Mariana de Oliveira Alves Bexiga

Bachelor in Computer Science

Closing the Gap Between Designers and Developers in a Low-Code Ecosystem

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Engineering

Adviser: João Costa Seco, Associate Professor, NOVA University of Lisbon

Co-adviser: Stoyan Garbatov, PhD, Research and Development Engineer, Outsystems

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

January, 2021

Closing the Gap Between Designers and Developers in a Low-Code Ecosystem

This document was created using the (pdf)LaTeX processor, based on the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

I would first like to thank my thesis advisor Assistant Professor João Costa Seco for giving me the chance to do my thesis in OutSystems and for the valuable advice during this process. I would also like to extend my deepest gratitude to my OutSystems advisor Stoyan Garbatov for the guidance and motivation throughout the whole process. This work would not be possible without them.

My sincere thanks to the thesis stakeholders and everyone from OutSystems for their dedicated involvement and insightful suggestions. Special thanks to my university colleagues who supported me and shared this journey with me.

Last but most certainly not least, I would like to thank my parents and my sister for their encouragement all through my studies.

Abstract

Digital systems play an important role in the strategy of companies nowadays as they are crucial to achieve their business goals as well as gain a competitive advantage. This is particularly true for systems designed for the end-users market. Not only has the number of such systems been growing steadily but the requirements and expectations of users regarding usability and performance have also increased.

Developing software systems is a complex process that comprises several stages and involves multiple professionals responsible for different tasks. Two stages of the process are the design and implementation of User Interfaces. UX and UI designers produce artifacts such as mockups and prototypes using design tools describing what should be the systems behavior, interactivity and look and feel. Next, Front-end and Back-end developers implement the system according to the specifications defined by designers.

Designers and developers use different methodologies, languages, and tools. This introduces a communication gap between both groups, and hence collaboration between them is not always smooth. This ends up causing less efficient processes, rework and loss of information.

Developers sometimes overlook the importance of User Experience and Front-end Development. The corresponding project stages suffer when collaboration between groups is not optimal. Problems are particularly striking during the transition from design artifacts to front-end code. The procedures for doing this are often manual, very error-prone, and time-consuming for developers that end-up completely re-doing the designers' work in the target web technology.

The aim of this dissertation is to improve the efficiency of collaboration between designers and front-end developers in the OutSystems ecosystem. This was carried out by developing a tool that transforms UX/UI design artefacts into low-code web-technology using model transformation and meta-modelling techniques. The approach has been evaluated in practice by a team of professional designers and front-end developers. Results show savings between 20 and 75% according to the project complexity in the effort invested by development teams in the above mentioned process.

Keywords: Designer-Developer Collaboration, Shared Artifacts, Front-End Development, OutSystems, Model-Driven Techniques

Resumo

Os sistemas digitais têm um papel muito importante hoje em dia na estratégia das empresas, uma vez que as ajudam a atingir os seus objetivos de negócios assim como a ganhar vantagem competitiva face às demais. Isto é particularmente verdade para os sistemas destinados ao mercado dos utilizadores finais. Não só aumentou significativamente o número de tais sistemas como aumentaram também os requisitos e expectativas dos seus utilizadores.

O desenvolvimento de sistemas de software é um processo complexo que envolve um número alargado de profissionais com diferentes perfis. Duas das fases são o design de interfaces e a sua implementação. Os UI e UX designers produzem um conjunto de ficheiros utilizando ferramentas de design descrevendo o comportamento, interação e aparência do sistema. Os Front-end e Back-end developers implementam as funcionalidades do sistema com base nas especificações dos designers. Designers e developers trabalham utilizando diferentes metodologias, linguagens e ferramentas. Isto introduz uma lacuna entre os grupos e dificulta a sua colaboração, originando processos menos eficientes, trabalho refeito e perda de informação.

Os engenheiros de software por vezes negligenciam disciplinas tais como a Experiência do Utilizador e o Front-end. Estas fases dos projetos são prejudicadas quando a colaboração entre os grupos não é ótima. Os problemas são particularmente evidentes durante a transição entre design e Front-end. Os procedimentos são manuais, altamente propensos a erros e demorados para os developers, que acabam por refazer o trabalho dos designers na tecnologia web desejada.

O objetivo desta dissertação é melhorar a eficiência da colaboração entre designers e front-end developers no ecossistema OutSystems. Isto foi levado a cabo desenvolvendo uma ferramenta capaz de transformar artefactos de UX/UI em low-code utilizando transformações de modelos e técnicas de *meta-modeling*. A abordagem foi avaliada em prática por uma equipa profissional de designers e front-end developers. Os resultados obtidos mostram poupanças entre os 20 e os 75% de acordo com a complexidade do projeto no esforço investido pelas equipas de desenvolvimento no processo mencionado acima.

Palavras-chave: Colaboração Designer-Programador, Partilha de Artefactos, Desenvolvimento Front-End, OutSystems, Técnicas Model-Driven

Contents

List of Figures

List of Tables

Acronyms

B2B Business-to-Business

B2C Business-to-Customer

B2E Business-to-Enterprise

LSG Live Style Guide

UCD User Centered Design

UI User Interface

UX User Experience

Chapter 1

Introduction

This chapter starts by providing context on the domain area of this work, it next presents a discussion on why it is worth investing effort researching the particular topic of collaboration between designers and developers. The main objectives of the work are identified along with the contributions for the state of the art. Finally, it presents the document structure.

1.1 Context

Rapid market changes and the advancement of technology require companies to adapt quickly to remain relevant. The success of a company is strongly related to how easily and promptly it reacts to the changes in the market it operates in. It is increasingly frequent for companies to use digital systems as a means of achieving their business goals and keeping up to date.

The number of people using digital systems has grown significantly in the last decades and so have their high expectations in terms of usability, reliability and performance. This has led the development of highly polished user interfaces to become an ever growing priority for companies. Good user experience is a crucial factor for setting a product apart from the competition [54]. Understanding users and paying attention to their interaction with systems is even more determinant in the end-user market. The expectation is that more organizations move in this direction.

Understanding users and paying attention to their interaction with systems is even more relevant for B2C [1] products [18]. Systems with front-end and user experience

[1] Business to Consumer (B2C) systems are designed for being used by consumers. Business to Business (B2B) and Business to Enterprise (B2E) systems aimed at being used by enterprise users and other businesses respectively.

concerns represent ways of creating business value and consequently the number of companies investing in these areas is increasing. Forrester Research [2] states that for each dollar invested in front-end development, companies get a return of 100 dollars. Even Google announced a new ranking algorithm that considers User Experience as one of the factors [94].

Software systems are complex and their design and development require the collaboration of a wide range of professionals. Collaborative work processes help designing systems that meet user needs in an effective and efficient way.

Software development requires the collaboration of a wide range of professionals. Initially, the system is idealized and its requirements defined. Later, the system is designed and implemented. Finally, the implementation is validated and the system delivered. Collaborative work processes help designing systems that meet user needs in an effective and efficient way. However, groups involved in software development use distinct methodologies, tools and constraints, whose differences hinder the project's productivity and efficiency [4].

UI and UX Designers are responsible for the system visual experience as well as how the system should behave (functionally). Developers are in charge of the creation of the system itself by developing the technical solution. Both groups have different goals and responsibilities and work according to different methodologies. The transition between the design of user interfaces and user experience phases and its implementation by the front-end developers is one of those cases.

The implementation of design specifications is based on high-fidelity representations created by specialized designers using dedicated tools. The process is performed manually and even for expert developers the conversion from design artifacts to web technology applications is not a trivial task. The diversity of components in UI designs makes the conversion a complex process. Designers and developers iterate to produce different UI versions for multiple contexts using their own tools until they are accepted by clients. This is a cumbersome, time-consuming and error-prone process and the final result is frequently different from what was designed.

This work is inserted in the OutSystems ecosystem, a low-code development platform where the processing of the UI design artifacts is done by front-end developers, who manually convert UI elements and corresponding customization to their OutSystems representation. This is a manual process and sometimes customization is not translated properly, making the end result different from the initial design. Furthermore, several stages of the translation process are common to all projects and perceived as monotonous and not challenging by front-end specialists. Nonetheless, these challenges are not specific to OutSystems and other technology ecosystems experience similar challenges.

Our approach is based on an abstract intermediate representation and meta-model to process and create models. It relies on model transformations to successfully convert UI design models made with a visual domain language [89] to an abstract intermediate representation. Later, the intermediate artifact is transformed to the corresponding

OutSystems [68] customized applications. Our approach is materialized through a prototype tool that receives a template application containing all base elements as input and generates customized OutSystems applications.

The use of the tool represents a significant reduction in the duration of the conversion and customization process carried out by professional teams, between 20 and 75% according to the project complexity. When applied to the OutSystems Customer Success department, the usage of the tool can save up to 122 days per year.

1.2 Motivation

To develop products with quality that meet the expectations of users, collaboration processes must be as efficient as possible. The development process is negatively affected when the collaboration between designers and developers is not optimal. Challenges include, but are not limited to, usage of tools that were not made to work with one another or people working and communicating in fundamentally different ways. All of this ends up harming companies making it difficult to achieve intended business goals.

Having a mechanism able to translate design specifications and applications look-and-feel from design tools to front-end code significantly improves the workflow productivity and reduces errors in the transition process. The design to code translation problem is not only relevant in the low-code platforms domain but also to other web technologies, where there are also limited opportunities to automatically convert artifacts.

Accounting for all of the above, it should be fairly evident that the collaboration between the two groups is a relevant topic to the industry nowadays. This is a relatively new and unexplored topic that should be properly researched for exploring new approaches that improve the effectiveness and efficiency of this collaborative process.

1.3 Objectives

The goal of this dissertation is to improve the efficiency of the collaboration process between designers and front-end developers working in the OutSystems ecosystem. This was achieved by developing a tool that converts high-fidelity design artifacts developed with design tools into OutSystems domain-language front-end reusable components. The approach is based on model transformations. While the effectiveness of our tool was evaluated in the context of the OutSystems platform, its architecture allows it to be applied to both other design technologies (e.g. Figma, Invision) as well as other web technologies (e.g. React, Angular).

1.4 Contributions

An initial contribution is an extended study about the current collaboration process between designers and developers at OutSystems and partner companies that work with

the OutSystems technology. Alongside that analysis, this dissertation will provide an overview of the topic's current state of the art.

The main contribution of this work is the introduction of a novel approach that improves the conversion of design artifacts to web technology representations. The prototype developed in this dissertation streamlines the transition between the UI design and front-end development.

Even though we are covering the Sketch to OutSystems conversion, due to the generic and technology-independent nature of the solution architecture, this approach can be applied to other design and web technologies.

Furthermore, due to the technology-independent nature of the solution architecture, this approach can be easily applicable to other design and web technologies. The tool is expected to bring value in the design-to-code field, making the work contribution not limited to the OutSystems ecosystem.

During the development process, our tool developed in this dissertation was used by two OutSystems professional teams from different departments who daily convert UI artifacts into OutSystems applications. Currently, the engine is part of both team's workflow. It is expected that it will be applied not only inside OutSystems but relevant for OutSystems customers and partners.

1.5 Document Structure

The remainder of this document is organized as follows:

- Chapter 2 - The OutSystems Platform gives context about the OutSystems Platform and technology.
- Chapter 3 - State of the Art presents the current state of the art in the problem space.
- Chapter 4 - Problem Identification identifies and justifies the choice of the problem to be solved.
- Chapter 5 - Related Work presents in detail the state of the art in the solution space.
- Chapter 6 - Technical Approach presents in detail the study of the selected problem based on the state of the art as well as an overview of the solution implemented.
- Chapter 7 - Implementation outlines our approach to convert design artifacts made with design tools to its web technology representation using an intermediate representation. The section also highlights the challenges of modeling and transforming both models as well as the evaluation of the tool.
- Chapter 8 - Evaluation focus on the evaluation methods used to measure the viability of the proposed solution as well as the results obtained.

- Chapter 9 - Conclusions covers the concluding remarks as well as relevant future work.

Chapter 2

The OutSystems Platform

This dissertation is being developed in the context of a collaboration between *NOVA LINCS*, *FCT NOVA* and the *OutSystems' Research and Development* department. In this chapter, we briefly describe the OutSystems platform's main components as well as the visual programming language used to define user interfaces.

2.1 OutSystems Platform

OutSystems is a low-code application development platform for web and mobile applications. OutSystems covers every stage of an application development process such as the design, development and lifecycle management. It aims to drastically accelerate application development by offering abstraction concepts to its end-users. Furthermore, it seeks to lower the skillset necessary for one to play an active role in the development of enterprise-grade applications. In particular, it allows for people with non-IT backgrounds to be fully capable application developers.

Platform Architecture

The OutSystems platform offers tools that cover all steps of an application lifecycle. The ones that deal with the actual construction part are Service Studio and Integration Studio. Service Studio deals with the development of OutSystems technology applications, while Integration Studio is focused on integrating OutSystems applications with external third-party technology systems [60, 68].

An overview of the OutSystems ecosystem architecture including applications and development process supporting tools can be seen in Figure 2.1. The main components that can be seen in Figure 2.1 are:

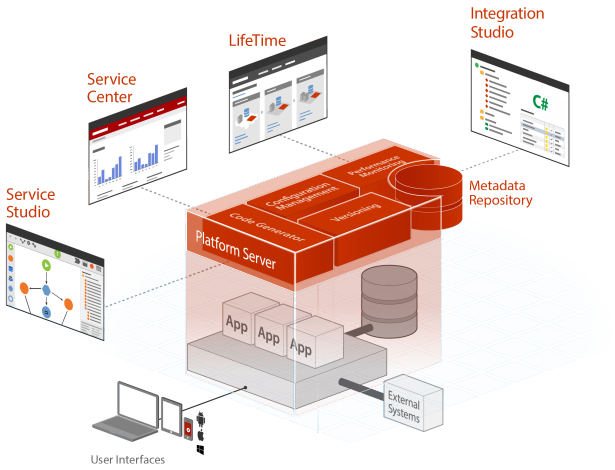

Figure 2.1: OutSystems Platform Architecture in [68]

- **Service Studio** is the OutSystems IDE for visually developing OutSystems web and mobile applications. It allows for the definition of different layers (UI, logic, data) of the application using different visual languages: screen flows, actions, data models, etc.

- **Integration Studio** for creating and managing components for integrating with third-party libraries, and external databases.. These extensibility points are published (uploaded, compiled and deployed) to the server as Extension modules and can be consumed by Service Studio modules for use in applications.

- **Platform Server** for generating, building, packaging and deploying applications to an application server. It has a set of specialized services, Code Generator, Deployment Services and Application Services. When clicking on the "1-Click Publish Button" shown in Figure 2.2, the code generator takes the application developed in Service Studio and generates code for all the layers of the application. The Deployment Services deploy the generated applications to the application server of every Front-End server.

2.2 OutSystems Visual Language

The core development of the business logic of an application is defined in actions developed in a visual flowchart line imperative language.The OutSystems language is a Strongly Typed language composed by a group of visual domain-specific languages that model all the application layers: processes, interfaces, logic and data. The manipulation of the elements that represent the OutSystems Language is done on Service Studio.

Figure 2.2 illustrates Service Studio interface. On the Application Layer Panes on the right side of the screen, users can select the desired application layer and manipulate

it. The first pane covers the **Processes** and helps design and manage the application business processes. In the **Interfaces** pane, a hierarchical component model, widgets and web blocks for applications are created and modified. Development is done using widgets or by creating reusable interface blocks. In this pane, it is possible to overview the interface component tree.

The third pane represents **Logic** and it is where the logic needed by apps is defined. The logic part of applications is implemented through Actions. OutSystems has built-in actions such as System Actions and Role Actions but it is possible to create custom actions.

The **Data** pane helps to define the data structures used by the application. In OutSystems, entities represent database tables or views and help implement the database model.

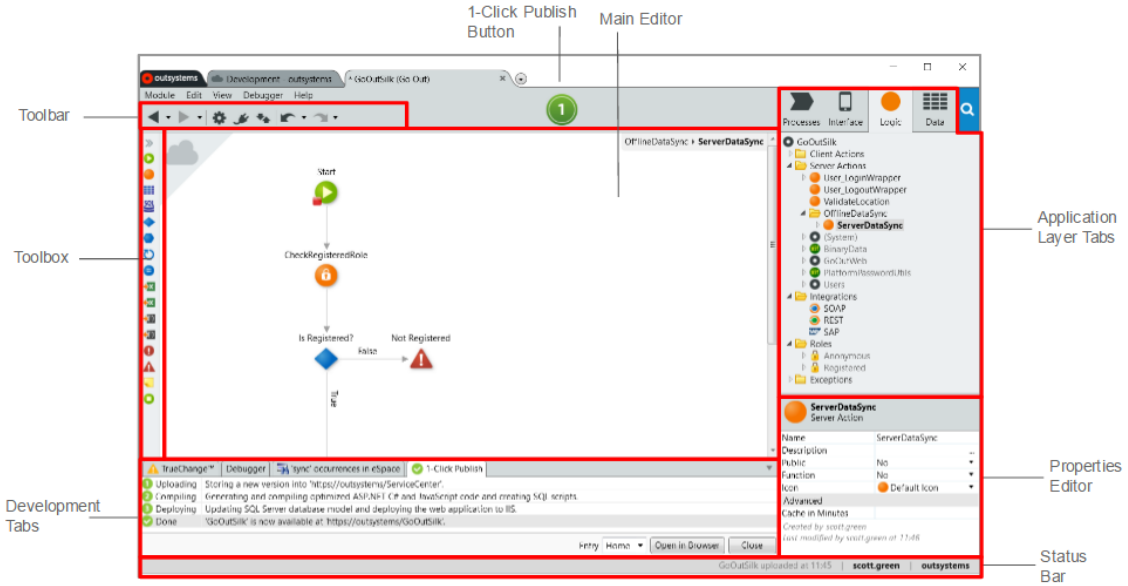

Figure 2.2: Service Studio Interface in [68]

2.3 User Interface Development

In OutSystems, applications' User Interfaces are named screens and web blocks and are organized hierarchically. Screens can be managed in the Interface Pane on the right side in Service Studio. When the tab is selected, the **Widget Tree** on the right side of the screen shows the components are organized as seen in Figure 2.3.

The OutSystems model is hierarchical and can be defined using the **Widgets** and tools available in the **Toolbox** on the left side of the screen. Widgets map directly to screen visual elements and are its basic building blocks.The selection of widgets represents the most common User Interface elements. The components available on the Toolbox compose the OutSystems Visual Language. A group of constructs of the Interfaces DSL

concrete syntax is illustrated in Figure 2.4. The user can drag-and-drop the items on the Toolbox to the Main Editor in the center of the screen to build and customize applications. Every widget is described by a set of properties that model the widget according to the application. The properties include source data, behavior and appearance among others.

Besides Widgets, the Toolbox includes components that have control for defining the application. Some examples are **Conditionals** and **Expressions**. **Conditionals** are represented by the If Widget and help control the content displayed based on a condition. **Expressions** are used for displaying text or a combination of values and operators that are calculated at runtime.

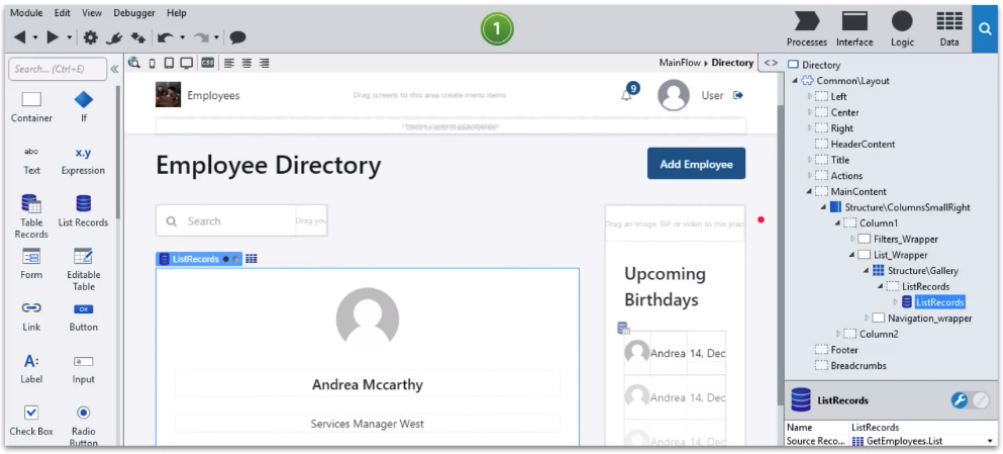

Figure 2.3: Service Studio User Interface Development

Some of the most relevant elements for building applications available in the **Toolbox** are **Tables**, and **Blocks**. Tables display data in and allow for other widgets between columns. **Blocks** allows users to reuse and refactor sets of widgets that occur multiple times in application screens and other blocks. If the block is changed, the changes are propagated to all of its instances in the application. Some examples of components that can be Blocks are headers and footers.

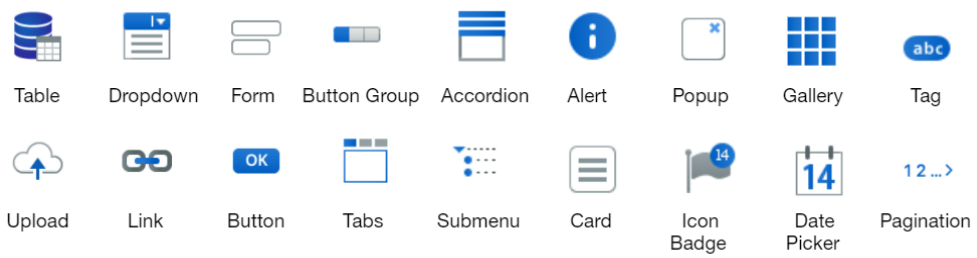

Figure 2.4: Concrete Syntax of the OutSystems UI DSL

2.4 OutSystems Applications

There are multiple types of applications that can be developed in OutSystems. Three of the more relevant are:

- **Reactive Web Applications** run on the browser and have responsive interfaces that adapt to different screens.
- **Mobile Applications** compile to a native Android or iOS application. Mobile applications have specific mobile UI patterns and UX.
- **Traditional Web Applications** are focused on server-side development.

