Development and Evaluation of a Model-Based UI Prototyping Experimentation Approach

Rakshit Bhat

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

Advisor: Mr. Sebastian Gottschalk

Department of Computer Science
Universität Paderborn / Paderborn University

This dissertation is submitted for the degree of
Master of Science

April 2023

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Eidesstattliche Versicherung

Nachname: Bhat Vorname: Rakshit

Matrikelnr.: 6918604 Studiengang: Informatiks

○ Bachelorarbeit ◉ Masterarbeit

Titel der Arbeit: Development and Evaluation of a Model-based UI Protoyping Experimentation Approach

☒ Die elektronische Fassung ist der Abschlussarbeit beigefügt.

☐ Die elektronische Fassung sende ich an die/den erste/n Prüfenden bzw. habe ich an die/den erste/n Prüfenden gesendet.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit (Ausarbeitung inkl. Tabellen, Zeichnungen, etc.) selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Abschlussarbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Die elektronische Fassung entspricht der gedruckten und gebundenen Fassung.

Belehrung

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist die Vizepräsidentin / der Vizepräsident für Wirtschafts- und Personalverwaltung der Universität Paderborn. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz NRW in der aktuellen Fassung).

Die Universität Paderborn wird ggf. eine elektronische Überprüfung der Abschlussarbeit durchführen, um eine Täuschung festzustellen.

Ich habe die oben genannten Belehrungen gelesen und verstanden und bestätige dieses mit meiner Unterschrift.

Ort: Paderborn Datum: 17.03.2023

Unterschrift: RBhat

Acknowledgements

And I would **like** to acknowledge . . .

‘’

Abstract

The user interface (UI) layer is one of the essential aspects of software applications since it associates end-users with the functionality. For interactive applications, the usability and convenience of the UI are essential factors for achieving user acceptability. Therefore, the software is successful from the end user's perspective if it facilitates good interaction between users and the system.

Table of contents

List of figures

List of tables

Listings

Chapter 1

Introduction

In this thesis, we explain our approach to improving User Interface (UI) prototyping and UI experimentation. 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], further increasing 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 [2]. 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 [3]. This ensures that the developers with high uncertainties in the early product development phase can improve the product by testing the underlying assumptions [4]. 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 [5]. 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 [6]. 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 [5]. 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 [7]. 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 [8]. 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.

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

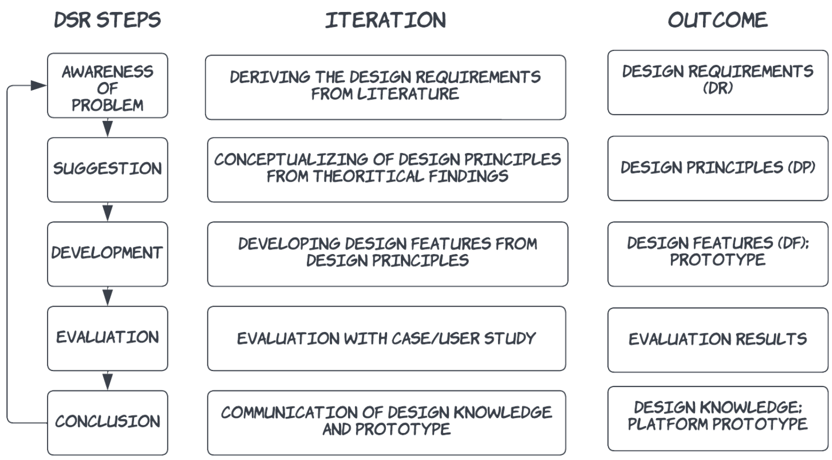


Fig. 1.1 Design Science Research Cycle [9]

Prescriptive knowledge about the design of artifacts, such as software techniques, models, or concepts, is what Design Science Research (DSR) aims to provide. Due to this design knowledge, future projects can design artifacts methodically and scientifically with the aid of study and practice [9]. Therefore, we will conduct the *first* iteration of the 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 (DP) defined for the whole process of experimentation [9]. In this design, the product designers will iteratively validate their prototypes with the users or the participants. Here, DPs capture and codify that knowledge by focusing on the implementer, the aim, the user, the context, the mechanism, the enactors, and the rationale [10]. 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 [9] consisting of five iteratively conducted steps (see figure 1.1). As a result, this design and application may provide design-focused information that adds to the DSR knowledge corpus [11]. Every element within a DSR project is built upon and systematically analyzed to add to the overall DSR knowledge corpus. 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.

Design Requirements: In DSR, abstracted Design Requirements (DR) refer to the general, high-level requirements that a design must meet to succeed [12]. These requirements are typically derived from the RQ identified as relevant to the problem. Abstracted DRs provide a broad framework for the design process and help to ensure that the design solution addresses the key issues and challenges identified in the research problem. They can also help guide the evaluation of the design solution and ensure that the solution is grounded in the design principles identified as necessary.

Design Principles: In DSR, concrete DPs refer to specific, detailed guidelines that a design must follow to be successful [12]. These principles are derived from the abstract DRs identified as relevant to the research problem and provide more specific guidance on how the design should be implemented. DPs can also be defined as the codification of our knowledge during the design study while identifying the DRs. Concrete DPs are essential in DSR as they help ensure that the design solution meets the needs and goals of the research problem and is grounded in the DPs we identified as necessary.

Design Features: In DSR, Design Features (DF) refer to the characteristics of the artifacts or solutions created through the process [12]. These DFs provide a means of evaluating the solution's effectiveness and can serve as guidelines for others who may want to adopt or modify the solution. DFs typically include both functional and non-functional aspects of the solution. Functional features relate to the specific capabilities and functionalities that the solution provides whereas, non-functional features relate to characteristics such as usability, scalability, etc. By explicitly specifying these DFs, DSR researchers can ensure that their solutions are aligned with the stakeholders' needs and meet the necessary standards of quality and effectiveness.

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 [13]. 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 a 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 [14]. 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 [15]. 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.

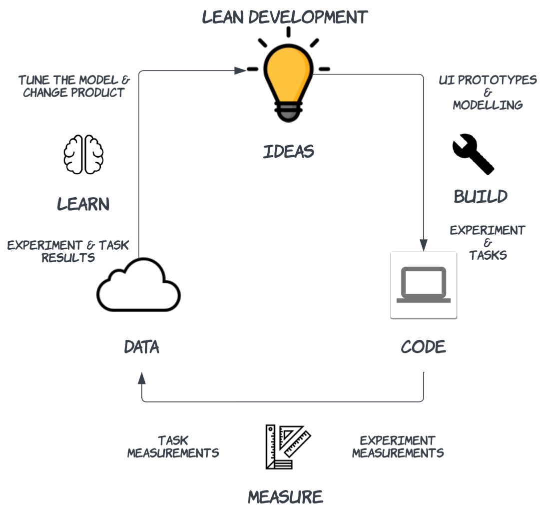


Fig. 1.2 LEAN Development Technique

In our solution, we use the LEAN development technique (see figure 1.2) for development as it is used to develop user-friendly products [16]. 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, UI experiments, and user tasks. In the *(2) Measure* phase, we plan to assign the experiments and tasks to the users and measure the task and the experimental measurements and perform some analysis on the data received. And finally, in the *(3) Learn* phase, we display the analysis results, tune our models to decide the better variant among the others and modify the prototype. As per figure 1.2, we complete one cycle of iteration and start a new one with the updated prototype.

1.5 Thesis Structure

The structure of our thesis is designed to provide a comprehensive overview of our research and solution approach. The *(1) Introduction* chapter includes a detailed analysis of the problem statement, motivation, research question, and our research and solution approach.

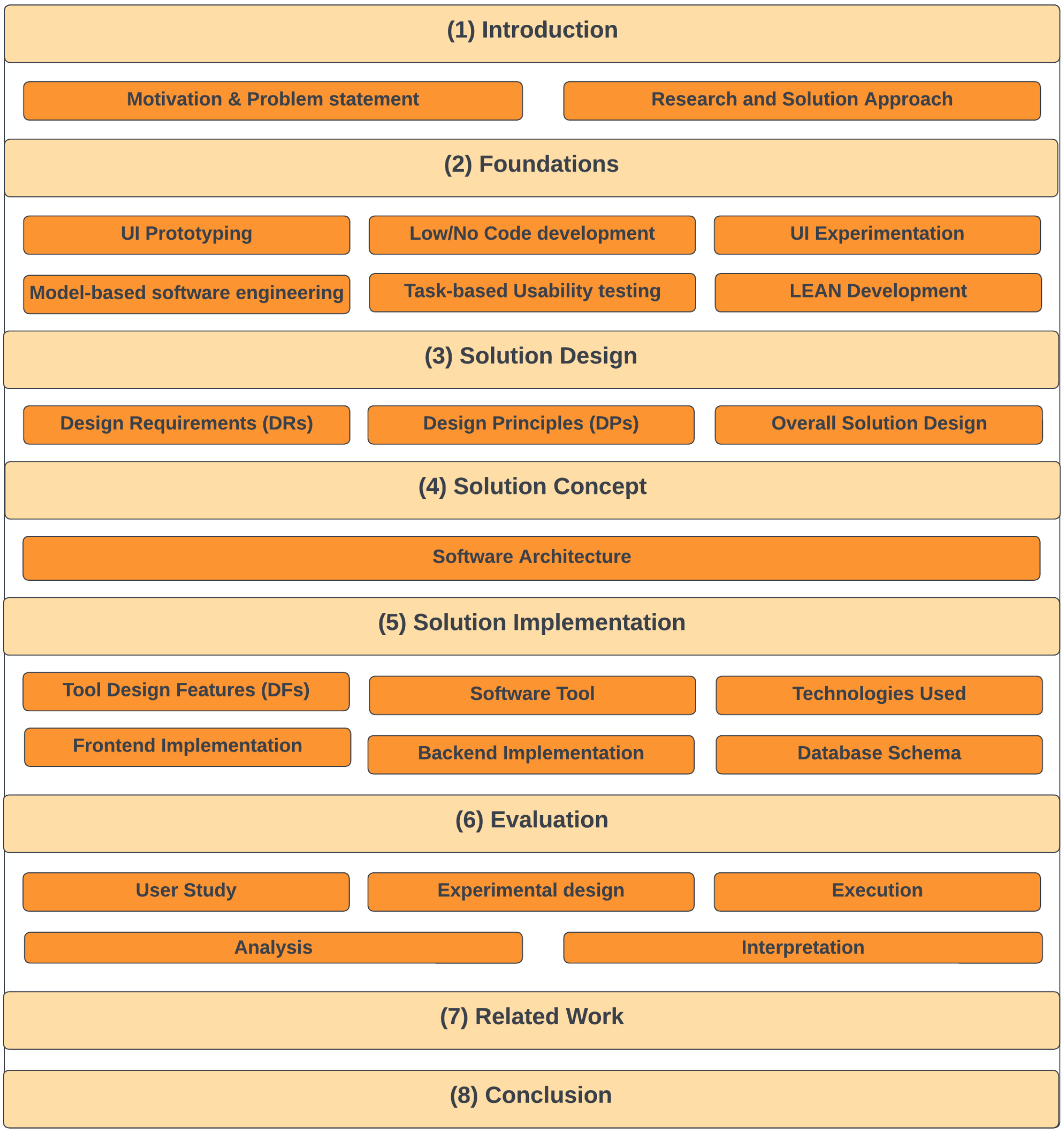


Fig. 1.3 Thesis Structure

Next, the *(2) Foundations* chapter is used to gain the knowledge required to understand the scope of our work, which includes UI prototyping, low/no code development, model-based software engineering, task-based usability testing, UI experimentation, and LEAN development. The *(3) Solution Design* chapter outlines the design requirements, design principles, and overall solution design. Similarly, the *(4) Solution Concept* chapter presents the software architecture, UI prototype management, UI experimentation management, persistence infrastructure, deployment infrastructure, and security infrastructure. Next, the *(5) Solution Implementation* chapter provides detailed insights into the tool design features, explains the software tool we developed, technologies used, frontend and backend implementation, and database schema. Next, the *(6) Evaluation* chapter showcases the user study, experimental design, execution, analysis, and interpretation. Next, the *(7) Related Work* section discusses current tools and state-of-the-art technologies and compares them with our DRs. Finally, in the *(8) Conclusion* chapter, we summarize our work, future work, and research contributions.

Chapter 2

Foundations

In the previous chapter, we defined the problems for software development and presented an overview of our solution. Consequently, this chapter is built upon the existing knowledge and theories developed in the field of Software Engineering. This chapter provides an overview of the relevant literature and identifies the key concepts and theories necessary for the research. Firstly in section 2.1, we will discuss *UI Prototyping* explaining the creation of a UI simulation and testing for refining the design ideas and user requirements. Then, in section 2.2, we explain *Low/No-code methods* and citizen development of the software products. Then, in section 2.3, we describe the *Model-Based Software Engineering (MBSE)* approaches and meta-models. In section 2.4, we clarify *Task-Based Usability Testing* for reviewing and refining the product. Later in section 2.5, we define the concept of *UI Experimentation* and *A/B testing*. Finally, in section 2.6, we explain the *LEAN software development process*, explaining the *Build, Measure* and *Learn* phases.

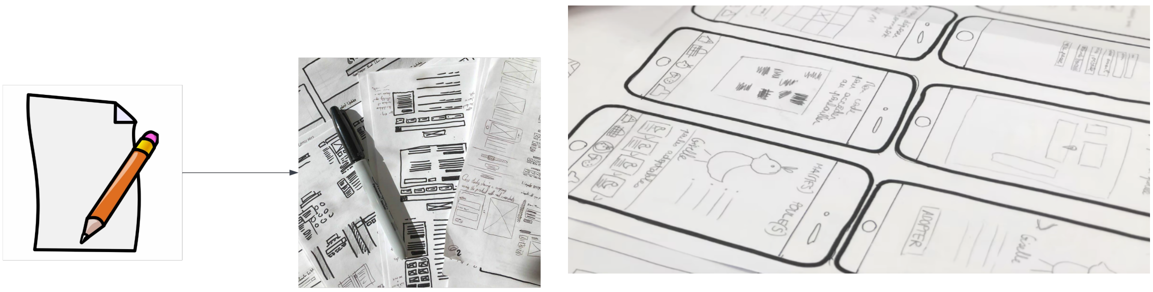
2.1 UI Prototyping

UI prototyping creates a simplified UI version to test and iterate on design ideas before building the final product. UI prototyping is a valuable tool for designers, allowing them to quickly and easily test and refine UI designs [17]. It also helps gather feedback from users and stakeholders [18] and identify any usability issues before investing significant time and resources into the final product. From paper to Hypertext Markup Language (HTML) code, everything could be a prototype, including various techniques like `Paper Prototypes`[1],

[1]**Paper prototyping:** It is a method of creating a preliminary version of a UI using paper and pen.

`Wireframes`[2], `Mockups`[3] and `Interactive Prototypes`[4]. Moreover, UI Prototypes can be classified as *High-Fidelity* and *Low-Fidelity* prototypes. The comparison by Jim Rudd et al. [19] on high and low-fidelity prototyping, based on its advantages and disadvantages, is explained further.

Low-Fidelity Prototypes: *Low-fidelity* prototypes (see figure 2.1) are prototypes usually with limited functions and little interaction prototyping effort. They mainly focus on explaining concepts, design alternatives, and screen layouts. Storyboard presentations, cards, proof of concept prototypes, paper prototypes, etc., come under this category. For Instance, paper-based prototypes are simple, low-fidelity mockups that can be quickly and cheaply created using paper and pencil (see figure 2.1a). In this case, the UI designers use simple text, lines, and forms to hand-draw concepts. Instead of aesthetics, the focus is on speed and creative ideas. To simulate user flows (as shown by figure 2.1b), designers lay paper screens on the floor, table, or pinned to a board. Therefore, these prototypes emphasize on communicating, educating, and informing rather than training, testing, and codification [20]. The advantages of low-fidelity prototypes are rapid development, lower development cost, addressing issues, and usefulness for a proof-of-concept [19]. Similarly, the disadvantages include limited error checking, difficulty with usability testing, navigation, flow limitation, etc.



(a) Sketches and Whiteboard for Prototyping

(b) Example on How the Paper-Based Prototypes Are Developed [21]

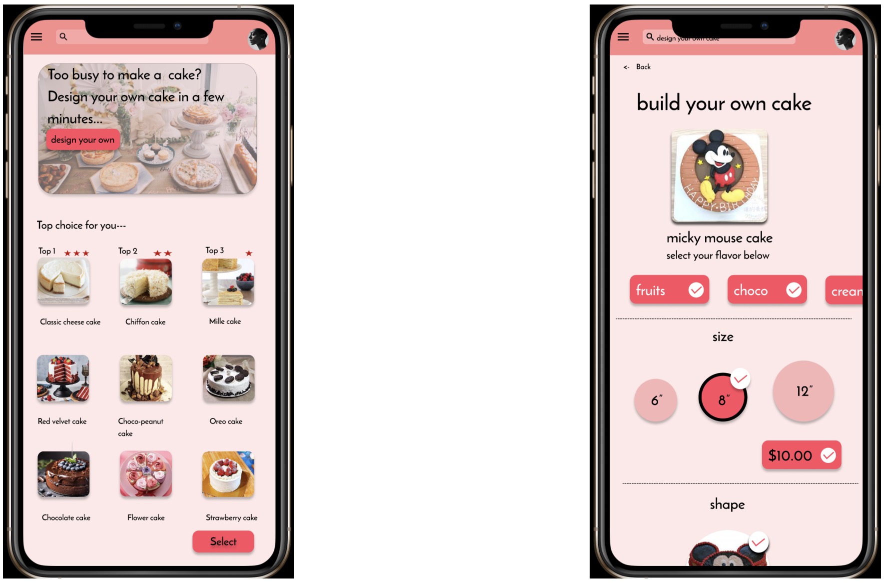
Fig. 2.1 Low Fidelity Prototyping

[2]**Wireframe:** A wireframe is an essential visual representation of the layout and structure of a product, showing the placement of text, images, and buttons.

[3]**Mockup:** A mockup is a more detailed version of a wireframe, typically created in color and with more realistic graphics and images.

[4]**Interactive Prototype:** An interactive prototype is a fully functional representation of a product, design, or feature.

High-Fidelity Prototypes: Contrary to low-fidelity prototypes, *high-fidelity* prototypes (see figure 2.2) have full functionality and focus on logical flow between screens, and the user models of the system [22]. These prototypes are detailed, interactive mockups of a UI designed to mimic the look and feel of the final product closely. High-fidelity prototypes are often created using interactive prototyping tools, which allow designers to create realistic, interactive mockups of UI designs [23]. These tools typically include a library of UI elements and templates and the ability to create interactive transitions and animations. The users can operate these prototypes, and the developers can collect information from the users through measurements. Some advantages of high-fidelity prototypes are that they are user-driven, used for navigation and tests, and can also serve as a marketing tool for attracting potential customers [19]. Therefore, these prototypes are typically more realistic and interactive than those created using other tools but may require more time and resources to develop and maintain. At the same time, high-fidelity prototypes allow designers and developers to test and refine complex UI concepts and interactions, ensuring that the final product is user-friendly and practical. Overall, low-fidelity prototypes and high-fidelity prototypes are both valuable tools in the design process, and the choice between the two will depend on the specific needs and goals of the project.



(a) Prototyping Page of a Cake Company (b) Prototyping Page to Customize a Cake

Fig. 2.2 High Fidelity prototype: Model-Based UI Prototyping [24]

UI prototyping is an evaluation and testing technique according to User-Centered Design (UCD) methodology since the 1990s [25]. The evaluation of prototypes by users gives crucial feedback in iterative approaches for Information Technology (IT) project management, especially agile methodologies [26]. Therefore, to build an exemplary UI, a company can use this approach: develop a preliminary version of the UI, test it with people, and make as many revisions as possible (without building the actual software) [17]. Figure 2.3 shows a cycle representing the iterative development with prototypes. The designers start the process by developing the UI prototypes, which the stakeholders (e.g., customers and product managers) review. The UI prototypes are refined from the feedback received, reiterating the cycle. Therefore, designing UI prototypes enables designers and stakeholders to communicate more effectively. Similarly, an interactive prototype helps visualize design concepts and

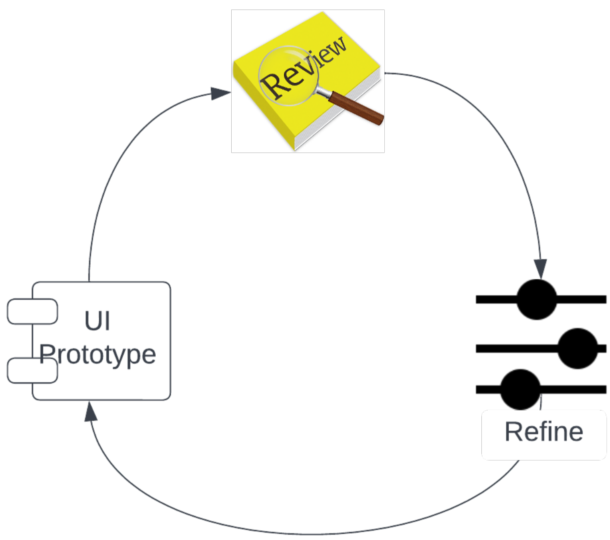


Fig. 2.3 Prototyping Steps for an Iterative Development

communicate new requirements and expectations about a prospective system. Iterative design requires multiple updates to the design's execution.

Since developing and updating the entire software system is complex and expensive, prototyping is a crucial step [27] in software development. Simultaneously, prototypes might exclude many requirements, making the software more accessible, smaller, and less expensive to construct and change [27]. Thus, the main difference between a prototype and a software application is that in a prototype, the screens are designed images with no additional capabilities to display the design and flow. Moreover, usability testing to validate user requirements and prototype functionality is a part of the evaluation process for UI prototypes [28]. Thus by using prototyping, there is usually more contact between the designers and users, resulting in fewer usability flaws and corrections at the end of development. And finally, these mockups are then converted into actual UI elements in a software application, and a logical flow is established.

Overall, UI prototyping is an essential part of the design process. It allows designers and developers to quickly test and refine UI concepts, gather feedback from stakeholders and users, and identify any usability issues before investing significant time and resources into building the final product. Whether using low-fidelity prototypes to test basic layout and navigation concepts or high-fidelity prototypes to test complicated UI concepts and interactions, prototyping can help ensure that the final product is user-friendly and effective. By iterating and refining their prototypes throughout the design process, designers can create better, more intuitive products that meet the needs of their users.

2.2 Low-Code / No-Code Development Platform

Low-code is a software development method that uses less human coding to enable users to construct and manage programs efficiently rather than writing extensive amounts of code [29]. It is a technique used by developers to help non-developers design and develop software applications using a *Graphical User Interface (GUI)* supported by a *Low-Code Development Platform (LCDP)*. An LCDP allows developers to create, deploy, and manage applications quickly and easily using high-level programming languages and techniques such as model-driven and metadata-based programming [30]. It simplifies building and deploying applications using declarative programming abstractions and one-step deployments. LCDP provides pre-built components, templates, and other resources that companies can quickly assemble to create functional applications. These platforms are designed to accelerate the development process and enable companies to quickly build and deploy custom software solutions. Similarly, there is another technique called *No-code development* supported by the *No-Code Development Platform (NCDP)* [31]. Unlike low-code, no-code platforms require no programming skills because they offer drag-and-drop techniques for building the apps. The non-developers can easily pick up the components which fit the UI and drag-and-drop them to the screen and finally create an entire application using this technique. Using the UI and ready-made automatic tools on these application development platforms, it is feasible to create apps relatively quickly. Due to its simplicity, flexibility, and low cost, companies have started using this platform to meet the high demands of software development and digitalization [29]. Additionally, with its self-configurable components, it lowers the expenses associated with initial installation, training, distribution, and maintenance [32].

High-Level Architecture of LCDP: As shown in figure 2.4, an LCDP is divided into small modules independent of the other components. Bock et al. [30] have classified these modules according to the three main approaches to systems development: *the static perspective*, *the functional perspective*, and *the dynamic view*.

Static Perspective: LCDPs typically allow data to be stored either in an internal Database Management System (DBMS) or in external systems (see the first part of figure 2.4). Many of these features/components commonly found in LCDPs fall under the *static perspective*. Similarly, most LCDPs include a component for defining Data Structure (DS), usually provided as a conceptual modeling tool that uses a classical Data Modeling Language (DML) such as the Entity-Relationship Model or a Domain Specific Language (DSL). Some LCDPs allow DS to be defined only through UI-based dialogs or lists. For example, you are allowed to upload a Comma Separated Value (CSV) file that contains data that is persisted into the

Database (DB) as per the columns in the file. A common feature of LCDPs is the ability to access external data sources using various Application Programming Interfaces (API).

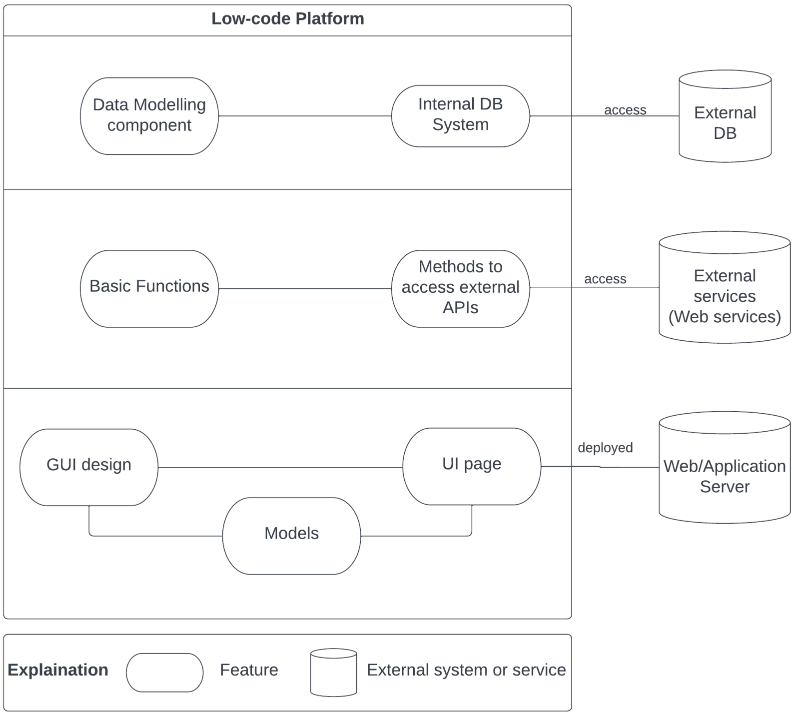


Fig. 2.4 High-Level Architecture of Low Code Development Platform

Functional Perspective: LCDPs also offer basic functional specifications (see the second part of figure 2.4). These typically include simple expression languages for decision rules and dialog-based methods for specifying program flow conditions. Each solution consists of a library of generic standard operations, such as mathematical functions. For example, LCDPs will enable the use of traditional approaches such as web services and Representational State Transfer (REST) services, and many modules provide support for a wide range of APIs from individual providers, such as Google APIs, and social media APIs [30].

Dynamic Perspective: The LCDP also includes a GUI designer module (see the third part of 2.4), which allows for the development of GUIs and their integration with other implementation elements. The GUI designers module specifies pre-defined widgets, although the range varies. It is generally easy to link GUIs and DSs in most of these platforms, as it is optional to implement the Model-View-Controller (MVC) pattern manually. In addition, many of these platforms offer specific support for adapting the GUIs to different target environments, such as desktop browsers, tablets, and smartphones [33]. Another common feature is the inclusion of a component for defining roles and user rights, which is usually part of the platform's governing architecture and is deployed along with the custom application [29].

