MASTER THESIS

From Design to Code: A Study on Generating Production Code From User Interface Design Software

---

Albin Frick

June 17, 2021

Master Thesis in Interaction Technology and Design, 30 ECTS
Master of Science in Interaction Technology and Design, 300 ECTS
UMEÅ UNIVERSITY

Abstract

Creating modern web applications is often done with a modular mindset using components (such as buttons, lists, layouts). These components are beneficial and do not need to be redesigned between projects if the design language is the same. However, if there is a need to have components with different design languages between each project, creating components can feel unnecessary. In collaboration with Knowit Experience, this thesis tries to solve this problem by creating an open-source prototype that converts components from the user interface design program Figma into web components. The thesis shows how to make this prototype user-friendly by usability testing. In addition to this, the thesis shows how to measure the effectiveness of this prototype with A/B testing. The results of the thesis are a working open-source prototype that is user-tested for all essential features.

Keywords— Web components, Figma, Web development, UI/UX, Code Generation

Contents

Glossary

component A set of fundamental elements arranged in a distinct layout . 14

Acknowledgments

Throughout the writing of this thesis, I have received a lot of support and assistance.

First of all, I would like to thank my supervisor, Mattias Andersson, whose expertise has been invaluable throughout the whole writing process. Your patience and insightful feedback brought my work to a higher level.

I would also like to thank the employees of Knowit Experience Norrland, who have enabled me to write this thesis. Thank you for your warm welcome, answers to all my questions, and high availability.

I would like to thank my fellow students and colleagues at the University of Umeå, who have given me valuable feedback for the thesis and the usability test done.

Last but not least, I would like to thank my fantastic parents, wonderful sisters, and lovely fiance. Without you, I would not have been able to complete my thesis. Thank you for your immense patient, feedback, and support.

1 Introduction

When building applications for companies, the workflow can become complex and hard to manage. Most often, the projects are carried out with an agile work form[1]. Agile work form means that the product is evolving, with a tight connection with the customer. Before development starts, some research needs to be done to find what the customer wants, how the application should be implemented, and how it should appear. When working on a big project in general, the design and the implementation of the product are done by different competence groups, mainly designers and developers.

Communication between designers and developers is essential to creating a good user experience (UX). The communication problem between designers and developers can be compared to the typical *impedance mismatch problem* in the data storage realm. Designers and developers are trying to achieve the same thing but see information differently. The front-end developer is trying to figure out how the design should be written in code. How the design elements should or should not affect each other, how each of them should be positioned, and so forth. The designers see the relationship between the elements, how the dynamic of contrast, colors, and white space is set for the whole application.

In later years most development for web applications is modular and built up by component. Components are design and/or functional elements in the application. A component could be a button, a navigation bar, a card[2], etc. Making components is done to save code, meaning that all components can be reused throughout a UI and between UI's. Many companies create extensive libraries of these components to be used within different projects. Using components libraries works well when the design language is the same for all projects. However, when a consulting firm's clients often do not have the same design language. The differences in design result in that the consulting firms often need to redesign their components between projects.

1.1 The Company & the Problem

Knowit is an IT company with about 2600 employees in Sweden, Norway, Denmark, Finland, and Germany. Knowit has three main branches, Experience, Insight, and Solutions. These branches handle different types of problems, enforce user experience with the customers brand (Experience), create system solutions to help customers digitalize (Solutions), and management-consulting (Insight). This project was carried out under the Experience branch as an important building block to find a more effective way to initiate new website building projects.

This branch is a consulting firm for customers that need digital tools with high availability, accessibility, and good user experience. Teams, including designers and developers, run every project under the Knowit experience branch. The designers make the design for the applications, including general designing guidelines, for each project, then the developers implement the design to a

functioning application.

Knowit uses an agile work form derived from the manifesto for Agile Software Development [3]. This is an iterative work form where the product is continuously improved with tests, analysis, and customer input. A simplified visualization of this work form can be seen in figure 1.

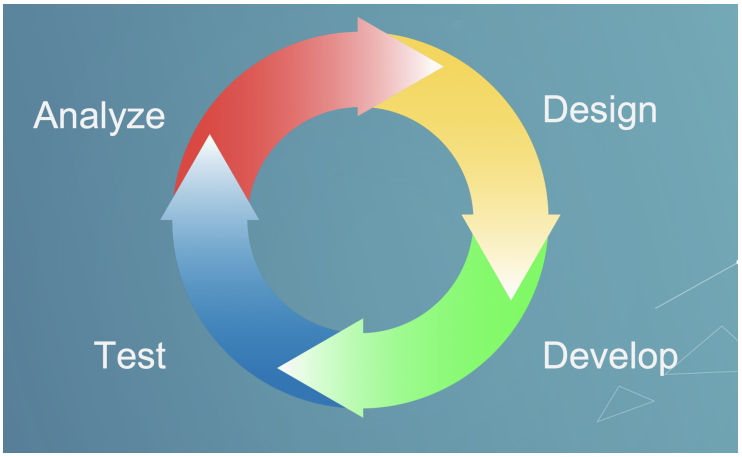

Figure 1: Simplified illustration of the agile workflow for Kowit.

The problem that has arisen is that the setup is similar for each project. The setup consists of setting up a color palette, typography, and fundamental components such as buttons, forms, etc. When the designer has done this, the developer needs to convert this to code. This process is often similar for each project but needs to be redone because most projects do not use the same frameworks and tools. This thesis is therefore focused on the handover from design to development, the most right arrow in figure 1.

1.1.1 Knowit Initial Requirements

The initial idea from Knowit was to make a design system[4] that followed the following requirements:

- The system needs to be applicable for all types of projects.
- The system needs to be modular. The choice to use parts of it must be an option.
- It has to be easy to change global parameters such as colors, fonts, margins, etc.
- The system must be open source.
- The system needs thorough documentation.

After some research, we concluded that there would not be enough time to administrate the design system after this project. Therefore the focus of the thesis was changed from making a design system to streamline the creation of components. This change was significant, but the before-mentioned requirements still stood for the new idea.

1.2 Objective

Most applications are built by components such as buttons, forms, cards[2], etc. These components often need to be redesigned and rebuilt from project to project, making it bothersome but necessary work.

This study aims to make this setup time extensively more efficient with a tool that generates components.

To reach the aim of the study, we need to answer the five questions stated below:

- Is it possible to automate the whole process from UI design program to browser runnable code?
- Can components be built that works for all JavaScript frameworks and static pages?
- How can a user-friendly tool be built that automates component generation?
- Will automation between design and development increase communication between designers and developers?
- Does this tool speed up the development process, and if so, how much?

1.3 Demarcations

This automation could be used to create a whole component library and be a part of a whole design system. This project was done under a limited time of 20 weeks. Therefore only a fundamental component generator will be produced. With these fundamental components, the only things that need to be considered are shape, color, typography, and layout. More advanced styling, such as gradients, SVGs, etc., will not be a part of the prototype. The prototype will not have full accessibility because the HTML elements used to build the components will be divs for containers and p-tags for texts. The focus for the prototype was on how the data flows from the design software to usable code.

2 Background

As mentioned in section 1.1, Kowit uses an agile work form. The broader perspective for Knowit, not taking the iterative part of the process into account. The process of creating a web application is done in three main steps:

1. Design
2. Develop
3. Publish

Design: The designer does the majority of design work at the beginning of the project. The designer and customer collaborate to create an interface until they are both satisfied. Some developers are also present during this first part of the process to give insight into what is possible to create for the desired system.