OutSystems applications are composed of multiple related modules and must have at least one module. Applications' data model, business logic, and web pages are defined inside modules. Independent functionalities of the same application are encapsulated in different modules. This software design technique is called **Modular Programming** [72].

Modules can be classified as **producers** or **consumers**. Producer modules expose their features, while consumer modules reuse elements from other modules. Interface features such as web blocks, web screens, images, or themes are examples of features that can be exposed by modules. There are also data, logic, and processes features that can also be shared between modules when their features are set public.

The reuse of features is done using **dependencies**. Dependencies can be classified as weak or strong, according to the type of components exposed by the producer modules. In strong dependencies, the consumer module executes the logic as if it is defined inside the module. In our context, strong dependencies can be established when elements such as Themes, Images, Blocks, and Resources are shared. In week dependencies, elements with associated logic run in the context of the producer's request. The consumer module only knows the signature of the element unlikely in strong dependencies, where the consumer knows the signature and the implementation of the feature that is reusing. Modules can have dependencies from modules not internal to the application [85].

Chapter 3

State of the Art

This chapter presents the related work on the problem space. Since Collaboration between Designers and Developers is a vast topic we had to explore it to identify the existing inefficiencies in the process and where improvements would have a higher impact. The State of the Art on the solution space is explained in Related Work.

In this chapter, collaboration is discussed in the context of software development process. We start by explaining the roles of designers and developers. Next, we present the work methodologies with special focus on the differences of both groups and the importance of collaboration and communication in the success of a project. Later, we analyze existing design tools in the market and their role supporting collaboration. Lastly, the software development process in the OutSystems ecosystem is explained.

3.1 Collaboration in Software Development

Communication involves exchange of knowledge between two parties through a specific medium [7]. Good communication happens when the meaning of the original message is kept and the information is correctly converted and understood by the final user [34]. As explained in [7], sharing information is a frequent practice in projects done through multiple tools and methods even if team members are not in the same physical space.

Collaboration is a dynamic process involving cooperation with others to accomplish a common goal that would hardly be achieved individually as explained in [14]. Research [25] advocates that collaboration should not be limited to the professionals responsible for the development but include clients and users from an early stage of the project.

3.1.1 Designers and Developers Work Methodologies

Two of the groups involved in software development are designers and developers. The groups play important roles in the conception of systems. In this section, designers include both UI and UX designers and developers include front-end, back-end and full stack developers.

Designers communicate visually and use design tools to produce a set of files that materialize their vision taking into account final-user needs and system requirements. Designers start by developing low-fidelity artifacts such as sketches and wireframes [1], and progressively create more detailed representations such as mockups [2] and finally prototypes [3]. Designers have their own practices. One example is **User-Centered Design**, a design process focused on understanding and prioritizing end-user needs. It is an iterative process where end-users play an important role in constantly evaluating the product that is being developed [54]. Other relevant design processes abstractions are Goal-Directed Design, Usage Centered Design, and Contextual Design even though in practice these processes are not strictly followed as analyzed in [20].

Developers are responsible for translating representations of the system made by designers into the final application. They work with abstractions and use IDEs and text editors as seen in [31]. Developers work mainly based on Agile – a popular and widely used methodology regarding software development processes. This process is value-driven, which means it goes through multiple iterations and makes sure value is added in each iteration. Autonomy, flexibility and quick working software delivery are other characteristics of this methodology as mentioned in research work [54, 12, 18, 31].

Design and development are based on different disciplines and even though both methodologies are concerned with delivering high-quality software and focus on clients and users, development is seen from different perspectives as mentioned in [19, 20, 41]. Agile methods intend to quickly deliver small software increments within a short amount of time. Design methods focus on usability and in producing usable products for the end-users of the system [18–20, 41]. Iterations and testing are done differently. Design iterations are done on the user interface and last hours or days, while agile iterations on codecan last weeks. Agile testing is done in a mostly automated way. UI and UX methods test their work with potential users as stated in [35].

Designers and developers need to continuously communicate and make decisions together to satisfy final users and deliver high-quality software. However, these groups have completely different mindsets, distinct work objectives and methodologies and inefficiencies arise motivated by such differences [4].

[1] Wireframes are low-fidelity representations of the structure of a system.
[2] Mockups are static mid or high-fidelity representations of the system.
[3] Prototypes are high-fidelity representations of the system that represent user interaction with the system.

3.1.2 Implications of the Collaboration Inefficiencies

Collaboration inefficiencies lead to rework, waste of effort and loss of information, potentially harming the final product quality as concluded in [4, 31].

Collaboration inefficiencies are especially harmful for disciplines that stand in the transition between groups such as Front-end and User Experience. Design artifacts such as high-fidelity mockups and prototypes are done by designers with their own tools. Developers are responsible for manually converting these artifacts to code. As mentioned in [33], this process is cumbersome and many errors are introduced during the process. This is even more relevant taking into account that systems with *Human-Computer Interaction (HCI)* concerns represent big sources of income and business value for companies as explained in [54, 18]. User Experience has gained relevance for businesses over time, however work processes are not designed to support it.

The work methodologies do not have interaction and collaboration between groups as central concerns. As a consequence, during the process, information is likely to be changed and lost between groups as mentioned in [31].

3.1.3 Approaches to Improve Collaboration

There is little research on how to improve collaboration and communication between groups, however work of [19, 20] suggest that **Agile and User-Centered Design Integration** can have several benefits. The two practices are particularly distinct and their integration is not straightforward. Tight deadlines, not enough user involvement and not incorporating their feedback into the development process make difficult the integration of Agile with UX design tasks as explained in [20].

Research in [19] states that successful integration is based on mutual awareness, expectations about acceptable behavior, negotiating process and engaging with each other. UX and UI Designers should work ahead of developers to ensure that developers have the assets needed for their work [20].

A hybrid approach was suggested in [30]. The work points that iterating design and development has a positive impact on the final product quality and challenges the current workflow claiming that both designers and developers should be involved in the project from the beginning. Both groups should discuss ideas and review code and base their work on the feedback gathered during meetings.

Forrester Research [15] reinforces the relevance of Product Managers as mediators for connecting and establishing channels between both groups can help collaboration not only between designers and developers but also between clients and stakeholders.

Mental Models play an important part when talking about team collaboration and cooperation. A mental model describes the thinking process of an individual and their perception of the world. It is influenced by factors like experience and knowledge and plays a role in how tasks are performed. When teams that have similar mental models,

they tend to have better performance as can be seen in the studies carried out in [3]. Having professionals with multidisciplinary knowledge and with compatible mental models can improve the work process and final result. New professionals should also be trained in that direction.

Another approach for improving collaboration is the adoption of **Design Thinking** by all the groups involved in software development. Design thinking is a creative process to solve design problems that focus on user needs. Combining knowledge about users with creative problem-solving methods creates great innovative systems as explained in [28, 43]. Another study [44] defends that design thinking should not only be adopted by designers but for everyone involved in the system development process. Gartner [24] reports that "by 2023, 60% of large enterprises will adopt design thinking as part of agile product delivery to drive better business outcomes.".

Collaboration revolves around collaboration events and artifacts as claimed in previous research work [11, 31]. Wireframes, user stories or UML diagrams help to guide and organize discussion between groups. Handover sessions are based on discussion and exchange of ideas and support bridging the work of designers and developers as explained in [25]. The **Promotion of Good Documentation** and the **Scheduling of Events** between groups improves the development process efficiency.

3.2 Design and Collaboration Tools

Design tools have different purposes and support designers carrying out their work. They can be classified into image editing tools, UI and UX oriented tools and design collaboration tools.

Image editing tools such as Adobe Photoshop [46], Adobe Illustrator [45] and Affinity Designer [48] provide designers mechanisms to create logotypes, icons or illustrations. Even though these tools are very powerful, their focus is not helping either the user interface design process or collaboration between groups.

There are tools specialized in designing User Interfaces and User Experience. Some tools are focused on wireframing and quickly design low-fidelity representations such as Balsamiq [53], Lucidchart [74], Pidoco [78] and Draw.io [55]. Sketch [89], Adobe XD [47], FluidUI [57], Moqups [75], Webflow [84] ProtoPie [81] and Indigo Design [62] are intended for designing high-fidelity mockups and prototypes. These tools provide libraries with pre-made UI assets, promoting consistency and streamlining the design process.

Beyond allowing designers to develop their system representations, some tools assist collaboration between designers and developers through comments and file-sharing options. Some examples are Figma [56] and InVision [64]. Other relevant tools with collaboration concerns are Zeplin [100], Mockplus [71], Avocode [51], Framer [58], UXPin [95], ProtoIO [80] and Axure [52]. The tools allow import designs from other design platforms and generate the corresponding code.

3.2.1 Tools Classification

We analyzed 23 design tools to understand which ones were aligned best with the topic of this work. To achieve this, the tools were classified from 0 to 5, according to four criteria:

- **Product** defines how powerful the product is. Based on image editing tools, interaction with prototypes, compatibility with other tools and operating systems
- **Community** is based on the product documentation and resources
- **Collaboration** is related on the tool collaboration features
- **Trends** classify how relevant the tools are or are expected to be based on analyst reports and interviews.

Table 3.1: Tools classification

weight	*1*	*3*	*5*	*5*	
Tools	**Product**	**Community**	**Collaboration**	**Trends**	**Weighted Score**
Figma	3	4	5	5	65
Sketch	2	5	4	5	62
Adobe XD	3	4	4	3	50
Framer	3	5	4	1	43
InVision	3	3	5	4	42
Lucidchart	2	3	4	2	37
Photoshop	3	5	0	3	33
FluidUI	3	3	4	1	31
Moqups	3	4	5	1	30
UXPin	3	4	5	1	30
Proto.io	2	3	4	2	30
Illustrator	1	3	0	3	30
Webflow	2	3	2	2	28
Avocode	3	3	5	1	27
Axure	3	3	4	1	26
Zeplin	2	3	5	1	26
Draw.io	2	3	5	1	26
Affinity Designer	3	2	1	2	25
Indigo.Design	3	3	2	1	24
Mockplus	2	4	0	1	24
ProtoPie	3	3	1	1	23
Pidoco	2	2	4	1	22
Balsamiq	1	2	4	1	21

According to our classification as well as based on interviews, Figma, Sketch and Adobe XD are the most relevant tools helping the work transition between groups.

3.3 Collaboration in the OutSystems Environment

To better understand the work dynamics and collaboration between designers and developers, we interviewed 25 OutSystems and partner company employees. The interviewee group included not only designers and developers from OutSystems but also other roles involved in delivering a project successfully.

The interviewee group included professionals from two partner companies. The main objective was to understand the main steps of a project as well as the tools and approaches used. This research sought to understand what differences exist between the process done at OutSystems and other companies that also work with OutSystems technology.

The partner companies [59, 69] interviewed were specialized in design and front-end development and most of their business is made using OutSystems technology.

The selection was made to have as wide a range of interviewees as possible to ensure a comprehensive view of the predominant practices and work dynamics without being strongly biased by any particular perspective.

3.3.1 OutSystems Software Development Process

Among other things, the OutSystems Customer Success department is responsible for designing and developing projects for OutSystems customers. It is frequent for customers who are new to OutSystems technology or do not have dedicated development teams to employ the services provided by Customer Success. Since B2C applications are aimed at being used by end-users and are particularly demanding, companies hire OutSystems professional services for delivering such high business impact initiatives. The current section covers the software development process in this department.

The usual flow in a project carried out by the OutSystems Customer Success department is divided into 4 main phases as illustrated on Figure 3.1. The number of professionals involved in a project is defined before a new project starts, according to its complexity.

The first stage of a new project is called the **Vision** or **Initiation**. This is the phase where the entire project and its experience are outlined based on the branding of the client and needs. Initiation starts with personas identification as well as their needs and related business goals. The main goal of this phase is to make decisions that represent the greatest customer value. The characterization of the project takes 10 days and it is made by a UX Designer, an Architect, and an Engagement Manager. The UX Designer draws the low-fidelity mockups of the system representing the most important aspects of the journey of users and their interaction. The non-functional requirements are guaranteed by the Architect. The Engagement Manager defines the scope of the initiative and ensures alignment on the implementation.

The low-fidelity mockups are then delivered to UI Designers who convert low fidelity system designs into high fidelity ones during the **UI Design** phase. The high fidelity

mockups are more detailed representations of the system and UI designers take 3 days to design them. The UI designs are later approved by the client.

It is frequent for projects to have a handover session - sessions where designs are discussed between designers and developers. The session helps clarify any doubts and contextualize developers about the work previously done.

The development of the project is divided into two stages, **Front-End Development** and **Back-end Development**. Front-end developers are responsible for developing the Live Style Guides OutSystems application and sample pages based on the high fidelity mockups previously done by designers. The Live Style Guide Creation Procedures takes one week. Design artifacts are shared with Front-end developers through a Figma or Invision link. The usage of such tools helps the transformation process by providing means to gather style information such as font family, colors from designs.

The next step of development takes about 3 months to be delivered and consists of the incremental and iterative development of the technical solution following the Agile methodology. This task is performed by the back-end development team. During this stage, developers take the components developed by the front-end team and code the business logic underneath. Users and stakeholders are highly involved in the process with the purpose of validating the work. This practice makes it easier to cope with business changes.

During the workflow, the project is submitted 3 times to an audit as a way of monitoring and a way of ensuring that the product is being developed according to the best practices of UX and Architecture assessment. This process is named *Great Apps Program (GAP)* and is performed by experts not involved in the project who evaluate and score the product.

It is common for the Customer Success department to deliver only the first version of the project to the client. From there, the development team of the client company is responsible for maintenance of the system.

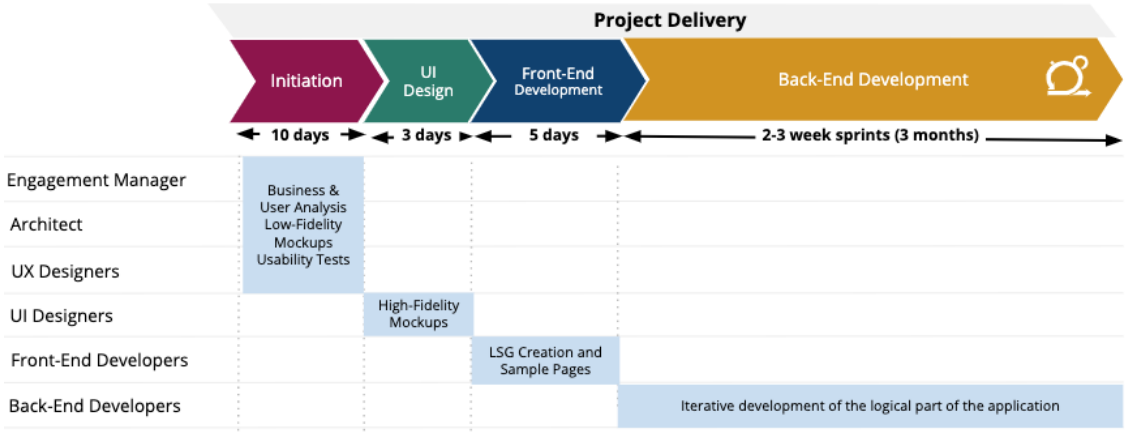

Figure 3.1: Customer Success Development Process

3.3.2 External Software Development Practices

To complement the information gathered from the interviews with OutSystems employees, we interviewed professionals from MediaWeb [69] and Hi-Interactive [59], two companies specialized in design and front-end development that work with the OutSystems technology.

The workflow of the two companies has two major variants. Companies can be part of a project from the beginning or have their services be required when the project has already begun. In the first case, both design and front-end teams contribute to develop the UX. This work is done in an iterative way using the low-fidelity design tool, Balsamiq. For designing high fidelity mockups, both companies use Sketch, Figma and InVision.

Similar to the OutSystems Software Development Process, when the UI design of the project is developed by another company, it is common for designers not being familiarized with the OutSystems framework and for development challenges to arise. Some of the components designed need to be built from scratch or redesigned, implying a project cost increase of 30 to 40 percent, according to Hi-Interactive. When designers are familiarized with OutSystems patterns, creating the Live Style Guide becomes easier and faster because designers know exactly which components already exist in OutSystems.

The effort needed for the Front-end Development phase is based on the customization needed. The standard duration is 10 days, 4 days are dedicated to the Live Style Guides creation and following 6 days for building sample pages. The Live Style Guides creation process is only done by senior professionals since it is not a trivial task. During the front-end development, 60 to 70 percent of the effort is invested in CSS and only around 30 percent in JavaScript.

Both companies agree that good documentation plays an important role in projects. It creates awareness among designers and developers and guarantees that the project is developed according to the established good practices. The two said that having Live Style Guides is highly recommended for projects with several applications. As the portfolio grows up, it is often necessary to have a more structured approach that supports a scalable and sustainable development model. Companies create **Design Systems** [4] for promoting component reusability, cohesion within the same scope and allowing a sustainable and scalable development process for supporting multiple lines of businesses.

To face collaboration inefficiencies, identifying dissimilarities earlier and ensuring the quality of the final product, companies try to have a professional capable of validating front-end and understanding if work is done according to its good practices.

Companies reinforce that it is important to establish communication channels as soon as possible. Hi-Interactive mentioned that when working with other companies, promoting workshops to show their work practices helps to bring teams together and has a positive contribution to the final result.

[4] Collections that provide all the necessary components to design and develop a product such as brand identity information, style guides, principles, and best practices. [21]

Chapter 4

Problem Identification

This chapter presents the procedure and techniques used to classify the problems identified in State of the Art. Finally, the problem which will be tackled is identified and its choice justified.

4.1 Problem Discovery Approach

The dissertation topic is wide and to understand the subject, it was necessary to start by identifying the existing inefficiencies in the process and analyze where improvements would have a higher impact.

Double Diamond was the process used for both Problem and **??**. This is a framework popularized by the British Design Council [96] whose objective is helping the discovery of solutions for problems. This approach divides the work into two similar stages. Referring to its name, each of these model phases is represented as a diamond as can be seen in Figure 4.1.

The first stage focuses on understanding the problem space and framing the central problem being solved. The focal point of the second stage is the search for possible solutions. This step ends with the choice of a solution adequate to the problem identified.

Both diamonds have a divergent part followed by a convergent one. The divergent part involves gathering insights and learning about the topic. Based on the information collected, the convergent part is the measured and deliberate decision of among all the identified topics which to select as a goal to follow-up.

The selection of the problem is done in the first diamond and the identification of the solution in the second one. In the context of the dissertation work, the first phase was named **Problem Discovery Phase**. During this phase, the collaboration between designers and developers was studied in detail. In addition, the most relevant problems

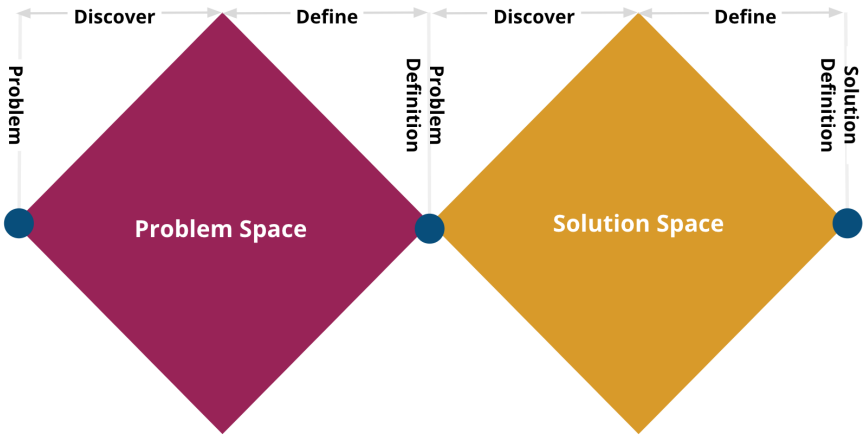

Figure 4.1: Double Diamond Diagram

were identified. The last step of this phase was the identification of the most relevant problem for which a solution will be proposed - **Problem Identification**.

4.2 Analysis of the Collaboration Process

The study of OutSystems collaboration process (see section 3.3) helped to understand the current software development process and the existing pitfalls. The problems found can be divided into three main categories, Efficiency, Work Methodologies and Ecosystem Problems.

4.2.1 Bottlenecks in the Development Process

During the Development Process, there are inefficiencies that make the project flow less productive leading to waste of effort, rework or loss of information.

When asked about friction points between groups, the most referred problem by professionals was the limited knowledge about the practices of other groups. Not being aware of practices leads to errors, inconsistencies and quality problems.

Projects are affected by the uncertainty about covering business requirements and frequent need to change those requirements. However, this problem is not easy to solve since it is impossible to predict project changes.

Professionals are not aware of the implications of their work when handing it over to others with different roles. This problem is frequent when work is done by junior developers who do not realize the problem exists. More experienced professionals are more aware about their work decisions.

When UI design and Front-end development are developed by different companies, frequently, designers are not familiarized with the OutSystems framework. Consequently, designs need to be changed and several components need to be built from scratch or redesigned.

In external software development practices presented in section 3.3.2 is common for designs to be iterated and changed after developers start programming. In this case, Front-end and Back-end developers need to detect the changes manually by visually comparing version. This is not a rigorous method and sometimes changes are not identified, small variations of colors, for instance. The tools used do not provide a mechanism for comparing versions.

Creation and usage of Live Style Guides have several problems. The Live Style Guide creation process has inefficiencies. There are parts of this process that are perceived as monotonous and not challenging by front-end specialists. Live Style Guides are often used incorrectly. The problem occurs when development teams of customer companies develop new projects based on the Live Style Guides delivered from OutSystems teams. When back-end developers do not use the components already available or need to customize them, they re-do the existing ones and errors are potentially introduced during the process. In the long run, it can affect the stability and smooth functioning of the final product. Less frequently, companies do not adopt or take advantage of Live Style Guides.

