducted research into the current technical possibilities of the iPad Pro and the Apple Pencil.

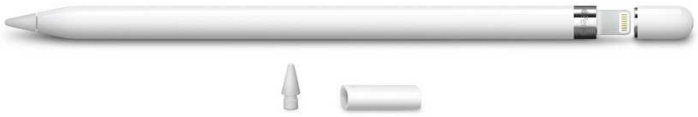
Despite our objective, which is to develop an app for the iPad, we must mention that there are several different tablets with pencils on the market that would have also been suitable for our task. While Google and Microsoft offer their own tablets with similar functionalities as the iPad, other interesting options for annotating documents are the E-Ink based tablets, such as the *reMarkable* tablet or the *Sony DPT-RP1/B Digital Paper Tablet*. In our opinion, however, the iPad offers the most holistic experience to date.

4.1 Technical Possibilities of the Apple Pencil

The first-generation Apple Pencil, as shown in Figure 4.1a, is approximately 175.7 mm long, with a diameter of 8.9 mm and a weight of 20.7 gm. It is compliant with all the newer iPad models, especially the iPad Pro, and it connects to the iPad using Bluetooth. The second-generation pencil, as seen in Figure 4.1b, is slightly shorter, with a length of 166 m and the same diameter of 8.9 mm and weight of 20.7 gm as its predecessor. It has a flat side to avoid it rolling off a surface and can be charged wirelessly by magnetically attaching it to the side of the iPad. Furthermore, it supports a double-tap gesture [Appd].

For better latency, the pencils report touches of up to 240Hz, compared to regular touch gestures that deliver data with only 60Hz. If the app cannot process these touches quickly enough, they are coalesced into a single touch event, representing all the locations since

[24] https://www.apple.com



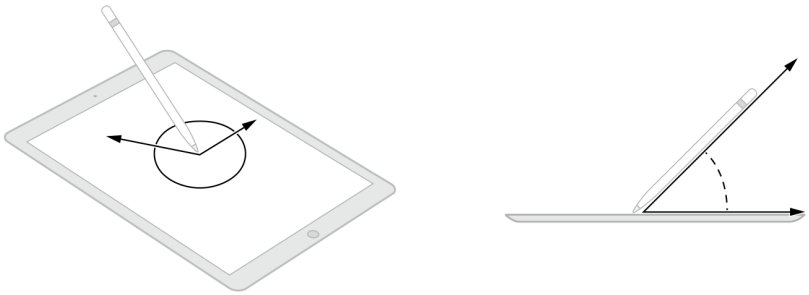
(a) First-generation

(b) Second-generation

Figure 4.1: Available Apple Pencils[24]

Figure 4.2: Apple Pencil compared to regular pencils

the previous touch event [Appj]. Pencil touches contain additional information, such as the azimuth, altitude and the force recorded at the tip, as shown in Figure 4.3.



(a) Azimuth (b) Altitude

Figure 4.3: Apple Pencil azimuth and altitude[25]

It is important to note that the additional data sent from the pencil to the iPad arrives with a short delay. To achieve faster response times, the iPad initially estimates the force, azimuth, and altitude values. When the pencil finally sends the real values, these estimated values can be replaced to allow better accuracy. It might also be the case

[25] https://developer.apple.com/documentation/uikit/pencil_interactions/handling_input_from_apple_pencil

that some of these values cannot be precisely determined, for instance, if the pencil's location is measured at the border of the display. For these special cases, the touch event object has a flag to indicate that updated values cannot be expected. With the release of the new iPadOS, Apple claims to have reduced the second-generation pencil's latency from 20 ms down to 9 ms [Appn], which is virtually no lag at all. However, during the implementation described in Chapter 6, we observed that attaining a good response time is challenging. This is especially the case when using the slower Core Graphics framework, rather than the much faster but significantly more complicated Metal API [Appm].

4.2 Inspirational Applications

There are many different apps available with similar functionalities to those we seek for our own solution. For this reason, we examined several of them before we began the workflow design. In addition, we experimented with different drawing apps to gain an impression of how the Apple Pencil is implemented.

- Active Reading
  - Adobe Acrobat Reader
  - LiquidText
  - MarginNote 3
  - PencilKit (by Apple)
  - PDF Viewer
  - PDF Expert
  - GoodNotes
  - Loose Leaf

- Drawing
  - Concepts
  - Paper
  - Pro Create
  - SketchBook
  - Sketches

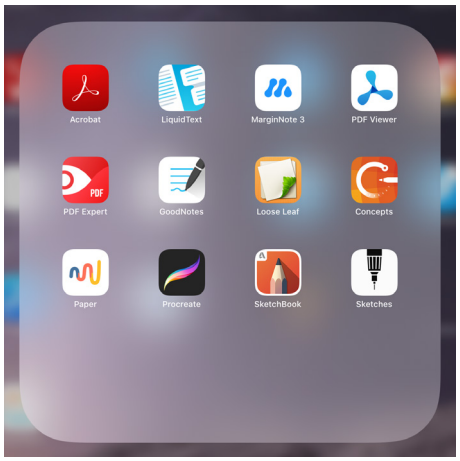


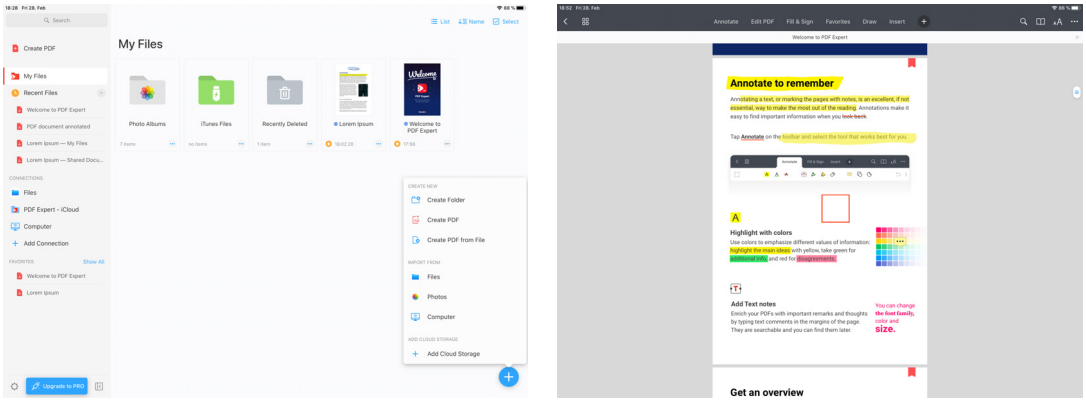
Figure 4.4: Inspirational Applications

Many of these apps are similar in terms of the types of tools they support and the basic interface design, but some cases offer very unique features. Unfortunately, it would exceed the scope of this thesis to present all the apps in detail, which is why we have only described a reasonable subset of them more closely.

4.2.1 PDF Expert

PDF Expert offers more functionality than the Adobe Acrobat Reader, especially the commercial version, which allows the user to edit the PDF document itself by enabling pages and content to be added. Additional PRO features include the ability to sign the

document, forms which can be filled out, and a customizable menu to add one's favorite tools. For this thesis, we focused upon the free version, which provides all the necessary functionalities for document annotation.



(a) File Manager - Grid View (b) Document View

Figure 4.5: PDF Expert - File Manager and Document View

Files can be managed in the program's file manager as shown in Figure 4.5a. Folders and files are presented either as a grid view or as a list. It is possible to create folders and subfolders to organize documents and sort their content by name, date, or size. The file manager sidebar offers the option to display the recently used files and enables the user to connect to different cloud storage providers such as Google Drive or Dropbox. It embeds the system's file picker to import files from the iCloud and the iPad itself. An interesting feature is the connection to a computer in the local network via web browser. The sidebar shows a list of documents starred as favorites. Files can be imported from various formats, such as Microsoft Word, Spreadsheets, Keynote, webpages, and others. Selected files are easily renamed, copied, moved, duplicated, and deleted. It is also possible to compress multiple selected files. Drag and drop is supported to move files between different folders and other sources, such as the favorites menu in the sidebar, for instance. Once files have been deleted, they can be found inside the "Recently Deleted" folder, from where they can be restored or permanently removed.

Once a document is opened, it is presented as illustrated in Figure 4.5b. A single tap on a free space inside the view hides all the unnecessary UI elements to avoid distraction. The pages can be aligned vertically, horizontally or with two pages side by side.

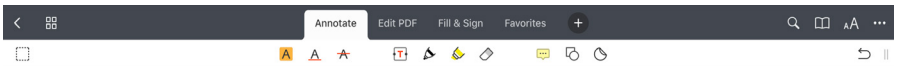
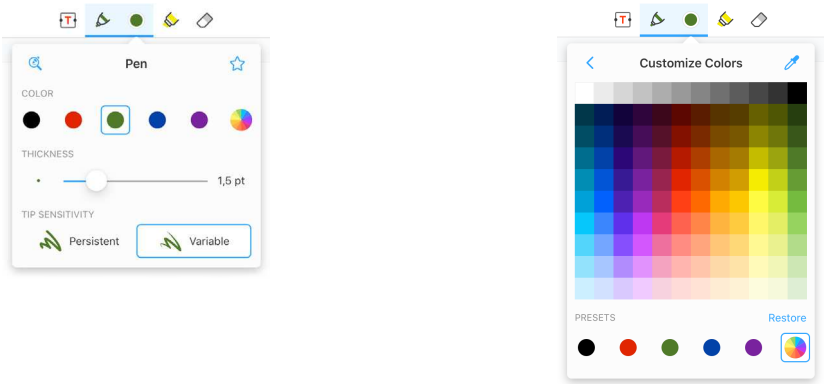


Figure 4.6: PDF Expert - Toolbar for Annotations

PDF Expert implements a wide variety of tools for highlighting, underlining, drawing, writing text, commenting and others as shown in Figure 4.6. Every tool provides

individual settings from which to choose. The pen, for instance, offers customization in terms of its thickness and whether the applied force should be considered when drawing. The app features two types of highlighters. One of them allows the user to select text, which is then highlighted, and the second tool can be used to freely highlight parts of the document. The opacity of this freestyle highlighter can be set to four different levels. Drawing shapes such as rectangles, circles, lines, and arrows is possible. Shapes support border thickness, border color, fill color and opacity. A nice detail is that shapes snap into place if they are close to an aspect ratio of 1:1. A playful feature is the stickers. They are downloaded from within the app. Once added to the document, it is possible to resize and flip them along the vertical axis.



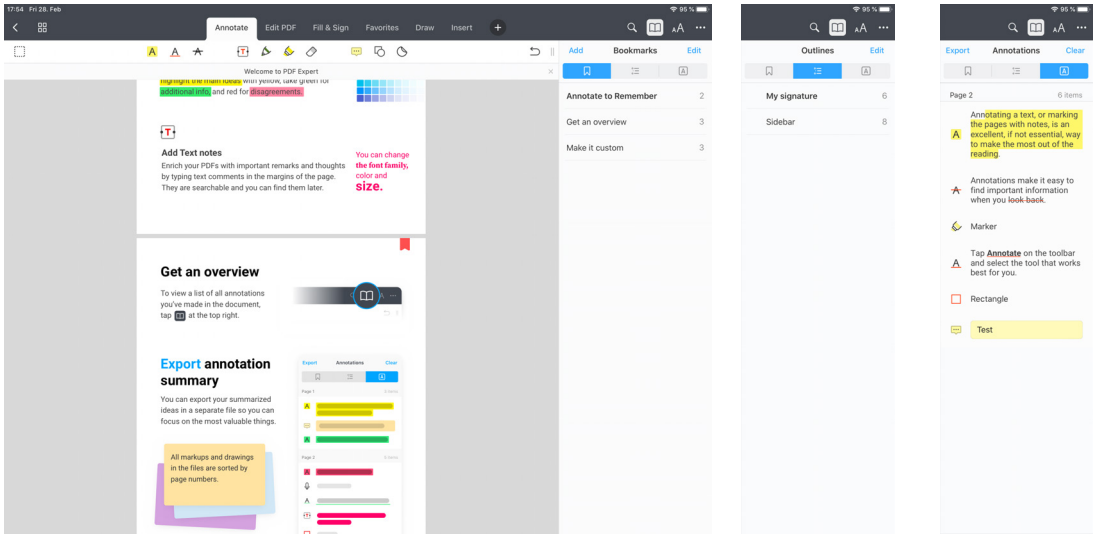
(a) Basic Color Picker (b) Expanded Color Picker

Figure 4.7: PDF Expert - Color Picker

The tool's colors can be selected independently from each other. Figure 4.7a visualizes the basic color picker, which offers five different presets. It is possible to extend it to a palette with 120 different color variations, as seen in Figure 4.7b. The selected colors overwrite one of the five preset colors to allow quicker access in the future. Additionally, it implements a pipette, which can be used to select colors directly from the document.

The PDF Expert's document view implements a sidebar to list bookmarks, outlines and annotations as indicated in Figure 4.8. If the document does not contain bookmarks, the *Add* button appends them manually. Another shortcut consists of tapping the top-right corner of the displayed page. This hidden function is explained by one of the helpful hints, as explained later in this chapter. Outlines visualize the structure of the document. They can be created by tapping the corresponding button in the outline tab and link to the assigned page. In addition to bookmarks and outlines, the annotations list displays the type of annotation in the form of a small icon and the assigned page. Annotations can either be searched or filtered by color.

Drawing is supported either with a single finger or by using the Apple Pencil. Two-finger drag-and-pinch gestures pan and zoom the document. If the tool is selected with the pencil, single-finger drawing is disabled and panning the document is allowed as well.



(a) Bookmarks (b) Outlines (c) Annotations

Figure 4.8: PDF Expert - Sidebar with Overviews

4.2.2 Interesting Details

The interface is reduced and simple, and it shows only the necessary elements. The color picker, for instance, is only visible after the corresponding tools have been selected. Unused toolbars can be hidden to improve clarity.

In general, almost all the functions are visibly represented by well-designed buttons as suggested in *Make Actions Visible* in Chapter 3.2.1. There are hidden functionalities that could be overlooked; however, PDF Expert provides hints for these features. The *Redo* button, for example, is hidden from sight and appears only after a long press of the *Undo* button. This behavior may not be known to the user and, therefore, a hint is provided after the *Undo* button is used a few times. A similar hint is shown when multiple tabs are being closed. A dialog indicates that tapping the tab opens a popup menu, which offers to close all other tabs. These types of hints are very unobtrusive and provide useful information to the user. The app observes the user's actions and if a certain intention is identified, better alternatives or shortcuts are suggested.

Another detail concerns the Apple Pencil support. As previously mentioned, if an annotation tool is selected with the pencil, the app disables drawing with a finger. Instead, a single touch can be used to pan the document in addition to the two-finger gesture.

While zooming the document, it is possible to snap the document into three different positions: to fit the toolbars, to fit the page, or to fit the width. Whenever the zoom gesture approaches one of these positions, the snapping behavior is indicated with a short, unobtrusive message. If the gesture is completed while one of these messages is

visible, the document snaps in accordingly.

A simple but no less important detail is the ability to lock the scrolling direction if the user intends to only pan the document vertically. For this purpose, the application observes the panning gesture and, if the gesture's movement vector has primarily a vertical component (y-axis), the horizontal component (x-axis) is ignored until the gesture is completed.

Possible Improvements

We located four points that could be improved in PDF Expert. First, the freestyle highlighter obscures the content it highlights, depending upon the selected opacity level. It is possible to cover the highlighted passages to 100%, which makes them unreadable. Only the highlighting tool for selected text avoids this by drawing *behind* the content. Unfortunately, it cannot be used to highlight images.

Second, it is not possible to draw annotations outside the document's boundaries, instead lines are simply clipped to the page. If the Two-Pages mode is active, two pages are aligned horizontally next to each other. Nevertheless, drawn annotations are still cropped to their original page.

Third, every document is opened inside a new tab. Over time, the open tabs accumulate, but there is no overview for these tabs such as the thumbnail overview for the pages of a single document.

A final improvement could be made by enabling the user to filter documents by tags. Colored tags can be added to documents and are displayed as small colored dots next to the filename. However, they are only visible in the grid view. We could not discern how to filter or sort by tag color.

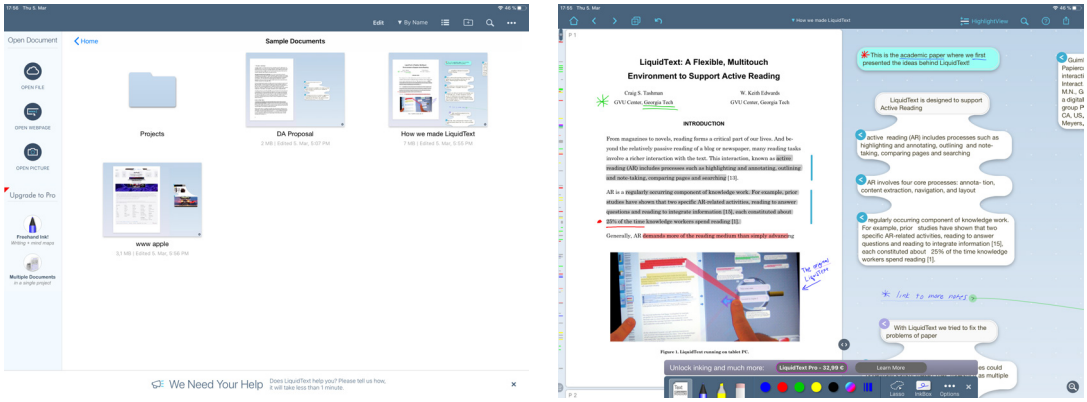
Conclusion

PDF Expert adheres very closely to the features defined by the PDF standard. Its strengths lie within the well-structured and clean UI, and in details, such as the unobtrusive indication of a hidden functionality, the comfortable color picker, and the snap-in while panning and zooming.

4.2.3 LiquidText

LiquidText is an app based upon the work of Craig S. Tashman and W. Keith Edwards. Its focus is upon the affordances of AR, which include highlighting, annotating, and note-taking [TE11], among others.

In this app, it is possible to import documents from the iCloud or the iPad's file system using the document picker provided by iOS. Websites and images can be imported as well. The imported documents are organized into folders and displayed as a grid or list. Thumbnails, titles, and additional information such as the size and the last modified



(a) File Manager - Grid View (b) Document View

Figure 4.9: LiquidText - File Manager and Document View

date visualize the document items. Documents can be moved between folders and sorted by date or by name, but they always seem to be in ascending order. Figure 4.9a demonstrates the grid view of LiquidText's file manager. Moving items via drag and drop is not supported.

