UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Model Based Experimentation on UI Prototypes

Rakshit Bhat

Exposé/Proposal – October 25, 2022.
Software Innovation Campus Paderborn (SICP)

Supervisor: Dr. Enes Yigitbas
Prof. Dr. Gregor Engels

Contents

1 Introduction

This chapter motivates the readers about the topic (see section 1.1), explains the problems faced by the companies during software development (see section 1.2), our research approach (see section 1.3), and finally, our solution approach (see section 1.4).

1.1 Motivation

Over the last decade, software development had a tremendous impact with increasing customer demand and requirements [1], which increases product complexity and ambiguity, significantly impacting software development. Therefore, the developers have come up with different techniques to meet this requirement criteria. Early user feedback from potential customers in the industry is crucial for creating successful software products because of the growing market uncertainties, and consumers' desire to receive integrated solutions to their issues rather than unique software developments [19]. With the increasing complexity of products, it becomes challenging to determine user requirements making it more difficult for developers to assess their opinions. As a result, the developers of these products are biased toward some requirements and can ignore what the user wants. So, the developers must detect the user's needs and requirements to reduce these risks early. Giving users a "partially functioning" system is the most excellent method to determine their requirements and suggestions [6]. This ensures that the developers with high uncertainties in the early product development phase can improve the product by testing the underlying assumptions [3]. Developers can use this feedback to validate the most critical assumptions about the software product. This validation can decide whether to add, remove or update a feature [15]. This process of determining the best fit for the product through user feedback is called experimentation. There has been an increase in interest in the types of experimentation that can take place in product development. Software products have shown the benefits of conducting experiments in many use cases with incremental product improvement [7]. In experimentation, the product designers design different UI variants (e.g., buttons with different colors), and the developer integrates these variants and assigns them to a distinct group of users. As per some evaluation criteria (e.g., more clicks on the button), the variant with better results is deployed for the entire set of users. So, an experiment can be valuable when it improves the software products. Hence, for experiments to be successful, they should offer one or more solutions that will benefit users.

1.2 Problem Statement

The motivation section shows some gaps in software development between the developers and the designers. This section explains the problems and determines their research and solution approach.

Problem 1: Product designers create many UI prototypes, and the developers implement them. To determine the best variant, the developers create experiments with the users [15]. This concrete implementation of designs uses a lot of resources and time for the developers. Therefore, the product designers need to be integrated into the development process so that they would be able to create experiments independent of the developers.

Problem 2: When the product designers develop the prototypes, testing them with many users is difficult as the product is still not developed. Therefore, it is not easy to conclude a "winner" variant with a small amount of data as it is statistically difficult to prove one of the variants outperforms the others [20]. Therefore, it is necessary to develop an idea that the designers can use to determine the best prototype or variant with a small group of users.

Problem 3: Most often, the software application collects data from the experiments. Some data is used in qualitative analysis, while others are in quantitative analysis. Many companies fail to reap the benefits of using both qualitative and quantitative analysis. Similarly, not all the data is used in the analysis phase reducing the software applications to improve based on customer feedback [18]. Therefore, finding a solution that combines qualitative and quantitative data analysis is necessary.

1.3 Research Approach

The process of creating experiments and testing their variants is usually not systematically arranged, creating anomalies, and leading to unsuccessful experiments. Therefore, this section identifies the research question (RQ) and defines an approach to answer the question.

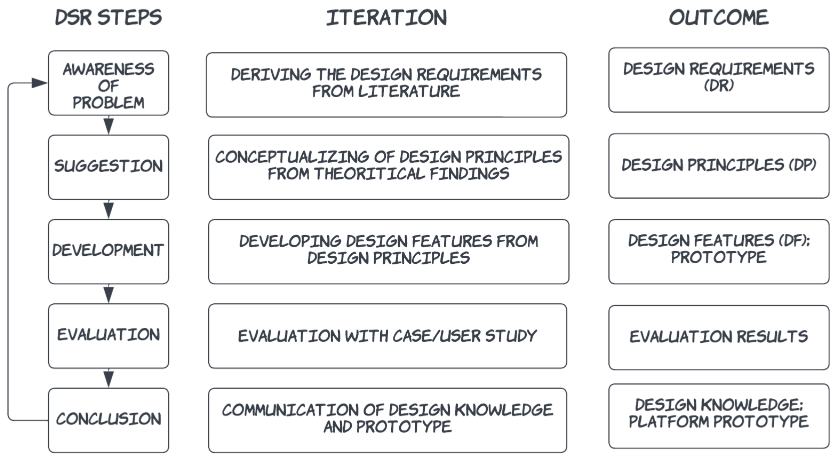

DSR STEPS
ITERATION
OUTCOME
AWARENESS OF PROBLEM
DERIVING THE DESIGN REQUIREMENTS FROM LITERATURE
DESIGN REQUIREMENTS (DR)
SUGGESTION
CONCEPTUALIZING OF DESIGN PRINCIPLES FROM THEORITICAL FINDINGS
DESIGN PRINCIPLES (DP)
DEVELOPMENT
DEVELOPING DESIGN FEATURES FROM DESIGN PRINCIPLES
DESIGN FEATURES (DF); PROTOTYPE
EVALUATION
EVALUATION WITH CASE/USER STUDY
EVALUATION RESULTS
CONCLUSION
COMMUNICATION OF DESIGN KNOWLEDGE AND PROTOTYPE
DESIGN KNOWLEDGE; PLATFORM PROTOTYPE

Figure 1.1: Design Science Research Cycle [14]

RQ: How to develop a platform suitable for product designers to conduct experiments on UI prototypes, increasing its usability and, simultaneously, independent of developers?

