UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Model Based Experimentation on UI Prototypes

Rakshit Bhat

Exposé/Proposal – October 11, 2022.
Our Group Name.

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

Contents

1 Introduction

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 [14]. 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 [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 [11]. This process is called `Continuous Experimentation (CE)`. 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 continuous experiments in many use cases with incremental product improvement. In CE, 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.

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 between the developers and the product owners.

Problem 2: The process of creating experiments and testing its variants is usually not systematically arranged, creating anomalies that give rise to unsuccessful experiments. (Use design principles)

Problem 3: In a small-scale company containing a few users, 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.

Problem 4: Most often, the software application collects data from the experiments, and not all the data is used in the analysis phase reducing the software applications to improve based on customer feedback. (Use Data-driven development)

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 [7]. This approach is formally known as a `Low-code` approach. So, this approach helps to have a UI interface for the non-developers to understand and develop the software products [9]. 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 elements 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 [12]. 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, a design science research (DSR) study should be conducted to obtain some Design Principles (DPs) defined for the whole process of experimentation [10]. 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 [10] consisting of five iteratively conducted steps. First, we identify the `(1) Awareness of the Problem based on a real-world 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, another iteration is undertaken, and/or our research contributions as `(5) Conclusions` are provided [13].

To solve the `third` problem, use supervised testing.

To solve the `fourth` 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.

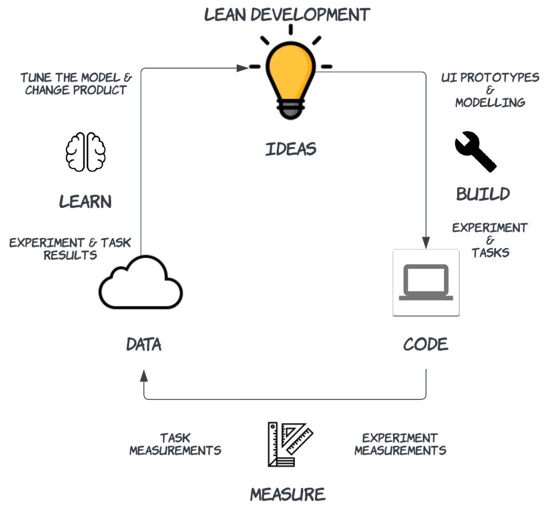


Figure 1.1: LEAN Development technique

Solution Approach: In our solution, we use the LEAN development technique (see figure 1.1) for development. LEAN development technique can be divided into 3 phases Build, Measure, and Learn. In the `(1) Build` phase, we plan to create the `UI Prototypes`, `Models`, create `Experiments`, and assign `Tasks` to the users. In the `(2) Measure` phase, we plan to measure the `Task and the Experiment measurements` and perform some analysis on the data received. And finally, in the `(3) Learn` phase, we would like to Display the `Experiment results` and `Tune` our models to decide the better variant among the others. The solution approach is detailed 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 the previous section. In our approach, we propose a Model-based Data-driven development approach involving a six-step solution:

- `UI Prototyping` using the drag and drop approach.
- `Modelling` these prototypes.
- Creating `Experiments` from the models and assigning them to the users.
- `A/B testing`.
- `Quantitative analysis`.
- `Qualitative analysis`.

As per the LEAN development Model (see Figure 1.1), the `UI Prototyping` and the `Modelling` falls under the `Ideation` phase, the `Experiments` and `A/B Testing` which falls under `Coding` phase measure the data, and finally the `Qualitative` and `Quantitative` analysis which falls under `Data` phase learn and tune our models.