4.2.2 Analysis of Work Methodologies

Work methodologies often are limited in terms of usability on UX. It is not usual to have developers present in usability tests and the number of handover sessions, where groups pass context about the work done to other groups, is low due to strict execution schedules. Lack of handover sessions implies information not being transmitted from group to group. Consequently, the project may end up changing course due to miscommunication.

Polishing the final product and making sure every component is implemented according to the specifications is a time-consuming process. This is one of the reasons why automated tests and scripting are used as a great way of identifying problems and bugs. The majority of testing is done only by back-end developers and focusing on the functional part of the application.

Designers are allocated for a short period of time to projects and therefore only have the opportunity to be present at the early stage of projects, when they are effectively working in the project designing mockups. One of the consequences is not accompanying the development phase. Not having designers present at the development stage gives rise to errors because sometimes developers do not develop what was originally designed or misinterpret the design. The majority of the problems are related to the product User Experience. Designers are not present during the whole project, so when their collaboration is needed during development, they are already working on other projects. Designers need to re-contextualize with the project and a lot of time is wasted. Another problem is related to the fact that designers are not aware of how their designs were implemented and if their initial idea was kept.

The resolution of these problems implies changes at the team and work organization level. This group of problems cannot be solved under the context of this thesis mainly

because both the resolution of these problems and their validation require high effort.

4.2.3 Ecosystem Problems

Lastly, some problems are motivated by the ecosystem. The number of front-end developers and back-end developers with front-end knowledge does not meet the market needs. Designers from external companies are not familiarized with OutSystems.

4.3 Problem Selection Methodology

The problem domain is wide and it was necessary to identify where in the development process optimizations would be more beneficial. RICE [86] was the scoring model chosen for evaluating the inefficiencies found according to four criteria, **Reach**, **Impact**, **Confidence** and **Effort**.

Table 4.1 presents the identified collaboration pitfalls and the correspondent classifications. **Reach** is measured taking into account the number of projects affected by the problem. **Impact** is related to the improvement in projects and for professionals with the resolution of the problem. In the table, problems with bigger impact have a higher value. **Confidence** score represents how positive we are about being able to effectively solve the problem. **Effort** shows the investment needed for the resolution of the problem. The final score is given by:

$$Score = Reach * Impact * Confidence/Effort$$

This approach helps in understanding which problems should be prioritized. Problems with higher scores are more relevant when compared to the ones with lower scores. This methodology was used to classify the different problems. The scores assigned to problems were attributed relatively and do not strictly follow the RICE score classifications.

4.3.1 Problem Identification

Based on the classifications on Table 4.1, the most relevant problem is the **Poor efficiency when creating Live Style Guides**.

All projects developed with OutSystems, if done accordingly to established practices, require the creation of Live Style Guides. Since a large number of projects involve this procedure, automatizing some of its steps would have a high impact in the overall development process.

Finding a way of automating the Live Style Guide creation helps reduce the time invested by Front-end developers in the non-challenging time-consuming parts of this process. With this improvement, Front-end developers can focus on the development of more complex patterns and components. The number of days dedicated to Front-end development in a project could be reduced from 5 days to potentially 3 or 2 since the development of project section 6.2 consumes much of that time. A solution to this

Table 4.1: Most relevant problems during development processes

Inefficiency	**Reach** 0-100%	**Impact** 0-3	**Confidence** 0-100%	**Effort** 0-100%	**Total**
Poor efficiency when creating Live Style Guides	90	3	70	10	1890
Designers not familiarized with OutSystems	30	2	60	10	360
Manual and Visual process to detect design changes	80	1	80	20	320
Not enough user testing	60	2	20	70	34
Live Style Guides used in a poor way	40	2	20	70	23
Designers do not follow development phase	90	1	20	80	22.5
Lack of front-end developers/developers with front-end skill-set	20	2	20	50	16
People not caring with the implications of their work when handing it over to others with different roles	15	1	30	30	15
Uncertainty when covering business requirements and frequent need to change those requirements	20	0.5	15	100	1.5

problem can also mitigate the impact of the skill gap that exists today between Front-end and Back-end developers. It may also facilitate Back-end developers to be responsible or participate in creating section 6.2 for projects where UX and UI requirements are simpler.

4.3.2 Solution Validation

The selected problem can be subdivided into smaller problems. This way, we are not completely dependent on a single phase or final solution. This property is highly desirable and relevant. Thanks to it we are able to progressively evaluate, test and rethink the work if needed, always knowing that our latest version is stable.

The problem allows an iterative resolution. This means that the solution can be continuously and progressively improved through semi-independent deliveries where each has its own contribution towards the end-goal. Each of these improvements increases the final solution value.

This approach enables progressive solution validations as well as successive assessments of the impact of adding new increments. Our plan included the validation of the proposed solution to the selected problem by applying the artifact developed to previous projects, that is, starting from the initial design and section 6.2 template and comparing

the result obtained with the natural result and thus inferring possible gains.

We also planned to evaluate the proposed solution by using the artifact produced in projects that are being developed and thus collecting qualitative feedback on the impact of the solution. This type of validations allow the solution to be adjusted over time based on feedback.

4.4 Impact Analysis

To understand how promising the optimization in the Live Style Guide creation process are, we analyzed the impact of those improvements. The development of Live Style Guides takes 5 days. However, depending on the pattern customization level, the number of days necessary may be higher. During this process, a front-end developer builds the actual Live Style Guide and creates the project sample pages using its components.

Our main goal was to automate the Live Style Guide creation process and consequently reduce the time needed to develop it from 5 to 3 days. Improvements in this process would benefit OutSystems and specialized partner companies as well as clients. OutSystems and specialized teams in partner companies are continuously involved in projects. Reducing the time invested in a project allows them to close to double their throughput. They would be able to do the double of the work with the same investment.

The average project duration is 3 months. As such, a customer is able to carry out at least 4 projects per year. As such, from the clients perspective, the impact seems irrelevant since it represents a difference of 8 days per year. However, OutSystems has a customer base of more than 1200 clients with several parallel projects. The combination of the days saved by clients in a year equals 9600 days which is equivalent to 26.96 years saved per year.

Chapter 5

Related Work

The conversion of User Interfaces designs to front-end code was the task of the development process to optimize in this work. Developing User Interfaces is a challenging process for all the professionals involved as mentioned in research [8, 23, 33]. Designers and developers produce different UI versions for multiple contexts using their own tools until they satisfy requirements and are accepted by clients. This is a cumbersome, time-consuming and error-prone process and the final result tends to be different from what was designed.

In this chapter we give an overview of methodologies and methods designed to optimize the process. We introduce languages and models used for modeling User Interfaces as well as techniques for converting UI designs to code. Since the conversion process consists in a model transformation, we explain its main concepts. Lastly, we analyze how low-code platforms are positioned in this topic.

5.1 Model Driven Engineering

Model-Driven Engineering (MDE) is a software development methodology known for improving quality by enabling development at a higher level of abstraction [40]. This technique is based on system abstractions that reflect its most relevant features and simplify its representation and understanding [16]. To describe a system usually it is necessary to have multiple views that define it from different perspectives. Those views of the system are called **models**. To define the constructs of the language in which models are expressed, we use **metamodels**. Metamodels *"define the structure of a modeling language"* [16] and *"well-formedness rules of the language in which models are expressed"* [32].

The language is called the **modeling language**. Modeling Languages can be classified into general-purpose and domain specific according to the size of their domain. The first

group has a greater number of generic constructs and can be used in many domains. On the other hand, domain specific modeling languages are designed for a specific domain.

5.1.1 Model Transformations

A common task in MDE are **model transformations**: *"the conversion of a source model into a target model following an established transformation definition"* [32]. The transformation definition is composed by a group of rules and concepts of models that define precisely how a source model is transformed into a target model. This process is named **meta-modeling**. Each of the rules maps a specific element of the source model language into the corresponding one in the target language. Using a meta-model is especially beneficial for visual modeling languages when trying to base the implementation of the tool upon the meta-model of the language.

Having a specific language for describing model transformations is especially helpful when referring to model elements. These languages must be expressive, precise and efficient. A technical space represents all formalisms associated with a particular technology, such as concepts, tools, and techniques.

When defining a model, the source and the target models may belong to technical spaces that may be different. Transformations can be classified as endogenous and exogenous based on that. When the models are expressed in the same language, the transformation is endogenous. Some examples are Reverse Engineering, Migration and Synthesis. When the languages are different, the transformation is exogenous. Optimization, Refactoring, Simplification and Component adaptation are examples of exogenous transformations [32].

A model transformation includes at least two models but is not necessarily equal to two. Figure 5.1 illustrates two examples of transformations that include more than two models. Transformations can additionally be classified as horizontal or vertical if the abstraction level is kept or not respectively. There are three different types of tools for

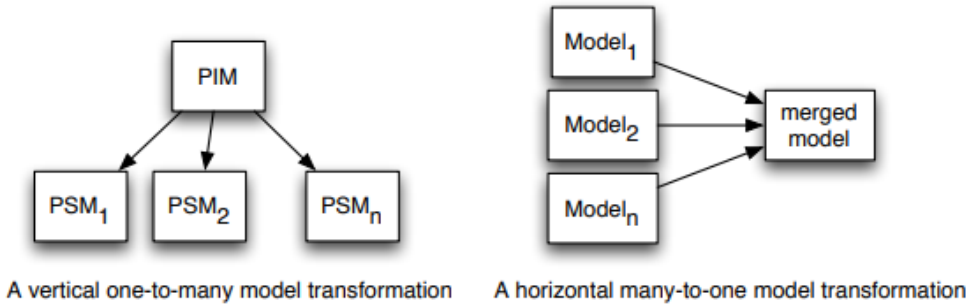

Figure 5.1: Examples of model transformations [32]

defining a domain-specific visual language by the specification of a meta-model with different architectural approaches. Some tools allow direct access and manipulation of

the internal model representation using procedural APIs. Other tools use an Intermediate Representation to export the model in a language such as XML and then use external tools to transform it. Finally, other tools provide a Transformation Language Support and a set of constructs or mechanisms for explicitly expressing, composing and applying transformations [40].

The models need to be expressed in a modeling language. UML is used for design models while programming languages are used for source code models. Transforming models correctly implies fully understanding all the models involved. The transformation of this dissertation accepts Figma or a similar Design and Collaboration Tools model as the source model and transforms it into the OutSystems Visual Language, our target model.

5.2 User Interface Model-Based Techniques

The usage of model-based techniques for developing User Interfaces has multiple benefits, namely the ease of accommodating changes and the precise definition of interface components and their relations as mentioned in [17]. When modeling an User Interface multiple models may be required for covering different UI aspects such as the presentation model, application model or navigation model.

UML comprises a set of modeling languages focused on modeling systems and its architectural aspects. Although UML notation is rich, it does not have mechanisms for describing *Human-Computer Interaction (HCI)* [37]. To solve the problem, an extension named UMLi [17] was created for helping representing User Interfaces.

New languages have emerged to model user interfaces, namely *User Interface Markup Languages*. These languages are target-independent languages for modeling UI and UX that help mitigating the conversion problem, improving the design to front-end transition as mentioned in [10]. Several examples are XML-based languages such as XIML [36], UIML [1], and USIXML [27]. These languages ease the automatic generation of code and do not require extra effort to adapt code to multiple platforms [42]. Multi-path UI development has been defined as an engineering method and tool that allows a designer to start a UI development by several entry points in the development cycle, and from this entry point get substantial support to build a high quality UI.

IFML [9] is a standard modeling language to create graphical representations of interfaces and behavior for desktop, mobile and client-server applications. IFML is not concerned with the appearance of applications but with their structure and behaviour. The language helps to model different front-end aspects such as navigation paths, user events and interaction and supports connection with the business logic, the data model and the graphical presentation layer.

5.3 UI to Code Conversion Techniques

Having mechanisms that automate the design to code translation can have a great impact on the process and decrease in the time spent coding UI designs. Besides, developers would have the opportunity to focus on developing more complex patterns [5].

The diversity of components in UI designs makes the conversion a complex process [13]. To overcome the translation problem from UI mockups to code, in this section we present some approaches that have been taken based on different techniques.

Heuristic-based Techniques

To identify UI components, the first step is always to decompose and interpret the input. Some approaches rely on histogram-based features such as Optical Character Recognition (OCR), Computer Vision and heuristic techniques to perform the input comprehension [13]. Optical Character Recognition methods are used for detecting words. Computer Vision methods identify edges and corners. They are used for extracting information from images and identifying elements.

REMAUI [33] approach takes a bitmap image as input and starts by applying OCR methods for recognizing the text as precisely as possible followed by Computer Vision techniques. The combination of the two techniques is used to define the component hierarchy. After that, similar components are identified to simplify the code generation and the result exported as an Android project. The biggest drawback of this approach is the quality of the code produced since it is difficult to interpret and change [23].

End-To-End Techniques

Other approaches such as Sketch2Code [23] and Pix2Code [5] use a neural network that covers the process from end to end, trained to understand the input and generate the corresponding code.

End-to-End Techniques combine a *Convolution Neural Network* (CNN) with a *Recurrent Neural Network* (RNN). The CNN performs unsupervised feature learning of the input image to a learned representation. The RNN is responsible for modeling a language on the textual description that corresponds to the input picture [5].

Sketch2Code prototype [23] converts hand-drawn low-fidelity sketches into front-end code for multiple platforms such as iOS, Android for Web. The CNN is trained with a UI sketch-draws dataset. It identifies the different components and generates an platform-independent object representation. Lastly, the representation is parsed and the code generated.

Pix2code converts high-fidelity mockups to iOS, Android or Web applications. The process of conversion has three main elements, a CNN, a RNN and a decoder.

Initially, the CNN understands the image, identifies its elements as well as their relations. A sequence of tokens representing the image is generated. The second element is a RNN represented as *Long Short-term Memory* (LSTM) that understands the text and

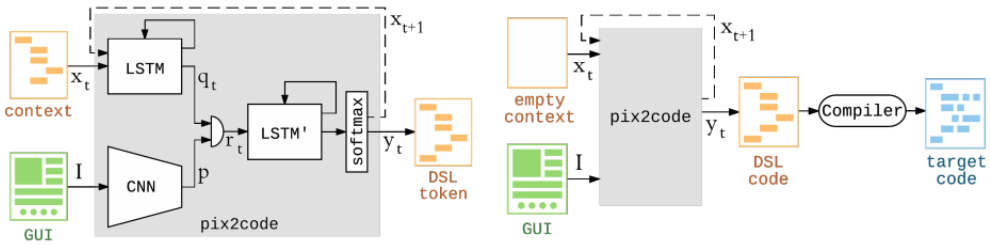

Figure 5.2: Pix2Code Architecture [5]

generates correct samples based on it. The last component is the decoder, that combines the results produced by both elements and generates the code according to the desired platform. The difference between these two models is that the second is less efficient since it is trained based on the platform while Sketch2Code is independent [23].

Pix2Code was the base for several approaches. Some approaches use a bi-directional LSTM [29], others suppress a pre-trained CNN [67]. Beltramelli [5] points out that the biggest inconvenience when using deep neural networks "is the need for a lot of training data for the resulting model to generalize well on new unseen examples". Another approach [13] converts UI images to GUI skeletons combining CNN with RNN. This method automatically collects UI data from real-world applications, unlike the previous work that defined rules for generating artificial UI data. Even Airbnb [90] developed a prototype that converts low-fidelity component wireframes to coded UI elements and tries to shorten the time between drawing an idea and its test.

Object Detection Techniques

Object detection algorithms recognize instances of an object category by extracting features and using learning algorithms [26]. One approach [26] uses it with a Deep Neural Network. First, the hand-drawn mockup is analyzed by an object detection technique named YOLO [38]. After the object identification process, components are organized according to their relative position and a hierarchical structure is defined. The hierarchical structure is then converted to code according to the desired platform.

5.4 Design to Code Conversion Tools

Several tools in the market support design and development processes separately, however, they do not provide mechanisms for converting or transitioning designers work to developers.

In the last few years, new tools focused on converting design artifacts to code have emerged whose main goal is to minimize the gap between design and front-end development. Several tools presented in the Design and Collaboration Tools section already allow to export CSS and HTML code. Developers mentioned that, despite that possibility, usually the generated code is not directly used due to its quality.

The conversion tools can be group into 3 groups: Sketch Plugins, Design File to Code and Image to Code. **Sketch Plugins** include products such as PaintCode [77] and AnimaApp [49] that extend Sketch functionalities and convert design components into the desired programming language code. **Design File to Code** tools take a design file as its input and convert it into code. Some examples are UniteUX [79], ReactStudio [83], Zecoda [99], Yotako [98] and Supernova [91].

Lastly, there is a tool in a market that converts **Image to Code**. Fronty [61] converts a JPEG screenshot to HTML and SCSS code based on UI framework Bootstrap.

5.5 Analysis of Competitive Platforms

Low-code development platforms promote the creation of web or mobile applications without or with minimal code with the drag and drop of components. The usage of these tools shortens development time and makes the entire process easier for users. Different low-code platforms have different functionalities and address front-end development differently.

Research about low-code platforms aimed to understand how other companies operating in the same market as OutSystems are positioned in this area and to guide the definition of this thesis solution proposal.

Multiple platforms such as ServiceNow [88], Salesforce Lightning [87] and Front-end as a Service platform Mobify [70], provide Sketch assets with the most common design patterns. Those resources are Sketch representations of the platform elements and help to guide designers by giving them the possibility to create their designs with UI elements that match the platform components.

The majority of the platforms analyzed do not have a mechanism to convert UI design to the platform components and recommended the usage of InVision or Figma to inspect those artifacts and boost translation.

Kony [66] developed Sketch and Photoshop plugins that translate those design files into components in their platform such as image assets and widgets that can be used in their product Kony Visualizer [63]. However, plugins have limitations. The Sketch plugin only allows exporting Artboards and Group Layers. These mechanisms are great helpers in the conversion process however only allow exporting particular elements and not complex final systems mockups. Unlike the ones mentioned above, some tools do not provide any mechanism to address this matter. Some examples are Google App Maker [50], QuickBase [82], Zoho Creator [73], TrackVia [92], Nintex Workflow Cloud [97] and KissFlow [65].

5.6 Summary and Discussion

The translation from design artifacts to their front-end representations has only recently started to be researched. Nevertheless, different tools and approaches to ease conversion

have already emerged making use of different techniques.

Although using MDD and IFML may seem promising, the tools are not mature enough and not appropriate for complex systems as explained in [22].

Methodologies analyzed in section section 5.3 try to solve the conversion problem based on techniques such as Computer Vision, Neural Networks and Object Detection. These tools take images or hand-drawn sketches as inputs. However, real-world design processes are performed using design tools and the input of these approaches is not aligned with the actual procedures. Despite that divergence, the relevance of studying these approaches relies on understanding the techniques used to interpret and convert images to code.

The low-code platforms market is expanding rapidly and it is expected that the number of users of these platforms will increase drastically in the near future as reported in [39]. Gartner [93] predicts that by 2024, 65% of application development will be carried out using low-code platforms. The existing conversion tools do not support these trends and consequently are not particularly future-proof, resulting in a poor growth model.

The majority of low-code platforms do not have mechanisms to address the inefficiencies in the design to code conversion process. From the tools analyzed in section 5.3, only Kony provides Sketch and Photoshop plugins for exporting designs into widgets, images or forms in their IDE. However, plugins have limitations. They were developed for exporting particular elements and do not allow users to completely replicate all the components in their work. The remaining tools only provide design assets for Sketch or do not provide any resources. A large number of low-code platforms recognize Figma and InVision as good transition tools and recommend its usage.

Chapter 6

Technical Approach

In this chapter, we present context information about the solution delivered by this dissertation. We start by describing the creation process of Live Style Guides presented earlier in chapter 3. Next, we explain in detail our base models: Sketch and OutSystems.

6.1 OutSystems UI Framework

The **OutSystems UI Framework** is a UI library for OutSystems web and mobile applications developed by OutSystems. It contains more than 70 UI patterns representing the most-used components [76]. **Patterns** are reusable UI elements containing an HTML structure that do not include business logic since their purpose is solely to define the elements' standard structure. Some examples are dropdowns elements and section expandable components. It is a good practice to promote frequently used UI elements to UI patterns.

6.2 Live Style Guides

Keeping consistency across related applications is crucial for providing final users a good experience. The usage of Live Style Guides helps achieve that consistency.

Style Guides work as libraries that include all documentation and guidelines regarding the appearance of applications. Live Style Guides extend Style Guides by including live examples and code snippets of the mostly used patterns. They work as repositories for both designers and developers and reflect the branding of customers according to the OutSystems standards. Besides allowing for consistency in projects, Live Style Guides are important helpers creating great user experience. Their usage reduces maintenance costs of projects and increases the development velocity when building pages and patterns.

Live Style Guides abstract CSS and JavaScript complexity from developers. These artifacts are based on previously made high-fidelity mockups of the system. The OutSystems LSG are based on the OutSystems UI framework.

6.3 OutSystems User Interfaces Architecture

OutSystems applications User Interfaces architecture is organized in layers as illustrated in Figure 6.1.

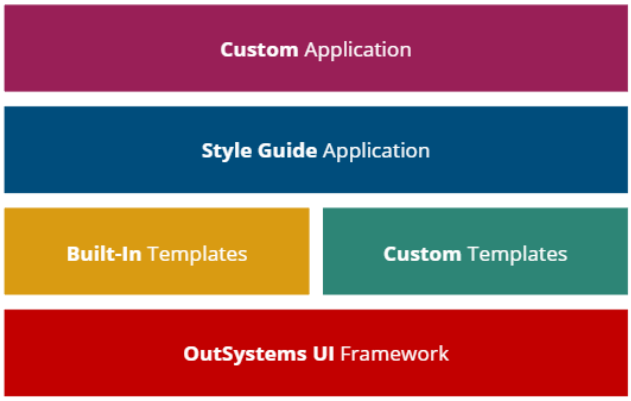

Figure 6.1: Style Guide Architecture

The base of the user interfaces architecture is the OutSystems UI. **Built-In Templates** are ready-to-use Screen Templates developed and maintained by OutSystems. Templates applications inherit from OutSystems UI and include Theme and Template modules. The Theme module defines the look-and-feel of the application and key visual elements using CSS and Layout. The Template module includes customization made in the OutSystems UI and in the Style Guide.

