MASTER THESIS

From Design to Code; A Study of Automation From User Interface Design Program to Production Code

---

Albin Frick

May 16, 2021

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

Contents

1 Introduction **2**
- 1.1 The Company & the Problem . . . 2
  - 1.1.1 Knowit Initial Requirements . . . 3
- 1.2 Objective . . . 3
- 1.3 Demarcations . . . 4

2 Background **4**
- 2.1 Competitors . . . 5
  - 2.1.1 Webflow . . . 5
  - 2.1.2 Visly . . . 6
  - 2.1.3 Competitors summery . . . 6

3 Theory **7**
- 3.1 Design . . . 7
  - 3.1.1 UI Design Applications . . . 7
  - 3.1.2 Figma . . . 7
  - 3.1.3 Figma Components . . . 8
  - 3.1.4 Figma Styles . . . 8
- 3.2 Develop . . . 8
  - 3.2.1 REST API . . . 9
  - 3.2.2 JavaScript and TypeScript . . . 9
  - 3.2.3 Node.js . . . 10
  - 3.2.4 Syntactically Awesome Style Sheets (SASS) . . . 10
  - 3.2.5 Web Components . . . 11
  - 3.2.6 LitElement . . . 12
- 3.3 Distribute . . . 12
  - 3.3.1 Package Manager . . . 12
  - 3.3.2 Open Source . . . 12
- 3.4 Testing . . . 13
  - 3.4.1 Usability testing . . . 13
  - 3.4.2 A/B Testing . . . 14
  - 3.4.3 Statistical Analysis . . . 14

4 Method **15**
- 4.1 Initial Research . . . 15
- 4.2 Creating the tool . . . 15
  - 4.2.1 Building the skeleton of the component . . . 17
  - 4.2.2 Styling the Component . . . 17
  - 4.2.3 Variables . . . 17
  - 4.2.4 Open Source . . . 17
  - 4.2.5 Userguide . . . 17
- 4.3 Semi-Structured Interviews . . . 18
- 4.4 Usability Testing . . . 18
- 4.5 A/B Testing . . . 19

5 Result **19**
5.1 Prototype 19
5.2 Initial interviews 19
5.3 User Testing 20
5.3.1 Iteration one 20
5.3.2 Iteration two 20

6 Discussion **21**
6.1 The Prototype 21
6.2 Interviews 22
6.3 Usability testing 23
6.4 A/B testing 23

7 Conclusion **24**

8 Future Work **24**

Abstract

Hello this is the abstract

1 Introduction

When building applications for companies there's a lot of moving parts. Most often the project is worked with an agile work form [1]. This 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 look. When working on a big project the design and the implementation of the product are done by different competence groups, mainly designers, and developers.

The communication (and or relationship?) between these two parts are essential to creating a good user experience (UX). The communication problem between designers and developers can be compared to the common *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 shouldn't effect each other, how each of the should be positioned, etc. 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 are done around components. Whereas components are design and/or functional elements in the application. A component could be a button, a navigation bar, a card[2], etc. This is done to save code meaning that all components can be reused through out an UI and between UI's. Many companies create big libraries of these components to be used within different projects. This works great for when the design languages. However when a consulting firms clients often does not have the same design language. This results in that the consulting firms often needs to redesign there components between projects.

1.1 The Company & the Problem

Knowit is an IT company with about 2600 employees with facilities in Sweden, Norway, Denmark, Finland, and Germany. Knowit has three main branches, Experience, Insight, and Solutions. These branches tackle 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 done under the Experience branch to help them with an important part of initiating projects.

This branch is consulting firm for customers that need a digital tool with high availability, accessibility, and great user experience. Every project under Knowit experience is run by teams of designers and developers. Where 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.

The problem that has araised is that the design and setup for every project is

very similar project to project. This is setting up a color palette, typography and basic components such as buttons, forms, etc. When the designer has done this the developer needs to convert this to code. This process is often very similar for every project but needs to be redone because most project doesn't use the same frameworks and/or tools.

