Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Human-Centered Computing (HCC)

How does an editor for dynamic resources for users with different levels of expertise look like and how can it be conceptualized and implemented within the constraints of an exisiting ecosystem?

Matthias Kind

Matrikelnummer: <IhreMatrikelnummer>

matthias.kind@fu-berlin.de

Betreuer: Florian Berger

Erstgutachterin: Prof. Dr. Claudia Müller-Birn

Zweitgutachter: Prof. Dr. Lutz Prechelt

Berlin, TODO

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den January 8, 2023

Matthias Kind

1 Abstract

In recent years, the shift from print to digital publishing channels has increased the need for tools that allow publishers to quickly build and configure apps and websites.

The goal of this bachelor thesis is to conceptualize, plan, and implement an UI editor for apps and websites used by magazine and news publishers, particularly in Germany and the UK. The editor was built for a properitary web framework called "Purple Experience" and was developed within the constraints of an existing software ecosystem, which posed challenges and limitations on the design and implementation of the tool.
To gain insights into the needs and workflows of the groups of users who will be using the editor, a variety of HCI methods were applied during the user research phase, including moderated observations and interviews.

The outcome of this research is useful as guidance for future software development projects for internal tools at companies, or in environments where constraints exist but a user base is already in place to provide valuable input and feedback. As a result of the user research phase, an interactive prototype was built using modern web technologies and was deployed to a controlled group of test users. This allowed for quick feedback and fast iteration until the tool was ready for a broader audience.

TODO reformulate

Learned: - if existing user base (manageable number): qualtitative research methods seem to provide more value / deeper feedback than quantitative
- if existing products: use tracking etc. to figure out who already uses tools how, as a base line for comparison
- use vizualizations and other methods to sort ideas by multiple dimensions (user value, impl. effort) -> easier to reason what to do next & to convince other stakeholders

2 Zusammenfassung

TODO: translate english abstract when finalized <Hier sollten Sie eine kurze, aussagekräftige Zusammenfassung (ca. eine Seite) Ihrer Arbeit geben, welche das Thema der Arbeit, die wichtigsten Inhalte, die Arbeitsergebnisse und die Bewertung der Ergebnisse umfasst.>

List of Figures

List of Tables

Vorwort

Allgemeine Hinweise zur Erstellung einer Abschlussarbeit

- Beachten Sie, dass diese Vorlage für einen zweiseitigen Ausdruck angelegt wurde.
- Über die Papierqualität können Sie entscheiden, aber wir empfehlen aber Seiten mit wichtigen, farbigen Grafiken auch in Farbe auszudrucken und dabei ein höherwertiges Papier zu verwenden.
- Bitte stimmen Sie mit dem Betreuer Ihrer Arbeit auch den Zweitgutachter ab. Die Anfrage des Zweitgutachters erfolgt von Ihnen. Es ist an dieser Stelle sinnvoll, die Anfrage mit einer kurzen Zusammenfassung der Arbeit zu stellen.
- Bitte beachten Sie, dass Sie Ihre Abschlussarbeit mit einer Klebebindung versehen, eine Ringbindung ist nicht erwünscht.

3 Introduction

3.1 Topic and context

In the evergrowing world of software development, many companies are now in the situation where they maintain a large software ecosystem and have complex dependencies, but still want to improve their systems by developing new components and tools. This poses the challenge of improving the software from aspects like user expeirence, scalability and maintainability while beeing restricted by the ecosystem.
Greenfield development[1], which is implicitly used in the majority of books about HCI, assumes no pre-exisiting constraints or limitations, or at least not in the extend they are commonly found in todays software enterprise. However, applying HCI methods for user research and user experience-focused design in brownfield development, where many choices are already made, must be approached differently.

In addition, the three major factors in HCI (Usability, Accessibility and Time-on-task, [2, pp. 38-40]) are often neglected due to tight deadlines and limited resources, leading to premature releases and unstable software.

My goal was to demonstrate how HCI principles and methods can be applied in a brownfield project, using a real-world case study as an examle. By having an exsisting user base which works with exisiting tools, it is possible to evaluate what "real users" need and make more informed design decisions.

The UI Editor, which I will describe more in detail in 5, is a tool relying on may external systems and is limited by the file system structure and configuration schemas imposed by the Purple Experience Franework. The editor was developed for magazine and news publishers in Germany (DACH) and the UK, and is intended to facilitate the editing of dynamic resources by users with variying leves of expertise.

3.2 Procedure for the research and implementation

While writing the software and thesis, I followed the an software design process described in [2, p. 104]. There, the process is divided into three phases: "Design Thinking", "Lean UX", and "Agile" (development and delivery).

For this case study, the abstract plan for the process looks as following:

[1]Greenfield- and brownfield development refer to software development concepts, where Greenfield projects start in a new environment and don't have legacy code, while brownfield projects are about upgrading or redeveloping software in an existing environment. [5]