We will conduct a design science research (DSR) study to answer our research question and obtain abstract design knowledge and an implementation tool. From the abstracted knowledge, we will obtain some Design Principles (DPs) defined for the whole process of experimentation [14]. In this design, the product designers will iteratively validate their prototypes with the users (or the crowds). Here, DPs capture and codify that knowledge by focussing on the implementer, the aim, the user, the context, the mechanism, the enactors, and the rationale [9]. The DPs explain the design information that develops features for software applications. We propose to use the variation of the cycle of Kuechler and Vaishnavi [14] consisting of five iteratively conducted steps (see figure 1.1). Therefore through the use of DSR, a group of issues is resolved by concentrating on a single issue and abstracting the consequences of the resolution.

1.4 Solution Approach

To solve the problems mentioned above, the designers should be able to create UI prototypes and experiments on their own on a set of users. Since we do not have a large set of users for testing the prototypes, we use supervised task-based usability testing [11]. The fundamental principle of task-based usability testing is to have the users attempt to use the prototypes to do certain activities or tasks (e.g., Locate a movie M1) and get feedback (e.g., the time required for the task to be completed by the user). We propose to use `Low-code` or `No-Code` approach to achieve this. This approach helps to have a UI for the designers to understand, develop, and create experiments and tasks with the software prototypes [12]. So, the designers would be able to create the UI prototypes and their variants, assign them to the users in an experiment, get feedback from the users and decide on the best prototype. At the same time, the low-code has become more accessible for Model-driven development [5]. Therefore, we plan to create models for the UI prototypes and have the feasibility for creating experiments and tasks. Because of using the models, it is easier to store the prototypes in the database and conduct experiments with the users.

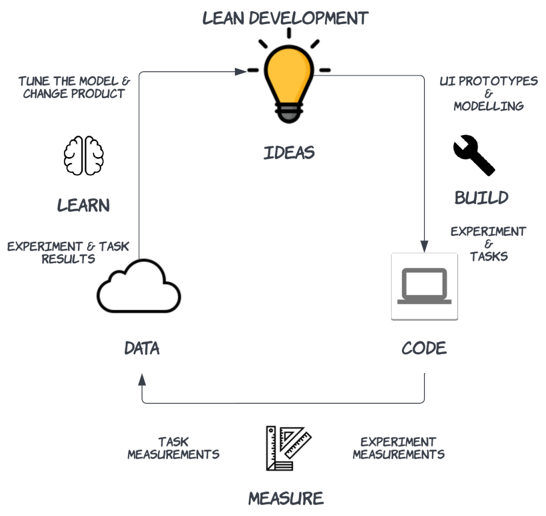


Figure 1.2: LEAN Development technique

In our solution, we use the LEAN development technique (see figure 1.2) for development as it is used to develop customers friendly products [10]. Using LEAN, the company creates a Minimum Viable Product (MVP) throughout development, tests it with potential customers, and leverages their input to make incremental changes. While this technique can be used for every product, there are also approaches specific to software products. LEAN development technique can be divided into a Build, Measure, and Learn cycle. In the `(1) Build` phase, we plan to create the `UI Prototypes`, `Models`, `Experiments`, and `Tasks` for the users. In the `(2) Measure` phase, we plan to assign the Experiments and Tasks to the users and measure the `Task and the Experiment measurements` and perform some analysis on the data received. And finally, in the `(3) Learn` phase, we display the `Analyses results`, `Tune` our models to decide the better variant among the others, and `Modify` the prototype. As per the figure 1.2, we complete one cycle of iteration and start a new one with the updated prototype. (See Chapter 3 for details).

2 Related Work

This section considers some existing frameworks and tools and compares them to our solution approach. We will only consider major features and neglect minor technical differences and implementation details. We will focus on the topics like `UI Prototyping`, `Task Based Usability Tasting`, `Continuous Experiments`, and `Model-based` approaches. In this chapter, we define some State of the art research (see section 2.1), and then we compare the tools based on the requirements (see section 2.2).

