Technology
Arts Sciences
TH Köln

Design Systems for Micro Frontends

An Investigation into the Development of Framework-Agnostic Design Systems using Svelte and Tailwind CSS

Bachelor Thesis

prepared by

Marvin Christian Klimm
Matriculation number: 11103348

submitted to the

TH Köln - University of Applied Sciences
Campus Gummersbach
Faculty of Computer and
Engineering Sciences

in the degree program
Media Informatics

1. Auditor: Prof. Christian Noss
TH Köln - University of Applied Sciences

2. Auditor: Prof. Dr. Stefan Bente
TH Köln - University of Applied Sciences

Gummersbach, February 2021

Addresses: Marvin Christian Klimm
Am Hepel 61
51643 Gummersbach
studies@marvinklimm.de

Prof. Christian Noss
TH Köln - University of Applied Sciences
Institute for Computer Science
Steinmüllerallee 1
51643 Gummersbach
christian.noss@th-koeln.de

Prof. Dr. Stefan Bente
TH Köln - University of Applied Sciences
Institute for Computer Science
Steinmüllerallee 6
51643 Gummersbach
stefan.bente@th-koeln.de

Abstract

German version below

English Version

This bachelor thesis deals with framework-agnostic design systems in the environment of micro frontends. The properties of micro frontends and design systems are developed individually, to finally get combined. Important disciplines of this research are in particular cybernetics and system thinking, whereby further properties in the relationships between the stakeholders and the design system are highlighted and defined. In addition, based on the highlighted properties, a practically oriented evaluation is prepared, which demonstratively verifies how framework-agnostic design systems can be realized using Svelte and Tailwind CSS. The insights gained from this work can be used for further considerations in other works within the domain, or similar domains. In particular the cybernetic view of systematic design offers potential for further investigations.

Deutsche Version

Diese Bachelorarbeit setzt sich mit framework-agnostischen Design Systemen im Umfeld von Micro Frontends auseinander. Dabei werden den die Eigenschaften von Micro Frontends und Design Systemen indivuell erarbeitet und schließlich gemeinsam zusammengeführt. Wichtige Disziplinen dieser Untersuchungen sind dabei inbesondere die Kybernetik und das System-Thinking, wodurch weitere Eigenschaften in den Beziehungen zwischen den Stakeholdern und dem Design System herausgestellt und definiert werden. Außerdem wird basierend auf den herausgestellten Eigenschaften eine praktisch orientiert Evaluation angefertigt, die demonstrativ überprüft, wie sich framework-agnostische Design Systeme gemeinsam mit Svelte und Tailwind CSS realisieren lassen. Die gewonnenen Erkenntisse dieser Arbeit können zur weiteren Betrachtung anderer Arbeiten in dieser Domäne, oder ähnlichen Domänen, herangezogen werden, wobei insbesondere die kybernetische Sichtweise von systematischen Design Raum für weitere Untersuchungen bietet.

Preliminary Remark

During my practical project and various conversations with other web developers I became interested in design systems and micro frontends. These two topics are relatively new concepts, which already have some literature and discussions, but have not yet been considered together in depth. Therefore, the following work will focus on the field of framework-agnostic design systems in micro frontends. In particular, the technologies Svelte and Tailwind CSS are considered, for which the following prototype repositories were created for evaluation purposes:

- Component library: `https://github.com/netzfluencer/svelte-tailwind`
- Component usage in a Vue app: `https://github.com/netzfluencer/vue-vite-svelte`

Contents

1 Introduction

> *"As application development becomes increasingly dynamic and complex, it's a challenge to achieve the effective delivery of accessible and usable products that are consistent in style."* (ThoughtWorks, Inc., 2019, p. 8)

1.1 Context

New techniques and new frameworks are frequently influencing the development of web applications by offering new solutions for new sets of problems. In the world of frontend development, single-page applications have already achieved great popularity and underlying frameworks like vue.js or react.js are among the most popular projects on Github (2020).