Develop: Some development starts at the beginning of the project too. Generally, the development that starts at the beginning of the project is primarily functional, meaning there is no graphical interface yet. When the customer has approved the design, the development starts on the visual parts of the system. The developer takes the design and tries to mimic the design with code to which the functionality can be attached.

Publish: When the web application is at the point that the customer, designer, and developers are satisfied, the application can be published. When the application is published, the customer's users have access to it. For most projects, the customer stays in touch with the designer and developers to manage the application. Management consists of fixing bugs, updating functionality, adding features, and updating features.

2.1 Prototype data flow

The tool that this project is creating will follow these three steps except the last one. The generated components are not a complete web application and should not, therefore, be published. However, the generated components need to be distributed between designer and developer. Therefore *publish*, for this project is replaced by *distribute*.

All these steps are often, if not always, done iteratively within them selfs and as a whole cycle. However, there must be some design to start a meaningful and valuable development effort. If there is nothing that has been developed, there is nothing to publish. So, therefore, these three must be done in order.

The tool that this project produced has a part in all these three steps.

1. Design - Using the UI program Figma
2. Develop - With Figmas API and generator prototype
3. Distribute - By copying files or using a package manager

A data flowchart for the system is shown in figure 2.

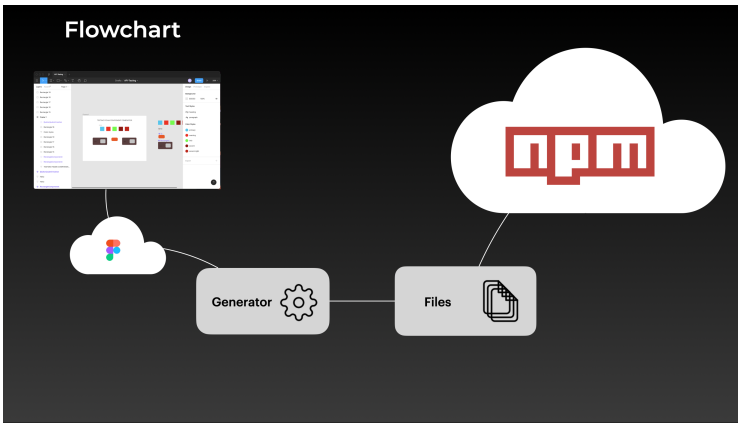

Figure 2: Flowchart for data flow in the system.

2.2 Competitors

The idea of making a design program generate functional code is not new. To get inspired and understand how these software work, we will look at two competitors in this field.

2.2.1 Webflow

Webflow was founded in 2013 and is a product of the famous Y Combinator program. Webflow allows the user to design, create and publish a website all from their web application. Webflow is a visual editing tool. The user does not need to know how to program since Webflow generates HTML, CSS, and JavaScript from the design. Most UI applications let the user move elements freely around the canvas. Webflow is a more static build tool where the elements in the design *snap* in place. Most of the design is made through the control panel, which can be seen in figure 3, and not on the canvas itself [5].

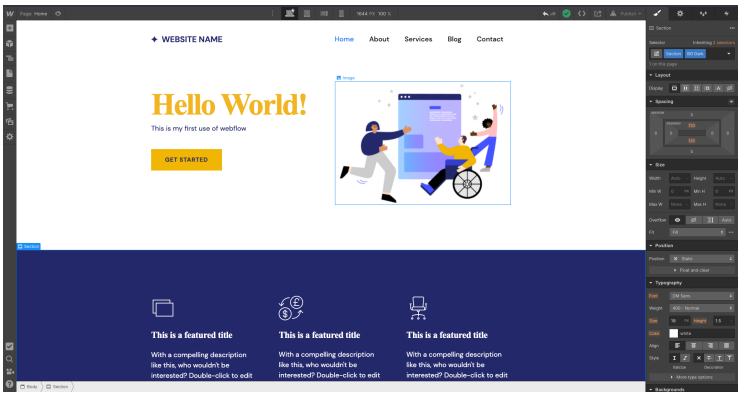

Figure 3: Screenshot of Webflows UI

2.2.2 Visly

Visly was founded in 2018 and is very similar to Figma in how the user designs the product. Visly uses the design to create React components [6]. React is a component-based JavaScript framework made by Facebook. Visly essentially makes it possible to create these components visually.

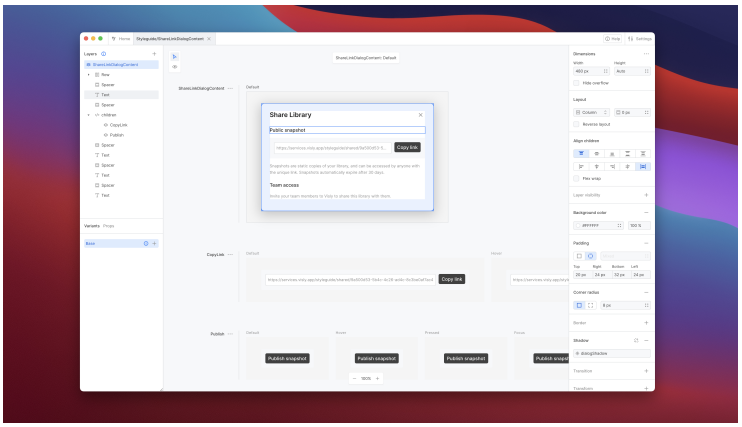

Figure 4: Screenshot of Vislys UI

2.2.3 Competitors summary

Visly and Webflow generate functional code from a graphical user interface (GUI) but in very separate ways. Webflow's service does not require experience with software development, whereas Visly does. Webflow places all functionality in the GUI, which becomes very extensive. Visly, on the other hand, requires the user to know software development and can therefore have a sleek and

straightforward GUI.

The disadvantages of both of these competitors are that they are locked with either a framework or software. This dependency is something that we want to avoid as far as possible because of the reason, stated in section 1.1.1, that Knowit wants to use the software in all projects. Knowit uses the UI program Figma when designing UI's for their projects. Knowit is happy with the functionality within Figma and does not want to change the software. Therefore the prototype was built around Figma as a UI design platform.

3 Theory

The method of this study is being made possible from the theory displayed in this section. This section will go through all theory surrounding the building and testing the prototype built in this thesis. These theories are:

- Design
- Design software
- Web development
- Programming languages
- Distributing software
- Testing methodologies.

3.1 Design

The design work of a project is primarily held at the start of the project. The project team is in contact with the customer and hash out what functionality and appearance the application will have. With software, it is often hard for customers to know what they want and what is possible to do. Therefore this design segment is often done in iterations.

A fairly common way of making the design of websites is first to create prototypes with little details, called low fidelity prototypes. More details are added to the prototype through iterations until the design could be mistaken for an actual application. This is called a high-fidelity prototype.

The medium with which these prototypes are created varies, but a User Interface (UI) design application is often used to make the final high fidelity prototype. This high-fidelity prototype is visually accurate with the final product meaning that all colors, typographies, and layout are complete. The design prototype then needs to be translated into code by the developer.

3.1.1 UI Design Applications

User interface (UI) design is often the first step in building an application. By designing the UI first, the designer can collaborate with the customer iteratively until they have a design agreed upon to move forward to development.

Sketch[7] was released in 2010 and was one of the first UI design applications and lead the market for many years. In later years applications such as Figma[8] (released 2016) and Adobe XD[9] (released 2015) have come to overtake Sketch's dominance.

In this thesis, Figma is the UI design application that is used because Knowit is already using this tool. Figma is also one of few design applications that have an open API which makes this project possible.

3.1.2 Figma

Figma is a vector based UI Design application that is web-based, which means that the whole program is run over a network[10] .

