Model Based Experimentation on UI Prototypes

Using Task Based Usability Testing

Rakshit Bhat

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

Advisor: Mr. Sebastian Gottschalk

Department of Computer Science
Universität Paderborn / Paderborn University

Master of Science

December 2022

I would like to *dedicate* `this` **thesis** to ...

Official Declaration
Eidesstattliche Erklärung

I hereby declare that I prepared this thesis entirely on my own and have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or a substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

Date	Rakshit Bhat

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

Chapter 1

Introduction

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

1.1 Motivation

Over the last decade, software development had a tremendous impact with increasing customer demand and requirements [2], which increases product complexity and ambiguity, significantly impacting software development. Therefore, the developers have come up with different techniques to meet this requirement criteria. Early user feedback from potential customers in the industry is crucial for creating successful software products because of the growing market uncertainties, and consumers' desire to receive integrated solutions to their issues rather than unique software developments [3]. 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 [4]. This ensures that the developers with high uncertainties in the early product development phase can improve the product by testing the underlying assumptions [5]. 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 [6]. This process of determining the best fit for the product through user feedback is called experimentation. There has been an increase in interest in the types of experimentation that can take place in product development. Software products have shown the benefits

of conducting experiments in many use cases with incremental product improvement [7]. In experimentation, the product designers design different UI variants (e.g., buttons with different colors), and the developer integrates these variants and assigns them to a distinct group of users. As per some evaluation criteria (e.g., more clicks on the button), the variant with better results is deployed for the entire set of users. So, an experiment can be valuable when it improves the software products. Hence, for experiments to be successful, they should offer one or more solutions that will benefit users.

1.2 Problem Statement

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

Problem 1: Product designers create many UI prototypes, and the developers implement them. To determine the best variant, the developers create experiments with the users [6]. 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 [8]. 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 [9]. Therefore, finding a solution that combines qualitative and quantitative data analysis is necessary.

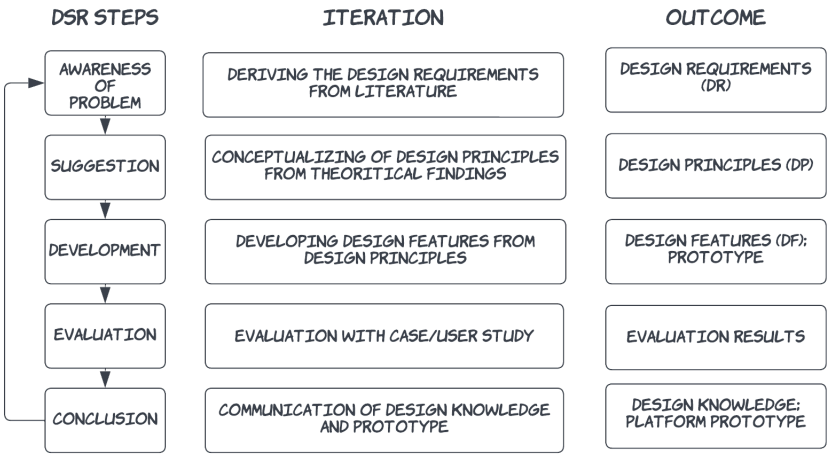


Fig. 1.1 Design Science Research Cycle [1]

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

We will conduct a design science research (DSR) study to answer our research question and obtain abstract design knowledge and an implementation tool. From the abstracted knowledge, we will obtain some Design Principles (DPs) defined for the whole process of experimentation [1]. In this design, the product designers will iteratively validate their prototypes with the users (or the crowds). Here, DPs capture and codify that knowledge by focussing on the implementer, the aim, the user, the context, the mechanism, the enactors, and the rationale [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 [1] consisting of five iteratively conducted steps (see figure 1.1). Therefore through the use of DSR, a group of issues is resolved by concentrating on a single issue and abstracting the consequences of the resolution.