Opened documents are rendered inside the left column of a resizable split view as depicted in Figure 4.9b. Its pages are aligned vertically and scroll continuously. The paid version allows the user to arrange multiple documents beneath each other. The workspace is placed inside the right column and can be scrolled and zoomed independently from the document.

LiquidText implements its own idea of how knowledge work can be performed in an effective way. Its workflow is based upon creating excerpts by first selecting text passages, then dragging and arranging them onto the workspace. During the selection, the area where the finger touches the content is magnified in a separate view to allow a more precise input. This avoids the common problem of the user's finger covering the area of the selection during the touch gesture.

Multiple excerpts can be grouped by being dragged onto each other and then closely positioned. Groups remain visually connected as seen in Figure 4.9b. Excerpts are linked to their position in the document, which is indicated by a small icon containing a blue arrow. Tapping this icon pans the document to the desired position. Selected text passages can be highlighted, and the paid version supports ink drawings and a freestyle highlighter. Drawing is not limited to the document, rather it is possible to draw on the document and the workspace, as well as onto excerpts. Drawings can be linked with other content, such as notes or excerpts, by simply connecting them by stroking a line with the pencil.

Interesting Details

Despite the mechanisms for extracting content from the document into the workspace and grouping it together, there are some unique features, which are all linked to the ability to squeeze parts of the document by using the pinch gesture.

First, instead of showing search results in a separate list view, the implemented functionality allows the user to squeeze the document so that the found occurrences are displayed closer together. Holding the page in place, while tapping a link button or tapping multiple link buttons at once offers the advanced functionality of comparing different parts of the document.

Figure 4.10: LiquidText - Content Search

Second, in the *highlight view* mode, the document can be squeezed to offer a better overview of the extracted text passages.

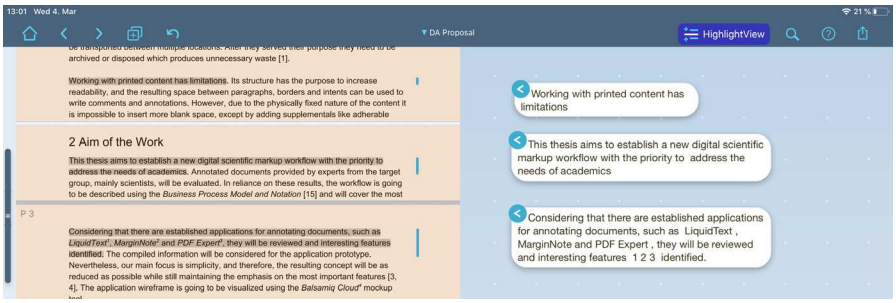


Figure 4.11: LiquidText - Highlight View

Third, parts of the document can be squeezed to increase the readability of other paragraphs to compare them more efficiently.

Another detail, already mentioned above, is that the paid version of LiquidText enables the user to draw and highlight passages anywhere across the document and the workspace. Drawn annotations can be linked to excerpts by simply drawing a line between them using the pencil.

To retain an overview, the LiquidText draws a short text summary diagonally onto the workspace if it is zoomed out too far, and the excerpts become unreadable.

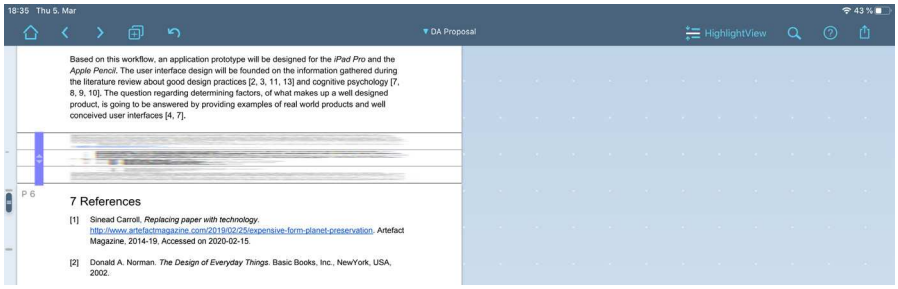


Figure 4.12: LiquidText - Squeezing Document Areas

Possible Improvements

LiquidText implements a very interesting approach to working with documents with a focus upon the AR process, which makes it difficult to suggest improvements.

The link buttons displayed on the top-left side of the excerpts are rather small, which makes them hard to tap without the pencil. Apple recommends a minimum tappable area of 44pt x 44pt [Appa]. The same applies for the document scrollbar, and especially for the *Table of Contents* button on the bottom-left side of the UI, which can be easily overlooked.

The app's color palette is also rather limited with five colors in the free version. The 32 colors of the paid version could have been better chosen but that is a subjective assessment.

Conclusion

LiquidText offers a unique approach to AR based upon the outcome of scientific research. The features it offers in terms of document squeezing push the boundaries of paper as an *analog* medium. Extracting content from the document and organizing (grouping) it on the workspace works well, and it is playfully indicated as simulated surface tension when two liquid drops touch each other.

4.2.4 MarginNote 3

MarginNote 3 offers a vast array of features. To describe everything in detail is not possible within the scope of this thesis, therefore, we have focused upon the most interesting details. MarginNote 3 distinguishes between *Documents*, *Mind Maps* and *Card Decks*, each listed in its own category, *Document*, *Study* and *Review*, as indicated in Figure 4.13.

The file manager displays items using a list or grid view, sorting them either by last modified date or alphabetically by name. A descending sort order is not supported. Documents can be organized by using folders and subfolders or by assigning one or more categories to filter the content.

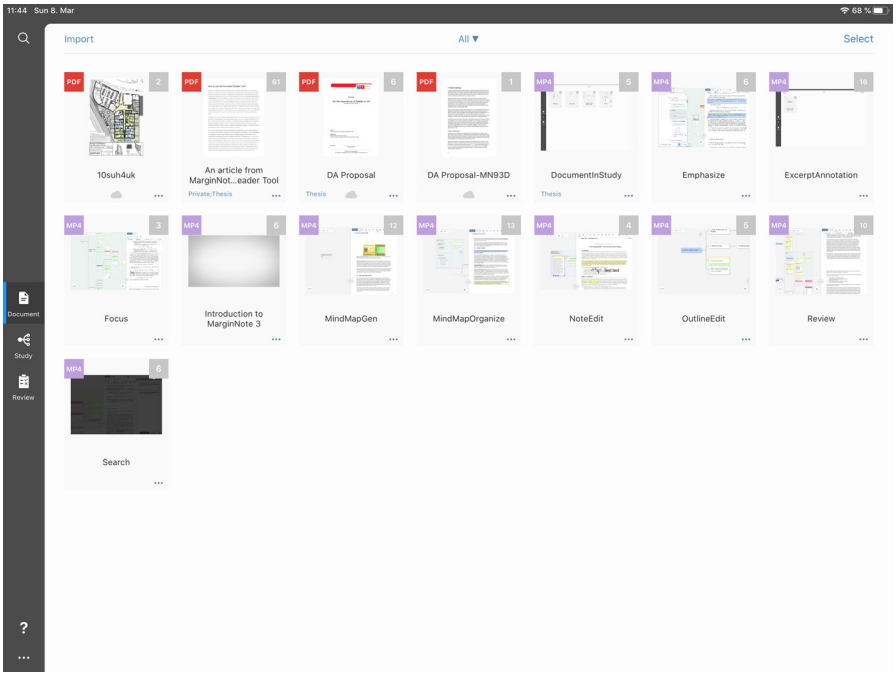
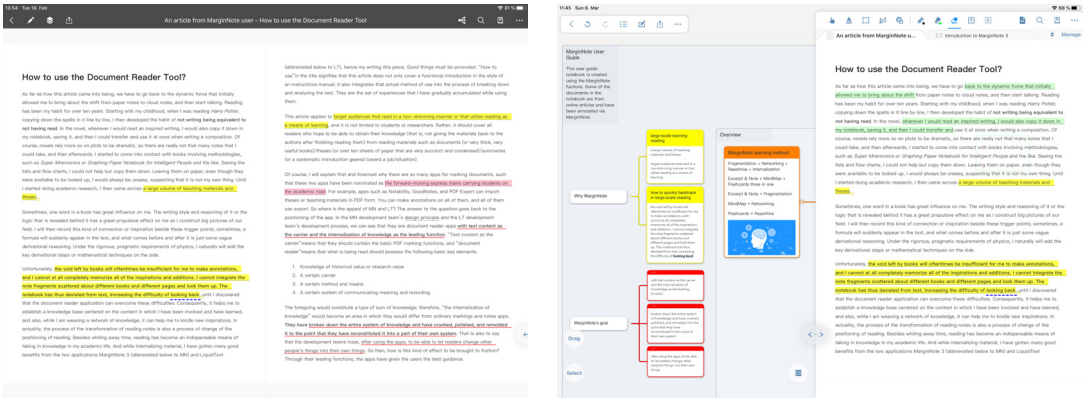


Figure 4.13: MarginNote 3 - File Manager

Similar to other apps, PDF files can be imported using the system file picker. MarginNote 3 supports adding files from Evernote, allows the importing of websites, offers to transfer files from a PC or macOS via WiFi and includes the ability to add MP4 videos from the iPad's photo library. Importing via WiFi is performed by a local website hosted by the app, allowing access over web browsers. However, it is not possible to add identical documents twice.

Documents are displayed either in the *Document Mode* (Figure 4.14a) or can be upgraded to so-called *Mind Maps* and consequently be presented in the Study Mode (Figure 4.14b). While the *Document Mode* renders only the document itself, the *Study Mode* splits the view into a workspace area that contains the *Mind Map* and the document. Multiple documents can be added to a single *Mind Map*, and up to three independent views can be enabled, either vertically or horizontally split, to compare different parts of the document. A slider enables the user to distribute the space between those views, similarly to LiquidText. In all the viewing modes, the document can be annotated with the provided tools, such as text selection, drawing a rectangle, drawing a free shape, a regular pen for adding ink-like drawings, a highlighter, an eraser and the adding of text.

Each tool offers a certain level of customization. The pen supports six different sizes and three different modes, in particular, drawing straight lines, drawing lines with a constant thickness, or drawing lines with a dynamic thickness depending upon the applied force. Tools can also be cloned to quickly access often used configurations. It is possible to select colors individually for each tool. The color picker initially presents five colors, each



(a) Document Mode (b) Study Mode

Figure 4.14: MarginNote 3 - Document and Study Mode

of them replaceable with custom colors.

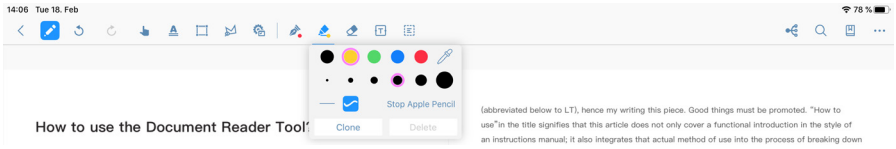


Figure 4.15: MarginNote 3 - Toolbar

Notes can be easily created by selecting text and dragging it onto the workspace. A magnifying glass view increases the precision during the selection, as already observed in LiquidText. For the Apple Pencil, a lasso tool is included for the user to freely select content. Once the notes are positioned on the workspace, they can be arranged and organized in many different ways, such as grouping and merging. Enabling a so-called *Focus Mode* for a selected note automatically adds all the subsequently created notes as a child.

For selected expressions, a research browser is available. Once the preferred information is found, on Wikipedia for instance, it can also be dragged to the workspace. Existing notes can also be exported as flash cards to the *Card Deck* to be reviewed as seen in Figure 4.16.

Interesting Details

MarginNote 3 initially opens documents in a PDF-compliant way. Only when the user desires to create a *Mind Map* does the document upgrade to MarginNote's own format. The *Mind Map* offers sophisticated functionalities that extract text passages and images from the document and organize and connect them as needed.

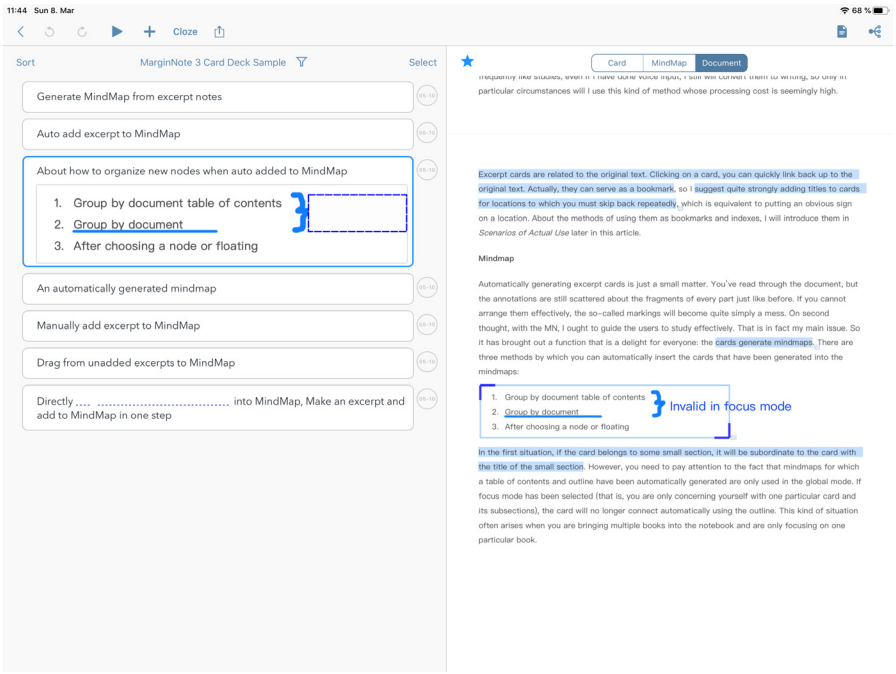


Figure 4.16: MarginNote 3 - Card Deck

Tools can be cloned to customize and easily switch between them. The cloned tools appear in the toolbar next to the original tool.

A feature called *page cropping* allows the user to reduce or increase the page bounds. The new frame is applied to every page in the document. Unfortunately, there does not seem to be a reset button to this feature, therefore, it is difficult to return to the original page size.

MarginNote 3 implements a learning aid, the so-called *Emphasize Mode*. Once this mode is activated, words and expressions are visually circled. Consequently, they can be underlined by selecting them by drawing a rectangle around the desired areas. After this is completed, tapping the *Recall Mode* button blanks out all the emphasized words and areas. Consequently, they can be filled in again as a learning exercise for these keywords.

Possible Improvements

While MarginNote 3 implements a large number of features, some details could be improved. Currently, it is not possible to draw annotations outside the document's bounds. Even if the *Study Mode* is active, the document is separated from the workspace itself. It is not possible to freely draw onto the workspace.

Custom colors are also supported. The basic color picker shows a pipette icon, indicating the option to pick a color directly from the document or screen. However, the pipette icon does not offer this functionality; tapping it reveals a custom color picker, a feature

that is not indicated by the pipette and can be easily misinterpreted. The newly chosen color replaces the already selected color in the basic color picker. To this point, we have not discovered any way to reset these colors to their default values.

The applied highlighter color is never identical to the selected color, which is a consequence of the added transparency. Nonetheless, the highlighted areas cover the content to a certain degree. Highlighting the same area multiple times allows the content to be completely covered.

Conclusion

MarginNote 3's focus is based upon learning, rather than simply annotating documents. As previously mentioned, it contains a large number of features for annotating documents but also for creating mind maps and card decks and other items. As a consequence, its interface has become quite complex. Its functionality is explained in tutorial videos shipped with the app, but nevertheless many features can be easily overlooked.

4.2.5 PencilKit (by Apple)

PencilKit [Appp] ships with the new iPadOS 13 and is tightly integrated with the operating system. It is also available as a library that can be integrated into custom apps. According to Apple, it has reduced the latency to 9 ms in combination with the new iPad and pencil models. This latency can only be achieved by using a Metal canvas and predicted touches [App19].

The integration of PencilKit into iPadOS 13 has added a pencil gesture; dragging the pencil from one of the bottom corners to the center of the screen captures the currently displayed content and allows it to be annotated. In combination with the Safari web browser, it is also possible to annotate the full website, even the hidden content that can only be reached by scrolling down. The result can be saved either to files or to photos. Additionally, it can be copied to other apps using the system's sharing action.

The PencilKit library consists of two main components, the canvas (PKCanvasView) and the toolbar (PKToolPicker), which can be easily implemented into apps. It supports tools for drawing, highlighting, selecting and erasing content. In addition, it has a color picker. The resulting images are both vector- and bitmap-based, which offers many ways of manipulating the drawn content using its tools.

As we experimented with PencilKit and implemented the sample app provided for research purposes, we observed that the canvas does not support drawing onto a PDF or any other content by itself. One option to solve this issue is to render the PDF in another view and place it in the background of the canvas. Unfortunately, this produces other limitations.

First, the PKCanvasView itself extends the UI ScrollView, but we could not take advantage of this. We needed to implement another ScrollView and arrange it behind the

[26] `https://www.apple.com`

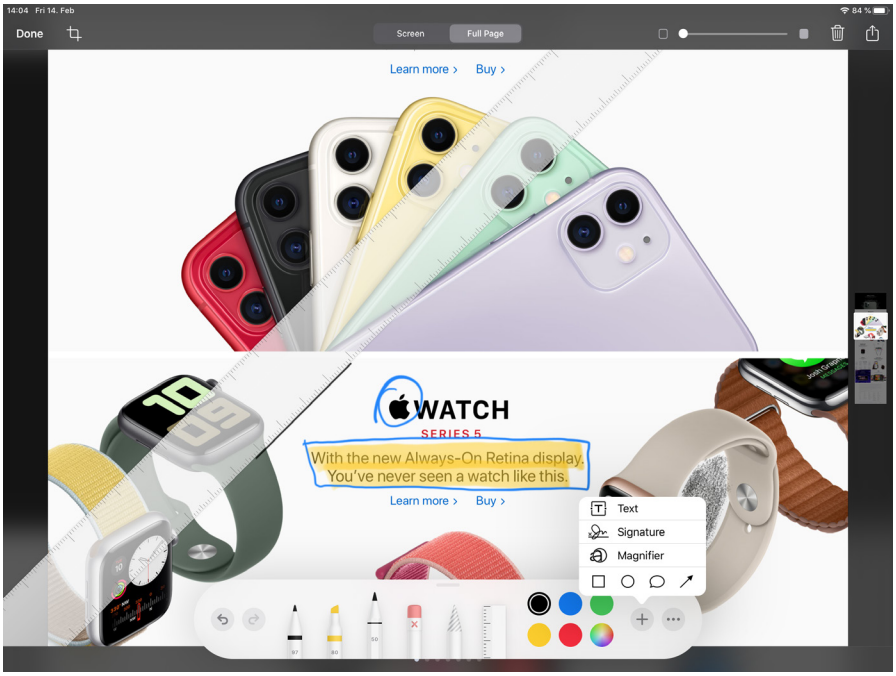


Figure 4.17: PencilKit - Annotating the Apple website[26]

canvas view. All the panning gestures needed to be forwarded to the second ScrollView to scroll both the canvas and PDF.