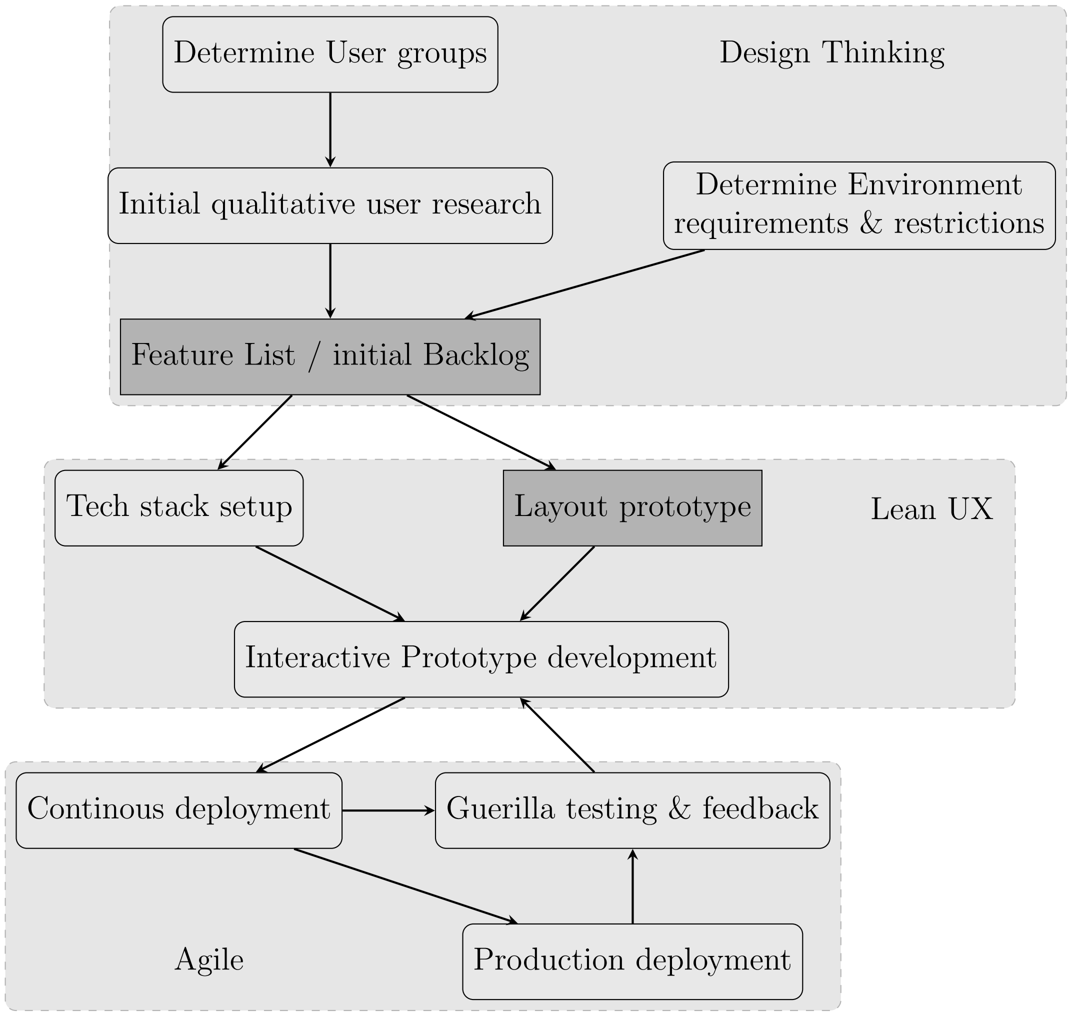

Determine User groups
Design Thinking
Initial qualitative user research
Determine Environment requirements & restrictions
Feature List / initial Backlog
Tech stack setup
Layout prototype
Lean UX
Interactive Prototype development
Continous deployment
Guerilla testing & feedback
Agile
Production deployment

Figure 3.1: Development Process for the UI Editor

4 Kapitel

- Abhängig vom Ziel der Arbeit und dem verwendeten Forschungsdesign unterscheidet sich dieser Hauptteil der Arbeit erheblich.

- Eine sehr allgemeine Struktur ist die folgende:
  - Hintergrund der Arbeit (Theoretische Einordnung der Arbeit)
    - Hier sollte enthalten sein, welche Anwendungen in diesem Bereich bereits existieren und warum bei diesen ein Defizit besteht.
    - Falls genutzt, sollten hier die entsprechenden Algorithmen erläutert werden.
    - Es sollten die Ziele der Anwendungsentwicklung, d.h. die Anforderungen herausgearbeitet werden. Dabei sollte die bestehende Literatur geeignet integriert werden.
  - Umsetzung (Praktischer Anteil der Arbeit)
    - Zunächst sollte die Softwarearchitektur und die genutzten Anwendungen, APIs etc. erläutert werden. Ebenfalls gehört dazu das Datenbankschema.
    - Es sollten die zentralen Elemente der Software (abhängig von der Aufgabenstellung) beschrieben werden, wie implementierte Algorithmen oder das Oberflächendesign.
    - Zentraler Quellcode sollte entsprechend aufgelistet werden:

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

  - Evaluation (zumeist nur für Masterarbeiten relevant)
    - Jede Software muss auch getestet werden. Dieses Tests werden entweder mit einem vorgegebenen Datensatz erfolgen oder aber die Evaluation erfolgt auf Basis von Experimenten. In diesem Kapitel sollte daher entweder der genutzte Datensatz oder der experimentelle Aufbau beschrieben werden.
  - Ergebnis und Diskussion

∗ Die Ergebnisse der Anwendung werden in diesem Kapitel vorgestellt und anschließend diskutiert. Wenn möglich sollte die Ergebnisse in Relation zu bestehenden Arbeiten in dem Bereich erörtert werden.

5 Theoretical background

Before discussing the design and implementation of the UI editor, it is necessary to provide a brief overview of the applied parts of HCI and the specific context, challenges and Opportunities in which the UI editor was developed.