Main Steps Of ***Low-Code*** *App Development* as per [34]:

Building: In this step, the platform gives you the freedom to alter the provided code and add hand-written custom code to it to specify more complex features in the app as you create the app step-by-step using visual editors and drag-and-drop interfaces. Modules, components, and chart-builders are already incorporated into low-code applications. Charts may be used to display data from modules, while modules are used to specify the type of data that will be stored in the app. Components and pages provide the type of user experience the app will have. These platforms also have a provision for automating repetitive tasks in the app.

Testing: Testing a software application is an integral part of the development cycle. However, the low-code development platform decreases the requirement for testing. Pre-build modules and components on low-code platforms are created with a certain level of application security. The developers of the low-code platform constantly monitor these modules, and they have previously gone through several unit tests. But, in a low code platform, testing can be performed in several ways [35]:
- *Automated testing:* Some low-code platforms have built-in support for automated testing, which allows developers to create and run tests that validate the functionality of their applications.
- *Manual testing:* Even with automated testing, it is necessary to perform manual testing to ensure that an application functions correctly. In a low-code platform, manual testing can be achieved by developers or by dedicated testers who are responsible for verifying the functionality of the application.
- *User Acceptance testing (UAT):* In some cases, it may be necessary to involve end users in the testing process. This can be done through user acceptance testing, which consists in having end users test the application and provide feedback on its functionality and usability.

Overall, testing in a LCDP involves a combination of automated and manual testing, as well as performance and acceptance testing [35], to ensure that the application is functioning correctly and meets the needs of end users.

Deploying: In this step, the application is deployed across apps and to the final users. In LCDP, the packages for installation, configurations, and application setup are included. In terms of deployment, most of the considered solutions offer advanced support, although the specific forms vary. Some systems require the low-code platform environment to be installed on a web server to deploy individual applications. In contrast, others allow the developed solutions to be deployed as self-contained applications on various devices and machines. This means that the LCDP gives freedom to the users in deploying the applications for the customers with just one click.
Docker[5] is a widely used open software platform that allows applications to be deployed as containers. It supports libraries, system tools, code, and various runtimes, making it possible to build, test, deploy, and scale applications across multiple environments. In an LCDP, docker can be used to securely deploy frameworks, services, and platforms in separate containers that communicate with each other using protocols such as Message Queue Telemetry Transport (MQTT)[6] and REST[7]. Additionally, a variety of options are provided for developers with little programming experience, those with coding expertise and seasoned programmers who wish to expand the functionality of the current design [29].

Overall, low-code/no-code development platforms are valuable for organizations looking to build and deploy custom applications quickly and efficiently that can help organizations of all sizes and industries bring their ideas to life.

[5] Docker: https://www.docker.com/
[6] MQTT: The Standard for IoT Messaging https://mqtt.org/
[7] REST API (also known as RESTful API) https://www.redhat.com/en/topics/api/what-is-a-rest-api

2.3 Model-Based Software Engineering

MBSE refers to maintaining and developing software while reusing existing code. Similarly, Model Driven Software Engineering (MDSE) is the term used to cover various techniques for creating software using codified models. At the same time, for creating models, the Model Object Facility (MOF) has defined four levels separating the reality, models, meta-models, and meta-meta-models (as shown in the figure 2.5).

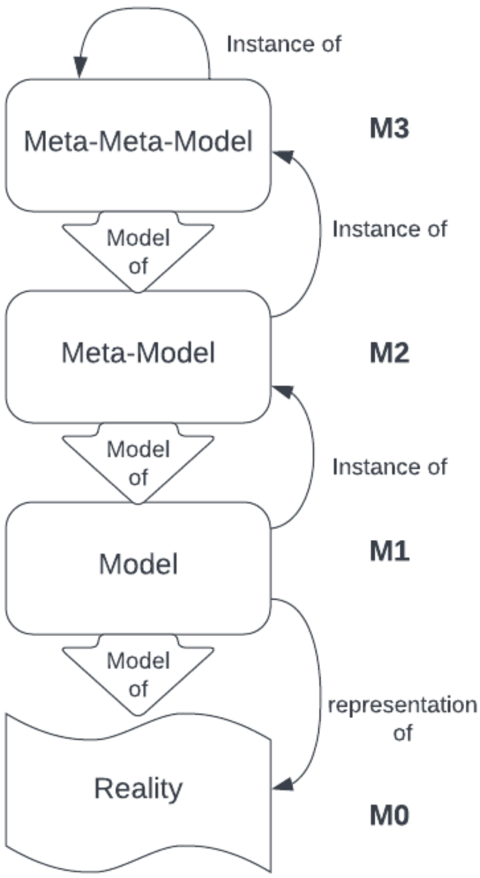


Fig. 2.5 Model Object Facility (MOF) levels

MOF is a standard that defines a meta-model for modeling information. It provides a common framework for creating and using models representing a system or part of a technique used to analyze design or document the procedure. MOF is part of the Object Management Group (OMG) [36] standards and is used in several different contexts, including software engineering, business process modeling, and enterprise architecture. It is based on the Unified Modeling Language (UML)[8] standard notation for modeling software systems. MOF defines a standard set of concepts and notations for creating and using models, which can represent different aspects of a system. This includes concepts such as classes, attributes, operations, and relationships, which describe the system's structure and behavior.

Meta Models: A meta-model is a model of a model or a simplified version of an actual model of a system of interest or a software application that formally represents the structure and behavior of a particular model type in a formal and standardized way. Meta models are often used in Software Engineering (SE) to describe the structure and behavior of DSL or other modeling languages [37]. They can be used to understand how a model works or to improve the performance of a model by making it more efficient or accurate. Meta models are often used in Model Driven Development (MDD), focusing on creating and using high-level abstractions to design and implement software systems [38]. By using meta-models, developers can create a high-level view of a system and then use that view to generate code automatically, which can help improve the development process's efficiency and reliability. A model is usually defined as an abstraction of real-world entities, and a meta-model is another abstraction of the models. Similarly, metamodelling is analyzing and developing the rules, theories, and models helpful in constructing meta-models.

The **M0 layer** refers to the physical system or systems that are being modeled. It represents the *Real World* or the physical components, subsystems, and techniques that comprise the overall system being designed or analyzed. It is the foundation upon which the other layers of the MBSE models are built.

The **M1 layer**, built on top of the M0 layer, represents the functional requirements and system behavior. The functional requirements include system inputs, outputs, and interfaces, and the system's behavior includes its response to different inputs and conditions.

The **M2 layer**, built on top of the M1 layer, represents the system's architecture and design. This layer describes how the system's components, subsystems, hardware, and software interact. The M2 layer includes detailed specifications and design constraints such as interfaces, protocols, and data structures.

Finally, the **M3 layer** represents the system's implementation and verification. It includes the system's specific design details and implementation plans, including the detailed design of its components, selecting particular hardware and software, and testing and validation plans. This layer ensures that the system is built according to the specifications and design constraints defined in the M2 layer and meets the functional requirements and behavior described in the M1 layer.

[8] Unified Modeling Language https://www.uml.org/

After considering the information above, the meta-model can be further developed and refined. There are two approaches to designing a meta-model: *Top-down and Bottom-up* [36]. In the top-down method, we start with the overall process and then break it down into smaller steps. In the bottom-up approach, we begin with the individual steps and group them into more extensive procedures. *Top-down development* of a meta-model involves starting with a high-level representation of the structure and behavior of the models that the meta-model will describe (see figure 2.5). This high-level representation (M3 from figure 2.5) might include the overall design of the models (e.g., the types of elements and relationships that are allowed) and the rules and constraints that must be followed when creating and using the models. Once the high-level structure of the meta-model has been defined, it can be further refined and expanded upon by adding lower-level details (M2 and M1 and M0 levels). They include the attributes and operations allowed for each element, how elements can be related, and the semantics of the relationships between elements.

Bottom-up development of a meta-model involves starting with the specific elements and relationships that will be included in the meta-model and gradually building up to a higher level of abstraction. Once the lower-level details of the meta-model have been defined, they can be organized and grouped into higher-level concepts and structures, such as classes, packages, or packages of packages, to form the overall design of the meta-model [36].

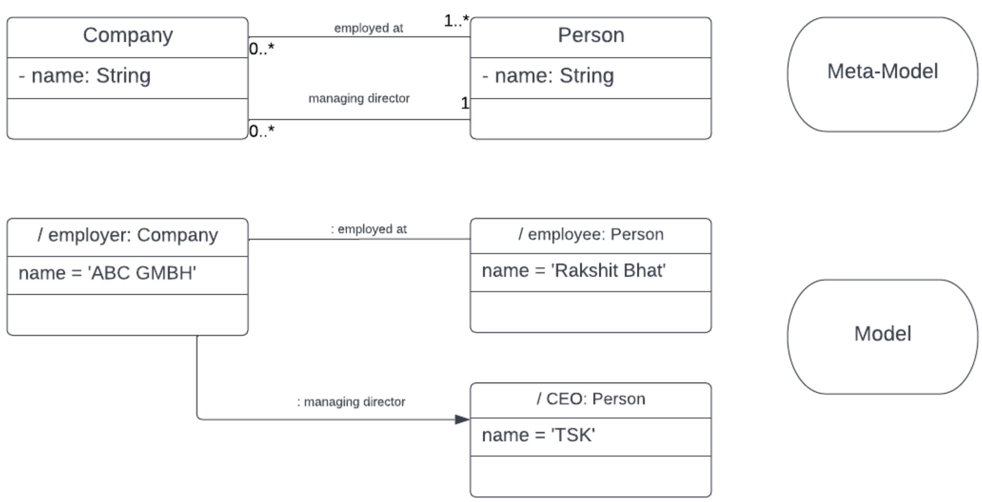


Fig. 2.6 Meta Models and Models

Models: As shown in figure 2.5, a *Model* is a simplified representation of the real world or reality. Before any code is written, the model is a schematic that describes how the software system should function. In software development, a model is a representation of a software system. It can be used to visualize the structure and behavior of a system or to analyze the system for potential issues or improvements. Models can take many forms, such as diagrams, graphs, or mathematical equations, and they can be created using various tools and techniques. Models are often used in MDD [38], focusing on creating and using high-level abstractions to design and implement software systems. In this approach, models are used as the primary source of design and implementation information, and the code is generated from the models using a code generator [39]. By using models, developers can better understand a system and identify potential problems or areas for improvement before they start writing code. From the example 2.6, we need to create models for every entity of reality (e.g., An employee and a CEO both are *Person* in meta-model but are separate entities in models). Using models, various adaptive model-driven user interface development systems are developed [40] and the authors defined twenty properties challenges for the Model-driven User interface and compared some tools that implement these properties. Similarly, modeling is a process and method of building models for some purpose or related to some domain. Therefore, models and modeling approaches are used to codify the UIs in different companies.

MBSE is a powerful approach to developing software that relies on using models to represent and analyze software systems. By using models to represent the various components and relationships within a software system, developers can more easily understand and analyze the system and make changes and updates more efficiently. However, these models do not emphasize offering visual notations to aid non-developers in creating such interfaces. Therefore, for our research, we use a recent method [39], which illustrates how to use low-code and Model-driven approaches to close the gap between designers and developers.

2.4 Task-Based Usability Testing

The main focus of usability testing is that seeing someone use an interface is the best approach to determine what functions well and what doesn't. It would help if you offered the participants some assignments to observe them. The word "task" is commonly used to describe these assignments. Assigning tasks to the accurate number of participants can help determine the quality of the UI and the problems faced by the users. Overall, the UI design can be improved using the participants' feedback. Task-based usability testing is one way to determine the software's overall usability [41] by measuring the percentage of the tasks the users complete. These tasks need to be some scenarios, not just "*do something*", because it sets the users a stage for *why* they would perform the tasks. To get qualitative feedback from the participants, in [42], the authors provide *three good practices* and task-writing tips for designing better task scenarios.

***(1) Make the Task Realistic*:** So, the participants should be able to execute the tasks which could be completed efficiently and with the freedom to make their own choices. The participants will attempt to accomplish the assignment without genuinely interacting with the interface if you ask them to do something they wouldn't typically do. Therefore, it is necessary to create realistic tasks.
E.g., from *Videostreamer* application: The goal is to offer some movies that the user should watch.
Bad task: Watch a movie if the actor 'Mr. T' and actress 'Ms. K'.
Good Task: Watch a movie with more than 6.5/10 ratings.
In the example, the participants should be free to compare movies based on their criteria.

***(2) Make the Task Actionable*:** Here, the participants should be told what they need to do rather than how they would do it. These types of tasks help us determine if the task isn't actionable enough. If the participants tell the moderator they cannot determine if they need to click on the link or if they are finding it hard to decide the next steps in a specific task, then it is a sign that the task needs to be more straightforward and actionable.

From *Videostreamer* application: The goal is to find a movie and show times.
Bad Task: You want to watch a movie Sunday afternoon. Go to the app and tell where you'd click next.
Good Task: Use the app to find a movie you'd be interested in watching on Sunday afternoon.

***(3) Avoid Giving Clues and Describing the Steps*:** There are frequent hints about how to use the interface or the software in step explanations. These tasks must be more balanced with the users' behavior and give less valuable results. The participants should expose the navigation and some features on their own, giving accurate feedback about the interface. But, at the same time, we should try to include the words used in the UIs as they help the users navigate smoothly and would not lead to some confusion.

From *Videostreamer* application: The goal is to change the user's movie preferences.
Bad Task: You want to change your movie preferences. Go to the website, sign in, and change your movie preferences.
Good Task: Change your movie preferences from 'action' to 'comedy'.

Therefore, it is crucial to create a realistic test environment during usability testing [41]. It is also essential to provide the information required for the participant to complete the task without guiding them on specific actions. If the task scenario is unclear enough, the participant may ask for more information or confirm that they are on the correct path.

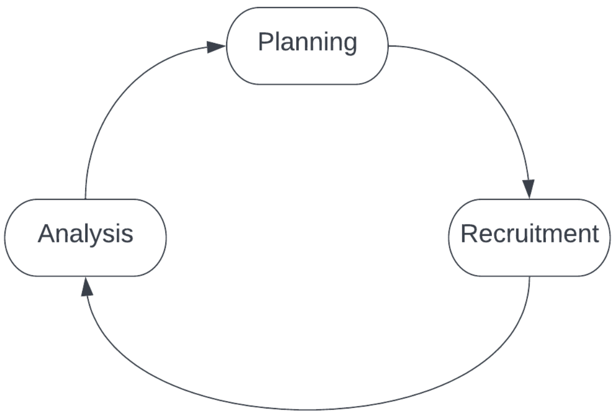


Fig. 2.7 Task-Based Usability Testing Steps

The fundamental principle of usability testing is to have actual users attempt to carry out essential tasks or assignments on your software, websites, mobile applications, or gadgets. For that, we divide the tasks into three steps.

Planning: In usability testing, planning is an important step. In this step, we decide on a process that fits our research question, hypothesis, and metrics. It is beneficial if we use mixed-methods services for a task-based usability study. The tasks and scenarios must be well-defined in the planning phase. At the same time, we can create pre-study and post-study questions.

Recruitment: In this step, the users are assigned to the task as it is essential to select the correct number of user group sizes. There should be a proper way to interview the users before assigning them tasks so that the participants understand them correctly. The participants should also be able to ask questions or doubts while performing the honorarium tasks.

Analysis: Task-based studies analyze metrics, issues, and insights in great depth. For metrics, we calculate the task completion rates, task time, and task and test level perception questions. The problems that the participants encounter while performing the tasks need to be reported automatically to the development teams by sending screenshots, quotes, etc. There should also be a section to analyze some insights about the software that has worked well for the users while performing the tasks.

Overall, task-based usability testing is a tool for evaluating the usability and effectiveness of software systems. By focusing on specific tasks that users need to complete, we can gather valuable insights into how well the system supports these tasks and identify areas where we could improve the user experience.

2.5 UI Experimentation

In this section, we discuss the role that experimentation plays in the software development process and how designers can "prototype with real data" to improve the usability of the UI. Experimentation is a vital part of the design process for UI products. It involves testing different design solutions and variations to see which performs best in terms of usability, aesthetics, and other desired characteristics [6]. We can do this through various methods, such as usability testing, A/B testing, and rapid prototyping. Experimentation allows designers to iterate and improve their designs by creating different *'Software Variants'* and gathering data and feedback from real users. It is an important part of creating user-centered designs that effectively meet the needs and expectations of the target audience [5]. Experimentation helps product teams test out ideas early in the process with real-world consumers rather than settling on a single solution and executing it in the final phase [43].

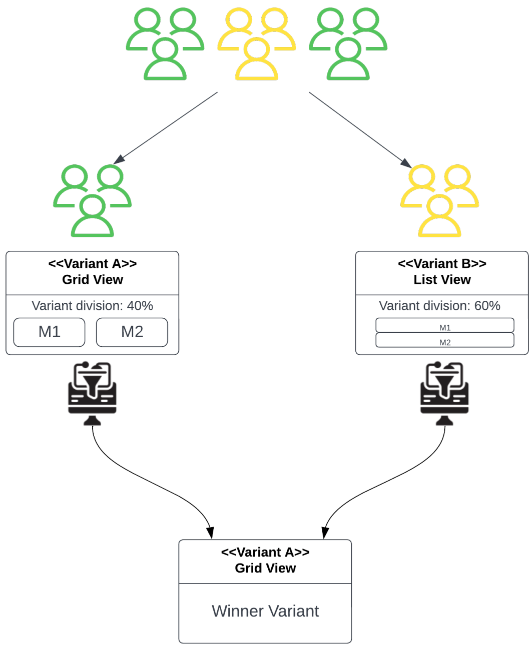


Fig. 2.8 A/B Testing

For conducting experimentation, various steps like *Users distribution*, *Continuous Experimentation (CE)*, *Variants distribution* exist.

User distribution: To conduct a successful experiment, the size of the study, or the number of participants, must be considered first. Statistics suggest that the more people you include in the investigation, the greater its statistical power, which impacts your level of confidence in your findings [44]. Then, the participants should be allocated into groups at random. There are several levels of treatment given to each group (e.g., some prerequisites to the participants before experimenting). For assigning the subjects to groups, users are divided into a *between-subjects design* vs a *within-subjects design*. In a between-subjects design, individuals receive only one of the possible levels of an experimental treatment whereas, in a within-subjects design, every individual receives each of the experimental treatments consecutively, and their responses to each treatment are measured. In figure 2.8, a between-subject design is used where only one of the variants is assigned to the user.

Continuous Experimentation and Variants distribution: CE primarily aims to get users' feedback on the software product's evolution. As per figure 2.8, CE generally uses A/B testing in a primary case of comparing two variants, A and B, which are controlled and test variables in an experiment. First, the users or the participants of the study are separated into groups and are assigned one of the two variants. Since we have `GridView` and the `ListView`, one group of the users are assigned with List and one with Grid View (see figure 2.8). Then both users would be given some tasks (see Section 2.4), and the analysis of these tasks gives the winner among the variants. And in the end, developers make evidence-based decisions to direct the progress of their software by continuously measuring the results of multiple variants performed in an experimental context with actual users [45]. CE is an extension to the introduction of continuous integration and deployment, and all are summarized as constant software engineering [46]. Additionally, continuous experimentation can help designers and developers stay up-to-date with the latest design trends and best practices. By regularly testing and iterating on the design, they can learn from the feedback they receive and incorporate new ideas and techniques into their design. This can help keep the product or service relevant and competitive [6].

Overall, CE is a critical approach to take when developing UIs for products and services. It allows designers and developers to gather valuable feedback and data and use it to improve their products' design and User Experience (UX).

2.6 LEAN Development Process

LEAN development is a software development methodology that emphasizes continuous improvement and eliminating waste to deliver customer value as efficiently as possible. It is one method within Agile development [47]. It is based on the principles of the LEAN manufacturing method, which was developed by *Toyota* in the 1950s and aims to eliminate waste and maximize value in manufacturing processes [48]. LEAN development's primary objective was to reduce loss, minimize waste, and encourage sustainable production. And therefore, as an Minimum Viable Product (MVP), in LEAN development, the product has the essential elements necessary to launch successfully and does not have to include any additional components.

In software development, we can apply LEAN principles to various aspects of the development process, including requirements gathering, design, coding, testing, and deployment [47]. The goal is to minimize waste and optimize the use of resources to deliver high-quality software products that meet the customer's needs.

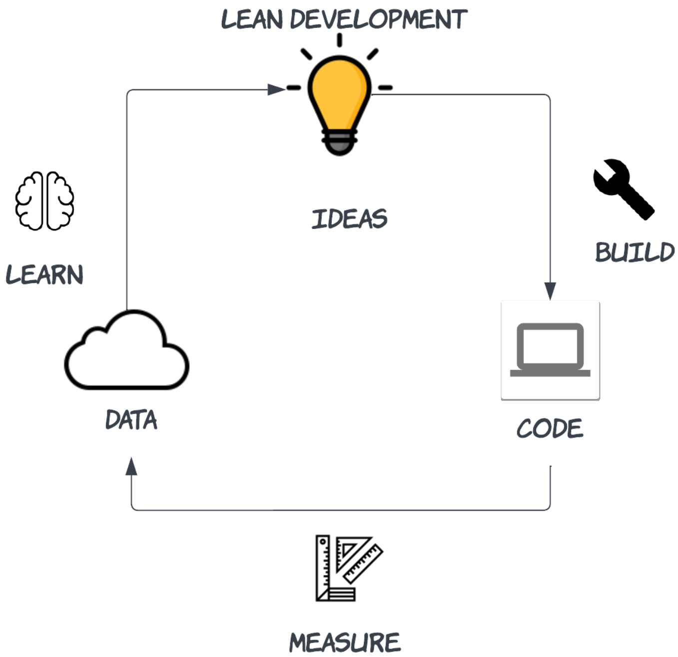


Fig. 2.9 LEAN Development Cycle[6]

[6]Image adapted from website: https://www.agile-academy.com/en/agile-dictionary/lean-development/

LEAN Development Cycle: The LEAN development cycle often includes three phases: *build, measure, and learn* (see figure 2.9). Steps like these are designed to help the development team continuously improve the product and development process.

Build: In the build phase of LEAN development, the development team works to build and deliver small increments of working software to the customer. This involves gathering requirements, designing, coding, testing, and deploying code [47]. The build phase's goal is to deliver customer requirements as quickly and efficiently as possible. To do this, the development team may follow model-driven development [49], continuous integration and delivery to ensure that the developed code meets the desired functionality and performance requirements.

Measure: In the measure phase of LEAN development, the development team collects data on the product's performance and the development process to identify areas for improvement [47]. It involves collecting data on customer usage and feedback, as well as data on the performance and efficiency of the development process. The measure phase aims to understand how the product is being used and its impact on the customer. It also identifies any bottlenecks or inefficiencies in the development process. The development team may use various tools and techniques, such as user testing, customer surveys, analytics data, and performance monitoring tools, to collect this data [50, 51]. They may also conduct regular reviews of the development process to identify improvement areas.

Learn: In the learn phase of LEAN development, the development team uses the data collected in the measure phase to identify areas for improvement and make changes to the development process and the product [47]. It may involve making changes to the product roadmap, adjusting development practices, or implementing new tools or techniques [52]. The goal of the learning phase is to improve the product continuously and the development process to deliver value to the customer more efficiently and effectively.

The final step in the cycle (see figure 2.9) would be to repeat the process, starting with identifying customer requirements and iterating through the *Build*, *Measure*, and *Learn*[9] phases again. Thus, the development team can constantly improve the product and the development process to deliver value to the customer more efficiently and effectively.

[9] LEAN cycle: https://www.einstein1.net/build-measure-learn/

As per [47], there are some key practices in LEAN development. These principles can be used to guide the discussion on which product development practices might be suitable for an organization, based on its specific circumstances.

Continuous delivery: Continuous delivery is the practice of regularly delivering small increments of working software to the customer rather than waiting until the end of the project to have a complete product. This allows the customer to see the project's progress and will enable them to give feedback and make changes as the project progresses. Continuous delivery helps to ensure that the software being developed meets the customer's needs and reduces the risk of delivering a product that does not meet their requirements [52]. It also allows the development team to identify and fix problems early in the process, saving time and resources. To practice continuous delivery, the development team must have a process for automatically building, testing, and deploying code changes [53].

Continuous improvement: Improving build quality in LEAN development refers to continuously improving the quality of the software being developed through effective coding practices, testing, and quality assurance processes [54]. To improve the build quality, we can use *Code reviews* [47], i.e., conducting regular code review sessions to help identify problems early on and ensure that code meets standards for readability, maintainability, and performance. Similarly, implementing *Quality Assurance* processes, such as testing, to catch and fix defects before they reach the customer can help improve the overall quality of the product.

Lean planning: It is a practice in LEAN development that focuses on maximizing value and minimizing waste in the software development process. It involves continuously prioritizing and reevaluating the work based on customer feedback to ensure that the team is first working on the most valuable features. As per [55], Lean planning aims to quickly identify and deliver the most valuable features to the customer while minimizing time and resources spent on non-critical tasks. This helps to ensure that the team is working on the right things and that the product meets the customer's needs.

In this way, LEAN development can help organizations deliver high-quality products to customers more efficiently and effectively.

Chapter 3

Solution Design

In the previous chapter, we defined various terminologies and concepts that we are using for our thesis. Consequently, in this chapter, we derive some design concepts required for our DSR. Therefore, we explore the DRs (see section 3.1), DPs (see section 3.2), and an overall solution design (see section 3.3 which gives an incite of the creation of our solution approach) to guide the result of our software tool. Design is a critical aspect of SE, providing the blueprint for implementing functional and non-functional requirements. The design phase is crucial in ensuring that software systems are developed efficiently, effectively, and with high quality. We develop our design approach based on the principles of DSR. Through this exploration, we will provide an in-depth understanding of the design process, highlighting the key factors that must be considered to create effective software solutions.

3.1 Design Requirements

DRs in DSR are typically defined as a set of constraints and specifications that must be met by the design artifact to be considered a successful solution. These are functional requirements, such as the features or capabilities the design/tool should have. In our findings of the DRs, we focus more on the functional requirements and ignore the non-functional. To derive a solution, we define the DRs with the help of the literature review and a comparison of some tools. For our research, we conducted a non-systematic literature review by reading research papers and looking at some renowned UI prototyping tools. The related literature is available in Appendix B. In this context, each DR refers to a generalized requirement that can be standardized and applied to future software applications. We covered a wide range of topics including *UI Prototyping*, *Low-Code/No-Code development*, *Model-Based Software Engineering*, *Continuous Experimentation*, *Task-Based Usability Testing*, and the *LEAN* *development process*. The following section presents *nine DRs* for our approach *(solution approach)*.

DR1: Heterogeneous Users states that *the approach should support diverse users with different needs, goals, and capabilities and integrate internal and external users.* It is supported by literature indicating that different users may have different needs, preferences, and levels of technical expertise [4]. By including a diverse group of users, you can get a broader range of feedback and insights into how the software performs for different users [23], and reduces the biases among developers [56]. In this context, the users can be from internal sources, such as employees, or external sources, like Amazon Mechanical Turk[1] for using the software.

DR2: Iterative Design states that *the approach should have an iterative, incremental method and identify and address any technical issues or design flaws early in the development process.* It is supported by literature indicating that the iterative approach involves the idea of breaking down development into small, incremental cycles of work rather than trying to deliver a complete product all at once [47]. The key benefit of an iterative approach is that it allows the development team to get feedback from users and stakeholders early in the development process and to make adjustments [5] to the product as needed.

DR3: Easy Development states that *the approach should be easy to develop and operatable by non-technical individuals with different techniques to create applications without extensive programming knowledge.* It is supported by literature indicating that a tool should have a UI that helps non-technical individuals to build software without including the developers. It can be achieved if the tool provides drag-and-drop interfaces [31], reusable pre-built components [57] (e.g., buttons, textbox, and other UI components) and a logical flow (E.g., *Screen1* is followed by *Screen2* and so on.). In many tools like `Figma`[2], `Invision`[3], and `Axure`[4], we saw these features.