Another caveat is that PencilKit cannot be customized. Adding or removing tools or slightly adjusting the color picker is not possible either. Its content can be extracted only *as is*, and, therefore, manipulating its content is not possible.

Overall, PencilKit appears to be an excellent library that still lacks important functionalities, but it is possible that Apple will add more features in the future. For now, we cannot take advantage of PencilKit and need to rely upon our own implementation based upon the Apple SpeedSketch example.

4.2.6 Other Applications

Acrobat Reader

The Adobe Acrobat Reader is likely to be the best-known program for reading PDF documents. It adheres to the PDF document specification, has a neat UI, and supports basic annotations such as highlighting, ink drawings, commenting and others.

PDF Viewer

This app offers similar functionalities as the Adobe Acrobat Reader and PDF Expert. It implements an overview, in which all the pages are displayed as thumbnails including an

option to show only annotated or bookmarked pages. Multiple documents can be opened within tabs.

GoodNotes

GoodNotes' primary purpose is to take notes. During the creation of a new notebook, it is possible to choose from different covers and designs. Furthermore, it enables users to select the paper style, such as blank, squared, ruled and others. Annotating PDF documents is also possible. They are presented in the same way as other notebooks. GoodNotes is one of the few apps that does not cover the highlighted content. Instead, the highlighted area seems to be rendered behind the text but still remains in front of images. As a result, highlighting works for both text and images.

Loose Leaf

The only open source solution is offered by Loose Leaf [Wulb]. Similar to GoodNotes, its focus is upon note-taking, and it also offers different paper styles. Additionally, images and other content can be added. Inserting PDF documents is possible but very limited. Each PDF page needs to be added individually. The most interesting part of Loose Leaf is that the framework's source code is available under the MIT license. It is implemented in Objective-C-[Wikd] and takes advantage of the graphics processing unit (GPU) accelerated Metal API.

4.2.7 Application Analysis Conclusion

The analyzed apps can be divided in two categories: those that have implemented PDF-related functionality only, such as PDF Expert, and those that offer additional features for AR and knowledge work, such as LiquidText and MarginNote 3.

The documents are organized in a very similar way across all the apps. Two layouts are extremely common, a grid view and a list view. While the grid view has more space for a detailed thumbnail of the document, the list view allows the display of a longer document name.

The PDF document standard is extremely limited concerning the requirements for AR. It consists of a predefined set of annotation types, restricts annotations to the page's bounds, and lacks more sophisticated functionalities, such as creating and managing excerpts, for instance. Apps such as LiquidText and MarginNote 3 compensate for this by introducing their own workspace and providing extended functionalities beyond the limits of the PDF format. The caveat of this approach is the lack of compliance and interchangeability during the document export.

We encountered different strategies with regard to how the aforementioned apps cope with this. Workspaces are stored and exported in their own (proprietary) formats, containing all the necessary information. Another very common behavior is that the additional

information, such as excerpts or mind maps, can either be exported separately or added as extra pages to the exported PDF document.

All the programs offer low latency when using the Apple Pencil, indicating that the canvases are likely to have been implemented using Metal [Appm].

Both LiquidText and MarginNote 3 allow the user to extract content by selecting text passages and dragging them onto the workspace where they can be arranged and structured. However, squeezing the document in LiquidText is a unique feature that no other app offers. As an alternative, MarginNote 3 allows the user to divide the document view into up to three separate views. A squeezed document in LiquidText keeps its logical order, while separate views in MarginNote 3 can be scrolled completely independently of each other. This can be regarded as both an advantage and a disadvantage.

Compared with our findings in Chapter 2, most of the apps offer too many functionalities. MarginNote 3, for example, offers an extremely large number of features and its UI has reached an level of complexity by which users might feel overwhelmed.

4.3 A new Digital Workflow

> "A workflow consists of an orchestrated and repeatable pattern of activity, enabled by the systematic organization of resources into processes that transform materials, provide services, or process information. It can be depicted as a sequence of operations, the work of a person or group, the work of an organization of staff, or one or more simple or complex mechanisms [Wikh]."

Based upon Chapter 2 and Chapter 4.2, we collected the following ideas that our workflow needs to cover. There are two types of main activities, *document organization* and *annotation handling*.

- Document organization focusing upon a good overview and functionality.
  - Create projects
  - Add PDF documents from different sources
  - Send PDF documents to the app
  - Rename documents or projects
  - Move documents between projects
  - Display documents for AR
  - Share/export documents
  - Delete documents or projects that are no longer in use
- Document annotations and text highlighting

- Choose between multiple colors for pen and highlighter
- Add annotations via pencil to a document
- Highlight text passages via pencil
- Erase annotations and highlights
- Move annotations (do not move highlights)
- Create custom pencils and highlighter
- Overview of annotations and highlights
- Pan/zoom to selected annotations and highlights

4.3.1 Business Process Model and Notation

To describe our workflow, we used the Business Process Model and Notation (BPMN 2.0), which is similar to the Unified Modeling Language (UML). BPMN is a standardized diagramming language for business processes and suitable for our purpose. We used the following subset of the available groups [Lucb] and adapted them for our benefits:

- **Events** - Circle
  Events define the start and the end of each workflow.
- **Activities** - Rectangle
  Activities represent actions. In our case, these actions are either performed by the user or the system.
- **Gateways** - Diamond Shape
  Gateways divide and rejoin the flow. Their optional content indicates different conditions. In our case, we only use a cross which indicates an exclusive-or (XOR) or leave them empty.
- **Connectors** - Lines
  As the name suggests, connectors link different events. The connector's arrow indicates the direction of the flow.
- **Pools** - Rectangles divided by vertical or horizontal lines
  Pools are divided into lanes the same way in which real swimming pools are divided into swim lanes. For our workflow, we used these lanes to distinguish between user and system activities.

4.3.2 Document Organization Workflow

The main purpose of document organization is to preserve a general overview of all the documents. It incorporates many aspects, beginning with *adding documents* from different sources, *organizing documents* such as maintaining a good overview within the app, to *sharing annotated documents* with other apps.

Create Projects

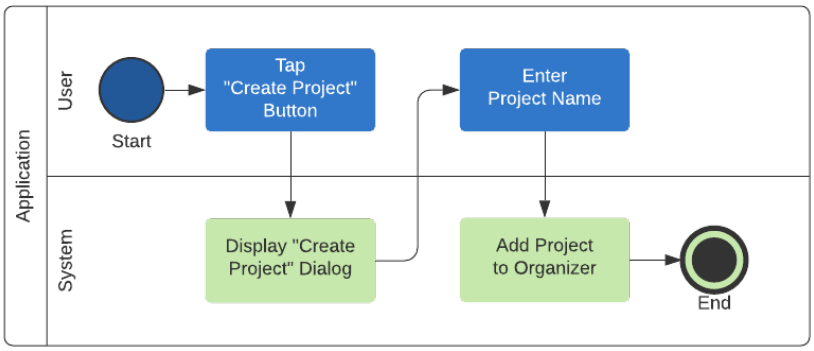


Figure 4.18: Creating Projects

Projects are similar to well-known folders in a file system, with one exception. Unlike folders and subfolders, they cannot have other projects as child objects. Each project is defined by its name.

Start: The app is in the foreground and the *Create Project* button is available.

End: The project has been added to the organizer and is displayed.

Workflow: Our app provides a *Create Project* button. Once tapped, dialog is displayed and requests the project's name to be entered. After the name has been entered and confirmed, the new project is added to the organizer.

Adding PDF Documents

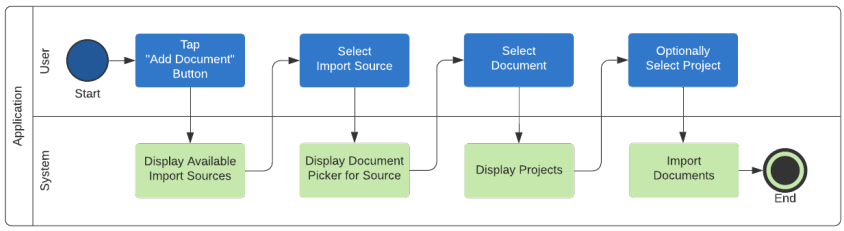


Figure 4.19: Adding Documents

Documents can be imported in two ways: Either from our app as described in Figure 4.19 or by using the *Share* button inside other apps as shown in Figure 4.20 in the next section.

Start: The app is in the foreground and the *Add Document* button is available.

End: The documents have been imported and are displayed.

Workflow: Our app provides an *Add Document* button. Once tapped, a list of possible document sources is displayed. The most common entry will be the system file browser, but the list can be extended to support other sources, such as alternative cloud drives or websites, as described in *Inspirational Applications* in Chapter 4.2. If the desired source has been selected, a document browser opens, and the user can select one or more PDF documents. After the selection is confirmed, a list of available projects is displayed, and the user can optionally select one as an import target. If no document is selected, the document is imported to the root folder of our app. Confirming the selection begins the import process.

Importing PDF Documents from Other Applications

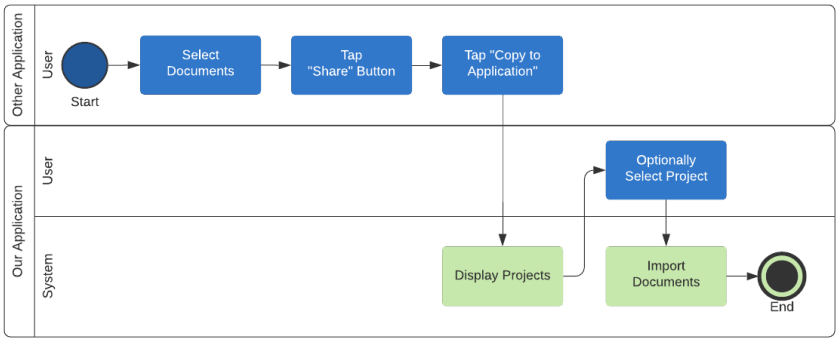


Figure 4.20: Importing Documents

Our app implements a so-called "share extension," and, therefore, appears as the target in the iPhone's share dialog. If, for instance, an email has a PDF file attached, the email program can copy the attachment to our app via this dialog.

Start: An external app is in the foreground and allows PDF documents to be shared.

End: The documents have been imported to our app and are displayed.

Workflow: Once a document is received by our app, the list of available projects is displayed in the same manner as in the previous workflow (Figure 4.19). The user can optionally select one as an import target. If no document is selected, the document is imported to the root folder of our app. Confirming the selection begins the import process.

Rename Documents or Projects

Start: The app is in the foreground and documents or projects have been added. Furthermore, at least one document or project is visible in the current view.

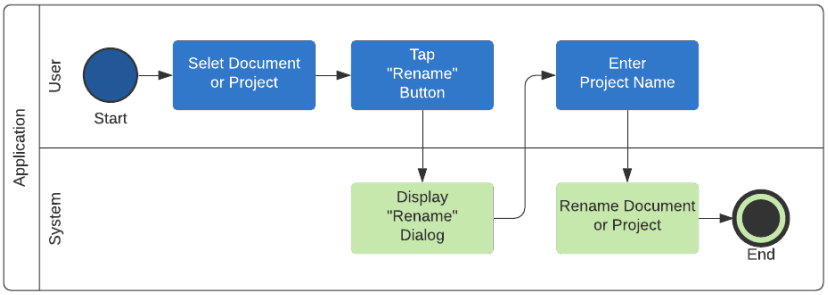


Figure 4.21: Rename Documents or Projects

End: The document or project has been renamed.

Workflow: After a document or a project has been selected, the *Rename* button is displayed. Tapping it opens a popup dialog, in which the existing name can be edited. After the dialog is closed, the document or the project has been renamed.

Moving Documents Between Projects

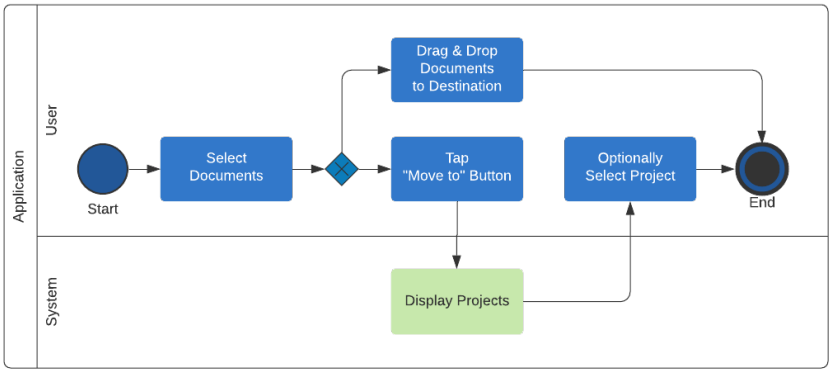


Figure 4.22: Moving Documents

Moving documents between projects delivers an important sense of ease to organizing the imported files.

Start: The app is in the foreground and documents and projects have been added. Furthermore, documents are visible in the current view.

End: The documents have been moved to the desired project or the organizer's root folder.

Workflow: Our workflow offers two approaches to moving documents. First, select one or more documents, then drag them onto the desired project and drop them. The projects will be moved immediately. If the organizer currently displays the content of a project, and a drag is initiated, a "parent" drop target is displayed to move the documents out of the current project. The alternative approach is to select one or more documents, then tap the displayed *Move to* button. Once tapped, a list of projects is shown, from which the desired target location can be selected. After the target has been confirmed, the documents are moved.

If projects have been selected in addition to documents, dragging does not work nor does the *Move to* button appear.

Displaying Documents

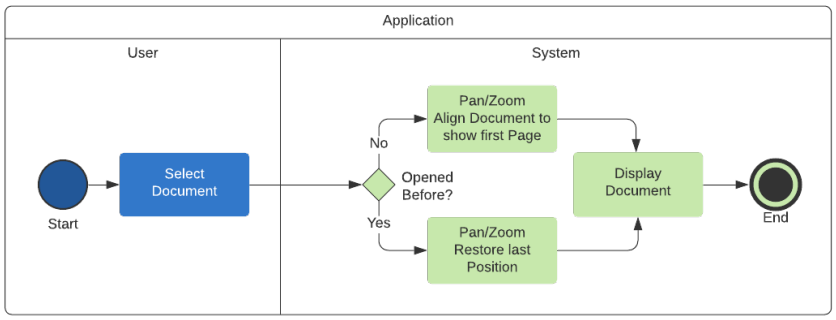


Figure 4.23: Displaying Documents

Start: The app is in the foreground and at least one document has been added.

End: The selected document is visible and ready for editing.

Workflow: The user selects a document in the document organizer and the document is opened. If it is displayed the first time, it is presented in a way that ensures readability and enough blank space for annotations. It is zoomed to 100% and aligned to the top-left corner of the screen. If the document is opened at another time, the last view position is restored, allowing the user to immediately continue working. The canvas enables the user to pan and zoom the document at any time to align it for their personal needs.

Sharing/Exporting Documents

Sometimes it is necessary to hand annotated documents back as feedback and, therefore, our workflow needs to manage this in a smooth way. The real challenge is not within the workflow itself but rather in the implementation as described in Chapter 6.

Start: The app is in the foreground, documents can be selected and the *Export* button is visible.

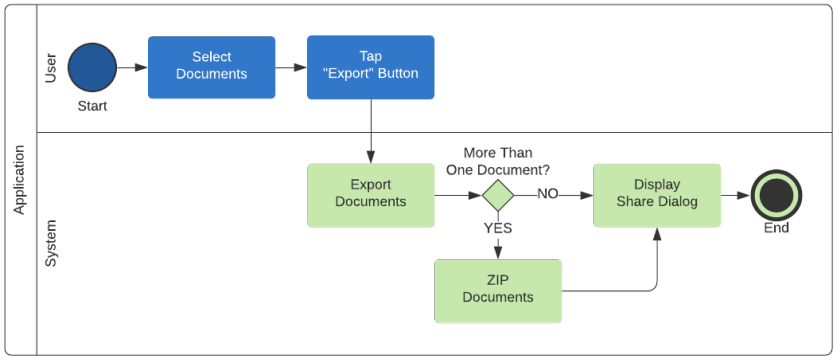


Figure 4.24: Sharing/Exporting Documents

End: The selected documents have been exported and the share dialog is visible.

Workflow: Once one or more documents have been selected, the *Export* button becomes available. If it is tapped, then the available documents are exported. If more than one document has been selected, the documents are bundled within a ZIP file. After the export is completed, the share dialog becomes visible.

Deleting Documents

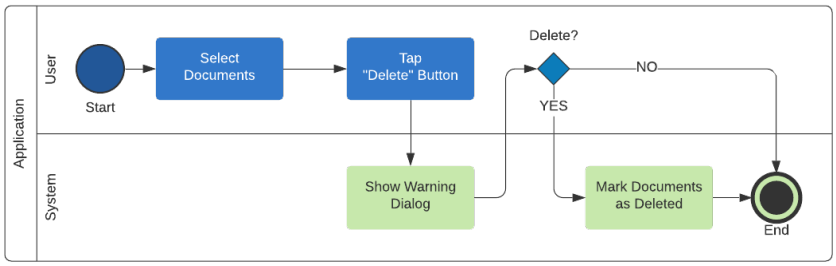


Figure 4.25: Deleting Documents

It is a good practice to make all destructive actions reversible. To meet this requirement, we implemented a trash bin. Deleted documents are only marked as deleted and disappear from the organizer; however, they can be listed in the trash bin. The mark can be removed at any time to *restore* the deleted documents. Documents marked as deleted can also be irreversibly purged, similar to emptying the trash bin as explained in the following section, *Purging Documents*.