2.1 State of the Art Research

Model-Based UIs: Various modeling languages are introduced for formalizing the User Interface (UI). A standardized modeling language for software product content, abstract UI model, user interactions, and control behavior is Interaction Flow Modeling Language (IFML) [4]. As a result, IFML focuses on the platform-independent display of the user interface that is transferable between many platforms and devices. Cameleon [2] is a framework that divides the user interface into several elements to maximize the parts' reusability in various user, platform, and environment situations. A platform-independent abstract UI, a platform-dependent concrete UI, and a device-dependent final UI are the layers the framework offers to accomplish this.

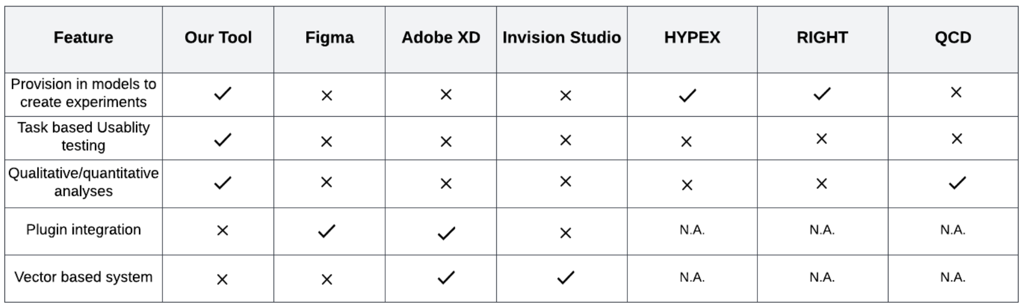
Continuous Experimentation: The idea of continuously testing the software product's fundamental assumptions on the users is through continuous experimenting. Several conceptual models and platform designs have been suggested to support the process. A feature with a high priority is chosen and added to the product using Hypothesis Experiment Data-Driven Development (HYPEX) [16], creating a feature backlog from strategic product goals. Another research is Rapid Iterative value creation Gained through High-frequency Testing (RIGHT) [8] that is very similar to HYPEX and is used to identify the hypotheses developed from the models. These hypotheses are tested using experimentation, and the best feature is implemented in the product. In contrast to HYPEX and RIGHT, Qualitative/quantitative Customer-driven Development (QCD) [17] focuses on combining qualitative customer feedback with quantitative customer observations.

However, none of the conceptual models and the continuous experiments focus on integrating the models in the experimentation of the UI prototypes. Therefore, we plan to develop a tool that combines all the above requirements. Figure 2.1 does a comparative analysis of the existing solutions and our proposed solution.

2.2 Comparison of Tools based on Requirements

This section compares the tools for UI prototyping based on the requirements; `(1) Model-Based` approach, `(2) Continuous Experimentation`, and `(3) Task-based Usability testing`.

UI Prototyping tools: The current popular platform for UI Prototyping are Figma[1], InVision Studio[2], and Adobe XD[3]. With a browser-based, cloud-hosted platform, Figma is a tool that facilitates collaboration and accessibility for UI/UX designers, developers, and anybody else on a team. Figma integrates various plugins like Autoflow for illustrating user flows and Figmotion for creating animations, enhancing its usability [13]. Similarly, InVision Studio allows designers to quickly build functional prototypes and share them with others with many well-designed tools. A vector-based method for building prototypes is available via Adobe XD, along with tools for adding interactions, transitions, and other dynamic features [13]. Because of its vector-based property, scaling and resizing the elements is done efficiently.



Feature	Our Tool	Figma	Adobe XD	Invision Studio	HYPEX	RIGHT	QCD
Provision in models to create experiments	✓	×	×	×	✓	✓	×
Task based Usablity testing	✓	×	×	×	×	×	×
Qualitative/quantitative analyses	✓	×	×	×	×	×	✓
Plugin integration	×	✓	✓	×	N.A.	N.A.	N.A.
Vector based system	×	×	✓	✓	N.A.	N.A.	N.A.

Figure 2.1: Comparative study for different tools

[1]Figma: `https://www.figma.com/`
[2]InVision Studio: `https://www.invisionapp.com/`
[3]AdobeXD: `https://www.adobe.com/products/xd.html`

3 Solution Idea

This section presents our plan to solve the problems discussed in the Introduction (see chapter 1). We propose a Model-based approach using the LEAN development process in our solution. As mentioned earlier, LEAN contains three phases Build (see section 3.1), Measure (see section 3.2), and Learn (see section 3.3). In the `Build` phase, we plan to create the `(1) Visualization of Prototypes`, and `(2) Model` these prototypes for Experiments and Tasks. In the `Measure` phase, we plan to `(3) Assign these prototypes to the users` and measure the `(4) Experiment and the Task measurements`. Finally, in the `Learn` phase, we would analyze the results by performing `(5) Different analyses`, and `(6) Improve our Prototype` from the analyses feedback and start a new iteration cycle.

