UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Model Based Experimentation on UI Prototypes

Rakshit Bhat

Exposé/Proposal – October 12, 2022.
Software Innovation Campus Paderborn (SICP)

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

Contents

1 Introduction

This chapter 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]. So, the developers have come up with different techniques to meet this requirement criteria. Similarly, increasing product complexity and ambiguity have a significant impact on software development. 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 [17]. With the increasing complexity of products, it becomes challenging to determine user requirements. Different people can have overlapping or contradicting opinions. And to reduce these risks, there has to be early detection of the user's needs and requirements. Giving users a "partially functioning" system is the most excellent method to determine their requirements and suggestions [4]. This ensures that the developers with high uncertainties in the early product development can validate by testing the underlying assumptions [2]. Developers can use this feedback to validate the most critical assumptions about the software product. This validation can be used to decide whether to add, remove or update a feature [12]. 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. Fabijan et al. [5] have shown the benefits of experiments in many use cases with incremental product improvement. In Experimentation, the product owner designs different software variants (e.g., Different Subscription fees for registration) of the product, and the developer integrates these variants and assigns them to a distinct group of users. The variant with better results as per some evaluation criteria (more number of user registrations) is deployed for the entire set of users. So, an experiment can be valuable when it solves real-world problems. Hence, for experiments to be successful, they should offer one or more solutions that will benefit users.

1.2 Problem Statement

This section explains the current problems faced by companies during software development. We try to define three problems and determine the research and the solution approach for the same.

Problem 1: The developers are responsible for creating the experiments, and the product owners give the ideas and feedback to the developers. The main problem is integrating the product owners and the non-developers into the development process to bridge the gap.

Problem 2: In a company, it is usually a problem to determine the number of test users needed for doing the experiments. Therefore, it is not easy to do experiments and get conclusive feedback on the "winner" variant. Because, with a small amount of data, it is impossible to prove one of the variants outperforms the others statistically [18].

Problem 3: Most often, the software application collects data from the experiments. Not all the data is used in the analysis phase reducing the software applications to improve based on customer feedback [16].

1.3 Research Approach

To solve the first problem, the developers focus more on automating the software code rather than coding everything stated in the product requirements [8]. This approach is formally known as a `Low-code` or `No-Code` approach. So, this approach helps to have a UI interface for the non-developers to understand and develop the software products [10]. One approach to support the product owners in developing the variants is UI Prototyping. UI prototyping creates new UI variants using predefined UI elements (e.g., Drag and drop the UI element Button into the screen). This helps product owners to be creative and innovative because it gives them visual feedback. So, if UI prototypes are developed in a low-code technique, they would be lightweight software that helps the product owner develop various prototypes and conduct experiments on the users. According to Cabot et al. [3], the low-code has become more accessible for Model-driven development. Similarly, while creating prototyping, the software should have connections between the screens (e.g., clicking on a button should go to the next screen), and this logic can be achieved using Models. Models are used broadly in prototyping because a model represents or describes the aspects of the systems that cannot be described adequately in a

system of interest [13]. Moderately accurate models can be created using an iterative approach in software development by getting continuous feedback from the users.

To solve the `second` problem, we use supervised task-based usability testing [9]. The fundamental principle of usability testing is to have real users attempt to use the software application to do certain activities. Observing users interact with an interface is the most efficient way to determine what functions well and what doesn't. The users will undertake realistic actions, giving us qualitative insights into the problems that users are experiencing and enhancing the design with these insights [14]. These activities can be the tasks or scenarios (e.g., Locate a movie M1) for the users to complete and analyze (e.g., the time/clicks required to locate the Movie M1).

To solve the `third` problem, the models should use data to measure the experiments' success for improvement. This process is called a Data-driven development approach. This approach uses meaningful, actionable consumer feedback regarding the effects of the product experiments by using the `Qualitative and Quantitative` data analysis. Using a combination of qualitative and quantitative data can improve an evaluation by ensuring that the strengths of another balance the limitations of one type of data confirming that the knowledge is enhanced by integrating different techniques.