Start: The app is in the foreground, documents can be selected and the *Delete* button is visible.

End: The selected documents have been tagged as deleted and are no longer visible in the document organizer.

Workflow: Once one or more documents have been selected, the *Delete* button becomes available. If it is tapped, a warning dialog appears. After confirming the action, the available documents are marked as deleted and disappear from the document organizer.

Purging Documents

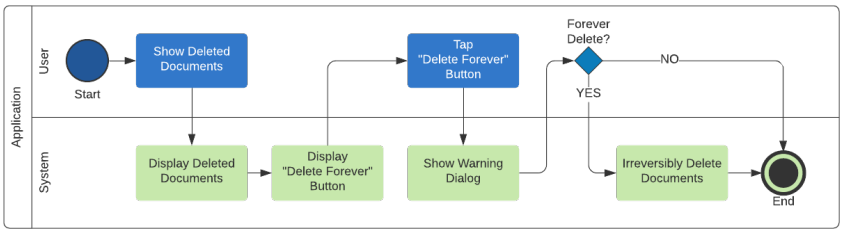


Figure 4.26: Purging Documents

Start: The app is in the foreground and at least one document has been marked as deleted.

End: All documents marked as deleted have been irreversibly deleted from the app.

Workflow: If the *Trash* button is tapped, the deleted documents and the *Delete Forever* button become visible. If the user taps "Delete Forever," a warning dialog appears similar to the regular delete forever warning dialog. After confirming the action, the available documents are irreversibly deleted from the app.

4.3.3 Annotation Workflow

Annotation handling involves *presenting the document* to the user, *writing annotations*, *highlighting text passages*, and manipulating existing annotations.

Annotating digital documents on the tablet needs to feel similar to or better than annotating hard copy documents, therefore, we need to implement a set of writing utensils that mirror real behavior in the best possible way.

Choosing Colors

Changing the color has to be as simple as changing the tool itself. Each tool has its individual assigned color palette, hence, the color of the pen is independent from the highlighter's color. The tools are configured globally, hence, the last tool selected and its color are remembered when switching between different documents. In our findings, only three colors were mentioned, but the general consensus indicated that more colors would be used if they were available. Therefore, we present a predefined set of colors

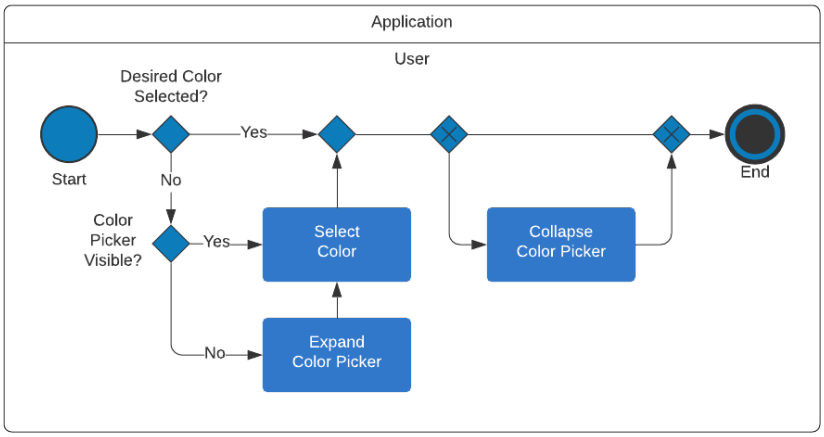


Figure 4.27: Choosing Colors

from which to choose, as described in Chapter 5.4. Colors can be changed by selecting the desired color in the color picker if it is visible. The color picker can be collapsed to avoid unnecessarily covering the canvas.

Start: The app is in the foreground and a document is displayed.

End: The desired color is selected.

Workflow: If the user would like to change the color, the color picker can be expanded by tapping on the *Color Palette* button if it is not already visible. As a result, the color picker becomes visible and the desired color can then be selected. After the procedure, the color palette can either remain visible or it can be collapsed by tapping the*Color Palette* button again.

Annotating/Highlighting Content

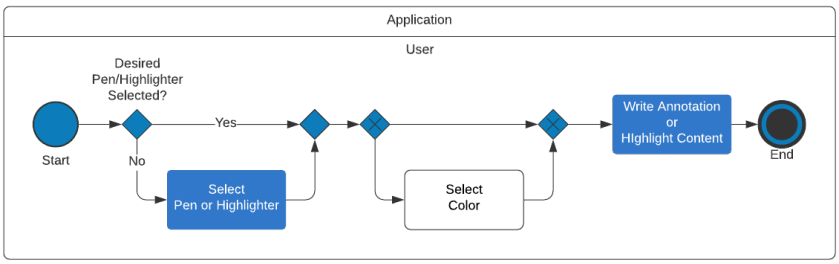


Figure 4.28: Annotating/Highlighting Content

After the document is displayed, the user needs to be able to write annotations or

highlight text passages in the same way as they would in a printed document lying open on the desk. The available tools have to be easily accessible and the user switching between them must be able to do so without any obstacles. Each tool has an assigned default color: green for the pencil and yellow for the highlighter. We have chosen green because it emphasizes the annotations against the common black text. Green also has a positive connotation as observed in Chapter 2.2.

Start: The app is in the foreground and a document is displayed

End: The desired tool is selected, and the annotations are drawn or content has been highlighted.

Workflow: If the desired tool is not active, it is simply selected by tapping on it in the toolbar. After the tool has been selected, changing the color is optional. Depending upon which tool is active, annotations can either be written in a similar way to holding a real pen or text passages can be highlighted.

Erasing Annotations or Highlights

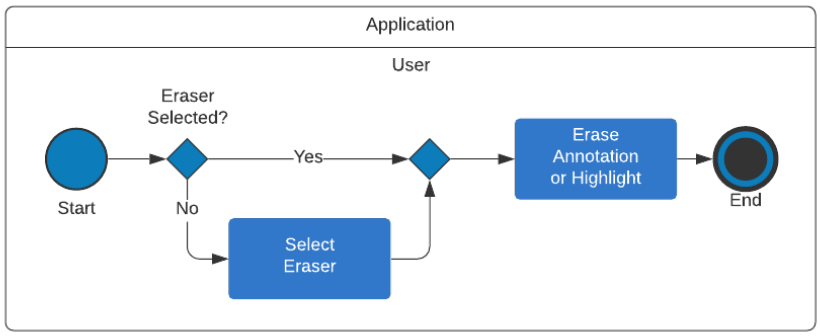


Figure 4.29: Erasing Content

The analysis of other apps demonstrated that erasing usually works in two modes. In the first mode, the whole annotation is removed, even if the eraser only touches a part of it. The second mode erases only the tiny part of the annotation that the eraser touches. Both modes have advantages and disadvantages, which we describe in more detail in Chapter 6. For our first implementation we have chosen the first mode since it is easier to implement.

Start: The app is in the foreground and a document is displayed.

End: The eraser is selected and the annotations have been erased.

Workflow: The eraser tool is selectable, just like the pen and highlighter. It works like a regular pen but removes every annotation or highlight it touches.

Move Annotations

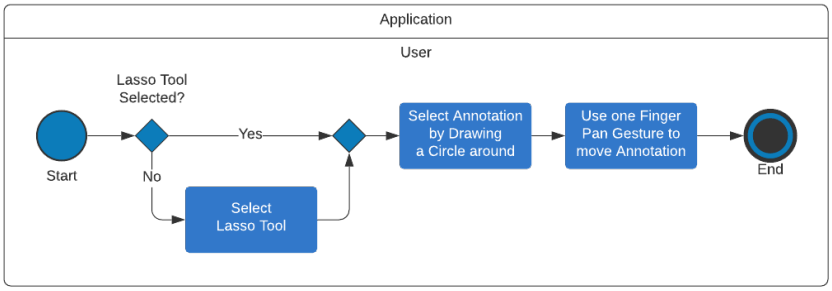


Figure 4.30: Move annotations

Moving annotations is a comfort function to avoid having to erase existing annotations and drawing them somewhere else. Instead, it should be possible to move them to another position on the workspace. Moving highlighted text passages is not necessary since the document's content would not be included in the moving operation. To move a document, the lasso tool has to be selected. It works like a regular pen but draws a dashed polygon. Once the lasso tool is lifted, every annotation it encloses is selected and ready to be moved with a single-finger gesture.

Start: The app is in the foreground and a document is displayed.

End: The lasso tool remains selected and the circled annotation has been moved to a different location.

Workflow: The lasso tool is selectable, similarly to the pen and highlighter. A dashed polygon can be drawn and once the pencil is lifted and the gesture finishes, every annotation touched or enclosed is considered to be selected. The selection can then be moved using a single-finger panning gesture.

Panning/Zooming the Document

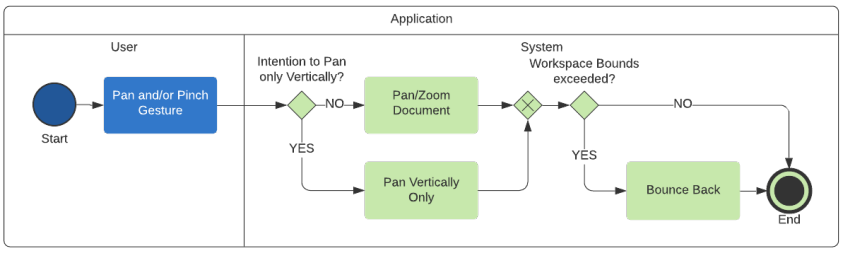


Figure 4.31: Panning/Zooming the Document

The customary gestures for zooming and panning are pinching with two fingers and swiping with one or two fingers, respectively. It is important that the standard gestures provided by the target platform are used in case they are different from the gestures mentioned before. When starting the gesture, the system identifies the user's intention. If an intended vertical panning is detected, the horizontal direction is locked. Otherwise, all directions and zooming are permitted. With the previously described gestures, the intention of a vertical pan is probable if only one finger is used and the swipe vector is closer to the y-axis than to the x-axis.

Start: The app is in the foreground and a document is displayed.

End: The document's desired zoom level and position are visible

Workflow: After a drag or pinch is initiated, the system decides whether a vertical scroll is intended. If this is the case, the document locks the horizontal scroll direction and only scrolls vertically. Otherwise, the document can be freely panned and zoomed. If the bounds of the workspace are exceeded, the document bounces back to keep the visible area inside the valid boundaries.

Undo/Redo

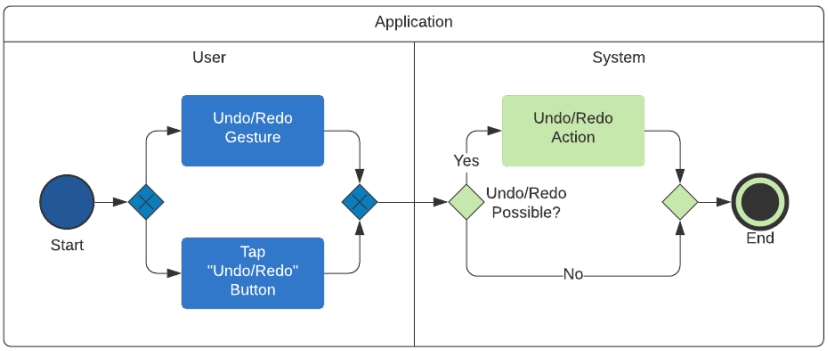


Figure 4.32: Undoing/Redoing Actions

Every drawing, highlighting or erasing action can be undone and redone via gestures or visible buttons. It is important that these gestures conform with the expectations of the iPad's standard gestures to improve the user's experience by transferring their already existing knowledge, as described in Chapter 3.2.1:

- Undo: three-finger swipe left (or three-finger double tap)
- Redo: three-finger swipe right

If there are no actions that can be undone or redone, then the appropriate buttons are disabled, and the gestures are ignored.

Start: The app is in the foreground and a document is displayed. There are actions available for either undoing or redoing.

End: The actions have either been undone or redone.

Workflow: The user triggers an undo or a redo either by tapping the associated buttons or by using the integrated gesture.

CHAPTER 5

Designing a Wireframe Prototype

5.1 Significant Concepts

Paper has established itself as a medium during the last centuries and become part of our working culture. Writing letters, signing contracts, publishing books, taking notes, and archiving information are just some examples of how paper is used. Over the years, people have grown accustomed to its properties and developed processes around its qualities. The inflexible nature of printed content has made it necessary to find a balance between the amount of information on a page and its readability. Consequently, paragraphs and insets build the blank space that can be used for annotations.

With the emergence of digital media, displayed information has become flexible and its usage has grown beyond the limitations of paper. Digital media enables people to manipulate the information and the way in which it is presented. By transcending paper, digital media offers seemingly unlimited possibilities.

Accordingly, the design objective for our app was to provide technologically adequate commenting mechanisms by enhancing the paper-like qualities of documents.

There were two main concepts upon which our app needed to focus:

- **Document organization:** Documents need to be imported, managed, and finally deleted once they are no longer in use.
- **Annotation canvas:** Documents need to be annotated, and these annotations need to be modifiable and outlined to create an overview.

5.2 Document Organization Concept

With regard to document organization, we identified several requirements that should be fulfilled.

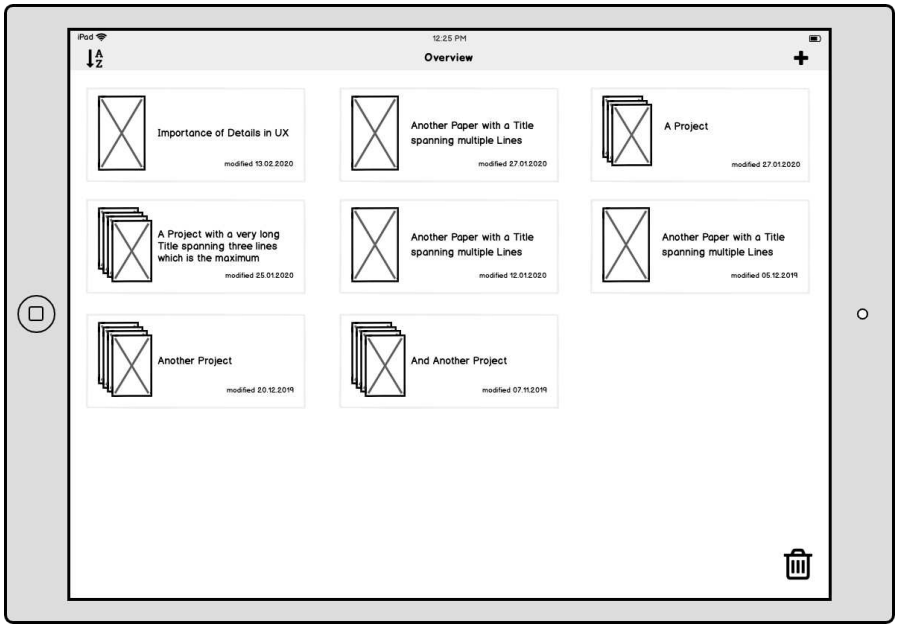
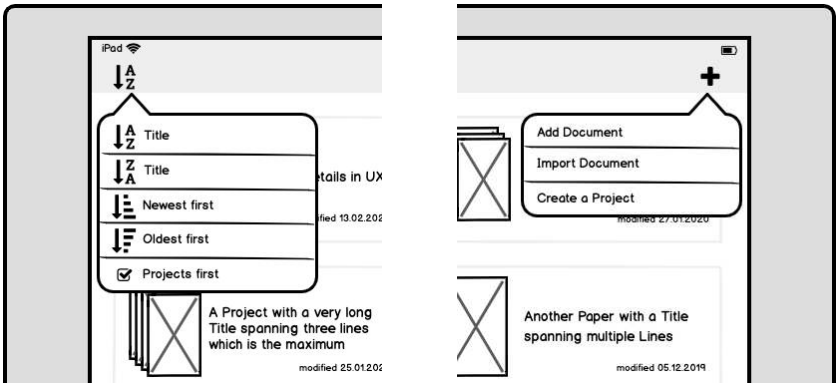


Figure 5.1: Document Organizer

Figure 5.1 visualizes the document organizer. Most of the apps that we reviewed in Chapter 4.2 support a grid view and list view. For our prototype, we have chosen to combine the advantages of list and grid views into a tile view. Each tile is large enough to contain a preview of the document and wide enough to contain a long name using up to three lines if necessary. Instead of small repetitive icons, thumbnails are more intuitive than a purely textual representation as explained in Section 3.2.1. In addition to the thumbnail and name, the tile contains the date when the document was last modified and in this way limits the information provided to the absolutely necessary. For future iterations of the prototype, we may further reduce this content and only show the modified date when the tiles are sorted by it. Documents can be grouped together by projects, which implements a similar concept to a folder in a file manager except that they cannot contain other projects. They are represented in the same way as documents, with the exception that their icon stacks up to five thumbnails. The top visible preview image is always the most recently used document's thumbnail. The navigation bar contains the title of the open project or "Overview" if no project is selected.

Content can be sorted by name and the date it was last modified, both ascending and descending as shown in Figure 5.2a. Alternatively, projects can be prioritized to appear before the other documents. A navigation bar menu item indicates the applied sort order

and when tapped, it opens a popover enabling the user to choose between the available options.



(a) Sorting Popover (b) Add Content Popover

Figure 5.2: Sorting and Adding Content

Content can be added using the *add* button at the top-right corner of the navigation bar. A popover lists the available options:

- Add documents from the system's default file browser.
- Import documents from different sources, such as web pages or cloud storage providers.
- Create a new project by entering its name.

If the app is newly installed and no documents or projects have yet been added, a placeholder, centered on the screen, provides a shortcut to the same options as those behind the *add* button in Figure 5.3.

The ability to undo actions is good practice and considerably improves the UX. Therefore, deleted objects are moved to a trash bin instead of immediately being purged. These items can be accessed using the *trash bin* button in the bottom-right corner. Items inside the trash bin can either be restored or irreversibly purged.

Long tapping an item activates the selection mode and expands the toolbar at the bottom of the screen as shown in Figure 5.4. Depending upon the type and the quantity of the selected items, different options are available:

- If only one document or project is selected, the *rename* button is enabled, leading to a popover to assign a new title.
- If one or more documents are selected (but no projects), the *move* and *export* buttons are enabled.