5.1 HCI

Let me start by concretizing the HCI aspects and why this thesis approaches the area from an not so common standpoint.
Many HCI books (e.g. [6] or [2]) implicitly assume Greenfield development, which ”[...] is in its most distinct form when a new product is created from scratch – a new product or product platform, based on new technology, using new methodology [...]” [1].
While [6, p. 392] for example mentions *Environmental requirements* and *Technical requirements* as part of the requirement discovery, the descriptions of these terms are more focused on physical limitations or user behaviours and technical requirements are mentioned only once and then not discussed further.
In contrast, I set my focus on the development inside a software company and an existing ecosystem, where resource, time and API limitations are present and existing users need to get productive with new software quickly.
Thus, the user research phase started with collecting information about diffrent exisitng users as well as potentially new user groups the editor should target and adapting common HCI methods for qualtitative and quantitative user research like Moderated observations, interviews and questionaires. I approached this phase with an open mind, recognizing that individual methods or applications of them might not be successful or don't provide value in this case study, but that much can still be learned from such experiences. As an example, after evaluation and test runs, I decided against Questionaires, because they showed less information content than other methods in this case study. To align the methods with the projects goal and to select methods worthwhile trying from the large pool of user research methods, I used two systems:
SMART criteria, which is a popular machanism to formulate goals and objectives that are clearly defined and also realistic to be implemented in time. [3]. I'll introduce them more in detail in chapter 7. The second one are ”three major factors that an HCI designer should consider” [2, pp. 37-41] according to Becker, which are

Usability Factor describes the spectrum from ”usable” to ”unusable” software. This is determined through the software design features implemented and how they help the user archieve their tasks in the environment provided.

Accessibility Factor is high when as many users from different backgrounds can use the software in different enviornments. It includes, but is not limited to access for people with physical and other disabilites, as well as the entry hurdle a new user must overcome. While it might look like this factor is not as important as the other two for specialized software mostly used internally, it should not be left out when designing and implementing. For example, even if currently no user with visual impairment is working with the software, it can always happen that an exisitng or new colleague suddenly has to rely on screen readers to continue his work.

Time-On-Task Factor refers to "[...] solutions that use up the appropriate of time to solve a problem." [2, p. 40]. Obviously, to save users as much time as possible, a fast system is required. But it is inevidable that the system, network delay and complexity of computation all have some minimal required time, and users understand that some operations might take time. More important is to reduce the perceived lag of interactions and give users feedback if a task takes longer to complete.

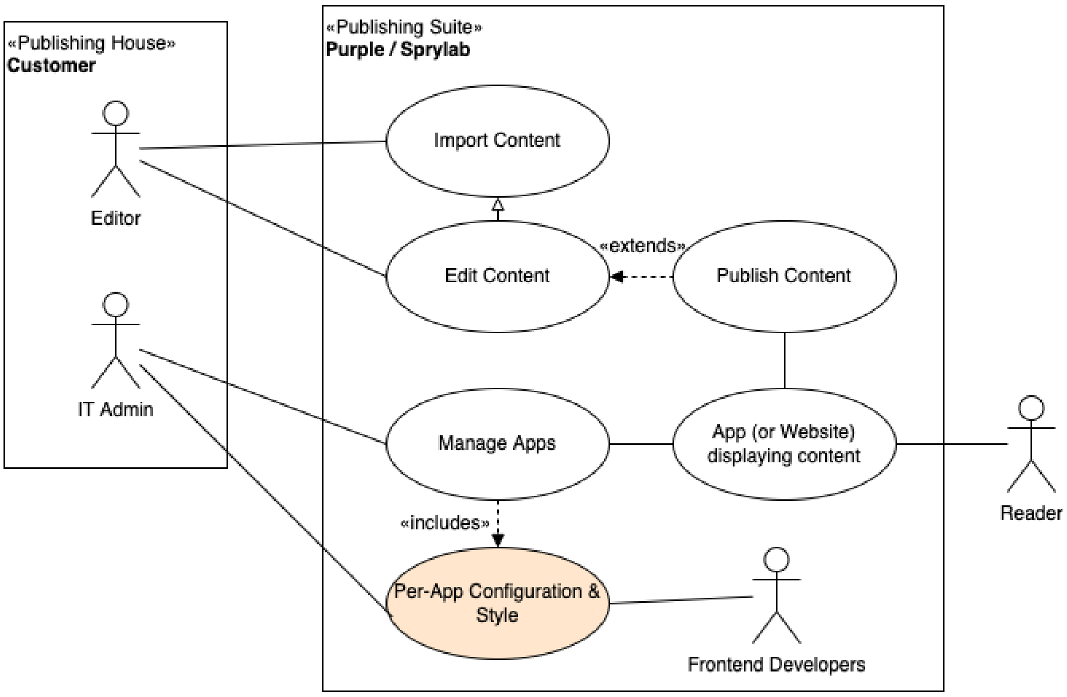
Evaluationg the outcomes of the research is at least as important as the research itself. I experimented with diffrent ways to prepare and process the data to to present it more clearly as well as to consider different factors when discussing how to progress with the prototype.

5.2 Project specific background

To understand the usecase and value of the UI Editor, we first have to declare the fundamentals of the environment the editor will be embedded in. The publishing houses resp. their digital departments (in the following *customer*) purchase the license for an app or website (in the following just *app*, as there is not much difference besides the end medium). Then, they can import content via multiple ways into the system, or the editors write the content directly inside the tools provided as an Software-As-A-Service (SaaS).

