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

November 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.

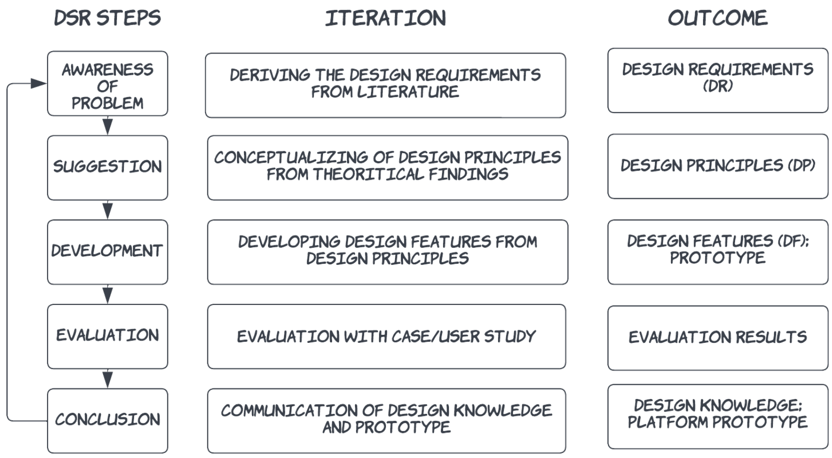


Fig. 1.1 Design Science Research Cycle [1]

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.

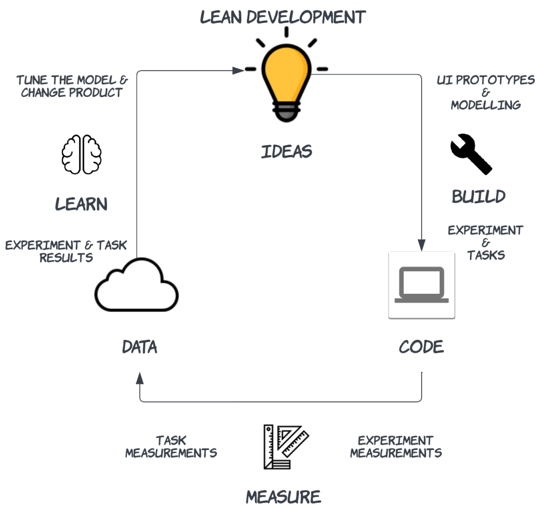


Fig. 1.2 LEAN Development technique

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 usage of crowdsourcing in software development (see section 2.1), UI Prototyping (see section 2.2), Low and No code (see section 2.3), Model Based Software Engineering (MBSE) (see section 2.4), Task Based Usability Testing (see section 2.5), and Experimental Product Design and define Design Principles (DP) (see section 2.6).

2.1 Crowdsourcing of Software Products

Iterative feedback from potential customers can help improve the development of software products [15]. To do that, we can use crowdsourcing. Crowdsourcing refers to outsourcing value-creating activities from a company by an open call to a large, undefined group of users to get feedback [16]. The word crowdsourcing is a combination of crowd and outsourcing. Crowdsourcing often involves less specialized and more generalized groups of participants than outsourcing [17]. Some advantages of crowdsourcing include lowered costs, improved speed, quality, flexibility, and scalability [18]. Researchers have used crowdsourcing in many research approaches, including *crowd testing, crowd funding, crowd ideation, crowd logistic, crowd production, crowd promotion,* and *crowd support* over the last few years [19]. In our approach to finding a solution, we focus more on crowd-testing and crowd-ideation.

Crowd Testing: The companies use crowd-testing to evaluate different running software products with the users. A growing trend in software testing is crowd testing, which utilizes the benefits, effectiveness, and efficiency of crowdsourcing and cloud platforms [20]. Crowd testing is considered when the software is more user-centric: i.e., software with a broad user base whose success is evaluated by user input. CrowdStudy [21] is a method that enables developers to assess the usability of their web interfaces using crowd workers from Amazon

Mechanical Turk[1]. CrowdCrit [22] is another tool that uses Amazon Mechanical Turk to support designers in validating created posters in the form of uploaded images. Similarly, *Interactive event-flow graphs* and *GUI-level (Graphical User Interface) guidance* [23] are the two techniques to increase crowd testers' coverage for GUI using crowd-testing.

Crowd Ideation: Design can be infused with creativity by online crowds, but using traditional strategies to harness them, such as large-scale ideation platforms, requires organization and time [24]. Hence, crowd ideation is used to build new and improved versions of existing software product ideas with the consumers. Under manipulations of task complexity, idea representation, and procedural guidance, Shixuan Fu et al. [25] examine how cognitive load is altered during idea generation and convergence with crowds. ERICA [26] is a tool that uses expert knowledge to validate diverse crowd answers. Crowdboard [24] is a tool used to engage crowds in real-time brainstorming, concept mapping, and other design processes at an early stage of the design process. There were, however, no approaches that directly addressed prototype application areas.

2.2 UI Prototyping

User Interface prototyping is an evaluation and testing technique according to User-Centred Design (UCD) methodology since the 1990s [27]. The evaluation of prototypes by users is a fundamental part of all iterative approaches for IT project management, especially agile methodologies [28]. 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 [29]. 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 complex and expensive, prototyping is a crucial technique [30]. Simultaneously, software prototypes might exclude many requirements, making the software more accessible, smaller, and less expensive to construct and change [30]. 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.