1.4 Solution Approach

To solve the problems mentioned above, the designers should be able to create UI prototypes and experiments on their own on a set of users. Since we do not have a large set of users for testing the prototypes, we use supervised task-based usability testing [11]. The fundamental principle of task-based usability testing is to have the users attempt to use the prototypes to do certain activities or tasks (e.g., Locate a movie M1) and get feedback (e.g., the time required for the task to be completed by the user). We propose to use `Low-code` or `No-Code` approach to achieve this. This approach helps to have a UI for the designers to understand, develop, and create experiments and tasks with the software prototypes [12]. So, the designers would be able to create the UI prototypes and their variants, assign them to the users in an experiment, get feedback from the users and decide on the best prototype. At the same time, the low-code has become more accessible for Model-driven development [13]. 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.

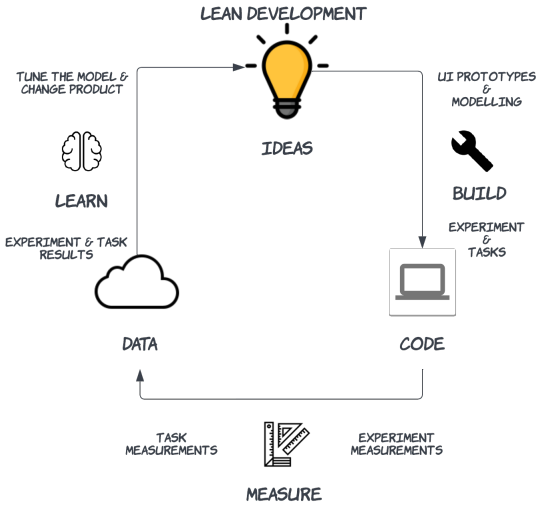


Fig. 1.2 LEAN Development technique

In our solution, we use the LEAN development technique (see figure 1.2) for development as it is used to develop customers friendly products [14]. Using LEAN, the company creates a Minimum Viable Product (MVP) throughout development, tests it with potential customers, and leverages their input to make incremental changes. While this technique can be used for every product, there are also approaches specific to software products. LEAN development

technique can be divided into a Build, Measure, and Learn cycle. In the `(1) Build` phase, we plan to create the `UI Prototypes`, `Models`, `Experiments`, and `Tasks` for the users. In the `(2) Measure` phase, we plan to assign the Experiments and Tasks to the users and measure the `Task and the Experiment measurements` and perform some analysis on the data received. And finally, in the `(3) Learn` phase, we display the `Analyses results`, `Tune` our models to decide the better variant among the others, and `Modify` the prototype. As per the figure 1.2, we complete one cycle of iteration and start a new one with the updated prototype.

Chapter 2

Background

To build the foundation of our approach, we present the UI Prototyping (see section 2.1), Low and No code (see section 2.2), Model Based Software Engineering (MBSE) (see section 2.3), Task Based Usability Testing (see section 2.4), and Experimental Product Design (see section 2.5).

2.1 UI Prototyping

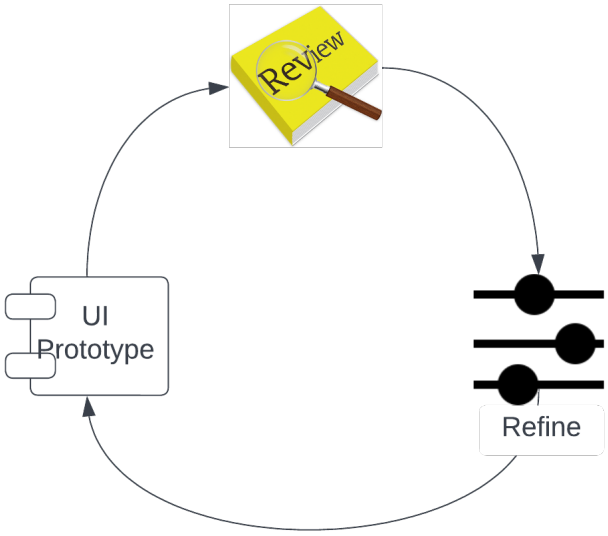
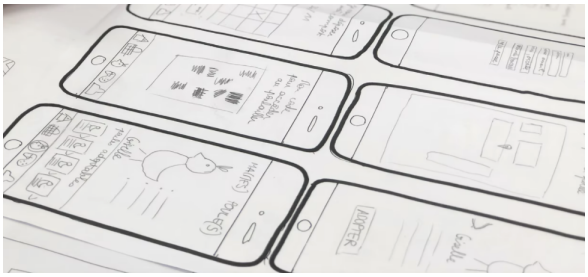