Figma has an open REST-API (Representational State Transfer) that supplies the information of the Figma document to a server over the Internet[11], [12]. Figma is also web-based, meaning that Figma itself also runs over a network connection. The API is then constantly updated after each design change, which is great for collaboration and keeping the REST-API updated.

3.1.2.1 Figma Components

Figma also allows its users to create components. A component is a set of elements that are combined. Figures 5 and 6 show an example of a card[2] component in Figma. The card component consists of four elements, two text elements, and two frame elements.

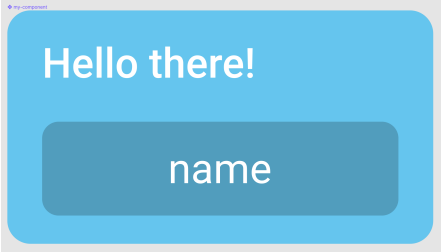

Figure 5: Figma component

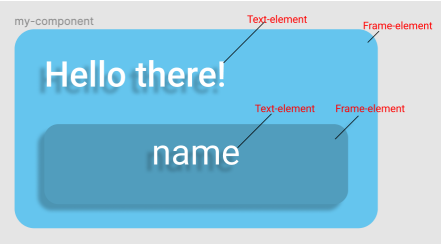

Figure 6: Elements of a component

When copying a component, the copied component is *attached* to the original component. When making changes to the original component, all *attached* com-

ponents will get the updates. The attached components can be changed without affecting the original component.

3.1.2.2 Figma Styles

Figma has a feature that lets the user store colors, texts, and effects as *styles*. This is a way for the user to store default styles for their design. For example, if the default color for a design is green, the user can store this green color as a color style. The user can use the color style throughout their user interface (UI). If they later want to change the default color for the UI, they only need to change the color of the color *style*, not all colored elements individually. Effectively the changes between figure 7 and 8 are from changing the color style. The same principle extends to typographies and effects such as shadows and blurs.

Figure 7: UI with blue color style

Figure 8: UI with green color style

3.2 Develop

Developing something from a design is essential to make the design functional with code. A design is often more like an image rather than an application. A website is usually built using the three main languages of the web: Hyper Text Markup Language - The standard markup language for creating Web pages (HTML), CSS, and JavaScript. HTML is a *"tagging"* language meaning all code is written with tags. All information is written between or in opening and closing tags. An opening tag signifies a tag with $<nameOfTag>$ and a closing tag $</nameOfTag>$. Within the opening tag, attributes can be assigned, such as a class or an id. HTML is often referred to as the body of a website. This is where the majority of the information on the web pages is stored. CSS is the clothes to the body. CSS is what defines the style for the website, such as sizes, paddings, and colors.
"CSS describes how HTML elements are to be displayed on screen, paper, or in other media"[13].
JavaScript is a scripting language that enables us to create complex features on

web pages. JavaScript enables us to update the content of the website dynamically.

In modern web design, many different frameworks and languages make it easier for developers to create products. All these tools have one thing in common. They all convert to the three browser languages mentioned above.

One of the thesis objectives is to create a tool that could be used in as many projects as possible. Knowit is, as explained in the intro, a large company that has many different projects using lots of different frameworks.

3.2.1 REST API

REST stands for **Re**presentational **S**tate **T**ransfer and is anarchitectural style of distributed hypermedia systems. Roy Fielding created this in 2000 with the release of his dissertation[14]. For an API to be called RESTful, it needs to fill the following six requirements[14], [15].

1. Client-server - the UI and data storage are separated:w.
2. Stateless - The server does not store any information about the client. The client must provide all information for every request.
3. Cacheable - A response can be explicitly or implicitly labeled as cacheable or non-cacheable. If the response is cacheable, the client has the right to store and reuse the response data for later.
4. Uniform interface - Simplifies and decouples the architecture between clients and servers, which enables each part to evolve independently. Four principles guide this: Resource-Based, Manipulation of Resources Through Representation, Self-descriptive Messages, Hypermedia as the Engine of Application State[16] .
5. Layered system - This architecture consists of hierarchical layers that constrain what each component can do, such as; a component can only interact with the layer it is on.
6. Code on demand (optional) - This allows the client to download and execute applets or scripts and therefore extending client functionality

These requirements make it lightweight and easy to understand and thereby introduce fewer problems into the system.

3.2.2 JavaScript and TypeScript

JavaScript is a programming language that reports its error much later than many other languages. Variables can take any shape in JavaScript. That is, for instance, a number or a string. This can at first seem as good that the language is highly dynamic and flexible. This flexibility is also very error-prone, where most of the errors are discovered after the code is run. This results in being obliged to test running the code after every change[17]. TypeScript was

created to fix the error-prone nature of JavaScript. TypeScript is an open-source programming language that is built upon JavaScript. TypeScript allows for the creation of types for variables, functions, etc., and thereby helps to find more errors before runtime[18]. TypeScript then complies back to JavaScript before it is run and can thereby run everywhere JavaScript is run[18].

3.2.3 Node.js

asynchronously till ordlistan Node.js is an open-source project that lets its users run code on the server asynchronously[19]. This allows for a more efficient way of running JavaScript code without a browser. Node.js is designed to handle HTTP effectively. Streaming and low latency have therefore been a high priority.

3.2.4 Syntactically Awesome Style Sheets (SASS)

SASS is an extension language to CSS that adds more functionality to regular CSS. With SASS, it is possible to have a stylesheet split up into multiple files, create functions, etc. SASS is then compiled to regular CSS so the browser can understand it. SASS is also called a preprocessor for CSS because of this. SASS has two different syntaxes; the indented syntax, commonly referred to as SASS, and Sassy CSS, referred to as SCSS. The indented syntax was the original syntax for SASS and is only dependent on indentation. SCSS syntax is very similar to regular CSS but with the qualities of SASS. Because of the resembles of standard CSS, SCSS is the easiest to learn and most famous of the two syntaxes. For this project, SCSS will be used because it resembles CSS.

3.2.4.1 Variables

Variables in a stylesheet are helpful, especially when dealing with colors. Often websites have color schemes from their design. By assigning each color to a variable, only one row of code needs to be changed if the color scheme is updated. Regular CSS does support variables, but they can be cumbersome to use, especially if the variables will be used globally in the stylesheet.

To style an element in CSS, we must first target the element with a CSS selector. A selector could be an HTML element such as a div or a heading. CSS also has additional selectors called pseudo-elements. Pseudo-elements are defined by a colon before their name, such as :after, :before, and :root. These elements do not add additional data to the website but help the developer to style it.

A variable in CSS is defined by adding two dashed before the variable name. The variable must also be defined within a selector and thereby only operate within that selector. The :root pseudo-element can be used to define variables globally within the stylesheet. When using the variables, they must be put inside the var()-function to work. Below an example of defining and using a variable can be seen.

```
1
```

```
:root{
  --myColorVariable: #ff9a67;
}

div{
  background-color: var(--myColorVariable);
}
```

SCSS variables are not required to be defined within a selector, making them globally reachable within the stylesheet by default. The dollar-sign (”$”) is used to define an SCSS variable. We can use an SCSS variable by simply adding the variable where we want to use it. The example below is achieving the same result as the previous example.

```
\$myColorVariable: #ff9a67;

div{
  background-color: \$myColorVariable;
}
```

3.2.4.2 Mixin and Include

CSS is notoriously known for having many code duplications. These duplications occur when the same styling for different elements is needed. SCSS has a solution to this called mixins. A mixin is a sort of function that stores multiple CSS rules defined with @mixin before the function name and used with @include. Below is an example of centering all children in an element with a mixin.

```
@mixin centered {
  display: grid;
  place-items: center;
};

div{
  @include centered;
}
```

3.2.5 Web Components

Web Components are a set of different JavaScript APIs and HTML features that make it possible to create reusable custom elements[20]. These elements are encapsulated away from the rest of the code. All major browsers support these web components. Because web components are run natively on HTML, CSS, and JavaScript, they are compatible with all JavaScript frameworks, such as React, Vue, and Angular. Web components are built with three technologies: Custom elements, shadow DOM, and HTML Templates.