DR4: Integrate Data Models states that *the approach should easily integrate data models (incorporating Create Read Update Delete (CRUD) operations of data models) and iterate them using various UI elements on the screens.* It is supported by literature indicating that by using the tool, citizen developers can easily access and integrate data models from multiple

[1] Amazon Mechanical Turk: https://www.mturk.com/
[2] Figma Prototyping tool: https://www.figma.com/
[3] Invision: https://www.invisionapp.com/
[4] Axure Rapid Prototyping: https://www.axure.com/

sources, including databases, APIs, and external systems [14], without having to write complex code or build custom integrations from scratch [15]. It accelerates the development process, reduces the risk of errors [30], and improves the software tool's overall quality. In many tools like `Figma`, `Invision`, and `Axure`, we saw the use of data models[5].

DR5: Classified UI variants states that *the approach should create multiple UI variants or versions, each with distinct UI elements or features.* It is supported by literature indicating different users may have other preferences regarding how they interact with an application [1]. At the same time, we saw many tools like Google Optimize[6], VWO[7], Convertize[8], and Freshmarketer[9] having provisions to create various UI variants or versions.

DR6: Conduct UI Split tests states that *the approach should conduct various UI split tests on the participants using different UI variants or versions.* It is supported by literature indicating that a product can be tested against different design solutions and variations [46] to see which variant performs the best in terms of usability, aesthetics, and other characteristics [6]. At the same time, continuously improve [45] the product based on the feedback of the best-fit variant.

DR7: Construct User Scenarios states that *the approach should observe and record how users interact with a software tool to evaluate the tool's ease of use.* It is supported by literature indicating that testing of the GUI of a software application is done using functional and usability tests [42]. This helps the developers to identify any usability issues [58] and improve them continuously [17]. And this helps in the identification and preliminary validation of user requirements in the early stages of development [23].

DR8: Collect Feedback states that *the approach should gather various user feedbacks (such as user behavior patterns, click rates or open-ended questions) from the split tests.* It is supported by literature indicating that feedback can be collected while observing the participants performing the tasks [59], like asking open-ended questions about their overall experience. At the same time, it should automatically record any feedback that participants give and analyze it while looking for some pattern in the data [50].

---

[5] These tools either have their implementation or depend on some third-party data model tool.

[6] Google Optimize: https://marketingplatform.google.com/about/optimize/

[7] https://vwo.com/de/

[8] https://www.convertize.com/

[9] https://www.freshworks.com/crm/marketing/

DR9: Aggregated Feedback states that *the approach should collect the gathered feedback and aggregate them to make improvements to the application.* It is supported by literature indicating that `Qualitative analysis` gathers an in-depth understanding of underlying reasons, opinions, and motivations [60]. Whereas `Quantitative analysis` measures and understands numerical data and helps identify patterns and trends [50]. An aggregation of the qualitative and quantitative analysis can provide a more complete picture of a situation and can be used to validate or disprove findings from one type of analysis [51].

DR10: Improvement states that *the approach should improve the prototypes from the results of the collected feedback in an iterative manner.* It is supported by literature indicating that visualization helps in prototyping by allowing the users to see and understand the design in a way that is easy to understand [61]. It also allows users to identify usability issues [17] early in the design process to make the end product user-friendly and easy to use by improving the prototype. In tools, various methods, like creating *Graphs*, *Charts*, *Plots*, etc. are used for visualization for improving the products.

3.2 Design Principles

DPs are guidelines or rules used to guide the design process in DSR [11]. They provide a framework for making design decisions [10] and help to ensure that the final solution meets the goals and objectives of the research. This section codifies our knowledge during the design study and derives DPs from abstract DRs in the iteration cycle of the DSR. The following shows the *nine DPs* for *our solution approach* that is built on the foundation of the mapped DRs (see figure 3.1).

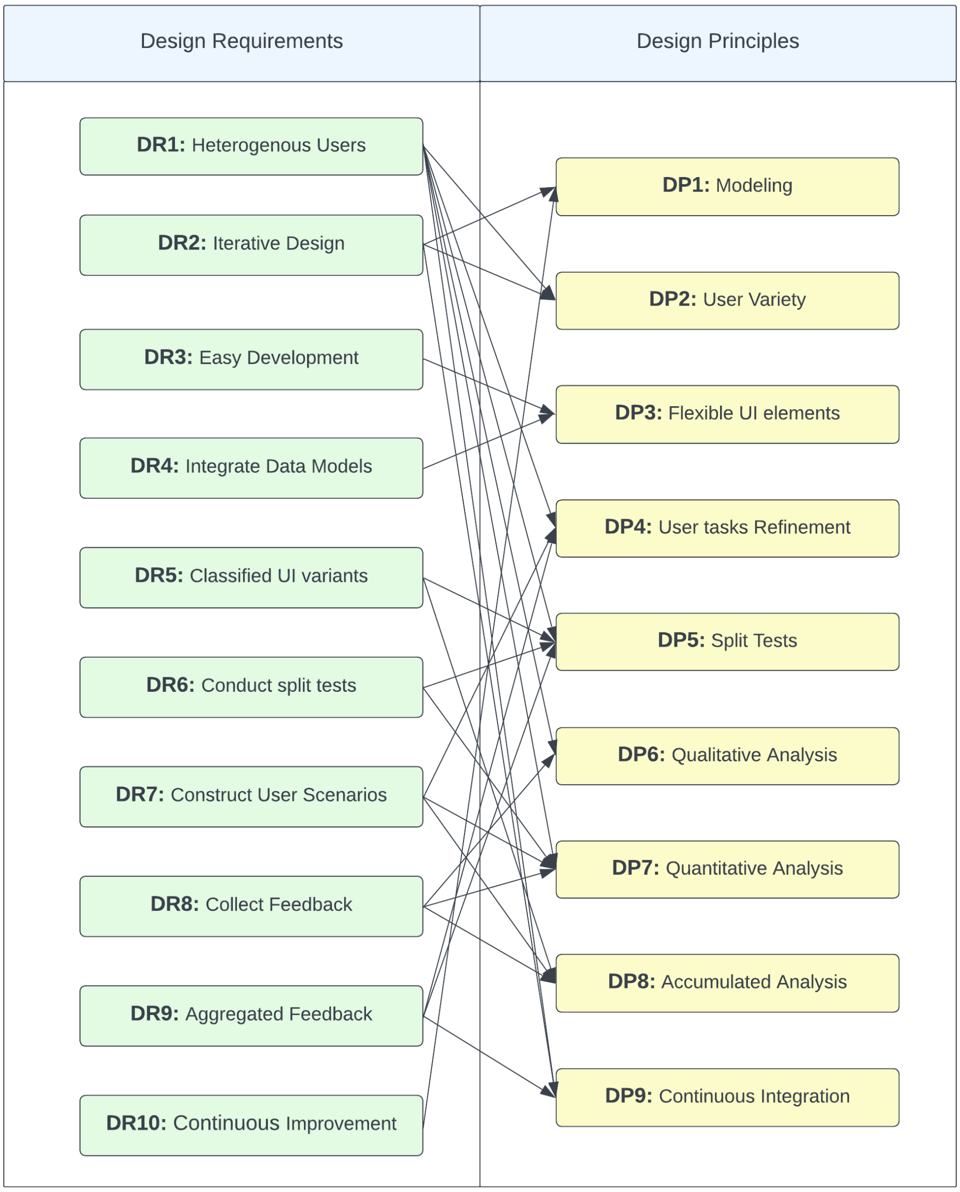


Fig. 3.1 A Map Between DRs and DPs

DP1: Modeling: *The solution approach is required to provide techniques for incorporating models that can be used to simplify and visualize the UI prototype so that developers can test its functionality and identify potential issues before the actual software is developed.*

Modeling increases transparency among various users, increasing their contribution to the product due to their excellent visualization and improvement capability (i.e., *DR10: Continuous Improvement*). Furthermore, modeling tools can automatically generate code or other documentation from the models, which can help reduce errors, improve efficiency, and decrease development time iteratively (i.e., *DR2: Iterative Design*).

DP2: User Variety: *The solution approach is required to provide techniques for incorporating diverse types of users, both the prototype creators i.e., admin users and participants i.e., users who participate in an experiment, so that developers can engage with various users during the evaluation process.*

Developers have many unclear generalities early in the product development process [4] that they can clarify by testing the underlying assumptions using different types of users (i.e., *DR1: Heterogenous Users*), and involving various product users using an iterative design for continuous improvement (i.e., *DR2: Iterative Design*). This helps to gather the user requirements smoothly, thus improving the product's usability.

DP3: Flexible UI Elements: *The solution approach is required to provide techniques for incorporating a library of UI elements and interactive components that can be easily customized and used in the prototype so that developers can demonstrate how the UI will function.*

A UI Prototyping tool is helpful when different interactive (i.e., *DR3: Easy Development*) re-usable components are used while creating the prototypes and integrate various data models (i.e., *DR4: Integrate Data Models*). These elements can be pre-designed UI elements such as buttons, menus, forms, etc. and interactive components such as textboxes, checkboxes, etc. It helps to get more feedback from the users or the participants, improving the product's usability and functionality.

DP4: User tasks Refinement: *The solution approach is required to provide techniques for incorporating task creation that simulates real-world scenarios and workflows and assign randomly to participants so that developers can collect and improve the UI prototype with the help of data.*

We can improve the usability of software by observing different users (i.e., *DR1: Heterogenous Users*) as they interact with the product and measuring how well they can accomplish specific tasks or goals (i.e., *DR7: Construct User Scenarios*). These tasks are some real-world scenarios and workflows, with clear instructions and defined success criteria. Moreover, the users can provide valuable insights into how easy or difficult it is to use the product and help identify areas where developers could improve the UI or design aggregating user feedback (i.e., *DR9: Aggregated Feedback*).

DP5: Split Tests: *The solution approach is required to provide techniques for incorporating the creation of different versions of UI prototypes and conduct tests to compare the performance of each version so that developers can make data-driven decisions about which UI design works best for their users.*

Split Tests (e.g., A/B tests) are performed on a randomly divided sample group of different users (i.e., *DR1: Heterogenous Users*) by exposing each group to the UI of different versions (i.e., *DR5: Classified UI variants* & *DR6: Conduct Split tests*) to find out the feature that is most usable and functional for the users. The results of the test are then used to determine which version is more effective by aggregating the feedback (i.e., *DR9: Aggregated Feedback*) from the user groups and finally optimizing the whole product.

DP6: Qualitative Analysis: *The solution approach is required to provide techniques for incorporating and analyzing qualitative feedback from participants so that developers can gain insights into the user experience and identify areas for improvement.*

We can conduct qualitative analysis systematically for examining non-numerical data to uncover patterns, themes, and insights from different users (i.e., *DR1: Heterogenous Users*). It can be achieved through various methods (i.e., *DR8: Collect Feedback*) such as content analysis or user feedback analysis, which involve coding, categorization, and interpretation of the data.

DP7: Quantitative Analysis: *The solution approach is required to provide techniques for incorporating quantitative analysis features so that developers can analyze and visualize data from A/B testing and other analytics for improving the UI prototype.*

We can conduct quantitative analysis systematically to test hypotheses, measure relationships between variables, and make statistical inferences about populations based on representative samples from different users (i.e., *DR1: Heterogenous Users*). It can be achieved by assigning tasks to various users in the study (i.e., *DR7: Construct User Scenarios*) and collecting their feedback (i.e., *DR8: Collect Feedback*) for their particular UI variant in the split test (i.e., *DR6: Conduct split tests*).

DP8: Accumulated Analytics: *The solution approach is required to provide techniques for incorporating different analytics and metrics so that developers can track the performance and behavior of the various UI prototype variants being tested.*

Performing various tests (e.g., usability testing) with diverse users (i.e., *DR1: Heterogenous Users*) ensures that the software is accessible and easy to use for different groups of people with an accurate evaluation of the task provided to the users (i.e., *DR7: Construct User Scenarios*). Moreover, diversity in software development feedback mechanisms (i.e., *DR6: Collect Feedback*), received by testing different UI versions (i.e., *DR5: Classified UI variants*), helps ensure the software is inclusive and accessible.

DP9: Continuous Integration: *The solution approach is required to provide techniques for incorporating the tool's design into small, incremental phases in the software development process so that developers can improve software delivery and make product changes and improvements based on customer feedback.*

Using a continuous incremental approach, we can continuously improve software products by delivering value to customers as quickly as possible [48], constantly refining and improving the product [55], and delivering the product as soon as possible. Here, the iterative design should be used (i.e, *DR2: Iterative Design*) to get continuous feedback from a variety of users (i.e., *DR1: Heterogenous Users*). The feedback which is collected should be a combination (i.e., *DR7: Aggregate Feedback*) of various feedback helping significantly improve the application.

3.3 Overall Solution Design

As shown in figure 3.2, we conceptualize the solution design from our codified DPs. The software platform consists of two types of roles consisting of the users for creating the Prototyping tool (Admin users like, *Product Owners*, the *Designers*) and the *Users or Participants* for testing the tool. In this section, we arrange our DPs using a LEAN development approach[10], a cycle consisting of *Build*, *Measure*, *Learn* phases.

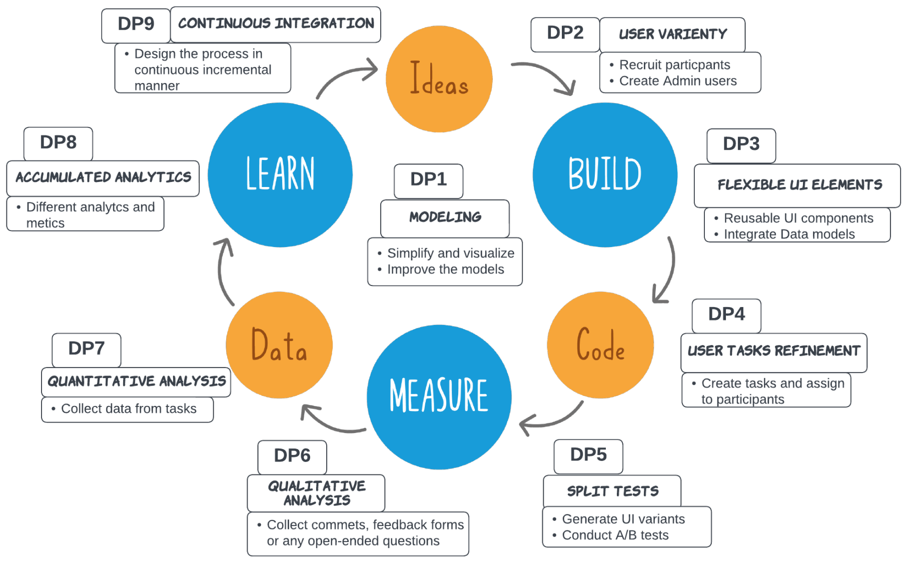


Fig. 3.2 Overall Solution Design for a UI Prototyping tool using LEAN principles

In the LEAN development cycle, the whole process is done in a model-based approach (i.e., DP1) for the improvement of the solution approach.

Build: Initially, different users register to our tool (i.e., DP2) having different rights. The tool administrators (i.e., the users with admin rights) build a UI prototype using various reusable UI components and create data models (i.e., DP3). Using the UI elements, these users make various screens and connect them to various data models to have a logical flow of the prototype. In the next step, users would create various tasks simulating real-world scenarios, assign them randomly to participants and collect feedback (i.e., DP4). After creating the tasks, users create various UI variants or versions for creating split tests (i.e., DP5).

[10]Adopted from LEAN process development: https://www.lean.org/explore-lean/product-process-development/

Measure: In this phase, we measure the data we receive from the tasks by conducting qualitative (i.e., DP6) and quantitative (i.e., DP7) analysis. The quantitative analysis is done from the data the users collect from the tasks. Similarly, the qualitative analysis is done by collecting the comments, open-ended questions etc. after the task is finished.

Learn: In this phase, the data received from the analysis is processed and visualized. The data is processed by aggregating and combining the results of the qualitative and the quantitative analysis (i.e., DP8). Finally, the models are given feedback from the data analysis, improving the UI prototype and updating it with the winner variant (i.e., DP9) as the default UI. The refined product can be deployed for the entire population (or all the users) using the deployment module of a no-code development platform.

Chapter 4

Solution Concept

In the previous chapter, we defined different DRs and DPs for our solution approach. Consequently, in this chapter, we present the conceptual design of the solution, which outlines the system's key features, components, and functionalities. This chapter bridges the gap between the solution design and the actual implementation of the solution by providing a detailed description of the solution approach. To visualize the system's architecture and interactions between components, we have defined software architecture (see section 4.1) to facilitate better planning and implementation decisions. In section 4.2, we explain UI prototyping component in detail. Similarly, in section 4.3, we explain how to create experiments and connect participant users to experiments using our tool. In section 4.4, we explain how data is stored in our database using the data models. In section 4.5, we explain the code generation and other deployment processes required to build our tool from a prototype to working software. Finally, in section 4.6, we explain how access control for various users is managed in our security infrastructure.

4.1 Software Architecture

In this section, we will discuss the software architecture of a UI prototyping tool with provisions for split and task-based usability tests. The software tool uses the MVC architecture, with the main components being the *UI Prototype Management* and the *UI Experiment Management*. The models used in the architecture are the *UI Prototype* for prototyping and the *UI Experiment* for conducting the split tests. The diagram (see figure 4.1) shows the different modules and components of the UI prototyping tool and how they interact with each other during the execution of the tool. We chose the MVC architecture for this tool because it separates the application logic into three interconnected components. The model component represents the data and the business logic of the application. The view

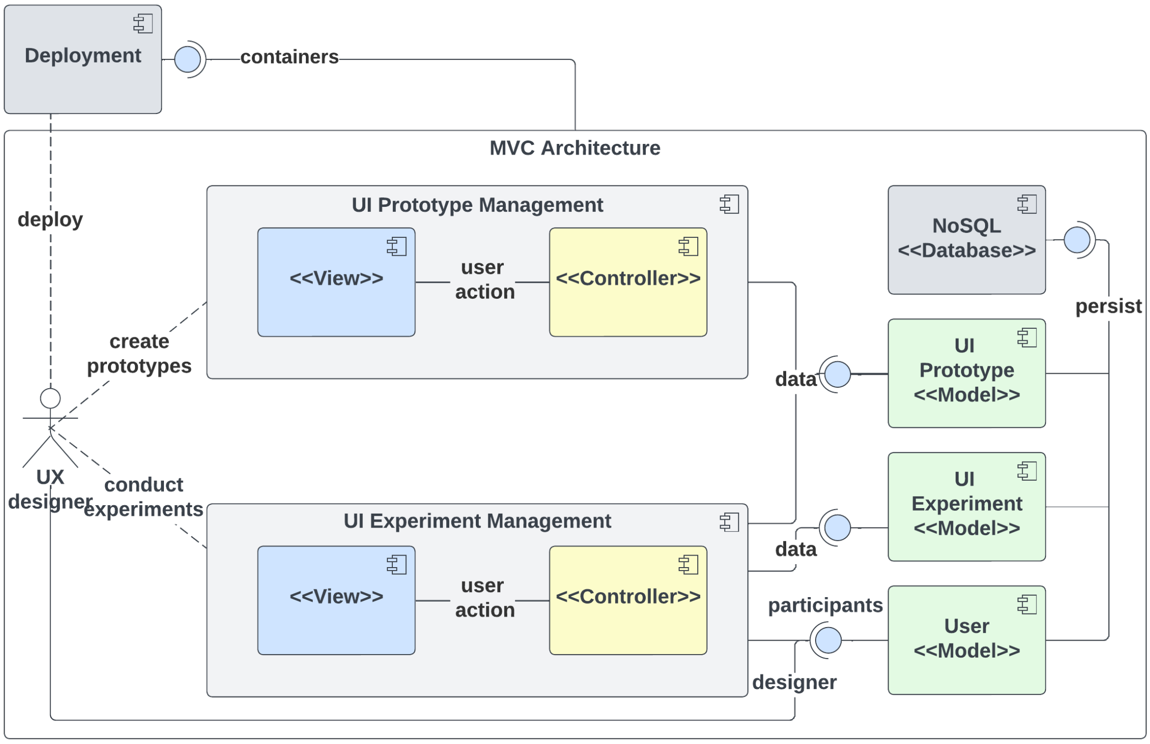


Fig. 4.1 MVC Architecture of our UI Prototyping tool

component represents the application's UI, and the controller component handles the user input and updates the model and the view. The separation of these components allows for easier application maintenance, scalability, and modifiability.

As shown in figure 4.1, the *UI Prototype Management* component manages the prototyping process creating different UI screens and adding UI elements. Conversely, the *UI Experiment Management* component manages split tests and task-based usability tests. Both of these components contain the controller and the view and are explained in detail in the next sections. These controllers are used for handling user input and updating the models and views accordingly. The *UI Prototype* model is connected to the *UI Prototype Management* component, and the *UI Experiment* model is connected to the *UI Experiment Management* component. Similarly, the *User* model is responsible for providing the user participants for the experiments component and at the same time providing a UX designer for executing our tool. As per the MVC architecture, the data is then persisted into the database using a service, at the same time, the user is also allowed to do the CRUD operations. Finally, for the database, we use a *NoSQL* database that provides flexibility and scalability for applications with changing data requirements.

The *UX designer* is responsible for prototyping and creating user experiments and scenarios. That user can access all the tool's features and create, edit, and delete prototypes and experiments. Finally, for deployment, we have one component that includes all the files and dependencies required to run the application. This component contains the server-side code, the client-side code, and the necessary libraries and dependencies. The deployment component is responsible for packaging the application and deploying it to the target environment.

In summary, the software architecture for our UI prototyping tool is designed using the MVC architecture, with the details explained in the next sections. Similarly, the role of the UX designer, the user responsible for prototyping and creating user experiments and scenarios, is also explained in the next section in detail.

4.2 UI Prototype Management

In our solution approach, we must have features for creating various UI prototypes. Therefore, using this component, we allow the UX designers to create UI prototypes as shown in figure 4.2. It consists of various sub-components for constructing UI elements, data models, and analyzing results.

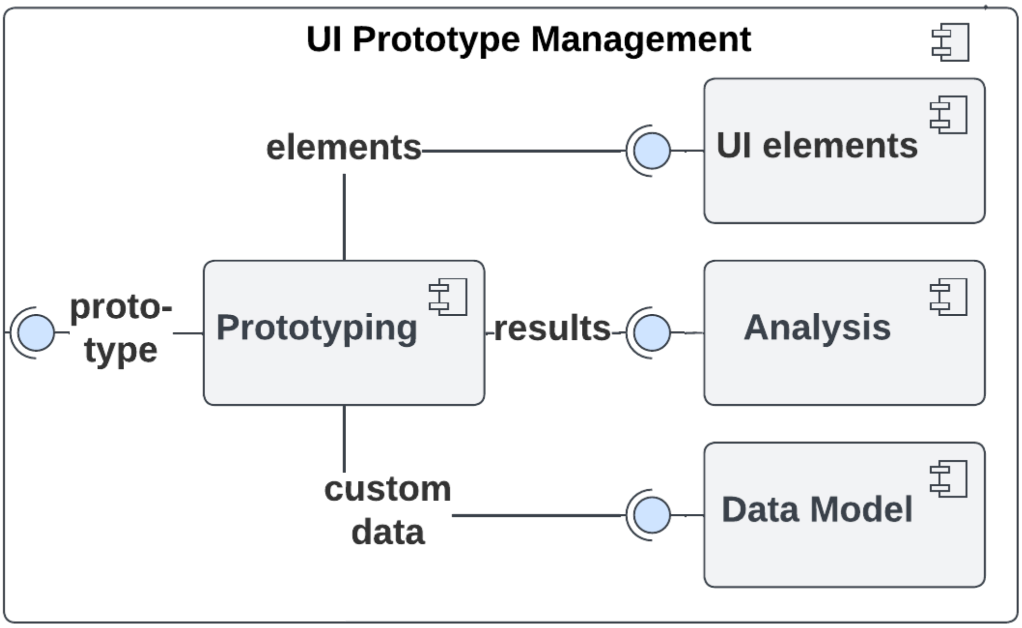


Fig. 4.2 Details of UI Prototyping Management connecting to a Database

UI Elements The *UI elements* component of figure 4.2 provides a collection of reusable and flexible UI elements that can be easily used in creating new UI prototypes. These UI elements include buttons, input fields, dropdown menus, and many others, and are typically organized into categories for easy access and management. Each of our UI elements includes a range of properties that can be customized to meet the user's specific needs. Providing these options allows users to create prototypes that accurately represent their vision without being limited by the tool's capabilities. Therefore, this component is crucial for quickly and efficiently creating new UI prototypes as it eliminates the need for developers and designers to recreate the same UI elements whenever they start a new project.

Data Model The *Data Model* component of figure 4.2 provides a way to customize the data used in iterating the design of the UI prototype. This component allows designers to define custom data models, which are then stored in a database and can be accessed and manipulated using a model. Similarly, these models are defined for their UI prototypes, which can be used to simulate real-world data and test the UI design in different scenarios. With this component, users can create data models, which are then stored in a database and can be accessed and manipulated using a model. This component is essential for creating realistic and comprehensive UI prototypes that more closely resemble the final product and accurately reflect the end users' needs.

Analysis The *Analysis* component of figure 4.2 provides a way to analyze the results of the experiments conducted by the UX designer. This component enables users to combine and aggregate the results of qualitative and quantitative analyses. This component allows designers to evaluate their UI prototypes' success, identify improvement areas, and make data-driven decisions about how to iterate and improve it. The analysis component can provide a range of data, including user feedback, metrics, and analytics, which can help designers to make informed decisions about the UI design. It allows users to quickly iterate on their designs based on the insights gained from the analysis, ensuring that the final product is optimized.

Prototyping The *Prototyping* component of figure 4.2 requires using all the other components mentioned above. The central component provides a way to create, manage, and test UI prototypes. By integrating the UI elements, data models, and analysis components, designers can create highly effective UI prototypes that meet the needs of the end users. The prototyping component typically includes features like drag-and-drop UI design tools, data uploading features, and the ability to quickly iterate and experiment with different UI design ideas.

In summary, implementing a prototyping management component simplifies the UI prototyping, helps improve the effectiveness of the UI, allowing to add UI elements, and data models and improves the prototype.

4.3 UI Experimentation Management

In our solution approach, we must have features for creating various UI variants and conducting experiments of A/B testing. Therefore, as shown in figure 4.3, using this component, we allow the UX designers to create experiments on the UI as defined in the UI prototype. It consists of various sub-components for constructing UI variants, assigning participants to experiments, managing participants' tasks, including qualitative questions, and collecting quantitative feedback.