Let us understand the solution approach with the help of an example "A video streaming service" called `VideoStreamer (VS)`. As shown in figure 3.1, this company has different stakeholders e.g., users, developers, product owners, etc. Usually,

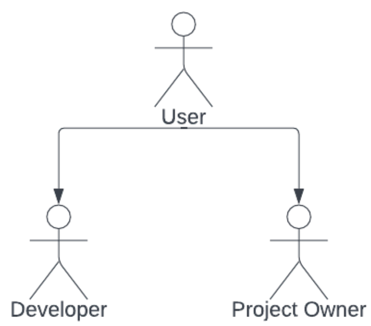


Figure 3.1: Different Stakeholders in the Company VS

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. This approach has become very popular among developers. So, the developers develop different variants for experimenting with the UI and get feedback from the product owners. 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 UI Prototyping

We plan to implement the UI prototyping in a low-code platform for our solution because it needs negligible installation, setup, training, and implementation work. The low-code platform enables rapid creation and deployment of business applications with the least amount of coding effort. We plan to set up a system for the product owner to modify the UI elements in a canvas-like structure in an application. For this, the product owners first create a canvas screen and input the name for that screen. As shown in the figure 3.2, they can add UI elements like buttons, input, 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 3.2 (right side).

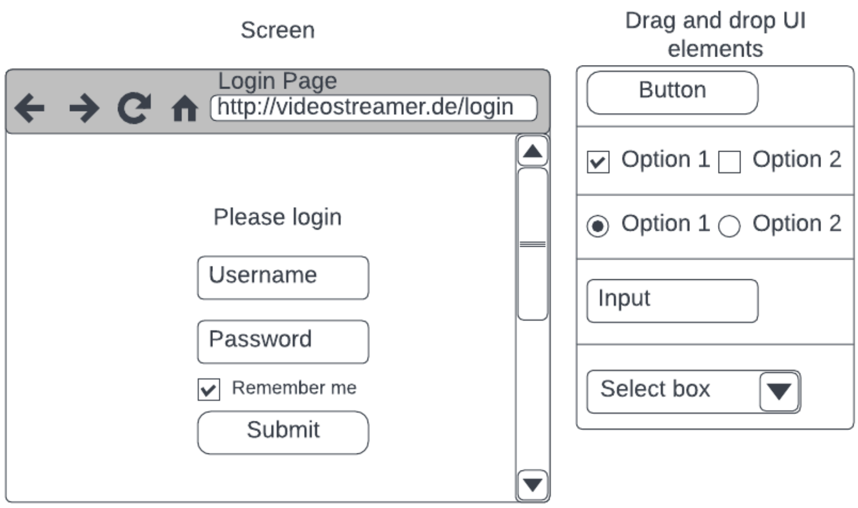


Figure 3.2: UI Prototyping using drag and drop od UI elements

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

Once the UI screen is finalized, the product owner can move to the next screen by some logic (E.g., clicking on a button go to the next screen). This way, the product owner can design the entire application, having a semantic flow, using the canvas screen, and adjusting the UI elements.

3.2 Prototype Modelling

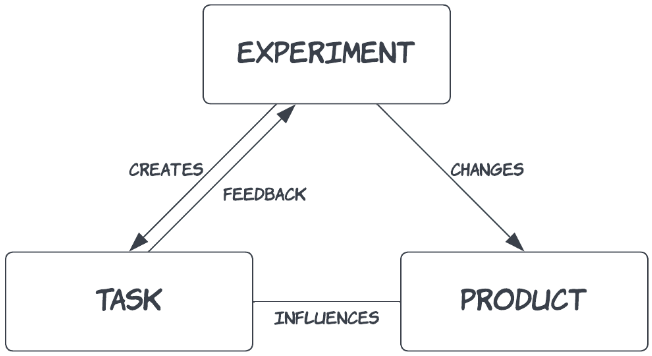


Figure 3.3: Triangle of Experiment, Task and Product

In our application, we plan to convert the screens the product owner develops into models. To do that, we need a modeling language that fits our requirements. Based on our domain, we need models for `Experiments`, `Tasks` and `Products`.