1.1.1 Knowit Initial Requirements

The initial requirements from Knowit was to make a design system[3] to solve to problem at hand. These following are those requirements:

- The system needs to be applicable to all types of projects.
- The system needs to be modular, you can choose to just use parts of it.
- It has to be easy to change global parameters such as colors, fonts, margins, etc.
- The system will be open source and easy to make changes to.
- The system needs a thorough documentation.

After some research the conclusion was made that there would not be enough time to administrate the design system after it was built. Therefor the angle of attack was changed to focus on the making of components. The before mentioned requirements still stood but instead of building a design system we wanted to build an automated process to generate code from a design.

1.2 Objective

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

The aim of this study is to make this setup time extensively more efficient with a tool that generates components.

To make this possible the following research questions must be answered:

- 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 the limited time of 20 weeks. Therefore only a basic component generator will be produced. With *basic* components the only things that will be taken in to account is shape, color, typography and layout. More advanced styling such as gradients, SVGs, etc. will not be apart of the prototype. The prototype will not be full accessibility becuase the elements used to build the components where only divs, for containers, and/or p-tags, for texts. The focus was on the data flow from design tool to usable code and therefore the component itself were not required to be perfect.

2 Background

The process of creating a web-application is typically done in three major steps: Lägga till en förklaring till vad dessa tre betyder?

- Design
- Develop
- Publish

The tool that this project is creating will follow these three steps except the last one. Instead of publishing however the components that will be generated need to be distributed between designer and developer. Therefor ***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 sort of design to start a meaningful and useful 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

In figure 1 we can see a flowchart over how the data flows through the system.

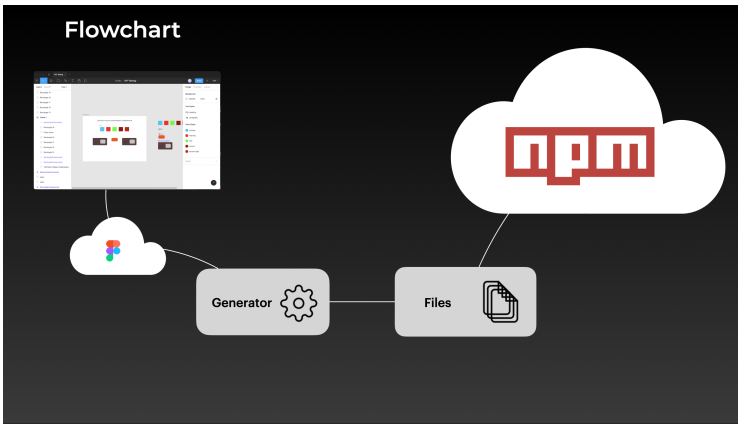

Figure 1: Chart over how data flows in the program

2.1 Competitors

The idea of making a design program generate functional code is not new. For inspiration and to get a general knowledge over how these programs work we will look at two competitors in this field.

2.1.1 Webflow

Webflow was founded in 2013 and is a product from 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 doesn't need to have any knowledge about programming since Webflow generates HTML, CSS and JavaScript from the design. Most UI applications lets the user move elements freely around the canvas. Webflow is a more static build tool where the elements in the design *snaps* in place. Most of the design is made thought the control panel, that can be seen in figure 2, and not on the canvas itself [4].

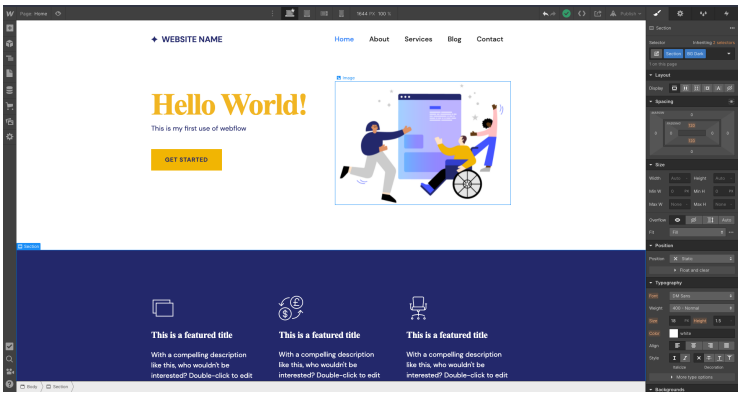

Figure 2: Screenshot of Webflows UI Lägg till rätt referens för screenshots

2.1.2 Visly

Visly is 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 [5]. React is a component based JavaScript framework made by Facebook. Visly essentially makes it possible to create these components visually.