The UI editor fits into the Use case "Per App configuration & style", with wich mostly Frontend Developers and Project Managers from Sprylab as well as some external customer's IT admins will interact. The goal is to lower the editing burden as much as possible, so that more of the configuration can be handed off to external customers while also improving usability for the developers of the company.
Now that we have established a rough understanding of the environment and usecase the UI editor will be placed in, I want to explain more about the configuration and styling itself. For that, it is important to understand the Frontend framework Sprylab uses for the delivery to apps and websites. It is called Purple Experience and is a Meta framework build ontop of Angular. The benefit is, that it is completely configurable via JSON files describing the routing, rendering of diffrent components, connecting data sources (an API

Figure 5.1: Use Case diagram showing interactions from publishers, readers and frontend developers with the system

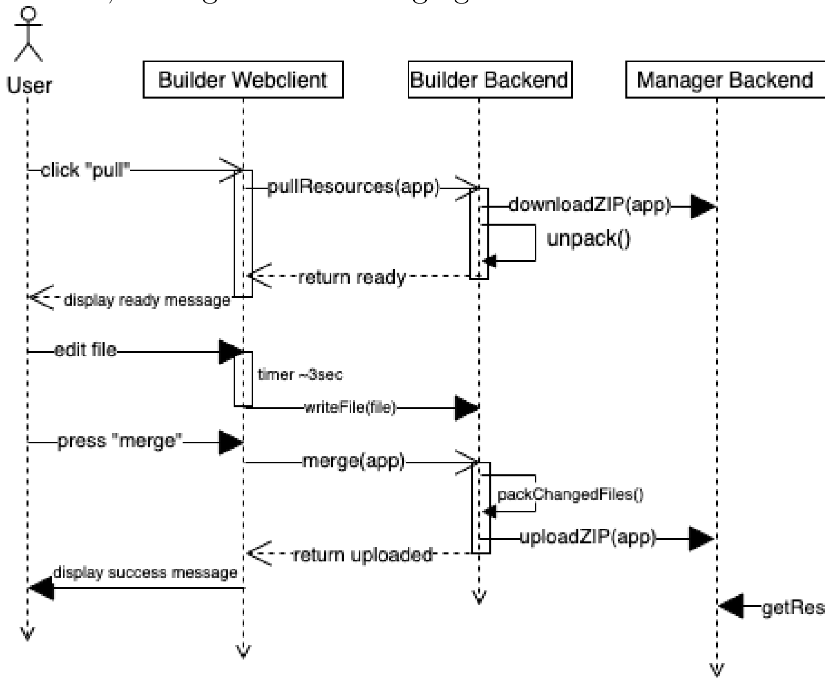


abstraction) with those components, loading assets like images and ads, and styling the whole page with CSS.
These configs and assets are stored on an file system called *dynamic resources.*

Dynamic resources are individually managed and loaded for every app. This way, on mobile phones the endusers download an native core app, which in turn just downloads the dynamic resources and executes the angular app with the configs provided from the resources. Similar, when a end user requests a website, the backend server just looks up the dynamic resources matching this app's Domain and renders the website using that config. This way, all customers can share the same server instance(s), or at least don't require extra build artefacts per app.
In addition, there are preview and live resources for every app, so that changes can be tested before they are released to the end users.
If you have worked with larger, deeply nested JSON files before, you may recall that they get convoluted quite fast. Also, manually handling ZIP files, changing assets and config files, packing everything back in a zip file and hoping one didn't introduce a typo anywhere is an inefficient an at times quite dangerous workflow.
At Sprylab, there exists an tool called "Storefront Editor", which is used as the foundation for this new editor. In the chapter 7 about User Research, I will outline the positive aspects and approaches which I reused for the new editor,

as well show the missing features and features the interview candidates noted as confusing, not working or slowing down their work.

The following UML sequence diagram displays a typical interaction of a user with the editor; pulling the current version, editing a file and merging the



changes. **TODO move to architecture?**

6 Related work

TODO

7 User research and analysis

Cites:

- [8] why companies dont conduct user research

Over ten years after the publication of Tomer Sharon's book ”It's our Research”, the listing of qoutes in the introduction about user research in software companies still feel as relevant as ever.
”Yeah, but this study will delay our launch date.”, ”Yeah, but we can't learn much from only five participants.”, ”Yeah, but research sounds so academic.” [9, p. 4] are only some of the statements that according to Sharon are often heard in software companies when discussing if UX research should be conducted.
The common pressure from different stakeholders often leads to quick implementation of features and workflows without first investing time to figure the user's needs out, which may be faster in the beginning, but can badly impact the user's acceptance of the product due to cumbersome and slow workflows, in the worst case leading to the user not using the product anymore.

To counteract this, it is crucial to conduct and evaluate user research methods, which is what I did for the development of the UI builder.

A starting point for qualitative user research is to define the goals through the help of the SMART criteria, which provide guidelines and formulated goals during research.

For the project, I defined the SMART criteria as following:

- **specfic** - improve the workflow of users modifying dynamic resources for the Purple Experience.

- **measurable** - interviews after testing period concerning working speed, confidence and joy when editing resources, automated user tracking

- **assignable** - research and implementation will mostly be conducted by me, with input from CTO & product owner, connections to external users through customer service team

- **realistic** - new software platform which reacts quicker, prvides more safety regarding errors and is scalable and extensible in the future. Limiting factors are time (as I only have three months for the first phase, including writinh this thesis)

- **time-related** - the new software should have at least the same feature set and be usable by company-internal users until the end of 2022

7.1 Identifying and categorizing users and user groups

In order to effectively design and implement the UI editor, it is crucial to understand the needs and preferences of the various users and user groups who will be using the tool. Therefore, the first step in the user research process was to identify and categorize the different users and user groups who will be using the editor.