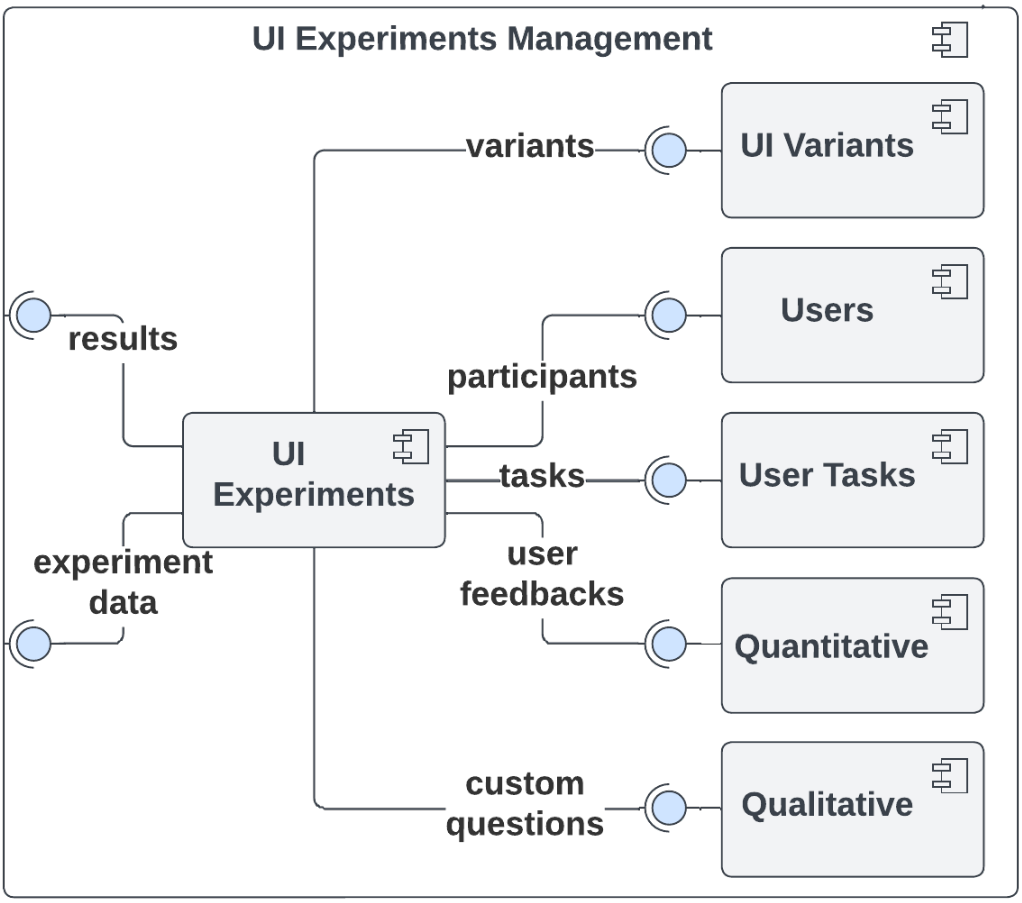


Fig. 4.3 Detailed diagram of UI Experiment Management

UI Variants The *UI variants component* of figure 4.3 allows the creation of multiple prototype variants, allowing for A/B testing and comparison between different design approaches. UX designers can easily customize these variants to create unique user experiences. This component provides a range of prototype variants that UX designers can customize to suit their specific needs. These variants can be created and saved in the system for easy access and modification and reused across multiple experiments. This component ensures that designers have access to various UI variants that can be easily modified to test different hypotheses or design options.

Users An important aspect while experimenting or A/B Testing is the users' participation. The *Users* component provides the ability to manage user participants in UI experiments. It enables UX designers to create user profiles with specific characteristics and demographics and assign them to particular experiments. This component also provides user management functionality, including adding, removing, and editing user profiles and the ability to group users for specific experiments.

User Tasks The *User Tasks* component allows UX designers to create and manage tasks for participants to complete during UI experiments. These tasks can be customized to fit specific user scenarios and test different aspects of the UI design. This component provides a task management system for participants in UI experiments. It allows designers to create and assign tasks to specific users and track their progress and completion. Tasks can be simple, such as clicking on a button or filling out a form, or more complex, such as completing a series of steps or navigating a website. This component ensures the UX designer can measure user performance and satisfaction by assigning tasks during the UI experimentation process.

Quantitative The *Quantitative* component provides a mechanism for collecting user feedback on specific UI elements from tasks and experiments. This feedback can make data-driven decisions and inform future design iterations. This analysis can include metrics such as time to complete a task, success rates, or user satisfaction ratings. The feedback can be used to evaluate the effectiveness of different design options and identify improvement areas. This component ensures that designers have access to quantitative data that can be used to make informed decisions during the UI experimentation process.

Qualitative The *Qualitative* component provides custom questions for the experiment and allows for gathering qualitative data from participants. These questions can be designed to gain insight into user perceptions, opinions, and attitudes toward the UI prototype. These questions can be used to gather qualitative feedback from participants, such as their opinions or impressions of the design, or to elicit specific information, such as their understanding of a particular feature. This component ensures that designers can capture user feedback specific to the experiment. Moreover, this component complements the quantitative component and provides users with a more holistic view of how users interact with their prototype.

UI Experiments The *UI Experiments* component ties the above sub-components together, providing a seamless experience for conducting UI experiments. Integrating all these components allows for efficient management of UI experiments and provides data analysis capabilities for informed decision-making. This component ensures that designers have a streamlined way to manage the entire UI experimentation process, from UI variants creation to data analysis. It allows designers to create and execute experiments, assigning users, tasks, and prototype variants. Conversely, it provides an analysis of the results of the experiments, including both qualitative and quantitative data, and stores the data in a database for future use.

In summary, implementing an experiments management component simplifies the A/B testing, helps improve the effectiveness of the UI, and allows you to manage and track the different interface variations.

4.4 Database Components

In our solution approach, we must have features for persisting the data in our database. Therefore, as shown in figure 4.4, using this component, we store the data from the models from our MVC architecture. In the component diagram, the database components are represented by *MongoDB* component and a *Persistence Infrastructure*.

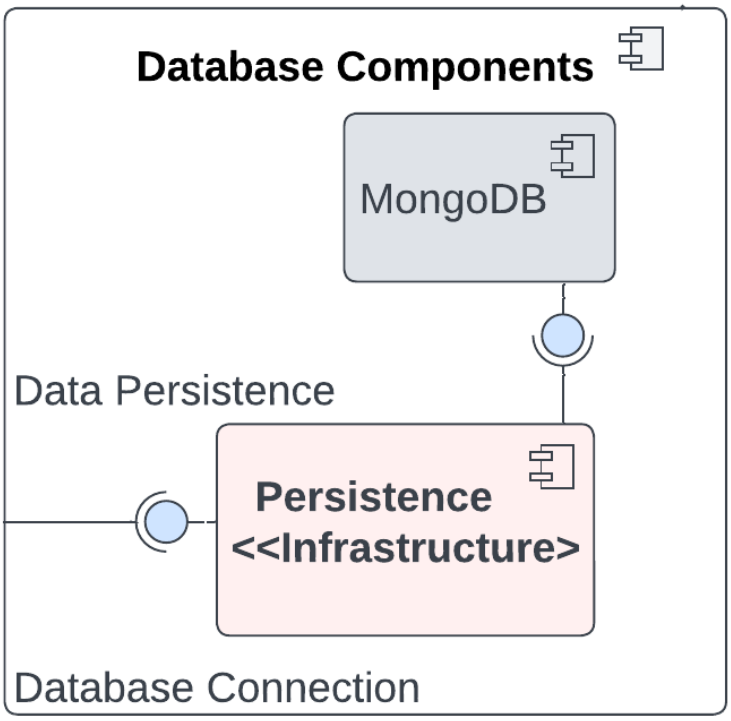


Fig. 4.4 Details of Database Management

Persistence Infrastructure The *Persistence Infrastructure* component is responsible for ensuring that the data in the database is persistent and available for future use. Therefore, a persistence infrastructure is necessary for our software to manage a system's data storage and retrieval. It provides a layer of abstraction between the application and the database, which enables us to easily switch between different database types and optimize the data storage for our needs. The component ensures data consistency and reliability, even during system failures or errors. It provides a set of APIs and tools for data access and manipulation, including CRUD operations in the database.

MongoDB *MongoDB* component represents the database components. It is a NoSQL database that stores data in collections instead of tables, allowing for flexible and scalable data storage. In our MVC architecture, MongoDB provides the application's database connection, allowing it to store and retrieve data. MongoDB is also known for its ability to handle large amounts of unstructured data, making it a popular choice for modern web applications.

In summary, when implementing a persistence infrastructure for a non-relational database like MongoDB, we developed the structure and functionality of the database and designed the persistence layer accordingly. Therefore, the system can store and retrieve data in an efficient, scalable, and reliable way.

4.5 Deployment Infrastructure

A deployment infrastructure is necessary for ensuring that the system is deployed and running in a secure, scalable, and reliable way. A deployment infrastructure is responsible for managing the deployment of components, ensuring that they are running on the appropriate servers or cloud-based environments, and monitoring their performance. We use a no-code approach and microservice architecture to simplify deployment infrastructure implementation. It allows for the creation of deployment pipelines and automates many of the processes involved in deploying components.

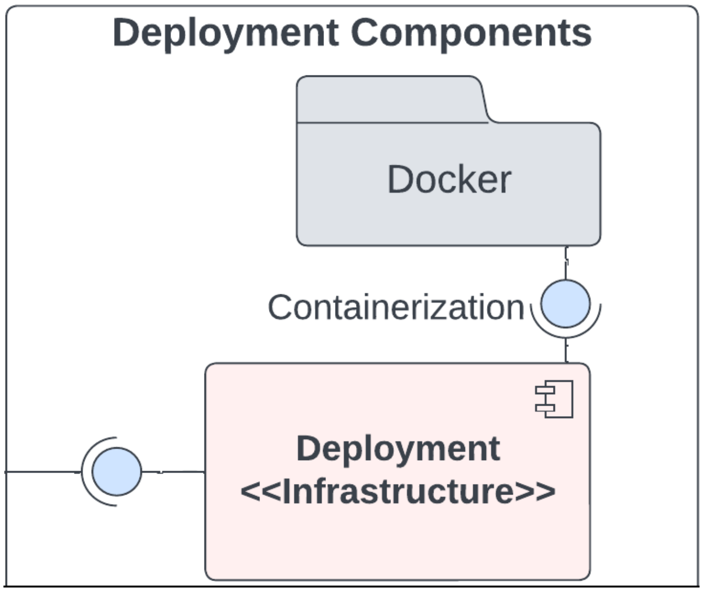


Fig. 4.5 Details of Deployment Components

Deployment Infrastructure We use a no-code approach using code generation to simplify the implementation of our deployment infrastructure by automating many of the tasks involved in deploying the system. So, we define a set of rules and templates for automatic code generation and generate some reusable components as defined by the UI prototyping management. These components' implementation uses templates containing its code (e.g., for a button element defined in the UI prototype, a template is generated containing all its properties and logic). Then, the code generator takes input data, including configuration files, database schemas, and models of system components and their interactions. Finally, the code will be generated using the templates and rules defined in the previous steps, and we will containerize the generated code.

Docker Containerization plays an even more important role in using a microservice architecture. It must support the deployment of many independent microservices that may be deployed across multiple servers or databases. From our component diagram, we first identify the system's microservices and their dependencies, such as database components, UI prototyping management, and UI experimentation management. It helps determine how the microservices should be deployed and how they should communicate. Next, we create Docker images for each microservice and its dependencies. Then, we define docker-compose files[1] in yaml[2] (see Appendix A) that describe how the docker containers should be deployed and connected. Finally, we build the *docker images* and deploy them as *docker containers*.

In summary, implementing a deployment infrastructure for a microservice architecture can simplify the deployment and management of microservices, making it easier to build and maintain complex systems.

[1] A Docker Compose file is a YAML file that specifies the microservices, their dependencies, and how they should be deployed and connected.

[2] YAML format for files: https://www.redhat.com/en/topics/automation/what-is-yaml

4.6 Security Infrastructure

Security infrastructure is essential to protect the system from unauthorized access and attacks. By implementing security measures, such as access control, we limit access to sensitive information and functions to only authorized users. Similarly, authentication is one of the fundamental security mechanisms used to verify the identity of users. Our security infrastructure, therefore, provides access control and some authentication mechanisms (see figure 4.6).

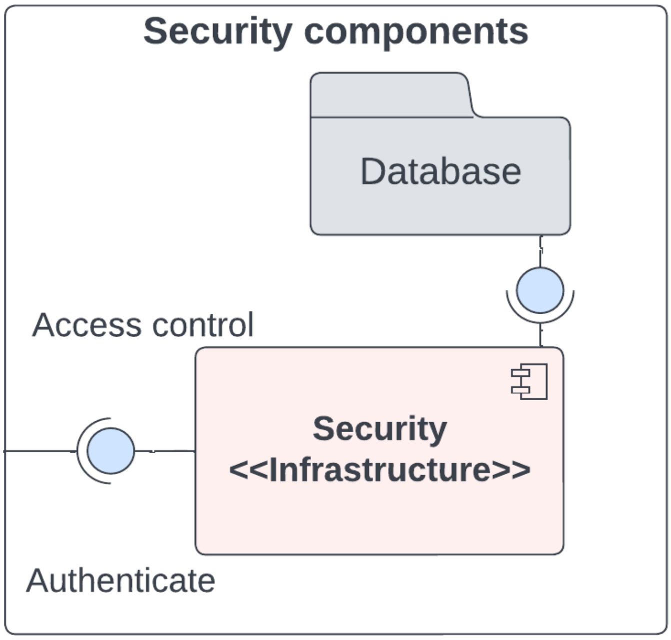


Fig. 4.6 Details of Security Components

Security The *Security Infrastructure* provides access control and authentication mechanisms. Access control involves defining policies and rules determining who can access resources under what circumstances. Therefore we implement access control mechanisms using role-based access control (RBAC) and then allowing or denying access based on those roles or groups. So, we first identify the roles or groups requiring system access, including participants, UX designers, and other users[3]. Then we define the access rights (e.g., the UI prototyping management should only be accessible to the UX deginers). Finally, we implement a mechanism that checks the user roles and then accesses the data or services accordingly.

[3]This may include different stakeholders like product managers, developers, etc.

Next, authentication involves verifying the identity of users and ensuring that they have the appropriate permissions to access the system. We provide an authentication mechanism using a password and username. It ensures that only authorized users can access our tool. Moreover, we define some validators for password strength to make sure the passwords are strong.

In summary, implementing a security infrastructure providing access control and authentication mechanisms for our tool can help prevent unauthorized access and protect sensitive data from being compromised.

Chapter 5

Solution Implementation

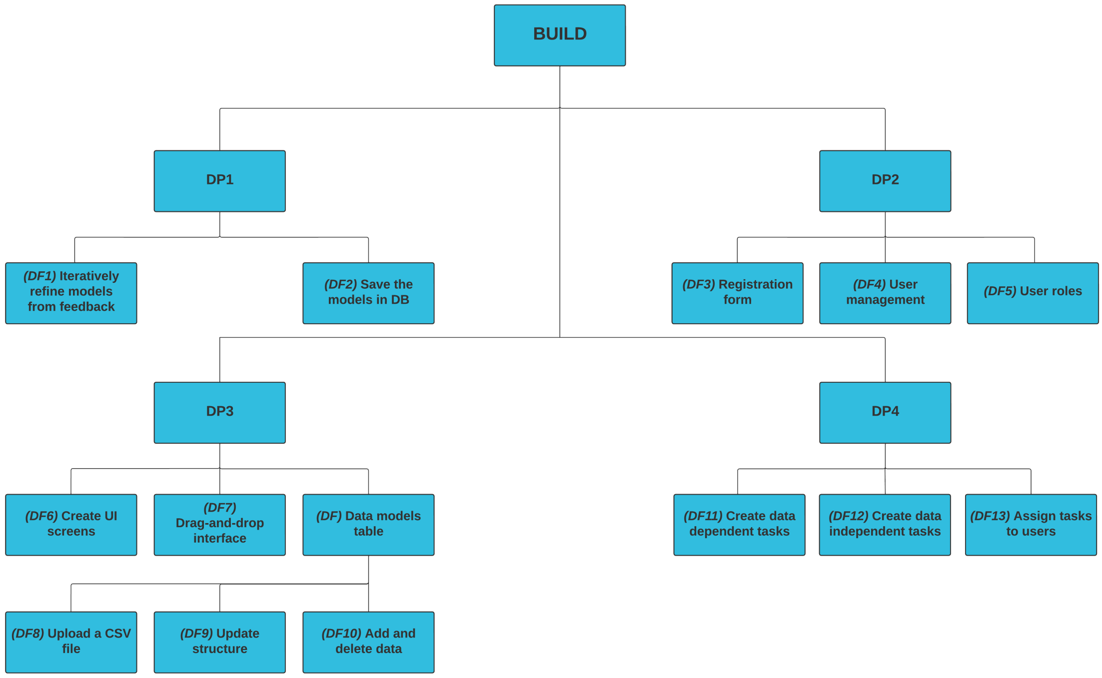
In the previous chapter, we displayed an architectural overview of our solution approach. Consequently, the solution implementation chapter of this thesis presents the details of how the proposed system was developed and implemented. Firstly, we define the DF (see section 5.1) that focuses on designing and developing new artifacts, methods, or systems. Next, the description of the technology stack and development tools is explained (see section 5.3). The chapter includes a detailed explanation of the critical features and functionality of the system and the processes used in frontend implementation (see section 5.4), database implementation (see section 5.6) backend implementation (see section 5.5), and the software tool (see section 5.2). Through this chapter, we will explain how we transformed the proposed solution into reality and its potential for real-world applications.

5.1 Tool Design Features

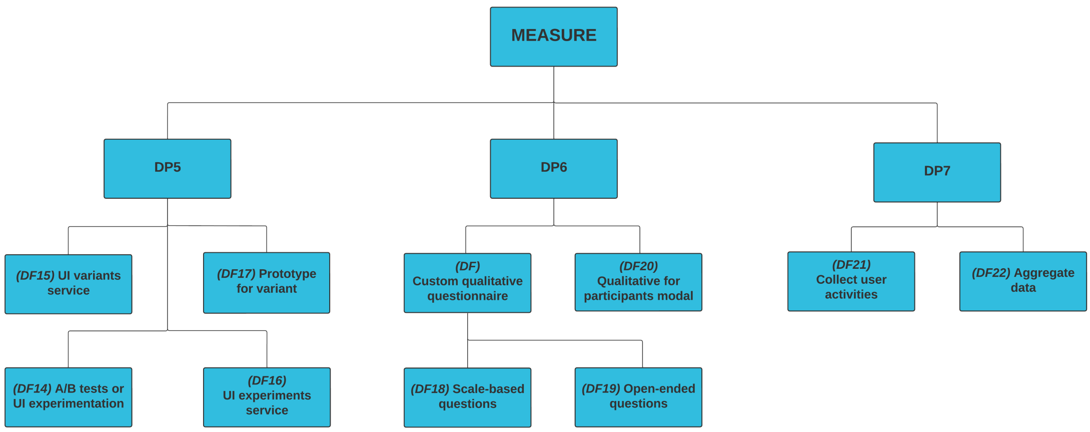
The DSR aims to produce a tangible outcome, such as a new software tool, a process improvement, or a theoretical framework with the derived DFs. It often involves multiple cycles of design, evaluation, and redesign, allowing for constant improvement of the artifact. In the development step of the *DSR*, we derive the features of our software tool using the DFs. In this section, we translate each DPs (we defined in chapter 3) into a set of DFs that we can directly implement into the tool or solution approach and we describe the DFs for each of the derived DPs. As shown in the figure 5.1, we see that the DFs are divided according to the *LEAN* development cycle.

Build: In the building phase of the LEAN development cycle, based on the **DP1: Modeling**, the solution tool provides features for creating the models (*DF1*) useful for persisting data in the database and updating them (*DF2*) by iterating and improving the models with the results of the experiments. It helps to create our prototypes in a model-based approach and improve our prototypes by improving the models throughout the LEAN development cycle. With the **DP2: User Variety**, the key features of our solution tool for the UI prototyping tool include registration for diverse users using a registration form (*DF3*) and user management, i.e., features that enable team members to create and manage user accounts (*DF4*), assign different levels of access and permissions (*DF5*). The user module maintains all these features and also the user authentication, authorization, and user profile management. Next, based on the **DP3: Flexible UI Elements**, the UX designer prototypes using the features of our solution tool, which includes the creation of different screens (*DF6*) and reusable UI elements using a drag and drop interface (*DF7*). The tool also provides a feature to add custom data i.e., creating data models by uploading a CSV file (*DF8*), revising structures of data model table (*DF9*), adding and deleting data from the table (*DF10*). Next, based on the **DP4: User tasks Refinement**, the UX designer creates user tasks for the users using the features of our solution tool, which includes creating tasks for users depending on the data model (*DF11*) and independent of data models (*DF12*), and assign the task to the experiments (*DF13*).

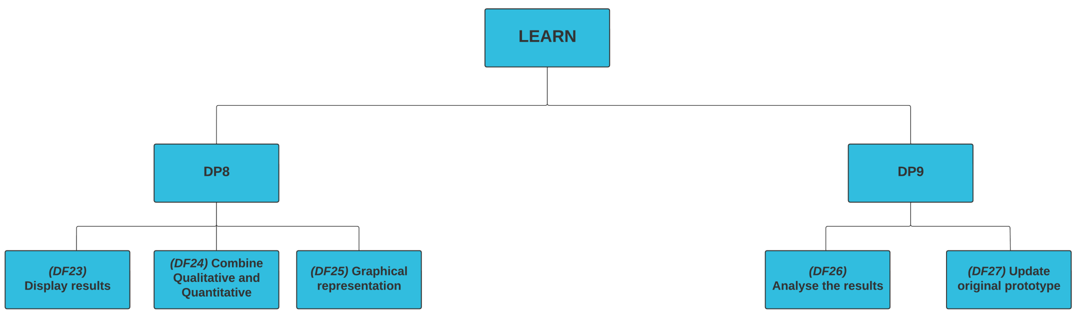
Measure: In the measuring phase of the LEAN development cycle, based on the **DP5: Split Tests**, the UX designer would be able to create A/B tests or UI experimentation (*DF14*) using the features of our solution tool, which includes creating and modifying the experiments (*DF15*), creating and updating the UI variants (*DF16*), and modifying the variants' prototypes such that each variant has a unique view (*DF17*). Next, based on the **DP6: Qualitative Analysis**, the UX designer would be able to do qualitative analysis on the users using the features of our solution tool, which includes creating the custom qualitative questionnaire with options in the answering formats like *scale-based*, i.e., the users will have to choose from options 1 to 10 (*DF18*) and open-ended questions, i.e., the users will have the freedom to answer whatever they think (*DF19*). And, the user participants will answer these qualitative questions after finishing the tasks with a modal appearing with different questions (*DF20*). Similarly, based on the **DP7: Quantitative Analysis**, the UX designer creates quantitative analysis on the users using the features of our solution tool, which includes collecting the task data feedback, i.e., collecting the time taken to finish the task, number of unsuccessful attempts, the path taken by the users to complete the task, etc. (*DF21*), and aggregating these feedback data (*DF22*).



(a) A Hierarchical Diagram of DFs from *Build* phase



(b) A Hierarchical Diagram of DFs from *Measure* phase



(c) A Hierarchical Diagram of DFs from *Learn* phase

Fig. 5.1 Hierarchical Structure of DFs

Learn: In the learning phase of the LEAN development cycle, based on the **DP8: Diversity in Analysis**, the UX designer compares the statistics using the features of our solution tool, which includes showing the results of the experiment (*DF23*), combining qualitative and quantitative analysis of each variant (*DF24*), and a graphical view to see the results (*DF25*). Finally, based on the **DP9: Continuous Design**, the UX designer should be able to analyze the results of the UI experiments (*DF26*) and update the original prototype with the results from the experiment (*DF27*) for continuous improvement.

Overall, the design features section in DSR delivers a comprehensive overview of the solution tool to address the identified problem or research question. Moreover, this section describes the DFs for our Proof of Concept (POC) solution tool.

5.2 Software Tool

The software tool developed for this research project is a POC UI prototyping tool that offers design features to create, customize, and test user interfaces. The tool was developed to help UX designers prototype and test different UI versions using split tests or A/B testing methods. The tool provides a simple and user-friendly interface for creating UI prototypes. It offers a variety of design features, such as customizable UI elements, data modeling, and experiment creation, to help designers create and test UI designs efficiently. In this section, with the help of the tool's screenshots, we will explain the features and functionalities in detail, including how to use the tool to create UI prototypes, set up experiments, and analyze the results. The tool also offers an experiment creation feature that allows designers to create split tests or A/B testing methods to test different versions of their UI prototypes. Designers can create multiple experiments and define experiment tasks, such as finding a movie to watch, to gather quantitative data from participants.

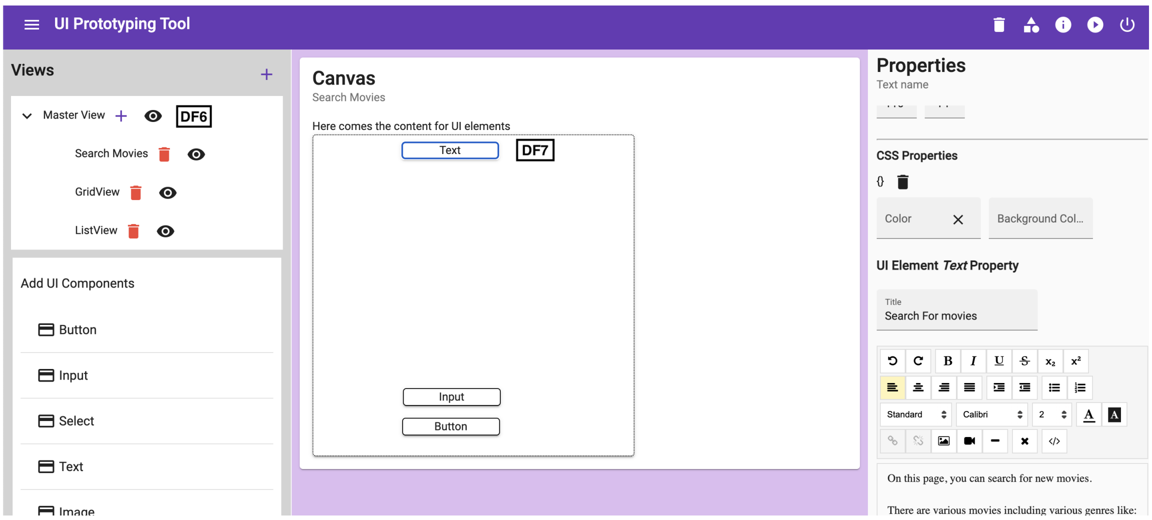


Fig. 5.2 Prototyping Page of the Tool

First, the UX designer needs to create an account as the admin user of the tool. After successful login, the tool provides a headstart option that generates an example movie-streaming UI prototype to help designers jump-start quickly. The generated prototype (see figure 5.2) includes a basic layout and a set of UI elements that can be customized to fit the designer's needs. Next, designers can create data models (see figure 5.3) for their UI prototypes by adding, modifying, or deleting data fields using the data model feature. This feature allows designers to test and manipulate data in their UI prototypes, making creating realistic and functional prototypes easy.

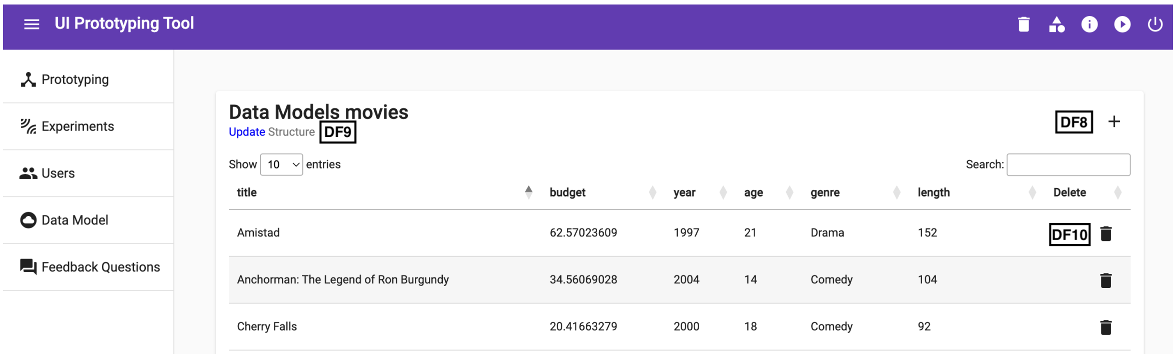


Fig. 5.3 Data Models Page of the Tool

The tool also offers an experiment creation feature (see figure 5.4) that allows designers to create split tests or A/B testing methods to test different versions of their UI prototypes. Designers can create multiple experiments and define experiment tasks, such as finding a movie to watch, to gather quantitative data from participants. Similarly, to gather participant feedback, designers can create custom questionnaires, such as open-ended or scale-based questions, and participants can answer them after completing experiment tasks.