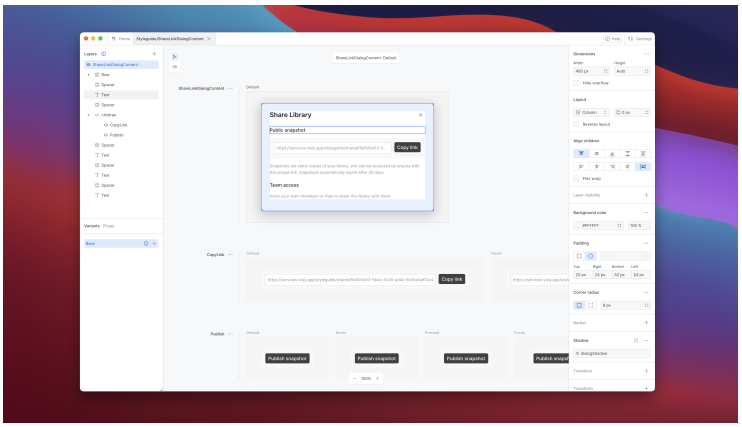

Figure 3: Screenshot of Vislys UI Lägg till rätt referens för screenshots

2.1.3 Competitors summery

The disadvantages with both of these competitors is that they are locked with either a framework or a program. This 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 be able to use the program 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 doesn't want to change program. Therefore the prototype must be built around Figma as a UI design platform.

3 Theory

The method of this study is being made possible from the theory that is displayed in this section.

3.1 Design

The design work of a project is largely held at the start of the project. Where the project team is in contact with the customer and hash out what the application and system will look like. With software there is often hard for a customer to know what they want and what is possible to do. This is a part of the design segment of the project and often needs some iterations to get things done.

A fairly regular way of making the design of websites is to first create prototypes with very low details and then build in details from there. This is Low Fidelity (LoFi) to High Fidelity (HiFi) prototyping. Where the customer can be a part of the steps to create something that they want and can be done. The medium with which these prototypes are created varies but most often than not a User Interface (UI) design application is used to make the final HiFi-prototype. This HiFi-prototype is visually accurate with the final product meaning that all colors, typographies and layout are complete. This then needs to be translated into code by the developer.

3.1.1 UI Design Applications

User interface design is often the first step in build an application. This is done to ensure the customer that the product is going to look as they want it to and that the team building the application is on the same page. **(Komma överens)**. Sketch [6] was released in 2010 and was one of the first UI design applications and lead the marked for may years. In latter years applications like Figma [7] (released 2016) and Adobe XD [8](released 2015) has come to take over Sketches overwhelming dominance.

In this thesis Figma is the tool that is being looked at because this tool is already being used by Knowit. 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 UI Design application which is web based. Meaning that the whole program is run over network.

Figma has an open REST-API, (Representational State Transfer), that supply's the information of the Figma document over the Internet [9], [10]. Figma is also

web-based meaning that the program runs over a network connection. Because of this the API is constantly updated after each change in the design, which great for collaboration and getting the API constantly updated.

3.1.3 Figma Components

Figma also allows its users to create components. This is a set of elements that is combined to a component. This is for example a button that could be used in many different places around the UI. A component is a great use case for this because when a component is made all its copies or children *"looks"* at the original component. This means that if a change is needed to be done to that component the original is the only one that is needed to be changed.

3.1.4 Figma Styles

Figma has a feature that let's the user store color, text and effect styles. These are a way for the user to store default styles for their design. This style can later be used in many different elements. For example if the default color for a design is green the user can store this green color as a color style. If the user changes its mind after a while and they want the default color for a design to be blue. The only thing that the users then needs to do is to change the color style and all the elements that uses this default color will change its color. The same principle extends to typographies and effects such as shadows and blurs too.