In a later chapter (7.5), I'll build concrete Personas for the different user groups utilizing the information gained from the interviews.
Because we already have existing users that work with the previous editors and other tools from the ecosystem, it was relatively easy to collect a list of internal and extneral users, which either I personally knew or I could write a short message asking about if and how they use existing tools and modify dynamic resources. I see that this won't be as easy when dealing with a larger user base or primary external customers, when this first step probably requires more effort to collect a user overview upfront.
With a list of many of the users, I started grouping them to understand the characteristics and needs of each user group, through which can ensure that the UI editor is tailored to their specific requirements and can be used effectively by all users[1].
I derived the follwoing commomn factors from the users, which made the communication and categorization a lot easier.

- **quantitative usage** There were users who relied on the tools for most of their work, while others like the external customers accessed the tool a few times a year.

- **common tasks** I roughly categorized the common tasks into three groups:

  Heavy configuration Mostly internal devs used the tools to build new apps and websites from scratch (or derived from exisiting apps), making many modifications, from structural changes to the seperate views, menus, data sources and more, over styling and translating messages to diffrent languages.

  Moderate configuration Project devs and customer support people copy resources from existing apps and adapt them for new brands, which often includes changing colors and logos, adapting texts or switching authentification flows.

  Small changes External customers often only use the tools to exchange some ads, translations or logos, which affects a small set of files.

- **expertise**

[1]When I refer to "all users", I mean the group of users that are expected to work with the tool. There is an expected technical and domain specific base knowledge that the Editor won't cover

Technical Depending on the area of education and working time in the web development industry, the expertise about web technologies, languages like CSS and JSON and often also intuition differs between users.

Domain- and Platform Specific There is a lot of vocabulary, functionality of the Purple Experience and other systems as well as permutations of configurations that users learn with time.

7.2 Qualitative user resarch

The existing user base enabled me easy access to subjects for qualitative user research methods. Using one or multiple ways of Triangulation [6, p. 264] can strengthen the significance of the research outcome. Limitations of a method or source can be removed by variation of those, resulting in a less distorted picture. Therefore I used metholodical triangulation (using multiple data gathering techniques) as well as triangulation of data (collecting data from different people and different sources). Moderated observations combined with interviews proofed to be a good fit for this case study, as they are interaction driven and the observer can react directly on behaviours / emerging topics and steer the process. This stands is contrast to more passive methods I found like passive observations or user recording and tracking analysis, where the outcome only depends on the prepared question / task and the users behaviour and which can't adapt to changed circuumstances etc. during the application.

The chosen structure for the observations and interviews looks as following:

Introduction (~5min) used to explain the circuumstances and the goal of the session, how we proceed, which data I will collect and how I will evaluate the data afterwards.

Moderated Observation (10-15min) have the observed perform specific tasks in a prepared environment

Semistructured Interview (~15min) ask prepared questions as well as open ones and discuss observation situation

The reason for Interview following the Observation is, that the observer and observed can discuss the situations or issues occured during the Moderated Observation before, which was a good point of entry into the open dialog after the mandatory prepared questions were asked. The other way around, I'd have no way to react to workflow and potential problems the observed encountered during the tasks.
As I had no prior experience conducting these methods with an scientific approach, it was important to me to test the whole process before scheduling all the other meetings. One of our working students agreed to be a test candidate

and we went through the tasks and questions I planned, after which he gave me feedback. Testing the methods and specific questions before conducting them on a broader audience helped finding questiones that were ambiguous or lead to a lot of repetition of already known facts. For the moderated observation tasks it gave me feedback on the difficulty and time they would take for others on average, which tasks needed clearer formulations and which were already good.
In total, I performed the user research on six persons, from which one was an core Purple Experience developer, two were working students from our project department, one was an developer from an different department who had worked with the software some time prior, one Customer Support Manager from our company and one external user from an publisher.

7.2.1 Moderated observation

I prepared a list of six tasks, with the last two beeing optional depending on the time left and the confidence of the user with the platform I perceived during the beginning. That way I could present the same first tasks to every interviewee regardless of their level of knowledge, and present the last two tasks if we had enough time left. Also, it was important that the tasks did not build on each other to prevent the observed person getting stuck because of an earlier mistake. To me, it was more important to see a variety of tasks getting performed than one task beeing executed without errors.
In practice, I prepared an example app on our staging system, noted the link down and downloaded the initial state so that I could easily reset the enviornment after each observation. The six tasks were

- Change english app menu entry ”Newsstand” to ”Home” on all platforms (Web, Android and iOS)
- Change the Advertisement banner target on top of the home page to https://google.com
- Change the exnglish text ”Latest Issues” on the home page to ”Read new Issues”
- Change color of ”Read new Issues” and ”Latest Articles” headers on the home page to the app’s primary color.
- *(Optional)* Add an dropdown on the home between ”Read new Issues” and ”Latest Articles”

    It should show all publications connected to the app

    It should set an URL parameter ”publication” to the id when selected

    Define the reset message as ”All publications”

- *(Optional)* Configure the filter of the ”Latest Articles” list to only show articles from that publication

These were constructed in a way that I anticipated some errors so I could see how users tried to figure out what went wrong and fix them. These cases then also occurred, for example the first task was often only done for one of the three platforms, the ”Latest Issues” text was not found in the translation files or the wrong CSS selector was used to recolor the header elements on the home page, leading to more elements beeing recolored.

The optional tasks were presented to four people, of which three solved them at least one of them successfully, only the core developer solved all six tasks completely. With the consent of the interviewees I recorded their screens during the observation to rewatch specific actions or flows if required.
TODO: noticed workflow patterns that can be improved? like file opening, switching files (quick links & file tabs)

7.2.2 Interview

For the interview, I chose a semistructured interview as the appropriate tool. The interviewer has a list of open and closed questions, through which he can ensure that important topics are covered, while still leaving room for deeper covering of specific points the interview might not anticipate before the interview.