Custom Templates represents customized templates for applications. When customization is needed, three approaches can be used depending on the number of changes needed. For some projects, it is enough to make changes to the application template. Other projects demand more changes, so the theme and the template need to be changed. Finally, for projects with a more disruptive look-and-feel, a new style guide is manually built from scratch.

Style Guide Application is an application that defines the styles and layouts for applications. It has the brand visual rules, colors and patterns. Its purpose and characterization are explained in section 6.2.

Custom Application is a group of modules connected by an Application Template. The Application Template can be Built-In, Custom or from the Forge [1].

[1] The Forge is a repository created by OutSystems that contains modules, connectors and UI components that can be used in applications.

6.4 Live Style Guide Creation Procedures

The Live Style Guide creation takes place after the UI Design phase of the Customer Sucess development process explained in section 3.3.1 and is based on a Sketch file as input. This stage is performed by a front-end expert. The standard duration of this process is 5 days but it can vary according to the project complexity. The creation of a LSG comprises the following steps:

1. The **Setup** is the first phase in the Live Style Guide Creation process. This step comprises the creation of the Live Style Guide and Theme application.

   Using a single base application with the default OutSystems UI styling, the front-end team is responsible for manually setting up both applications before the customization stage. Initially, every module is cloned and renamed with the customer's brand name. After that, all instances of *[CustomerName]* in the module are manually substituted by the actual customer name. Finally, new dependencies between the cloned modules are established and the previous ones removed since the new modules were consuming features from the standard applications. The Setup takes around 2 hours to be completed.

2. The next three days are dedicated to **Customization**. At this stage, the OutSystems UI components and templates are customized according to the UI high-fidelity mockups created previously by the UI Designers. The customization process comprises the creation of a CSS stylesheet. This artifact is added to the Theme module.

3. After styling the elements, it is the **Custom Patterns Development** stage. Usually, more complex projects have highly customized UI patterns and widgets specifically made for it. These components are patterns that structurally do not exist in the OutSystems UI framework presented in section 6.1. Since they are different from existing components, they need to be created from scratch. Usually, experts dedicate 1 day to custom patterns development.

4. The effective development of Live Style Guides ends with the **Sample Pages Creation** using the components already developed and based on high-fidelity mockups of the system. The number of days spent creating sample pages varies according to the complexity of the project and the standard duration of this stage is 2 days.

5. **Pre-Release and Delivery** The final steps are the **Pre-Release** where the quality tests are done and the **Release and Delivery** where the work developed by the team is presented to the client.

Resultant Artifacts

By the end of the Live Style Guide creation process (see section 6.4), the artifacts delivered

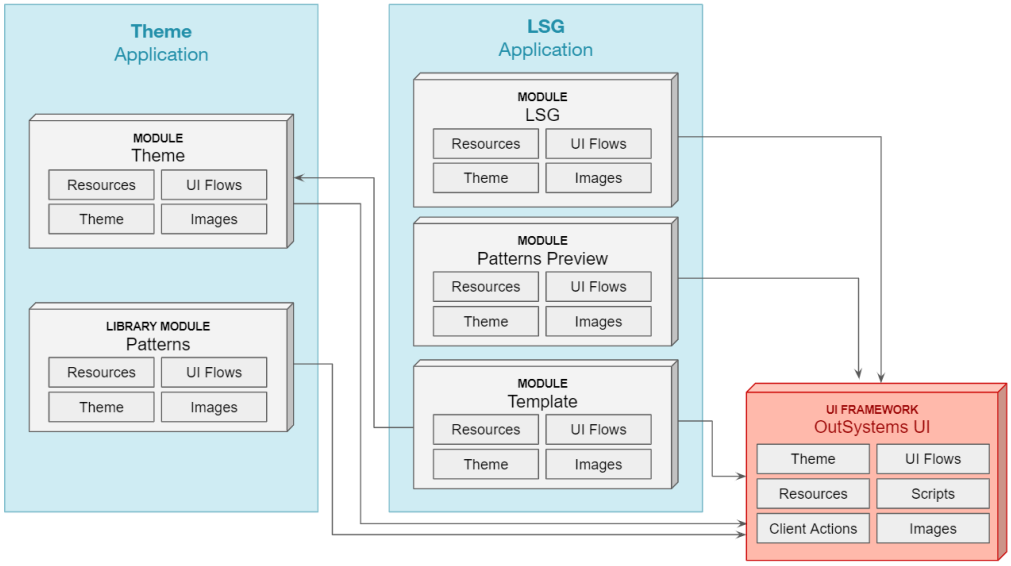

Figure 6.2: LSG OutSystems Applications Organization

by the front-end experts to the client are two OutSystems applications: Theme and Live Style Guide, illustrated in Figure 6.2. The Theme Application has project styles and custom patterns and goes into production unchanged. The application part, depicted on the left hand side of Figure 6.2 includes:

- **Theme Module** which is the base module of both applications. The module base is the OutSystems UI which means that the base CSS Theme inherits it. In the Theme module, style rules not defined in the OutSystems UI are rewritten using CSS in the module stylesheet during the customization step. Resources such as the application logo and fonts are also part of the module. These assets are saved, respectively, in the Images and Resources folders of the module and shared across modules through dependencies.

- **Patterns Module** where new specific patterns that do not exist in the OutSystems UI and need to be manually created are added. The Patterns module is a library module. Library modules do not have state and do not access or store information in the database. The purpose of this type of module is to ensure reusability among applications.

The Live Style Guide application depicted on the right hand side of Figure 6.2 includes documentation of the project styles and custom patterns and does not go to production. The application includes:

- **Patterns Preview Module** contains all the OutSystems UI Framework widgets and ui patterns that are used in the LSG module. It is a dependency of the LSG module.

- **Template Module** when new applications are created based on the style specifications defined in Theme and LSG applications, the template module is the starting

point and new apps inherit already customized elements such as themes, layouts, blocks, or logic.

- **LSG Module** that includes a live demo of the OutSystems UI widgets and custom Patterns.

6.5 Base Models

The source and target models of our approach were selected based on the OutSystems software development process presented in section 3.3.1. As the most used design tool in the ecosystem is Sketch [89], a representation of the Sketch DSL was chosen as the source model. Since the project is inserted in the OutSystems context, the target model of our solution is the OutSystems model.

Although we are using Sketch and OutSystems as examples, to validate and illustrate our proposal on how to transform design artifacts into web technology artifacts, the solution architecture proposed is designed to be generic and not to be limited to a specific design or web technology. This way, in the future it will be easy to scale and generalize for other technologies.

Other design tools, such as Figma [56] have similar meta-models in comparison with Sketch [89]. Consequently, their processing and manipulating would require a similar approach. Since our tool already supports the conversion to OutSystems, introducing these tools would require only their transformation to the intermediate representation.

6.6 Sketch Visual Domain Specific Language

Sketch [89] is a vector graphic editor used by designers to produce high-fidelity representations of user interfaces. The Sketch language is a visual Domain Specific Language (DSL) where every element is represented by a Layer. The manipulation of the elements is done through Sketch Interface as can be seen in Figure 6.3.

Different types of layers are characterized by different sets of elements and attributes. The most relevant attributes and classes for our tool from a Sketch file are shown in Figure 6.4.

The most relevant types of layers that compose the Sketch concrete syntax and that are frequently used in the process we are studying are the following:

- The **Group** represents a collection of 1 or more layers. A group is a type of layer used for organizational purposes.
- The **Page** element which is an instance of a group layer representing a canvas in the document.
- The **Artboard** element which is an instance of a group layer representing a collection of layers.

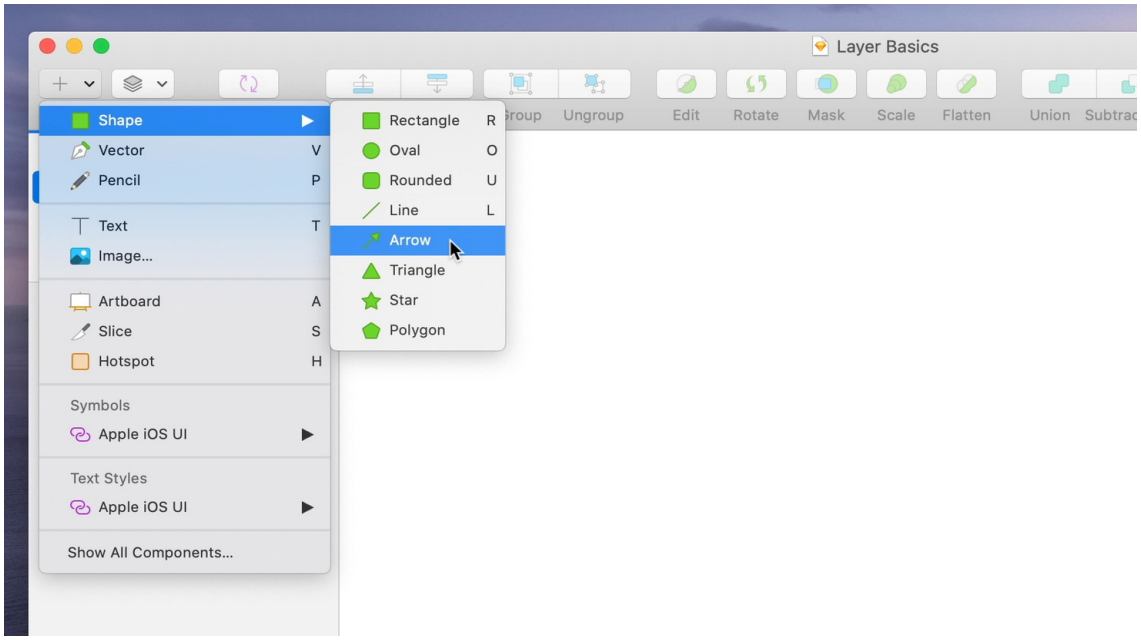

Figure 6.3: Sketch Interface from [89]

- The **Image** layer whose content is an image file.
- The **Shape** element that is the most common layer type. A shape layer is used for introducing pre-made or new shapes on the document.
- The **Text** element which is a layer whose content is a text element.
- The **Symbol Master** that is an instance of an Artboard (and consequently of a Group layer). A layer frequently used across the document can be promoted to a symbol. Every time changes are made in the Symbol master, modifications are propagated to all its instances.
- The **Symbol Instance** layer that is an instance of a Symbol Master layer. A symbol instance layer has a reference for its master and keeps its structure. However, attributes of the inner layers of a Symbol Instance may be changed such as colors, shapes or text styles. A Symbol Instance layer stores the new attribute values as Override Values, elements of the Symbol whose value is changed in relation to the original Symbol Master.

Sketch layers are also characterized by a group of elements:

- **Frame** - that contains the dimensions of the element as well as its coordinates in the parent layer.
- **Points** - which describes the radius of the four corners of the layer. A layer can have 4 points representing the 4 corners or no points when the layer is a rectangle.

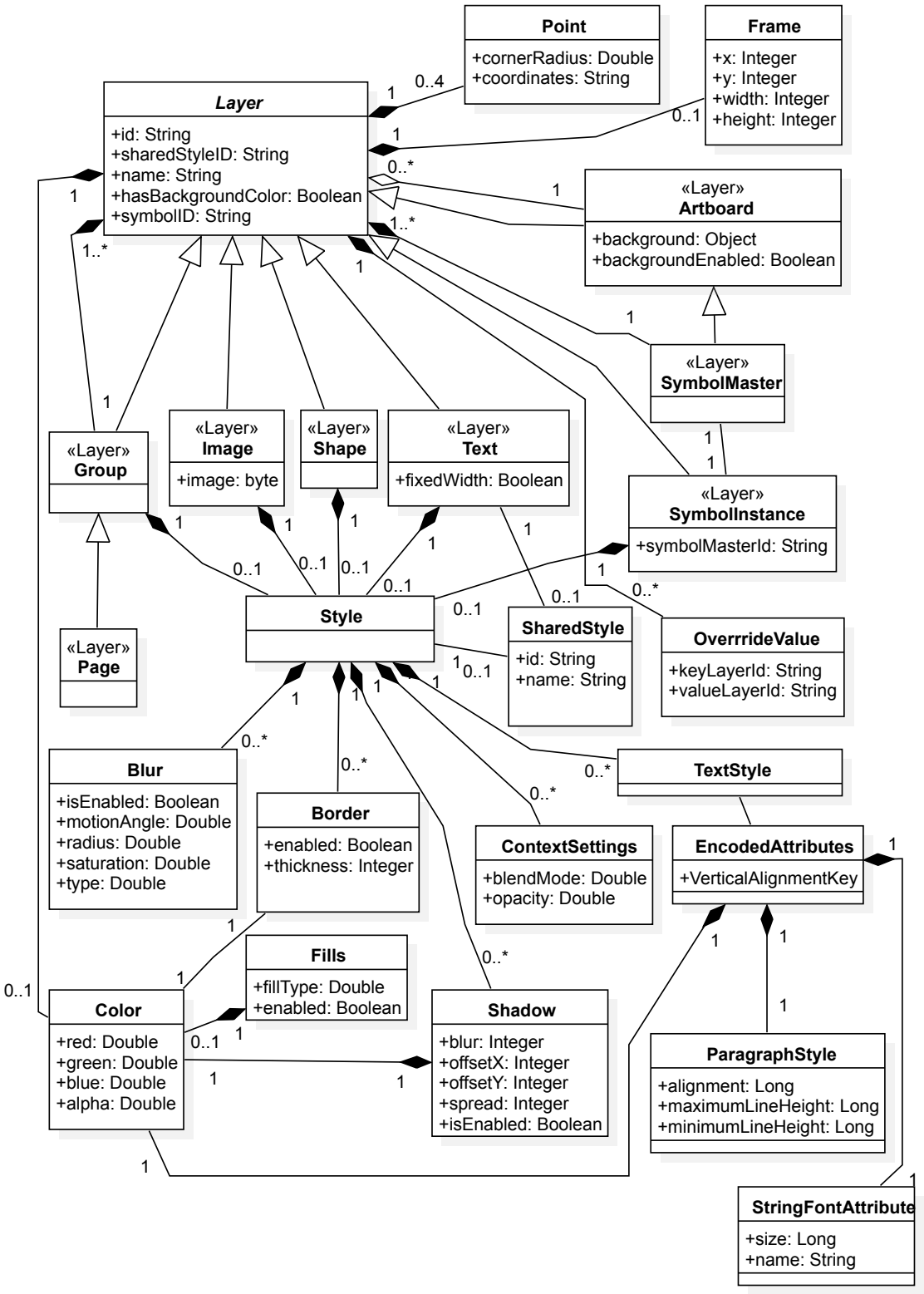

Figure 6.4: Sketch Page Domain Meta-model

- **Color** - that represents the layer background color. If the background color is transparent, the layer has no color.
- **Override Value** - which is a list of pairs where the key represents the original inner layer of the symbol master and the value its new value. A layer can have elements of the Symbol whose value is changed in relation to the original Symbol Master.

Group, Image, Shape, Text and Symbol Instance Layers have a Style property. Styles are composed by Blur, Shadow, Border, Text Styles and Shared Styles.

6.7 LSG Sketch Model

The Sketch Visual Domain Specific Language is powerful and its notation allows the creation of a wide variety of compositions. However, there is no one-to-one mapping between design concepts, namely those used in Sketch and the web technology UI elements.

For the reasons mentioned above, we restricted Sketch's modeling space to a Sketch file that works as a template. The use of this file as a sharing point facilitates the identification of the elements. Mapping a group of layers to a widget would not be possible without a template and using a model-based methodology given the variety of Sketch trees allowed to represent the component.

The file used as our Sketch template contains representations of every component present in the OutSystems UI framework crafted on Sketch. The file was developed during an extended period of time by the OutSystems UI team and corresponds to a continuous optimization of the current framework. Whenever the OutSystems UI framework is updated, the file is also updated. In enterprise projects that follow the established practices, the Sketch file is used as a base for UI designers to customize components according to the client branding. Our source model is the current version of the Live Style Guide Sketch file.

The Live Style Guide Sketch file structure is illustrated in Figure 6.5. The document is divided into four pages:

Styles

The Styles Page contains base design information divided into five artboards. Each artboard has information about an element, namely, typography (Figure 6.6), colors (Figure 6.7), shadow styles (Figure 6.8), states (Figure 6.9) and text styles (Figure 6.10).

Symbols

The Symbols Page contains all the Symbol Master Layers representing logos, headers, buttons among others. The symbols in this page are customized with the styles defined in the Styles Pages. Instances of the symbols defined in the Symbols page are used in the

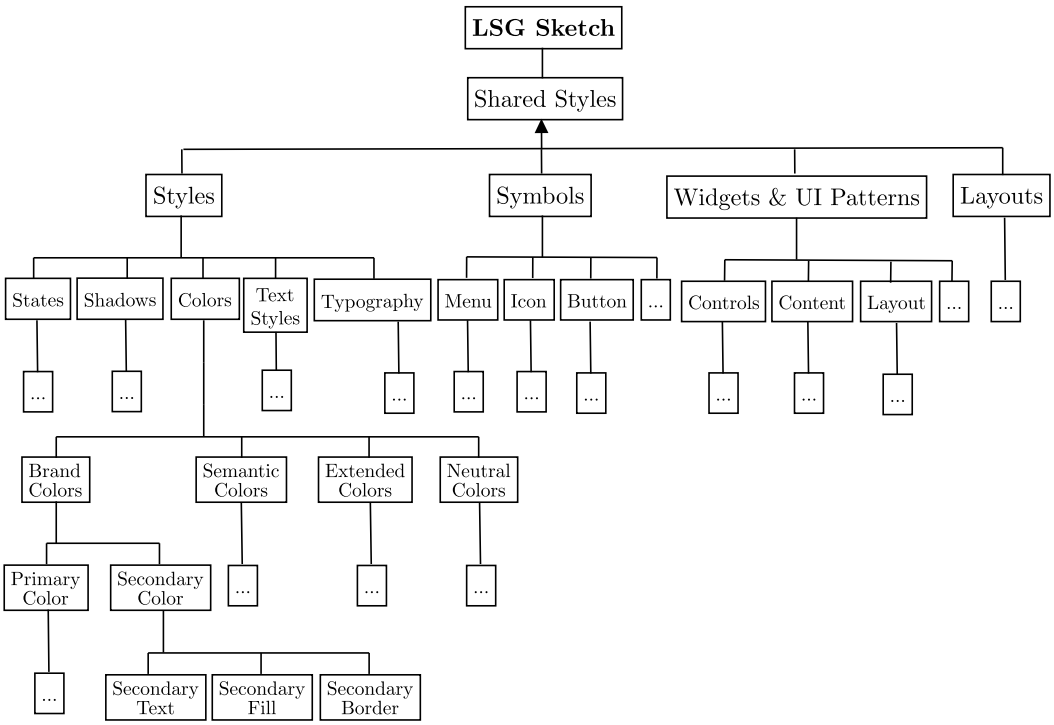

Figure 6.5: LSG Sketch File Structure

Layouts and UI Patterns/Widgets pages. Every time the master symbol is edited, changes are made to every instance.

UI Patterns and Widgets

The UI Patterns and Widgets page contains the UI Patterns and Widgets of the OutSystems Visual Language divided into thirteen artboards. One of the artboards is illustrated as example in Figure 6.11. Every component is represented by multiple instances representing all the widget states such as visited, hover or filled.

However, depending on the version of the LSG Sketch template, the organization of the document varies slightly. In order to have a more flexible tool, the processing of the UI Patterns and Widgets pages is done by identifying the groups present in each artboard and not the default organization of each page. Therefore, the tool can handle structurally different formats as long as its semantics are equivalent.

Layouts

The Layout page includes pre-built screen components arrangements such as Login, Top Menu and Side Menu.