1.4 Solution Approach

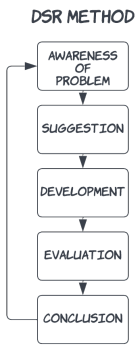


Figure 1.1: Design Science Research Cycle [11]

The process of creating experiments and testing their variants is usually not systematically arranged, creating anomalies, and leading to unsuccessful experiments. To have a standard, we plan to conduct a design science research (DSR) study to obtain some Design Principles (DPs) defined for the whole process of experimentation

[11]. 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 [6]. 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 [11] consisting of five iteratively conducted steps (see figure 1.1). First, we identify the `(1) Awareness of the Problem` and provide a `(2) Suggestion of a possible solution`. Next, we work on the `(3) Development of the software artifact` and conduct an `(4) Evaluation` of it. Based on the evaluation results, we provide `(5) Conclusions` [15].

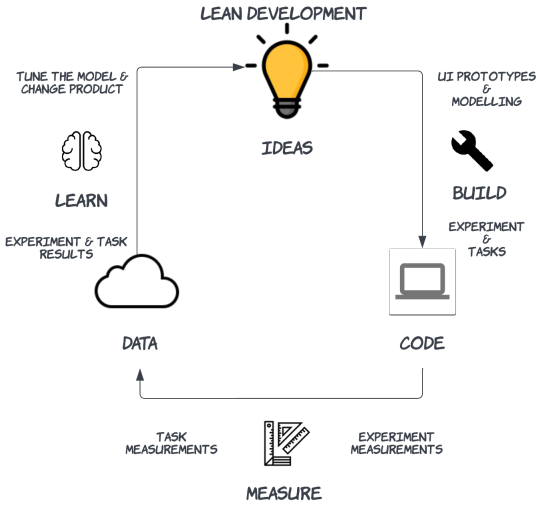


Figure 1.2: LEAN Development technique

In our solution, we use the LEAN development technique (see figure 1.2) for development as it is used to develop customers friendly products [7]. 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. The solution approach is explained more in detail in Chapter 3.

2 Related Work

This section considers some existing frameworks 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`, `Crowdsourcing`, `Data-driven approaches`, `Continuous Experiments` and `Model-based approaches`.

For example, you could write about:

- Figma[1]
- Others[2]
- ...

Note: This template is just an example. You can of course combine Sections 1 and 2 into one section or use a different structure.

[1] https://www.figma.com/
[2] https://webflow.com/blog/prototyping-tools

3 Solution Idea

This section presents our plan for obtaining the objectives discussed in section 1. In our approach, we propose a Model-based and Data-driven development approach using the LEAN development process. As mentioned earlier, LEAN contains three phases Build, Measure, and Learn. 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 performing `(5) Different analyses`, and `(6) Improve our Prototype` from the analyses feedback.

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, designers, etc.) and non-developers (e.g., product owners). Usually, the developers are responsible for developing various features required for the company. But, the problem is, in the beginning, knowing which design fits better for a UI is impossible. Therefore, we need to perform experiments on the UI using A/B testing. So, the developers develop different variants for experimenting with the UI and get feedback from the non-developers. This feedback process wastes a lot of time, so our solution is to give freedom to the product owners and reduce the gap between the product owners and the developers by allowing the product owners to create the UI using UI prototyping.

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 non-developers to modify the UI elements in a canvas-like structure in an application using UI Prototyping. For this, the non-developers 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. We propose to use angular[1] for developing the application. Various UI elements are available for the drag and drop of the UI elements, as shown in figure 3.1 (right side).