Fig. 2.1 Prototyping steps for an iterative development

User Interface (UI) prototyping is an evaluation and testing technique according to User-Centred Design (UCD) methodology since the 1990s [15]. A UI prototype is a mock-up of the User interface. Before a real product is created, prototypes are used to test the usability and user experience of the interface. The evaluation of prototypes by users is a fundamental part of all iterative approaches for IT project management, especially agile methodologies [16]. And to build an exemplary user interface, iterative refinement must be used: develop a preliminary version of the user interface, test it with people, and make as many revisions as possible [17]. Figure 2.1 shows a cycle that can be used for the iterative development of prototypes. The designers start the process by developing the UI prototypes, which stakeholders (e.g., customers and product managers) review. The UI prototypes are refined from the feedback received, and the cycle is reiterated. Therefore, designing UI prototypes enables designers and stakeholders to communicate more effectively.

An interactive prototype helps visualize design concepts and 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

(a) Sketces and whiteboard for prototyping in initial stages of development



(b) Example on how the paper-based prototypes are developed [18]

Fig. 2.2 Low fidelity prototyping

complex and expensive, prototyping is a crucial technique [19]. Simultaneously, software prototypes might exclude many requirements, making the software more accessible, smaller,

and less expensive to construct and change [19]. Similarly, usability testing to validate user requirements and prototype functionality is part of the evaluation process for UI prototypes. When prototyping is used, there is usually more contact between the designers and users, resulting in fewer usability flaws and corrections at the end of development. The main difference between a prototype and a software application is that in a prototype, the displays are designed images with no additional capabilities to display the design and flow. The mockups are converted into real UI elements in a software application, and a flow is available.

From a paper to HTML code, everything could be a prototype. Jim Rudd et al. [20] have compared high and low-fidelity prototyping, explaining the advantages and disadvantages. *Low-fidelity* prototypes (see figure 2.2a) are usually limited functions with little interaction prototyping effort. They mainly focus on explaining concepts, design alternatives, and screen layouts. Storyboard presentations, cards, and proof of concept prototypes come under this category. UI designers use simple text, lines, and forms to hand-draw concepts. Instead of aesthetics, the focus is on speed and many ideas. To simulate user flows (as shown by figure 2.2b), designers lay paper screens on the floor, table, or pinned to a board. These prototypes emphasize communicating, educating, and informing rather than training, testing, and codification. The advantages of low-fidelity prototypes are rapid development, lower development cost, addressing issues, and usefulness for a proof-of-concept. Similarly, the disadvantages include limited error checking, difficulty with usability testing, navigation, flow limitation, etc.