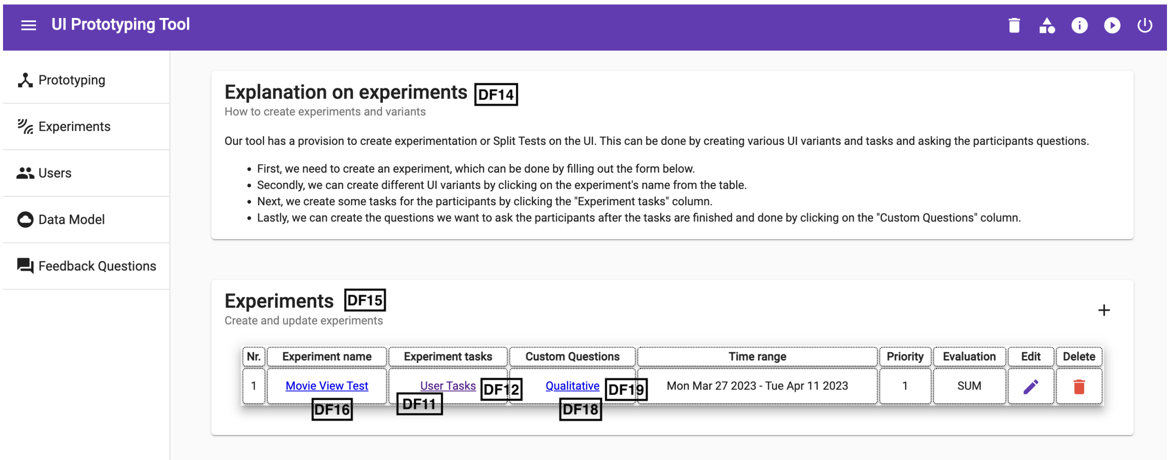


Fig. 5.4 UI Experiments Page of the Tool

Next, the UX designer can deploy the tool to see the actual UI elements and the screens that were prototyped. Now, the UX designer recruits some participants and instructs them to participate in a UI experiment. As the user participant logs in to check the tool, they are navigated to the tools page. This page contains the tasks that are assigned to the participant by the UX designer (see figure 5.5).

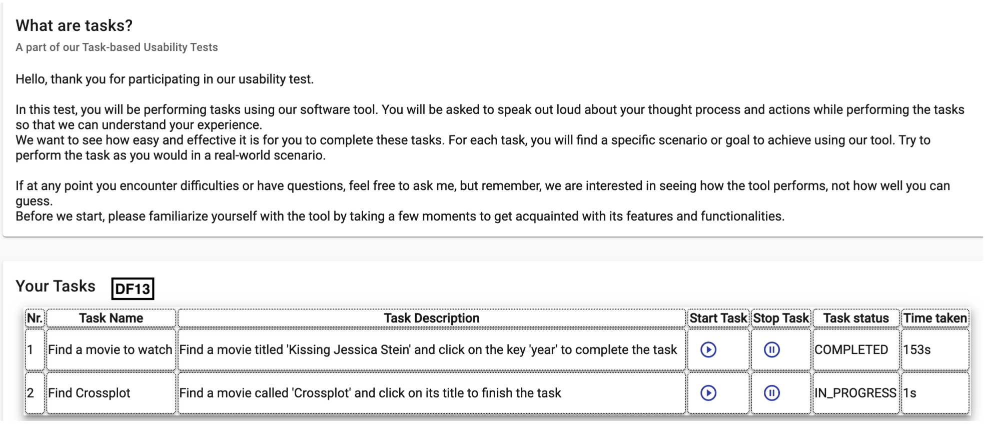


Fig. 5.5 Tasks Assigned to the Users

Next, the user participant clicks on the *Start Task* button (see figure 5.5) to make the task status *IN_PROGRESS* and the tool starts measuring the user movements like time taken to finish the task, the path taken by the user participant, and the number of unsuccessful attempts.

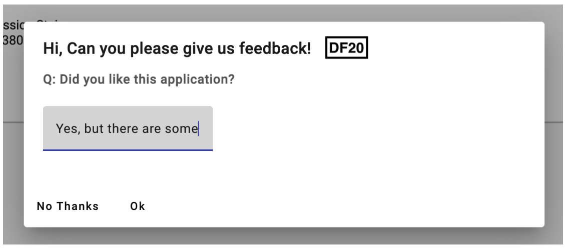


Fig. 5.6 Users Answering the Qualitative Questions

Finally, when all the user participants finish the experiment, the tool provides an analytics page (see figure 5.7) to help UX designers analyze the results of their experiments and decide which version of their UI prototype is more effective and, improve the original prototype.

It is important to note that the tool is a proof of concept rather than industry-ready. It was developed using the DFs to create a simple and user-friendly interface for prototyping and testing UI designs. The tool's functionalities are limited and may need to be more suitable for complex UI design projects. However, it can be a starting point for further development and refinement of UI prototyping and testing tools.

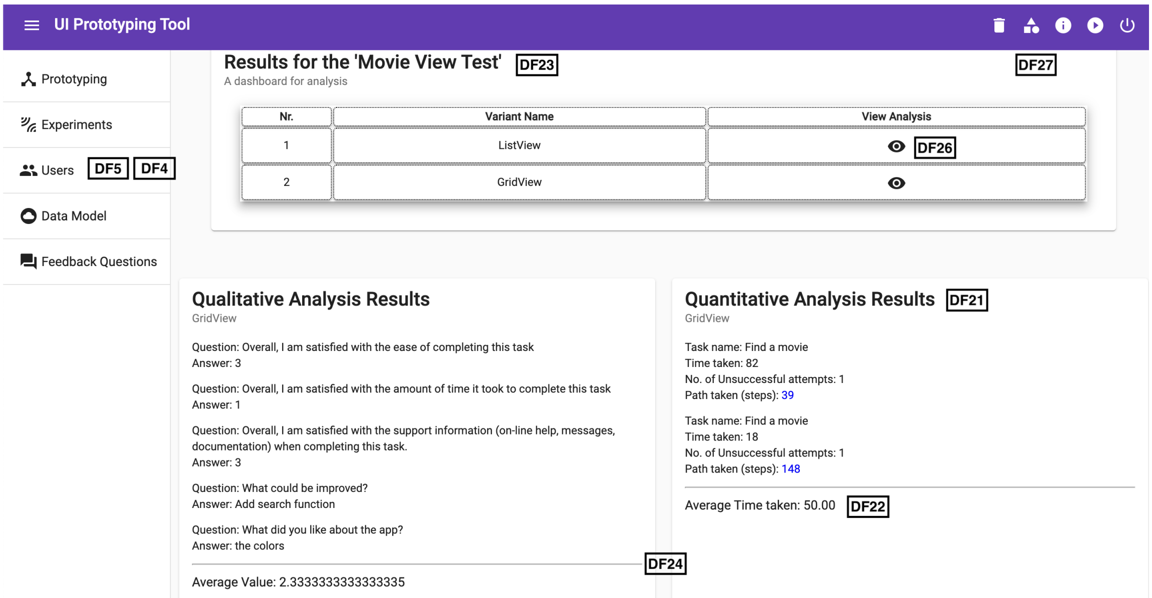


Fig. 5.7 Experiment analysis Page of the Tool

5.3 Technologies Used

We have created a software tool that allows us to test the features (DFs) and underlying principles of the developed DFs with actual users. To test our solution approach, we developed a rapid prototyping tool for the first cycle of our DSR.

The implementation of our UI Prototyping Tool uses various technologies. Our UI prototyping tool was developed using Angular[1], Loopback[2], and MongoDB[3]. Angular is a JavaScript (JS) framework used for building web-based applications. It provides comprehensive tools for creating dynamic and responsive UI components and handling user interactions. With Angular, we are using several other UI components which are available on Node Package Manager (npm)[4].

Loopback is a Node.js[5] framework used for building RESTful APIs. It provides an intuitive interface for creating API endpoints and managing data persistence. Loopback's support for various data sources, including relational and NoSQL databases, makes it a versatile choice for web applications. We connect our database using the data managers provided by the loopback framework. The framework's ability to generate API documentation and testing tools simplifies our development process.

MongoDB is a NoSQL document-oriented database used for storing unstructured data. We store our prototyping tool's data using a JSON[6] format in an unstructured manner. It provides a scalable and flexible solution for managing volumes of data. MongoDB's support for automatic sharding and replication ensures high availability and fault tolerance. The database's dynamic schema and rich query language make it easy to adapt to changing data requirements.

By leveraging these technologies, our UI prototyping tool delivered a powerful and user-friendly interface while ensuring efficient data retrieval and storage. Based on these technologies, we build a microservice architecture explained in the next section so that our DPs and DFs can be easily implemented.

[1] Website on Angular: https://angular.io/
[2] Website of Loopback 4: https://loopback.io/
[3] Website of MongoDB: https://www.mongodb.com/
[4] Website of NPM: https://www.npmjs.com/
[5] Website of NodeJS: https://nodejs.org/en/
[6] Website of JSON: https://developer.mozilla.org/en-US/docs/Glossary/JSON

5.4 Frontend Implementation

This section discusses the front-end implementation of our software tool. As discussed in the previous section, we developed the UI using *Angular*, a popular front-end framework. Angular provides a comprehensive architecture (see figure 5.8[7]) that includes components, templates, event binding, property binding, directives, and injectors. Angular's architecture is organized into modules containing specific functions designed to achieve particular goals. These modules can be imported and exported between different Angular applications. In each application, a root module is launched at the start of the application and imports other modules to add additional functionality.

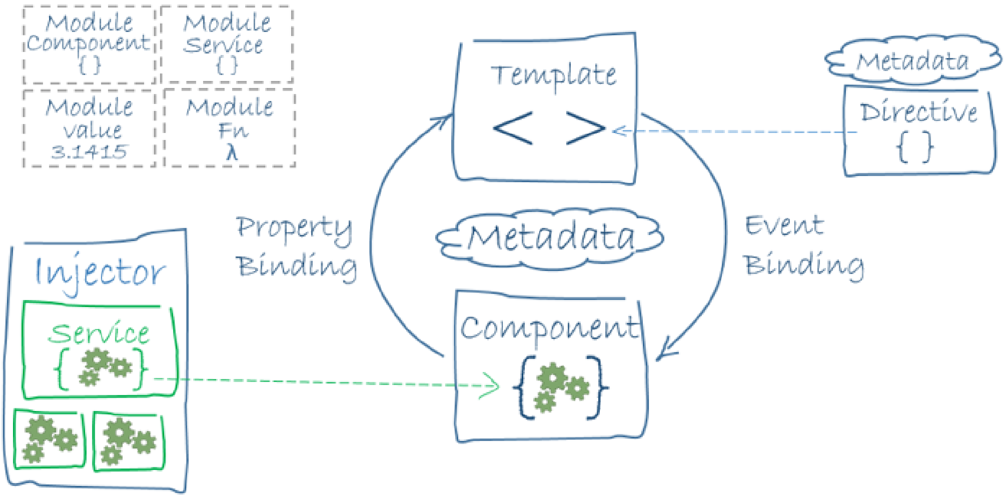


Fig. 5.8 Angular Architecture[7]

In our implementation, the Angular application has a root entry point, a module file (see Listing 5.1), containing all the necessary imports to function properly. Within our NgModule, we have imported various user-developed modules and third-party modules like *MatInputModule*, *NgxMatFileInputModule*, and *DragDropModule* for UI elements. By importing these modules, we can utilize their functions and features within our application to enhance its overall functionality and user experience. We are also using *Angular Material Design*[8] which is providing a library of pre-built UI components that follow Material Design guidelines, such as buttons, forms, and navigation menus. It saves time designing and developing UI elements from scratch and instead focuses on customizing them to fit our needs. *Nginx File Input*[9] is another third-party module we have imported into our application.

[7]Fig taken from: https://angular.io/guide/architecture
[8]Website for Angular material design: https://v7.material.angular.io/
[9]Website for Angular Material File Input https://www.npmjs.com/package/@angular-material-components/file-input

This module allows users to upload files directly from their local device, enabling us to save them to our server for future use. Finally, the *Drag and Drop module*[10] allows users to easily rearrange elements on the UI by dragging and dropping them to their desired location. Utilizing this module can provide a more interactive and dynamic user experience for prototyping.

```
// import Components
import { LoginComponent } from './components/login/login';
...

// import Modules
import { DragDropModule } from '@angular/cdk/drag-drop';
import { AppRoutingModule } from '../app-routing.module';
import { NgxMatFileInputModule } from '@angular-material-components/
    file-input';

// import Material design
import { MatSidenavModule } from '@angular/material/sidenav';
...

@NgModule({
    declarations: [
        LoginComponent,
        ...
    ],
    imports: [
        DragDropModule,
        AppRoutingModule,
        NgxMatFileInputModule,
        MatSidenavModule,
        ...
    ],
    exports: [
        LoginComponent,
        ...
    ]
})

export class AllComponentModule { }
```

Listing 5.1 The File Defining All the Modules

[10] Website for the Angular material Drag and drop https://v7.material.angular.io/cdk/drag-drop/api

Our implementation is based on the MVC architecture, which contains services and directives to talk to the server and some middleware to add a user token to every request to authenticate the user. Similarly, we explain as an example the implemented *left-panel*, *middle-panel*, and *right-panel* components. At the same time, we use the *observer pattern*[11] for the interaction between these components and various services to interact with the server.

```
import { NestedTreeControl } from '@angular/cdk/tree';
import { Component, OnInit } from '@angular/core';
...
@Component({
    templateUrl: './left-panel.component.html',
    ...
})
export class LeftPanelComponent implements OnInit {

    // Define variables ...
    constructor(private shared: CommunicationService, private
        viewService: ViewsService, ...) { }

    async ngOnInit() {
        // get master view from server
        let master: View = await firstValueFrom(this.viewService.
            getMasterView())
    }

    // CRUD of view
    addView(isMaster: boolean = false, node: View = new View()) {
        let master: View = this.dataSource.data[0]
        master.children.push(View.getView(false, name))
        this.dataSource = new MatTreeNestedDataSource<View>()
    }
    ...

    // Emit events and the other panels/components capture them
    addElement(elementName: string) {
        this.shared.setAddUIElement(elementName)
    }

}
```

Listing 5.2 The Typescript File for the Left Panel

[11]Website for Observer pattern: https://refactoring.guru/design-patterns/observer

Left panel The left-panel component contains a list of UI elements that can be added to the screen. It allows users to navigate and manipulate the elements on the screen easily, improving the UX. In our implementation (see Listing 5.2), firstly, we added a `@Component` decorator to the component's TypeScript file to specify the location of its template file, CSS file and selector. Next, we added a function that fetches the master view, if present in the database, and assigns it to a variable in the component's code. This function is called during the initialization of the component called `ngOnInit`. We also included functionality for CRUD operations for the views or UI screens in the prototyping phase. It includes adding, editing, deleting, and viewing views or UI screens. Finally, we had several functions that emit events to other panels or components that are listening to these events. It allows for communication and coordination between different parts of the application. Similarly, the *template* of the left panel contains a tree structure that displays the views and their children. This structure allows users to easily navigate through different UI screens and select the one they want to work on. The left panel template also contains various UI elements that the user can add to the UI screen in the middle panel. We have grouped them into different categories based on functionality, such as form elements, buttons, text elements, etc. Each category can be expanded or collapsed to make it easier to find the desired element.

Middle Panel In our implementation, firstly, in the constructor, the middle panel component (see Listing 5.3) creates a renderer to listen to mouse and keyboard events. This component subscribes to the required events that are emitted by other components. For example, it listens to adding UI elements from the left panel component. After the event is subscribed it adds the required UI element to the screen using the `addUIElement()` method and the provided information. It then uses the *Drag API* to make the element draggable within the virtual screen and adds various listeners. When the element is moved and placed at a certain position, the draggable interface provides the element's position, which is then added to the data structure of the current element. This component also provides CRUD functionality for the UI elements, i.e. to add, update, and remove elements. For instance, it adds listeners for the keyboard events, such as delete or backspace key press, and removes the element from the screen when the event is triggered. Similarly, we implemented a *template* for the middle panel component that allows the user to build the prototype visually. The template is designed to provide a virtual screen for the user to place UI elements. It contains a box that defines the dimensions of the screen and displays the name of the current view that is being edited. The user can place UI elements within the box by dragging and dropping them onto the screen. These elements are provided by the left panel component and can be customized using the properties panel component on the right. The middle panel component subscribes to events emitted by the left and right panel components, allowing it to update the screen in real-time.

```
@Component({...})
export class MiddlePanelComponent implements OnInit {
    // Define variables
    @ViewChild('cardContent') el!: ElementRef
    ...
    constructor(private rf:RendererFactory2,private drag:DragDrop) {
        this.renderer = this.rf.createRenderer(null, null);
    }
    async ngOnInit() {
        // Subscribe to events e.g. add UI element
        await firstValueFrom(this.shared.getAddUIElement())
        this.addUIElement()
    }
    addUIElement() {
        // Define Component
        let component: ComponentContainer = new ComponentContainer()
        component.name = this.toAddElement
        ...
        // Make it draggable
        let dragRef: DragRef = this.drag.createDrag(
           recaptchaContainer).withBoundaryElement(this.el)
        // Add Element, push to array, add listener
        const text = this.renderer.createText(this.toAddElement)
        this.elementsOnCanVas.push(component)
        this.addListener(recaptchaContainer, component)
        this.getPosition(dragRef, component.id)
        ...
    }
    async getPosition(dragRef: DragRef, id: string) {
        const val = await firstValueFrom(dragRef.ended)
        toAdd.cssProperty.dropPoint = dragRef.getFreeDragPosition()
    }
    async addListener(elm: Element, el: ComponentContainer) {
        // Delete the element
        const event = await this.renderer.listen(elm, 'keydown')
        if (event.key == 'Delete' || event.key == 'Backspace') {
            elm.remove() ...
        }
    }
}
```

Listing 5.3 The Typescript File for the Middle Panel

Right Panel In our implementation, the right panel component serves as a property editor for the selected UI element in the middle panel. It contains various functions which are explained using the Listing 5.4. It listens to events emitted by the left panel and middle panel components to update the selected element's properties. It also listens to events to edit the canvas or screen and to delete the element from the canvas. The right panel contains various input fields and dropdowns to edit the properties of the selected UI element, such as color, text, font, size, etc. When the user selects an element on the canvas, the right panel updates with the properties of that element. Similarly, when a new UI element is added to the canvas, the right panel displays the default properties for that element. Additionally, the right panel contains a function for adding new interactions to the UI element, such as `onClick` event. This function allows the user to define a JavaScript function that will be executed when the UI element is clicked. The function can be added to the element's properties in the right panel and saved to the database along with the other properties. Moreover, the template of the right panel component displays the properties of a selected UI element. It is dynamically updated based on events emitted by the middle panel component. When the middle panel emits an event to update the selected element, the right panel template displays the properties of that element. Similarly, when the middle panel emits an event to update the canvas or screen, the right panel template updates the properties displayed on the screen. Overall, the right panel template is designed to provide real-time updates to the properties of the UI element based on user actions and events.

```
// imports ...

@Component({...})
export class RightPanelComponent implements OnInit {
    // Define variables
    ...

    constructor(...) { }

    ngOnInit() {
        // Functions for subscribing to various events
        this.updateSelectedElement()
        this.getProperties()
        this.updateCanvasView()
        this.updateDeletionUIElement()
    }
    async updateCanvasView() {
        this.element =
        await firstValueFrom(this.shared.getCanvasView())
        this.elementName = this.element.name
        if (this.element.type == ContainerType.VIEW) {
            this.element.cssProperty = new CSSProperty().json
            this.element.cssProperty.height = '200'
        }
    }
    // Other functions ...

    addNewInteraction() {
        const interaction: OnClickInteraction =
        new OnClickInteraction();
        interaction.id = uuidv4();
        this.element.interactions = [interaction]
    }
}
```

Listing 5.4 The Typescript File for the Right Panel

5.5 Backend Implementation

This section discusses the backend implementation of our software tool. As discussed in the previous section, we developed the backend server using *Loopback v4*, a popular open-source Node.js framework. We implemented the backend following the MVC architecture pattern. In our implementation, we have leveraged the core concepts of LoopBack 4 to create a scalable and modular architecture for our application. According to the loopback 4 architecture (see figure 5.9 [12]), the backend implementation contains controllers, services, repositories, models, data sources, authentication middleware, and many more. One of the key concepts we utilized was the ability to generate these components using the LoopBack 4 Command-Line Interface (CLI). This approach allowed us to quickly and easily create the necessary components of our backend and reduce development time. Another important concept we incorporated was the use of decorators to define the various endpoints and routes for our application. This approach enabled us to map our data models to REST APIs, simplifying CRUD operations. Additionally, we used the powerful dependency injection system provided by LoopBack 4. This functionality allowed us to modularize our application and simplify the management of dependencies between different components. Thus, the backend implementation provides a robust and scalable architecture to handle the application's data management and authentication needs.

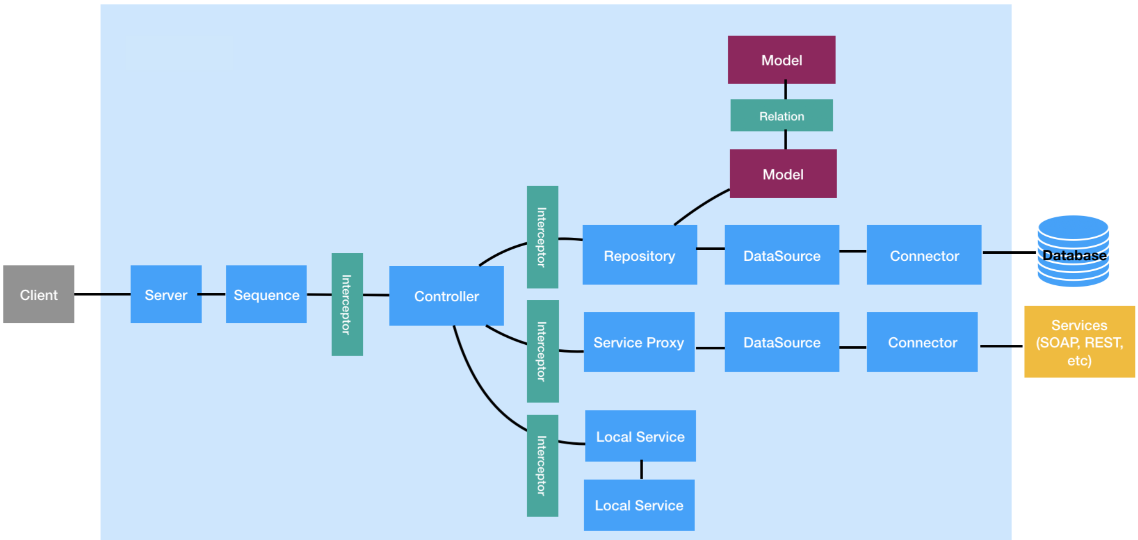


Fig. 5.9 Loopback 4 Architecture[12]

Next, we will provide an overview of some of the specific components that were generated by the LoopBack CLI and further customized by us.

[12]Image adapted from: https://loopback.io/pages/en/lb4/imgs/loopback-overview.png

Datasource In the backend implementation using LoopBack 4, we used a MongoDB database as the data source for our application. We used the CLI to generate a MongoDB data source for our application by running the command `lb4 datasource`. It generated a data source file (see Listing 5.5) with the necessary configuration for connecting to our MongoDB instance. We customized the data source configuration to include the database name and other options, such as the connection string, username, and password in an env variable `DB_CONNECTION`.

```
// imports ...

const config = {
  name: 'mongo',
  connector: 'mongodb',
  url: process.env.DB_CONNECTION,
  useNewUrlParser: true
};

// Observe application's life cycle to disconnect datasource when
   application is stopped
@lifeCycleObserver('datasource')
export class MongoDataSource extends juggler.DataSource
  implements LifeCycleObserver {
  static dataSourceName = 'mongo';
  static readonly defaultConfig = config;

  constructor(@inject('datasources.config.mongo', {optional: true})
        dsConfig: object = config) {
    super(dsConfig);
  }
}
```

Listing 5.5 The Typescript File for the MongoDB Datasource

Controller Using Loopback 4, we created RESTful APIs for our application. The REST controller was generated using the LoopBack CLI tool using the command `lb4 controller <name>`, which provided a boilerplate code structure for creating REST endpoints. To connect the REST controller to our application's business logic, we connected it to the repository and services. It allowed us to define the operations that could be performed on the model and provide implementation details. We also implemented APIs for file uploads (see the Listing 5.6), specifically CSV files and images, necessary for our prototyping tool. We created a function called `fileUpload`, which takes in a `@requestBody` containing the file to be uploaded. This function used the Nginx File input module to process file upload. Once the file is uploaded, we save it to our server folder.

```
// imports ...
export class FileUploadController {
    constructor(...) {}

    @authenticate('jwt')
    @post('/files/{key}', {
        responses: {...},
    })
    async fileUpload(
        @requestBody.file()
        request: Request,
        @inject(RestBindings.Http.RESPONSE) response: Response,
        @param.path.string('key') key: string,
    ): Promise<object> {
        this.handler(request, response, async (err: unknown) => {
        this.uploadCsv(files["files"][0]["path"], key)
        })
    }
    private uploadCsv(file: File) {
        ...
    }
}
```

Listing 5.6 The Typescript File for the Rest Controller for File Upload

Model In our implementation, we used the CLI to generate our models. A data model in Loopback is essentially an entity representing a collection of data or a resource. It can have various properties that define its schema, such as the data type, validation rules, default values, etc. In our implementation, we defined our data model with various properties (as shown in Listing 5.7) like `id` of type *string*, `key` of type *string*, an *array* of `data` and some other properties. We also defined validation rules and default values for some of these properties. Overall, a model in Loopback 4 provides a structured way of organizing and storing data in our application.

Other components We implemented an *Interceptor* using the `@authenticate('jwt')` decorator to secure the API endpoints and authenticate the user. This decorator validates the JWT token sent in the request header and authenticates the user based on the token's validity. If the token is valid, the user can access the endpoint; otherwise, a `401` unauthorized error is returned. *OpenAPI* is a specification that allows developers to define, create, and consume RESTful APIs. We used the Loopback OpenAPI specification to generate API documentation for our application. The OpenAPI specification defines the API's endpoints, parameters, request and response bodies, and authentication requirements. The specification generates interactive documentation, client libraries, and code snippets to help developers integrate with the API. We configured the OpenAPI specification by defining the application's metadata, security schemes, and endpoints in the Loopback configuration file. Finally, we used a predefined docker file generated from the Loopback CLI to deploy the application. The file specified the base image, environment variables, exposed ports, and dependencies required to run the application. The docker file contained all the necessary instructions to build and run the application in a containerized environment.

```
@model({ settings: { strict: false } })
export class DataModel extends Entity {
    @property({
        type: 'string',
        id: true,
        generated: true
    })
    id?: string;

    @property({
        type: 'string'
    })
    key?: string;

    @property({
        type: 'array',
        itemType: 'object',
        required: false,
        default: []
    })
    data: any;

    [prop: string]: any;

    constructor(data?: Partial<DataModel>) {
        super(data);
    }
}
```

Listing 5.7 The Typescript File for the Data Model

5.6 Database Schema

In our database implementation, we used MongoDB as our database, which is a non-relational database system. MongoDB is a NoSQL database management system that uses a document-oriented data model. It is designed to scale horizontally across many servers, making adding new nodes to an existing cluster easy to increase capacity and improve performance.