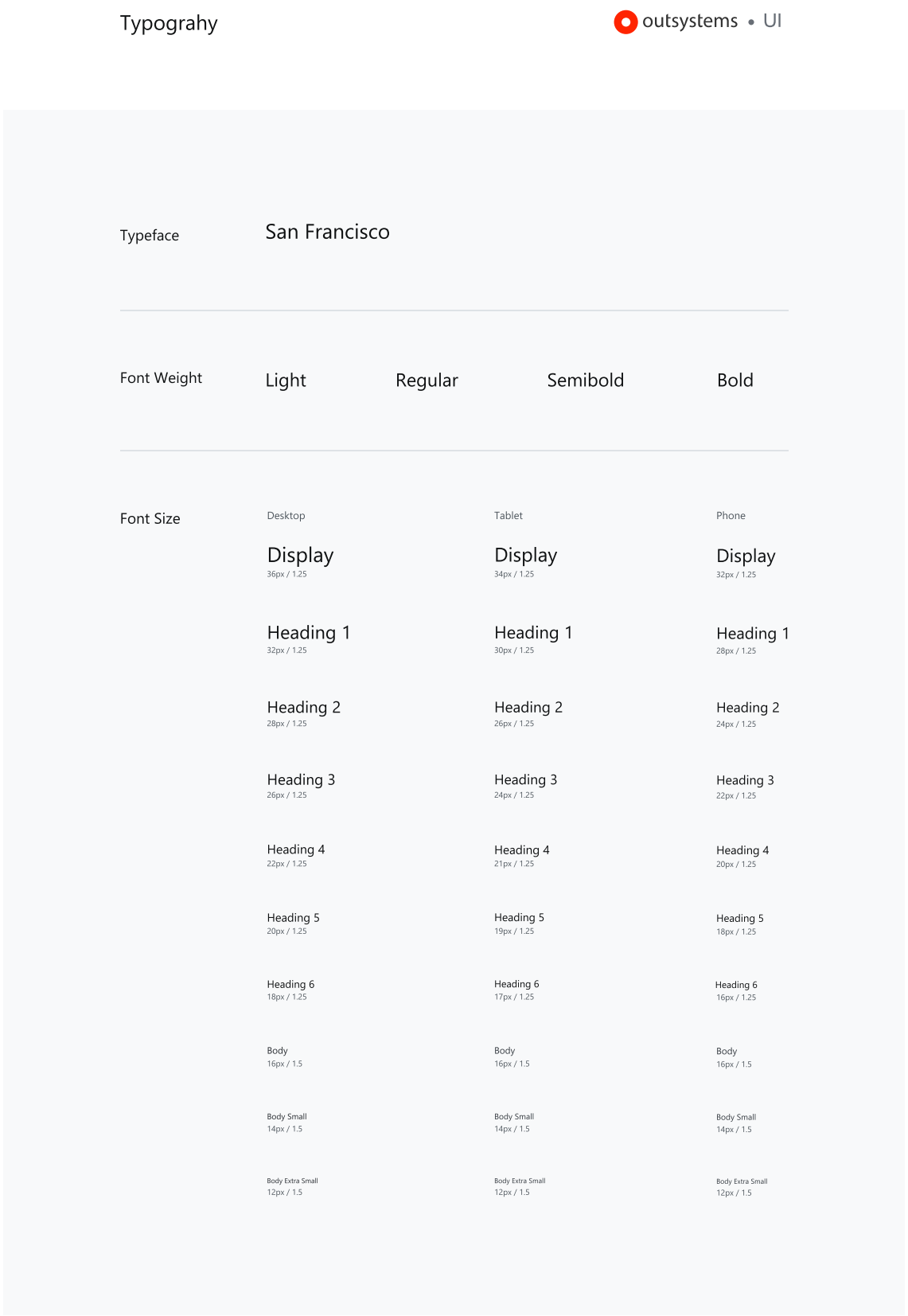

Figure 6.6: Typograpgy Artboard from the Styles Page

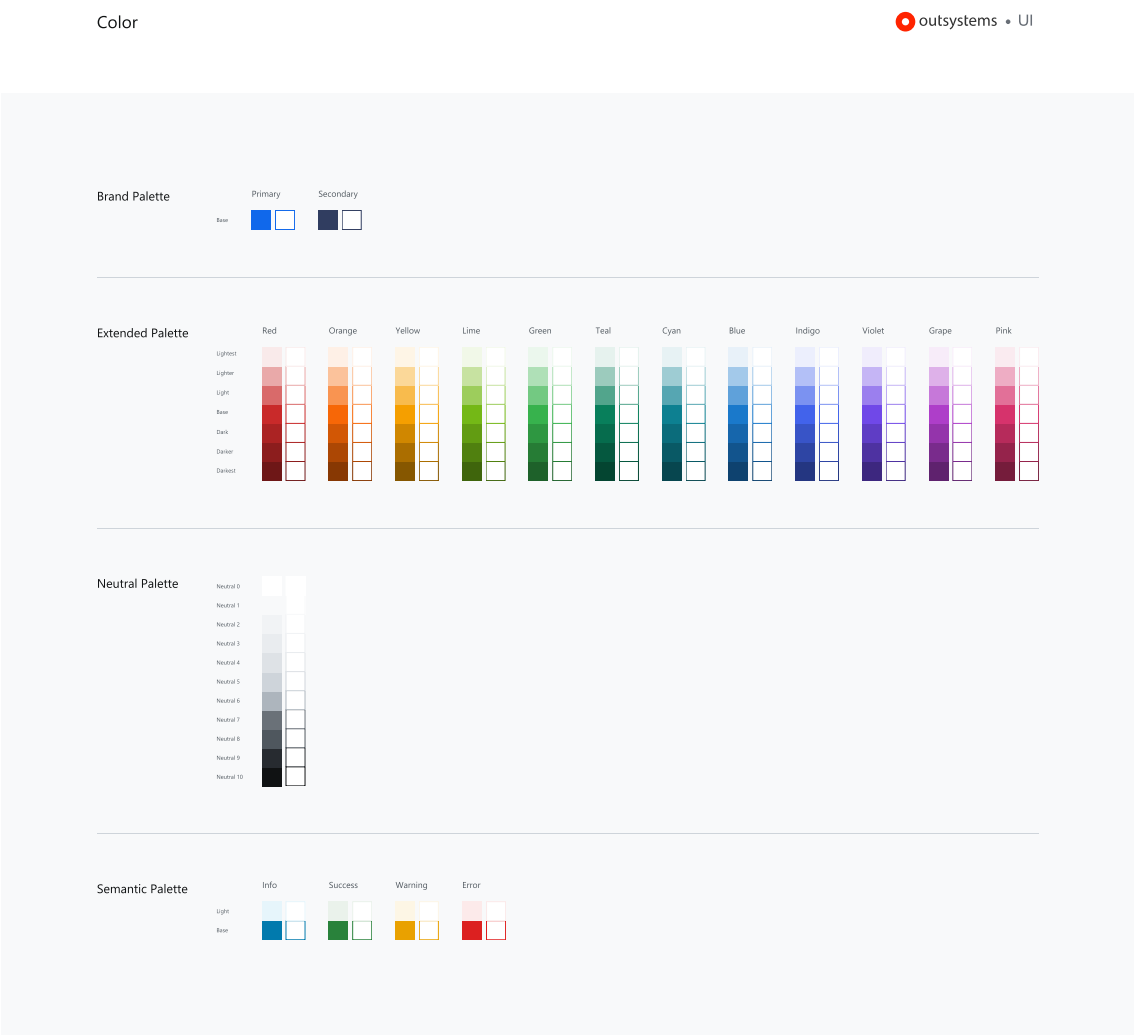

Figure 6.7: Color Artboard from the Styles Page

Besides these pages, the model contains a set of style rules that can be applied and shared across the document layers named Shared Styles. The usage of these style elements makes the process of customizing layers easier and eliminates the need to redo styles. Every time a layer uses a Shared Style item, the layers have a reference to it.

6.8 OutSystems Model

OutSystems' platform components such as Service Studio and the Compiler manipulate a representation of the OutSystems Visual Language called the **Model**. The Model has classes that correspond to the elements available in the OutSystems Visual Language.

An instance of the Model is represented by a hierarchy of objects whose root is an eSpace. An eSpace is a module where it is possible to develop the application and do tasks such as create screens or define the logical part of the application. Instances of the

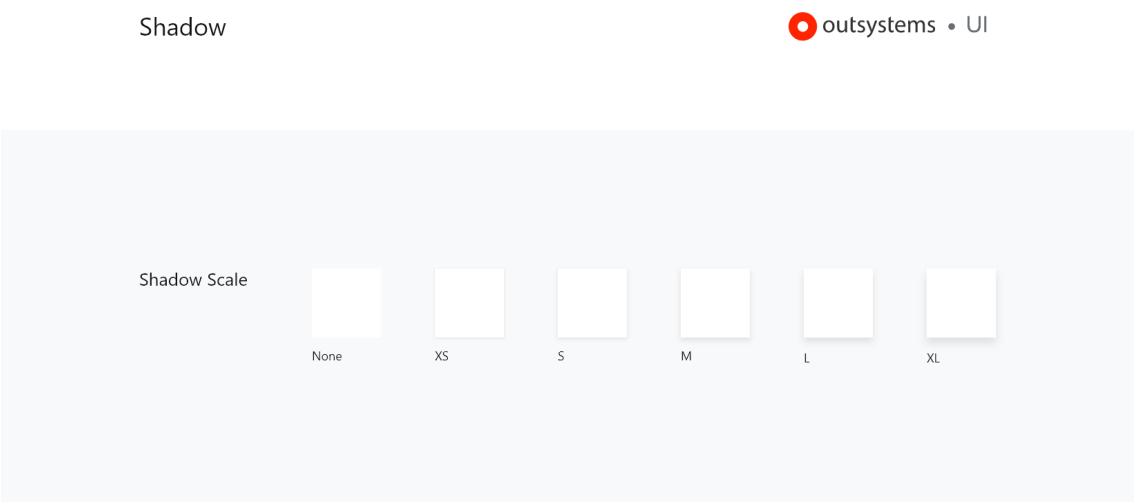

Figure 6.8: Shadows Artboard from the Styles Page

Model are organized hierarchically as illustrated in the code block below. OutSystems Applications are OAP (.oap) files, OutSystems Application Pack. Each module is an OML (.oml) file (OutSystems Markup Language) and contains all information about a module.

```
<Nodes.WebScreen Key="u39uHLOjwUiwnLVaxIF3VA" X="3196" Y="2960" Name="WebScreenl">
    <Widgets>
     <WebWidgets.Container Key="p8j0s309mkKyjY3JoVOsE" Align="Center">
      <ChildWidgets Count="3">
       <WebWidgets.Text Key="xw5+7pj3M9k6u7fS5p3qMiLA" Value="Hello, World!"/>
       <WebWidgets.InlineExpression Key="g3dIOfAMNkKIJIOChTr2H+A" Name="TimeFxpr" Example
            ↪ ="10:15" EscapeContent="Yes">
       </WebWidgets.InlineExpression>
      <WebWidgets.Text Key="HgEEDdwgnEeHkHUbrv71FA" Value="."/>
      </ChildWidgets>
     </WebWidgets.Container>
    </Widgets>
<Title>
    <ParsedExpression>
     <Text Type="tuROOBXPvOEyU76NWO+1ux0" Value="Hello"></Text>
    </ParsedExpression>
</Title>
</Nodes.WebScreen>
```

Listing 6.1: Sample of the OutSystems Model

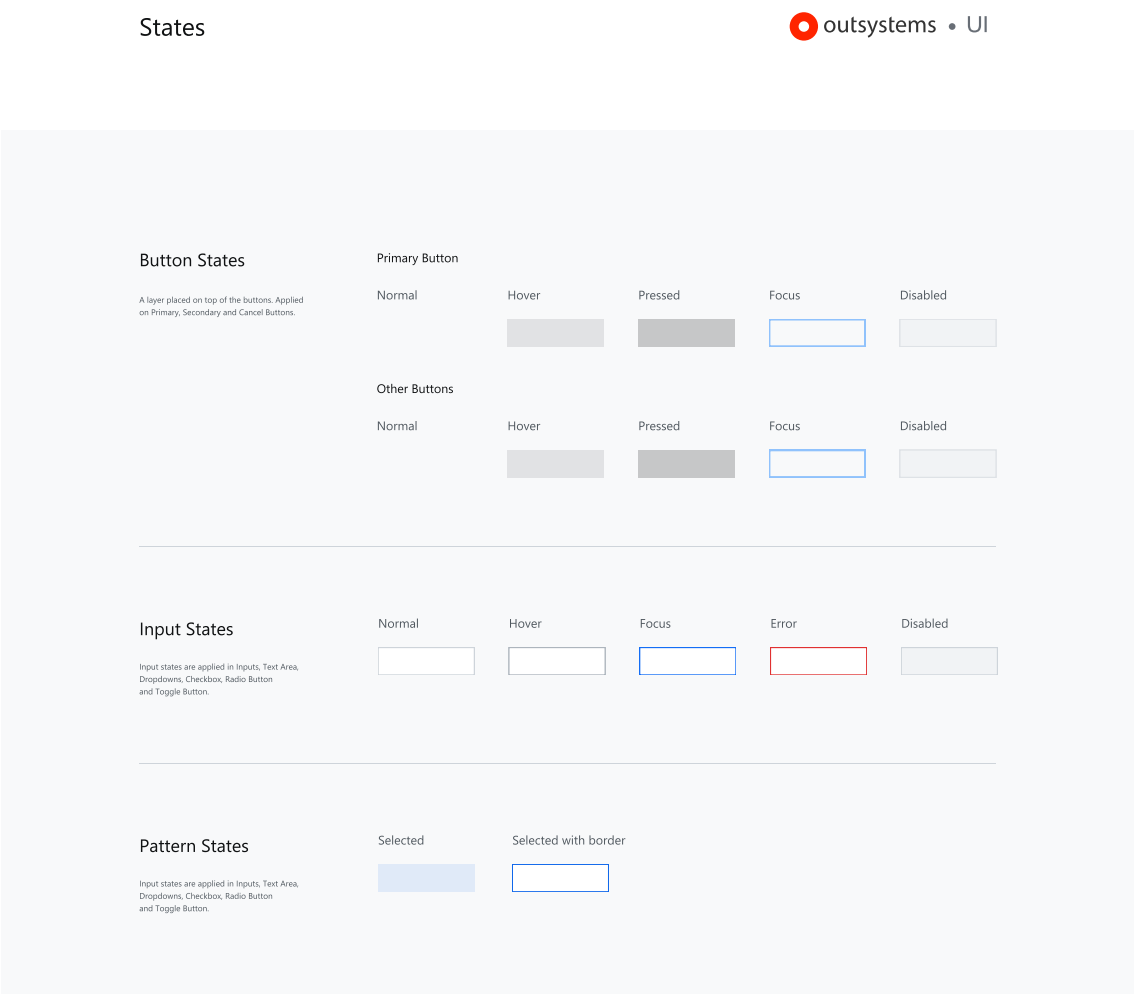

Figure 6.9: States Artboard from the Styles Page

6.9 OutSystems Applications Customization Levels

The *Starter LSG Template Application* inherits the OutSystems UI predefined appearance. However, changing the application or widget appearance can be done by writing additional Cascading Style Sheets (CSS) or JavaScript.

In the customization process described in /autorefsubsec:customization, we identified a process of transforming a template LSG to a fully customized app that can be used in the project. Applications customization can be divided into 4 levels:

- **Level 1** - The simplest level of customization comprises the customization of CSS styles. Style customization includes overriding variables declared in the OutSystems UI framework values. The element group includes colors, typography, text styles, states and shadows.
- **Level 2** - Customization that includes overriding the styling rules defined by the

Figure 6.10: Text Styles Artboard from the Styles Page

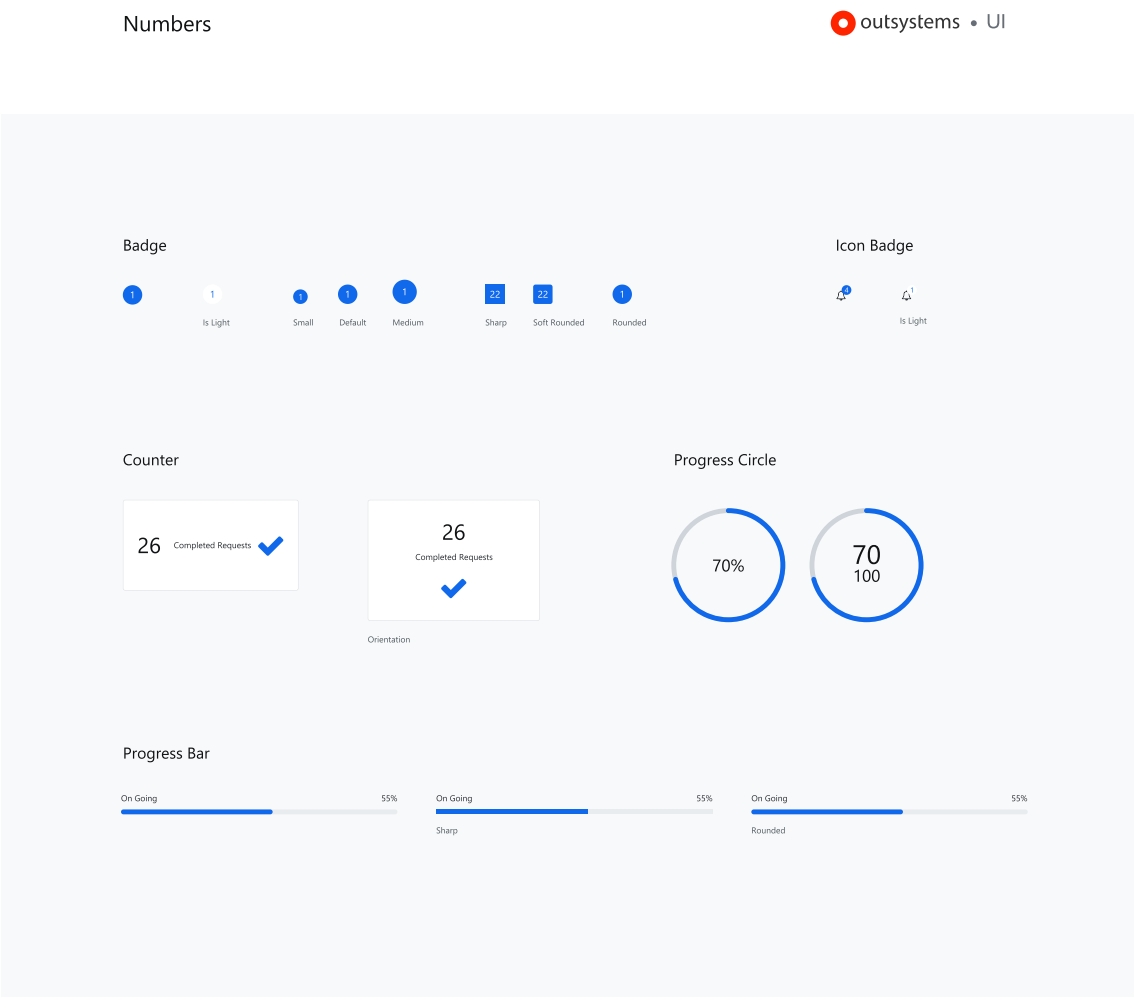

Figure 6.11: Desktop Numbers Artboards from the UI Patterns and Widgets Page

OutSystems UI framework. In this level, no new properties are added to elements.

- **Level 3** - Customization that goes beyond the attributes defined in the OutSystems Design System. New CSS rules are added, and, in existing CSS rules, new properties can be added.

- **Level 4** - Customization that changes the default UI pattern structure and is not contemplated in the Level 3. The creation of custom Widgets and UI Patterns is included in this level.

6.10 Solution Architecture

As explored in the previous chapters, the Live Style Guide manual creation process comprises inefficient and non-challenging tasks. This process is in the transition between

design and front-end development and implies the conversion of high-fidelity design representations to an OutSystems domain language application.

In the OutSystems context there is no tool or mechanism to help and ease the conversion process. Taking into account the LSG creation procedures explained in section 6.4, the solution proposed is going to cover the **Setup** and **Customization** phases described in section 6.4. There is space for automation in this phase since this is the phase where developers invest more time.

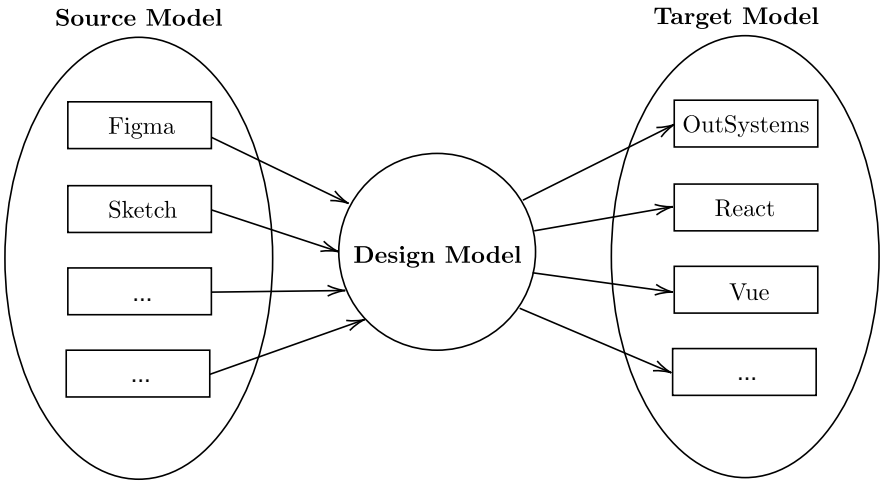

Figure 6.12: Logical Solution Architecture

The logical architecture of the proposed solution is illustrated in Figure 6.12. The solution is based on a unidirectional model transformation from the design representation to the web technology. The architecture has three main components, the **Source Model**, the **Design Model**, and the **Target Model**. The source model is a representation of a visual design language. For manipulating the OutSystems model, we are going to use Model API. The design model is an intermediate language that abstractly represents the model. The target model is a web technology domain language. In the thesis context, the target model is the OutSystems model.

The solution architecture proposed is designed to be generic as possible and not dependent on a specific design or web technology. This way, in the future it will be easy to extend with the remaining widgets and ui patterns and generalize for other technologies.

The majority of approaches discussed in chapter 5 are located on the design side. We have undergone a rethinking of the problem by trying to develop a solution independent from any technology for translating design artifacts to OutSystems domain language could bring more benefits not only for the OutSystems ecosystem but for other web technologies.

6.11 Intermediate Representation Generation

To bridge the Design and the Web Technology Model and allow extensibility to other technologies, we used an intermediate representation. The intermediate representation helped having a tool independent representation of the design artifact. The intermediate representation metamodel is illustrated in Figure 6.13.