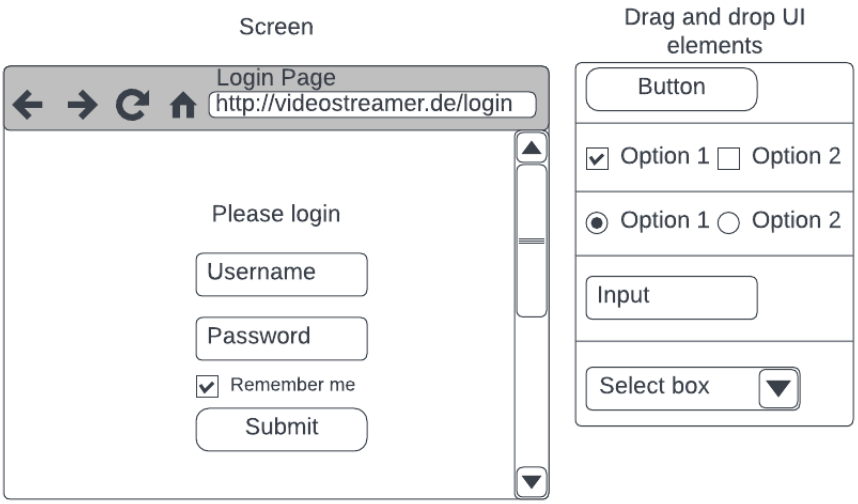


Figure 3.1: UI Prototyping using drag and drop of UI elements

Once the non-developers finalize the UI screen, 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 non-developers 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 non-developers 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 non-developers 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

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

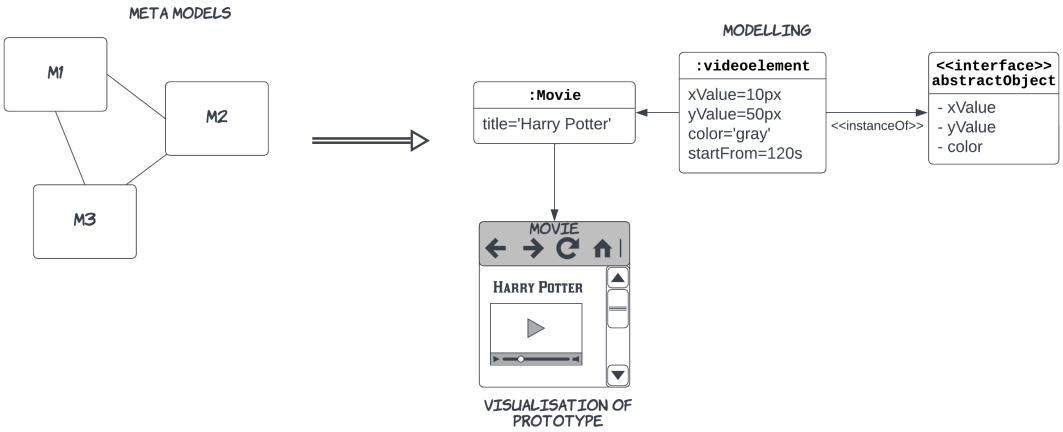


Figure 3.2: Example Modelling of Prototypes

to store 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").

Experiment: In the `Experiment Model`, 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". Moreover, the models should be able to accept measurements (e.g., `ClickRate` to measure the user clicks, `ViewTime` to measure the time spent by the user) to determine the best fit among the variants. This way, we can create simple Experiments on the users (e.g., changing the position of the UI elements using the drag-and-drop feature).

Task: In the `Task Model`, we store a sequence of activities that the user needs to perform and obtain feedback as measurements from the users for testing the software product. This activity sequence is stored in the models and linked to the users. The

models should be able to accept the Measurements (like the Experiment models) to get feedback from the users (e.g., `ClickRate` and `ViewTime`).