Consider an example of a "A video streaming service" called `VideoStreamer (VS)`. We divide the different stakeholders of this company into developers (e.g., programmers, etc.) and non-developers (e.g., product designers). Usually, the designers are responsible for improving the UI and the UX (User Experience) of the product and the developers for developing them. But, the problem is, in the beginning, knowing which is the best prototype is impossible. Therefore, we need to perform experiments on the UI prototypes using A/B Testing. So, we would like to develop a platform for non-developers to experiment with different users and determine the best prototype. We would use "designers" for non-developers using our tool/application for the entire thesis.

3.1 Build

In this step (as per figure 1.2), we create ideas for the product and visualize the prototypes (explained in section 3.1.1) and then model them together for creating experiments and tasks for the users (explained in section 3.1.2). We plan to implement the build step in a low-code or no-code technique for our solution because it needs negligible installation, setup, training, and implementation work. The low-code or no-code technique enables software applications' rapid creation and deployment with little or no coding effort.

3.1.1 Visualising the Prototypes

We plan to create a system for designers to modify the UI elements in a canvas-like structure in an application using UI Prototyping. For this, the designers first create a canvas screen and input the name for that screen. As shown in figure 3.1, they can add UI elements like buttons, input buttons, select buttons, etc., on the canvas screen. Various UI elements are available for the drag and drop of the UI elements, as shown in figure 3.1 (right side). We propose to use Angular[1] for developing the application.

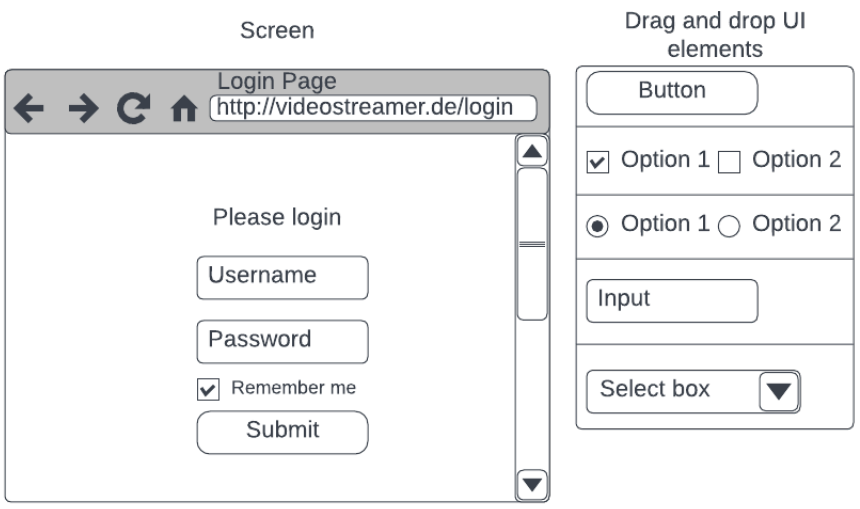


Figure 3.1: UI Prototyping using drag and drop of UI elements

Once the designers create multiple UI screens (i.e., multiple variants) for a view, they can move to the next screen by some logic (e.g., clicking on a button to go to the next screen). This way, the designers can design the entire application, having a sequential flow (i.e., the user will register, go to the Login page, next to the Dashboard page, etc.), using the canvas screen, and adjust the UI elements. At the same time, the designers can build the entire application quickly without any programming knowledge.

3.1.2 Modelling the Prototypes

In our application, we plan to convert the screens the designers develop into models. To do that, we need a modeling language that fits our requirements. As per figure 3.2, we first create the meta models, then build concrete models out of these to store

[1]Angular: `https://angular.io/`

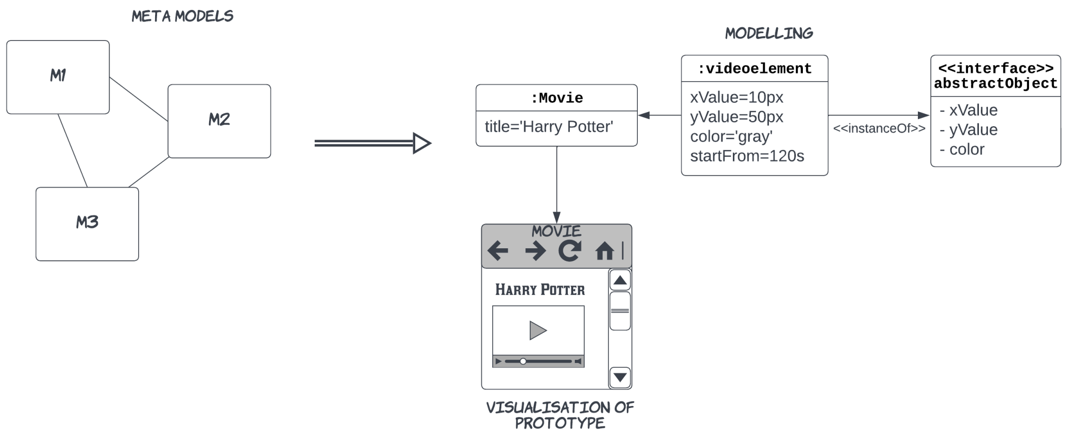


Figure 3.2: Example Modelling of Prototypes

the properties of the prototypes as attributes. Therefore, we need concrete models for `Experiments`, `Tasks` and `Prototypes`.