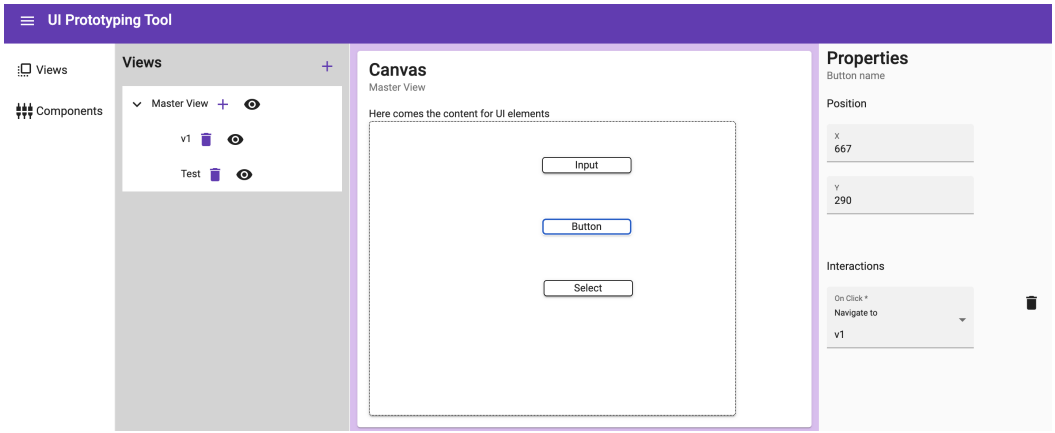


Fig. 2.3 High Fidelity prototype: Model-based UI Prototyping

Contrary to low-fidelity prototypes, *High-fidelity* prototypes (see figure 2.3) have full functionality and focus on flow, and the user models of the system [21]. The users can operate these prototypes, and the developers can collect information from the users through measurements. Other advantages of high-fidelity prototypes are that they are user-driven,

used for navigation and tests, and can also be served as a marketing tool for attracting potential customers [20].

Steps for creating UI Prototypes:

- **Learn about your consumers' requirements:** Here, our primary goal is to understand the principles that underlie our client's values to better our service to suit their needs. In this step, the design team meets with the product management team and brainstorms to develop some use cases and end-user needs. After the discussions, the team can present the customers to make sure they are on the right track and the design matches their requirements.
- **Sketch out the product:** This is an important step for UI designers. In this step, Low fidelity prototypes can be developed e.g., The design team can create the paper prototypes and start brainstorming on different possible options. Further, the team can create high-fidelity prototypes by using some tools. There are some tools available like Figma[1], Invision[2], Adobe XD[3], Axure RP[4] and many more. Using these tools, the designer can develop some high-fidelity digital prototypes.
- **Develop into Software:** This step includes the software developers and the designers coordinating and developing the software in iterations. The designers would give feedback to the software developers and they would be continuously developing the product.

After performing these steps, it is necessary to test if everything is working as expected by the user requirements. It is necessary to ensure that the product is usable for the end-users and that developers get all the essential features in the evaluation phase.

[1] Figma: https://www.figma.com/
[2] Invion AG https://www.invisionapp.com/
[3] Adobe XD: https://www.adobe.com/products/xd.html
[4] Axure RP: https://www.axure.com/

2.2 Low Code / No Code Development Platform

Low Code 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). Similarly, there is another technique called no code supported by the *No Code Development Platform* (NCDP) [22]. Unlike low code, no-code platforms require no programming skills because they offer some pre-build templates for building the apps. Using the visual user interface and ready-made automatic tools on these application development platforms, it is feasible to create apps relatively quickly. These technologies are visual app development methods that speed up app creation using drag-and-drop editors and pre-built components. By allowing software developers to concentrate on challenging coding areas, low code / no code accelerates platform development. Due to its simplicity, flexibility, and low cost, companies have started using this platform to meet the high demands of software development and digitalization. Low code is a software development method that uses less human coding to enable users to construct and manage programs efficiently [23]. Additionally, it lowers the expenses associated with initial installation, training, distribution, and maintenance [24].

*Main Steps Of **Low-Code** App Development*

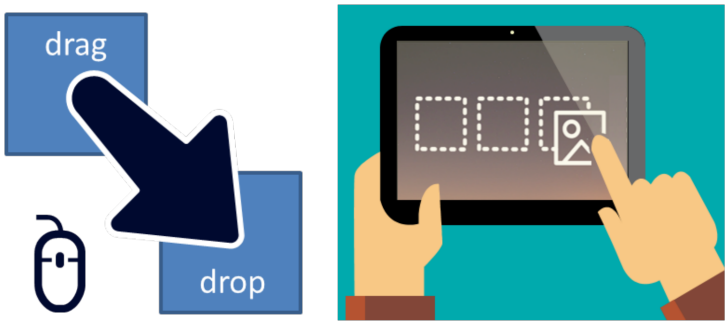


Fig. 2.4 Step 1: Building of Low Code App development

Building: In this step (as shown in the figure 2.4), 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.