3.2 Develop

To develop something from a design is essentially to make the design functional with code. A design is often more like a picture rather then an application. A website is almost always built using the three main language of the web, which is HTML, CSS and JavaScript. HTML is a *"tagging"* language which means that every thing is built with tags. HTML is often referred to as the body of a website. This is where the majority of the information of 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 to mention a few things.
"CSS describes how HTML elements are to be displayed on screen, paper, or in other media"[11].
JavaScript is a scripting language that enables us to create complex features on web pages. JavaScript enables us to update content dynamically, essentially making the body move.

In modern web design there are a lot of different frameworks and languages that makes it easier for developers to make create products but all these tools have the one thing in common. They all convert to one of the three browser languages mentioned above.

One of the goals of the thesis is to create a tool that could be used in as many

projects as possible. Knowit is, as explained in the intro, a fairly large company and has a lot of different projects with lots of different frameworks.

3.2.1 REST API

REST stands for **Re**presentational **S**tate **T**ransfer and is a architectural style of distributed hypermedia systems. That was created by Roy Fielding in 2000 with the release of his dissertation [12]. For an API to be called *RESTful* it needs to fill six requirements [13].

1. Client–server - the UI and data storage are separated
2. Stateless – The server does not store any information of 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 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. This is guided by four principles: Resource-Based, Manipulation of Resources Through Representation, Self-descriptive Messages , Hypermedia as the Engine of Application State [14] .
5. Layered system - This architecture consists of hierarchical layers that constrains what each components can do, such as a component can only interact with the layer that it is on.
6. Code on demand (optional) - This allows the client to download and execute applets or scripts and there for extending client functionality

All of these requirements makes it light weight and very easy to understand and thereby introducing 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 something good that the language is highly dynamic but this is also very error prone. Where most of the errors are discovered after the program is run. This results in being obliged to test running the code after every change [15]. To solve some of these problems TypeScript was created. TypeScript is an open source programming language which is built upon JavaScript. TypeScript allows to create types for variables, functions, etc., and thereby helps to find more errors before runtime. TypeScript is then complied back to JavaScript before it's run and can thereby be used everywhere JavaScript is used.

3.2.3 Node.js

Node.js is an open source project that lets its users run code on the server asynchronously. Which is a way efficient way of running JavaScript code without a browser. Node.js is designed to handle HTTP effectively. Streaming and low latency has therefore been a high priority.

3.2.4 Syntactically Awesome Style Sheets (SASS)

SASS is an extension language to CSS that makes CSS supercharged. With SASS it is possible to have the stylesheet split up into multiple files, create functions, etc. SASS is then compiled to regular CSS so the browser is able to understand it. SASS is also called a preprocessor for CSS because of this. SASS has two different syntaxes, the indented syntax commonly referred to just SASS and Sassy CSS (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 normal CSS SCSS is the most easy to learn and most popular of the two syntaxes. For this project SCSS will be used because of the resembles to CSS.

3.2.4.1 Variables

Variables in a stylesheet is useful especially when dealing with colors. Often a website has a set color scheme from the design. This can easily be set as a variable if the colors needs to be changed for some reason. Then there is only one entry that needs to be changed. Regular CSS do support variables but they are a little "clunky". To set a variable you set two dashes infront of the variable name. The variable must also be within a selector. To make the variable golobally reachable through out the CSS file it can be placed under the :root-psuedo element. The clunky part of this implementaion is that you cannot use this variable as it is later on in the code. You have to surround the variable with *"var()"*. An example of this can be seen below.

```
1
2  :root{
3    --myColorVariable: #ff9a67;
4  }
5
6  div{
7    background-color: var(--myColorVariable);
8  }
```

SCSS makes this much more intuitive with defining the variable globally without a selector and locally just inside one. A variable is assign using the "$" character.

```
1  \$myColorVariable: #ff9a67;
2
3  div{
4    background-color: \$myColorVariable;
5  }
```