3.2.5.1 Custom Elements

With the help of a set of JavaScript APIs, we can define custom elements and their behavior. By creating custom elements, we can encapsulate HTML func-

tionality outside of the main page itself. By doing this, the HTML code becomes much more readable.

3.2.5.2 Shadow DOM

When the browser reads in an HTML page, a document object model tree (DOM tree) is created. In figure 9, we can see this tree. Shadow DOM is a set of JavaScript APIs that lets us attach an encapsulated "shadow" DOM tree to an existing node in the DOM tree. This "shadow" DOM extends the main document DOM like a branch. The difference from a regular branch is that the main DOM is not aware of the "shadow" DOMs data or functionality and vice versa. The "shadow" DOM is then essentially its own tree with its own stylesheet that cannot be modified or overwritten from the main DOM.

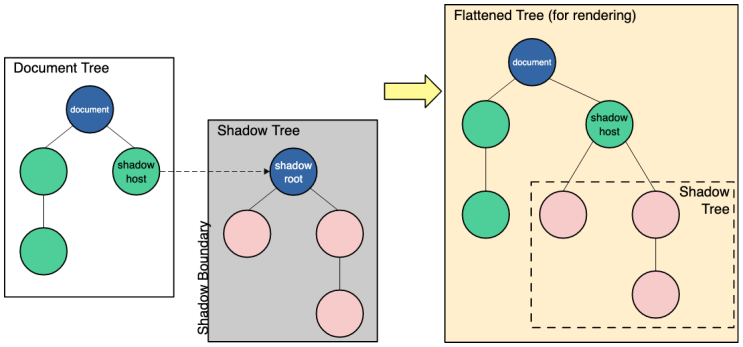

Figure 9: Depiction of a shadow DOM attached to the main DOM

3.2.5.3 HTML Templates

The HTML *template*-tag and *slot*-tag enable you to write markup templates that are not displayed on the rendered HTML page. Which then can be reused throughout the HTML page. HTML templates enable web components to be reused multiple times, with different instances, in the DOM tree.

3.2.6 LitElement

Web components can be manually built using custom elements, shadow DOM, and HTML templates mentioned above. A group of engineers from the Google Chrome team in the Polymer Project [21] have built a lightweight class called LitElement. LitElement combines functionality from the web components technologies to a class that makes it easy to create web components with concise and malleable code.

3.3 Distribute

After a component has been designed and developed, the components can now be used in a project. Often the created component is used by more than one part, and therefore it should be distributed seamlessly. There are many ways to distribute a component, but the most widely used way is to use a package manager.

3.3.1 Package Manager

The most traditional way of installing and updating software is to download an installer. This installer will then install the software to the system, which can then be used. If an update is needed, the software can notify the user to update or update itself in the background.

A package manager is a way to install and update software with ease compared to a more traditional installer. A package manager bundles up the source code into a package. This package can then be distributed over the Internet and installed and updated without an internal installer.

The package manager that this project will use is Node Package Manager (NPM), which is the world's largest software registry[22]. NPM has a straightforward interface, whereas if something needs to be installed, the following can be typed:

```
1 \$npm install PACKAGE-NAME
```

3.3.2 Open Source

The term open source refers to something people can modify and share because its design is publicly accessible. - Opensource.com

Open source originated from developers creating software that was designed to be open and modified by anyone. In an open-source project, the source code for the software must be available for others to use. To signify that the project is open-source, an open-source license is added to the source code. These licenses affect how the user can use, study, distribute, and modify the software's source code. The most used licenses state that the user can do anything they wish with the software. Some licenses state that if alterations are done on the software, the altered version must also be open-source.

3.4 Testing

When a product has been created, tests can be run on the product. The test's main objective is to find contingent bugs in the software and flaws with the UI. Further, the testing is done to ensure that assumptions taken when creating the product are verified.

3.4.1 Usability testing

Usability testing or "user research" is a broad term. As Lewis[23] described it: "Usability testing, in general, involves representative users attempting representative tasks in representative environments, on early prototypes or working versions of computer interfaces."

Usability testing is essentially performed to find flaws in an interface by putting the user in the environment of using the interface. Usability testing is done in all stages of development, from paper prototypes to high-fidelity screen mock-ups.

Lazar et al.[24, Chapter 10] consider usability testing a cousin to traditional research methods, where traditional research refers to methods such as experimental design[24, Chapter 3] and ethnography[24, Chapter 9]. Similarities can be found in experimental design with measurement of task performance and time performance, surveys, and observation techniques from ethnography. The participants in usability testing, as in traditional research, must remain anonymous, be informed of their rights, and can leave the research at any time. What separates usability testing from traditional research is often the end goals. For usability testing, the end goals are to create the best product possible, with the time and resources at hand, while the traditional research methods want to find answers to questions that are universal for the field researched.

Wixson proclaims in his study that usability testing is closer to engineering than traditional research [25]. Usability testing, as engineering, is focused on creating a successful product with limited time and resources. Tests are done in iterations where the prototype will be changed between each test to fix the flaws found during the test. The next iteration of tests will then be used to verify the fixed flaws simultaneously as it searches for other flaws. This is, in most if not all traditional research, considered unacceptable.

To get more credible data out of a test and not just for improving the product, the test environment should be kept as similar as possible for each user. By doing this, the data can be statistically analyzed, making it easier to make more accurate decisions of what to do next.

Usability testing can collect quantitative data such as time- and task performance. However, the majority of data that is collected is qualitative. As mentioned before, the most significant end goal for a usability test is to uncover flaws in the user interface, which is often subjective for the user.

"Often in industry, schedule and resource issues, rather than theoretical discussions of methodology, drive the development process [25]."

3.4.1.1 Sufficient Amount of Test Users

In usability testing, it has become widely accepted that five users are the most efficient number of participants for usability testing. Nielsen and Landauer suggest the number five out of a cost ratio to numbers of tests[26]. Nielsen

started the *"discount usability movement"* in 1989 with his presentation of the paper: "Usability Engineering at a Discount"[27]. This *movement* tries to find cost-effective methods for usability testing. From Nielsen and Landauer's study, they found that the number of usability problems found in a usability test with n users is:

$$N(1-(1-L)n)$$

Where N is the total number of usability problems and L is the proportion of problems found by one test user. With this, they determined that when using five users, 80% of usability problems are found.

Lazar, et al. explains that many researchers disagree with the assertion of five test users for successful usability testing because of Nielsen and Landauer approximation of the total number of usability problems[24, Chapter 10.5.3]. This number N is probably unknown, and therefore the number of test users n cannot be found.

The topic of what number of test users are sufficient for usability testing has been discussed for over 30 years. What is important to point out is that performing usability testing is valuable to do than not to do.

So instead of saying, "how many users must you have?," maybe the correct question is "how many users can we afford?," "how many users can we get?" or "how many users do we have time for?" [24]

3.4.2 A/B Testing

A/B testing, or bucket testing, is a user experience (UX) research method where two variants of a program/interface are tested. These two variants are referred to as A and B, hence the name A/B testing. The A and the B variant are tested on the user, and then their responses are compared and evaluated. Often just a small change is made in a UI and evaluated on many users.