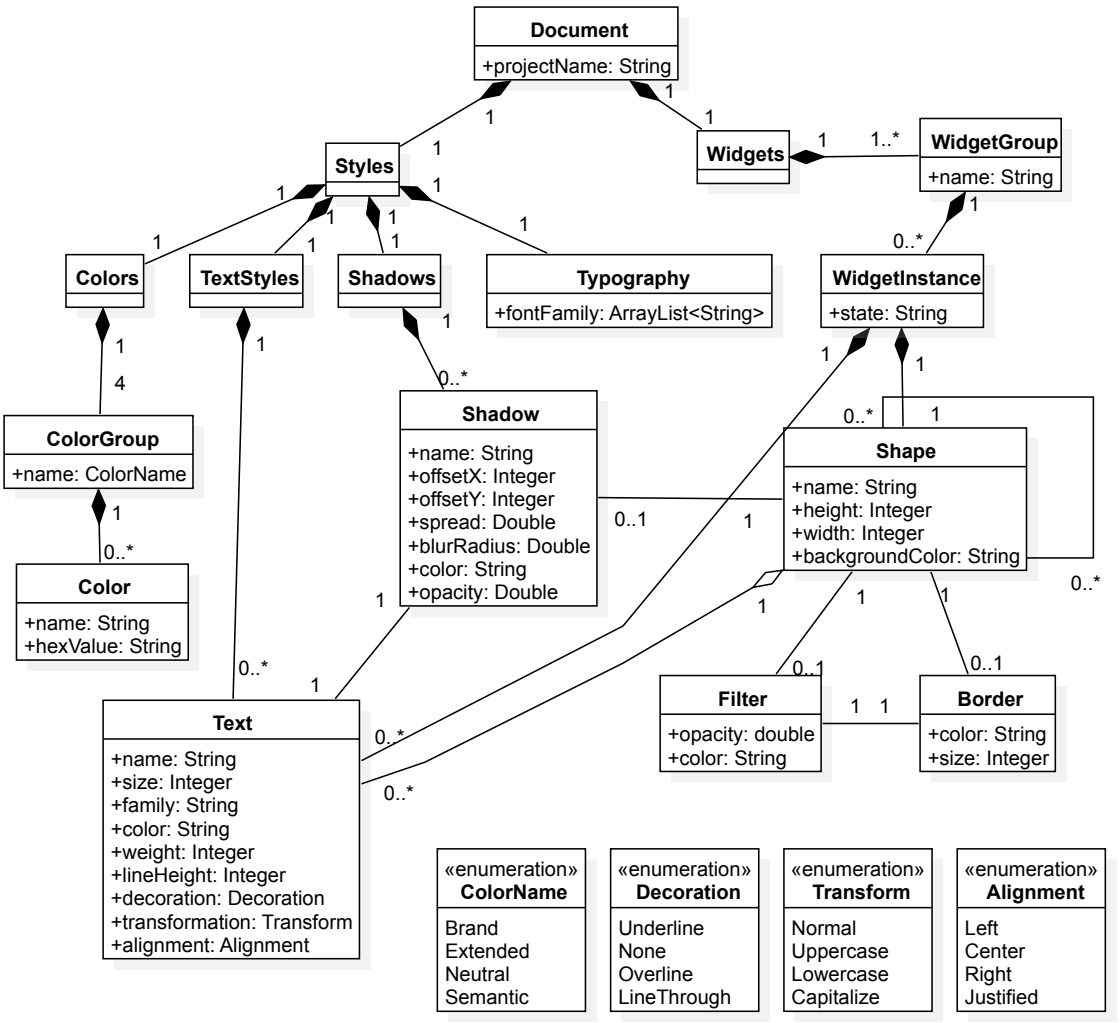

Figure 6.13: Intermediate Representation Meta-model

The generated intermediate representation is a XML document inspired by the structure of the XML-based languages analyzed in chapter 5. The following code block shows an example of the intermediate representation schema:

```
<!--Intermediate.xml-->
<?xml version="1.0" encoding="utf-8"?>
 <live-style-guide name="hvl-livestyleguide">
  <styles>
    <color>
      <brand-color>...</brand-color>
      <extended-color>...</extended-color>
      <neutral-color>...</neutral-color>
      <semantic-color>...</semantic-color>
    </color>
  </styles>
  <uipatterns>
  </uipatterns>
 </live-style-guide>
```

Listing 6.2: Intermediate Representation Example

Like the OutSystems UI Kit for Sketch, the file is divided into 2 sections: **Styles** and **UI Patterns**. The Styles element has the style rules corresponding to the first level of customization. Variables used across the document and elements, such as colors, font sizes and borders, are declared in this section. In the UI Patterns section, all the widgets and UI Patterns are represented. The rules for mapping elements are described in Table 6.1.

Page layers are represented in the intermediate representation as Groups. Group and Symbol Instance Layers are Group Elements.

The Text elements described in Table 6.1 rule number three are the representation of the source model's Text Layers. These elements are characterized by a group of attributes, namely font family, weight, size, line height and coordinates X and Y.

UI Patterns are represented in our Intermediate Representation as compositions of Text Elements and Layer Elements.

6.12 High Level Tool Architecture

Our solution takes the form of a web application developed using the OutSystems IDE and has an extension with the tool code. The high level interaction diagram is shown in Figure 6.15.

Our main goal is to automate the translation of high-fidelity design artifacts to OutSystems applications conversion of the OutSystems Software Development Process, thus reducing the time invested in this step and the amount of inconsistencies and errors introduced. The way the solution architecture was designed allows for scalability for other technologies in the future since it does not rely on a specific design or web technology.

We aim to transform a design tool file instance to instances of the OutSystems Model. Taking into account that both models are based on the OutSystems UI design system, we are modeling the customization made upon it and using it to generate customized OutSystems applications.

Table 6.1: Customization Transformation Framework

Rule Nr	Source Model	Intermediate	Web Technology
Layers			
1	Page	Group	-
2	Group	Group Element	-
3	Text	Text Element	CSS Attributes
4	Shape	Element	CSS Attributes
5	Image	Image Element	CSS Attributes
6	Symbol Instance	Element	CSS Declaration
Elements			
7	Point	Coordinates + Radius	Border- Radius
8	Fills	Hex Color + Opacity	Background- Color
9	Border	Hex Color + Size	Border-size, Border-Color
10	Shadow	Spread, Offset x Offset y, Color	Box-Shadow
11	Blur	Radius, Blur Type, Saturation	Filter
12	Text Font	Font Name	Font-Family Font-Weight
13	Paragraph Style	Text Element	Alignment Line-Height
14	Text Color	Hex Color	Color

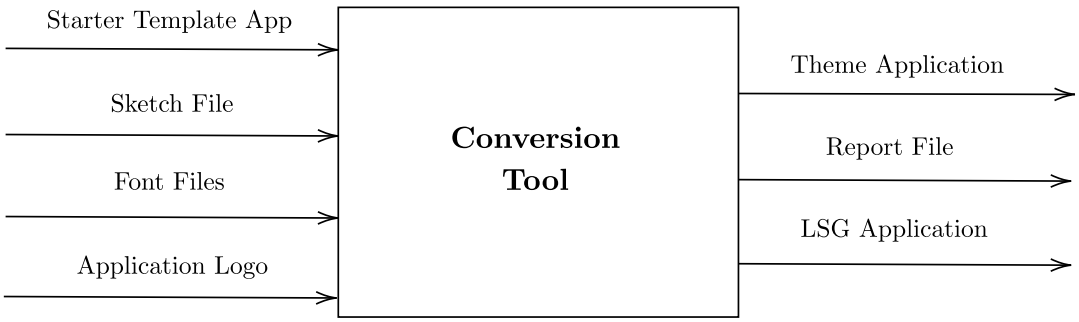

Figure 6.14: System Black-Box Model

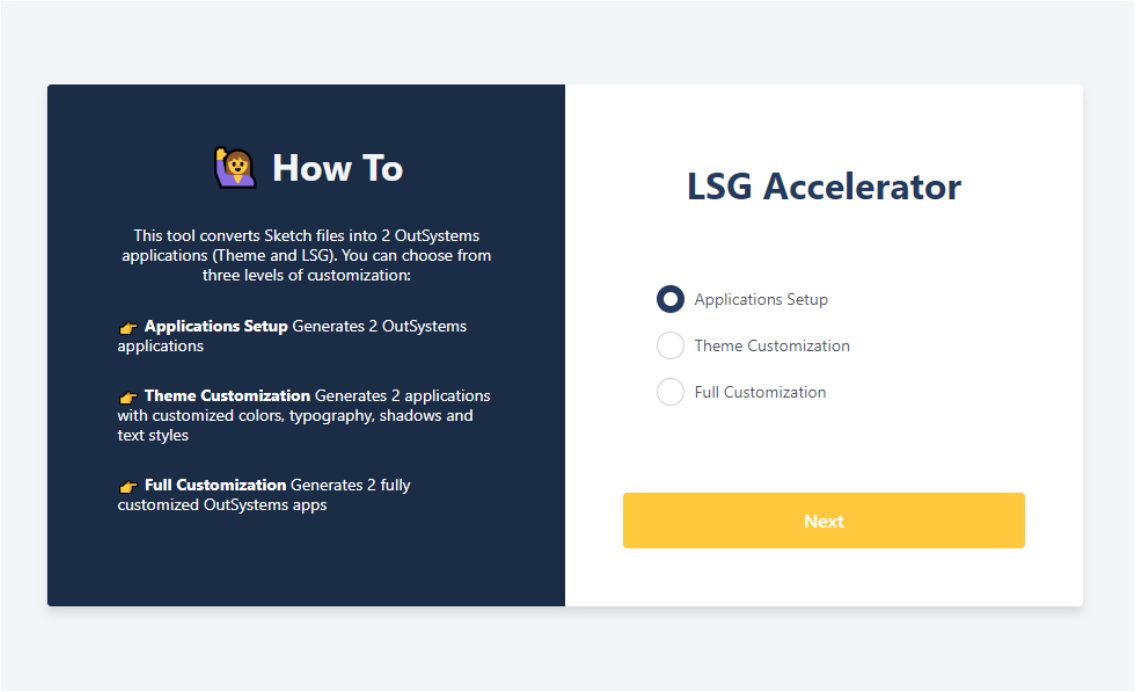

Figure 6.15: Tool Web Application Interface

The system inputs and outputs are illustrated in Figure 6.14. As inputs, our tool receives an OutSystems LSG Sketch file instance, .ttf or .woff files related to the applications fonts and finally an image representing the application logo. The outputs of our tool are two customized OutSystems applications that represent the LSG and a report file. The report file specifies which components were properly customized and which components require manual intervention.

Chapter 7

Implementation

In this chapter we present the technical details of our implementation process. We start by explaining how Sketch files are processed and how information is extracted. Next, we present the customization process and explain the methods used to extract information about styles and widgets. Finally, we describe how new instances of OutSystems applications are generated and how we customize them.

7.1 High Level Concepts

The high level concepts manipulated by our tool are shown in Figure 7.1. The architecture is divided into three main groups, Sketch, Intermediate and OutSystems.

After analyzing the Customer Success development process explained in section 3.3.1, we concluded that the flow of information in the OutSystems ecosystem occurs in one direction, from the design to web representation. For this reason, our tool performs a unidirectional model transformation from left to right in the diagram of Figure 7.1.

A Sketch file is given as input. The file is an instance of the LSG Sketch file which means that it overrides the default LSG Sketch file with the client branding. The difference between the two files (Customization Δ [1]) is identified and the Customization Δ calculated. The Customization Δ is later used to create the Intermediate Representation. Later, the representation is transformed and used to customize an instance of the OutSystems Live Style Guide.

The transformation process is exemplified in Figure 7.3. The Sketch representation of a Button Group is transformed into its intermediate representation that is finally transformed into the resulting OutSystems LSG Button Group widget. Figure 7.4 presents the default styling inherited from the OutSystems UI.

[1] Difference between the Sketch instance and the LSG template.

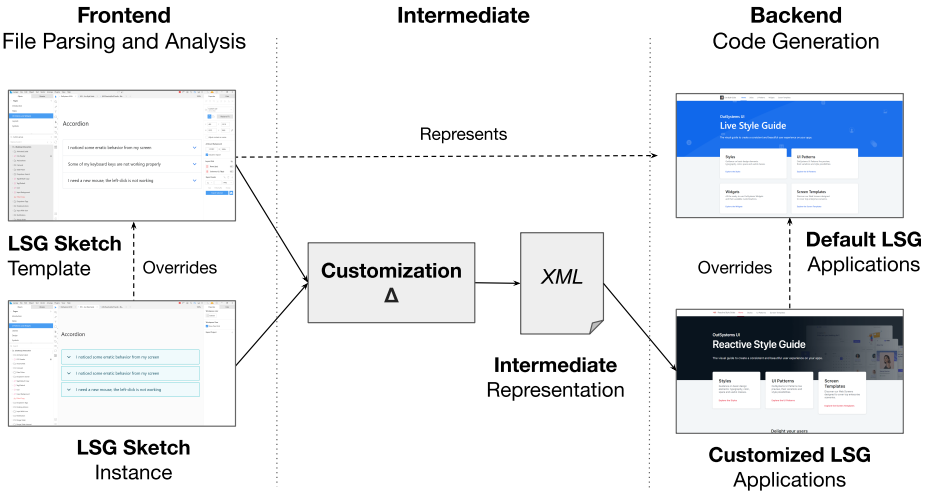

Figure 7.1: Tool High-Level Concepts

The tool functioning and the steps described previously are explained with detail in the following subsections.

7.2 Parsing and Analysis

Sketch documents are compressed archives containing JSON encoded data. For each Sketch page is generated a JSON file. Before being processed by our tool, the Sketch file is unzipped and JSON files extracted. After, we map the JSON files obtained to the page to which they correspond.

We start by segmenting each JSON file, identifying the elements needed for the conversion and confirm that every layer type is known - lexical analysis.

Then, for each element tree, we verify if it is syntactically correct. Different layer types have different properties and allow the existence of different type inner layers. In this stage, we analyse the compatibility between types for each layer and inner layers, making sure that the hierarchy of layer types is correct.

Lastly, we do a semantic analysis: where we confirm the concordance between the layer types and its attributes. Some layers reference other layers (e.g. symbol master) or shared styles properties. During this stage, when a layer references other layers or elements, the existence of the referenced elements is checked.

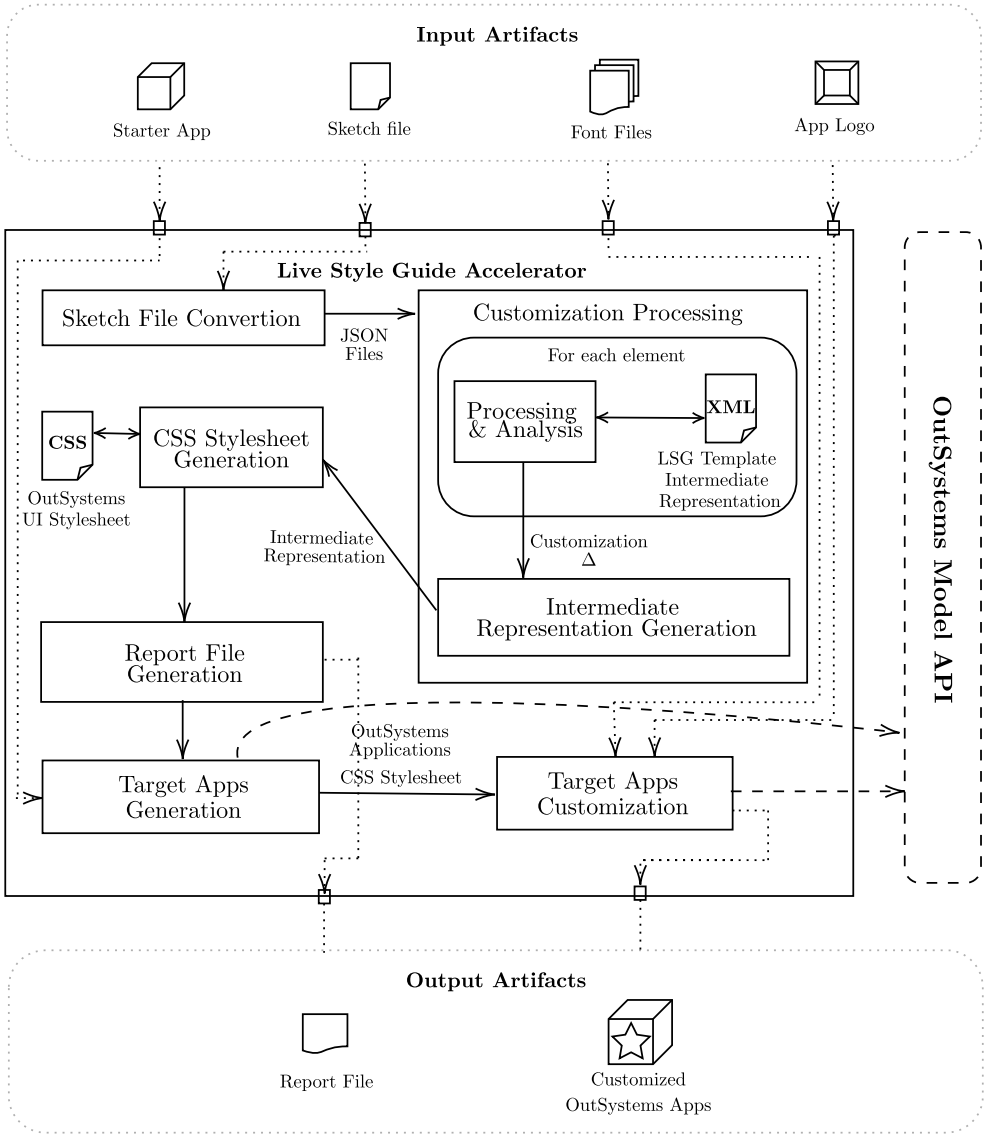

Figure 7.2: Tool Process Diagram

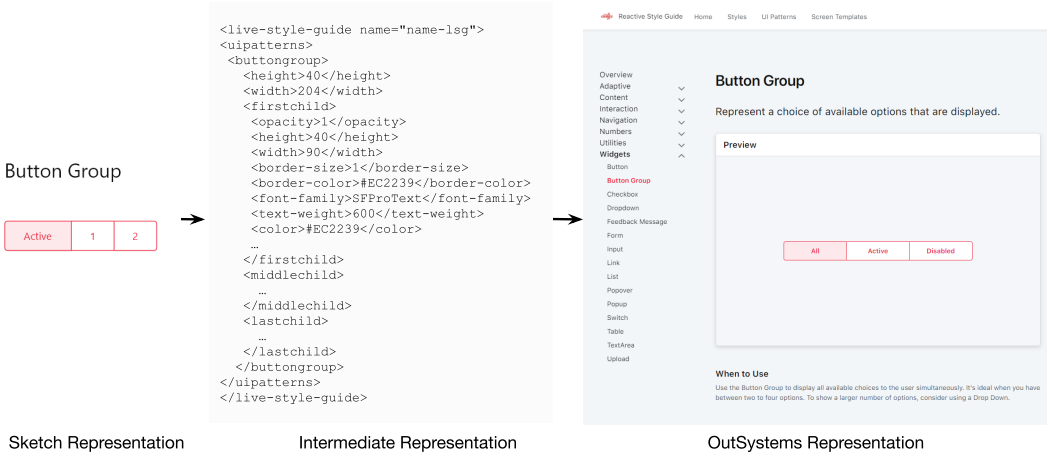

Figure 7.3: Transformation of the Button Group Component

Figure 7.4: OutSystems UI Button Group Widget

7.3 Customization Processing

The customization of OutSystems applications can be classified into four levels as explained in section 6.9. The first three levels of customization refer to the Customization step of the Live Style Guides creation process (see section 6.4). These are the steps covered by our work. Our customization processing is divided into two steps: Styles components customization and widgets and ui patterns customization.

7.3.1 Styles Components

The Styles Processing stage corresponds to the first level of customization and to the Styles page page components processing presented in section 6.7. Mapping styles concepts is straightforward since elements needed to process are all available and only 1 layer processing is required. No Symbol layers are used and consequently no extra processing and layer recognition tasks are needed. The design concepts match with the web technology concepts.

Styles concepts include Colors, States, Typography, Text Styles and Shadows. Elements position and label in the page hierarchy is enough to process nodes since models do not suffer changes to its structure.

Widgets and UI Patterns are represented in the corresponding Sketch page and distributed across multiple artboards. The structure illustrated in Figure 6.5 corresponds to LSG Sketch template version 1.0.0. However, different versions are organized differently. In order to accommodate those variations, processing is done through the groups of artboards and not by the organization of artboards.

7.3.2 Widgets and UI Patterns Components

The OutSystems UI framework has more than 70 widgets and UI Patterns. In order to test the viability of the approach, we restricted the group to the more relevant 10. The selection was made by the front-end experts team and included the most frequently changed elements. The group of widgets includes: accordion, alert, avatar, badge, button, button group, card, dropdown, menu, pagination, popover, table records, tabs, tag, wizard.

Components Identification

Unlikely the Styles components, when we process the Widgets and UI Patterns page, we do not rely on the artboards to find the groups of components. This approach was motivated by the different artboard organization of the OutSystems UI Kit file versions. Our approach skips the artboard level and uses the layer groups inside the artboards as a guide to identify component groups. For version to version, the groups are kept only when the artboard organization changes. This means that our tool can deal with structurally different formats as long as the semantic information is equivalent to the OutSystems UI.

Processing Constraints

Style elements are represented by a single layer. UI Patterns and Widgets, on the other hand, are represented in our source model as **trees of layers**. Processing such elements implies identifying, processing and mapping all their layers as well as the relationships between them.

Sketch trees can be created in multiple ways given the flexibility and freedom of language. However, some of the elements in the target model have no direct mapping in the source model. One example is the attribution of borders. Layer nodes contain an array of Borders that represent the layer borders. Borders are enabled for every side of the layer. It is not possible to define borders that do not surround all the sides of the layer. Other types of border customization, that do not surround all the sides of the layer, are not supported by the Sketch syntax. Designers usually introduce new layers resulting in a wide variety of trees for each widget/UI pattern. This technique makes it particularly difficult to identify layers and consequently have a uniform layer processing method. After customization, UI Patterns and Widgets trees may have no resemblance

to the original structure defined in the OutSystems Live Style Guide Sketch document or with the web technology structure. Due to the complexity of customizing a UI Pattern, currently, only a subset of UI patterns are being customized. For each UI Pattern, our tool has a tree collection of structures that can be processed, including the default tree for each widget defined in the LSG template.

Atomic Processing

Both the OutSystems UI framework and the OutSystems UI Kit for Sketch follow the **Atomic Design** [21] principles which is a design methodology that breaks down systems' UI elements into small components with different properties. Those elements are called atoms and are the smallest units of the system that keep their meaning. Atoms can be organized into groups called molecules or into even more complex structures like organisms. This design methodology promotes elements reusability and supports systems scalability. Following the same approach, our identification of components in Sketch also relies on Atomic Design.