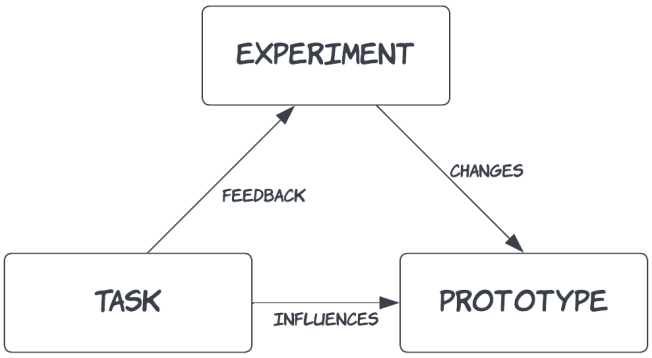


Figure 3.3: Triangle of Experiment, Task, and Prototype

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 can create various tasks for the users and get some feedback in the form of data from the Task model.

Experiment-Prototype: From an experiment, it can be decided which is the best variant for the prototype 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 product, and thus it creates a relationship between the task and the product.

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 owners continuously investigate new features and product enhancement opportunities. The owners better 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 earlier, we also create tasks for the users allocating the tasks to the users performing experiments. As shown in figure 3.4, the experiment model creates 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 tasks are created for the specific experiment (e.g., Locate Movie M1). So, all the users with their experiment variant will try to complete the task. The 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.

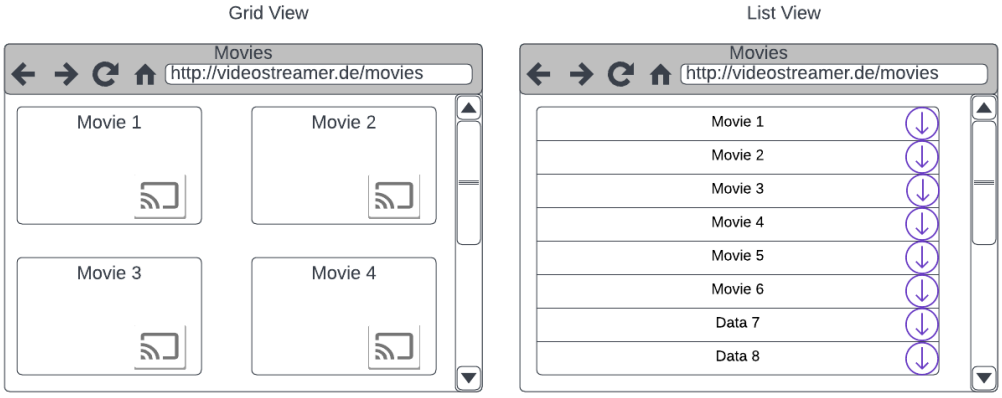


Figure 3.4: Example Experiment variants

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 stores the voluntary (e.g., the number of clicks the user makes) and involuntary (e.g., the time required for the user to complete a task) user activities. This attribute gets updated when the users perform the Task assigned to them by the `Task Model`. (e.g., The task is to locate a movie for a registered user, so the user must log in, go to the movies page and search for the film, or scroll to find the film and click on the movie's title to play it.) The UI variant (List or Grid View) assigned to the user plays a vital role in making this

process smooth. Therefore, measuring the Time or the Clicks required to reach the goal can be a critical factor in deciding the better variant. After collecting the data from the experiment variants, we do an analyses (see section 3.3.1) and improve 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 experiment's variant, and improve our prototype (see section 3.3.2).

3.3.1 Analysis

In our solution, we focus on data-driven development. 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). But, it isn't easy to generalize the results using means or descriptive statistical methods. Therefore, a significance test can be calculated to validate the probability of the event's occurrence, claim from the mean, and declare the winner variant.

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, interviews, etc. 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 to the users (e.g., questions like "What do you think about the Look and feel of the software application?", "Are the items on the page easily locatable?", etc.). The users' responses can be studied using some tools using the inductive or deductive coding technique. So, in the end, we get feedback from this process on which variants are

better for the users. This information is forwarded to the models for modifying the prototypes.