Prototype: In the `Prototype Model`, we store information about the prototypes and all their components as independent parts. This model stores all the different screens available in a software application and links them. As per figure 3.2, the Modelling part shows the abstract and the concrete models required for prototyping the UI elements. (e.g., The “abstractObject” is an interface that is inherited by all the UI elements that are concrete elements “videoelement” of the containers or the screens “Movie”). The abstractObject stores the attributes (e.g., position, style, text of the element) that can be manipulated to create the variant screens for experiments. E.g., if the designers want to create multiple variants of the screen, they can move the UI elements, or change their color or name.

Experiment: As explained before, we store different properties of the UI elements along with the variant information. As per figure 3.2 the “videoelement” under modeling contains attributes to store the position of the video viewer, which is a concrete element. Similarly, it can store some extra properties of the “videoelement” that are different from the “abstractObject”. This way, we can create simple Experiments on the users (e.g., changing the position or style of the UI elements) by assigning them different variants created in the Prototyping step.

Task: In the `Task Model`, we store a sequence of activities that the user needs to perform and measure the Task Success (TS). This activity sequence is stored in the models and linked to the users. A task must be precisely defined, e.g., create an account on Videostreamer app and locate Movie M1. A percentage is frequently used to represent TS. For instance, if 7 out of 10 succeeded and three failed, TS would be 70%. At the same time, we would also measure the Task Time, e.g., the time required to do the task as mentioned above. Then, we calculate the average Task Time for the whole group or only those who accomplished the task. This way, all these pieces of information are saved in the model and later used in the analyses (see section 3.3.1).

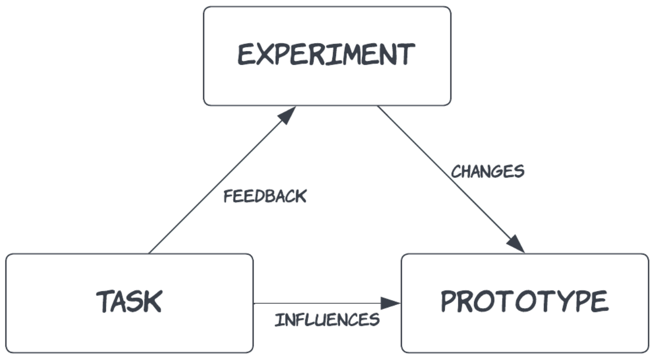


Figure 3.3: Triangle of Experiment, Task, and Prototype

We try to relate these models as depicted by the figure 3.3 in a triangle relationship. `(1) Experiment-Task, (2) Experiment-Prototype, and (3) Task-Prototype` relationships.

Experiment-Task: An experiment gets feedback from the tasks (e.g., the task success (TS) and the task time) to determine the best variant of a view.

Experiment-Prototype: From an experiment, it can be decided which is the best variant for a view and can thus modify and improve the prototype in an iteratively continuous process.

Task-Prototype: A task should be created based on the current state of the prototype, and thus it creates a relationship between the task and the prototype.

3.2 Measure

The prototypes, experiments, and tasks are ready to be assigned to the users. So, in this section, we explain how they are allocated to the users (see section

3.2.1) and collect feedback (see section 3.2.2) from them for improving the prototypes.

3.2.1 Prototypes for User Testing

The product designers continuously investigate new features and product enhancement opportunities. The designers understand their customers' experiences by creating that feedback loop with users. As a result, they need to conduct experiments. In these experiments, the population (i.e., all users) is divided into small user groups, each receiving a unique variant. This type of setup is called the "`Between-group`" design experiment. As mentioned, the tasks we created earlier for the experiments are allocated to the users performing experiments. As shown in figure 3.4, the experiment model assigns different variants for the prototype `View`: the `Grid view` (on the left) and the `List view` (on the right). These experiments are conducted on the users by dividing the users into groups and assigning one of the variants to a group. Then, the users perform the tasks for the specific experiment (e.g., Locate Movie M1), and all the users with their experiment variant will try to complete the task. The task data are measured (explained in section 3.2.2), analysed (explained in section 3.3.1), and finally, we find the best variant for the prototype.