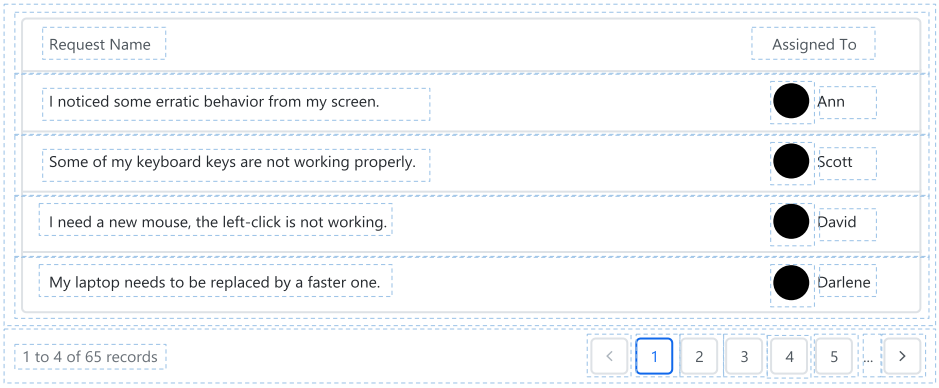

Figure 7.5: Table Records Widget Atomic Decomposition

To map elements represented by a group of layers, we use a granular approach based on the identification of micro-patterns. Instead of handling the element as a whole, we try to find matching atomic reusable elements within its components. This approach eases the tool evolution and improves the return of investment in doing so. If we processed the patterns as a whole and since the processing of all patterns is unreachable, there would be a large number of patterns that would not be customized. This way, we increase the effectiveness of the tool, since even though many patterns are not fully recognized, there is a large number that will be partially recognized and converted.

Table 7.1: Layer Attributes

Layer Type	Attributes
Symbol Instance	Override Values, Frame (Height and Weight, X, Y), Shadow
Text	Frame(X and Y), Line Height, Color, Font, Weight, Size, Shadow, Shared Style
Image	Fills (File name), Frame (height and weight, X, Y), Background
Shape	Frame (Height and Weight, X, Y), Shadow, Border, Background

Conversion Framework

Frequently, trees have Symbol Instance layers. Symbol Instances are references for Symbol layers defined in the Symbols page. The processing of such layers is done using a lazy approach. Every time a symbol is needed, it is processed and saved in memory for future use. In addition to Symbol Instance layers, trees have other node types namely shape, text, image and group layers. Each type of layer is processed differently, since the relevant elements of each layer vary as illustrated in Table 7.1.

When a tree or a subtree is not recognized, components are treated as Custom Patterns, that is, a pattern whose structure and nodes our tool is not able to identify and process and no intermediate representation is generated.

Currently, our tool is capable of transforming a subset of Widgets / UI Patterns identified by the OutSystems front-end experts team as being the most relevant and frequently used and customized. For each one, after identifying and processing micro-patterns, we have to establish the relation between them and create a new processed structure representing it. This work is done individually for each one since we need to know the base structure of the element.

7.4 Customization Delta Calculation

After the generation of the Intermediate Representation, we need to identify the difference between the Sketch instance and the LSG template. We call the difference, the **Customization** Δ.

Since the LSG Sketch template represents the OutSystems UI default theme, the changes made on top of this document represent the customization done. This calculation will separate the customization from the OutSystems UI base.

The Δ is identified by comparing the intermediate representation of the OutSystems LSG Sketch file with the intermediate representation of the customized instance. The

difference between the two documents is used to customize the applications.

7.5 Customization Delta to Target Model

As explained in section 7.3, OutSystems applications inherit the default OutSystems UI CSS styling rules. Customizing applications can be done through the OutSystems IDE interface or by adding new CSS declarations that override the default ones.

Since the elements present in the Sketch file represent the default OutSystems UI components, mapping the Δ elements to OutSystems implies mapping the only the customizations done to these elements to the corresponding CSS rules. During this stage, we compare the values declared in the OutSystems UI CSS style sheet for each rule with the corresponding Δ elements. The relationship between these components and the corresponding OutSystems UI Style sheet CSS Selectors had to be previously established in order to compare the values. CSS Declarations may be hierarchical and one CSS Selector may affect multiple Δ elements. For this reason, mapping is not straightforward since one CSS selector may correspond to multiple Δ elements. One example of this is the selector .btn that affects all button instances.

When the customization Δ does not contain the representation of a certain item, its structure is not recognized. In this case, a note is added to the report file mentioning that the component needs to be verified and customized manually.

In contrast, when a Δ element exists however its content is nonexistent, no differences were identified between the customized element and the corresponding OutSystems UI base after the Δ generation explained in section 7.4. In this case, the element is not compared with the corresponding CSS rules since no adjustments were made and the component only inherits the OutSystems UI styling. In the report file, the item is marked as reviewed.

Finally, if the customization Δ contains an element and a group of attributes that describe it, the corresponding OutSystems component needs to be customized. The attribute values are compared with the CSS rules values. New CSS selector and corresponding declarations are added to the style sheet when the values are different.A CSS style sheet is the resultant artifact of this stage.

7.6 Target Model Instance Generation

The last task performed by our tool is the generation of the target model instance, two customized LSG OutSystems applications. Outsystems applications are generated with the help of an OutSystems API that allows the manipulation of its model. The applications are created from a starter base application representing the default Live Style Guide and Theme applications. Applications and their modules are personalized with the customer name and the corresponding logo provided by the user. The CSS file generated in the

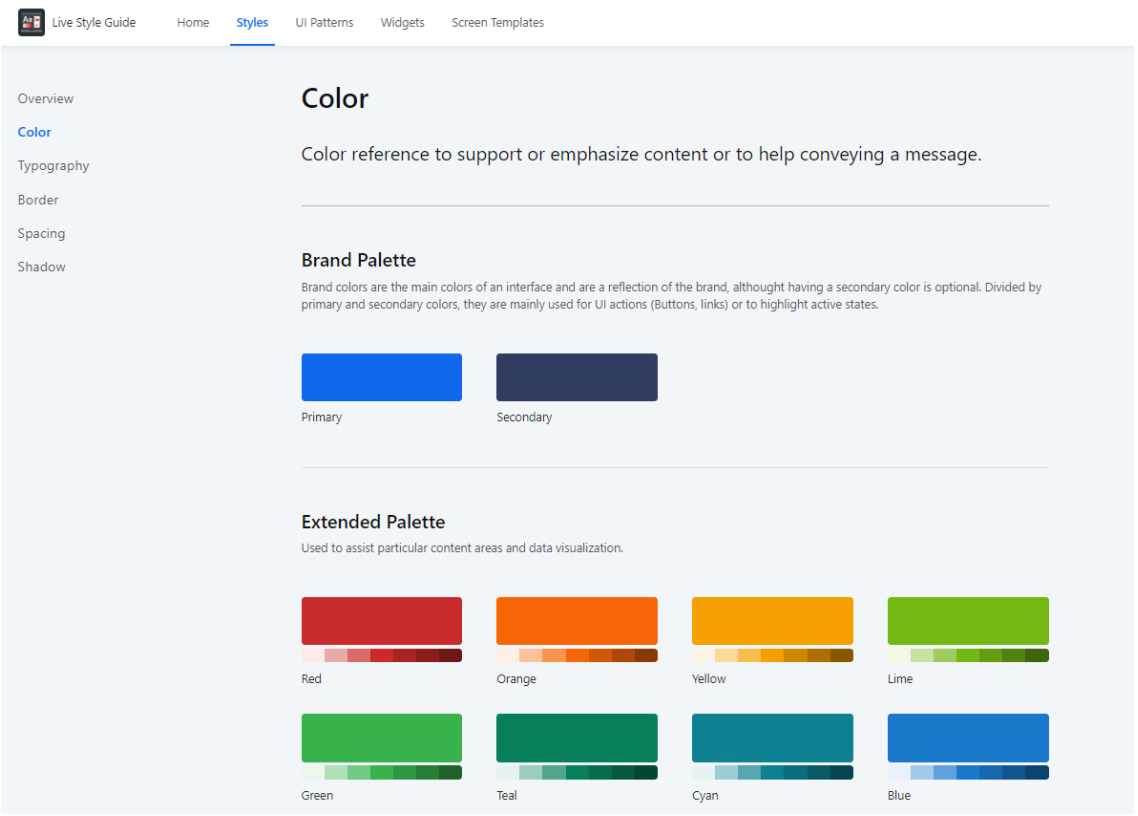

Figure 7.6: Sample OutSystems Live Style Guide

previous step is added to the Theme module of the Theme application overriding its default look-and-feel.

For the generated project to be correct, it is necessary that fonts non existing by default in the OutSystems IDE and declared in the Sketch file be supplied by the user. Those files are added as resources to the theme module of the Theme application. An example of a customized OutSystems LSG application is shown in Figure 7.6.

The OutSystems language has extensibility points that can be used to enrich applications. We can take advantage of those points to inject custom CSS into applications and complement the generated code.

If there are manual changes to the generated OutSystems code (e.g. for further tweaks or to accommodate a complex custom pattern) and there is a need to re-generate the code because of changes to the Sketch design, people need to be careful, otherwise they may lose the manual changes because of the newly generated code. The loss of manual customizations can be easily avoided if people leverage the merge capabilities of the OutSystems platform - these allow the user to cherry pick the relevant changes from both versions (the one with manual changes and the one resulting from the design file changes).

The default starter application is the *Reactive Starter App* and consequently the generated applications are also Reactive applications. Our tool enables the usage of different

Starter applications with a similar structure of the *Reactive Starter App*. However, our tool was not tested with other types of OutSystems applications. The main goal was to allow users to use different versions of the *Reactive Starter App* and eliminate dependence on the current version of the tool default app.

7.7 Report File

Alongside with the two applications, our tool generates a .txt report file. This file contains detailed information about the customization process of every element. As can be seen in the block below, for each element, beside its name, the report file includes one of the three different messages:

- **Customized** - when the element was fully processed and customized.
- **Needs Manual Intervention** - when the element was partially customized and customization needs to be manually completed.
- **Structure Not Recognized** - when the structure of the Sketch element was not recognized and the element was not customized.

```
############## REPORT FILE ###############
Fonts – CUSTOMIZED
---------- ROOT CUSTOMIZATION ----------
Typography Size – CUSTOMIZED
Colors Customization – CUSTOMIZED
Shadows – CUSTOMIZED
App Settings Customization – CUSTOMIZED
Font Family – CUSTOMIZED
---------- RESETS CUSTOMIZATION ----------
=== Buttons Customization ===
.btn – CUSTOMIZED
.btn:hover:active – NEEDS MANUAL INTERVENTION
.btn:hover:active – CUSTOMIZED
.desktop .btn:hover – CUSTOMIZED
Size Buttons – ERROR: STRUCTURE NOT RECOGNIZED
```

Listing 7.1: Report File Example

Chapter 8

Evaluation

This chapter presents the evaluation methods used to measure the quality of the results obtained with our tool and highlights the most relevant findings.

The results obtained show that for 10 real-world projects, the usage of our tool can save between 20% and 75% of the time invested in the creation of a LSG. Savings are highly dependent and inversely proportional to the project complexity.

We start by presenting an automated approach to compare CSS stylesheets and explain the reasons why it proved to be not viable to gather the results intended. Later, we introduce the two methods used to evaluate our approach:

- Interviews with the front-end experts to collect feedback about the results generated by our tool and compare it with manually created LSG previously developed by the team.

- The number of components effectively, partially and not customized for each project.

Finally, we discuss the results obtained and generalize gains to the OutSystems Customer Success department and customer base. We conclude that savings exceed 96 days for the Customer Success department and between 7431 and 11923 days per year if applied to the OutSystems customer base.

8.1 Automated Stylesheet Comparison

The majority of the customization work done by our tool is injected into the OutSystems applications as a CSS style sheet. Evaluating results presupposes comparing how similar the input and the output are and comparing the results obtained manually and with our tool.

Initially, we intended to compare in a quantitative and automated way the output of our tool with the manual result. We started by using tools to compress and clean both style sheets in order to ease comparison. However, the approach was not viable. The manual artifacts were developed by several experts with different styles making the comparison particularly difficult and uneven. Our goal was to compare not only how similar both stylesheets were but also measure the similarities in the application of the styles defined in those documents to elements. After researching mechanisms to compare both stylesheets, we did not find any viable approach to compare two CSS style sheets.

8.2 Front-end Experts Interviews

Evaluation Method

To evaluate the results of our approach, we made available a web application to the OutSystems Front-End Experts team. This team is composed of 11 professional developers, whose daily work is dedicated to dealing with complex front-end development topics. Part of their responsibility is materializing the UX/UI designs of enterprise-grade applications into hand-crafted Style Guides and Samples.

While developing the tool, we leveraged the artifacts (designs as well as resulting style guides and sample pages) of 10 recently completed enterprise projects. This made it possible to validate the correct behaviour of the tool in real-life scenarios. Furthermore, these past projects were great references against which to evaluate not only the tool effectiveness and impact but also the quality of the resulting code (e.g. compliance with established programming practices).

To evaluate the actual impact of our tool, we asked 5 OutSystems Front End Experts to evaluate and compare the results of their own work in the 10 past projects against the results obtained by using our tool.

We made the following questions to experts:

1. Have you ever used the tool before?
2. For each project:
    a) How many days did it take for the project to be done manually?
    b) At least, how many days do you think the result obtained with the tool could save?
    c) At most, how many days do you think the result obtained with the tool could save?
3. Do you imagine yourself using the tool in future projects?
4. Is there any functionally that should be improved/added?

Results

To better evaluate the results obtained, the projects analyzed were divided into three groups according to their complexity:

- **Low Complexity** projects take 4 days to complete. The customization process includes changing the styles of the applications namely colors, typography, shadows and text styles. These elements correspond to the variables of the project CSS Stylesheet. Usually, Widgets and UI Patterns do not need to be customized. Custom Patterns and Sample Pages are not created in low complexity projects.

- **Medium Complexity** projects have the standard duration of 5 days. The major difference between medium and low complexity projects is the creation of new custom patterns (some projects may also require the creation of sample pages). When comparing both groups of projects, the time invested in the different stages of the customization process is not proportional.

- **High Complexity** projects take 10 or more days to complete. These projects usually are from clients who already have their own design systems and want to create a Live Style Guide based on it. Therefore, quite often it is necessary to make changes to the structure of the OutSystems UI and the LSG module. These tasks dramatically increase the time needed to create a LSG. Apart from that, these projects comprise the creation of a large amount of custom patterns and sample pages.

The 10 real-world projects analyzed were composed by **3 low-complexity projects, 5 medium-complexity projects and 2 high-complexity projects**. The manual duration of the project and a minimum and maximum of days that the usage of the tool could save them as shown in Table 8.1.

Table 8.1: Projects Savings According Complexity

Project	Complexity	Manual Duration	Min Savings	Max Savings
1	Low	4.0	3.0	3.0
2	Low	4.0	3.0	3.5
3	Low	4.0	3.0	3.5
4	Medium	5.0	1.0	1.5
5	Medium	5.0	1.0	1.5
6	Medium	5.0	1.0	2.0
7	Medium	5.0	1.0	1.5
8	Medium	5.0	1.5	2.0
9	High	5.0	1.0	2.0
10	High	10.0	2.0	3.0

For **low-complexity projects**, in average, our tool saves between **3** and **3.5** days (66.6 to 83.3%). For **medium-complexity projects**, on average, it saves between **1** and 2 days (22 to 40%). Finally for **high-complexity projects**, savings range in average between **1.5**

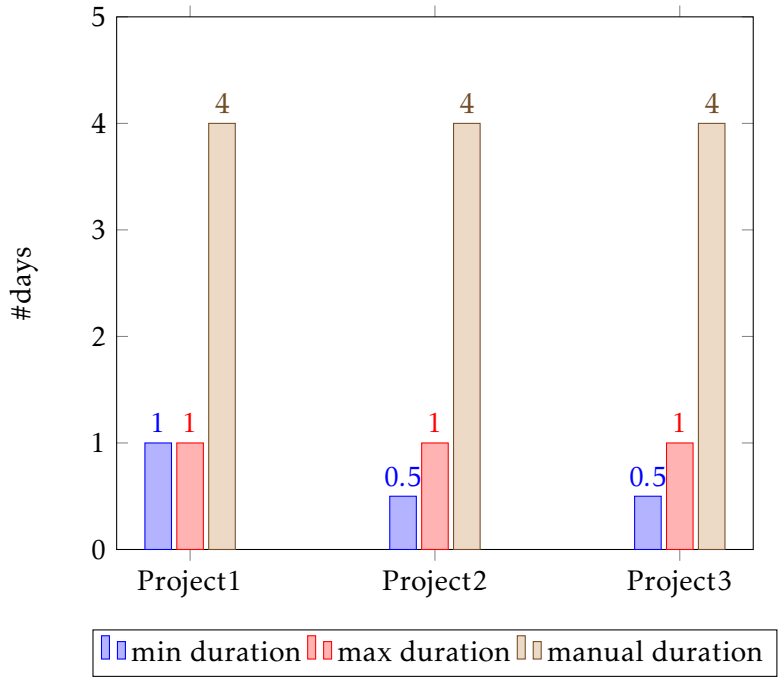

Figure 8.1: Low Complexity Projects Savings

and **2.5** days (17 to 33%). Experts referred that savings could be even higher once they have confidence and experience with the tool.

When asked about their intentions to use the tool in the future, the interview group unanimously said that they intend to continue using it in future projects. The tool reduces drastically the time needed to setup and customize applications. Consequently, front-end experts can invest the time saved in further added value tasks such as the validation process and sample pages creation.

The new functionalities that should be added to the tool mentioned by experts during interviews referred to features that would improve their workflow (e.g. feature to convert font files from .ttf to .woff or .woff2). While pertinent from a practical point of view, these requirements are not relevant in a research-only perspective.

8.3 Customization Ratio Calculation

Evaluation Method

In order to have a better idea of the automation level, we measured how many OutSystems components were automatically generated and how many components needed manual intervention. We used the 10 real-world projects provided by the Advanced Development team. The values were obtained taking advantage of the report file generated by our tool.

For the analysis, we considered as a component every instance of a Sketch element that had a corresponding CSS rule (e.g. the multiple states of a button were considered different components). The total number of components per project equals 50.

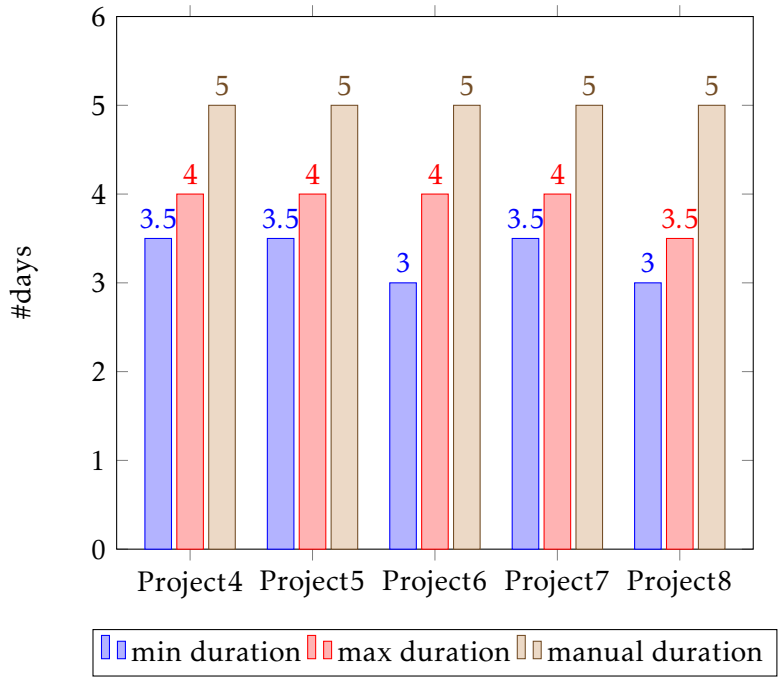

Figure 8.2: Medium Complexity Projects Savings

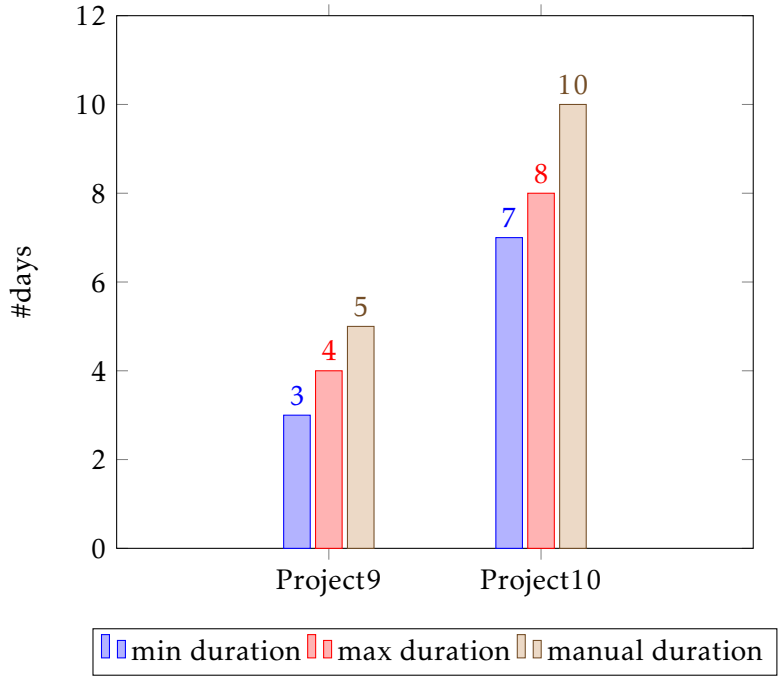

Figure 8.3: High Complexity Projects Savings

Results

Table 8.2 presents the number of components fully, partially and not customized for the 10 projects analyzed.

Table 8.2: Number of Components Customized

Project	Complexity	Fully Customized	Partially Customized	Not Customized
1	Low	43	1	6
2	Low	36	1	13
3	Low	43	2	5
4	Medium	43	1	6
5	Medium	42	3	5
6	Medium	43	2	5
7	Medium	42	3	5
8	Medium	42	3	5
9	High	42	2	6
10	High	35	7	8

8.4 Analysis

While the initial attempt at comparing style sheets in an automated and quantitative way proved to be non-viable , nevertheless it led to interesting insights. While comparing the manually vs automatically generated artifacts, we concluded that **our tool solves aspects that get overlooked in the manual process**. Our tool detects all variations of color, including those small enough that were actually not detected during the manual process. This is one of the challenges initially raised by the front-end experts while researching the friction points of the manual process.