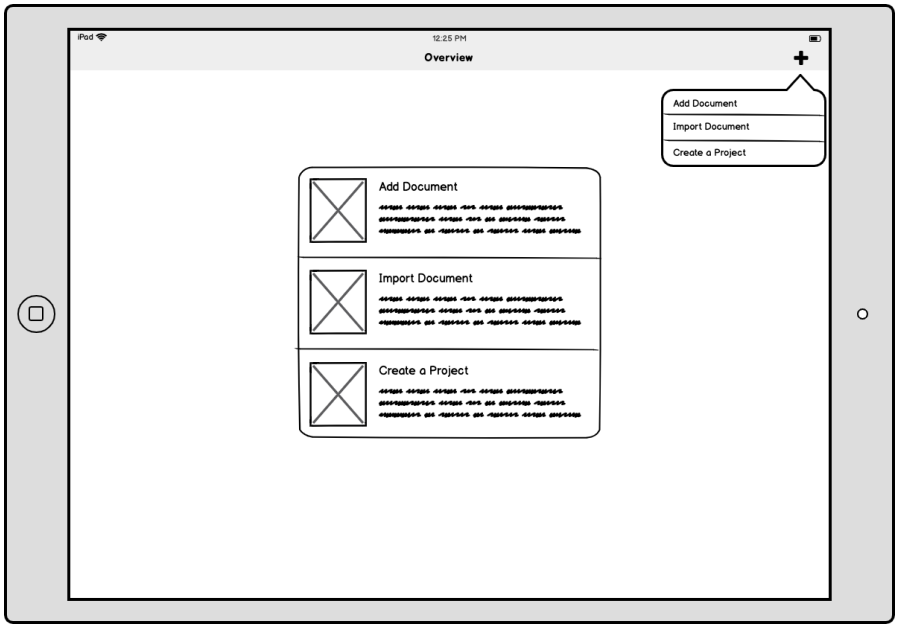


Figure 5.3: Empty Documents Organizer

- If only one project is selected, the *export* button is enabled.
- If one or more documents and/or projects are selected, the *delete* button is enabled.

Tapping the *move* button displays a popover dialog to select the destination, as indicated in Figure 5.5. Additionally, selected documents can be moved by drag and drop gestures. This functionality is part of the user's expected behavior when interacting with such an interface.

A tap on the *export* button exports either the selected documents or the content of the selected project. During the export process, a popover is visible that keeps the user informed of the progress. Once the export is completed, the share dialog allows the user to select the target app.

The *delete* button opens a popover dialog asking for confirmation. Deleting items moves them into the trash bin at first. This procedure can be undone until the trash bin is irreversibly purged.

5.2.1 Details

The prototype offers a few details to improve the UX.

Every project's preview stacks up to five thumbnails of its content. To increase the familiarity with a project, the top image always represents the last modified document.

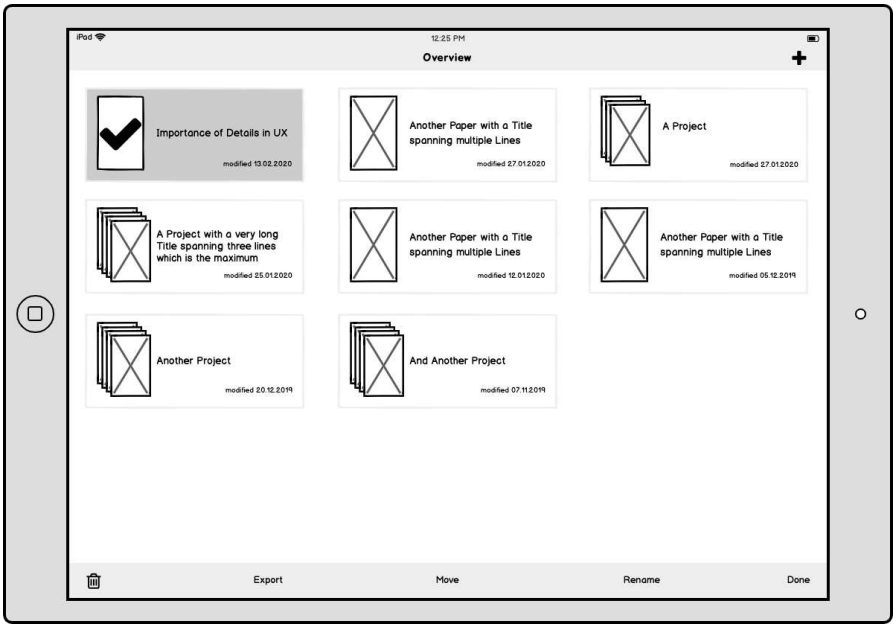


Figure 5.4: Toolbar for Selected Items

Users should always have the option to undo critical actions such as deleting documents. Therefore, a trash bin has been implemented that lists all the deleted items. To finally delete them, the trash bin can be purged.

The *trash bin* button becomes visible only if deleted items exist. Otherwise, it remains hidden to reduce the user's cognitive overload as explained in Chapter 3.2.1. If the trash bin is accessed, the toolbar's color changes to a reddish tone, indicating the difference between the organizer and the trash bin. Red also acts as a warning and signals to the user to be careful with any potential actions taken at this point (Section 3.2.2).

The confirmation dialogs have meaningful button titles. For instance, the buttons in the popover for deleting items are labeled *cancel* and *delete*, rather than *yes* and *no*, to further reduce the cognitive load.

5.3 Workspace Concept

The workspace is the heart of our app. In the prototype, we narrowed the necessary features down to a minimum. The prototype displays the PDF document seamlessly on an infinite canvas without drawing its page borders. Furthermore, it performs rendering in a feasible amount of time and is capable of handling pencil input in a responsive way. Smooth panning and zooming are mandatory for a good user UX. Figure 5.6 illustrates the layout involving the *Sidebar*, a vertical toolbar on the left and the *Overview*, a list containing all the existing annotations, on the right. The navigation bar remains hidden as it is not needed. The *Overview* can be collapsed if it is not needed.

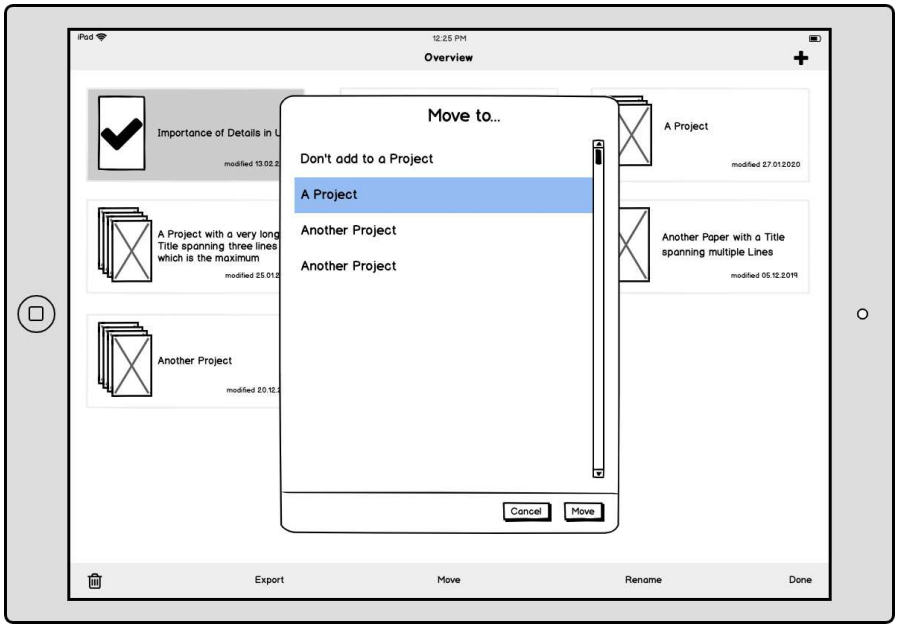


Figure 5.5: Move Documents

Common swipe and pinch gestures are used to pan and zoom the workspace. Panning works either by using a single finger or by using two fingers. If a single finger is used and only a vertical pan is intended by the user, the workspace locks the horizontal direction to increase the user's comfort. Locking the scroll direction is very common, as we observed during our app review in Chapter 4.2.

Zooming is limited to between 50% and 200% of the document's original size. The document's actual zoom level is displayed at the top of the document. This view also acts as a button and tapping it zooms the document back to 100%. If the zooming or panning limits are exceeded, the ScrollView applies the well-known rubberlike bounce effects.

While zooming the document, similar snapping positions appear, similar to those in the PDF Expert described in Chapter 4.2. The document snaps around its original file size and when the document's page height is close to the workspace height.

One of our main concepts is that the document becomes part of an infinitely large workspace, exceeding the bounds of the document's pages. Due to technical limitations, the term *infinite* is interpreted as *large enough* so that the workspace bounds are de facto never exceeded. As a result of such an infinite canvas, it is possible that a person might scroll too far from the visible document, such that it can no longer be found. There are several solutions to this, such as offering a button that brings the viewport back to the document or applying an indicator demonstrating the direction in which the document can be found. However, we have decided upon an alternative approach, therefore, we initially limit the size of the workspace and successively extend it once

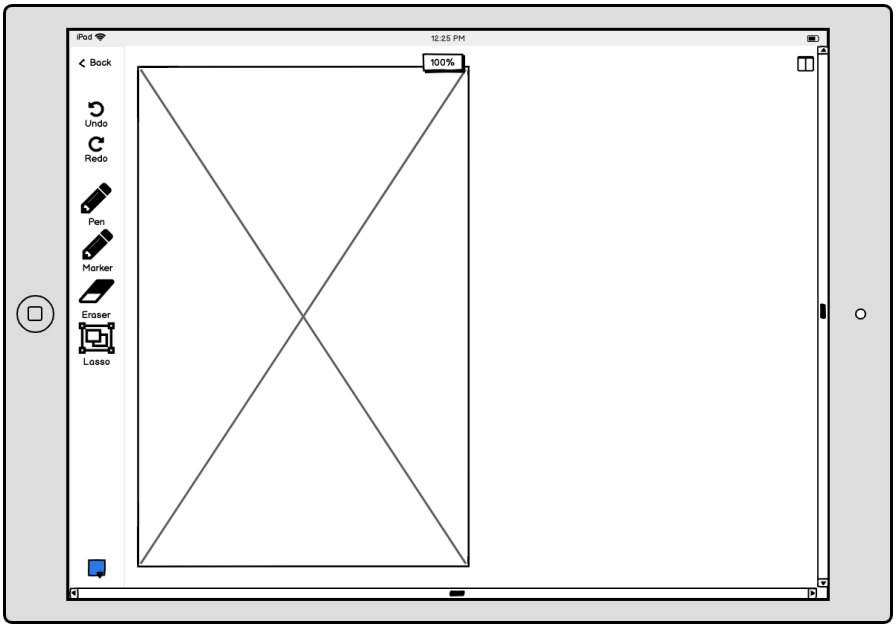
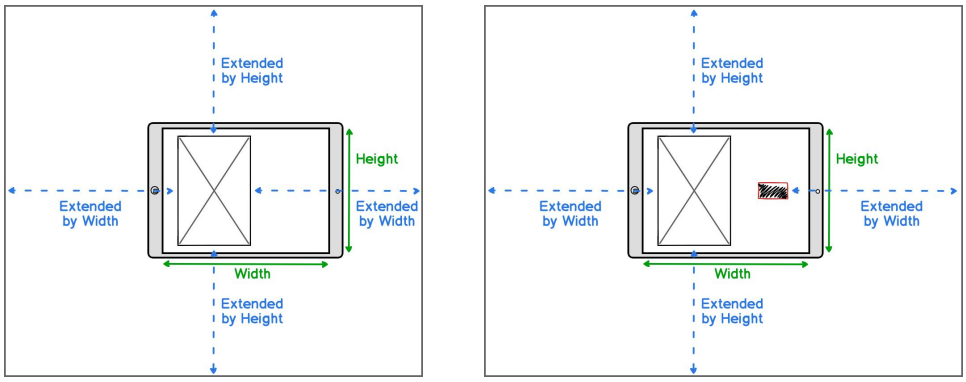


Figure 5.6: Workspace with Sidebar and Overview

annotations consume more space.

For the prototype, we have limited the workspace to the size of the document, plus one additional screen size for each direction, as seen in Figure 5.7a. Every time an annotation is appended, the workspace extends its boundaries. This behavior is visualized in Figure 5.7b.



(a) Original Workspace (b) Extended Workspace

Figure 5.7: Extending the Workspace

5.3.1 Sidebar

We refer to the toolbar attached to the left side of the UI as the *Sidebar*. It contains the available tools, the buttons for the *undo/redo* functionality, a colored button to toggle the color picker and the *back* button at the top left. The *Sidebar* has a white transparent background, therefore, is not distinguishable from the workspace unless a document is panned behind it. In Figure 5.6, the width of the toolbar is indicated by a slim gray line; however, in the final implementation the *Sidebar* has no visible border. It is kept as slim as possible and its empty space forwards pencil touch events through to the canvas.

5.3.2 Undo/Redo

The toolbar's *Undo* and *Redo* buttons are disabled if there is no executable action available. This functionality only records actions that affect the document's annotations such as writing, highlighting, erasing, and moving. It does not register panning, zooming or other UI-related actions or gestures, nor does it register color changes or pencil selections. Unlike in other apps, both buttons are always visible. Additionally, system gestures for undo and redo actions are also supported as this is the expected behavior described in Chapter 3.2.1.

5.3.3 Using the Tools

Needless to say, only be one tool can be selected at a time by tapping it either with a finger or the Apple Pencil.

- The pen, which is sometimes referred to as "ink," allows the user to draw annotations. The thickness of the line depends upon the force applied.
- The highlighter can be used to emphasize text passages and other content. The Apple Pencil's azimuth angle is not pertinent when using this tool; instead, the pencil's width is always aligned vertically to the page.
- The eraser removes every annotation it touches from the canvas completely. It is not possible to erase only parts of a stroke. This behavior may be subject to change in future iterations of the app.
- The move tool helps to select annotations and move them to a different location. This is useful in cases when the user would like to free up occupied space. The Apple Pencil is used to circle existing annotations. All intersected or enclosed annotations can be dragged around as a group using a single-finger gesture. Highlighted text passages are ignored in this behavior.

5.3.4 Color Picker

The color of each tool can be selected individually. The active color is observable in the color picker button of the *Sidebar* and in the selected tool itself. Tapping it expands or

collapses the color picker. Figure 5.3.4 shows the expanded picker. Depending upon the selected tool, the available colors are different as described in further detail in Chapter 5.4.

Figure 5.8: Expanded Color Picker

5.3.5 Overview

5.3.6 Details

With regard to the workspace, its UI components and its behavior, many important aspects need to be considered.

To increase the user's comfort while drawing, single tapping toggles the visibility of all unnecessary UI elements.

The trailing side of the *Sidebar* acts as a docking position for the document. If the document's leading side is dragged close to the bar, snapping is indicated by highlighting the targeted border for a short period of time. If the dragging gesture is completed during this time, the document aligns itself along the *Sidebar*. This alignment is not permanent; it is lifted when the user starts panning again.

While annotations can be moved to different positions, it was considered unnecessary to move highlights as well. Moving a highlight would imply the necessity of moving the highlighted text passage as well. Consequently, they are ignored for move actions.

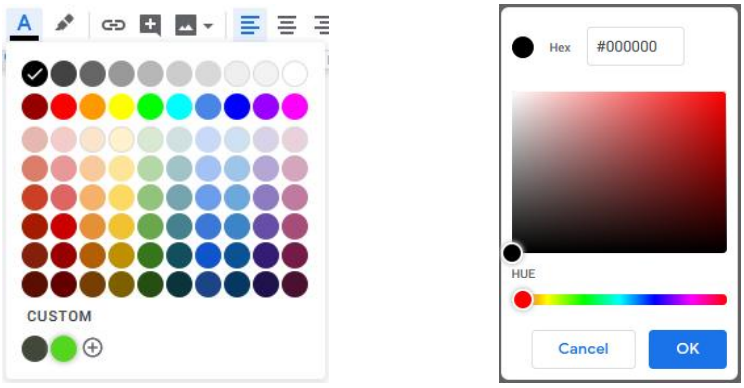
The annotation preview in the *Overview* list needs to be able to identify the context of the annotations. If a text passage is underlined, for instance, the rendering must show not only the line but also the text above. The preview of a vertical line beside a text block needs to be included as a whole. For this prototype, we have covered only the cited cases, but in future iterations, this functionality may be extended.

If the user taps on an annotation in the *Overview*, the workspace pans until the annotation is visible. Whenever possible, the document is panned vertically only. Only in cases when the annotation lies horizontally outside the visible area is panning along the y-axis performed as well.

Finally, the last zoom and workspace position are saved and restored for each document when the document is reopened, hence, enabling the user to immediately start working.

5.4 Picking the Colors

A good color picker should have the ability to make picking colors easy and at the same time allow the user to pick their desired color. While too many color options can overwhelm a user, it might be a frustrating experience if the required color is not available. Many apps implement a color wheel that also has another disadvantage: it makes picking the same color twice a challenging task. During our research for a perfect color picker we found an interesting thread on *StackExchange* about the "Most User Friendly Color Picker" [pla]. The most popular answer noted that the primary goal of a color picker is to serve the user's basic needs and provide a quick color pick by initially limiting the available colors. Correspondingly, during our review, we found several apps that initially provide a limited set of basic colors. If these colors are not satisfactory, they offer an extended color picker with more choices. This leads to the secondary goal: Providing the possibility of picking a custom color, implementing a color wheel or another more complex color picker.



(a) Limited set of Colors for Basic use (b) Color Picker for Custom Colors

Figure 5.9: Google Docs Color Picker[27]

The *StackExchange* thread states that a good selection of colors should consist of bright colors, grayscale and a value range of the bright colors ,and this is demonstrated with the color picker implemented by *Google Docs* shown as Figure 5.9.

Google provides a basic color picker showing ten grayscale colors ranging from black to white and ten bright colors including the most important colors, with red, yellow, green, and blue among them. Furthermore, it provides six different variations for each of these ten bright colors (5.9a). In the bottom area of the color picker a space is reserved for the most recently selected custom colors and a button that displays the custom color picker (Figure 5.9b).

The author of the StackExchange answer further suggests a set of alternative bright colors (Figure 5.10), which are a little more unobtrusive than the basic colors provided

[27] https://docs.google.com

by *Google Docs*.