3.2.4.2 Mixin and Include

Often when writing CSS code you run into the problem of duplication of code. SASS has a solution to this called mixins. Mixins lets you store multiple CSS rules in a variable/function that can be used multiple times throughout the stylesheet. To use the mixin you need to include it in your code, both mixins and includes are signified with the @ before the word. Below is an example of centering all children in an element with a mixin.

```
1  @mixin centered {
2    display: grid;
3    place-items: center;
4  };
5
6  div{
7    @include centered;
8  }
```

3.2.5 Web Components

Web Components are a set of different JavaScript APIs together with HTML features that makes it possible to create reusable custom elements[16]. These elements is encapsulated away from the rest of the code. These web components are supported by all major browsers. Because web components are run natively on HTML, CSS and JavaScript they are compatible with all JavaScript frameworks, such as React, Vue and Angular. To make this possible there are three main technologies at work; Custom elements, shadow DOM and HTML Templates.

3.2.5.1 Custom Elements

A set of JavaScript APIs that allow you to define custom elements and their behavior. This is a way to encapsulate functionality on the HTML page rather then having everything nested together.

3.2.5.2 Shadow DOM

Shadow DOM is also a set of JavaScript APIs that lets you attach a encapsulated "shadow" DOM tree to an element. This "shadow" DOM is attached to the main document DOM like a branch. The difference from a normal branch is that the main DOM is not aware about 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 main DOM.
Lägga till bild?

3.2.5.3 HTML Templates

The HTML *template*-tag and *slot*-tag enables you to write markup templates that are not displayed in the rendered page. Which then can be reused through-

out the HTML page. HTML templates is what enables web componets to be reused multiple times, with different instances, in the DOM tree.

3.2.6 LitElement

Instead of creating web components the usual way by manually initiating custom elements, the shadow DOM and HTML templates, we can use LitElement. LitElement is a great light weight class to simplify making web components[17]. LitElement is built by the Polymer Project [18], which is a group of engineers from the Google Chrome team. LitElement combines functionality from the web components technologies to a class that make it easy to create these web components with concise and malleable code.

3.3 Distribute

After a component has been designed and developed the components should be used in a project. Often the created component is used by more then one part and therefore it should be distributed easily. There is a lot of ways to distribute a component but the most widely used way is to use a package manager.

3.3.1 Package Manager

[19]