A/B testing is verified using two-sample hypothesis testing from the field of statistics. This means that decisions that will be made are entirely based on data. Then there is no guessing on where to go next.

To explain this further, we will use an example. This example will follow five steps:

1. Identifying the goal of the test.
2. What will be tested?
3. Create a "control" and "challenger".
4. Split users into equally large sample groups.
5. Decide how significant the result of the test needs to be.

In this example, we will use an user interface of a data management application, see figure 10.

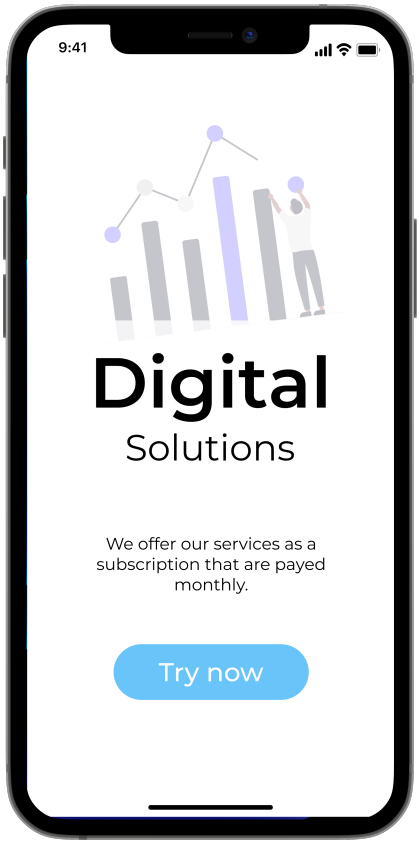

Figure 10: Variant A for the A/B-test

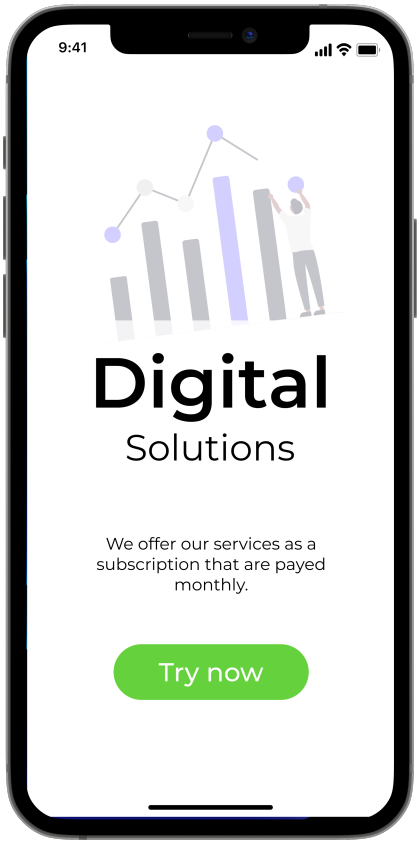

Figure 11: Variant B for the A/B-test

Identifying goal
We want more users to try the digital solutions in the application. Therefore the goal for the test will be to optimize the click rate on the ”Try now” button.

What will be tested?
We will only focus on the color of the ”Try now” button. This is the only change that will be done. Because of this, we can be sure what effect the color of the button has on click rate.

Create a ”control” and ”challenger”.
The two variants tested are often called the ”control” and the ”challenger”. In our example, the ”control”, which has not changed, is variant A and the ”challenger”

is variant B, which has a green button instead of blue. These variants can be seen in figures 10 and 11.

Split users into sample groups.
The users are split into equally large sample groups where they either are shown variant A or variant B.

Decide how significant the result of the test needs to be.
Before the test begins, a statistical significance must be established. Usually, this number is set to 95%-99%, meaning a 5%-1% chance that the test is showing a false positive.

Let's say we run the test on 20000 users, where 10000 gets variant A, and 10000 gets variant B. We record every time a user presses the button. In table 1, we can see a small portion of these recordings.

	Group	Pressed
user 1	A	yes
user 2	A	
user 3	B	yes
user 4	A	
user 5	B	
user 6	B	yes
user 7	A	
user 8	B	yes
user 9	A	yes
user 10	B	yes

Table 1: Table of a sample of users.

To establish the significance value for the test, we remove all values in the *group* column. Then we can simulate the group for each user, e.g., randomly apply a group to each user.

By doing this simulation, a distribution of the combination of A and B can be observed. When the distribution is known, we can estimate the probability of all A and B combinations. In our example, we saw that in the empirical test, two users in group A pressed the button, and four users in group B pressed the button. We then look at our simulated distribution of what probability there is that the empirical test results or more extreme occurred.

3.4.3 Statistical Analysis

When data has been collected a statistical analysis needs to be done to be able to make any "hard" conclusions. A lot of decisions need to be made when analyzing the collected data. What statistical method to be used, the confidence threshold, and how the interpretation and significance of the test results should be. If the

wrong method is used or if the interpretation of the results is inappropriate the conclusions drawn from the study can be erroneous [24].

Preparing Data: before we can do anything with the data often the data must be cleaned and organized.
Descriptive statistic: when the data has been cleaned and organized it can be a good idea to run some tests to understand the nature of the data. This can unfold what patterns or tendencies lays in the data. This makes it easier to choose the correct statistical method for the collected data at hand.
Analyze: when we understand the nature of the data we can analyze the data with the help of a statistical analysis method. This method could be a T- or F-tests, chi-squared test, etc. depending on the data collected.
Results: when the analysis is done the results must be interpreted according to the methods used.

4 Method

In this section, the methods used to answer the research questions (see section 1.2) will be presented. At the beginning of the project, a literature study was performed. This to get an overview of what similar work existed in the field. Knowit, the company connected to the study, wanted an easier way to create components for web development projects. A tool was built to fulfill this want.

Throughout this process, Knowit was involved with semi-weekly checkups for support. Semi-Structured interviews were carried out on the employees of Knowit to steer the development of the tool to fit their needs.

When a prototype was somewhat complete, iterations of usability testing were performed. This was done to ensure that the tool was usable for more people than just the author. Lastly, to investigate what impacted the created tool could have, an A/B test was scheduled to be performed.

4.1 Initial Research

At the start of the project, a meeting was held with a team at Knowit to determine their needs and their requirements on the project. After some discussion, there was an interest in creating an automated generating of code from their UI design program, Figma.

With this in mind, research began on what competitors there were in the field of code generation for web applications, and two competitors were most interesting, Webflow and Visly (see section 2.2). Some interesting small projects were also found that were used as inspiration for creating the tool. One project, especially creating color variables from Figma made by Karl Rombauts, was used [28].

How to encapsulate the Figma design elements into code was one of the most severe difficulties. Different technologies were considered to find the fit for

Knowit's requirements. These technologies were using a JavaScript framework (such as Angular or React), plain HTML-CSS-JavaScript, or Web components with LitElement.

One of the requirements was that the tool could be applicable for all types of projects. Using a JavaScript framework would mean that some projects would not support the tool. Therefore that technology was counted out. Using plain HTML-CSS-JavaScript could work for all projects, but it would be hard to manage since the generated code would be static and hard to encapsulate. LitElement was chosen as the technology for encapsulating the Figma design elements into code. LitElement is run natively in HTML-CSS-JavaScript, and therefore works with all projects. LitElement is also much easier to manage than just generating plain HTML-CSS-JavaScript.

4.2 Creating the tool

The first step of creating the tool was discussing with Knowit what could be possible to achieve under the 20 week project time. As seen from the literature study, the big problem was how to condense the elements from the code to be used efficiently.

The tool that was to be built would fetch data from Figmas REST-API, interpret the response from the API, and build LitElement objects from the interpretation.