Figure 5.10: Less Obtrusive set of Bright Colors

Our user research in Chapter 2 and Chapter 4.2 demonstrated that a wide variety of colors is not that important for annotating documents and, therefore, we have limited the available colors to the alternative bright colors suggested in the *StackExchange* thread, including a set of five grayscale colors. Since the colors for our text highlighter implementation need to fulfill the different requirements described in Chapter 6.5.9, they have been selected accordingly. For our first prototype app, we have not implemented a color picker for custom colors, instead we have followed the principle of other note-taking apps, such as *Apple PencilKit*.

CHAPTER 6

Implementing the Application

6.1 The Application Architecture

The following three different architecture patterns are the most common: *Model-View-Controller* (MVC), *Model-View-Presenter*, and *Model-View-ViewModel* (MVVM) [Sin17]. We used MVVM for the app because it provides a better separation of concerns than the MVC pattern [Wikg] suggested by Apple.

6.1.1 Model-View-Controller

Apple recommends the MVC pattern for iOS apps as shown in Figure 6.1.

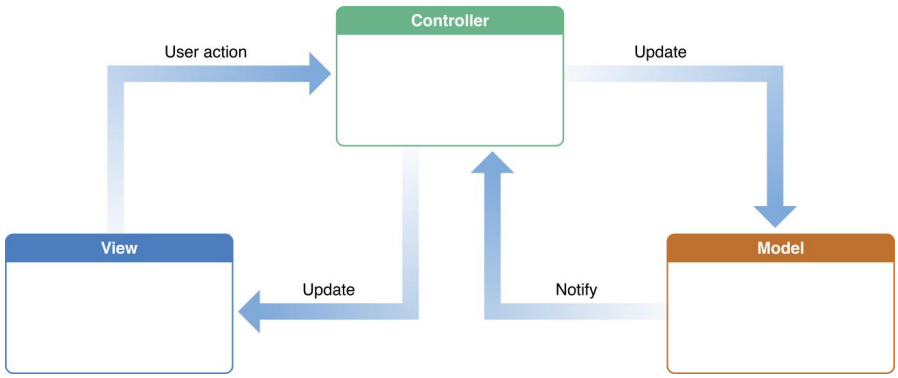


Figure 6.1: Model-View-Controller as suggested by Apple[28]

While MVC sounds excellent in theory, the UIViewController class, as the name already indicates, combines view and controller functionalities (Figure 6.2). This often results in

[28] https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

complex and bloated source code. Even worse, unit testing becomes very challenging, as the UIViewController's view and lifecycle need to be simulated [Hua]. The MVVM pattern [Wikc] addresses these issues.

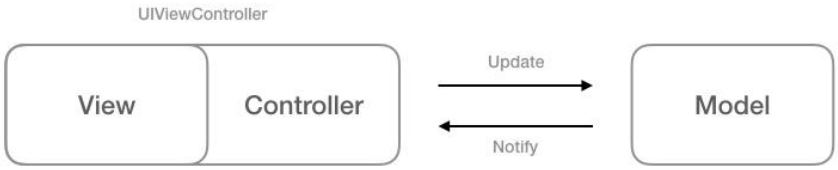


Figure 6.2: MVC with UIViewController in reality[29]

6.1.2 Model-View-ViewModel

Figure 6.3 demonstrates that the MVVM pattern consists of three different components: the *view*, the *model* and the *viewmodel*.

- The *view* contains only visual elements and handles layout changes, animations, and other UI-related tasks.
- The *model* provides access to different services, such as the data layer or network service.
- The *viewmodel* is the connecting link between the view and the model. It contains the business logic and fetches data from the model and forwards data through a technique called "binding" [Wikc].

Figure 6.3: Model-View-ViewModel[39]

The UIViewController implementations cover the tasks of the view component. They create the layout and provide functions for navigation and popup dialogs. Each implementation has a corresponding viewmodel class that handles the business logic. The view

[29] https://medium.com/flawless-app-stories/how-to-use-a-model-view-viewmodel-architecture-for-ios-46963c67be1b

subscribes to the data provided by the viewmodel using the observer pattern [Rea] contributed by ReactiveX[30] and the rxswift[31] library. Each viewmodel maintains references to the required service implementations, that is, the model. The first prototype implementation requires three different services: one to access the database, another to interact with the file system and a third for storing configuration data in the UserDefaults [Appq].

6.2 Services

6.2.1 Data Repository

The Data Repository Service initializes the GRDB[32] database and provides access to the different entities. It stores document information, project information, annotations, strokes and the collected information for each stroke point.

Creating a table entry for each point of a stroke leads to an enormous number of database entries, which eventually results in a performance bottleneck, especially when frequent writing and reading operations are necessary. Different approaches can be taken to address this issue. To reduce the database load, the data is read only once when the document is opened and held in an in-memory structure until the document is closed again. If the in-memory structure is updated, the changes are forwarded to the database. Two types of database operations are possible, which can be resource intensive due to the number of entities involved:

1. Adding a stroke with points.
2. Deleting a stroke with points.

Adding a stroke with points creates an entry for the stroke itself and an entry for each point that belongs to the stroke. It is necessary to insert these entries immediately, since the database is required to assign an identifier for each object, and these identifiers are used for the in-memory structure as well.

Deleting a stroke, however, has been optimized by only marking the stroke entity as *deleted*, while the point entities are not initially modified. When the document is closed, every stroke entity that is currently marked as *deleted* is removed from the database, including its assigned points. Marking entities as deleted instead of deleting them immediately produces a performance boost in combination with the possibility of undoing and redoing executed actions, which would result in frequent insert and delete operations.

A second approach to reducing the number of database entries is to encode the information held by the points into a string and storing them inside a text field of the stroke entity

[30] `http://reactivex.io`
[31] `https://github.com/ReactiveX/RxSwift`
[32] `https://github.com/groue/GRDB.swift`

Called method	Stored as individual entities	Stored as JSON encoded string
loadAnnotations (990 annotations)	15ms	12ms
loadStrokes (103080 points)	5675ms	7927ms

Both approaches have been implemented with the result that there is no significant performance increase when loading a document. Using JSON-encoded information for each point (location, force, angle, and azimuth) proved to be approximately two seconds slower compared to the first solution. The results were retrieved under the assumption that the JSON-encoded string had to be decoded immediately.

6.2.2 File Repository

Providing access to the imported documents is the File Repository's primary task. All the imported PDF documents are stored in the app document folder. Methods for storing and retrieving thumbnails were also implemented. Thumbnails are written to the app's cache folder.

6.2.3 Preference Repository

The Preference Repository stores the user's individual configuration and UI preferences. It keeps track of document independent settings, for instance, which pencil is currently selected, which color is chosen and whether the *Overview* (described in Section 5.3.5) is collapsed or expanded. This information is used to ensure that the app's UI is retrieved in the same state as it was left.

6.3 Least Recently Used Image Cache

List and grid views are commonly used to display a large number of items. If the cells need to display individual images, the consumed memory can increase and, consequently, the performance level decreases. In the worst case, memory can run out and the system is forced to kill the app. Another problem is the decrease in performance while scrolling. List views are optimized to destroy the cell's content but not its instance once a cell moves out of the visible area. These instances are reused for newly appearing cells to avoid multiple memory allocation and deallocation operations. For those cells holding images, this implies that each image needs to be reloaded. Depending upon the source, this can take a conspicuous amount of time and be visible to the user. If the loading is not performed in a background thread, it can even block the UI and have a negative impact upon the UX.

To avoid these scenarios, two measures have been applied. First, images have to be loaded asynchronously in a background thread. Second, images are cached using a *Least*

Recently Used Cache (LRU cache). The implementation of the LRU cache is based upon the tutorial by Marco Santa [San].

The principle of an LRU cache is simple. Data, in this case image data, are added to the map and kept in the memory. Once a previously defined memory capacity is exceeded, the LRU images are removed until the limit is reached again. The LRU implementation is used for the thumbnails of the document organizer and the annotation overview of the workspace.

6.4 Implementing the Document Organizer

The implementation of the Document Organizer is straight forward and offers no challenges. Its grid view is built using UICollectionView with a UICollectionViewFlowLayout. The flow layout ensures that the grid items are aligned properly from left to right, row by row.

The cell's layout is defined in the Thumbnail CollectionViewCell class. It contains space for an UIImageView containing the thumbnail, a UILabel for the item's title and another UITextView in the bottom-right corner to display the modification date. The image of a checkmark overlaps the thumbnail and is hidden by default. If an item is selected, this checkmark becomes visible to indicate the current state. The same class is used to render both the document and project cells.

An instance of UILongPressGestureRecognizer was implemented to trigger the selection mode when a cell item is pressed for longer than one second. While this mode is active, multiple items can be selected and deselected by simply tapping on them. Additionally, a toolbar slides in from the bottom of the screen offering the available actions as described in the prototype in Chapter 5.4.

Each cell's thumbnail is rendered asynchronously in a background task and stored in the app's cache folder. Additionally, loaded thumbnails are cached using the LRU approach as described in Section 6.3 to reduce the file system's reading operations.

6.5 Implementing the Workspace

The *Workspace* implementation posed the main challenge. As described in Chapter 5.3, it integrates document presentation, the *Sidebar* and *Overview*. For this purpose, we have implemented the WorkspaceViewController class. The primary concern was to decide whether to use Core Graphics (CG) or implement an *Apple Metal* [Appm] layer.

6.5.1 Core Graphics vs Metal

With regard to implementing graphic routines, there were two possibilities: The quick but much more complicated low-level Metal API or the slow CG. Core Graphics is based on the Quartz drawing engine and provides low-level 2D rendering features, which

are sufficient for most cases. It can perform path-based drawing, transformations, color management, off-screen rendering, patterns, gradients and shadings, image data management, image creation, and image masking, as well as PDF document creation, display, and parsing [Appi]. However, CG has the disadvantage that it utilizes the central processing unit (CPU) rather than the GPU. Consequently, for computation-intensive drawing, Core Graphics will reach its limits very quickly.

Metal is similar to OpenGL but optimized to extract every possible fragment of performance from Apple hardware. It provides near-direct access to the GPU. Despite all its benefits, however, Metal is much more complicated to implement than CG.

For the first prototype, we have chosen CG rather than Metal, since there are many tutorials available on how to implement a performant drawing canvas. The inspirational implementation was presented by Apple itself [Appl]. At a first glance, the drawing performance seemed sufficient for our purpose. During the implementation, however, we noticed certain bottlenecks, especially when drawing longer lines. At the time of writing this thesis, Apple has released a video presenting the new PencilKit stating that Metal needs to be used for the lowest input lag.

6.5.2 Workspace Requirements

The *Workspace* needs to fulfill several requirements. First, it has to be able to render PDF documents and our annotations. Since large documents have a large memory footprint, the *Workspace* is required to have mechanisms that maintain a low memory consumption.

A second necessity is to be able to scroll over the whole content to enable zooming between 50% and 200% of its size. Smooth and lag-free scrolling is crucial to a good UX, due to the fact that the slightest lag is noticeable. It is mandatory for the scrolling gestures to meet the user's expectations and implement behavior such as bouncing if the panning exceeds the ScrollView's limits.

The most critical part, however, is the canvas that is capable of handling the Apple Pencil's input. This input needs to offer low latency and feel as natural as possible, just like a real pen or highlighter. The pencil implementation will be described at a later point.

6.5.3 Rendering PDF Documents and Annotations

Both the PDF document and our annotations are rendered inside a custom UIView implementation, the *DocumentView*, which is backed by a CATiledLayer. This type of layer is capable of breaking down large images into smaller tiles and multiple levels

of detail, rendering them asynchronously in a background thread [Appe]. It keeps the memory footprint low, even for large PDF documents and provides smooth rendering while panning and zooming and changing the detail level if necessary to avoid pixellation. Unfortunately, it also has limitations and does not fulfill all the requirements. There are no callbacks to indicate when all the tiles have been fully drawn, for instance. The

second issue is more critical for us. It is not possible to pre-render its content and update the CATiledLayer at once. Unfortunately, this would be necessary to avoid blinking and graphics glitches during undo and redo operations. For now, these issues can be set aside, but in future iterations of the app we will need to find a solution for them.

Prior to our implementation, as mentioned above, we experimented with a different approach. We tried placing the document and each annotation in separate views. The document itself was rendered inside a view backed by its own CATiledLayer, but each annotation was placed in its own custom UIView instance. All these views were added as subviews of a UIScrollView instance. While this approach worked quite well, combining and separating different annotations has proven to be quite cumbersome. Another caveat was that the memory footprint increased with the number of annotations added to the document. The significant advantage of this approach is that the document's CATiledLayer does not have to be redrawn when a new annotation is added.

6.5.4 Panning and Zooming

Providing excellent panning and zooming is essential for a good UX. At first, we implemented a custom ScrollView, supporting the infinite content size and the vertical scroll lock, as described in Chapter 5.3. However, the custom deceleration and bouncing behaviors, which we implemented based on different articles such as *Recreating Apple's Rubber Band Effect in Swift* [Bar15] and *UIScrollView's Inertia, Bouncing and Rubber-Banding with UIKit Dynamics* [Hol14], did not meet our high expectations. While the algorithms were sound, our main concern was that interrupting the deceleration animation, by touching down a finger, subsequently led to graphical glitches and content "jumps." These were issues that we were not capable of solving. As a consequence, the resulting *WorkspaceScrollView* extends UIScrollView and inherits all its behaviors, including zooming and panning gestures. Zooming is limited to between 50% and 200% of the original document size. The zoom gesture can be combined with the two-finger swipe gesture.

Vertical-Scroll-Lock

For the one-finger swipe gesture, the vertical scroll lock was implemented. If the gesture is initiated, the movement vector is analyzed. If the initial movement is more in line with the y-axis than the x-axis, the horizontal movement is ignored for the rest of the gesture and only the vertical scroll is performed. For the first prototype, the direction remains locked if the vector's x-component is smaller than the vector's y-component.

$$\begin{pmatrix} d_x \\ d_y \end{pmatrix} => d_x < d_y$$

Infinitely Large Workspace

Every time an annotation is appended, the overall bounds of the document, including its annotations, are calculated. If these bounds change, the size of the workspace is

extended. There are two cases that must be considered. If the bounds increase to the right side or to the bottom, only the ScrollView's content has to be updated. If the bounds' size increases to the left side or to the top, however, the ScrollView's content offset needs to be updated as well. Updating the content offset turned out to be quite a challenge because this change encourages the CATiledLayer to redraw its content, which is performed asynchronously in a background thread. There is no way to force an update synchronously or to render the content and update the layer in a single pass. As a consequence, this results in a few milliseconds of content tearing. Unfortunately, Apple does not provide callback methods for the CATiledLayer that would be triggered after the rendering is complete. This tearing turned out to be especially perceivable while performing undo actions. In future iterations, we may have to experiment with other concepts for an infinite workspace since a possible solution does not seem to exist for the CATiledLayer and UIScrollView.

6.5.5 Drawing Canvas

As previously mentioned, the canvas is a crucial part of the app, as it is required to handle feedback from the Apple Pencil and render the resulting strokes at a low input lag. Our CanvasView implementation is based upon the *SpeedSketch* example from Apple, which proved to have sufficient performance for the prototype. Despite this, we did perceive a rising lag once the strokes reached a certain length.

The CanvasView was added on top of the ScrollView's content rather than on top of the ScrollView itself. This was necessary because forwarding the touch events between the canvas and ScrollView proved to be much easier this way. The only aspect to consider was hiding the canvas during scrolling and zooming and updating its bounds afterwards.

The canvas processes pencil input in the same manner as the *SpeedSketch* example. Once it is complete, the resulting data are converted to the document's coordinate space and stored. Additionally, the input is either assigned to an existing annotation or added to a new annotation.

6.5.6 Grouping Strokes to Annotations

Annotations are nothing more than a logical group of strokes. While every annotation contains one or more strokes, every stroke is assigned to exactly one annotation. Before a stroke can be assigned, its surrounding context is analyzed, and its bounds are extended accordingly, as explained in Section 6.5.7. The extended bounds are primarily used to enable a better annotation overview as described in Section 6.5.12. They are also compared against the bounds of existing annotations. A stroke is added to the first annotation where these bounds overlap. If there are no overlapping annotations, a new annotation is created specifically for this stroke.

6.5.7 Detecting a Stroke's Context

Most of the time, annotations are meant to be associated with the document's content. They comment upon text, underline words, indicate spelling mistakes and more. As described in Chapter 5.3.6, it is important to preview this context together with the annotation. Therefore, the annotation's surroundings must be identified. A simple approach is to identify the bounds for each line of text in the document and compare them against the stroke at the time the stroke is drawn. This approach is rather basic but is already delivering good results, as we distinguish between two different cases:

- Underlining text passages
- A vertical line to the leading or trailing of a paragraph

The PDFKit library provides the necessary functionality to find the selectable lines of text for every page in the document and return their bounds. This information is cached to improve the performance.

To decide whether a stroke underlines a word, the height of each selectable line of text is doubled, as shown in Figure 6.4. If the stroke is within one of these enlarged bounds, then it is considered to underline a word, and the stroke's bounds are extended to include the affected line of text.

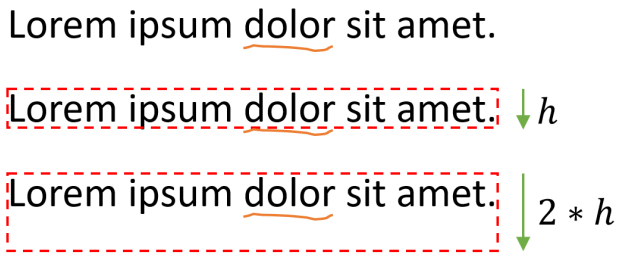


Figure 6.4: Detecting Underlined Text

To detect whether a stroke is intended to be a vertical line at the side of a paragraph, it is first verified whether the stroke vertically intersects a line of text. If this is the case, then the stroke's distance from the leading and trailing borders of the text bounds are used to indicate whether the stroke is at the left or right side of the text. This procedure is visualized in Figure 6.5. If the stroke is too far away from the paragraph, it is not associated with it.