A package manager is a way to install and update programs with ease. The package manager that this project will use is Node Package Manager (NPM) which is the world's largest software registry[20]. A package manager lets you bundle your code up to a package and distribute it over the Internet. NPM has a very easy interface whereas if you want to install something you can just type:

```
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 were design to be open and could be modified by anyone. Nowadays the term *open source* is more broader where almost all projects can be open source.

Making software open source is quite easy. The source code for the software must be available for others to use. Often the software has a open source license attached to it. These licenses affect how the user can use, study, distribute and modify the source code of the software. The most used licenses states that the user can do anything they wish with the software. Some licenses states that if there are alterations done on the software that alteration must also be open source.

3.4 Testing

When a product has been created it's a good idea to test the product before going out to production. Testing can mean many different thing but almost all forms of testing have the common gene of making sure the assumptions taken when creating the product is verified.

3.4.1 Usability testing

Usability testing or "user research" is a very broad term. As Lewis [21] 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 done 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 screen mock-ups with no functionality to implemented existing systems.

Usability testing can be considered a cousin to traditional research methods. When 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 have the ability to leave the research at any time. What separates usability testing from traditional research is often the end goals. For usability testing the end goals is to create the best product possible, with the time and resources at hand, while the traditional research methods wants to find answers to questions that is universal for the field researched.

Wixson proclaims in his study that usability testing is closer to engineering than traditional research [22]. Usability testing, as engineering, is involved with creating a successful product, with limited time, and resources. Often in a real world scenario the prototype will be changed between each test to fix the flaws that were found during the test. The next test will then be used to verify the fixed flaws simultaneously as it searches for further 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. This is closer to the experimental design approach.

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 biggest 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 [22]."

3.4.1.1 Sufficient Amount of Test Users

In the realm of usability testing it is widely accepted that the most effecient amount of users for usability testing is five people [23]. Where 80 percent of the interface flaws will be found with five users. Nielsen and Landauer also asserted the number five but later in 1993 suggested that the number of test users depend on the size of the project[24]. Nielsen and Landauer suggests seven testers for a small project and 15 testers for a medium-to-large project.

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?" [25]

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 is tested. These two variants is referred as A and B, hence the name A/B testing. The A and the B variant is tested on the user and then their responses is compared and evaluated. Often just a small change is made in a UI and evaluated on a lot of people

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

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 wrong method are used or if the interpretation of the results are inappropriate the conclusions drawn from the study can be erroneous [25].

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 on the data to understand the nature of it. This can unfold what patterns or tendency's 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 of the data collected.
Results: when the analysis is done the results must interpreted according to the methods used.

4 Method

In this section the methods used to fulfill the research questions (see section 1.2) will be presented. In the start of the project a literature study was performed to get an overview of what similar work had been done in the field and see what could be used as inspiration and what not do to. A tool was built to fill the need of the company connected to the study, Knowit. Throughout this process Knowit was involved with semi-weekly checkups to hold the project on the right course. Semi-Structured interviews where done on the employees of Knowit to steer the development of the tool to fit them. When a prototype was somewhat complete iterations of usability testing were done on the employees on Knowit and on students in University of Umeå where perform. This do insure that the tool were usable for more people then the developer himself. Lastly to investigate what impacted the tool created could have a A/B test where scheduled to be preformed.

4.1 Initial Research

In the start of the project a meeting was held with a team at Knowit on what their needs where and what requirements they had 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 was in the field of code generation for web applications, here two competitors where most interesting, Webflow and Visly,(see section 2.1). Some interesting small projects where also found that were saved and used as inspiration for creating the tool. One project especially of creating color variables form Figma made by Karl Rombauts was used [26]. How to encapsulate the Figma design elements into code where one the biggest problems. Many different technologies were researched to find the fit for Knowits requirements.

4.2 Creating the tool

Meetings with Knowit and where discussions of what could be possible to do under the 20 weeks of work that should be carried out. As seen from the literature study the big problem was how to condense the elements in from the code to easly be used.

Identifying the tools that should be used.

- TypeScript - for building the tool.
- Exploring Figmas API.
- LitElement - For building the components.
- Building the html elements and styling with recursive functions

To build the tool an experimental approach with **TypeScript** was used. This means that code where tested and possibilities where explored during the build-

ing of the program. Why **TypeScript**? because of the unknown possibilities at the start of the development there where some thoughts on making the generator part of the component itself. Thereby the browser had to be able to run the code which made JavaScript a good match. Because of the error prone nature of JavaScript (discussed in section 3.2.2) TypeScript, the supercharged version of JavaScript, was chosen. TypeScript need to be compiled to JavaScript before it can be run in a browser but that hassle is worth the benefits of getting error warnings before runtime.

To understand what could be done with Figmas API it first had to be examined. Figmas website for developers[9] was read through and also some initial HTTP-requests where sent to the API, using platform Postman [27]. The initial response from the API where quite large. This meant that setting the TypeScript types correct for all values in the response would take alot of time. To mitigate this the Visual Studio Code [28] extension quicktype[29] was used to generate types from the JSON response.