Experiment: In the `Experiment Model`, we store different properties of the UI elements along with the variant information.
E.g., from our `Videostreamer app`: If the experiment is on a `Button` element, our model should have information about the position of the button, the style of the button (storing the color, font, etc.), the title of the button, etc. All this information would be stored in the model as its properties or attributes. Moreover, the model can obtain this information from the UI prototyping done by the product owner using drag and drop feature.

Product: In the `Product Model`, we store information about the product and all its components as independent parts.
E.g., from our `Videostreamer app`: A video streamer would have different screens (see Fig 3.4 right side), a Home screen, a Video search screen, a Video display screen, etc. These screens are modeled to relate different screens and the elements within a specific screen. We plan to create Meta-models (see Fig 3.4 left side) and

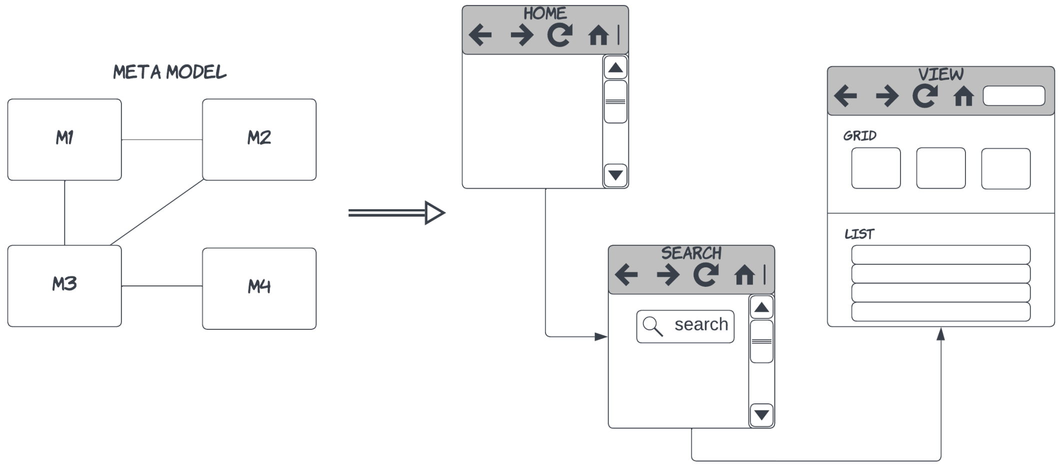


Figure 3.4: Example Meta Model

develop relationships between them. Moreover, the models should be able to accept parameters for dynamic runtime changes and measurements to determine the best fit among the variants.

Task: In the `Task Model`, we store a sequence of events that the user needs to perform and obtain feedback as measurements from the users for testing the software product.
E.g., from our `Videostreamer app`: We would give a task to a user U1, saying "Navigate to the movie M1" and we would measure the number of steps/clicks and the time required for the user to perform this task.

There are some relationships between these models as depicted by the figure 3.3 viz. `Experiment-Task, Experiment-Product, Task-Product` 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-Product: From an experiment, it can be decided which is the best variant for a product and can thus modify and improve the product in an iteratively continuous process.

Task-Product: 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.

3.3 Experimenting using A/B Testing

The product owners continuously investigate new features and product enhancement opportunities by conducting frequent experiments. The owners better understand their customers' experiences by creating that feedback loop with users. As a result, they can constantly give a practical solution. There are various types of experimenting[2] options available for the product owners. In our solution, we try to find which of these better fit our requirements and find an approach for conducting experiments with the UI elements. The product owner should be able to create various experiments by moving the UI elements and having multiple views for a particular screen.

E.g., from our `Videostreamer app`: As shown in figure 3.5, the experiment model can create different variants for the component `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. This type of setup is called the "`Between-group`" design experiment. Then, the users would be given specific tasks as per the `Task Model`, and the measurements will provide feedback to the `Experiment Model`. The data obtained are analyzed (explained in section 3.4), and finally, we find the best variant for a product's component.