6.5.8 A Natural Pen Behavior

The Apple Pencil transmits location, force, azimuth, and altitude values, which can be used to implement natural pencil behavior. For future conceptions, we will use the term

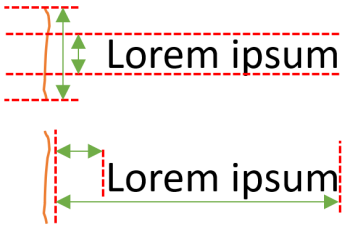


Figure 6.5: Detecting a Vertical Line Next to a Paragraph

"point" instead of location since a location is represented by the CGPoint class. To draw a stroke between two points, we will execute the following steps, as implemented in SpeedSketch:

1. Determine the distance between the two points that define a line segment.
2. Determine the perpendicular force for the two points.
3. Determine the normal unit vectors for each point.
4. Multiply the normal unit vectors with the resulting force value for both points.
5. Draw and fill the line segment as a path.

First, we determine whether the distance between the last drawn point p_1 and the actual point p_2 is large enough to justify its own draw operation, otherwise, we omit it until we find a point p_n that is far enough away from p_1. We define the stroke from $p_1 p_n$ as a line segment. Second, we determine the perpendicular force for each of the two points that represent the line segment by using the formula $force/sin(altitude)$. To reduce the impact of the force on the stroke thickness, we apply the square root to it. If, for some reason, there is no altitude value present, only the force value is returned, which is replaced by 1.0 if no force value is present either. To fine-tune our calculated force value, we multiply it by 2.0 and add an offset of 0.1. These values are taken directly from the SpeedSketch example. Third, we determine the normal unit vectors v_1 and v_n for p_1 and p_n. To calculate these, we need the point p_0 that precedes p_1 and p_{n+1} which succeeds p_n. Fourth, we multiply the normal unit vectors with the calculated force values. Finally, fifth, we stroke and fill the line segment as a path between the following points: $p_1 + v_1$, $p_n + v_n$, $p_n - v_n$ and $p_1 - v_1$.

The resulting pencil behavior feels very natural and its reduced force allows the user to draw fine strokes, which is sufficient for this prototype. For future iterations, we will consider fine-tuning the pressure curve according to the article "The perfect pressure curve for your iPad Lettering" [Hal16].

6.5.9 Highlighting Text Passages

A text highlighter is the second important tool. Most of the apps that we have observed simply draw a transparent color on top of the content. As a consequence, even black text suffers from reduced readability, something which our implementation avoids. Furthermore, we ensure that the highlighted area does not darken or change color when stroking over an already highlighted area with the same color.

For our implementation, we have altered the calligraphy pen of SpeedSketch. In the example, the azimuth angle of the Apple Pencil defines the angle of the stroke and the pencil's force slightly adjusts the width of the line.

Since the highlighter is primarily applied to text passages, we have locked the azimuth at 90°, as the tool is always held at the same angle. Given that the Apple Pencil's tip is pointy and not flat like a regular highlighter or calligraphy pen, this is not an issue and delivers a better feeling when holding it and stroking.

Instead of using transparent colors to draw on top of the content, our solution relies upon *CGBlendMode.multiply* provided by CG [Appf]. The drawn color values are multiplied with the background color, hence, the resulting color is at least as dark as either one of those colors. Given that a color can be described as a three-dimensional vector using floating point values for its red, green, and blue components, each component can assume values between 0.0 and 1.0 indicating the color intensity. A multiplication of two colors is performed by multiplying each component of the first color with the corresponding component of the second color; in terms of color vectors we multiplied two colors using the *Hadamard* product:

$$\begin{pmatrix} R_{source} \\ G_{source} \\ B_{source} \end{pmatrix} \circ \begin{pmatrix} R_{target} \\ G_{target} \\ B_{target} \end{pmatrix} = \begin{pmatrix} R_{source} * R_{target} \\ G_{source} * G_{target} \\ B_{source} * B_{target} \end{pmatrix} = \begin{pmatrix} R_{result} \\ G_{result} \\ B_{result} \end{pmatrix}$$

In our case R_{source} would be the color of the highlighter and R_{target} would be the content color. Having a yellow highlighter to mark black text on a white background leads to the desired result, that is, all the white ($= (1.0, 1.0, 1.0)$) is colored yellow, while the black text remains untouched.

$$\begin{pmatrix} R_{highlighter} \\ G_{highlighter} \\ B_{highlighter} \end{pmatrix} \circ \begin{pmatrix} R_{white} \\ G_{white} \\ B_{white} \end{pmatrix} = \begin{pmatrix} R_{highlighter} * 1.0 \\ G_{highlighter} * 1.0 \\ B_{highlighter} * 1.0 \end{pmatrix} = \begin{pmatrix} R_{highlighter} \\ G_{highlighter} \\ B_{highlighter} \end{pmatrix}$$

$$\begin{pmatrix} R_{highlighter} \\ G_{highlighter} \\ B_{highlighter} \end{pmatrix} \circ \begin{pmatrix} R_{black} \\ G_{black} \\ B_{black} \end{pmatrix} = \begin{pmatrix} R_{highlighter} * 0.0 \\ G_{highlighter} * 0.0 \\ B_{highlighter} * 0.0 \end{pmatrix} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

For the blend mode to suit our needs, two conditions need to be managed:

1. The highlighter color must not change its color if the stroke overlaps itself.

2. The highlighter color must not change if we stroke over an already existing highlighted area.

Solving these conditions can be achieved in different ways, each one being more or less CPU-intensive. The best way would be to avoid intersections of either the currently drawn stroke with itself or with already drawn highlighted areas. This implies that we would have to merge every newly highlighted area by drawing a stroke to the area of the previously drawn stroke and filling it only once. Overlapping stroke segments would be flattened out and become a part of a single area that is being filled. Previously drawn highlighted areas of the same color would be merged to one large shape. Existing areas with a different highlighter color would have to be intersected and the new shape subtracted from them so that no overlapping occurred. To perform these operations, we found two open source libraries that enhance the existing UIBezierPath by increasing performance with caching [Wulc] and adding methods for clipping [Wula]. For our first prototype implementation, however, we have circumvented these shape operations by picking highlighter colors that result in the same color when multiplied by themselves, thus, we have solved the first point.

This leads to the conclusion that we can only pick colors for highlighting that use either 1.0 or 0.0 as values for their components and limit our marker color set to pure yellow, green, red, blue, cyan, and magenta. Multiplying yellow with itself results in the same color.

$$\begin{pmatrix} R_{source} \\ G_{source} \\ B_{source} \end{pmatrix} \circ \begin{pmatrix} R_{target} \\ G_{target} \\ B_{target} \end{pmatrix} = \begin{pmatrix} 1.0 \\ 1.0 \\ 0.0 \end{pmatrix} \circ \begin{pmatrix} 1.0 \\ 1.0 \\ 1.0 \end{pmatrix} = \begin{pmatrix} 1.0 * 1.0 \\ 1.0 * 1.0 \\ 0.0 * 0.0 \end{pmatrix} = \begin{pmatrix} 1.0 \\ 1.0 \\ 0.0 \end{pmatrix}$$

$$\begin{pmatrix} R_{source} \\ G_{source} \\ B_{source} \end{pmatrix} \circ \begin{pmatrix} R_{target} \\ G_{target} \\ B_{target} \end{pmatrix} = \begin{pmatrix} 1.0 \\ 1.0 \\ 0.0 \end{pmatrix} \circ \begin{pmatrix} 1.0 \\ 1.0 \\ 1.0 \end{pmatrix} = \begin{pmatrix} 1.0 * 1.0 \\ 1.0 * 1.0 \\ 0.0 * 0.0 \end{pmatrix} = \begin{pmatrix} 1.0 \\ 1.0 \\ 0.0 \end{pmatrix}$$

Using these colors, which fulfill our requirement, we can ignore intersections with shapes of the same color. Only intersections with shapes of different colors have a negative side effect. As previously mentioned, the multiplying blend effect results in a darker color than both original colors and if two different highlighter colors overlap, it will multiply them and in the worst-case scenario lead to a completely black color.

$$\begin{pmatrix} R_{yellow} \\ G_{yellow} \\ B_{yellow} \end{pmatrix} \circ \begin{pmatrix} R_{blue} \\ G_{blue} \\ B_{blue} \end{pmatrix} = \begin{pmatrix} 1.0 \\ 1.0 \\ 0.0 \end{pmatrix} \circ \begin{pmatrix} 0.0 \\ 0.0 \\ 1.0 \end{pmatrix} = \begin{pmatrix} 1.0 * 0.0 \\ 1.0 * 0.0 \\ 0.0 * 1.0 \end{pmatrix} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix} = \begin{pmatrix} R_{black} \\ G_{black} \\ B_{black} \end{pmatrix}$$

Since this special case can be avoided by the user by simply not painting over existing highlighted areas with different highlighter colors, we are going to ignore it for our first prototype. Nevertheless, we need to consider solving this issue by applying the aforementioned measures in further versions of our app.

6.5.10 Erasing Content

The eraser deletes strokes rather than complete annotations; however, it is not capable of removing parts of a stroke. During the pencil gesture, every touched stroke is highlighted and collected. When the gesture ends, these strokes are deleted in a single step. Consequently, such actions are registered to the *UndoManager* and can be reversed.

6.5.11 Moving Annotations

This tool consists of two parts. First, selecting strokes with the pencil and, second, moving them using a drag gesture. Strokes can be selected by circling them with the pencil. The implementation is identical to the pencil implementation in Section 6.5.8, except that the stroke is rendered as a dashed line with a fixed width. The drawn polygon is closed once the pencil is lifted and intersecting or enclosed strokes are selected. These strokes are rendered into an independent view, which can be moved using a swipe gesture.

For future versions, we will consider extending this functionality to allow the zooming and rotating of the selection as well.

6.5.12 Annotation Overview

The overview is backed by the UITableView class. Each cell displays an individual annotation, including the surrounding document content, but annotations which are enclosed by other annotations are excluded. In view of the fact that the table's width is fixed, some annotations are scaled to fit. For each cell, the content is rendered in a concurrent background thread, and the resulting image is cached using the method described in Section 6.3.

Subsequent updates to this list are debounced by two seconds. Due to the fact that for each new stroke at least one cell has to be refreshed, this step is necessary to significantly reduce the CPU load while drawing strokes. Furthermore, too many updates of the list can be distracting and draw attention from the rest of the UI.

Tapping a cell scrolls the WorkspaceScrollView to the assigned annotation. UIScrollView offers two methods for this purpose:

```
func scrollRectToVisible(_ rect: CGRect, animated: Bool)
func setContentOffset(_ contentOffset: CGPoint, animated: Bool)
```

While the first method would be preferable for our purpose, it did not work as expected. Despite the correct parameters, the UIScrollView did not scroll down far enough to show the whole annotation. For this reason, it was necessary to use the second method and calculate the content offset by hand.

6.6 Implementing a Color Picker

As described in Section 5.4, a color picker with two different sets of colors was implemented. The first set contains 10 bright colors and five grayscale colors ranging from black to white for the pen. The second set of colors contains the bright colors mentioned in Section 6.5.9 and is available for highlighting text passages.

(a) Available Pen Colors

(b) Available Highlighter Colors

Figure 6.6: Implemented Color Palettes

Figure 6.6 shows the expanded color picker for the pen and the highlighter. It is aligned to the bottom area of the CanvasView with the intention of overlapping as little as possible of the document itself, while still maintaining a tidy appearance. The selected color is visualized as a colored button on the bottom left of the CanvasView, shown as the box on the left side in Figure 6.6. Touching the box with a finger or the Apple Pencil expands or collapses the color picker and the available colors become either visible or hidden. By applying the proximity rule mentioned in Chapter 3.2.3, we enable the user to perceive the selectable colors as being associated. They appear as a group compared to the button that shows the currently selected color and has a slight offset.

6.7 Exporting Documents

The primary objective is to export content in such a manner that it maintains compatibility with the PDF standard and can be displayed by other PDF readers. To achieve this, content and markup need to be preserved. Apple's PDFKit library does not support both reading and writing PDF documents. The significant advantage is that CG is based upon the Quartz advanced drawing engine [Appi] which "uses PDF as an internal model for graphics representation" [Sir00].

Nevertheless, it needs to be pointed out that Quartz uses the bottom-left corner as the origin for its coordinate space, rather than the top-left corner. Therefore, all drawing operations have to be mirrored along the y-axis and then translated to compensate for the resulting offset.

Given the fact that the workspace is not restricted to the document's bounds, each page is resized to fit the annotations inside if necessary. This is achieved by enlarging each page and distinguishing between three different cases: The first page, the last page, and the pages in between. If annotations occupy space outside the original bounds, the

affected pages are enlarged horizontally. The first page, however, might have annotations above the page frame. If this is the case, the resulting page is vertically enlarged as well. The same applies for the last page, which can have annotations below its frame. The default case covers all the pages in between the first and last page, which have to be enlarged only horizontally if necessary.

Resizing the page has several drawbacks. Printing enlarged pages, for instance, becomes more difficult or can lead to reduced readability if the page is resized to fit the paper's dimensions. Another caveat is that the original page sizes are lost after the export. In future versions, we need to enhance the export functionality.

Apple's PDFKit offers the PDFAnnotation class to append annotations to the document. There are many subtypes available [Appo], however, only two of them are suitable for the export: The *ink* and *highlight* subtypes.

For this purpose, the aforementioned PDFAnnotation class has been extended for each of the two subtypes and a custom drawing method has been implemented, as suggested in the PDFKit documentation [Appb]. The resulting classes are *PDFInkAnnotation* and *PDFHighlightAnnotation*.

Experiments with this approach have shown that it works well for free hand annotations using the *ink* subtype, but the following issue has been encountered with the *highlight* subtype: The Adobe Acrobat Reader renders a yellow ellipse inside the annotation frame, in addition to our own custom drawing code.

Another problem is posed by the fact that we allow strokes and highlights to be drawn on top of each other, giving them a position along the z-axis and blending them as described in Chapters 6.5.8 and 6.5.9. If strokes and highlights intersect or overlap each other, we group them together into a single annotation object. During the export, we would have to divide them up again and create multiple instances of PDFInkAnnotation and PDFHighlightAnnotation for each type to preserve their positions along the z-axis. In the first prototype, we have solved all these issues by rendering both strokes and highlights inside the PDFInkAnnotation class. With this concept, we can preserve the z-axis and cope with the rendering issues of the *highlight* subtype. In future versions, this behavior will be subject to change.

Another problem posed the fact that we allow strokes and highlights to be drawn on top of each other, giving them a position along the z-axis and blending them as described in chapters 6.5.8 and 6.5.9. If strokes and highlights intersect or overlap each other, we group them together to a single annotation object. During the export, we would have to split them up again and create multiple instances of PDFInkAnnotation and PDFHighlightAnnotation for each type to preserve their position along the z-axis. In the first prototype, we have solved all these issues by rendering both, strokes, and highlights, inside the PDFInkAnnotation class. With this concept, we can preserve the z-axis, and cope with the rendering issues of the *highlight* subtype. In future versions, this behavior is subject to change.

CHAPTER 7

Evaluation and Outlook

7.1 Stretching the Content

During the development and testing phase of the app, the idea of stretching the content emerged. This feature works in the opposite way to squeezing the text, as offered by LiquidText described in Chapter 4.2.3. While testing the app, it often occurred that annotations were written at the side of the document and then arrows were drawn towards text passages. The arrow between the lines impairs the readability of the text nearby. As mentioned in Chapter 1, digital media is not bound to the physical limitations of paper. As a consequence, implementing a feature that inserts blank space by moving lines of text apart is possible. During the implementation of the context-sensitive expansion of the annotation bounds in Chapter 6.5.7, we discovered that the PDF document provides the information necessary to identify lines and their boundaries. This information can consequently be used to implement a gesture that enables the user to stretch the selected lines or paragraphs apart as indicated in Figure 7.1. The challenge when implementing this feature is to distinguish between the regular pinch zoom gesture and the stretch gesture. In a first effort, this distinction can be made using a simple delay. If the user touches on two lines of text longer than a defined amount of time, the touched passages are highlighted, thereby, suggesting that dragging is possible. Once paragraphs have been stretched apart, the resulting space in between can be used for annotations.

The challenge when implementing this feature, is to distinguish between the regular pinch zoom gesture and the stretch gesture. In a first effort, this distinction can be done by a simple delay. If the user touches down on two lines of text longer than a defined amount of time, the touched passages are highlighted, suggesting that dragging is possible.

Once paragraphs have been stretched apart, the resulting space in between can be used to write annotations.

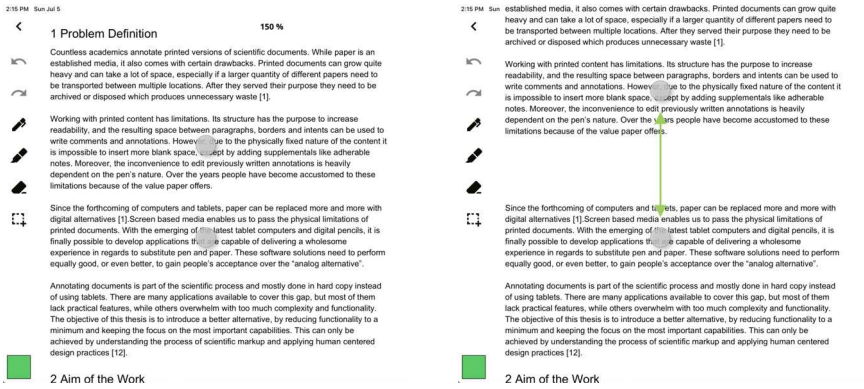


Figure 7.1: Stretching the Content

7.2 Suggestions for Improvement

During the evaluation of the first prototype, a set of possible improvements was identified.

7.2.1 Trash Bin Behavior

One principle of the first concept was to hide unnecessary UI elements to reduce the complexity of the GUI. As a result, the trash bin is only visible if there are deleted elements inside. Consequently, the existence of this functionality is concealed for the person who uses the app at first. To provide awareness that documents can be recovered after they have been deleted, this behavior was changed during the development process. This resulted in the trash bin icon no longer being rendered invisible but instead being shown in its disabled state.

7.2.2 Erasing Parts of the Annotation

Deleting strokes is essential to correct mistakes. Currently the eraser tool is only capable of erasing strokes as a whole. Enabling users to remove parts of a stroke adds a certain level of comfort, such that little mistakes in writings can be corrected without having to re-write the whole word.

7.2.3 Recognize the Annotation's Context

After a stroke has been drawn, its surroundings are analyzed, and the annotation's frame is altered to include the identified context. For the first implementation described in Chapter 6.5.7, this analysis is performed on a rather basic level. Ideas should be collected to enhance this algorithm to become more sophisticated, for instance, to detect groups of annotations.