The tool was built using an experimental approach, meaning that code was tested, and possibilities were explored during the development of the program. TypeScript was chosen as the programming language because Knowit uses it in most projects and would therefore be familiar to them.

Figmas API was examined to understand what was possible to do with it. Figmas website for developers[11] was read through and also some initial HTTP-requests were sent to the API, using platform Postman [29]. The initial response from the API was quite large. This meant that setting the TypeScript types correct for all values in the response would take much time. To mitigate this the Visual Studio Code [30] extension quicktype[31] was used to generate types from the JSON response.

From the API response, we could see that most, or at least enough, of the data were the same as styling in CSS. This meant that styling elements with CSS were possible from the API.

The information from Figmas API was interpreted and stored as classes of colors, typographies, and components. The strategy was to generate a string that contained the LitElement. Essentially the program generated code as a string. This string is later written into a new TypeScript file that is then compiled into a JavaScript file that could be run in a browser.

4.2.1 Building the HTML for the component

The Figma components have a parent-children structure, meaning that all elements in the component are related. A recursive function was used to search through each element and their children dynamically. This replicates the relationship between the elements and thereby creating a proper HTML structure.

If the implementation of the before mentioned structure is not done correctly, the elements would not be nested into each other.

4.2.2 Styling the Component

One of the requirements from Knowit was that the components should be easy to alter. This meant that the style of the developer must be able to alter the style of the generated component.

In the first attempt to make this happen, all CSS rules for an element were assigned a *property* in LitElement. A property allows the developer to pass data into the LitElement. By assigning a property for all CSS rules of the element, the style could be changed after generating the component.

This was later redesigned because the user could not add *new* CSS rules to the component if they wished to. This problem was fixed by storing the CSS rules in maps[32] and pushing them into the correct CSS selector. Instead of creating a property for each style attribute, only one property for each element is created. If the user wishes to add or change the styling of a component, they target the Figma element as an attribute to the component and inserts regular CSS. The component then creates a duplicate of the styling map for the targeted element and inserts the new styling attributes into the component.

4.2.3 Variables

Figma has a feature called styles. This is a way for the user to store and reuse colors, texts, and effects. This is something that is also very normal to do in a developer environment. Therefore a decision was made to create an SCSS *variable* file where these would be stored. Because of the time constraint of the project, only colors and texts were implemented. This was done similarly to the components where the "code" for the SCSS variables was written to a string that later was written into an SCSS file.

4.2.4 Open Source

One of Knowit's initial requirements where that the software produced should be open source (see section 4.2.4). The project was handed access to Knowit Experience Norrlands GitHub page. The software was uploaded publicly to this GitHub repository[33]. The MIT[34] open source license where then attached to the repository stating that the software is free to use but has no liability or warranty.

4.2.5 Userguide

The program built does not have a graphical user interface, again because of the time constraint. The user is instead using a command-line interface (CLI). This makes it a bit harder to learn because there are no visual queues of what to input into the program. A user guide was created in the form of a README on GitHub[35] to solve this issue. Alon side the prototype itself, this user guide was altered after the usability tests.

4.3 Semi-Structured Interviews

This tool is intended to be used by people in many different areas of expertise, from designers to front-end developers to back-end developers. A semi-structured interview model was used [36] to understand the work these people do in their respective roles to be able to build/develop a tool that is functional for all parts of the workflow (all the different roles in the workflow). The semi-structured interview was carried out with a script of questions that are asked to every interviewee. Unlike the structured interview, the semi-structured interview allows for further explanation and follow-up questions from the interviewer. These interviews were done with seven employees of Knowit Experience Umeå and Sundsvall.

Because of the broad nature of the tool created, it was important to get participants that worked with all affected areas of expertise. The interviews were done digitally over Microsoft Teams. The script used for these interviews can be found in the appendix (The script is only in Swedish).

4.4 Usability Testing

Usability tests were done to make sure that the prototype was usable for somebody else than the author. Furthermore, to set up the prototype for further testing regarding the effectiveness of the prototype.

Two iterations of usability testing were carried out on nine participants, four in the first iteration and five in the second iteration. The participants had to have a background in web development, NPM, and Figma. Therefore, the participants chosen for the tests were employees of Knowit and students from the interaction and design program at the University of Umeå. The test was designed as a scenario with four different tasks. The participant first got a link to the GitHub repository where the tool and the user guide were situated. The tasks were to create a viable Figma component that could be converted to a web component using the created tool. After that, the participant should install the tool on their computer. Use the tool to convert the Figma component and then insert the component, using NPM locally, in a test project supplied by the test administrator.

This was a way to test the whole chain from designing Figma components, converting these to web components, and to finally using the components in a

project. These tests could also be used to determine what was working and not in the user guide. After the tasks were done, the questions about the experience were asked. Finally, the test administrator opened up for suggestions regarding improvements to the tool or the user guide.

The test script was altered after the first iteration to target more features of the prototype. These features were the use of color and text styles. The questions from the first iteration were still asked. This was done to confirm that the changes made between iteration one and iteration two were effective.

4.5 A/B Testing

To have a metric that can be measured, discussed, and statistically verifiable whether or not the prototype is effective in real-world use, A/B tests will be performed. The primary goal of the test is the efficiency of the prototype. This will be an abnormal test, where instead of changing a small variable in an interface, the whole system will be tested.

The task for the participants of the test is to create a simple website from a description that can be found in the appendix. The test participants will be working in pairs of designers and developers, where they ought to collaborate to complete the task.

The "control" variant for the test will be creating a website as the participants are used to, and the "challenger" will be creating the website using the prototype. To establish a goal time to know when a test has passed. The "control" variant will be run multiple times to get an average time. This average time will then be used to signify the goal time. If the test is run faster than the goal time, it will pass.

The significance value for the test will be set to 95%, meaning that the test will be performed until this value is met.

5 Result

In this section, the results from the project will be presented. These results are in sections of prototyping, interviews, and usability testing.

5.1 Prototype

The prototype created in this project can create web components dynamically from a Figma document, create SCSS variables, and mixins from Figmas colors and text styles. The components, variables, and mixins are ready to be used as a package out of the box with NPM. This prototype is available on Knowit-experience-norrlands GitHub page and is free to use and modify under the MIT open source license.

The prototype has been usability tested, see section 5.3, which showed indications of being arbitrarily usable for all essential functions.

5.2 Initial interviews

The interviews were done to get an understanding of how this tool could be structured around the employee's workflow for all competence groups.

From the interviews, there was found that the majority liked the idea and thought that the created prototype could be a helpful tool. The interviews involved three different competence groups, designers, front-end developers, and back-end developers. The interviewees that were back-end developers were the ones with the most hesitation. There where much uncertainty about whether or not a tool like this could be helpful. The main concerns were the responsiveness of the components and to have a link between Figma and the components directly.

The interviewees that were designers and front-end developers were more positive. They thought that the tool was interesting and could potentially help with communication between them and the developers.

There was no objection to the implementation at the time because this way of working is a lot different from what they were used to.

5.3 Usability Testing

When the tool was functional, two usability tests were done to ensure that the tool was usable for more than just the author. The goal is that a developer should be able to use the tool with just the user guide (README) from github.com.

5.3.1 Iteration one

The first iteration of the test gave a list of flaws with the interface.

- The importance of the Auto-layout feature in Figma. The line about it was read but not understood that it was a dependency for the program.
- Hard to differentiate between explanations for Figma and the program.
- When setting up a new Figma document, all users had a hard time what they were allowed to name their document.
- TypeScript must be installed globally to run the TypeScript compiler. This was not mentioned in the documentation.
  - When the generation has been completed, the TypeScript files must be compiled using the *tsc*-command. This needs to be done to use as a package but was misunderstood.