7.3 Quantitaive user research

- Not used survey / questionaire -> lay down reasons why not necessary in that situation
 - Tracking of user behaviours
 - on site using G analytics & squaky.ai (TODO: remove g analytics? ask Peter) - using server logs to understand usage patterns

7.4 Process and vizualize the outcomes of the initial user research phase

7.4.1 2x2 Opportunity Matrix

This two-dimensional vizualization of a set of proposed features prooved helpful when prioritizing tasks with other stakeholders, as it shows the (approximated) cost of implementation as well as the value the feature can have for users.

The matrix I used is a slightly modified adaption from [2, p. 181], replaced the term ”idea originality” on the x-axis with ”Value”.

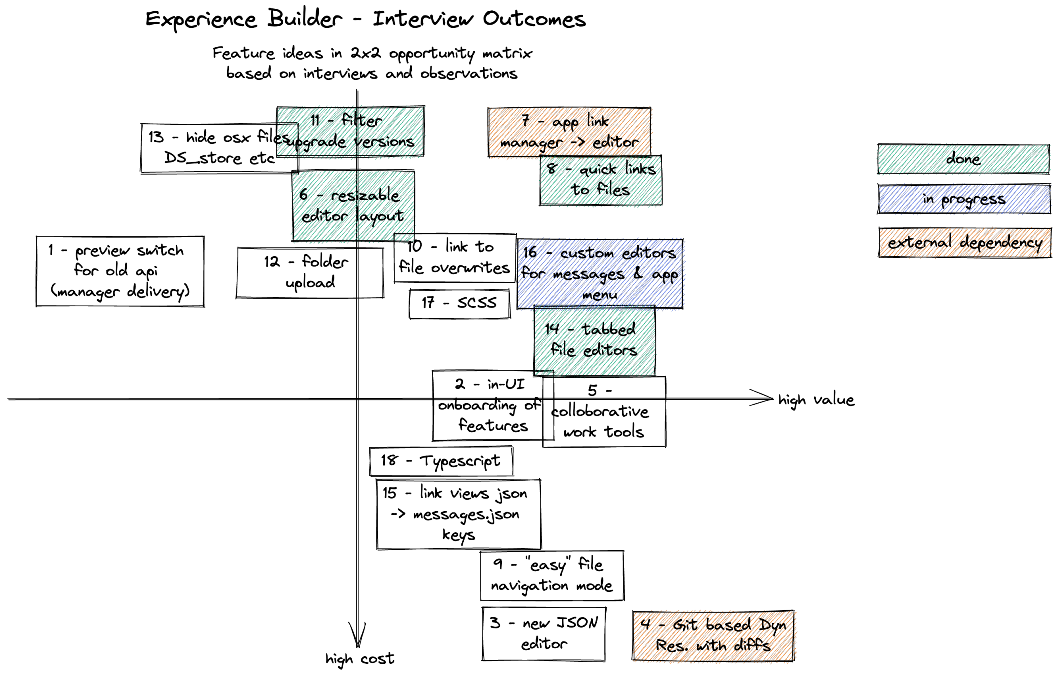

Experience Builder - Interview Outcomes
Feature ideas in 2x2 opportunity matrix based on interviews and observations
13 - hide osx files DS_store etc
11 - filter upgrade versions
7 - app link manager -> editor
8 - quick links to files
6 - resizable editor layout
1 - preview switch for old api (manager delivery)
12 - folder upload
10 - link to file overwrites
16 - custom editors for messages & app menu
17 - scss
14 - tabbed file editors
2 - in-UI onboarding of features
5 - colloborative work tools
high value
18 - Typescript
15 - link views json -> messages.json keys
9 - "easy" file navigation mode
3 - new JSON editor
4 - Git based Dyn Res. with diffs
high cost
done
in progress
external dependency

Figure 7.1: 2x2 Opportunity Matrix during the early phases of development

7.5 Building Personas

Personas are descriptions of fictional users of the product, incorporating assumptions and optinally data for a user group. They aim to give developers and designers more context and depict real potential users, which makes it easier for a developer to empathize with the user. The following three Role-based Personas are derived from 7.1 and the outcomes of the interviews, based on the description of Personas in [6, pp. 403-405]

7.5.1 John - Purple Expeirence Product Developer

Background and Skills

John (34) is a senior Angular Web Developer at Sprylab, working there for two years. He was born in Berlin and lives in Lichterfelde with his wife and mostly works from home. He is passionate about Angular, Typescript and Developer Experience in general, studied Computer Science at the Beuth Hochschule and hosts Angular conferences.

Goals and work with the Editor

- Test newly developed features and the related configurations

- Configure test apps for development and QA purposes
- Support in case Project Developers like <TODO> encounter problems
- John works with the editor multiple times a week

7.5.2 Steffi - Project Developer

Background and Skills

Steffi (23) studies media informatics and works as a working student at Sprylab since a year. This is her first job in the industry and she is learning new things every day. Her skills include writing CSS and understanding modern web technologies, but she still struggles using native and custom debugging tools if something goes wrong.

Goals and work with the Editor

- Configure new apps based on existing templates and adapt them to customer's requirements
- Add new components or change data sources for existing apps
- Add custom HTML pages or Javascript snippets to intergrate external services
- Change styles, color schemas or icons when a customer has a rebranding
- Steffi uses the editor as a primary tool for her work

7.5.3 Karsten - IT department at a publishing house

Background and Skills

Karsten (46) worked in the publishing industry for 20 years, but only during the last years his company, aga magazine publisher, tries to catch up with the digital development and trends. He is still struggling with his role and is thankful for every trick or tool that makes his life managing the digital products easier.