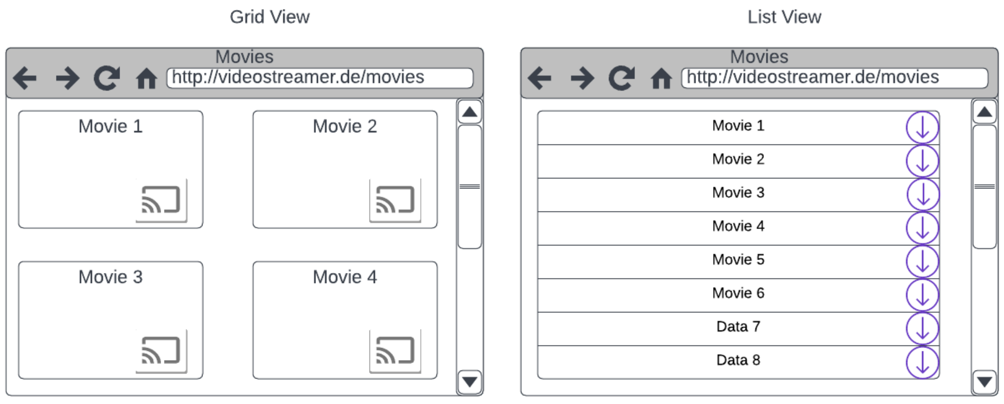


Figure 3.4: Example Experiment variants

3.2.2 Measuring the Experiment and Task results

To measure the experiment variants' performance, we add attributes to the models called "Measurements". This attribute is used to store the TS, a percentage to determine the number of users successfully completed the tasks. We also measure the time the user requires to complete a task called Task Time. The task time attribute in the model gets updated as soon as the user starts to perform the task. After collecting the data from the experiment variants, we do an analyses (see section 3.3.1) for determining the best variant and improve the prototype (see section 3.3.2).

3.3 Learn

In this section, we do the analyses (see section 3.3.1) from the data collected during the experiments, get feedback on the better variant, and improve our prototype (see section 3.3.2) from the feedback.

3.3.1 Analysis

To determine the "Winner" among the variants of a product's component, we perform data analytics on the feedback data we receive from the `Task Model` and the `Experiment Model`. We perform the `Quantitative` (presented in section 3.3.1), `Qualitative` (presented in section 3.3.1), and the `Task` (presented in section 3.3.1) analyses of the data.

Quantitative Analysis: Quantitative analysis uses mathematical and statistical methods to determine the behavior of the data. In quantitative research, various descriptive statistics methods like Means, Median, Variances, Standard Deviations, etc., are used to find some causal relationships in the data. After collecting the data, the experimenting server can calculate the measurements' mean (assume the mean is specified in the experiment) and show the results.

Qualitative Analysis: To further improve, we need to perform qualitative analysis. Qualitative analysis is used to determine the users' behavior and semantics. Qualitative research data is usually unstructured, coming from open-ended surveys. Our goal is to turn the unstructured data into a detailed description of the critical aspects of the problem. In our solution, we propose to perform a qualitative analysis of the data by asking some open-ended questions (e.g., "Are the items on the page easily locatable?", etc.) to the users. These questions can be asked to the users either after using the prototypes or after finishing certain tasks. The user's responses are studied, and we get feedback from this process on which variants are better for the users.

Task analysis: We also receive the data from the `Task Model`. Its data can be used to calculate the efficiency of the variant. E.g., if the task is to locate a particular movie, we can calculate the time required to complete the task as measurements for the users. Then, we calculate the average Task Time for the whole group or only those who accomplished the task. Usually, less time required for the task to be completed for the users is desirable. This testing type is called “`Supervised testing`” [11]. So, we observe the users, and they know the result (i.e., the movie they need to locate). This process gives us more accurate feedback on which variant performs better, and we can then update the prototype as per our triangle relationship (see fig 3.3).

3.3.2 Improving the Prototype

As per the LEAN development process, this is the final step in the cycle (see figure 1.2) of that iteration. After analyzing the data from experiments, we can conclusively claim with statistical evidence the “winner” variant of a component. This helps us to tune our model by adding a “default” attribute and setting its value to be of the “winner” variant. So, in the next iteration cycle, for the component, the “winner” variant is selected as default for future experiments. After modifying our model, we can visualize the changes by observing the UI Prototypes. E.g., In the `View` component of the `Videostreamer` app, if the `GridView` is determined to be more usable as per the experiment and is declared the winner, then in the next iteration, the `default` attribute in the `View` component will be set to `GridView` for the further experiments.

4 Structure of the Thesis

We plan to have our final thesis structure approximately in the following format:

1. Introduction
    - Motivation
    - Problem Statement
    - Research Approach (DSR Method)
    - Solution Approach
2. Background/Foundation (Here, explain in detail the topics needed for developing a basis for the thesis. Some of the topics include:)
    - Continuous Experimentation
    - Low code / No Code techniques
    - Model-based Software engineering
    - UI Prototyping
    - Crowdsourcing
    - Task-based Usability Testing
3. Related Work
    - State-of-the-art research
    - Comparison
4. Design (Design principles)
    - Build
    - Measure
    - Learn
5. Solution Implementation
    - Design Features
    - Architecture

   - Technologies used
   - Frontend
   - Database Schema
   - Backend (Server)
   - Proof of concept tool

6. Evaluation
   - User case study
   - Limitations and Risks
7. Conclusion and Future Work

These chapters are used for creating the Work Packages and a timeline for the thesis (explained in Chapter 5).