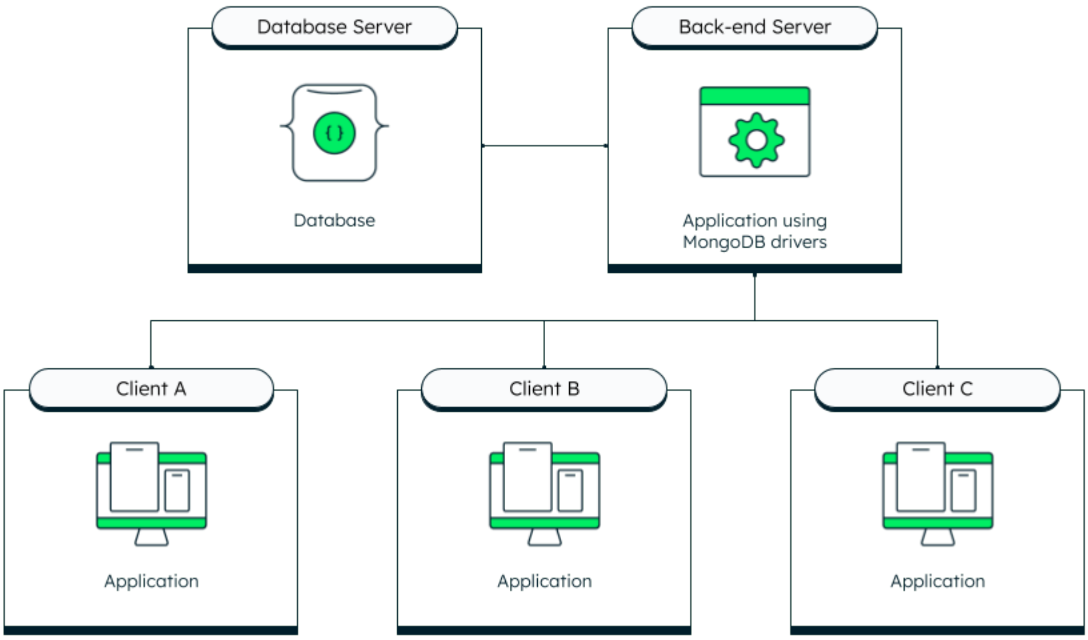


Fig. 5.10 Peer-to-peer architecture of our MongoDB implementation

MongoDB follows a peer-to-peer architecture (see figure 5.10[13]), where each node can act as a client, server, or both, and nodes can communicate directly. Our implementation architecture includes using MongoDB Atlas, a cloud-based service for managing MongoDB databases. It allows us to easily manage and scale our database with features such as automatic scaling and backups. The data is stored in a collection of documents, which can be considered similar to rows in a traditional relational database. Each document consists of a set of key-value pairs and can have a flexible schema, allowing for dynamic data structures. It is in contrast to a traditional relational database with a fixed schema. We have implemented a 3-tier architecture for our application, with the frontend, backend, and database layers. Using JS code allows us to perform CRUD operations on the database.

We created two main tables, *View* and *Experiment* (see figure 5.11 containing the Entity Relationship diagram), to store the data for our prototype and experimentation phases. The

[13] Website for MongoDB architecture: https://www.mongodb.com/basics/database-architecture

first table we created is called *View* and is used to store the various UI screens in our application for the prototyping section. The *View* table in our database schema is a crucial component of the prototyping feature. It stores various UI screens with properties like *ID, Name, property, elements, and children*. The ID and Name are both strings and ID is used for uniquely identifying each view. The *property* is a JSON object storing different properties like CSS properties, screen dimension properties, etc. Similarly, the *elements* property persists in an array of UI elements that are part of that view. These UI elements have properties like *ID, Name, cssProperties, interactions*, etc.

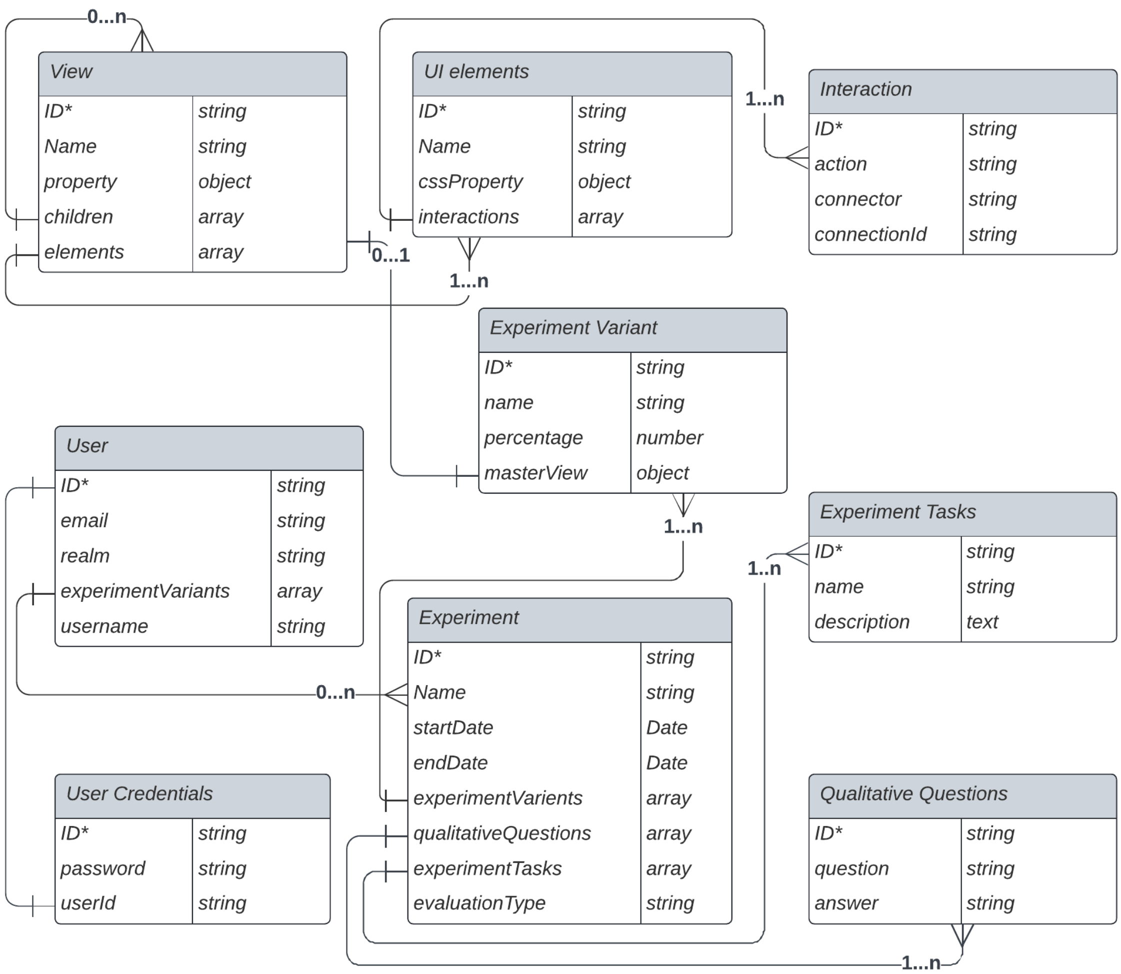


Fig. 5.11 Entity Relation Diagram of the Tool

Furthermore, the *View* table also contains the *children* property, which stores an array of *Views*. It's a cyclic reference, allowing us to create complex views by nesting views within each other. For instance, a parent view might contain two child views, each with unique UI elements and properties. By storing this information in the *View* table, our application can easily retrieve all the necessary information to render the UI, including the properties of each UI element and how they relate to each other in the view hierarchy. This information is crucial for the prototyping feature as it allows users to create and visualize complex UI designs.

The second table in our MongoDB database schema is the *Experiment* table. This table stores details related to the experiment and contains properties such as *ID, Name, startDate, endDate, experimentTasks, qualitativeQuestions, experimentVariants, and evaluationType*. The ID and Name are both strings and the ID is used for uniquely identifying an experiment in the database. The *startDate* and *endDate* properties are used to store the dates between which the experiment was conducted.

The *experimentTasks* property is an array of experiment tasks to be performed as part of the experiment. Each task has a unique ID, a name, and a description. Similarly, the *qualitativeQuestions* property is an array of qualitative questions to be answered by the participants after completing the experiment. Each question has a unique ID, a question, and an answer. The *experimentVariants* property is an array of experiment variants to be used. Each variant has a unique ID, a name, and a percentage. The *percentage* property is used to specify the percentage of participants assigned to the variant. Additionally, each variant is associated with a *masterView*, a `View` entity representing the prototype of the user interface for that variant. Finally, the *evaluationType* property is used to specify the type of evaluation that will be performed for the experiment.

In addition to the *View* and *Experiment* tables, our database schema also includes a *User* table for storing user information with different user roles such as admin, participant, etc. This table contains properties such as *ID, email, realm, username, and experimentVariants*. The *ID* property serves as the unique identifier for each user, while *email* and *username* store the user's email and username, respectively. The *realm* property stores the user role assigned to the user, and *experimentVariants* is an array of experiments assigned to the user, precisely the variant of the experiment assigned to the participant. By storing this information in the database, we can keep track of user information and their assigned experiment variants. This information is used in the experiment to ensure that participants are assigned to the correct experiment variant and that their responses are recorded accurately. Finally, the *User* table plays a critical role in our application by ensuring that user information is stored accurately and can be easily accessed and managed.

Overall, our database implementation allows us to store and manage complex hierarchies of UI screens and interactions and store data related to our experimentation phase.

Chapter 6

Evaluation

In the previous chapter, we explained the implementation of our tool. Based on that, in this chapter, we will discuss the details of the evaluation process, including the user case study (see section 6.1), experimental design (see section 6.2), execution (see section 6.3), analysis (see section 6.4), and interpretation (see section 6.5) of the results. For the setup, we recruited participants from Paderborn University.

6.1 User Case Study

To evaluate the effectiveness of our approach, we conducted a case study that involved recruiting participants, developing prototypes, and working on user scenarios. We recruited 15 participants who are students at Paderborn University. The case study is based on the evaluation stage of the first cycle of our DSR (as discussed in section 1.3). In conducting and reporting our case study research, we followed the established guidelines of Runeson and Höst [62] (see figure 6.1) to increase the quality of the study outcomes. These guidelines helped ensure our research was rigorous, transparent, and credible. By adhering to these guidelines, we aimed to provide a detailed and comprehensive account of our case study, enabling others to replicate our research and build upon our findings. This case study also aims to develop the RQ as explained in the next section.

6.2 Experimental Design

In this section, as per Runeson and Höst, we first define the objective, case study, research questions, and the methods we follow to complete the evaluation.

Objective: The objective of the user case study was to evaluate a POC tool for creating and testing UI designs. The goal was to assess the tool's ability to generate UI prototypes, create split or A/B tests, and collect participant feedback to select the best variant.

Case Study: The case study involved a user scenario where John, a UX designer, was responsible for designing the UI for a new movie-streaming app. John used the POC tool to create UI prototypes and perform split tests to select the best variant.

Research Questions (RQs): For the case study, we defined a couple of research questions.
RQ1: Can a POC tool be used to prototype the UI of a new movie-streaming app and create split or A/B tests to select the best variant of the app's interface design?
RQ2: What feedback can be gathered from participants to evaluate the effectiveness of different app interface design variants?

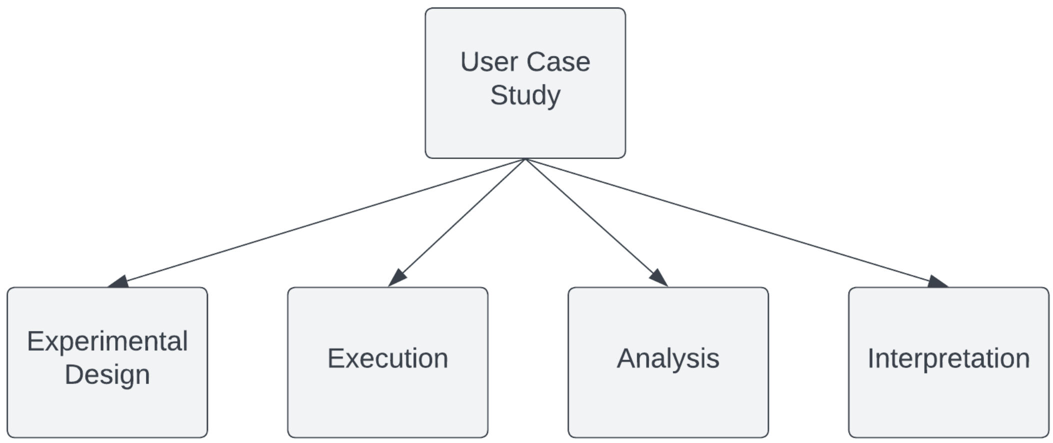


Fig. 6.1 User Case Study for Analysis

Method: For the User Case Study, we aimed to recruit diverse participants from different courses enrolled at Paderborn University to ensure a wide range of perspectives and experiences on UI prototyping and UI Experimentation. We wanted to include individuals with varying levels of experience in prototyping and user scenario development, from beginners to experts. To recruit participants, we used Doodle[1], an online tool for setting up the appointment, and a Line survey[2] hosted by our university for creating the questionnaire.

We developed a user scenario that required participants to use our UI prototyping tool hosted on the Paderborn University server. Therefore to access our tool, the participants needed to turn on the VPN[3] provided by the University. We also examined them using an open BigBlueButton[4] video conference session. We also considered the ethical clarifications by informing the participants that we were not recording the video conference session and that the survey was anonymous to ensure their privacy and encourage honest feedback while mentioning our data collection and storage strategy.

After using the tool, participants were asked to answer a questionnaire in the survey to provide qualitative and quantitative feedback for our tool and the DPs we developed. The questionnaire was designed to evaluate the usability, effectiveness, and overall satisfaction with our tool. We also asked for suggestions for improvement and gathered additional comments and feedback from the participants to gain a deeper understanding of their experiences.

[1] Website for Doodle: https://doodle.com/
[2] Website for the survey: https://umfragen.uni-paderborn.de/admin/
[3] Website for University Paderborn VPN config: https://imt.uni-paderborn.de/vpn-zugang/
[4] Website for BBB: https://open-bbb.uni-paderborn.de/

6.3 Execution

In this section, we explain the execution of our user case study in detail. First, we explain the planning phase of the execution and then explain the methodology of how we conducted the survey.

Planning: Before conducting the user case study, we carefully planned the experimental design to ensure the validity and reliability of our results. First, we defined a hypothesis to be tested. We wanted our POC tool to fulfill the Design principles and validate the DPs. To achieve this, we developed a user scenario that involved the participants using our tool to create a prototype and conduct a user scenario. We recruited diverse participants enrolled in various courses at Paderborn University, with varying experience in prototyping and user scenario development. To recruit participants for our case study, we shared a Doodle link via different communication channels, such as email and social media. The Doodle link contained the necessary information about the study, including the purpose, duration, and requirements. It also offered different time slots for the participants, ensuring that the survey could accommodate a diverse group of participants with varying schedules. This method helped us efficiently reach out to potential participants and allowed them to select a time that suited them best.

Secondly, we used the Lime survey, hosted on the Paderborn University server, to conduct a survey questionnaire to collect qualitative and quantitative feedback from the participants. The questionnaire was divided into three sections. The first section contained the System Usability Score (SUS) questionnaire, a widely used and reliable tool for measuring the usability of software systems. This section aimed to collect quantitative feedback from the participants regarding the usability of our prototype tool. The second section contained questions about the design principles we proposed in our thesis. This section aimed to collect participants' ratings and opinions on the effectiveness and feasibility of the design principles. The third section contained open-ended questions to gather participants' qualitative feedback and suggestions for improving our prototype tool and the DPs. We used the Lime survey to facilitate the data collection process and to ensure anonymity and privacy for the participants.

Methodology: The methodology of our user case study involved using a proof of concept tool accessible via UPB VPN that combined qualitative and quantitative data collection and analysis techniques. The participants were then asked to conduct a user scenario using our tool while being observed through an open BBB video conference session. The scenario was that John, a UX designer, was creating a new movie-streaming app and wanted to prototype different UI designs and conduct split tests to select the best variant. The study was conducted in several phases.

In the first phase, John explored the tool and generated an example movie streaming prototype using the headstart button. He then added new movies to the prototype using the data model feature. In the second phase, John created split tests for different app interface versions. He used the tool's A/B testing feature to create two versions of the app's view screen and changed the UI to make some changes in the variant's prototype. In the third phase, John created tasks for participants to complete and gather feedback on the app's interface. He also created a set of questionnaires, including open-ended and scale-based questions, to collect qualitative data from participants. In the fourth phase, John recruited participants or used dummy users generated from the tool to test the experiments and monitor the results. After completing the experiment, John navigated to the experiments page and analyzed the statistics to determine which version of the app's interface was more effective. Based on the feedback from the usability tests and questionnaires, John iterated on the app's design and updated the prototype in the tool. Overall, the execution of the experiment involved prototyping, split testing, task creation, and feedback gathering to design and improve the movie-streaming app's interface. An example prototype, created by one of the user participants is shown in Appendix C.

6.4 Analysis

After collecting the data through LimeSurvey and the tool, we analyzed the feedback and results to draw insights and conclusions about the effectiveness of our research questions and the software tool we designed. We also examined the qualitative feedback from the participants to gain a deeper understanding of their preferences and needs. Next, we explain the qualitative and quantitative analysis of the data collected from the participants in detail.

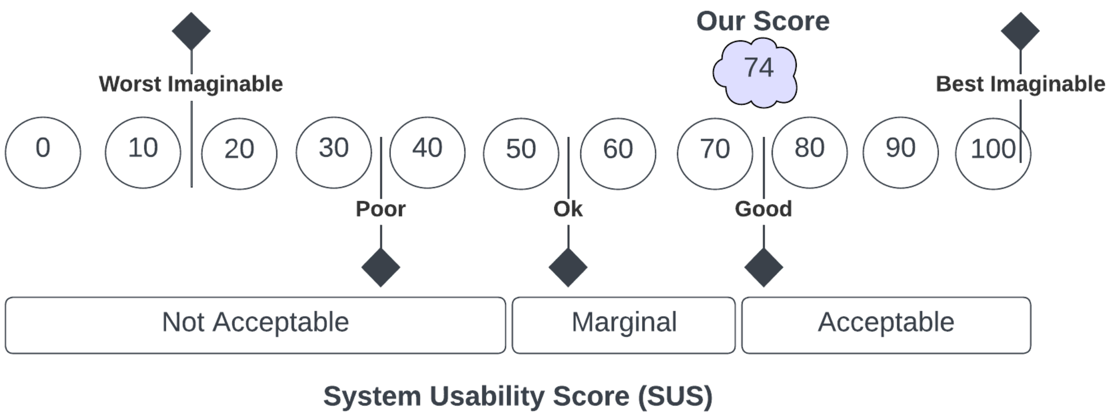


Fig. 6.2 System Usability Score of Our Tool

Quantitative analysis In the first part of the quantitative section of our analysis, we utilized the SUS to gather feedback on the usability of our tool. The SUS is a widely used and well-established questionnaire for assessing the usability of a system. It consists of 10 statements that participants rate on a 5-point scale from 1 (strongly disagree) to 5 (strongly agree). The scores from each report are then combined and transformed to create a final SUS score ranging from 0 to 100. After administering the SUS questionnaire to our participants, we computed the average score to obtain a quantitative measure of the usability of our tool. We also analyzed the individual scores for each statement to identify areas where the tool performed well and where improvements could be made. It allowed us to gain insights into the strengths and weaknesses of our tool from a quantitative perspective. It provided a basis for making data-driven decisions about improving the tool's usability. Our analysis of the SUS survey responses revealed an average score of *74* (see figure 6.2), indicating that the overall usability of the app prototype was rated as "*good*" by the participants. This score is above the average SUS score of 68, suggesting that the app prototype had a high level of usability.

The second part of our survey focused on rating the DPs of the solution tool using a 5-point scale. Participants were asked to rate their agreement with statements related to each DP. To analyze the data, we calculated the mean and standard deviation for each DP, as well as plotted a boxplot (see figure 6.3) to visualize the distribution of responses. The boxplot showed that the majority of participants had a positive rating for all DPs, with some variability in the extent of agreement. Overall, the results indicated that the design principles were well-received by participants and can be considered strengths of the app's design.

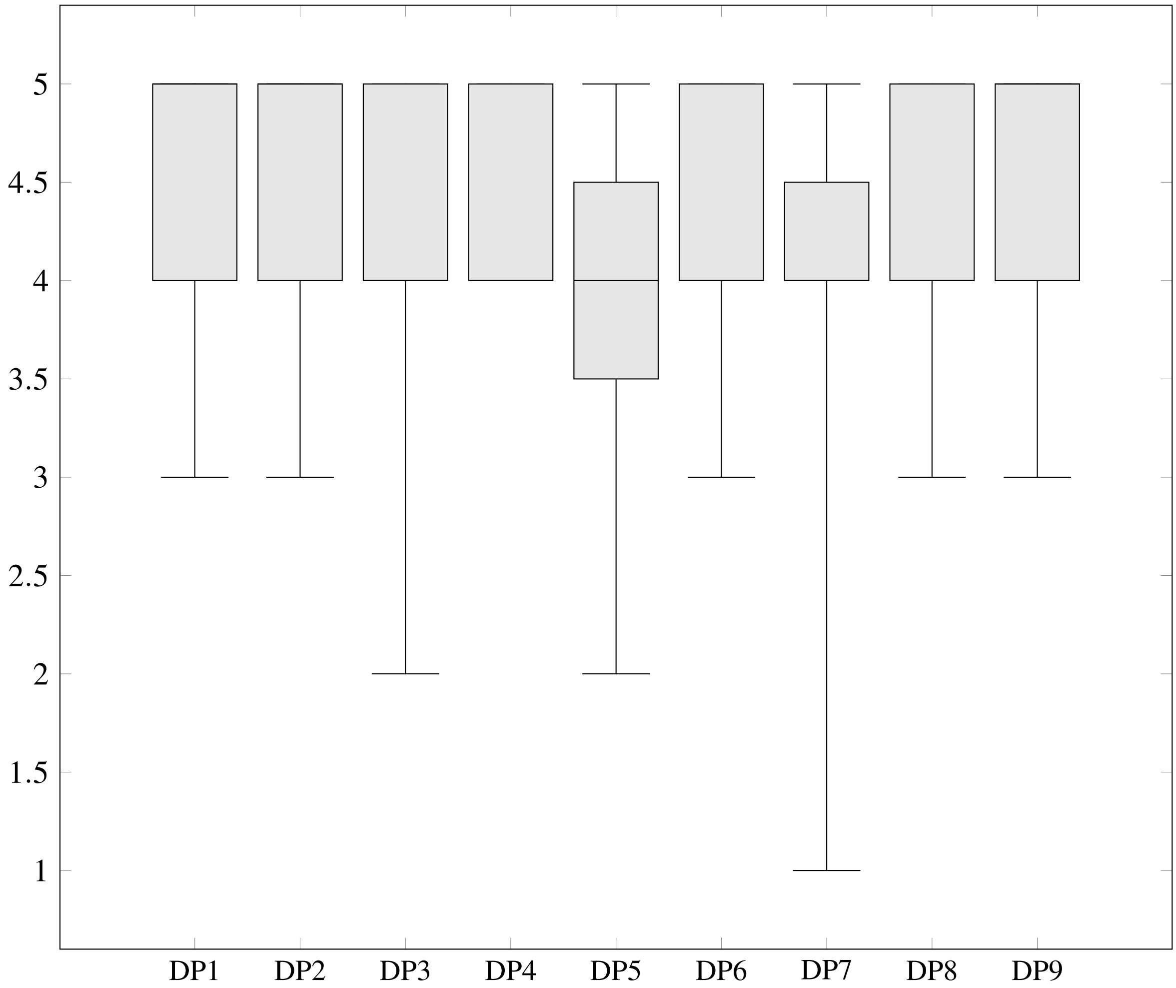


Fig. 6.3 Box Plot Analysis of the DPs

Qualitative analysis For the qualitative analysis, we had three open-ended questions in the survey. The first question asked participants if they could complete their scenario efficiently using the software and, if not, what difficulties they encountered. Many participants noted that they could complete their scenario without any issues, while a few mentioned that they had trouble navigating the software or understanding how to use certain features. The second question asked participants if there were any areas where the tool could be improved to meet their needs better. Several participants suggested adding more customization options for UI elements. The third question asked participants if they would recommend this UI prototyping tool to others and why or why not. Many participants said they would recommend the tool, citing its ease of use and ability to create and test UI prototypes quickly. Some participants noted that the tool could be improved in certain areas but still felt it was valuable overall.

6.5 Interpretation

In this section, we interpret the feedback obtained from the user case study and analyze the quantitative and qualitative analysis. The feedback and user comments will be used to identify the areas where the tool can be improved to better meet the needs of its users. We will also discuss the lessons learned from the feedback and analysis and how they can be applied to future tool iterations. The insights gained from this section will be valuable in guiding the development and improvement of the tool to provide an optimal user experience for its users.

Quantitative data The SUS is a widely used measure of the perceived usability of a system. The average SUS score of 74 (see figure 6.2) indicates that the users found the tool reasonably usable. However, the score could be higher, suggesting that there is still room for improvement. The SUS score provides a broad measure of overall usability but does not provide specific details on areas that may require improvement. Therefore, it is necessary to look at the individual user feedback and the box plots of the DPs to identify areas that need improvement.

To further analyze, we looked at the individual DPs box plots (see figure 6.3). Box plots are a way of graphically representing the distribution of data. The box represents the middle 50% of the data, with the median value marked by a line in the box. The whiskers extend to the highest and lowest data points within 1.5 times the upper and lower quartiles' interquartile range (IQR). Any data points beyond the whiskers are marked as outliers. Looking at the box plots for the DPs, we can see that DP1, DP2, DP6, DP8, and DP9 all have similar distributions with median scores of 4 or 5, upper quartiles of 5, and lower quartiles of 4. These DPs were generally well-received by users, with most participants rating them as good or excellent. On the other hand, DP3, DP5, and DP7 have lower median scores of 4, indicating that users rated them as slightly less usable than the other DPs. DP3 and DP5 have wider distributions, with lower quartiles of 2 and 3.5, respectively, indicating that some users found them particularly challenging. DP7 has a particularly low lower quartile of 1, meaning that some users found it very difficult to use. Overall, the box plot analysis suggests that some areas of the tool are particularly challenging for users, and these should be the focus of improvement efforts. Specifically, improvements to DP3, DP5, and DP7 could be prioritized to make them more usable and user-friendly in the next cycle or iterations of the DSR.

Qualitative data Three open-ended questions were asked to the participants from the responses to the question *Were there any areas where the tool could be improved to better meet your needs?*, we received 15 responses. A few respondents didn't have any specific suggestions (A_1 and A_2) (here A_n means answer from nth participant). However, some participants recommended adding new features or enhancing the existing ones. For instance, A_3 suggested including the ability to configure whether users can go back in the questionnaire, editing, and reordering created tasks and questionnaires, and providing more data analysis options. A_4 proposed adding a more intuitive button to extend experiments or tasks. A_5 and A_6 recommended adding the ability to see how much space UI components take when creating views and improving the analysis graph's size and readability. A_7 suggested enhancing the tool's appearance to make it more attractive. A_8 recommended making the User Guide more intuitive and easier to follow, possibly with examples. A_9 suggested improving the aesthetics and reducing the complexity of certain features. A_{10} recommended adding more variations for UI elements, such as drop-downs, and refining the result analysis to be more visually appealing. A_{11} suggested several improvements, such as providing freedom to change the size of elements in a view, displaying the tasks in the testing view, and reducing the number of feedback confirmations during tasks. A_{12} suggested improving the data model and views sections' explanations. A_{13} recommended improving the image variable. A_{14} and A_{15} suggested providing clearer instructions to make the tool easier to use.