Development savings are dependent on the design complexity. Based on feedback collected, conclusions point that for projects without custom patterns, the tool can reduce the time the Front End team needs to invest bootstrapping an application's customization by 3 out of 4 days. For intermediate complexity projects with custom patterns and sample pages whose duration is 5 days, we expect savings between 1.5 and 2.5 days. Finally, for projects with higher customization requirements, the conservative expectation is for savings of at least 1.5 out of 10 days.

While the results are still qualitative in nature, we believe the informed opinion of a group of domain experts is invaluable, especially considering the complex and non-linear nature of the process through which UX/UI designs get transformed into live style guides.

The OutSystems Customer Success department does 62 projects per year. From a sample of projects developed by the team, we concluded that the ratio between high, medium and low-complexity projects is 18:9:4. Consequently, we can conclude that the number of high, medium and low-complexity projects developed by the team yearly are 36, 18 and 8 respectively. Then, we can extrapolate that by using the approach developed with this dissertation, the weighted average of annual savings around 122 days per year.

Considering that year over year, the volume of projects is only increasing, the gains of using such a tool are really promising.

When applied to the OutSystems' ecosystem client base that contains more than 1200 companies, the usage of our tool represents savings between 7431 and 11923 days (24,5 to 39%) - considering that each customer develops 4 projects per year with the standard duration of 3 months each.

Notwithstanding the qualitative nature of the evaluation process, we believe that the results collected - 20 to 75% savings in time invested by highly specialized and experienced professionals - are very strong indicators that the approach followed with this work is a good fit for dealing with the problems we proposed to tackle initially.

Chapter 9

Conclusions

Our work is motivated by the currently existing gap in efficiency and effectiveness when it comes to UX and UI designers collaborating with front-end developers. This is a relevant problem today because good UX/UI is an essential market differentiator for most companies while hiring skilled UX, UI, and front-end professionals continues to be a particularly challenging topic.

To mitigate the above challenge, with this dissertation, we presented an approach to automate the conversion of high-fidelity design artifacts into low-code web-technology reusable UI components and applications. The conversion process is based on a unidirectional model transformation and an intermediate representation independent from both technologies.

The biggest challenge was identifying the source model elements since they are context-dependent. Mapping of design to web concepts is not a straightforward process, where each element needs to be identified and processed independently. To overcome the challenge, our solution relies on an instance of a design technology meta-model with a well-defined structure as our source model and a dataset of customized components to identify and process components.

According to the results obtained, the usage of the developed tool represents a great improvement in the Live Style Guide creation process. Depending on the degree of UX and UI customization, professional teams can save between 20 to 75% of the time invested in creating Live Style Guides. This implies that with the adoption of our tool the OutSystems Customer Success department can reduce the time needed to develop projects around 122 days per year. If applied at scale to the OutSystems customer base, at the end of a year, these savings can translate into up to 11923 days - assuming a customer base exceeding 1200 companies that develop 4 projects per year, lasting 3 months each. The effort saved can be invested in other tasks that represent greater value to the customer.

The results obtained highlight the great impact of our tool and the appliance of the used techniques to bridge design and web technologies.

The design of the tool developed in this dissertation was closely followed by experts and stakeholders who started using it even before its development process was complete. The tool has been adopted by two teams of experts from different OutSystems departments and is currently part of their daily workflow.

The work done during this dissertation motivated the writing of an article, which has been accepted for publication, for the Modeling in Low-Code Development Platforms Workshop of the ACM / IEEE 23rd International Conference on Model Driven Engineering Languages and Systems titled "*Closing the Gap Between Designers and Developers in a Low-Code Ecosystem*" [6]. The publication focuses on the collaboration process between groups and explains the tool developed under the topic of Model-Driven Engineering.

9.1 Future Work

Future work can be categorized into engineering and research topics. From the engineering perspective, future work includes widening the set of supported widgets and UI patterns within the identified categories.

An interesting topic is the introduction of new design tools such as Figma or InVision. Alongside with Sketch, these tools are also used by the UI Design experts including external design teams. Early research showed that Figma has a similar model to Sketch and consequently extending the tool to this design technology can be done following a similar approach to Sketch.

In terms of research, an interesting opportunity to explore is the conversion of whole application pages from their design representation to a web technology. A particular challenge here is safekeeping the referential identity of the components being reused in such compositions. Addressing this challenge may imply changes to the workflow of designers. So, achieving a sustainable balance of efficient workflow for frontend professionals, while not shifting the inefficiency to the design side should be a concern.

Finally, the current implementation relies on the OutSystems UI framework concepts. Future research could examine the viability of decoupling to the concepts of OutSystems UI framework.

Bibliography

[1] M. Abrams, C. Phanouriou Ł, A. L. Batongbacal, S. M. Williams, and J. E. Shuster. *UIML: an appliance-independent XML user interface language*. Tech. rep. 1999.

[2] R. B. Andrew Hogan, Deanna Laufer with David Truog, William Willsea. *The Six Steps For Justifying Better UX*. Tech. rep. 2016.

[3] P. Badke-Schaub, A. Neumann, K. Lauche, and S. Mohammed. "Mental models in design teams: a valid approach to performance in design collaboration?" In: *CoDesign* 3.1 (2007), pp. 5–20. ISSN: 1571-0882. DOI: 10.1080/15710880601170768.

[4] J. Belenguer, J. Parra, I. Torres, and P. J Molina. "HCI Designers and Engineers: It is possible to work together?" In: *Closing The Gaps: Software Engineering and Human-Computer Interaction* (2003).

[5] T. Beltramelli. "pix2code: Generating code from a graphical user interface screenshot." In: *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2018* (2018), pp. 1–9. DOI: 10.1145/3220134.3220135. arXiv: 1705.07962.

[6] M. Bexiga, J. C. Seco, and S. Garbatov. "Closing the Gap Between Designers and Developers in a Low-Code Ecosystem." In: *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion). Workshop on LowCode: Low-Code Development Platforms*. 2020.

[7] A. Bilczynska Wojcik. "Communication management within virtual teams in global projects." Doctoral dissertation. Dublin Business School, 2014.

[8] M. Brambilla and P. Fraternali. *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*. Dec. 2014. ISBN: 978-0128001080.

[9] M. Brambilla and P. Fraternali. "Implementation of applications specified with IFML." In: *Interaction Flow Modeling Language* February (2015), pp. 279–334. DOI: 10.1016/b978-0-12-800108-0.00010-2.

[10] M. Brambilla, A. Mauri, and E. Umuhoza. "Extending the Interaction Flow Modeling Language (IFML) for model driven development of mobile applications front end." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8640 LNCS.August (2014), pp. 176–191. ISSN: 16113349. DOI: 10.1007/978-3-319-10359-4_15.

[11] J. Brown, G. Lindgaard, and R. Biddle. "Stories, sketches, and lists: Developers and interaction designers interacting through artefacts." In: *Proceedings - Agile 2008 Conference* (2008), pp. 39–50. DOI: 10.1109/Agile.2008.54.

[12] A. Bruun, M. K. Larusdottir, L. Nielsen, P. A. Nielsen, and J. S. Persson. "The Role of UX Professionals in Agile Development: A Case Study from Industry." In: *Proceedings of the 10th Nordic Conference on Human-Computer Interaction*. NordiCHI '18. Oslo, Norway: Association for Computing Machinery, 2018, 352–363. ISBN: 9781450364379. DOI: 10.1145/3240167.3240213. URL: https://doi.org/10.1145/3240167.3240213.

[13] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu. "From UI design image to GUI skeleton: A neural machine translator to bootstrap mobile GUI implementation." In: *Proceedings - International Conference on Software Engineering* 6 (2018), pp. 665–676. ISSN: 02705257. DOI: 10.1145/3180155.3180240.

[14] M. L. Chiu. "An organizational view of design communication in design collaboration." In: *Design Studies* 23.2 (2002), pp. 187–210. ISSN: 0142694X. DOI: 10.1016/S0142-694X(01)00019-9.

[15] C. Condo, J. Wise, and B. Seguin. *Adopt Product Management To Connect Design And Development*. Tech. rep. 2019, p. 9.

[16] A. R. Da Silva. "Model-driven engineering: A survey supported by the unified conceptual model." In: *Computer Languages, Systems & Structures* 43 (2015), pp. 139–155.

[17] P. P. Da Silva and N. W. Paton. "User interface modeling in UMLi." In: *IEEE software* 20.4 (2003), pp. 62–69.

[18] T. S. Da Silva, M. S. Silveira, F. Maurer, and F. F. Silveira. "The evolution of agile UXD." In: *Information and Software Technology* (2018). ISSN: 09505849. DOI: 10.1016/j.infsof.2018.04.008.

[19] J. Ferreira, H. Sharp, and H. Robinson. "Agile Development and User Experience Design Integration as an Ongoing Achievement in Practice." In: *Agile Conference*. 2012, pp. 11–20.

[20] D. Fox, J. Sillito, and F. Maurer. "Agile Methods and User-Centered Design: How These Two Methodologies are Being Successfully Integrated in Industry." In: *Agile 2008 Conference*. 2008, pp. 63–72.

[21] B. Frost. *Atomic design*. Brad Frost Pittsburgh, 2016.

[22] M. Hamdani, W. H. Butt, M. W. Anwar, and F. Azam. "A Systematic Literature Review on Interaction Flow Modeling Language (IFML)." In: *Proceedings of the 2018 2nd International Conference on Management Engineering, Software Engineering and*

Service Sciences. ICMSS 2018. Wuhan, China: Association for Computing Machinery, 2018, 134–138. ISBN: 9781450354318. DOI: 10.1145/3180374.3181333. URL: https://doi.org/10.1145/3180374.3181333.

[23] V. Jain, P. Agrawal, S. Banga, R. Kapoor, and S. Gulyani. "Sketch2Code: Transformation of Sketches to UI in Real-time Using Deep Neural Network." In: *ArXiv* abs/1910.08930 (2019).

[24] M. O. Jim Scheibmeir, Mark Driver, Thomas Murphy, Jason Wong. *Gartner Cool Vendors in Application Development and Platforms*. 2019.

[25] T. Kadlec. *Implementing Responsive Design: Building Sites for an Anywhere, Everywhere Web*. 2013, pp. 1–6. ISBN: 9780321821683. URL: http://glensalmon.com/img/ResponsiveDesign-anintroduction.pdf{\%}5Cnpapers3://publication/uuid/3BB392F8-889E-44E0-8192-F169E880BD34.

[26] J.-g. Ki and K.-y. Kwon. *Detection of GUI Elements on Sketch Images Using Object Detector Based on Deep Neural Networks*, pp. 86–90. ISBN: 978-981-13-0310-4. DOI: 10.1007/978-981-13-0311-1. URL: http://link.springer.com/10.1007/978-981-13-0311-1.

[27] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and V. López-Jaquero. "USIXML: A Language Supporting Multi-path Development of User Interfaces Engineering Human Computer Interaction and Interactive Systems." In: *Lecture Notes in Computer Science* 3425 (Jan. 2005), pp. 134–135.

[28] T. Lindberg, C. Meinel, and R. Wagner. "Design Thinking: A Fruitful Concept for IT Development?" In: (). DOI: 10.1007/978-3-642-13757-0.

[29] Y. Liu, Q. Hu, and K. Shu. "Improving pix2code based Bi-directional LSTM." In: *2018 IEEE International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*. IEEE, Nov. 2018. DOI: 10.1109/auteee.2018.8720784. URL: https://doi.org/10.1109/auteee.2018.8720784.

[30] E. Marcotte. *Responsive Web Design, Second Edition*. 2014, pp. 1–153. ISBN: 9781937557195.

[31] N. Maudet, G. Leiva, M. Beaudouin-Lafon, and W. E. Mackay. "Design breakdowns: Designer-developer gaps in representing and interpreting interactive systems." In: *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW* (2017), pp. 630–641. DOI: 10.1145/2998181.2998190.

[32] T. Mens and P. Van Gorp. "A taxonomy of model transformation." In: *Electronic Notes in Theoretical Computer Science* (2006). ISSN: 15710661. DOI: 10.1016/j.entcs.2005.10.021.

[33] T. A. Nguyen and C. Csallner. "Reverse engineering mobile application user interfaces with REMAUI." In: *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015* (2016), pp. 248–259. DOI: 10.1109/ASE.2015.32.

[34] N. Norouzi, M. Shabak, M. R. B. Embi, and T. H. Khan. "The Architect, the Client and Effective Communication in Architectural Design Practice." In: *Procedia - Social and Behavioral Sciences* 172 (2015), pp. 635–642. ISSN: 18770428. DOI: 10.1016/j.sbspro.2015.01.413. URL: http://dx.doi.org/10.1016/j.sbspro.2015.01.413.

[35] J. Patton. "Hitting the target: Adding interaction design to agile software development." In: *OOPSLA 2002 - ACM SIGPLAN Object Oriented Programming Systems Languages and Applications Conference - Practitioners Reports* (2002). DOI: 10.1145/604251.604255.

[36] A. Puerta and J. Eisenstein. *XIML: A Universal Language for User Interfaces User interface languages, model-based systems, user-interface management systems, interface models INTRODUCTION*. Tech. rep. 2001.

[37] R. G. Reddy. *User Interface Modeling*. 2011.

[38] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[39] J. R. Rymer, C. Mines, A. Vizgaitis, and A. Reese. *The Forrester Wave ™ : Low-Code Development Platforms For AD & D Pros , Q4 2017*. 2017.

[40] S. Sendall and W. Kozaczynski. "Model transformation: The heart and soul of model-driven software development." In: *IEEE Software* 20.5 (2003), pp. 42–45. ISSN: 07407459. DOI: 10.1109/MS.2003.1231150.

[41] T. Silva da Silva, A. Martin, F. Maurer, and M. Silveira. "User-Centered Design and Agile Methods: A Systematic Review." In: *Agile Conference*. 2011, pp. 77–86.

[42] N. Souchon and J. Vanderdonckt. "A review of XML-compliant user interface description languages." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2844 (2003), pp. 377–391. ISSN: 03029743. DOI: 10.1007/978-3-540-39929-2_26.

[43] B. Stewart. *Hype Cycle for User Experience, 2019*. Tech. rep. August. 2019.

[44] C. Vetterli, W. Brenner, F. Uebernickel, and C. Petrie. "From palaces to yurts: Why requirements engineering needs design thinking." In: *IEEE Internet Computing* 17.2 (2013), pp. 91–94. ISSN: 10897801. DOI: 10.1109/MIC.2013.32.

Webography

[45] *Adobe Illustrator*. URL: https://www.adobe.com/products/illustrator.html (visited on 01/20/2020).

[46] *Adobe Photoshop*. URL: https://www.adobe.com/products/photoshop.html (visited on 01/20/2020).

[47] *Adobe XD*. URL: https://www.adobe.com/products/xd.html (visited on 01/20/2020).

[48] *Affinity – Professional Creative Software*. URL: https://affinity.serif.com/en-us/ (visited on 02/06/2020).

[49] *Anima ToolKit - Sketch to HTML*. URL: https://www.animaapp.com/ (visited on 01/12/2020).

[50] *App Maker | Google Developers*. URL: https://developers.google.com/appmaker (visited on 02/19/2020).

[51] *Avocode*. URL: https://avocode.com/ (visited on 01/20/2020).

[52] *Axure RP 9 - Prototypes, Specifications, and Diagrams in One Tool*. URL: https://www.axure.com/ (visited on 02/06/2020).

[53] *Balsamiq. Rapid, effective and fun wireframing software. | Balsamiq*. URL: https://balsamiq.com/ (visited on 02/06/2020).

[54] M. Brhel, H. Meth, A. Maedche, and K. Werder. *Exploring principles of user-centered agile software development: A literature review*. 2015. DOI: 10.1016/j.infsof.2015.01.004.

[55] *draw.io*. URL: https://www.draw.io/ (visited on 02/06/2020).

[56] *Figma*. URL: https://www.figma.com/ (visited on 01/20/2020).

[57] *FluidUI.com - Create Web and Mobile Prototypes in Minutes*. URL: https://www.fluidui.com/ (visited on 02/06/2020).

[58] *Framer - A lightning fast interactive design tool.* URL: https://www.framer.com/ (visited on 02/06/2020).

[59] *Hi Interactive | User-centred experiences*. URL: https://www.hi-interactive.com/ (visited on 02/18/2020).

[60] *How OutSystems solves the problem | Evaluation Guide | OutSystems*. URL: https://www.outsystems.com/evaluation-guide/how-outsystems-solves-the-problem/ (visited on 01/13/2020).

[61] *Image to HTML converter. Create website in minutes | Fronty*. URL: https://fronty.com/ (visited on 01/12/2020).

[62] *Indigo.Design | Unified Platform for UX/UI Designers and Developers | Infragistics*. URL: https://www.infragistics.com/products/indigo-design (visited on 02/06/2020).

[63] *Integration with Design Tools*. 2019. URL: https://docs.kony.com/konylibrary/visualizer/visualizer\%user\%guide/Content/Integration\%with\%Design\%Tools.htm (visited on 01/12/2020).

[64] *InVision*. URL: https://www.invisionapp.com/ (visited on 01/20/2020).

[65] *Kissflow - Digital Workplace*. URL: https://kissflow.com/ (visited on 02/19/2020).

[66] *Kony. Accelerate digital success.* URL: https://www.kony.com/ (visited on 02/05/2020).

[67] A. S. Lee. *Generating Webpages from Screenshots*.

[68] *Low-Code Development Platform for Enterprise Applications | OutSystems*. URL: https://www.outsystems.com/platform/ (visited on 01/13/2020).

[69] *Mediaweb | Agile the Modern Web*. URL: https://mediaweb.pt/ (visited on 02/18/2020).

[70] *Mobify Documentation Home*. URL: https://docs.mobify.com/ (visited on 01/12/2020).

[71] *Mockplus*. URL: https://www.mockplus.com/ (visited on 01/20/2020).

[72] *Modular Programming - OutSystems*. URL: https://www.outsystems.com/learn/lesson/2159/modular-programming (visited on 10/09/2020).

[73] *Online app maker | Create custom apps for your business - Zoho Creator*. URL: https://www.zoho.com/creator/ (visited on 02/19/2020).

[74] *Online Diagram Software & Visual Solution | Lucidchart*. URL: https://www.lucidchart.com/pages/ (visited on 02/06/2020).

[75] *Online Mockup, Wireframe & UI Prototyping Tool · Moqups*. URL: https://moqups.com/ (visited on 02/06/2020).

[76] *OutSystems UI*. URL: https://outsystemsui.outsystems.com/OutsystemsUiWebsite/ (visited on 01/15/2020).

[77] *PaintCode - Turn your drawings into Objective-C or Swift drawing code*. URL: https://www.paintcodeapp.com/ (visited on 01/12/2020).

[78] *Pidoco - Online Wireframe and UX Prototyping Tool*. URL: https://pidoco.com/en (visited on 02/06/2020).

[79] *Progress Unite UX*. URL: https://www.progress.com/unite-ux (visited on 01/13/2020).

[80] *Proto.io - Prototypes that feel real*. URL: https://proto.io/ (visited on 02/06/2020).

[81] *ProtoPie - Interactive prototyping tool for all digital products*. URL: https://www.protopie.io/ (visited on 02/06/2020).

[82] *Quick Base: The World's Most Loved Low-code Platform*. URL: https://www.quickbase.com/ (visited on 02/19/2020).

[83] *React Studio*. URL: https://reactstudio.com/ (visited on 01/13/2020).

[84] *Responsive web design tool, CMS, and hosting platform | Webflow*. URL: https://webflow.com/ (visited on 02/06/2020).

[85] *Reuse and Refactor - OutSystems*. URL: https://success.outsystems.com/Documentation/11/Developing_an_Application/Reuse_and_Refactor/Libraries (visited on 10/09/2020).

[86] *RICE Scoring Model | Prioritization Method Overview*. URL: https://www.productplan.com/glossary/rice-scoring-model/ (visited on 01/27/2020).

[87] *Salesforce Lightning Design System*. URL: https://www.lightningdesignsystem.com/ (visited on 01/12/2020).

[88] *ServiceNow Design System*. URL: http://designsystem.servicenow.com/{\#}!/resources/platform-ui-assets (visited on 01/12/2020).

[89] *Sketch*. URL: https://www.sketch.com/ (visited on 01/20/2020).

[90] *Sketching Interfaces – Airbnb Design*. URL: https://airbnb.design/sketching-interfaces/ (visited on 01/16/2020).

[91] *Supernova Studio | The World's First Design to Code Platform*. URL: https://supernova.io/ (visited on 01/13/2020).

[92] *TrackVia | The Most-Trusted Application Building Platform*. URL: https://trackvia.com/ (visited on 02/19/2020).

[93] *Understanding Low Code Development and Low Code Platforms*. URL: https://www.c-sharpcorner.com/article/what-is-low-code-development/ (visited on 02/11/2020).

[94] *User Experience Will Affect Google Rankings Starting Next Year*. URL: https://www.inc.com/peter-roesler/user-experience-will-affect-google-rankings-starting-next-year.html (visited on 10/09/2020).

[95] *UXPin | UI Design and Prototyping Tool*. URL: https://www.uxpin.com/ (visited on 02/06/2020).

[96] *What is the framework for innovation? Design Council's evolved Double Diamond | Design Council*. URL: https://www.designcouncil.org.uk/news-opinion/what-framework-innovation-design-councils-evolved-double-diamond (visited on 01/27/2020).

[97] *Workflow Automation Software Simplified - Nintex Workflow*. URL: https://www.nintex.com/workflow-automation/advanced-workflow/ (visited on 02/19/2020).

[98] *Yotako*. URL: https://www.yotako.io/ (visited on 01/13/2020).

[99] *Zecoda Turn Designs Into Code Automatically*. URL: https://zecoda.com/ (visited on 01/12/2020).

[100] *Zeplin*. URL: https://zeplin.io/ (visited on 01/20/2020).