While single-page applications are predominantly delivered as monolithic frontends another backend-inherited architectural approach, described by the term *micro frontends*, has gained interest and adoptions (ThoughtWorks, Inc., 2020*a*, p. 8). Similar to the concepts of microservices, micro frontend based applications compose the final product out of multiple smaller and independent maintained frontends, allowing teams to work and scale more effectively. Normally each micro frontend serves a specific purpose of an application and is owned by a specific team. As micro frontends aim to reduce coupling and dependence between each other, the teams gain more freedom and responsibility for the implementation of their autonomous micro frontend, which opens the need for a global and consistent design approach.

Design systems are also gaining attention (ThoughtWorks, Inc., 2019, p. 8) as a solution to ensure consistent design within growing products by introducing systematic guidance to teams and are already prevalent in many monolithic frontends. Although design systems are occasionally referred to as pattern libraries (or component libraries), these libraries are, in general, only parts of design systems. Design systems consist of interconnected patterns, principles, and guides helping stakeholders to create a product that achieves the product's purpose. Especially micro frontends target to support further dynamics and freedoms in the application development. On the one hand, that context highlights the relevance of a systematic design approach to help ensure unity in a product. On the other hand, design systems introduce some sort of coupling which could evolve an anti-pattern to the idea of micro frontends and therefore advances the requirements for design systems in the context of microarchitectures.

One of these requirements of design systems for micro frontends is to create and deliver patterns, that are usable in every micro frontend by not requiring a specialized environment. That means, that a pattern, like a button, can be used by any micro frontend without the

need for a specific framework setup. Therefore an approach to create *framework-agnostic* design systems with sets of defined patterns is necessary for the context of micro frontends. It is important to mention, that the framework-agnostic characteristic is only relevant for the created patterns, but not for the creation itself. As long as a pattern does not require a micro frontend to use a specific framework it can be considered as framework-agnostic even though the pattern had been created by a framework or compiler during development. A framework-agnostic example could be a CSS-based pattern library, created by using a CSS preprocessor that transpiles SASS[1] to normal CSS and offers each frontend the opportunity to apply these patterns by using CSS classes. Contrary, if this pattern library would also have some dynamic JavaScript functionality, based upon a framework like jQuery, it could not be considered as framework-agnostic anymore.

As patterns within single-page applications tend to be created as components with characteristics like predefined markup, state management, and reactivity, a framework-agnostic pattern library based on CSS would be limiting. Therefore it is beneficial to search for another compatible approach. *Svelte* is a relatively new technology that is experiencing a recent gain in interest within the web community (ThoughtWorks, Inc., 2020*b*, p. 5) and serves the purpose to create components based upon JavaScript-like Code compiled to common (completely framework-less) JavaScript instructions, or alternatively to the standardized custom elements (WHATWG, 2020). These both component formats qualify Svelte as a suitable and worth to inspect tool for the creation of framework-agnostic patterns that are usable in multiple micro frontends of the same product.

Tailwind CSS, a CSS framework that sets up an alternative approach, compared to traditional CSS styling, is also recently rising in the awareness of developers (ThoughtWorks, Inc., 2020*a*, p. 30). In comparison to other popular frameworks Tailwind CSS is currently obtaining the greatest satisfaction ratio among surveyed developers, with a still rising trend, by delivering lower-level atomic utility CSS classes (Greif and Benitte, 2020*a*). In the context of design systems for micro frontends, this utility-first approach might be well suited for the tokenization of design languages and connecting diciplines across micro teams (cf. section 4.3) while creating various frontends with diverse technologies. But the danger of prevailing a framework-agnostic anti-pattern is even higher. This would be the case if development teams are required to set up Tailwind CSS in their own micro frontends in order to use the design system and its pattern library. Although it might appear that using Tailwind CSS results in losing the agnosticism of a design system it could also turn out the opposite way. If Tailwind CSS is used for setting up CSS utilities of the design system, which are then delivered as common atomic CSS classes to the micro frontends the design system would be considered fully framework-agnostic to each micro frontend environment, while also introducing an even deeper systematic design approach into the tokenization and usability of the pattern library forming design language. Still, the micro frontends could optionally rely on the design systems Tailwind configuration to create other frontend-specific utilities, but they would not have to from the technological point of view.