[1]Amazon Mechanical Turk: https://www.mturk.com

Jim Rudd et al. [31] have compared high and low-fidelity prototyping, explaining the advantages and disadvantages. *Low-fidelity* prototypes are usually limited function, 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. 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. Contrary to low-fidelity prototypes, *High-fidelity* prototypes have full functionality and focus on flow, and the user models of the system [32]. 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 [31].

2.3 Low Code / No Code

2.4 Model-based Software Engineering

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. The development of domain-specific languages (DSLs) is becoming essential in language engineering due to the growth in model-driven engineering (MDE) [33]. MDSE has become an integral part of developing User interfaces, and they have been named Model-driven User Interfaces (MUIs). Based on that, adaptive model-driven user interface development systems are developed [34]. In this research, the authors defined twenty properties challenges for the Model-driven User interface and compared some tools that implement these properties.

Companies use different modeling languages to codify the UIs. Cameleon [35] 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. A standardized modeling language for software product content, abstract UI models, user interactions, and control behavior is Interaction Flow Modeling Language (IFML) [36]. As a result, IFML relies on the platform-independent display of the UI that can be utilized on several platforms and devices.

However, these modeling languages do not emphasize offering visual notations to aid non-developers in creating such interfaces. A recent method [37] illustrates how to use low-code approaches to close the gap between designers and developers.

2.5 Task-based Usability Testing

2.6 Experimental Product Design

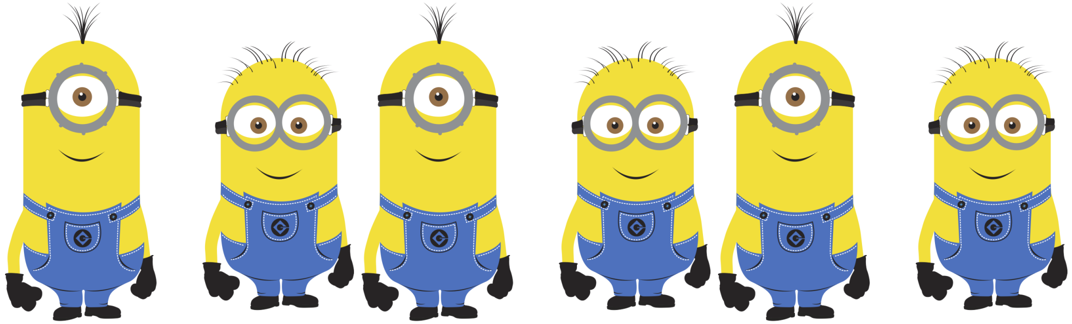
Description

The first topic is dull

The second topic is duller

- **The first subtopic** is silly
- **The second subtopic** is stupid

The third topic is the dullest



(a) Tom and Jerry (b) Wall-E (c) Minions

Fig. 2.1 Best Animations

2.7 Landscape

I can cite Wall-E (see Fig. 2.1b) and Minions in despicable me (Fig. 2.1c) or I can cite the whole figure as Fig. 2.1

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

[16] Jan Marco Leimeister. Crowdsourcing. *Controlling & Management*, 56(6):388–392, 2012.

[17] Enrique Estellés-Arolas and Fernando González-Ladrón-de Guevara. Towards an integrated crowdsourcing definition. *Journal of Information science*, 38(2):189–200, 2012.

[18] John Prpić, Araz Taeihagh, and James Melton. The fundamentals of policy crowdsourcing. *Policy & Internet*, 7(3):340–361, 2015.

[19] D Durward, I Blohm, and JM Leimeister. Crowd work. business & information systems engineering, 58 (4), 281–286, 2016.

[20] Thomas D. LaToza, W. Ben Towne, André van der Hoek, and James D. Herbsleb. Crowd development. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 85–88, 2013.

[21] Michael Nebeling, Maximilian Speicher, and Moira C Norrie. Crowdstudy: General toolkit for crowdsourced evaluation of web interfaces. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 255–264, 2013.

[22] Kurt Luther, Jari-Lee Tolentino, Wei Wu, Amy Pavel, Brian P Bailey, Maneesh Agrawala, Björn Hartmann, and Steven P Dow. Structuring, aggregating, and evaluating crowdsourced design critique. In *Proceedings of the 18th ACM conference on computer supported cooperative work & social computing*, pages 473–485, 2015.

[23] Yan Chen, Maulishree Pandey, Jean Y Song, Walter S Lasecki, and Steve Oney. Improving crowd-supported gui testing with structural guidance. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[24] Salvatore Andolina, Hendrik Schneider, Joel Chan, Khalil Klouche, Giulio Jacucci, and Steven Dow. Crowdboard: augmenting in-person idea generation with real-time crowds. In *Proceedings of the 2017 ACM SIGCHI Conference on Creativity and Cognition*, pages 106–118, 2017.

[25] Shixuan Fu, Xusen Cheng, Triparna de Vreede, Gert-Jan de Vreede, Isabella Seeber, Ronald Maier, and Barbara Weber. Exploring idea convergence and conceptual combination in open innovative crowdsourcing from a cognitive load perspective. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019.

[26] Nguyen Quoc Viet Hung, Duong Chi Thang, Matthias Weidlich, and Karl Aberer. Erica: Expert guidance in validating crowd answers. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1037–1038, 2015.

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

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

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

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

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

[32] 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.

[33] Jesús Sánchez Cuadrado and Jesus Garcia Molina. A model-based approach to families of embedded domain-specific languages. *IEEE Transactions on Software Engineering*, 35(6):825–840, 2009.

[34] 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.

[35] 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.

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

[37] 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.

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.