- When creating a local NPM-package, there was no way for the users to understand what the name for the package was.
- When styling the parent of the component, the target string is the component's name in camel-case, which was not described. The styling of the parent can also be done with the regular *Style* attribute, which was suggested by one of the users.

All of the flaws found could be corrected for the next iteration of the test. The questions after each test revealed that the users thought the prototype had a logical flow but that there was a high learning curve. They liked the use of pictures in the user guide and suggested adding more if possible.

5.3.2 Iteration two

The second iteration of the test had five participants, where one was a developer from Knowit, and four were students of Interaction and Design studying at the University of Umeå. In addition to the first iteration, tasks about Figma styles were added. The second iteration of the test went much more smoothly than the first. All users could create and convert components without any intervention from the test operator. The second iteration still found flaws in the user guide and prototype. These flaws can be seen in the list below:

- The users found it hard to find the Auto-layout in Figma.
  - The importance of Auto-layout was understood during the second iteration.
- When creating a Figma access token, users now found where to create the token but not how to do it.
- When using color and text styles, the users found it hard how to implement them. They often confused the implementation of the two.
- When styling or changing text on the component, the examples are based on an image at the start of the user guide. The users did not understand that the image at the start was connected to the examples. A suggestion from the user was to duplicate the image and show it again further down.
- Most users had a hard time understanding why and when to compile the TypeScript component files. A suggestion from a user was to incorporate the compilation within the *"convert"*-script.
- Two users thought that the components should be used with camelCase instead of kebab-case when initiating the component in the HTML.
- Some users found it hard to find different headings in the user guide. Their suggestion was to make the headings bigger and thought it unnecessary to have them nested.

The flaws that were found in the second iteration of the tests could all be fixed. Changing the *"convert"*-script was one of the most significant changes but allowed the user guide to be shortened and the prototype to be run faster.

5.4 A/B Testing

No results to show.

6 Discussion

In this section, a discussion about the results of the study will be taken place. Furthermore, a discussion on what could have been done differently. What was done well, and what could have been improved for all the stages of the study. The prototype that was produced, the usability, and the A/B testing that unfortunately could not be executed.

6.1 The Prototype

Most time and effort for this project went into creating the prototype. There were a lot of problems that needed to be solved where the biggest one was; How can a component be built that works for all JavaScript frameworks, including static pages. The research focused on finding solutions that made it possible to work with static pages. Because if a solution that works natively within HTML-CSS-JavaScript has a much higher chance of working together with frameworks. The web component was a clear choice.

As explained in the theory section, Web components are a mixture of three technologies, custom elements, shadow DOM, and HTML templates. This would work natively for all frameworks and static pages. The problem with using web components is that it is complicated to make the components loosely coupled between projects. The LitElement class from Polymer provided a solution. In addition, this also made the creation of the components much more straightforward. A problem with LitElement was that to use them *open imports* need to be handled. This is when the import does not target a file or a function from a module. This is only supported when running in node.js, not in browsers at the moment. To solve this issue, a bundler must be used, such as Webpack or Rollup. This was not optimal, but the upside of LitElement still transcends this downside. This answers two of the research questions stated at the beginning of the project:

- *Can components be built that works for all JavaScript frameworks and static pages?*
- *Is it possible to automate the whole process from UI design program to browser runnable code?.*

One of the biggest struggles with creating the tool was to make sure that the prototype built the right amount of elements in the right place. This was solved with recursive functions. These functions call themself until they handle the "youngest" child and build the elements from there and go from generation to generation. This is a good way of building the components but it can become a performance taxing function if the components become too big. This has not been tested, but there have been indications of slow performance. Optimizing these functions needs to be carried out to make the prototype viable for more extensive components and projects.

The prototype has some features that, unfortunately, could not be tested within the project timeframe. Such as slotting. Where the user can decide a slot in the component where the user can insert an element or another component into it. This feature was believed to be necessary for carrying out the required task in later tests, which was a wrong assumption. The time spent on developing the feature could have been used for further testing, which would have been more efficient.

6.2 Interviews

The interviews intended to find out how the employees of Knowit work, what their day-to-day routines were. The hope was that the interviews could help shape the prototype into something that felt more familiar to them. There was not as much gathered from the interviews as hoped. Because the new way of making the components with this tool was very new, it was hard to get any similarities with the work they do today. The way of using components and controlling them through properties is something they are working on within some React projects. This was already planned to be done for the components in the prototype but was a good verification.

The interviews were carried out when the tool had been built for a while. Because of this, some assumptions built up of what the results of the interviews would be. These assumptions were discussed with Knowit, where the majority agreed with prior project experiences. These discussions were held before any interviews took place. The results from the interviews were not discussed with Knowit until all interviews were completed.

One assumption going into the interviews was that the people working with back-end would be the most excited. This because they would not have to make any adjustments from the front-end with the tool. However, when using the tool, there are more things that the designer has to think about:

- How Figma *"sees"* the structure of elements inside the component, not just the visual structure.
- Auto-layout needs to be used for all elements in the component for the prototype to work.

Because of this, another assumption for the interviews was that the designers would be more pessimistic about using the tool.

Both of these assumptions were wrong, and from the interviews, opposite viewpoints were shown. To get a statistical significance, this would require more data points, e.g., too few quantitative data points were collected about how and which opinions were separate between the competence groups.

The tool forces a more structured design and creates a commonplace where all elements and variables have the same name between competence groups. The majority of the interviewees saw this as something positive. They argued that the tool could increase communication between competence areas. This could result in faster development and a better working environment—more of this in section 8.

6.3 Usability testing

The usability testing in this thesis was done with the more industrial style of working with the resources at hand. The project was carried out over 20 weeks during the Covid 19 pandemic meaning that all testing was done remotely. As a result of this, it was harder to get users to do the tests. All test that was made where a walkthrough for the whole prototype, from design to web component. Smaller tests were considered to check individual parts of the system. This was not used because there were not enough users and time. Because of the broadness of the prototype, it was also crucial that the users could make it from start to finish without any help.

From the two iterations, seven flaws were found in each iteration. This seems as though the supposedly fixed flaws from the first iteration remained. What is important to note is that the second iteration had added tasks. There were tasks added in the second iteration to maximize the range of the test. The second iteration could confirm the fixes for the flaws in the first iteration and find new flaws in untested features of the system.

To say that the whole system is user-friendly would be an overstatement since the testing has only been performed on the system's essential functions. The usability tests showed that these essential functions, such as setting up a Figma document and converting its components, are usable and achievable for novice users. The tests also show that altering the components after they have been generated is logical for the users.

I would argue that this is a very efficient way of building a tool that automates component generation. Thereby answering one of the research questions: *How can a user-friendly tool be built that automates component generation?*

6.4 A/B testing

Unfortunately, there was no time to do A/B testing. This is still in the study because we found it important that to point out that this could be a good next step. This test was first and foremost supposed to answer the research question: *Does this tool speed up the development process and if so how much?* With the results from these tests, we could answer this question with statistical significance. This test would be very demanding on finding users but if the results show that it is more efficient to develop with the prototype there is a good incentive to invest in developing it into a real product.

The test is designed to be performed in pairs of a developer and designer so there was also a hope to get some answers to the research question: *Will automation between design and development increase communication between designers and developers?* This would be done by observing the users during testing. If there is more or less conversation and if the pair find it easier to solve the problem.

7 Conclusion

This thesis had five research questions.