[1] SASS: Syntactically Awesome Style Sheets

1.2 Scope of the Thesis

As presented in the previous section and introduced by the quote at the beginning of this chapter this thesis aims to investigate the dynamic and complex topic of defining, developing, and using design systems for micro frontend environments. There are many directions that would be worth investigating for the numerous use cases of modern web development. For one party comparison metrics of various implementations would be relevant, for others exact long-term investigations of real-world usages in specific domains, and also for others it would be important to take advantage of proven procedure models. As the title of this thesis also indicates the goal of this thesis is at first hand the general investigation of micro frontend design. This is because the subtopics *design systems* and *micro frontends* already have some well-founded, but also few, literature, and a more in-depth combination of both does not yet register any major dedicated work. Therefore, this thesis aims to provide a solid introduction to the development of framework-agnostic design systems for micro frontends by looking in particular at the theory of the system. Thereby known and generally accepted works from systems thinking and cybernetics will be used to generate a suitable picture of design systems in microarchitectures which might also include new insights. Furthermore, the findings are visualized by sophisticated models, in order to provide foundations for further scientific works in other areas of design systems. In addition, the property of framework-agnosticism shall be defined and an evaluation of the findings based on the technologies mentioned in the title and in the previous section, Tailwind CSS and Svelte, will be made with the aim of assessing the feasibility of such implementation.

2 Fundamentals of Design Systems

This chapter elaborates the basics of design systems. It starts by defining the concept of design systems and explaining the reasons for creating a design system. Furthermore, it provides an introduction to the characteristics and elements of a design system.

2.1 Describing Design Systems

Currently, no general definition of *design systems* is present, and within the web community, various understandings are present (Kholmatova, 2017, p. 86). For example, one compares design systems to a Lego bricks box for UIs (Geers, 2020, sec. 1.1.4) while another explicitly exemplifies a negotiation of this comparison (Immich, 2019, p. 35). This section deals with describing the general nature and behavior of design systems which combines multiple ideas and definitions in a big picture.

2.1.1 Determining a Systematic Approach

The term *design system* indicates that the definition of a *system* should be highlighted first. The general system theory (GST) describes a system as follows:

> *"A system is a set of elements in interaction."* (Bertalanffy, 1968)

While the GST is a general definition that is applicable for various domains, there is also a standard definition of a system in the software engineering domain presented by the ISO, IEC, and IEEE:

> *"System: Combination of interacting elements organized to achieve one or more stated purposes."* (ISO/IEC/IEEE 15026-1:2019(en), 2019)

Scheithauer (2017) deals with the question *"What is not a system?"* and exemplifies that a Lego box is not a system, as it is a set of not interacting elements. Furthermore, its attributes are evaluated through simple addition (e.g., the weight of the box).

Although the previous definitions described the elements as "interactive" it is also common to define them as "interconnected":

> *"A set of things working together as parts of a mechanism or an interconnecting network."* (Oxford University Press, 2020)

> *"A system is an interconnected set of elements that is coherently organized in a way that achieves something."* (Meadows, 2008, p. 11)

These definitions are opening another abstract view into systems where elements, connections, and a purpose are defining a system. If these three properties are present, interactivity is also possible. If, for example, a building-instruction guide for a Lego car is added to the elements of the previously mentioned Lego box still no interactivity is present although an interactable system is established which has the purpose to enable someone to build the car.

By projecting the metaphor of a lego box to the domain of systematic design a collection of patterns are lacking a systematic usage behavior. Different stakeholders can still use these patterns but the results and usage-contexts are not secured to be similar. This opens up the next section 2.1.2 in which the goals of shifting to a systematic design approach are presented.