Goals and work with the Editor

- Exchange logos and colors when the magazines he supervies get a redesign
- Add new ads to different views when a new campaign starts
- Manage URLs to external sites when they change
- Karsten uses the Editor once a month on average

8 Prototyping

After collecting the inital user feedback, I started drawing minimal digital "paper" prototypes using Figma to gather vizualizations of the proposed UI layouts. Two ideas emerged from the interviews: a (file-)editor-centric layout and a preview-centric layout.
The editor-centric layout is inspired by modern text editors / IDEs like VS Code (`https://code.visualstudio.com/`), which was mentioned as reference during the interviews multiple times. There, the central pane is the editor for the currently open file, while in the sides additional panes for file management, preview and more can be shown. The familarity especially to developers, who are used to IDE layouts, could help new users adopt patterns to work with the UI they use in other tools as well.

The idea for a preview-centric layout was inspired by popular generic website builders like `https://wix.com` or `https://wordpress.com`, where the user can see the page in an interactive mode, move, rename or place elements, and then has on the side additional panels like one with information & options about the currently selected one.
Ultimately our choice fell to the editor centric layout, the following are some of the reasons for it over the preview-centric one:

- The configuration structure of the Experience framework was not built with preview-based editing in mind. A lot of functionality is hidden inside the components and invisible for the user, often components only appear under specific conditions that are not easily reproducable in the editor environment. Thus, editing in a preview-centric mode could in many situations lead to more confusion by the editors than speed the process up.

- After evaluation of some avaialble libraries and examples, we concluded that building a reliable and usable preview-centric editor is more complicated, and even without the time restricition of my bachelor thesis, I proposed to not go this way, as it was unclear if it even could result in a viable product in reasonable time. For editor-centric UIs, many thirdparty libraries exist, that can be integrated into the UI. Some relevant are Monaco Editor (the VS Code Open source text editor part) for editing generic web related files like CSS and JS with automatic syntax highlighting and error detection, and an JSON Editor for work with json configs where we could provide a schema.

- The userbase consists mostly of tech-affine people who are used to layouts of IDEs, and the old tool also had a similar editor centric layout. As Jakob's Law of the Internet User Experience states (cf. [7] and [10, p. 2]), the user's understanding of a website is directly tied to his/hers mental model of that system. Introducing a unconventional workflow comes with the danger that the user gets confused, makes mistakes or in the worst case doesn't like working with the tool anymore.

9 Implementation and deployment

The phase of implementation and deployment followed an agile development process, where I could deploy changes easily to get fast feedback from users. I structured the chapter into the following parts:

9.1 Software stack

A common point appearing in brownfield software projects, or generally in larger companies and software ecosystems, is the limitation in technologies that should be used for a new project. Else, one may chose the latest and greatest language or framework for that usecase, but maintainability and availability of persons to review and collaborate are important as well. To reduce overhead of learning new languages, tech stacks and keeping the infrastrucuture manageble, often a small set of languages and frameworks is provided by the devops- or infrastructure team. In my case, the most important point was the availability of additional persons with knowledge. The contraints from devops side were not that tight, since the code gets deployed in Kubernetes, a containerized environment anyways, so as long as it can run on a linux VM, it could get deployed.

For this project I decided to use Typescript (`https://www.typescriptlang.org`) on both back- and frontend. The advantages are wide availability of persons in the company who also work with it, a mature ecosystem, and shared type declarations between server and client, reducing the risk of working with incompatabile data types by accident.
The rest of the stack is fairly common in the web developemnt industry too, the frontend uses React JS as a rendering and reactivity framework, with additional libraries for state management, UI Components and API query management on top.
For the backend, I used Node as a Javascript runtime, combined with the most used HTTP server framework for Node, Express JS [4]. On top of express, a framework called TSOA (`https://github.com/lukeautry/tsoa`) is used to improve the developer expeirence. It provides a class based architecture simi-

lar to the well known Spring platform for java, including dependency injection, type safety and automatic API documentation.

Frontend, backend and shared libraries are stored in a monorepo to speed up development and deployments and make refactorings more consistent.

9.2 CI/CD

CI/CD, short for Continous Integration and Continous Deployment, is a core practice of agile software development and provides a log of value to HCI projects. The first part, Continous Integration, means "[...] developers add to a shared repository frequently that integrates their code." [2, p. 81] The integration consists of automatic builds and tests (cf. 9.3) to improve quality of the code and confidence of the developers to publish changes more frequently.

As Sprylab uses and private gitlab, for the Expeirence Builder I set up a Gitlab CICD Pipeline. The stages may change when additional build steps or tests are added, but currently it consists of the following stages: *Build*, *Package* and *Deploy*.
Deploy is only active on the develop and master Branches and contains the code to upload the docker image to the company's staging and production clusters and update the kubernetes deployment to run the latest build. The other two stages are run for every Commit of an Merge request. This has the benefit that as soon as a developer pushes his code to a branch that has a Merge Request open, he and all others can see if the latest commit could be merged safely or if there is more work to do. The *Build* stage installs the dependencies, builds all packages in the monorepo and executes unit- and e2e-tests afterwards. If any of these steps fail, the Merge request can't be merged until this issue is resolved.

Of course the benefit of CI/CD depends on the setup of the build chain, amount, coverage and quality of tests and other factors. To further improve code quality, we integrated Sonarqube into the repository on gitlab (`https://www.sonarqube.org`), which is a service that automatically scans the code and reports code quality, security issues and technical debt added in a commit.