Fig. 2.5 Step 2: Testing of Low Code App development

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 (as shown in figure 2.5), we still need to perform the test on the entire application after integration for scrutiny, which is performed in this step.

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. And the app's deployment can be done on various services (see figure 2.6) like Cloud-based services. This means that the LCDP gives freedom to the users in deploying the applications for the customers with just one click.

Some features that make the LCDP or NCDP favorable for development [23, 25, 26]

- **Re-usable components:** LCDP usually contains pre-configured components, modules, logic, templates, and many more that can be customized per customer requirements.

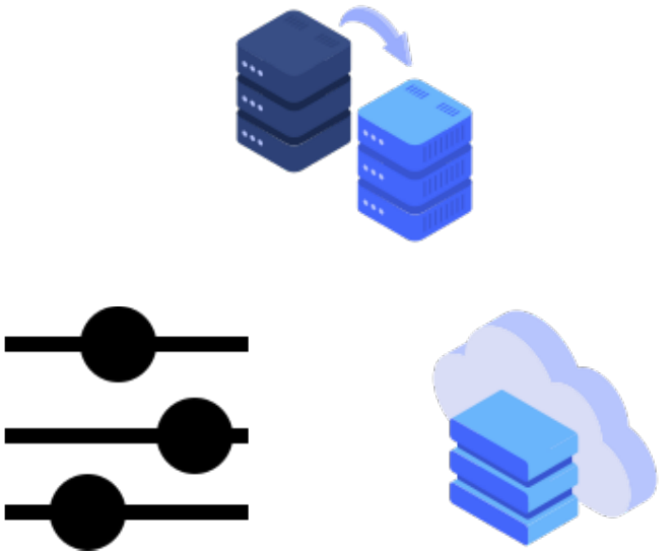


Fig. 2.6 Step 3: Deployment of Low Code App development

This helps to Build apps with consistency and scalability. Also, the scope for testing decreases as the components of the LCDP is pre-tested for security and performance.

- **Flexible and Model-driven development:** The drag-and-drop functionality helps developers increase productivity, whereas citizen developers build all apps. LCDP ensures to have a GUI useful for the non-developers. Therefore, it encourages the stakeholders outside the IT department to participate in development. Low code has become famous in model-driven development. The model-driven product helps to visualize how the app works while it is built.

- **Easy deployment:** This feature ensures that the artifacts are created, and the platform is ready to deploy. LCDP has packages that contain various deployment packages (e.g., `Dockerfile` to deploy on docker[5])

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 [23].

[5]Docker: https://www.docker.com/

2.3 Model-based Software Engineering

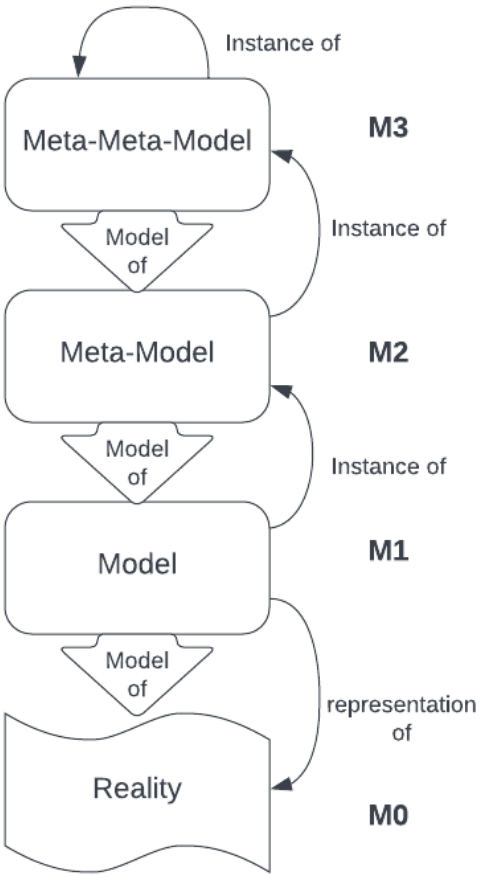


Fig. 2.7 Model Object Facility (MOF) levels

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 2.7).