2.1.2 Purpose and Goals of Design Systems

According to Couldwell (2020) (Couldwell, 2020, p. 21), there are 7 main goals for building, using, and maintaining design systems:

1. **Efficiency and speed**: Decreasing the time to design, develop and deploy a product or feature
2. **Consistence and user experience**: Creating predictable and accessible interfaces
3. **Creating stronger brands**: Establishing a strong consistent and maintainable identity throughout a product
4. **Focus on what matters**: Developers and designers are more focused on identifying and solving new problems than looping through the same or similar challenges.
5. **Organisation**: Uniforming development approaches like naming conventions and structures.
6. **Providing a team language**: Naming conventions and consistent terminologies provide help to communicate across the team.
7. **Source of truth**: Setting up highly validated standards for processes and products across the team.

In conclusion to the goals, the main purpose of design systems is supporting and serving the development of products (Geers, 2020, sec. 12.1.1).

2.1.3 The Essence of Design Systems

The definitions of systems from section 2.1.1 raises questions regarding the (interconnected) elements and purposes of design systems. The purpose and goals had been presented in the previous section. This section deals with giving an overview of the elements of design

systems. Design systems are frequently designated as *pattern libraries* offering UI elements and patterns to developers and designers (Immich, 2019, p. 35). Section 2.1.1 points out that a Lego bricks box is not a system and leads to the conclusion that a ***pattern library*** is not a system but can be an element of a design system. Other important parts are the *principles* and ***values*** of a design system, which are shaping the product's design language (Kholmatova, 2017, p. 18).

Every design system's primary aim is the establishment of a ***design language*** that is used by teams to help achieve ***the product's purpose*** (Kholmatova, 2017, p. 37) as well as a coherent user interface (Immich, 2019, p. 35). The goal of the design language is to evaluate a scalable and maintainable ***codebase*** that coherently reproduces the UI (Godbolt, 2016, p. 26) by relying on continuously evaluated ***principles*** and ***patterns*** (Kholmatova, 2017, p. 18). These essences of design systems are promoting the conclusion that design systems are a programmatic representation of a websites or web apps visual language (Godbolt, 2016, p. 27) and therefore are comparable to spoken languages. Furthermore Couldwell (2020) highlights the importance of ***branding*** in systematic design as the essence which is separating one competitor from another competitor by establishing a unique, strong, and focused voice (Couldwell, 2020, p. 40). To conclude, design systems are the result of a *"systematic approach to design"* (Couldwell, 2020, p. 15) while also supporting to keep this design approach. The contained elements are interrelated to each other, which means that design, code, principles, and documentation are cohesively considered by the design systems interactors to build consistent interfaces and experiences. A systematic design approach helps to not solve the same or similar problems in isolation but in unity to create robust design and code solutions (Couldwell, 2020, p. 15).

2.1.4 Qualities of Design Principles

It is common that developers and designers often rely on their own implicit standards to evaluate their work (Suarez et al., n.d.). Design principles, branding guidelines, and system values are not only important essences of design systems; they are also building the foundation of any well-functioning design system (Kholmatova, 2017, p. 46). They are guidelines that cover all design properties where they are applicable (Couldwell, 2020, p. 87), but also can evolve different focuses in various design systems (Kholmatova, 2017, p. 47). This means that one design system could follow a more brand-focused approach while another follows a more team-focused approach. A design system can also cover multiple design properties where one property follows different guidelines than another property (Couldwell, 2020, p. 87). For example, one property could be a marketing-/landing page of a product while another is the product itself. Both might have the need for a shared identity while also serving different goals with the help of suitable guides. Although there are various approaches for establishing design principles in general all qualities of these have the goal to lead to effectiveness (Kholmatova, 2017, p. 49). This section mentions the most relevant ones.