5 Work Packages and Timeline

This section defines the work packages (see section 5.1) and the timeline (see section 5.2) to fulfill the tasks in the following months.

5.1 Work Packages

We divide our thesis into Work packages (WP) or Tasks. We can use this to track the progress of the thesis and get feedback.

WP1 (Introduction): In this package, we would like to start by introducing the topic, identifying the problems by defining the problem statement, and finally, defining our research and solution approach.

1. Motivation
2. Problem Statement
3. Research and Solution Approach

WP2 (Foundation): In this package, we would like to define some terminologies according to the research (e.g., Continous Experimentation, Low-Code, etc.)

WP3 (Related Work): Here, we would like to find some State-of-the-art research and compare them with the research questions or requirements we define for the thesis.

1. State-of-the-art research
2. Comparison

WP4 (Design): Here, we would like to design our tool as per the LEAN development cycle, containing the Build, Measure, and Learn phases.

1. Build: We would like to build the tool for the designers so that they can create prototypes and experiment with the users.
2. Measure: In this step, we would assign the experiment variants and assign tasks to the users to measure their feedback.
3. Learn: Finally, we would learn from the experiment by analyzing and improving the prototype. This finishes one cycle iteration, and the designers repeat the steps with the updated prototype.

WP5 (Implementation): In this package, we would like to implement the prototypes by developing the meta-models, defining the architectures, and coding them for visualization of the prototypes.

1. Architecture: We would define the architecture for the experimentation server (containing experimentation and task manager) and the prototypes (containing prototype manager)
2. Technologies used: Here, we define the technologies that we use for the implementation of the platform
3. Proof of concept tool: Developing, Testing, and Deployment of the application (using docker[1])

WP6 (Evaluation): In this package, we would test our prototypes by creating the experiments and the tasks for the users and measuring feedback from them.

1. Create a user case study and assign experiments and tasks to the users
2. Analyze the measurements
3. Limitations and Risks

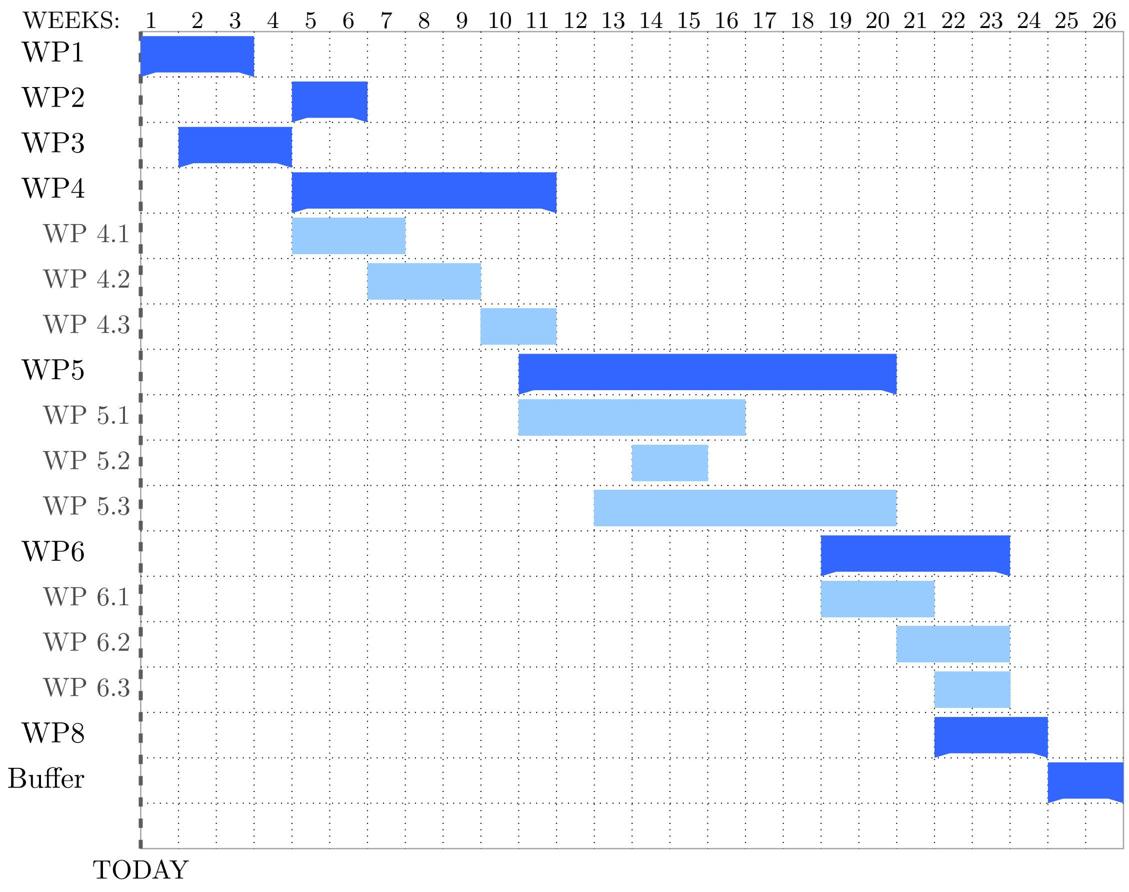
WP7 (Conclusion): Here, we conclude the thesis and write the scope for future work.