Based on these responses, the tool has several areas for improvement. Some participants suggested adding new features, while others recommended improving existing features' usability and aesthetics. Some of the most common suggestions included improving the appearance and usability of the tool, refining the result analysis, and providing more thorough explanations of features in the User Guide. The development team can use these suggestions to enhance the tool's functionality and make it more user-friendly for future users.

Chapter 7

Related Work

In the previous chapter, we explained the evaluation of our solution approach and the tool we developed. Consequently, in this chapter, we present a comprehensive overview of the related work in two sections. The first section (see section 7.1) will discuss the tools commonly used for UI prototyping and UI split testing. We will provide an in-depth analysis of the strengths and limitations of each tool and highlight its unique features and functionalities. The next section (see section 7.2) focuses on comparing our tool with the existing ones. Similarly, in the sections 7.3 and 7.4 we will discuss some State Of The Art (SOAT) technologies and compare them. We compare different DRs we developed in chapter 3 with the other available tools.

7.1 Current Tools

This section will explore some of the existing softwares or Related Approach (RA) available in the market. We have identified some existing tools that are commonly used for UI prototyping and A/B testing. UX designers widely use these tools to create and test UI prototypes with users. Some of these tools are industry-standard and have been around for many years, while others are relatively new. We will briefly describe each tool's key features and capabilities to provide a comprehensive understanding of the tools we will be comparing. The comparison will help us identify the gaps and limitations of existing tools and evaluate our tool's uniqueness and innovation. Through this section, we aim to provide a comprehensive understanding of the existing tools and technologies and highlight our tool's contributions to the field of UI prototyping and UI split testing.

RA1 Figma: Figma[1] is a widely popular user interface design tool designers and design teams use for prototyping, UI design, and collaborative work. It has gained popularity due to its versatile features, user-friendly interface, and real-time collaboration capabilities, making it a go-to tool for many designers. Figma has a broad range of features that allow designers to create complex UI designs easily. It includes an extensive library of UI elements, customizable templates, and vector networks to make designing more manageable. Additionally, it offers prototyping features that allow designers to simulate complex user interactions and animations, which can help designers validate their decisions quickly. Figma is an excellent tool for teams working on the same design project, as it offers real-time collaboration. Designers can work together on the same design file, share feedback, and update the design in real-time, making it an efficient and collaborative tool for design teams.

RA2 InVision: InVision[2] is a UI prototyping and collaboration platform that allows designers to create interactive and animated prototypes for web and mobile applications. It offers a range of features, including vector-based design tools, advanced animations and interactions, and an extensive library of pre-built UI components. Its design tools are based on vector graphics, allowing for easy scaling and resizing of elements. The platform also offers various commenting and annotation tools, making it easy for team members to communicate and collaborate on design decisions. InVision also offers advanced animation and interaction capabilities, allowing designers to create complex, engaging interactions that simulate real-world user experiences. This feature includes support for animations, transitions, and gestures and the ability to create interactive elements such as buttons, menus, and forms.

RA3 Axure: Axure[3] is another popular prototyping tool used in the industry. It is a wireframing and prototyping tool that allows designers to create complex interactions and dynamic content. With Axure, designers can create interactive prototypes with conditional logic, animations, and data-driven interactions. It also offers collaboration features, which enable teams to work together on the same project in real-time. Its ability to handle complex logic and conditional interactions sets it apart from many other prototyping tools. However, the learning curve can be steep, and there may be better choices for designers looking for a quick and easy way to create simple prototypes. Finally, Axure offers integrations with other design and collaboration tools, which can benefit teams using multiple tools in their workflow.

[1] Website for Figma: https://help.figma.com/hc/en-us/articles/360040314193-Guide-to-prototyping-in-Figma
[2] Website for InVision: https://www.invisionapp.com/defined/prototype
[3] Website for InVision: https://www.axure.com/prototype

RA4 Adobe XD: Adobe XD[4] is a popular tool for designing and prototyping digital products, including websites, mobile apps, and other interactive experiences. It is widely used by UX designers, product managers, and other professionals in the design industry. One of the key features of Adobe XD is its ability to create interactive prototypes. With XD, designers can create clickable, interactive mockups of their designs, allowing them to test user flows and interactions before writing any code. These prototypes can also be shared with stakeholders and users for feedback. In addition to prototyping, it is possible to integrate external plugins to conduct A/B testing in Adobe XD. One such plugin is UserTesting, which allows designers to recruit participants and set up tasks for A/B testing directly within Adobe XD.

RA5 Proto.io: Proto.io[5] is a powerful UI prototyping tool widely used in the industry. It offers a variety of features to create interactive prototypes, including the ability to add animations, transitions, and gestures. Proto.io provides an intuitive drag-and-drop interface and supports multiple platforms, including iOS, Android, and the web. One of the standout features of Proto.io is its ability to simulate the final product. This feature allows designers to create prototypes that look and feel like the final product, providing a more realistic user experience. Proto.io also provides advanced collaboration features, making sharing and collaborating on designs with team members and stakeholders easy. Another key feature of Proto.io is its ability to create A/B tests. The tool offers a dedicated A/B testing feature, which allows designers to create multiple versions of a design and test them against each other. The tool also provides detailed analytics and user feedback, helping designers make informed design decisions.

RA6 Google Optimize Google Optimize[6] is an A/B testing and personalization tool developed by Google. It is a cloud-based tool that can help businesses optimize their website or app by creating and running experiments. Google Optimize has a user-friendly interface allowing users to create experiments without coding knowledge. One of the key features of Google Optimize is the ability to create A/B tests with different variations of a website or app. Users can set up experiments to test various elements of their website or app, such as headlines, images, or calls to action. Google Optimize also allows users to create multivariate tests that test multiple elements simultaneously. Another important feature of Google Optimize is the ability to create personalization experiments. Users can create targeted

[4]Website for Adobe XD: https://www.adobe.com/products/xd/learn/get-started-xd-prototype.html

[5]Website for Adobe XD: https://proto.io/developers/

[6]Website for Google Optimize: https://developers.google.com/optimize/devguides/experiments?technology=ga4

experiences for specific audiences based on location, behavior, or demographics. This allows businesses to create more relevant and engaging experiences for their users. Google Optimize also integrates with other Google products, such as Google Analytics, allowing users to analyze experiment results and gain insights into user behavior. Additionally, Google Optimize supports third-party integrations.

RA7 VWO VWO[7] (Visual Website Optimizer) is another popular tool for A/B testing and conversion rate optimization. It is a cloud-based platform that enables users to run experiments on their websites or mobile apps to improve the user experience and increase conversions. VWO offers a variety of features for A/B testing, including split URL testing, heatmaps, session recordings, surveys, and personalization. Users can create and run experiments with a simple WYSIWYG[8] editor or use custom code for more advanced experiments. VWO also provides a powerful targeting engine that allows users to target specific segments of their audience, such as first-time visitors, returning visitors, or users who have abandoned a shopping cart. This targeting engine helps to ensure that experiments are relevant to the user and are more likely to lead to a positive outcome.

RA8 Convertize Convertize[9] is a website optimization tool that provides a wide range of features for A/B testing, personalization, and targeting. It enables marketers to create and test multiple websites or landing page versions to determine which design, layout, or content performs best for their target audience. Convertize offers a visual editor allowing users to create different variations of their website without coding skills. Users can drag and drop elements, such as images, text, and buttons, to create different versions of their websites, which can then be tested against each other. One of the unique features of Convertize is its AI-powered Autopilot mode, which allows users to automate the optimization process. With Autopilot, Convertize automatically adapts the website to each visitor by personalizing the content, design, and layout to maximize conversions. Another notable feature of Convertize is its SmartEditor, which uses a library of pre-built templates and machine learning algorithms to recommend changes to website elements, such as headlines, images, and CTAs, that will likely improve conversions.

[7] Website for VWO: https://help.vwo.com/hc/en-us/articles/360021171954-How-to-Create-an-A-B-Test-in-VWO-
[8] Website for WYSIWYG: https://www.howtogeek.com/752396/what-is-a-wysiwyg-editor/
[9] Website for Convertize: https://docs.convertize.io/docs/how-to-do-ab-testing/

RA9 Freshmarketer Freshmarketer[10] is a comprehensive conversion optimization suite that offers a range of tools to help businesses optimize their website and improve their online performance. It allows users to create and execute A/B tests, track visitor behavior, and gather customer feedback from a single platform. Freshmarketer's A/B testing tool enables users to create multiple webpage variants and test them against each other to determine which is most effective at achieving a specific goal, such as increasing conversions or decreasing bounce rates. The platform also offers advanced targeting options to help users deliver personalized experiences to particular audience segments. In addition to A/B testing, Freshmarketer offers a range of other optimization tools, including heatmaps, session replays, and form analytics, to help users gain deeper insights into visitor behavior and identify areas for improvement. Freshmarketer's customer feedback tools, such as surveys and polls, allow users to collect valuable customer insights and use that feedback to inform future optimization efforts.

RA10 Zoho PageSense Zoho PageSense[11] is a web optimization tool designed to help businesses optimize their website's user experience, increase visitor engagement, and boost conversions. Zoho PageSense offers many features to help businesses optimize their website, including A/B testing, heatmaps, funnel analysis, and more. With Zoho PageSense's A/B testing feature, businesses can create multiple variations of their website pages and test them against each other to determine which performs best. Zoho PageSense offers a visual editor that allows users to create and edit website page variations without requiring coding knowledge. In addition to A/B testing, Zoho PageSense offers heatmaps, allowing businesses to visualize how users interact with their website pages. This feature enables companies to identify areas of their website that are not performing well and make data-driven decisions to improve the user experience.

[10] Website for Freshmarketer: https://support.freshmarketer.com/en/support/solutions/folders/50000000186/page/3

[11] Website for Zoho Pagesense: https://help.zoho.com/portal/en/kb/pagesense/run-a-b-and-split-url-tests/create-and-launch-a-test/articles/a-b-test

7.2 Comparison of Tools

In comparing the different tools based on the DRs (see table 7.1), we can see that only some tools fulfilled all the DRs. RA5 and RA6 fulfilled the most DRs, with RA5 fulfilling five out of ten DRs and RA6 fulfilling six out of ten DRs. For *Heterogeneous Users (DR1)*, only RA5, RA6, RA9, and RA10 fulfilled this requirement completely. Most of the other tools only partially fulfilled this requirement. Most tools fulfilled this requirement for *Iterative Design (DR2)*, but some only partially fulfilled it. For *Easy Development (DR3)*, RA1, RA2, RA3, RA4, and RA5 fulfilled this requirement. The rest of the tools did not fulfill it. For *Integrate Data Models (DR4)*, only RA3 fulfilled this requirement completely. Most of the other tools only partially fulfilled this requirement. For *Classified UI Variants (DR5)* and *Conduct Split Tests (DR6)*, RA5, RA6, RA7, RA8, RA9, and RA10 fulfilled this requirement, but the rest of the tools still need to.

Comparison between different Tools										
Tool	**DR1**	**DR2**	**DR3**	**DR4**	**DR5**	**DR6**	**DR7**	**DR8**	**DR9**	**DR10**
RA1	◑	●	●	◑	○	○	○	◑	○	◑
RA2	◑	●	●	◑	◑	◑	○	◑	○	◑
RA3	◑	●	●	●	○	○	○	◑	○	◑
RA4	◑	●	●	◑	◑	◑	○	◑	◑	◑
RA5	●	◑	●	○	●	●	◑	●	◑	●
RA6	●	●	○	○	●	●	◑	●	●	●
RA7	◑	●	○	○	●	●	◑	●	◑	●
RA8	◑	◑	○	○	●	●	◑	●	◑	◑
RA9	●	●	○	○	●	●	●	●	◑	◑
RA10	●	●	○	○	●	●	●	●	◑	●

Legend: No Fulfillment (○) Partial Fulfilment (◑) Complete Fulfilment (●)

Table 7.1 Table Comparing Different RAs Against DRs

For *Conduct User Tasks (DR7)*, only RA9 and RA10 fulfilled this requirement completely. The rest of the tools only partially fulfilled this requirement or did not fulfill it. Similarly, for *Collect User Feedback (DR8)*, most tools partially fulfilled this requirement, except for RA5, RA6, RA7, RA9, and RA10 which fulfilled this requirement completely. For *Aggregated Feedback (DR9)*, most tools only partially or still need to fulfill this requirement. Only RA5 fulfilled this requirement completely. For *Improvement (DR10)*, most of the tools only partially fulfilled this requirement or still need to fulfill it. Only RA5, RA6, RA7, and RA10 fulfilled this requirement completely. Overall, each tool had its strengths and weaknesses in fulfilling the different DRs, and no tool could completely fulfill all our DRs.

7.3 State-Of-The-Art Technologies

This section explores some SOAT or cutting-edge technologies for UI prototyping and A/B testing of UI. UX designers widely use these technologies to create and test UI prototypes with users. This section aims to provide a comprehensive understanding of the existing SOAT and highlight our technology's contributions to UI prototyping and UI split testing using the DRs. We will provide a comprehensive overview of each technology's key features and capabilities to understand the these technologies we will compare comprehensively. The comparison will help us identify the gaps and limitations of existing technologies and evaluate our technology's uniqueness and innovation.

RA11 Rapid software prototyping approach L. Luqi et al. [63] propose a rapid software prototyping methodology that utilizes a visual design tool and object-oriented technology. The authors argue that traditional software development methodologies need to provide adequate support for rapidly creating prototypes that can be used to communicate design ideas with stakeholders. To address this issue, they propose a framework that combines a visual design tool for creating user interfaces with object-oriented technology to build functional prototypes quickly. The authors demonstrate the effectiveness of their methodology through a case study, which shows that the proposed framework can significantly reduce the time and effort required to develop functional prototypes.

RA12 Continuous Prototyping approach Lukas Alperowitz et al. [64] propose a continuous prototyping approach to bridge the gap between design and development in continuous software engineering. The authors argue that traditional software development methodologies must provide adequate support for continuous prototyping, a critical aspect of iterative design and development processes. To address this issue, they propose a framework that enables designers and developers to continuously prototype and refine their design ideas as part of the software development process. The authors demonstrate the effectiveness of their approach through a case study, which shows that continuous prototyping can significantly improve the quality and speed of software development and increase stakeholder satisfaction.

RA13 Data-Driven Approaches to User Interface Design Pimenov et al. [65] present a case study on data-driven approaches to user interface design. The authors argue that traditional user interface design methods rely on expert opinions and intuition, which may not always reflect the actual needs and preferences of users. To address this issue, they propose a data-driven approach that utilizes user feedback and analytics to inform design decisions. The authors demonstrate the effectiveness of their approach through a case study, which shows that data-driven design can lead to significant improvements in user satisfaction and task completion rates. The paper provides insights into the potential of data-driven approaches to user interface design and highlights the importance of incorporating user feedback and analytics in the design process.

RA14 A Tool for Online Experiment-Driven Adaptation Ilias Gerostathopoulos et al. [66] propose a tool for online experiment-driven adaptation. The authors argue that traditional software development approaches must provide adequate support for online experimentation and transformation, which are critical for improving the UX and achieving business goals. To address this issue, they propose a framework that enables developers and designers to create and deploy online experiments that can be used to test and optimize different aspects of the software system. The authors demonstrate the effectiveness of their approach through a case study, which shows that the proposed tool can significantly improve the effectiveness of online experimentation and adaptation. The paper provides insights into the potential of experiment-driven transformation to improve software systems and highlights the importance of incorporating data-driven approaches in software development.

RA15 A promising tool for customer value evaluation Peitsa Hynninen et al. [67] propose using A/B testing as a promising tool for evaluating customer value. The authors argue that traditional methods of customer value evaluation, such as surveys and focus groups, may not provide accurate or reliable results due to biases and limitations. To address this issue, they propose the use of A/B testing, which is a randomized experiment that compares the effectiveness of two or more alternatives in achieving a specific goal. The authors demonstrate the effectiveness of A/B testing through a case study, which shows that A/B testing can provide valuable insights into customer preferences and behavior. The paper provides insights into the potential of A/B testing as a tool for customer value evaluation. It highlights the importance of incorporating data-driven approaches in marketing and customer experience strategies.

7.4 Comparison of SOAT

In our chapter on design chapter 3, we identified 10 DRs for the development of effective user interfaces. To evaluate the state-of-the-art research (SOAT) based on these DRs, we analyzed five studies, namely SOAT11, SOAT12, SOAT13, SOAT14, and SOAT15 (see table 7.2).

Comparison between different SOAT										
Tool	**DR1**	**DR2**	**DR3**	**DR4**	**DR5**	**DR6**	**DR7**	**DR8**	**DR9**	**DR10**
SOAT11	○	●	●	◐	○	○	○	○	○	●
SOAT12	○	●	●	●	◐	○	○	◐	○	●
SOAT13	◐	◐	◐	●	○	○	◐	●	●	◐
SOAT14	●	◐	○	○	◐	●	◐	●	◐	◐
SOAT15	●	◐	○	○	●	●	◐	◐	◐	◐

Legend: No Fulfillment (○) Partial Fulfilment (◐) Complete Fulfilment (●)

Table 7.2 Table Comparing Different SOATs Against DRs

SOAT11 fulfilled DR2, DR3, and DR10, partially fulfilled DR4, and failed to fulfill DR1, DR5, DR6, DR7, DR8, and DR9. SOAT12 fulfilled DR2, DR3, DR4, and DR10, partially fulfilled DR5 and DR8, and failed to fulfill DR1, DR6, DR7, DR9. SOAT13 partially fulfilled DR1, DR2, DR3, DR7, DR8, and DR10, fulfilled DR4 and DR8, and failed to fulfill DR5 and DR6. SOAT14 fulfilled DR1, DR6, and DR8, partially fulfilled DR2, DR5, DR7, and DR9, and failed to fulfill DR3 and DR4. Finally, SOAT15 fulfilled DR1, DR5, DR6, and DR10, partially fulfilled DR2, DR7, DR8, and DR9, and failed to fulfill DR3 and DR4.

Overall, we observed that none of the SOATs fully fulfilled our identified DRs. However, SOAT15 achieved the most DRs, with four fulfilled and four partially fulfilled, while SOAT11 and SOAT12 each achieved three fulfilled DRs. Our analysis demonstrates the need for continued research in developing effective user interfaces that fulfill all identified DRs.

Chapter 8

Conclusion

In the previous chapter we did a comparative study of different tools and technologies. Consequently, this chapter summarizes the key findings, contributions, limitations and potential areas for further research. The first section summarizes and provides an overview of the research and its contributions (see section 8.1). In contrast, the next section discusses the challenges and shortcomings encountered during the study (see section 8.2). Finally, the last section explains the possible improvements and extensions that can be made to the software tool, as well as the research questions that can be explored in future studies (see section 8.3).

8.1 Summary

In our master thesis, we identified the problems related to UI prototyping and UI experimentation and formulated a *Research Question* to address these issues. To solve this problem, we followed the DSR approach and conducted the first cycle of DSR in our thesis. As part of this approach, we conducted a *User case study* to evaluate the effectiveness of our DPs and identify any necessary corrections. The outcome of the DSR was more *refined DPs* which can be used to develop a tool for UI prototyping and experimentation using a data-driven approach for user feedback.

Software Tool: The tool is designed to provide an easy and efficient way for developers to prototype and experiment with UI designs, allowing them to receive feedback quickly and make adjustments as necessary. Similarly, the tool is designed to incorporate all the DPs as DFs and uses modern technologies to build the solution. We implemented the tool using MVC architecture and used docker and micro-service architecture for deployment.

8.2 Limitations

As with any research study, some limitations must be considered while interpreting the results. While we have made every effort to design and implement rigorous research, several potential shortcomings need to be acknowledged. It is important to understand these limitations to avoid any misconceptions about the scope and applicability of our research. In this section, we will discuss the limitations of our master thesis, which may have affected the generalizability and validity of our findings.

Limited prototype creation: One limitation of our software tool is that it only allows the user to add one prototype at a time. This limitation is due to the deployment and docker multiple instance creations during runtime, which makes it challenging to scale up the number of prototypes. It can be frustrating for users who must work on multiple prototypes simultaneously, as they would have to close one prototype before working on another. Additionally, the inability to add multiple prototypes simultaneously can slow the user's workflow and limit their productivity.

Generalizability of Findings: Conducting only one user case study in our master thesis may limit the generalizability of our results and conclusions. Since the case study was conducted with a specific set of users and a specific problem, the findings may not apply to different contexts or user groups. Additionally, the case study may have yet to capture the full range of issues and challenges users may face when using the tool. As a result, it may not be easy to draw robust and generalizable conclusions from the study. However, our thesis aimed not to provide a comprehensive evaluation of the tool but to demonstrate the feasibility of our DSR approach and the potential of our tool for UI prototyping and experimentation. Therefore, while the limitation of a single case study is acknowledged, it does not invalidate our research findings.

Usability Although we conducted a user case study to evaluate the usability of our software tool, it is important to note that the survey only provided us with a limited scope of feedback. Some aspects of the tool's usability were not addressed in the study, so there could be additional usability issues that were not identified. In some scenarios, the study's results may only partially reflect the tool's usability. Some users may find the tool difficult to use or may prefer other tools for UI prototyping and experimentation. Thus, the results could have been influenced by factors such as the participants' prior experience with similar tools, their level of expertise, or their personal preferences.

8.3 Future Work

The software tool developed in this thesis has significantly contributed to UI prototyping and UI experimentation. However, there are some areas where further improvements and developments can be made. In this section, we discuss potential future works that can enhance the functionality and performance of the software tool. These future works include the ability to layout different screen sizes and scale them, implementing voice-to-text conversion to automatically create prototypes and experiments with the user's voice instructions, extracting intelligence from data collected from user feedback, and allowing multiple users to collaborate at the same time. We also discuss the potential technologies that can be utilized to implement these future works.

Layouting for differnent screens: In future work, an essential feature to be added to the tool could be the ability to design for different screen sizes and scale them effectively. With the advent of smartphones and tablets, users access software applications on various devices, which differ in screen size and resolution. Therefore, the ability to design for different screen sizes is crucial to ensure that the software tool is accessible and usable on other devices. To achieve this, the software tool could provide pre-defined templates for popular devices or enable users to define custom screen sizes. The tool could also have a preview mode that allows the user to see how the design looks on different screen sizes, with the ability to adjust elements as needed. Additionally, the tool could automatically scale elements based on the screen size to ensure the design is proportionate and aesthetically pleasing. Incorporating these features would enhance the usability of the software tool and improve its functionality.

Voice-to-text Conversion and Coding: Incorporating voice-to-text conversion capability into the software tool could significantly improve its usability and efficiency. With this feature, users can easily create prototypes and experiments by dictating their instructions to the tool. This feature could reduce the time and effort required to develop and modify UI designs, especially for users who may have difficulty with traditional keyboard and mouse inputs. One technology that could be used for implementing this feature is natural language processing (NLP) algorithms. NLP is a field of artificial intelligence that focuses on analyzing and understanding human language. With NLP, the software tool could recognize and interpret user voice commands, converting them into actionable tasks. Additionally, machine learning techniques could be applied to improve the accuracy of the voice-to-text conversion over time as the tool learns to recognize different user accents and speech patterns. Overall, incorporating voice-to-text conversion could significantly improve the software tool, making it more accessible and user-friendly. Further research and development in the area of NLP could be a fruitful area of future work for the continued improvement of the tool.

Extract Intelligence from User feedbacks: Another future work for the software tool can involve implementing a data analysis component to extract intelligence from the data collected from user feedback. This component can process user feedback data in real time and provide insights into user behavior, preferences, and needs. The tool can dynamically adjust the prototypes and experiments with the insights provided to better match the user's expectations and preferences. Machine learning technologies such as sentiment analysis can be employed to implement this. Sentiment analysis can gauge the overall sentiment of the user feedback and continuously improve the prototype. Additionally, clustering and classification algorithms can group similar feedback and identify patterns that can help improve the tool's design and functionality. The software tool can improve its user-centered approach and enhance the overall user experience by incorporating this feature. The extracted intelligence can be used to identify areas for improvement and help guide the design decisions in future iterations of the tool.

Real-time collaboration: One potential future work for the software tool is enabling real-time collaboration among multiple users. This feature would allow different users to work on the same project simultaneously, making it easier for teams to collaborate on complex design projects. This functionality could be achieved by integrating a collaborative platform that allows multiple users to work on a single document simultaneously. Additionally, a chat or messaging feature would enable team members to communicate in real-time as they work on the project. Another technology that could be used to enable collaboration is real-time collaborative editing software. These tools allow multiple users to work on the same document simultaneously. One such technology is the Operational Transformation (OT) algorithm, which enables multiple users to make simultaneous edits to a document and ensures that those edits are synchronized across all users in real-time. Integrating this technology into the software tool would enable users to collaborate on a project in real time, providing an efficient and streamlined workflow for design teams.

Multiple prototypes To overcome the limitation *Limited prototype creation*, one possible future work and solution could be to explore container orchestration technologies such as Kubernetes[1]. Kubernetes allows for the management of containerized applications, making it easier to deploy and manage multiple prototype instances. This solution would enable our

[1] Website for Kubernetes: https://kubernetes.io/

software tool to support the addition of numerous prototypes, thereby improving the user experience and increasing productivity. By implementing Kubernetes, we could also ensure that our software tool can scale up as more users start using it, making it a more robust and reliable tool for UI prototyping and experimentation.

References

[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.

[2] David J Teece. Business models, business strategy and innovation. *Long range planning*, 43(2-3):172–194, 2010.

[3] Alan M. Davis. Software prototyping. volume 40 of *Advances in Computers*, pages 39–63. Elsevier, 1995.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] Shirley Gregor, David Jones, et al. The anatomy of a design theory. Association for Information Systems, 2007.

[11] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.

[12] Jane Webster and Richard T Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS quarterly*, pages xiii–xxiii, 2002.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] John D Gould and Clayton Lewis. Designing for usability: key principles and what designers think. *Communications of the ACM*, 28(3):300–311, 1985.

[18] Carlye A Lauff, Daniel Knight, Daria Kotys-Schwartz, and Mark E Rentschler. The role of prototypes in communication between stakeholders. *Design Studies*, 66:1–34, 2020.

[19] Jim Rudd, Ken Stern, and Scott Isensee. Low vs. high-fidelity prototyping debate. *interactions*, 3(1):76–85, 1996.

[20] Reinhard Sefelin, Manfred Tscheligi, and Verena Giller. Paper prototyping-what is it good for? a comparison of paper-and computer-based low-fidelity prototyping. In *CHI'03 extended abstracts on Human factors in computing systems*, pages 778–779, 2003.

[21] UXPin. What is a prototype. 2022.

[22] Mark van Harmelen. Exploratory user interface design using scenarios and prototypes. In *Proceedings of the Conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and Computers V*, pages 191–201, 1989.

[23] Paweł Weichbroth and Marcin Sikorski. User interface prototyping. techniques, methods and tools. *Studia Ekonomiczne*, (234):184–198, 2015.

[24] Figma high-fidelity prototype. https://www.figma.com/community/file/988854619809579450.

[25] Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland, and Tom Carey. *Human-computer interaction*. Addison-Wesley Longman Ltd., 1994.

[26] Ken Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.

[27] Pedro Szekely. User interface prototyping: Tools and techniques. In *Workshop on Software Engineering and Human-Computer Interaction*, pages 76–92. Springer, 1994.

[28] G. F. Hoffnagle and W. E. Beregi. Automating the software development process. *IBM Systems Journal*, 24(2):102–120, 1985.