Meta Models: A meta-model is defined as a model of a model or a simplified version of an actual model of a system of interest or a software application. Meta models can act as an intermediary between the input and the output relations and represent mathematical relations or algorithms. A model is usually defined as an abstraction of real-world entities, and a

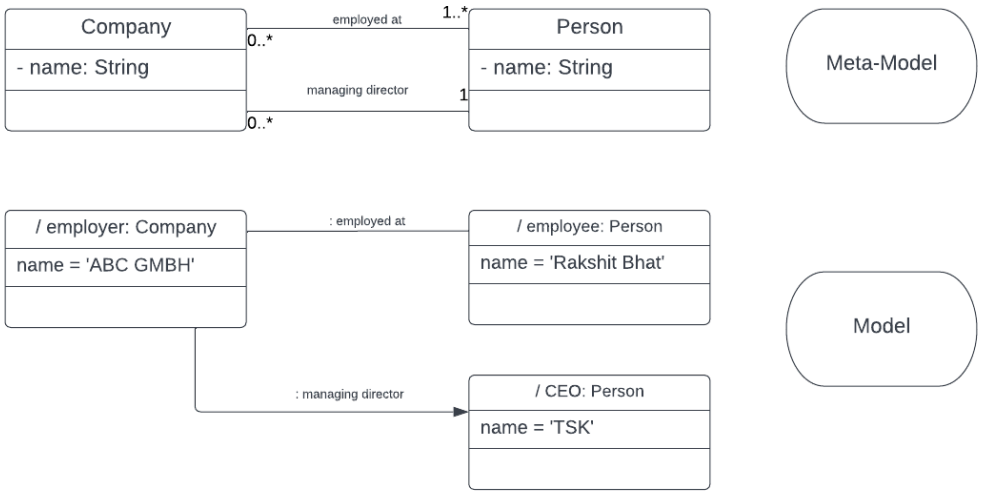


Fig. 2.8 Meta models and Models

meta-model is another abstraction of the models. Similarly, metamodelling is analyzing and developing the rules, theories, and models helpful in constructing meta-models. From figure 2.8, the meta-models are very similar to the UML class diagrams, are used to define the models or grammar of the models, and the models are instances of them. The meta-model is technically described using its metamodeling structures which is self-describing. This offers a consistent semantic approach for Model Driven Architecture (MDA) artifacts representing models and meta-models.

Models: As shown in figure 2.7, a *Model* is a simplified representation of the real world or an unavoidable reality. Before any code is written, the model is a schematic that describes how the software system should function. From the example 2.8, 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 [27]. In this research, 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, modeling languages are used to codify the UIs in different companies. Cameleon [28] is a framework that divides the UI into several elements to maximize the parts' reusability in various user, platform, and environment situations. A platform-independent abstract UI, a platform-dependent concrete UI, and a device-dependent final UI are the layers the framework offers to accomplish this.

However, these modeling languages do not emphasize offering visual notations to aid non-developers in creating such interfaces. For our research, we use a recent method [29], 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 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 [30] by measuring the percentage of the tasks the users complete. To observe the participants, they need to be assigned some "activities" or *tasks*. 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 [31], 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 'ABC movie'.
Good Task: Watch a movie with more than 6.5/10 ratings.

***(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. E.g., 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. From *Videostreamer* application: The goal is to find a movie and show times.
Bad Task: You want to see 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 seeing 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.

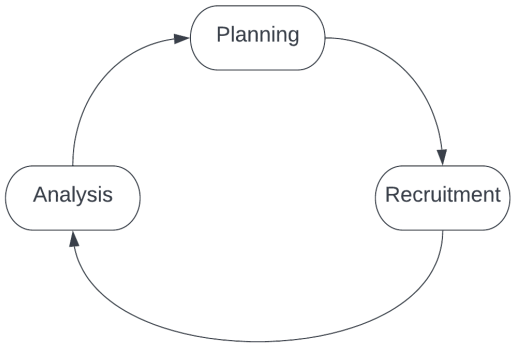


Fig. 2.9 Task-based Usability Testing steps

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 activities.