1. Conclusion
2. Future work

[1]Docker: `https://www.docker.com/`

5.2 GanttChart for Timeline

This section gives a rough estimation of the time required for the WPs execution. We divide the entire time period of the thesis into 26 weeks, with the first four weeks dedicated to the Proposal or the Exposée.



Bibliography

[1] Faheem Ahmed, Luiz Fernando Capretz, and Piers Campbell. Evaluating the demand for soft skills in software development. IT Professional, 14(1):44–49, 2012. doi: 10.1109/MITP.2012.7.

[2] Lionel Balme, Alexandre Demeure, Nicolas Barralon, Joëlle Coutaz, and Gaëlle Calvary. Cameleon-rt: A software architecture reference model for distributed, migratable, and plastic user interfaces. In European Symposium on Ambient Intelligence, pages 291–302. Springer, 2004.

[3] Steve Blank. Why the lean start-up changes everything. https://hbr.org/2013/05/why-the-lean-start-up-changes-everything, May 2013.

[4] Marco Brambilla and Piero Fraternali. Interaction flow modeling language: Model-driven UI engineering of web and mobile apps with IFML. Morgan Kaufmann, 2014.

[5] Jordi Cabot. Positioning of the low-code movement within the field of model-driven engineering. In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381352. doi: 10.1145/3417990.3420210. URL `https://doi.org/10.1145/3417990.3420210`.

[6] Alan M. Davis. Software prototyping. volume 40 of Advances in Computers, pages 39–63. Elsevier, 1995. doi: https://doi.org/10.1016/S0065-2458(08)60544-6. URL `https://www.sciencedirect.com/science/article/pii/S0065245808605446`.

[7] Aleksander Fabijan, Pavel Dmitriev, Helena Holmström Olsson, and Jan Bosch. The benefits of controlled experimentation at scale. In 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 18–26, 2017. doi: 10.1109/SEAA.2017.47.

[8] Fabian Fagerholm, Alejandro Sanchez Guinea, Hanna Mäenpää, and Jürgen Münch. The right model for continuous experimentation. Journal of Systems and Software, 123:292–305, 2017.

[9] Shirley Gregor, Leona Chandra Kruse, and Stefan Seidel. The anatomy of a design principle. 21(6).

[10] Mark A Hart. The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses eric ries. new york: Crown business, 2011. 320 pages. us$26.00. Journal of Product Innovation Management, 29(3):508–509, 2012. doi: https://doi.org/10.1111/j.1540-5885.2012.00920_2.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-5885.2012.00920_2.x.

[11] Jasmin Jahić, Thomas Kuhn, Matthias Jung, and Norbert Wehn. Supervised testing of concurrent software in embedded systems. In 2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), pages 233–238, 2017. doi: 10.1109/SAMOS.2017.8344633.

[12] Faezeh Khorram, Jean-Marie Mottu, and Gerson Sunyé. Challenges and opportunities in low-code testing. In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381352. doi: 10.1145/3417990.3420204. URL https://doi.org/10.1145/3417990.3420204.

[13] Urvashi Kokate, Kristen Shinohara, and Garreth W Tigwell. Exploring accessibility features and plug-ins for digital prototyping tools. 2022.

[14] William L. Kuechler and Vijay K. Vaishnavi. On theory development in design science research: anatomy of a research project. European Journal of Information Systems, 17:489–504, 2008.

[15] Eveliina Lindgren and Jürgen Münch. Raising the odds of success: the current state of experimentation in product development. Information and Software Technology, 77:80–91, September 2016. ISSN 0950-5849. doi: 10.1016/j.infsof.2016.04.008.

[16] Helena Holmström Olsson and Jan Bosch. The hypex model: from opinions to data-driven software development. In Continuous software engineering, pages 155–164. Springer, 2014.

[17] Helena Holmström Olsson and Jan Bosch. Towards continuous customer validation: A conceptual model for combining qualitative customer feedback with quantitative customer observation. In International Conference of Software Business, pages 154–166. Springer, 2015.

[18] Brian L. Smith, William T. Scherer, Trisha A. Hauser, and Byungkyu Brian Park. Data–driven methodology for signal timing plan development: A computational approach. Computer-Aided Civil and Infrastructure Engineering, 17, 2002.

[19] David J. Teece. Business models, business strategy and innovation. https://asset-pdf.scinapse.io/prod/2140699752/2140699752.pdf.

[20] Kathryn Whitenton. But you tested with only 5 users!: Responding to skepticism about findings from small studies. https://www.nngroup.com/articles/responding-skepticism-small-usability-tests/, 2019.