Significant Designations

Kholmatova (2017) exemplifies that verbs like *"simple, useful or enjoyable"* are in general not providing any help in designing a system. It is better to break these descriptions down by explaining their specific meaning in the context of the product (Kholmatova, 2017, p. 49-50).

Tools, not Rules

A design system needs to help the product team to achieve the product's purpose. Therefore it needs to serve the developer as a guide to solve problems. By setting up a *"tools, not rules"* mentality, governance still happens while also supporting freedom and creativity for the team (Clark, 2019). This means that every guide should be usable as a tool in the first hand and not be treated as a forcing condition.

Practical and Actionable

While significant designations are introducing a good understanding of context a principle also should guide actionable advice. This could be for example questions leading the actor to the evaluation of design elements (Kholmatova, 2017, p. 52). To make a principle practicable it helps to show an example where this specific principle already gave guidance. This can be observed in Google's Material Design Guide[1] where a positive example and a negative example are placed next to each other.

Relatable and Memorable

Remembering more than four things at a time is difficult for humans, that's why it is good to limit the number of principles and support recognition by using existing and relevant relations (Kholmatova, 2017, p. 56-57). The use of acronyms can also promote remembering principles. Finally, the most important remembering factor is regular use and referencing of the principles in daily discussion within the team (Kholmatova, 2017, p. 56-57).

2.2 Modeling Design Systems

Currently, there are not many models of design systems present, especially there are no popular models visualizing the behavior of design systems. This section starts by expounding the *Foundations Model* of Couldwell (2020) to visualize the hierarchical layers of elements within a design system. The section then continues to present a suitable model of the basic flows and connections inside design systems by using the system-thinking methodologies of Meadows (2008). This is relevant as descriptions by words and sentences are limited to logical linearity although *"systems happen all at once"* (Meadows, 2008, p. 5).

[1] Material Design Guide: `https://material.io/design`

2.2.1 The Foundations Model

Couldwell (2020) introduces the *Foundations Model* (Fig. 2.1) as a general example for a design system model that serves as a tool to align teams on their terminologies and structure of their own design system. The model is flexible in the used terminologies which means that they can get adjusted to company-familiar labels. As the name indicates the most important layer is the foundation layer which needs to be well thought and established as all other layers are based upon this foundation. The foundation includes the product's and/or company's design principles, values, and branding guides. Onto the foundation different pattern layers follow which Couldwell (2020) categorizes into simple components on which more complex patterns are following.

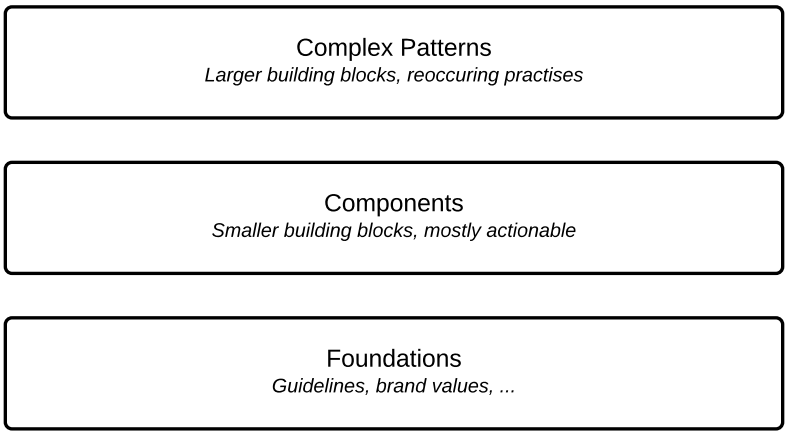

Figure 2.1: The Foundations Model (Couldwell, 2020, p. 84)

This model demonstrates the general hierarchy of elements in the design system, but it does not communicate a system behavior that is modeled by the general connections within the system. To illustrate the behavior another model needs to be created.