- Is it possible to automate the whole process from UI design program to browser runnable code?
- Can components be built that works for all JavaScript frameworks and static pages?
- How can a user-friendly tool be built that automates component generation?
- Will automation between design and development increase communication between designers and developers?
- Does this tool speed up the development process, and if so, how much?

The three first questions could be answered by researching web technologies, creating a prototype that converts Figma components to web components, and ran usability tests on the prototype. The last two research questions did not get answered. However, the thesis supplies steps on how to get these answers through A/B testing.

The indications from users and interviewees have been positive about what the prototype can do and become a helpful tool for developers and designers.

8 Future Work

From the interviews and tests that have been performed, there have been many positive indications that the tool could be helpful. The next step for the prototype is either to create a graphical user interface (GUI) or to expand on the functionality of the component generation. The benefits of creating a GUI are that the user does not need to rely on a user guide. It also makes it much easier to handle Figma documents stored in the program. Creating a GUI would take more time, but usability testing would be easier to perform, and the user experience has more potential to be better. It would be easier to steer the user in the right direction. The learning curve for the user could be shortened to be most about using and altering the generated components.

The tool as of now only supports the most basic styling parameters such as width, height, positioning, etc., but there would not be much work to add more of these. The ones that would be a great addition are gradients, shadows, and rotation because these can become tricky to do manually, especially for more complex components.

The prototype as of now is highly dependent on the auto-layout feature in Figma. There are possible to work around this and make for a more general

solution. This could mean less work for the designer to adapt their design to work with the generator and make development more efficient.

Depending on the situation, there are arguments to go ahead with A/B testing before and after developing a GUI. If there are resources to do both, the recommendation is to develop the GUI first. Having a GUI potentially makes the learning curve smaller and thereby gets less spread on the results. If there only is resources to do the tests or the GUI, the argument could be made to do the tests first. In the worst case, this would give indications on the efficiency of the tool and at best statistically establish the tool as efficient. Then the question becomes easier if there is an opportunity to invest in the tool.

References

[1] D. Cohen, M. Lindvall, and P. Costa, "An introduction to agile methods.," *Advances in Computers, Vol 83*, vol. 62, no. 03, pp. 1–66, 2004.

[2] N. Babich. "Simple Design Tips for Crafting Better UI Cards," Medium. (Jun. 22, 2020), [Online]. Available: `https://uxplanet.org/simple-design-tips-for-crafting-better-ui-cards-19c1ac31a44e` (visited on 03/28/2021).

[3] "Manifesto for Agile Software Development." (), [Online]. Available: `https://agilemanifesto.org/` (visited on 06/08/2021).

[4] W. Fanguy. "A comprehensive guide to design systems | Inside Design Blog." (), [Online]. Available: `https://www.invisionapp.com/inside-design/guide-to-design-systems/` (visited on 02/28/2021).

[5] "Responsive web design tool, CMS, and hosting platform | Webflow." (), [Online]. Available: `https://webflow.com/` (visited on 02/26/2021).

[6] F. Inc. "React – A JavaScript library for building user interfaces." (), [Online]. Available: `https://reactjs.org/` (visited on 03/18/2021).

[7] Sketch. "The digital design toolkit," Sketch. (), [Online]. Available: `https://www.sketch.com/` (visited on 03/11/2021).

[8] Figma. "Figma: The collaborative interface design tool." (), [Online]. Available: `https://www.figma.com/` (visited on 02/09/2021).

[9] Adobe. "Adobe XD | Fast & Powerful UI/UX Design & Collaboration Tool," Adobe. (), [Online]. Available: `https://www.adobe.com/se/products/xd.html` (visited on 03/11/2021).

[10] "About Figma, the collaborative interface design tool.," Figma. (), [Online]. Available: `https://www.figma.com/about/` (visited on 06/10/2021).

[11] Figma. "Figma," Figma. (), [Online]. Available: `https://www.figma.com/developers/api` (visited on 02/09/2021).

[12] *Representational state transfer*, in *Wikipedia*, Feb. 5, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=1004990383 (visited on 02/09/2021).

[13] "CSS Introduction." (), [Online]. Available: https://www.w3schools.com/css/css_intro.asp (visited on 05/08/2021).

[14] R. Fielding. "Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)." (), [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (visited on 03/11/2021).

[15] restfulapi.net. "What is REST," REST API Tutorial. (), [Online]. Available: https://restfulapi.net/ (visited on 03/11/2021).

[16] "What is REST?" (), [Online]. Available: https://www.restapitutorial.com/lessons/whatisrest.html# (visited on 05/08/2021).

[17] A. Taivalsaari, T. Mikkonen, D. Ingalls, and K. Palacz, "Web browser as an application platform," in *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, IEEE, 2008, pp. 293–302.

[18] "Why You Should Use TypeScript in 2021," Serokell Software Development Company. (), [Online]. Available: https://serokell.io/blog/why-typescript (visited on 06/10/2021).

[19] Node.js. "About," Node.js. (), [Online]. Available: https://nodejs.org/en/about/ (visited on 06/10/2021).

[20] "Web Components | MDN." (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Web_Components (visited on 02/09/2021).

[21] polymer. "Polymer Project." (), [Online]. Available: https://www.polymer-project.org/ (visited on 03/29/2021).

[22] "About npm | npm Docs." (), [Online]. Available: https://docs.npmjs.com/about-npm (visited on 03/29/2021).

[23] J. R. Lewis, "Usability testing," *Handbook of human factors and ergonomics*, vol. 12, e30, 2006.

[24] J. Lazar, J. H. Feng, and H. Hochheiser, *Research Methods in Human-Computer Interaction*. Morgan Kaufmann, 2017.

[25] D. Wixon, "Evaluating usability methods: Why the current literature fails the practitioner," *interactions*, vol. 10, no. 4, pp. 28–34, 2003.

[26] J. Nielsen and T. K. Landauer, "A mathematical model of the finding of usability problems," in *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*, 1993, pp. 206–213.

[27] W. L. i. R.-B. U. Experience. "Discount Usability: 20 Years," Nielsen Norman Group. (), [Online]. Available: `https://www.nngroup.com/articles/discount-usability-20-years/` (visited on 05/25/2021).

[28] K. Rombauts, *KarlRombauts/Figma-SCSS-Generator*, Apr. 29, 2021. [Online]. Available: `https://github.com/KarlRombauts/Figma-SCSS-Generator` (visited on 05/09/2021).

[29] "Postman | The Collaboration Platform for API Development," Postman. (), [Online]. Available: `https://www.postman.com/` (visited on 04/23/2021).

[30] "Visual Studio Code - Code Editing. Redefined." (), [Online]. Available: `https://code.visualstudio.com/` (visited on 04/23/2021).

[31] "Convert JSON to Swift, C#, TypeScript, Objective-C, Go, Java, C++ and more • quicktype." (), [Online]. Available: `https://quicktype.io/` (visited on 04/23/2021).

[32] "Array.prototype.map() - JavaScript | MDN." (), [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map` (visited on 04/25/2021).

[33] *Knowit-Experience-Norrland/FigmaConverter*, Knowit-Experience-Norrland, May 17, 2021. [Online]. Available: `https://github.com/Knowit-Experience-Norrland/FigmaConverter` (visited on 06/10/2021).

[34] "The MIT License | Open Source Initiative." (), [Online]. Available: `https://opensource.org/licenses/MIT` (visited on 05/12/2021).

[35] "Build software better, together," GitHub. (), [Online]. Available: `https://github.com` (visited on 05/09/2021).

[36] A. Galletta, *Mastering the Semi-Structured Interview and beyond: From Research Design to Analysis and Publication*. NYU press, 2013, vol. 18.