From the 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 was possible from the API. To create a website we need some markup, HTML, to attach the styling to. HTML was created in 1993, back then modularity and component compatibility where not important. The first suggestion to solve this issue was to build a separate HTML-file and then use that as a sub-file to the main HTML-file. This could've worked for a static page with some browsers. But the requirements for the project are that the components can be used for all frameworks too. To solve this the tool LitElement was used. LitElement (3.2.6) is a class that builds web-components (**??**), HTML templates (3.2.5.3) and shadow-DOM (3.2.5.2). Which combined makes for a really good way of building components. LitElement is a class in TypeScript and JavaScript where we choose to use the TypeScript version to keep the program code consistent.

The information from Figmas API is stored as classes of colors, typographies, and components. Where the components class can contain the other two. The strategy was to generate a string that contained the LitElement. Essentaially the program generated code as a string. This string is later written into a new file.

4.2.1 Building the skeleton of the component

To build the HTML inside the LitElement the data-object from Figmas API where run through a recursive function that is run on the component and all it's children (elements).

4.2.2 Styling the Component

For the first "attempt" styling where done in a similar way but because of the nature of the shadow-DOM every CSS-attribute where assigns a property which was reachable from outside the component. This was later redesigned because

of the fact that the user could not add a new CSS-attribute to the component if they wished to. To fix this the CSS-attributes where stored in maps [30] and then pushed in to a style element. Instead of creating a property for each style attribute, only one property for each element is created. If the user wishes to add and/or change the styling of an component they target the element as an attribute to the component and inputs regular CSS. The component then creates a duplicate of the styling map for the targeted element and inserts the new styling attributes to the component.

4.2.3 Variables

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

4.2.4 Open Source

One of Knowits 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. The MIT[31] open source license where then attached to the repository stating that the software is free to use but has not liability or warranty.

4.2.5 Userguide

The program built does not have a graphical user inteface, 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 to the program. To solve this a user guide was created in the form of a README on GitHub[32]. Alon side the prototype itself this user guide was altered after the usability tests.

4.3 Semi-Structured Interviews

This tool is involved with people in many different areas of expertise, form designers to front-end developers to back-end developers. To get a better view of how these people work and create the best tool for them a semi-structured interview model was used [33]. The semi-structure interview is done with a script of questions that are asked to every interviewee. Unlike the structure interview the semi-structured interview allows for further explanation and follow up questions from the interviewer. These interviews where done on 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 effected 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). (lägg till appendix).

4.4 Usability Testing

To make great product it must have great functionality but also great usability. These test where 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 was carried out on nine participants, four the first iteration and five the second. The participants had to have a background in web development, NPM, and Figma. Therefore the participants chosen for the tests where employees of Knowit and students from the interaction and design program in University of Umeå. The test were laid out 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. There after 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 design to component usage for the tool and to determine what was working and not in the user guide. After the tasks where done the questions about the experience where asked. Finally the test administrator opened up for suggestions regarding improvements of the tool or the user guide.

The test script was altered after the first iteration to target more features of the prototype. These features where the use of color and text styles. The questions from the first iteration where still asked. This to confirm that the changes made between iteration one and 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. A number tests will be done with two participants in each test. The two participants is complied by one designer and one developer that work together to get the tasks done. The test participants will get a design of a web page written out on a document. The participants task is then together design and code the website. Once with the tool created in this project and once without it. If they create the page without the tool first or second will be randomized for each test to try to dampened the learning effect. When the test have been performed a statistical analysis, section 3.4.3, will be performed to

see what the data can say about the tool.

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 are free to use and modify under the MIT open source license.

The prototype has been usability tested and has shown indications of being arbitrarily usable for all basic functions.

5.2 Initial interviews

The interviews where done get a sense of how this tool could be structured around the employees work flow.

From the interviews there was found that majority liked the idea and thought that it could be a useful tool. The interviewees that worked with back-end where the one's with most hesitation. There where a lot of uncertainty whether or not a tool like this could be useful. The main concerns were the responsiveness of the components and to have a link between Figma and the components directly.

From the other side of the spectra there more a more positive approach. These are the designers of the almost daily work with Figma. They thought that the tool was interesting and that it could potentially help with communication between them and the developers.

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

5.3 User Testing

When the tool was functional two usability tests where done to ensure that the tool was usable for more than just the author. The thought is that a developer should be able to use the tool with just the userguide (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 for the program.
- When setting up a new Figma document all users had a hard time to what they were allowed to names their document to.
- 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 components 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 quite a high learning curve. They liked the use of pictures in the user guide and suggesting to add 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 in the University of Umeå. In addition to the first iteration task about Figma styles where added. The second iteration also gave a list of flaws.

- 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. This was not understood by the users that the image in 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 incorporate the compilation within the *"convert"*-script.
- Two users found 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 not to be necessary to have them nested.