Having a fast and reliable CI/CD process early on during prototyping and development prooved very valuable, as I could quickly react to user's input on new features or bugs, implement and deploy them quickly on a staging system and get feedback on the new behaviour in less than 15 minutes. Through the seperation of staging and production system, I could deploy quick fixes with more confidence even when beta testers were working on the production system, as I could verify that my changes worked in a production-like environment without interrupting users.

9.3 Automated Testing

As already elaborated, the quality and quantity of automated tests plays a huge role when using CI/CD, as it drastically reduces the time required by Quailty Assurance testers to go through all the edge cases on every change. For the Experience Builder I stuck to two of the most common testing levels: unit tests and End-to-End tests (also known as E2E or System tests).

Unit tests are mostly used to test one "component" in an sandboxed environment. For the server, this meant testing Services and classes or even finer grained; single functions. On the client, we distinguished between UI testing of single react components, and business logic code that is encapsulated in classes or Javascript modules.

They also are helpful during development to test software patterns before dogin large refactorings and to do test-driven development (TDD), where the specifications and constraints can be laid out as code with invariants, pre- and postconditions, and then the implementation is performed while continously running the tests again until the don't fail anymore.
Espeically for TDD, but also for the CI/CD Pipelines, the speed of the tests is important. If a single test run takes multiple minutes, the developer is blocked during that time and can't progress on the task, but the duration is also not long enough to start working on another task in the meantime. That's why I tried to integrate a fast test runtime comatible with UI- / browser testing as well as node runtimes for the server code. After evaluating diffrent commonly used frameworks, I settled for Vitest (`https://vitest.dev/`), which fullfills all the requirements, runs tests in parallel, thus reducing the time between saving the change and seeing the test result to often less than a second, and has easy integration mechanisms into our build tools.

While unit tests are a good way to verify encapsulated behaviours, when many components interact with each other, new errors can emerge often as it is quite unrealistic to have all intenral and extenral APIs behaving exactly as expected for every input, and Web based UI applications have unmanagable number of factors that influence the behaviour of UI, network and timings.
E2E tests are supposed to cover a typical user interaction with the service to vaildate the interaction between business logic components, UI and the user itself. I started with using a custom setup of a headless browser[1] (`https://pptr.dev/`) in combination with vitest, but writing and especially debugging the tests prooved slow and error prone.
At an internal training day some colleagues introduced a new E2E test framework called Playwright (`https://playwright.dev/`), which allows recording a test case in a "normal" browser window, it then generates the base code for

[1]Headless browsers are browser instances that don't render the actual content to a user's screen, but run as a CLI application and still execute all Javascript, CSS and HTML.

the test automatically and only needs to be adapted in a few places. After looking through some examples and seeing how it can get integrated into our pipelines, I started porting the exisitng tests to playwright, and after a few hours the tests run on the new framework, now with much better debugging tools and the ability to add new tests much faster.
This can be an example for others, that investing time to investigate new tools and port code to them if they bring value, can improve developer expeirence and thus also speed and confidence.

9.4 Scalability

- REDIS pubsub + observables - code example

9.5 User Testing, Feedback, Beta and Monitoring

- deployments to staging - small test group at start - internal beta - problem: people didn't want to adapt new platform?? - beta flags

9.6 Communication and Documentation

- teams - jira - missing documentation

10 Conclusion and outlook

- Die Zusammenfassung sollte das Ziel der Arbeit und die zentralen Ergebnisse beschreiben. Des Weiteren sollten auch bestehende Probleme bei der Arbeit aufgezählt werden und Vorschläge herausgearbeitet werden, die helfen, diese Probleme zukünftig zu umgehen. Mögliche Erweiterungen für die umgesetzte Anwendung sollten hier auch beschrieben werden.

Bibliography

[1] Johanna Wallén Axehill et al. “From Brownfield to Greenfield Development – Understanding and Managing the Transition”. eng. In: *INCOSE International Symposium* 31.1 (2021), pp. 832–847. ISSN: 2334-5837.

[2] Christopher Reid Becker. *Learn Human-Computer-Interaction.* 2020. ISBN: 978-1-83882-032-9.

[3] Kat Boogaard. “How to write SMART goals”. In: (2021). URL: `https://www.atlassian.com/blog/productivity/how-to-write-smart-goals` (visited on 01/05/2023).

[4] Vano Devium. *Top Node.js Web Frameworks.* URL: `https://github.com/vanodevium/node-framework-stars` (visited on 12/13/2022).

[5] *Greenfield vs Brownfield: Understanding the Software Development Differences.* URL: `https://www.johnadamsit.com/software-development-greenfield-vs-brownfield/` (visited on 01/02/2023).

[6] Jennifer Preece Helen Sharp Yvonne Rogers. *Interaction design - beyond human-computer interaction, 5th Edition.* Wiley, 2019. ISBN: 978-1-119-54725-9.

[7] Jakob Nielsen. “End of Web Design”. In: (2000).

[8] Jim Ross. “Excuses, Excuses! Why Companies Don’t Conduct User Research”. In: (2016). URL: `https://www.uxmatters.com/mt/archives/2016/03/excuses-excuses-why-companies-dont-conduct-user-research.php` (visited on 01/02/2023).

[9] Tomer Sharon. *It’s our reserach: Getting Stakeholder Buy-in for User Experience Research Projects.* Elsevier Inc., 2012. ISBN: 978-0-12-385130-7.

[10] Yon Yablonski. *Laws of UX.* O’Reilly, 2020. ISBN: 978-3-96009-156-1.

Appendix

10.1 Erster Teil Appendix

10.2 Zweiter Teil Appendix