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.

2.5 Experimentation

Experimental Product Design (EPD) has become integral to optimizing UI and *User Experience (UX)*. 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 [32]. 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. For conducting experiments, various steps like **Users distribution**, **Continuous**

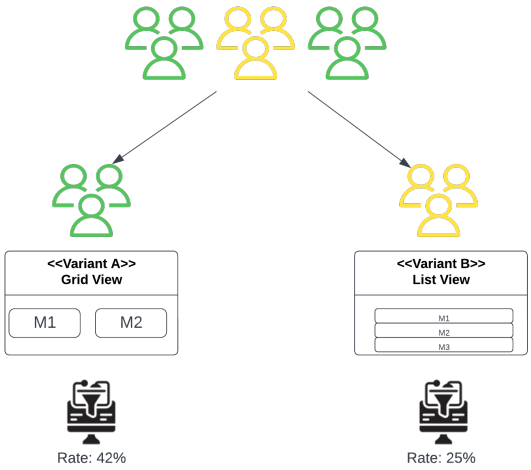


Fig. 2.10 A/B Testing

Experimentation, **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 [33]. 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 2.10, a between-subject design is used where only one of the variants is assigned to the user.

Continuous Experimentation and Variants distribution: Continuous experimentation (CE) primarily aims to get users' feedback on the software product's evolution. As per figure 2.10, CE generally uses A/B/n 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 2.10). 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 [34]. CE is an extension to the introduction of continuous integration and deployment, and all are summarized as constant software engineering [35].

Examples of A/B Testing: Some tools conduct A/B Testing and are used to improve User Interface. *Optimizely*[6], *Adobe Target*[7], *Petri*[8], *Google Analytics and Google Optimize*[9] are some of the research tools which can be used to conduct A/B Testing.

[6]Optimizely: https://www.optimizely.com/
[7]Adobe Target: https://business.adobe.com/in/products/target/adobe-target.html
[8]Petri: https://github.com/wix-incubator/petri
[9]Google Analytics and Google Optimize: https://vwo.com/compare/google-optimize/

Chapter 3

Related Work

3.1 State of the Art Research

3.2 Comparison

3. Do not use ‘ditto’ signs or any other such convention to repeat a previous value. In many circumstances a blank will serve just as well. If it won’t, then repeat the value.

Table 3.1 A badly formatted table

	Species I		Species II	
Dental measurement	mean	SD	mean	SD
I1MD	6.23	0.91	5.2	0.7
I1LL	7.48	0.56	8.7	0.71
I2MD	3.99	0.63	4.22	0.54
I2LL	6.81	0.02	6.66	0.01
CMD	13.47	0.09	10.55	0.05
CBL	11.88	0.05	13.11	0.04

Table 3.2 Even better looking table using booktabs

Dental measurement	Species I		Species II	
	mean	SD	mean	SD
I1MD	6.23	0.91	5.2	0.7
I1LL	7.48	0.56	8.7	0.71
I2MD	3.99	0.63	4.22	0.54
I2LL	6.81	0.02	6.66	0.01
CMD	13.47	0.09	10.55	0.05
CBL	11.88	0.05	13.11	0.04

Chapter 4

Design

4.1 Design Principles

4.2 Build

4.3 Measure

4.4 Learn

Chapter 5

Solution Implementation

5.1 Design Features

Chapter 6

Evaluation

6.1 User Case Study

6.2 Limitations and Risks

Chapter 7

Conclusion

7.1 Conclusion

7.2 Future Work

References

[1] 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.

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

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

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

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

[6] 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.

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

[8] 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.

[9] 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.

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

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

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

[13] 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.

[14] 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.

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

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

[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] UXPin. What is a prototype. 2022.

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

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

[21] 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.

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

[23] 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.

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

[25] Felicien Ihirwe, Davide Di Ruscio, Silvia Mazzini, Pierluigi Pierini, and Alfonso Pierantonio. Low-code engineering for internet of things: a state of research. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–8, 2020.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

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

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

[33] 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.