[29] Ender Sahinaslan, Onder Sahinaslan, and Mehmet Sabancıoglu. Low-code application platform in meeting increasing software demands quickly: Setxrm. In *AIP Conference Proceedings*, volume 2334, page 070007. AIP Publishing LLC, 2021.

[30] Alexander C Bock and Ulrich Frank. Low-code platform. *Business & Information Systems Engineering*, 63(6):733–740, 2021.

[31] Austin Miller. Low code vs no code explained. https://blogs.bmc.com/low-code-vs-no-code/?print-posts=pdf, 2021.

[32] Raquel Sanchis and Raúl Poler. Enterprise resilience assessment—a quantitative approach. *Sustainability*, 11(16):4327, 2019.

[33] 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.

[34] Vagisha Arora. What are the three steps of low-code app development? https://www.planetcrust.com/what-are-the-three-steps-of-low-code-app-development, 2022.

[35] Christoforos Zolotas, Kyriakos C Chatzidimitriou, and Andreas L Symeonidis. Restsec: a low-code platform for generating secure by design enterprise services. *Enterprise Information Systems*, 12(8-9):1007–1033, 2018.

[36] Omg, meta-object facility. http://www.omg.org/mof/.

[37] Frédéric Jouault and Jean Bézivin. Km3: a dsl for metamodel specification. In *International Conference on Formal Methods for Open Object-Based Distributed Systems*, pages 171–185. Springer, 2006.

[38] Iyad Zikra. Implementing the unifying meta-model for enterprise modeling and model-driven development: An experience report. In *IFIP Working Conference on The Practice of Enterprise Modeling*, pages 172–187. Springer, 2012.

[39] Mariana Bexiga, Stoyan Garbatov, and João Costa Seco. Closing the gap between designers and developers in a low code ecosystem. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–10, 2020.

[40] Pierre A Akiki, Arosha K Bandara, and Yijun Yu. Adaptive model-driven user interface development systems. *ACM Computing Surveys (CSUR)*, 47(1):1–33, 2014.

[41] Frank Doesburg, Fokie Cnossen, Willem Dieperink, Wouter Bult, Anne Marie de Smet, Daan J Touw, and Maarten W Nijsten. Improved usability of a multi-infusion setup using a centralized control interface: A task-based usability test. *PloS one*, 12(8):e0183104, 2017.

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

[43] Miklos Philips. Evolving ux: experimental product design with a cxo. https://uxdesign.cc/evolving-ux-experimental-product-design-with-a-cxo-2c0865db80cc, 2020.

[44] Effie Lai-Chong Law, Virpi Roto, Marc Hassenzahl, Arnold POS Vermeeren, and Joke Kort. Understanding, scoping and defining user experience: a survey approach. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 719–728, 2009.

[45] Rasmus Ros and Per Runeson. Continuous experimentation and a/b testing: a mapping study. In *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, pages 35–41. IEEE, 2018.

[46] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 2017.

[47] Mary Poppendieck and Michael A Cusumano. Lean software development: A tutorial. *IEEE software*, 29(5):26–32, 2012.

[48] Freddy Ballé and Michael Ballé. Lean development. http://www.lean.enst.fr/wiki/pub/Lean/LesPublications/LeanDevBalleBalle.pdf, 2005.

[49] Nicolas Navet, Loïc Fejoz, Lionel Havet, and Altmeyer Sebastian. Lean model-driven development through model-interpretation: the cpal design flow. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[50] Forrest W Young. Quantitative analysis of qualitative data. *Psychometrika*, 46(4):357–388, 1981.

[51] 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.

[52] Eric Ries. *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Currency, 2011.

[53] Vidroha Debroy, Senecca Miller, and Lance Brimble. Building lean continuous integration and delivery pipelines by applying devops principles: a case study at varidesk. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 851–856, 2018.

[54] Sandra EP Silva, Robisom D Calado, Messias B Silva, and MA Nascimento. Lean startup applied in healthcare: A viable methodology for continuous improvement in the development of new products and services. *IFAC Proceedings Volumes*, 46(24):295–299, 2013.

[55] Tanja Suomalainen, Raija Kuusela, and Maarit Tihinen. Continuous planning: an important aspect of agile and lean development. *International Journal of Agile Systems and Management*, 8(2):132–162, 2015.

[56] Katrin Burmeister and Christian Schade. Are entrepreneurs' decisions more biased? an experimental investigation of the susceptibility to status quo bias. *Journal of business Venturing*, 22(3):340–362, 2007.

[57] Robert Waszkowski. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52(10):376–381, 2019. 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019.

[58] Kari Kuutti, Katja Battarbee, Simo Sade, T Mattelmaki, Turkka Keinonen, Topias Teirikko, and A-M Tornberg. Virtual prototypes in usability testing. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, pages 7–pp. IEEE, 2001.

[59] Karen M Staller. Confusing questions in qualitative inquiry: Research, interview, and analysis, 2022.

[60] Philipp Mayring. Qualitative content analysis: theoretical foundation, basic procedures and software solution. 2014.

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

[62] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14:131–164, 2009.

[63] 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.

[64] Lukas Alperowitz, Andrea Marie Weintraud, Stefan Christoph Kofler, and Bernd Bruegge. Continuous prototyping. In *2017 IEEE/ACM 3rd International Workshop on Rapid Continuous Software Engineering (RCoSE)*, pages 36–42. IEEE, 2017.

[65] Denis Pimenov, Alexander Solovyov, Nursultan Askarbekuly, and Manuel Mazzara. Data-driven approaches to user interface design: A case study. In *Journal of Physics: Conference Series*, volume 2134, page 012020. IOP Publishing, 2021.

[66] Ilias Gerostathopoulos, Ali Naci Uysal, Christian Prehofer, and Tomas Bures. A tool for online experiment-driven adaptation. In *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 100–105. IEEE, 2018.

[67] Peitsa Hynninen and Marjo Kauppinen. A/b testing: A promising tool for customer value evaluation. In *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*, pages 16–17. IEEE, 2014.

Acronyms

API Application Programming Interfaces

CE Continuous Experimentation

CLI Command-Line Interface

CRUD Create Read Update Delete

CSV Comma Separated Value

DB Database

DBMS Database Management System

DF Design Features

DML Data Modeling Language

DP Design Principles

DR Design Requirements

DS Data Structure

DSL Domain Specific Language

DSR Design Science Research

GUI Graphical User Interface

HTML Hypertext Markup Language

IT Information Technology

JS JavaScript

LCDP Low-Code Development Platform

MBSE Model-Based Software Engineering

MDD Model Driven Development

MDSE Model Driven Software Engineering

MOF Model Object Facility

MQTT Message Queue Telemetry Transport

MVC Model-View-Controller

MVP Minimum Viable Product

NCDP No-Code Development Platform

npm Node Package Manager

OMG Object Management Group

POC Proof of Concept

RA Related Approach

REST Representational State Transfer

RQ Research Question

SE Software Engineering

SOAT State Of The Art

SUS System Usability Score

UAT User Acceptance testing

UCD User-Centered Design

UI User Interface

UML Unified Modeling Language

UX User Experience

Appendix A

How to Install Our Application

Our UI Prototyping tool (as shown in figure A.1) utilizes AngularCLI[1], MongoDB[2], and NodeJS[3] in conjunction with a microcontroller architecture that employs docker containers. To use the tool, you will need to download and install these technologies beforehand. NodeJS serves as a runtime environment for executing JavaScript code and comes with the Node Package Manager (NPM), which can be used to include external JavaScript code packages in the software. Moreover, you need to install Loopback[4] which is a framework for NodeJS. AngularCLI, on the other hand, is a command-line interface (CLI) used to develop and maintain Angular applications. Additionally, MongoDB drivers can be installed or can also be the cloud version of MongoDB. Lastly, for cloning the repository, you can install the Git[5].

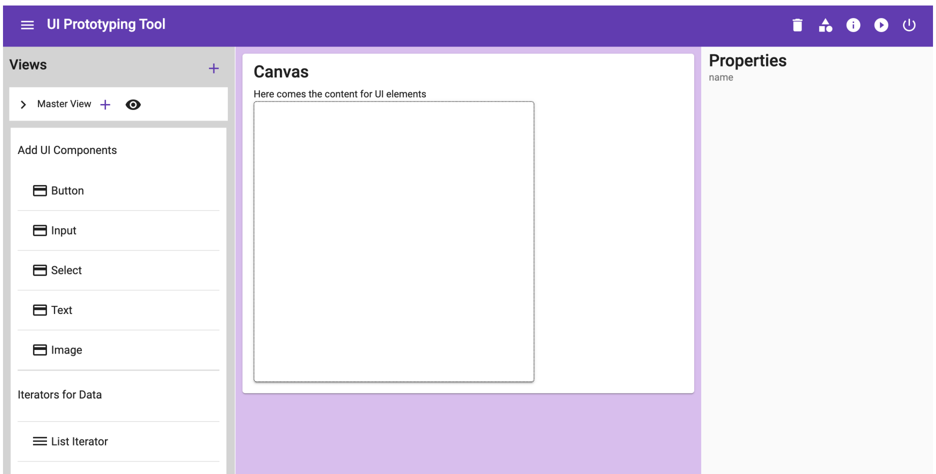


Fig. A.1 Index page of our UI Prototyping tool

[1]Website of AngularCLI: https://angular.io/cli
[2]Website of MongoDB: https://www.mongodb.com/
[3]Website of NodeJS: https://nodejs.org/en/
[4]Website of Loopback: https://loopback.io/doc/en/lb4/Getting-started.html

Based on these prerequisites, there are additional instructions for installing the tool.

```yaml
version: "3.9"
services:

  ui-prototyping:
    container_name: ui-prototyping-app-docker
    build: ./ui-prototyping-tool/
    restart: unless-stopped
    ports:
      - 8083:8083
    networks:
      - myNetwork

  ui-prototyping-no-code-app:
    container_name: ui-prototyping-no-code-app-docker
    build: ./no-code-built-app/
    restart: unless-stopped
    ports:
      - 8084:8084
    networks:
      - myNetwork

  ui-prototyping-server:
    container_name: ui-prototyping-server-docker
    build: ./ui-prototyping-server/
    ports:
      - 3001:3001
    networks:
      - myNetwork

  nginx:
    container_name: nginx-final-docker
    restart: unless-stopped
    build: .
    ports:
      - 80:80
      - 443:443
    networks:
      - myNetwork
    volumes:
      - ./certbot/conf:/etc/letsencrypt
      - ./certbot/www:/var/www/certbot
    depends_on:
      - server
```

(a) A configuration file for docker-compose file

```nginx
upstream ui-prototyping {
    server ui-prototyping-app-docker:8083;
}

upstream no-code-app {
    server ui-prototyping-no-code-app-docker:8084;
}

upstream ui-prototyping-api {
    server ui-prototyping-server-docker:3001;
}

server {

    listen 80 default_server;
    listen [::]:80 default_server;

    server_name _;

    location / {
        proxy_pass http://ui-prototyping/;
        proxy_redirect     off;
    }

    location /no-code-app/ {
        proxy_pass http://no-code-app/;
        proxy_redirect     off;
    }

    location /ui-prototyping/api/ {
        proxy_pass http://ui-prototyping-api/;
        proxy_redirect     off;
    }

    error_page  404              /404.html;


    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

(b) Nginx Configuration for our tool

Fig. A.2 Configuration Settings

Tool configuration steps:

1. **Install Docker:** To begin with, you need to install Docker on your computer. You can download it from the official website[6] based on your operating system.

[5] Website of Git installation: https://git-scm.com/downloads
[6] Website of Docker Installation: https://www.docker.com/

2. **Clone Repository:** Get the latest version of the tool from GitLab repository[7]. We use GitLab which is hosted by the University of Paderborn.

   (a) Clone the GitLab repository directly from GitLab[8].

   (b) You can fork the repository[9]

3. **Start docker daemon:** Start the docker engine or the service for running the docker containers[10].

4. **Run Application:** Run the script file (`./restart-docker.sh`) which is available to run the docker containers. This also runs the `docker-compose.yml` file internally. You can update the file if you want to change the ports or add TLS signatures (see figure A.2a).

5. **Check on Browser:** Use the UI Prototyping tool to develop the UI prototypes, create experiments and improve the prototype by opening `http://localhost/` in your web browser.

[7] Link for repo: https://git.cs.uni-paderborn.de/rakshitb/thesis
[8] Website for cloning repository instructions: https://docs.gitlab.com/ee/gitlab-basics/start-using-git.html
[9] Website for forking a repository: https://docs.gitlab.com/ee/user/project/repository/forking_workflow.html
[10] How to start docker deamon: https://docs.docker.com/config/daemon/start/

Appendix B

Literature Review Summary

In this thesis, a non-systematic literature review was conducted to explore the current state of UI prototyping tools and develop the DRs for our DSR. Several research papers were reviewed, along with an examination of some renowned UI prototyping tools. The review focused on the features and capabilities of these tools and any potential limitations or drawbacks. Through this process, valuable insights were gained into the strengths and weaknesses of existing UI prototyping tools, which can inform the development of more effective tools in the future. This section below contains a table containing the papers (see section B) we reviewed and the tools (see section B) we tested.

Software tools

In this section, we briefly discuss some of the tools and how they helped us formulate the DRs. Our non-systematic literature review identified ten tools that could help us with our solution approach (see table B.1). Among these, we found five UI prototyping tools that could be useful in designing and refining the user interface of our solution. These tools included Axure, Figma, Proto.io, InVision, and Adobe XD. Additionally, we identified five split testing tools that could be helpful in testing and improving the effectiveness of our solution. These split testing tools included Optimizely, VWO, Google Optimize, Freshmarketer, and Zoho PageSense. By incorporating these tools into our solution approach, we hope to create a user-friendly and effective solution that meets the needs of our users.

Literature and State of the Art (SOAT) technologies

Our non-systematic literature review led us to state-of-the-art research in several key areas relevant to our solution approach (see table B.2). We identified research on UI prototyping,

Software Tools			
#	**Name**	**Type**	**View**
1	Figma	UI Prototyping Tool	Link
2	InVision	UI Prototyping Tool	Link
3	Axure	UI Prototyping Tool	Link
4	Adobe XD	UI Prototyping Tool	Link
5	Proto.io	UI Prototyping Tool	Link
6	Google Optimize	Split Testing Tool	Link
7	VWO	Split Testing Tool	Link
8	Convertize	Split Testing Tool	Link
9	Freshmarketer	Split Testing Tool	Link
10	Zoho PageSense	Split Testing Tool	Link

Table B.1 Table for different UI prototyping and A/B testing tools

which can help us design and refine our solution's user interface to meet our users' needs. We also found research on low-code/no-code development, which can help us to build our solution quickly and efficiently. Additionally, we explored model-based software engineering, which can provide us with a structured approach to software development. We also discovered research on continuous experimentation, which can help us to test and improve our solution over time. Task-based usability testing was another key area of research we explored, which can help us to ensure that our solution is easy to use and meets the needs of our users. Finally, we looked into the LEAN development cycle, which can help us build our solution iteratively and incrementally while minimizing waste and maximizing user value. By incorporating the insights from these research areas into our solution approach, we hope to create a robust and effective solution that meets the needs of our users.

Literature and SOAT Research			
#	**Name**	**Type**	**View**
1	Rapid software prototyping	UI Prototyping	Link
2	Automating the software development process	UI Prototyping	Link
3	User interface prototyping	UI Prototyping	Link
4	Exploratory user interface design using scenarios and prototypes	UI Prototyping	Link
5	Low-code application platform	LCDP	Link
6	Low Code vs No Code	LCDP / NCDP	Link
7	No-code platform	NCDP	Link
8	Adaptive model-driven user interface development systems	MBSE	Link
9	OMG, Meta-Object facility	MBSE	Link
10	Turn User Goals into Task Scenarios for Usability Testing	Task Based Usability Testing	Link
11	But You Tested with Only 5 Users!	Task Based Usability Testing	Link
12	Continuous experimentation and A/B testing: a mapping study	Split Testing	Link
13	Evolving UX: experimental product design with a CXO	Split Testing	Link
14	Why the lean start-up changes everything?	LEAN	Link
15	Lean software development: A tutorial	LEAN	Link
16	Continuous planning: an important aspect of agile and lean development	LEAN	Link

Table B.2 Table For Different SOAT Research

Appendix C

User Case Study

The User case study was a 2-week research conducted with 15 students of Paderborn University. The study aimed to evaluate the usability of a software application through the use of a survey. The participants were emailed a user scenario and instructions for joining an online survey. The survey lasted 1 hour per participant and was designed to collect feedback on the usability of the software application. Throughout the 2-week study, the participants were asked to use our software tool and complete a set of tasks while using it. They were then asked to provide feedback on their experience using the software application through the online survey. The survey results were used to assess the software application's overall usability and identify any areas that required improvement. The study provided valuable insights into how the software application could be improved to enhance user experience and increase usability. Overall, the User case study was a successful project that provided valuable feedback on the usability of the software application. Out of 15 participants, we present one of the user's prototypes and user scenarios.

Email template: Here you see the email template (see below) which was sent to every participant before starting the survey.

Survey questionnaire: Next, see the figure C.1 to see the first part of the survey questionnaire, figure C.2 to see the second part of the survey questionnaire, figure C.3 to see the final part of the survey. The first part consisted of the SUS questionnaire, whereas, the second part consisted of a scale of 1-5 for rating our DPs. Finally, the last part displays the qualitative questionnaire (see figure C.3).

```
From:  Rakshit
To:  Participant
Date:  Wed 05/06/2019 15:00
Subject:  Invitation to participate in a survey
```

Dear Participant,
You have been invited to participate in a survey.
Please join this BBB room: https://open-bbb.uni-paderborn.de/b/rak-xtb-6xa-fzz

Note: Make sure you are connected to the UPB network or have UPB VPN on.

The survey is titled:
"UI Prototyping tool (Master Thesis)"

User Scenario: Imagine you are John, a UX designer creating a new movie-streaming app for a company. As the UX designer, John is responsible for designing the app's user interface. Unfortunately, John is still determining which UI fits the best for the customers. So, he is looking for solutions where he can prototype the UI and also create split or A/B tests to select the best variant. In the split tests, John wants to create various UI variants and then test the variants with different users and, finally, decide on the best variant from the user feedback.

So, John finds an interesting proof of concept tool and logs in as an admin user with the credentials: username - *admin@co.in* & password - *helloworld*. John finds the tool explanation on the top bar right and checks it. John explores the tool, finds a headstart button that displays (Generate Example UI prototype) on the top bar right and clicks on it to generate an example movie streaming prototype. This headstart option provides a basic layout and a set of UI elements that he can customize.
Then, John navigates to the option Data Model in the sidebar on the top left. He finds an already created "movies" model in the data models section. John adds new movies by clicking on the "movies" data model ...

To participate, please click on the link below.

Sincerely,
Rakshit Bhat
(rakshitb@campus.uni-paderborn.de)

Click here to do the survey:
https://umfragen.uni-paderborn.de/index.php/237123?lang=en

SUS Questionnaire: This is a screenshot (see figure C.1) where we had 10 questions for calculating the SUS.

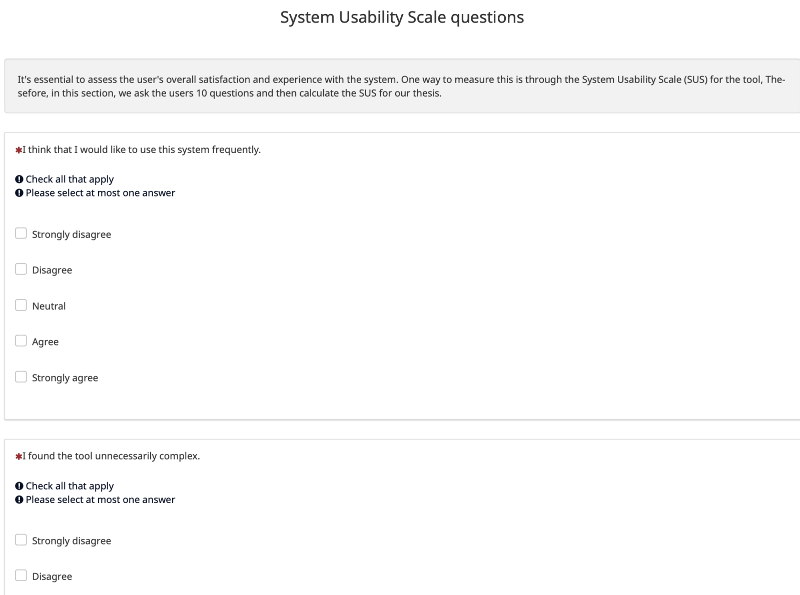


Fig. C.1 Survey SUS Questionnaire Sample

DP ratings: This is a screenshot (see figure C.1) where we had a 1-5 scale for rating our DPs.

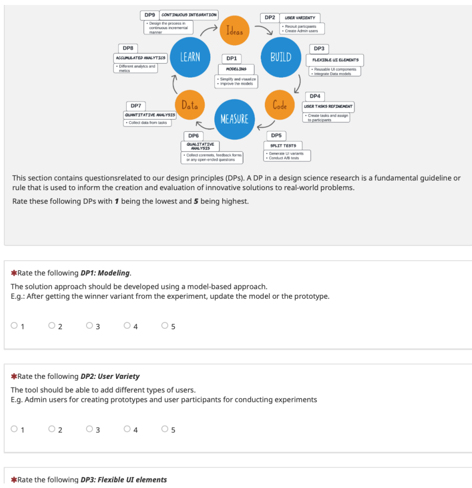


Fig. C.2 Survey DPs Questionnaire Sample

Open-ended questions: This is a screenshot (see figure C.1) where we had three open-ended questions for rating our tool and our solution approach.

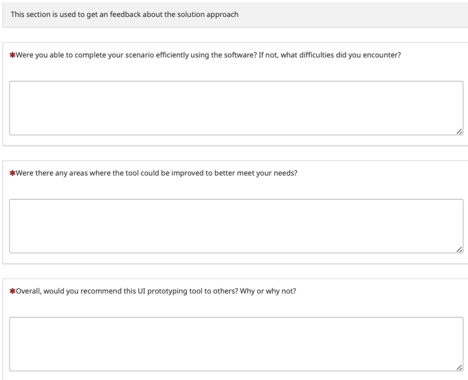


Fig. C.3 Survey Open-ended questionnaire sample

Example participant prototype: These are the three screenshots of one of the participant's prototypes that were created. See figure C.4, figure C.5, and figure C.6.

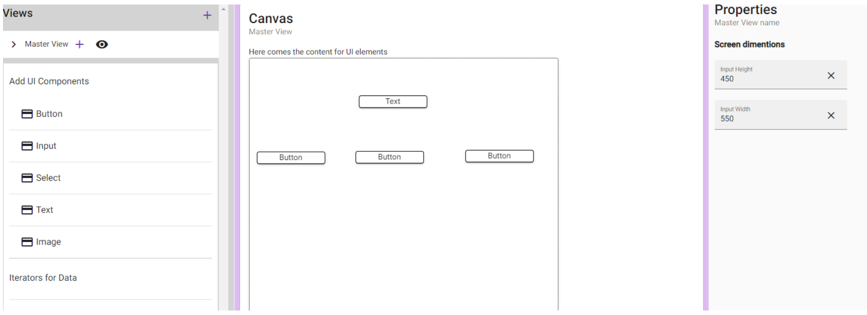


Fig. C.4 Participant Master View

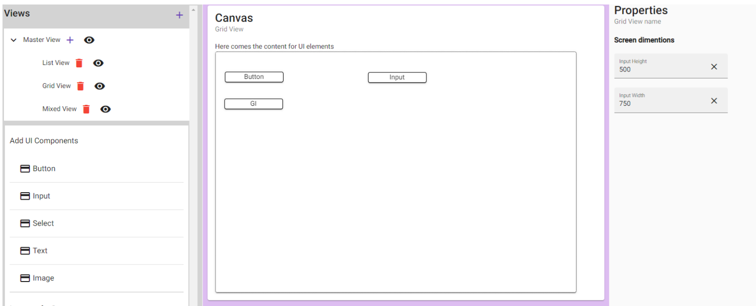


Fig. C.5 Participant Grid View

SUS responses: These are the SUS responses of all our participants exported from the Survey in a CSV format (see figure C.7).

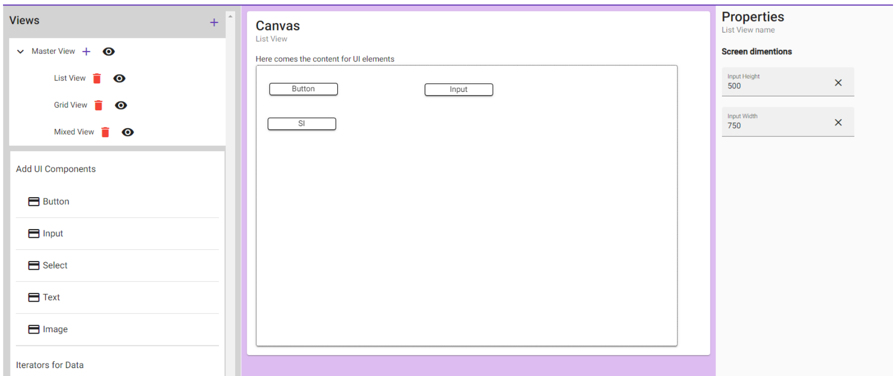


Fig. C.6 Participant List View

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	X	Y	X0	Y0	SUS
P1	5	2	5	1	5	1	4	1	5	1	24	6	19	19	***95***
P2	4	1	5	2	5	1	5	1	4	1	23	6	18	19	***92.5***
P3	4	1	4	1	4	1	5	2	4	2	21	7	16	18	***85***
P4	4	1	5	1	5	1	4	1	4	1	22	5	17	20	***92.5***
P5	4	1	4	2	3	2	4	2	4	3	19	10	14	15	***72.5***
P6	4	1	4	1	3	2	4	1	4	1	19	6	14	19	***82.5***
P7	4	4	3	4	4	2	4	2	4	3	19	15	14	10	***60***
P8	4	4	4	1	4	3	4	2	3	1	19	11	14	14	***70***
P9	4	3	4	2	4	2	4	4	4	4	20	15	15	10	***62.5***
P10	3	1	4	2	4	1	4	2	4	1	19	7	14	18	***80***
P11	3	4	3	2	4	3	2	4	2	4	14	17	9	8	***42.5***
P12	4	2	4	4	4	1	4	2	2	2	18	11	13	14	***67.5***
P13	4	2	3	4	4	2	5	1	4	1	20	10	15	15	***75***
P14	3	1	4	2	4	3	3	1	4	1	18	8	13	17	***75***
P15	4	3	3	4	4	2	4	2	3	4	18	15	13	10	***57.5***
														Average	***74***

Fig. C.7 Participant Responses SUS

DPs responses: These are the DPs responses of all our participants exported from the Survey in a CSV format (see figure C.8).

	DP1	DP2	DP3	DP4	DP5	DP6	DP7	DP8	DP9
P1	5	5	5	5	5	5	4	4	5
P2	5	5	4	4	5	3	5	5	4
P3	5	5	5	4	4	3	4	4	3
P4	3	5	5	5	5	5	4	4	4
P5	5	3	2	4	4	5	3	4	4
P6	5	4	4	5	4	5	5	4	5
P7	5	4	4	4	4	4	3	3	5
P8	5	5	4	5	4	5	4	5	5
P9	4	5	5	4	5	5	4	5	5
P10	3	4	5	5	4	3	4	4	4
P11	5	5	4	5	3	5	4	5	5
P12	4	5	4	5	5	4	5	4	5
P13	5	5	4	4	5	4	2	4	5
P14	5	4	5	4	4	5	4	5	4
P15	3	3	3	4	2	4	4	3	4

Fig. C.8 Participant Responses of DPs