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 unfortunatly could not be executed.

6.1 The Prototype

The most time and effort for this project went in to creating the prototype. There was 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. To figure this out the research went in to find something that works for static pages. Because if something works natively within HTML-CSS-JavaScript that has a much higher chance of working together with frameworks. The web component was a clear choice.

Web component are as explained in the theory 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 doing it this way is that there is complicated to make these components loosely couple between projects. The LitElement class from Polymer was the solution. This also made the creation of the components much easier. A problem with LitElement was that to use them you need to handle *open imports*. 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 combat that a bundler must be used, such as Webpack or Rollup.This was not optimal but the upside of LitElement still trumped 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?* and *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 that "digs" down to the "youngest" child and builds the

elements from there an goes from generation to generation. This is a good way of building the components but it can become a performance taxing function if the components becomes to big. This has not been tested but there has been indications that there needs to be done optimization of these functions to make the prototype viable for bigger components and projects.

The prototype has some features that unfortunately was not tested. Such as slotting. Where the user can decide a slot in the component where you can insert en element or another component into it. This feature was thought to be necessary to do the required task in later tests which was wrong. The time spent on developing the feature could have gone to testing which would have been more efficient.

6.2 Interviews

The intention of the interviews was 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 of 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 with in some react projects. This was already planned to be done for the components in the prototype but was a good verification.

The interviews was certainly interesting. There was a shown that a tool with this overlap could help with communication shown from most competence groups.

The assumption for the interviews was the people working with back-end would be most exited because with the tool they would not have to make any adjustments from the front-end. When using the tool there are more things that the designer has to think about. How the structure of their component are in Figma not just visually. Also the need of Auto-layout that needs to be used for the prototype to work. Because of this another assumption for the interviews was the designers would be more negative to using this tool.

Both of these assumptions was wrong and from the interviews done the opposite was shown. To be get a statistical significance of this would require more data points, e.g. more interviews from each competence area. This shows that the tool could be used to increase communication between competence areas. Which 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 had 20 weeks and was done during the Covid 19 pandemic meaning that all testing was done remotely. This

also meant that it was harder to get users to do the tests. All test that was made where a walk through for the whole prototype, from design to web component. It was considered to make smaller tests to check individual parts of the system. This was not used because there wasn't enough users and time. Because of the broadness of the prototype it was also highly important that the users could make it from start to finish without any help.

From the two iterations we can see that there were seven flaws found each iteration. At first glance this seems as though the fixes done after the first iteration did not work. This is not correct because the flaws in the second iteration is much different. The test was altered with added tasks in the second iterations which means that the average flaws per task went down from the first to second iteration.

To say that the whole system is user friendly would be an over statement. What the usability tests has shown is that for the basic functions such as setting up a Figma document and converting its components. The tests also shows 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 suppose 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 shows that it is more efficient to develop with the prototype there is a good incentive to invest in developing it to 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 are 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 of the questions could be answered through researching web technologies, creating a prototype that converts Figma components to web components, and run usability test the prototype. As of the last two research questions there was not enough time test them properly.

8 Future Work

Futher developement should be done with:

The next step in the development would be to create a graphical user interface. This would be a lot easier to steer the user in the right direction and making the user experience a lot better. The learning curve for the user could be shortened to be most about using and altering the generated components.

- Gradients - effects - rotation...
- package.json for output
- separat kunna få ut variabler
- Graphical interface
- More general approach, not using Auto-layout
-

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

[3] 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).

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

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

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

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

[8] 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).

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

[10] *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).

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

[12] 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).

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

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

[15] 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.

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

[17] Polymer. (). "LitElement," [Online]. Available: https://lit-element.polymer-project.org/ (visited on 02/26/2021).

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

[19] *Package manager*, in *Wikipedia*, Dec. 26, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Package_manager&oldid=996441348 (visited on 02/10/2021).

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

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

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

[23] R. A. Virzi, "Refining the test phase of usability evaluation: How many subjects is enough?" *Human factors*, vol. 34, no. 4, pp. 457–468, 1992.

[24] 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.

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

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

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

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

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

[30] (). "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).

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

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

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