7.2.4 Enhanced Export Options

The current export function enlarges the PDF page if the annotations exceed the original page bounds. This functionality has to be enhanced to cover additional use cases.

First, the enlarged page is cropped directly to the outermost annotations. Therefore, an additional white border needs to be added to improve the overall appearance of the exported document.

Second, printing becomes more challenging. Since the generated document can have different page sizes within one document, some of them are larger than the original pages. To address this issue, different export options have to be provided. One option would be to shrink the content to fit the original page bounds. A disadvantage of this procedure would be that the content could become unreadable. For this reason, another option would be to enlarge pages to the next standardized page size. If the original document contains pages of DIN A4 size, the enlarged content could be placed into DIN A3 instead. If it does not fit into DIN A3, then DIN A2 is the next possible choice.

On top of these options, an export function using the app's own document format has to be implemented. This is necessary to be able to share documents without losing information. The export must contain the PDF document itself, the annotations and highlights.

7.2.5 Improved Data Model

For the implementation of our app, we chose the modern MVVM architecture. Storing geometric information that contains many points, such as strokes, eventually results in a bottleneck when updating the UI. Parsing thousands of entities when loading a document has been shown to consume a noticeable number of seconds. As long as this loading time is within reason and the user is notified about the ongoing process, this does not pose an issue. However, when adding, editing and removing strokes, it is mandatory that the data model is quick. Performance became an issue especially when implementing the undo and redo functionality. The first approach was to maintain an in-memory model of the data. Updates to this model are subsequently written to the database to increase performance. In future versions, this concept can be further improved.

Another experiment using a different data model was performed and described in Chapter 6.2.1. Instead of storing the points for each stroke in separate entities, all the points of a stroke were encoded into a JSON structure and stored in a single text column of the stroke entity. However, this approach did not deliver the necessary performance boost. The performance gain while querying the entities was counteracted by the time necessary to decode the JSON structure. In future iterations, this approach may be considered again by using a more suitable encoding method and applying further optimizations.

7.2.6 Improved Document Rendering

As described in Section 6.5.1, the rendering performance and pencil lag can be improved significantly by using a Metal canvas rather than CG. Additionally, more complex features, such as the described stretching of paragraphs in Section 7.1, would be possible.

7.2.7 Enhance the Annotation Overview List

In the first prototype implementation, the annotation overview leads a very passive existence. It displays all the annotations in a list view as opposed to other apps, which provide a more sophisticated overview in which excerpted annotations can be arranged as desired. One of the main purposes of the annotation overview is to offer a structured summary of all the annotations, not only within a single document but also across multiple documents. In future versions, we suggest implementing an annotation overview inside the document organizer where all annotations within a project are presented and will allow the user to jump directly to the annotation inside the associated document.

7.2.8 Hyperlinks and References

In the current version of the app, references and hyperlinks are only rendered as text, however, they provide no functionality. In future versions, this behavior should be changed and allow users to open webpages and jump to the references passage in the document.

7.3 Conclusion

There are many apps in the market offering a vast number of different features. Most of them simply allow users to annotate PDF documents by writing notes and highlighting text passages. Other apps provide additional features such as the creation of mind maps. Nevertheless, the PDF document, a digital version of paper, remains static. Only LiquidText introduces an inventive new feature to "squeeze" document passages closer together, thereby, enabling knowledge workers to compare paragraphs in a modern, dynamic way. While LiquidText has taken the first step, many other possibilities exist, for instance, our suggested concept of creating more space for annotations by moving paragraphs apart. However, all these new possibilities come at a cost; the stored documents are no longer compliant with the PDF standard. Another challenge arises when printing these documents, since they are likely to exceed printable dimensions. These challenges could be solved by introducing a new open document standard that supports the aforementioned features. With the recent technological advancements in tablet computers, printing documents on paper will eventually become obsolete.

List of Figures

Bibliography

[Abr14] Luis Abreu. Why and how to avoid hamburger menus. https://lmjabreu.com/post/why-and-how-to-avoid-hamburger-menus/, May 2014. Accessed on 2020-03-28.

[ÅC05] Gabriel Åberg and Jessica Chang. Applying cognitive science research in graphical user interface (gui). 2005.

[Appa] Apple. Adaptivity and layout. https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/adaptivity-and-layout/. Accessed on 2019-10-24.

[Appb] Apple. Adding custom graphics to a pdf. https://developer.apple.com/documentation/pdfkit/adding_custom_graphics_to_a_pdf. Accessed on 2020-03-28.

[Appc] Apple. App store. https://www.apple.com/ios/app-store/. Accessed on 2020-03-28.

[Appd] Apple. Apple pencil. https://www.apple.com/lae/apple-pencil/. Accessed on 2020-03-28.

[Appe] Apple. Catiledlayer. https://developer.apple.com/documentation/quartzcore/catiledlayer. Accessed on 2019-10-24.

[Appf] Apple. Cgblendmode. https://developer.apple.com/documentation/coregraphics/cgblendmode. Accessed on 2019-10-24.

[Appg] Apple. Color. https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/color/. Accessed on 2020-03-28.

[Apph] Apple. Color and contrast. https://developer.apple.com/design/human-interface-guidelines/accessibility/overview/color-and-contrast/. Accessed on 2020-03-28.

[Appi] Apple. Core graphics. https://developer.apple.com/documentation/coregraphics. Accessed on 2019-10-24.

[Appj] Apple. Handling input from apple pencil. https://developer.apple.com/documentation/uikit/pencil_interactions/handling_input_from_apple_pencil. Accessed on 2019-10-02.

[Appk] Apple. Human interface guidelines. https://developer.apple.com/design/human-interface-guidelines. Accessed on 2019-10-24.

[Appl] Apple. Leveraging touch input for drawing apps. https://developer.apple.com/documentation/uikit/touches_presses_and_gestures/leveraging_touch_input_for_drawing_apps. Accessed on 2019-10-24.

[Appm] Apple. Metal - accelerating graphics and much more. https://developer.apple.com/metal/. Accessed on 2020-03-28.

[Appn] Apple. New features available with ipados. https://www.apple.com/ipados/features/. Accessed on 2020-03-28.

[Appo] Apple. Pdfannotationsubtype. https://developer.apple.com/documentation/pdfkit/pdfannotationsubtype. Accessed on 2020-03-28.

[Appp] Apple. Pencilkit. https://developer.apple.com/documentation/pencilkit. Accessed on 2019-10-24.

[Appq] Apple. User defaults. https://developer.apple.com/documentation/foundation/userdefaults. Accessed on 2019-10-24.

[App18] Apptimize. The ultimate guide to the hamburger menu and its alternatives. https://uxplanet.org/the-ultimate-guide-to-the-hamburger-menu-and-its-alternatives-e2da8dc7f1db, May 2018. Accessed on 2020-03-28.

[App19] Apple. Introducing pencilkit. https://developer.apple.com/videos/play/wwdc2019/221/, 2019. Accessed on 2020-03-28.

[Atw19] Cabe Atwell. Burned-out flash crippling tesla model s and x units. https://www.eetasia.com/news/article/Burned-Out-Flash-Crippling-Tesla-Model-S-and-X-Units, Dec 2019. Accessed on 2020-03-28.

[Azh18] Azhar. How to pick colors for your app without a struggle. https://uxdesign.cc/how-to-pick-colors-for-your-app-without-a-struggle-bc46c5e19574, Dec 2018. Accessed on 2020-03-28.

[Bar] Pascal Barry. Deconstructing dieter rams' ten principles for good design - part 1. https://uxdesign.cc/deconstructing-dieter-rams-ten-principles-for-good-design-part-1-1fc5bc2d1e51. Accessed on 2020-03-28.

[Bar15] Victor Baro. Recreating apple's rubber band effect in swift. https://medium.com/thoughts-on-thoughts/recreating-apples-rubber-band-effect-in-swift-dbf981b40f35, May 2015. Accessed on 2020-03-28.

[Boy11] Alan Boyle. Gamers solve molecular puzzle that baffled scientists. https://www.nbcnews.com/sciencemain/gamers-solve-molecular-puzzle-baffled-scientists-6C10402813, Sep 2011. Accessed on 2020-03-28.

[Car] Sinead Carroll. Replacing paper with technology. http://www.artefactmagazine.com/2019/02/25/expensive-form-planet-preservation. Accessed on 2020-03-29.

[CI15] Microsoft Canada Consumer Insights. Attention spans. https://idoc.pub/download/microsoft-attention-spans-research-report-klzzje88qylg, 2015. Accessed on 2019-09-22.

[Com] European Commission. Cookies. https://wikis.ec.europa.eu/display/WEBGUIDE/04.+Cookies. Accessed on 2020-03-28.

[Cun] Andrew Cunningham. Why isn't your old phone getting nougat? there's blame enough to go around. https://arstechnica.com/gadgets/2016/08/why-isnt-your-old-phone-getting-nougat-theres-blame-enough-to-go-around/. Accessed on 2020-03-29.

[DD09] Sabeen Durrani and Qaiser S. Durrani. Applying cognitive psychology to user interfaces. In U. S. Tiwary, Tanveer J. Siddiqui, M. Radhakrishna, and M. D. Tiwari, editors, *Proceedings of the First International Conference on Intelligent Human Computer Interaction*, pages 156–168, New Delhi, 2009. Springer India.

[DJKM17] C.W. De Jong, K. Klemp, and E. Mattie. *Ten Principles for Good Design: Dieter Rams*. Prestel, 2017.

[DLD+13] Rémy Dautriche, Camille Lenoir, Alexandre Demeure, Cedric Gerard, Joëlle Coutaz, and patrick reignier. End-user-development for smart homes: Relevance and challenges. page 6, 01 2013.

[FR09] G.J. Feist and E.L. Rosenberg. *Psychology: Making Connections*. McGraw-Hill Higher Education, 2009.

[Gmb] SodaStream GmbH. Sodastream offizielle website. https://www.sodastream.at/. Accessed on 2020-03-28.

[Hal16] Halfapx. The perfect pressure curve for your ipad lettering. https://halfapx.com/pressure-curve-for-ipadlettering/, Aug 2016. Accessed on 2020-03-28.

[Hat] HatchedLondon. Keep it simple, stupid. https://hatchedlondon.com/keep-it-simple-stupid/. Accessed on 2020-03-28.

[Hol14] Arek Holko. Uiscrollview's inertia, bouncing and rubber-banding with uikit dynamics. https://holko.pl/2014/07/06/inertia-bouncing-rubber-banding-uikit-dynamics/, Jul 2014. Accessed on 2020-03-28.

[Hua] ShihTing Huang. How not to get desperate with mvvm implementation. https://medium.com/flawless-app-stories/how-to-use-a-model-view-viewmodel-architecture-for-ios-46963c67be1b. Accessed on 2019-12-12.

[Ini] Web Accessibility Initiative. Web content accessibility guidelines (wcag). https://www.w3.org/WAI/standards-guidelines/wcag/. Accessed on 2019-12-04.

[Isa11] Walter Isaacson. *Steve Jobs: A Biography*. Thorndike Press, 2011.

[ISO] ISO. Ergonomics of human-system interaction - part 210: Human-centered design for interactive systems. https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-1:v1:en. Accessed on 2019-11-27.

[Lea] Leatherman. Super tool 200. https://www.leatherman.com/super-tool-200-86.html. Accessed on 2019-10-20.

[Luca] LucidChart. Bpmn & bpmn 2.0 tutorial. https://www.lucidchart.com/pages/bpmn-bpmn-2.0-tutorial. Accessed on 2020-02-15.

[Lucb] LucidChart. Diagrams for dummies: A bpmn tutorial. https://www.lucidchart.com/blog/diagrams-for-dummies-a-BPMN-tutorial. Accessed on 2020-02-15.

[Mae06] J. Maeda. *The Laws of Simplicity*. Simplicity: Design, Technology, Business, Life. MIT Press, 2006.

[MBM07] M.R. Morris, A.J.B. Brush, and B.R. Meyers. Reading revisited: Evaluating the usability of digital display surfaces for active reading tasks. In *Tabletop 2007 - 2nd Annual IEEE International Workshop on Horizontal Interactive Human-Computer Systems*, pages 79–86, 2007.

[NAB79] Raymond Nickerson, Marilyn Adams, and Bolt Beranek. Long-term memory for a common object*1. *Cognitive Psychology - COG PSYCHOL*, 11:287–307, 07 1979.

[Nar19] Tom Nardi. Worn out emmc chips are crippling older teslas? https://hackaday.com/2019/10/17/worn-out-emmc-chips-are-crippling-older-teslas/, Oct 2019. Accessed on 2020-03-28.

[Nat19] Daniel Nations. How many ipads have been sold? https://www.lifewire.com/how-many-ipads-sold-1994296, Sept 2019. Accessed on 2020-03-28.

[Nav] Rod Nave. The rods and cones of the human eye. http://hyperphysics.phy-astr.gsu.edu/hbase/vision/rodcone.html. Accessed on 2019-12-04.

[Nie93] Jakob Nielsen. Response times: The 3 important limits. https://www.nngroup.com/articles/response-times-3-important-limits/, Jan 1993. Accessed on 2020-03-28.

[Nor02] Donald A. Norman. *The Design of Everyday Things*. Basic Books, Inc., New York, NY, USA, 2002.

[Pas11] Konstantinos Paschalidis. https://uxdesign.cc/the-application-of-cogntive-psychology-to-user-interface-design-81599ad7fb55, 2011. Accessed on 2019-10-15.

[pla] plainclothes. Most user friendly color picker. https://ux.stackexchange.com/questions/88055/most-user-friendly-color-picker. Accessed on 2019-11-24.

[Ram] Dieter Rams. Good design. https://www.vitsoe.com/gb/about/good-design. Accessed on 2019-10-20.

[Rea] ReactiveX. Getting started. https://github.com/ReactiveX/RxSwift/blob/master/Documentation/GettingStarted.md. Accessed on 2019-10-20.

[Rin14] Matt Rintoul. User experience is a feeling. https://www.uxmatters.com/mt/archives/2014/10/user-experience-is-a-feeling.php, Oct 2014. Accessed on 2019-10-15.

[RLP+] Virpi Roto, Ming Lee, Kari Pihkala, Brenda Castro, Arnold Vermeeren, Effie Law, Kaisa Väänänen-Vainio-Mattila, Jettie Hoonhout, and Marianna Obrist. All about ux. http://www.allaboutux.org/ux-definitions. Accessed on 2019-10-15.

[Ruf19] Gustavo Henrique Ruffo. Tesla cars have a memory problem that may cost you a lot to repair. https://insideevs.com/news/376037/tesla-mcu-emmc-memory-issue/, Oct 2019. Accessed on 2020-03-28.

[San] Marco Santarossa. How to implement cache lru with swift. https://marcosantadev.com/implement-cache-lru-swift/. Accessed on 2020-03-28.

[SGW11] D.L. Schacter, D. T. Gilbert, and D. M. Wegner. *Psychology (2nd Edition)*. Worth, New York, 2011.

[Sin17] Ankit Sinhal. Mvc, mvp and mvvm design pattern. https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad, Jan 2017. Accessed on 2020-03-28.

[Sir00] John Siracusa. Inside quartz. https://arstechnica.com/gadgets/2000/05/mac-os-x-dp4/4/, May 2000. Accessed on 2020-03-28.

[Sof] TGRMN Software. Bulk rename utility. https://www.bulkrenameutility.co.uk. Accessed on 2020-03-28.

[Sot19] Jorge Soto. Dieter rams: 10 principles of good design in a digital world. https://www.empathy.co/blog/dieter-rams-10-principles-of-good-design-in-a-digital-world/, Sept 2019. Accessed on 2019-12-06.

[Sta] Statista. Number of apps available in leading app stores as of 4th quarter 2019. https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. Accessed on 2019-11-24.

[TE11] Craig S. Tashman and W. Keith Edwards. Liquidtext: A flexible, multitouch environment to support active reading. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page 3285–3294, New York, NY, USA, 2011. Association for Computing Machinery.

[Wal19] Andrew James Walls. Dieter rams 10 principles of good design. https://hackernoon.com/dieter-rams-10-principles-of-good-design-e7790cc983e9, Oct 2019. Accessed on 2019-10-20.

[War16] Stiftung Warentest. Mogelpackung: Toblerone auch in deutschaland mit weniger zacken. https://www.test.de/Mogelpackung-Toblerone-auch-in-Deutschland-mit-weniger-Zacken-5115737-0/, 12 2016. Accessed on 2019-10-20.

[Wika] Wikipedia. ipad (1st generation). https://en.wikipedia.org/wiki/IPad_(1st_generation). Accessed on 2019-11-27.

[Wikb] Wikipedia. iphone (1st generation). https://en.wikipedia.org/wiki/IPhone_(1st_generation). Accessed on 2020-03-29.

[Wikc] Wikipedia. Model–view–viewmodel. https://en.wikipedia.org/wiki/Model-view-viewmodel. Accessed on 2020-03-28.

[Wikd] Wikipedia. Objective-c. https://en.wikipedia.org/wiki/Objective-C. Accessed on 2020-03-28.

[Wike] Wikipedia. Progressive web application. https://en.wikipedia.org/wiki/Progressive_web_application. Accessed on 2020-03-28.

[Wikf] Wikipedia. Rare-earth element. https://en.wikipedia.org/wiki/Rare-earth_element. Accessed on 2020-03-28.

[Wikg] Wikipedia. Separation of concerns. https://en.wikipedia.org/wiki/Separation_of_concerns. Accessed on 2020-03-28.

[Wikh] Wikipedia. Workflow. https://en.wikipedia.org/wiki/Workflow. Accessed on 2020-03-27.

[Wit03] Jason Withrow. Cognitive psychology & ia: From theory to practice. http://boxesandarrows.com/cognitive-psychology-ia-from-theory-to-practice/, Aug 2003. Accessed on 2019-12-04.

[Wula] Adam Wulf. Clippingbezier. https://github.com/adamwulf/ClippingBezier. Accessed on 2019-11-21.

[Wulb] Adam Wulf. Loose leaf. https://github.com/adamwulf/loose-leaf. Accessed on 2020-03-28.

[Wulc] Adam Wulf. Performancebezier. https://github.com/adamwulf/PerformanceBezier. Accessed on 2019-11-21.