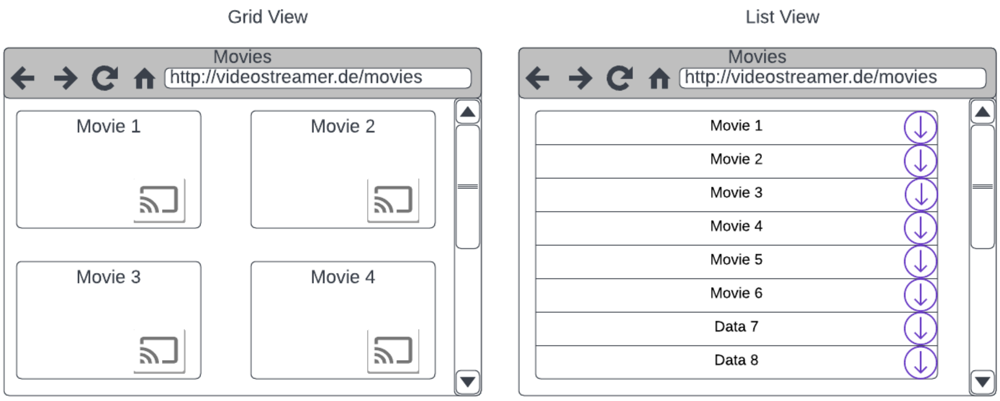


Figure 3.5: Example Experiment variants

[2]A/B Testing: `https://www.hotjar.com/conversion-rate-optimization/glossary/ab-testing/`

3.4 Data Analysis

In our solution, we focus on model-based and data-driven development. As per figure 3.6, the models provide some data to the data analysis service, receive feedback from it, and tune and improve the model in a continuous and iterative manner. For determining the “Winner” among the variants of a product’s component, we perform data analytics on the feedback data that we receive from the `Task Model` and the `Experiment Model`. We propose to do both `Quantitative` (presented in section 3.4.1) and `Qualitative` (presented in section 3.4.2) analyses of the data.

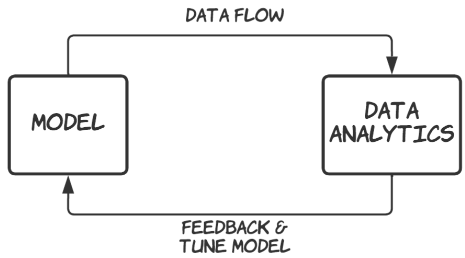


Figure 3.6: Model based Data Driven development

3.4.1 Quantitative Analysis

Quantitative analysis uses mathematical and statistical methods to determine the behavior of the data and improve performance by tuning the models. In quantitative research, various descriptive statistics methods like Means, Median, Variances, Standard Deviations, etc., are used. We try to find some causal relationships in the data.

In our solution, from the `Experiment Model`, we can calculate the measurements on various product components.
E.g., from our `Videostreamer app`: For the View component, the measurements could be “Watch time” to measure the duration of time spent by the user on that component’s variant, “Click Rate” to count the clicks, etc. Then, the experimenting server can calculate the measurements’ mean (assume mean is specified in the model by the product owner). To validate this data, a significance test can be calculated to determine the probability of the event’s occurrence, claim from the mean, and declare the winner variant.
Secondly, We also receive the data from the `Task Model`. Its data can be used to calculate the efficiency of the variant.
E.g., from our `Videostreamer app`: If the task is to locate a particular movie, we

can calculate the number of clicks and time required for the users to reach the end result (i.e. the movie in the task).
This testing type is called "`Supervised testing`" [8]. 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 product as per our triangle relationship (see fig 3.3).

3.4.2 Qualitative Analysis

In order to further tune our models, we need to perform qualitative analysis. Qualitative analysis is used to determine the users' behavior and semantic analysis. In qualitative research, the data is usually unstructured as they are from open-ended surveys, interviews, photos, drawings, responses from focus groups, 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 qualitative analysis of the data by asking some open-ended question to the users.