Task analysis: We also receive the data from the `Task Model`. Its data can be used to calculate the efficiency of the variant. A task model contains measurements for getting feedback from the users. E.g., If the task is to locate a particular movie, we can calculate the number of clicks and time required as measurements for the users to reach their destination. This testing type is called "`Supervised testing`" [9]. 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, the `GridView` is determined to be more usable as per the experiment and is declared the winner. Therefore, in the next iteration, the `default` attribute in the `View` component will be set to `GridView` for the following experiments.

4 Structure of the Thesis

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

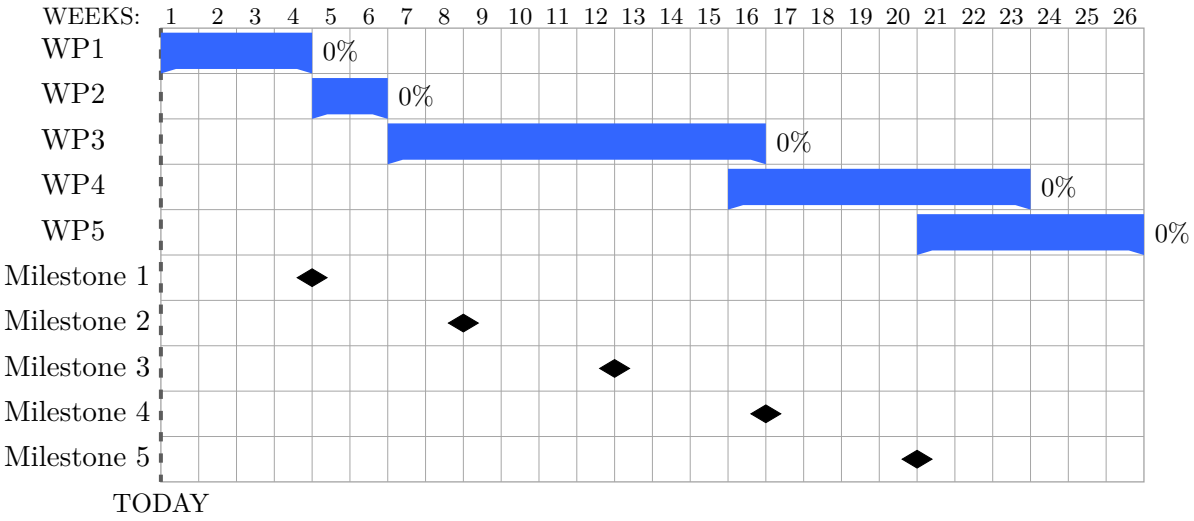
1. Introduction
    - Motivation
    - Problem Statement
    - Research Approach
    - Solution Approach
2. Background/Foundation (Here, explain in detail the topics needed for developing a basis for the thesis. Some of the topics include:)
    - Continuous Experimentation
    - Low code / No Code techniques
    - Model-based Software engineering
    - Data-driven Engineering
    - A/B Testing, etc.
3. Design
4. Implementation
5. Evaluation
6. Conclusion and Future Work