[34] 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.

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

Appendix A

How to install LaTeX

Windows OS

TeXLive package - full version

1. Download the TeXLive ISO (2.2GB) from
   https://www.tug.org/texlive/
2. Download WinCDEmu (if you don't have a virtual drive) from
   http://wincdemu.sysprogs.org/download/
3. To install Windows CD Emulator follow the instructions at
   http://wincdemu.sysprogs.org/tutorials/install/
4. Right click the iso and mount it using the WinCDEmu as shown in
   http://wincdemu.sysprogs.org/tutorials/mount/
5. Open your virtual drive and run setup.pl

or

Basic MikTeX - TeX distribution

1. Download Basic-MiKTeX(32bit or 64bit) from
   http://miktex.org/download
2. Run the installer
3. To add a new package go to Start » All Programs » MikTex » Maintenance (Admin)
   and choose Package Manager

4. Select or search for packages to install

TexStudio - TEX editor

1. Download TexStudio from
   http://texstudio.sourceforge.net/#downloads

2. Run the installer

Mac OS X

MacTeX - TEX distribution

1. Download the file from
   https://www.tug.org/mactex/

2. Extract and double click to run the installer. It does the entire configuration, sit back and relax.

TexStudio - TEX editor

1. Download TexStudio from
   http://texstudio.sourceforge.net/#downloads

2. Extract and Start

Unix/Linux

TeXLive - TEX distribution

Getting the distribution:

1. TexLive can be downloaded from
   http://www.tug.org/texlive/acquire-netinstall.html.

2. TexLive is provided by most operating system you can use (rpm,apt-get or yum) to get TexLive distributions

Installation

1. Mount the ISO file in the mnt directory

   ```
   mount -t iso9660 -o ro,loop,noauto /your/texlive####.iso /mnt
   ```

2. Install wget on your OS (use rpm, apt-get or yum install)

3. Run the installer script install-tl.

   ```
   cd /your/download/directory
   ./install-tl
   ```

4. Enter command 'i' for installation

5. Post-Installation configuration:
   http://www.tug.org/texlive/doc/texlive-en/texlive-en.html#x1-320003.4.1

6. Set the path for the directory of TexLive binaries in your .bashrc file

For 32bit OS

For Bourne-compatible shells such as bash, and using Intel x86 GNU/Linux and a default directory setup as an example, the file to edit might be

```
edit $~/.bashrc file and add following lines
PATH=/usr/local/texlive/2011/bin/i386-linux:$PATH;
export PATH
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;
export MANPATH
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;
export INFOPATH
```

For 64bit OS

```
edit $~/.bashrc file and add following lines
PATH=/usr/local/texlive/2011/bin/x86_64-linux:$PATH;
export PATH
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;
export MANPATH
```

```
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;
export INFOPATH
```

Fedora/RedHat/CentOS:

```
sudo yum install texlive
sudo yum install psutils
```

SUSE:

```
sudo zypper install texlive
```

Debian/Ubuntu:

```
sudo apt-get install texlive texlive-latex-extra
sudo apt-get install psutils
```

Appendix B

Installing the CUED class file

LaTeX.cls files can be accessed system-wide when they are placed in the <texmf>/tex/latex directory, where <texmf> is the root directory of the user's TeXinstallation. On systems that have a local texmf tree (<texmflocal>), which may be named "texmf-local" or "localtexmf", it may be advisable to install packages in <texmflocal>, rather than <texmf> as the contents of the former, unlike that of the latter, are preserved after the LaTeXsystem is reinstalled and/or upgraded.

It is recommended that the user create a subdirectory <texmf>/tex/latex/CUED for all CUED related LaTeXclass and package files. On some LaTeXsystems, the directory look-up tables will need to be refreshed after making additions or deletions to the system files. For TeXLive systems this is accomplished via executing "texhash" as root. MIKTeXusers can run "initexmf -u" to accomplish the same thing.

Users not willing or able to install the files system-wide can install them in their personal directories, but will then have to provide the path (full or relative) in addition to the filename when referring to them in LaTeX.