E.g., from our `Videostreamer app`: We can ask questions like "What do you think about the Look and feel of the software application?", "Are the items on the page easily locatable?". The users' responses can be studied using some tool using the inductive or deductive coding technique. If we select the inductive coding approach, we will scrutinize the reactions and those which are similar and group them. These groups will be coded into labels, and we will formalize and analyze the categories. To understand the process better, see figure 3.7. So, we in the end we get a feedback from this process on which of the variants is better for the users. This information can be forwarded to the models for reforming them.

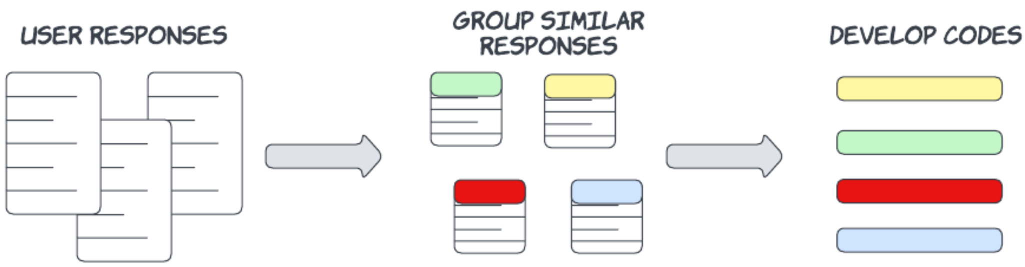


Figure 3.7: Qualitative analysis using inductive coding

4 Structure of the Thesis

Please give a general overview on how your thesis is divided into sections and chapters, for example, a table of contents.

Typical thesis could have the following structure:

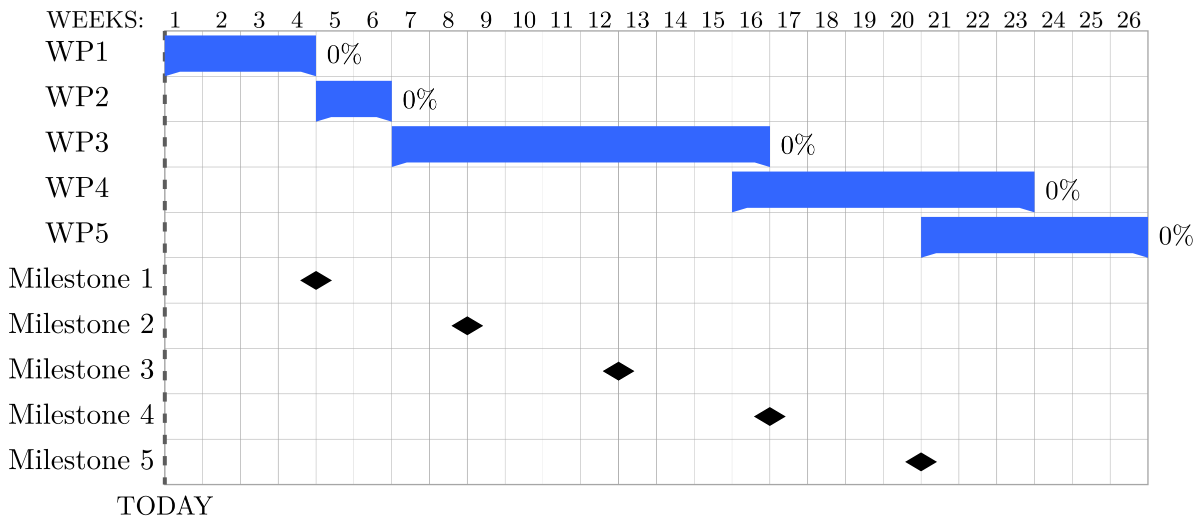
1. Introduction
2. Background
3. Design (design of your approach, it will contain some charts and architecture overview)
4. Implementation (detailed implementation description)
5. Evaluation (test environment, used tools and libraries, evaluation results)
6. Conclusion and Future Work