5 Work Packages and Timeline

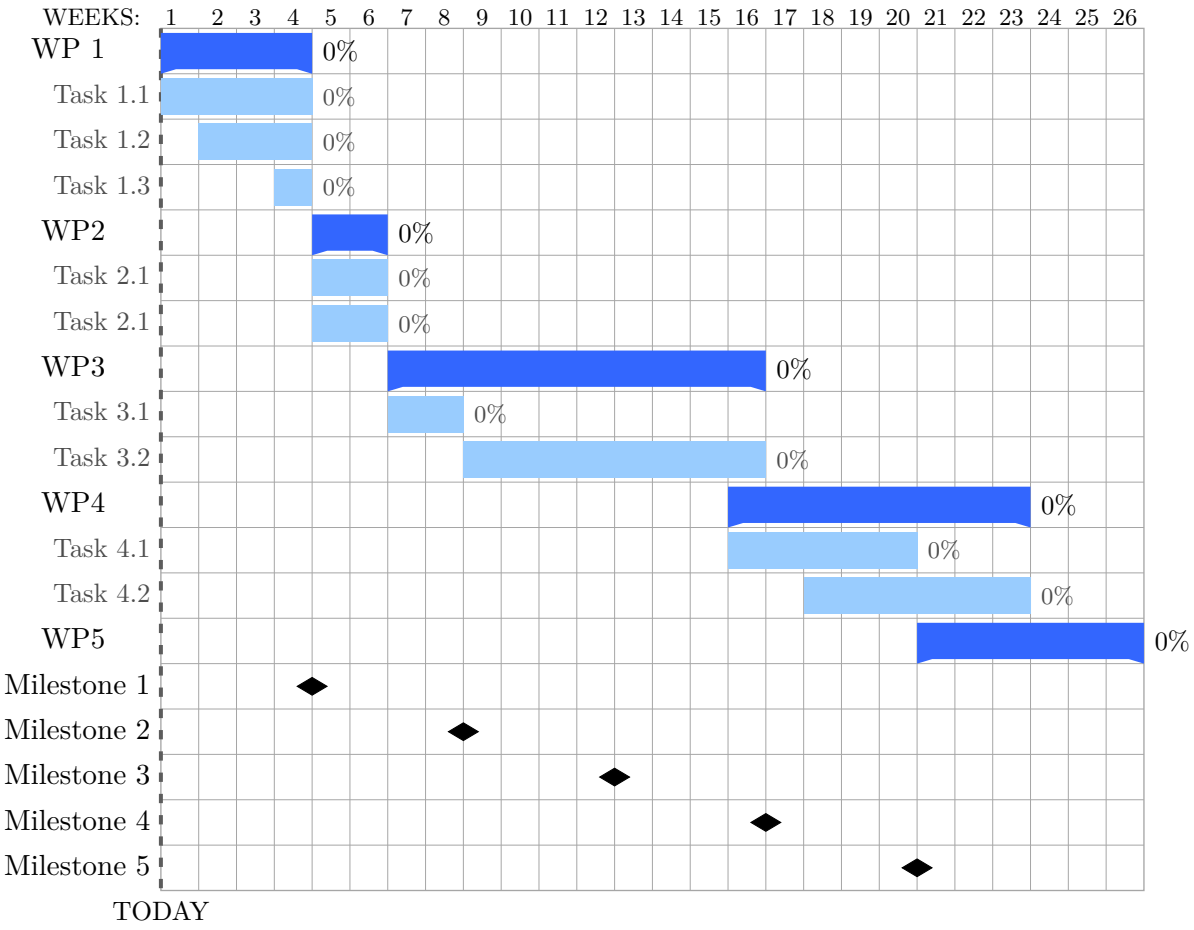
In this section, define work packages and milestones you are going to fulfill in the next months. Afterwards, create a timeline depicting the starting and ending date for each work package and milestone. You can use a nice Gantt chart (see examples below). Proper list of work packages and dates is also enough.

5.1 GanttChart Master Thesis

5.1.1 GanttChart Simple



5.1.2 GanttChart Detailed



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

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] G. F. Hoffnagle and W. E. Beregi. Automating the software development process. IBM Systems Journal, 24(2):102–120, 1985. doi: 10.1147/sj.242.0102.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] L. Luqi and R. Steigerwald. Rapid software prototyping. In Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences, volume ii, pages 470–479 vol.2, 1992. doi: 10.1109/HICSS.1992.183261.

[14] Marieke McCloskey. Turn user goals into task scenarios for usability testing. https://www.nngroup.com/articles/task-scenarios-usability-testing/, 2014.

[15] Enes Yigitbas Sebastian Gottschalk, Suffyan Aziz and Gregor Engels. Design principles for a crowd-based prototype validation platform. 2021. doi: 10.1007/978-3-030-91983-2_16.

[16] 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.

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

[18] 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.