5 Work Packages and Timeline

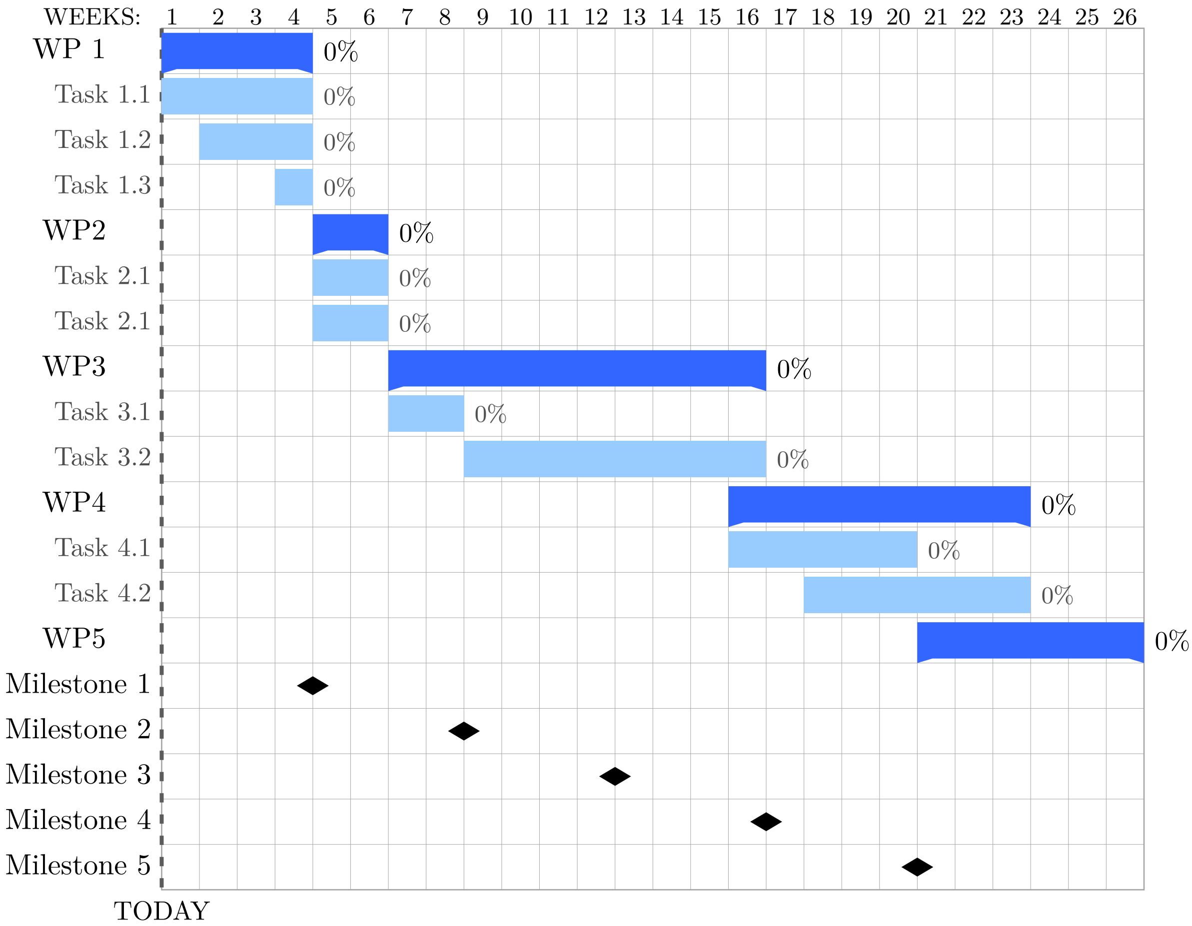
In this section, define work packages and milestones you are going to fulfill in the next months. Afterwards, create a timeline depicting the starting and ending date for each work package and milestone. You can use a nice Gantt chart (see examples below). Proper list of work packages and dates is also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

[8] 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.

[9] 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`.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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