2.2.2 Model of Systematic Design

To create a basic model of a design system it is beneficial to look at the *"design system first"* mentality mentioned by Frost (2016) which sets the consideration of adjusting design systems in front of developing and designing web applications or pattern libraries (Fig. 2.2). If something needs to get done in the web application the design system first mentality supports to first look at possible affections in the union of the whole ecosystem.

With the design system first mentality in mind and by using the terminologies of Meadows (2008) the flows within a design system can be modeled (Fig. 2.3). Flows are represented by *"arrow headed pipes which are leading into or out of the stocks"* (Meadows, 2008, p. 18)

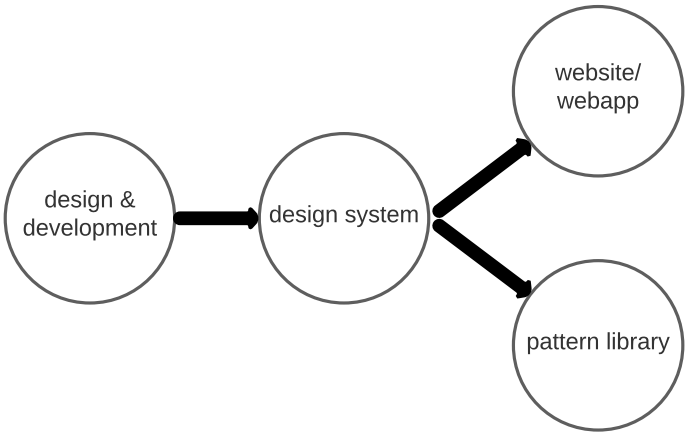

Figure 2.2: Design System First - Mentality (Frost, 2016, Changing minds, once again)

and are responsible for the change of stocks. Stocks are *"shown as boxes"*(Meadows, 2008, p. 18) and are defined as elements of a system that are measurable.

> *"A stock is the memory of the history of changing flows within a system."* (Meadows, 2008, p. 18)

The diagram shown in figure 2.3 therefore models how tasks are getting transformed to patterns and to the codebase of the final web applications codebase. As the stocks of the final implementations change over time it can also happen that older solutions (the stored in the final stocks) are getting removed in favor of newer solutions or new requirements. The clouds in the diagram are indicating the source of an inflow or the *"sink"* of an outflow which can be ignored within the scope of the modeled system.

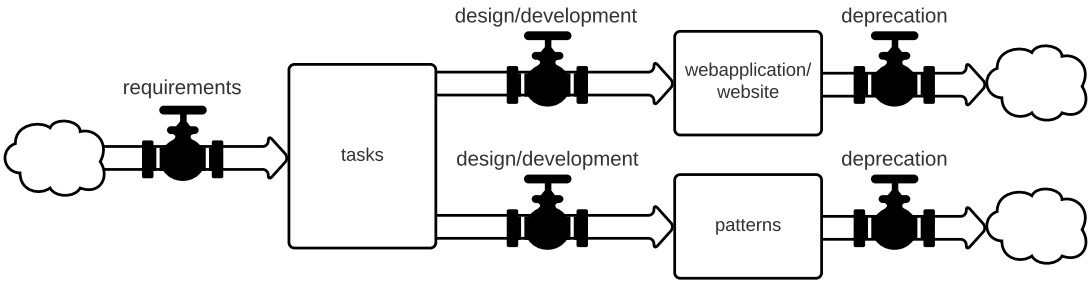

Figure 2.3: Flows in Design Systems

Although the flows of systematic design are modeled there is a crucial lack of important design system elements which were mentioned in section 2.1.3. Therefore the next step is to model these elements into the diagram by also establishing the significant connections of the system. Elements that are missing in figure 2.3 are the brand guidelines, design principles,

and values - all of which can be categorized into the foundation layer by Couldwell (2020). It could be argued that some of these elements also have the characteristics of stocks, as they are measurable in amount with the possibility to experience change over time but further stocks would also increase the complexitiy of the model. Therefore firgure 2.4 shows a basic model of design systems that has the mission to support, as values, branding and principles are changing less dynamically compared to patterns.

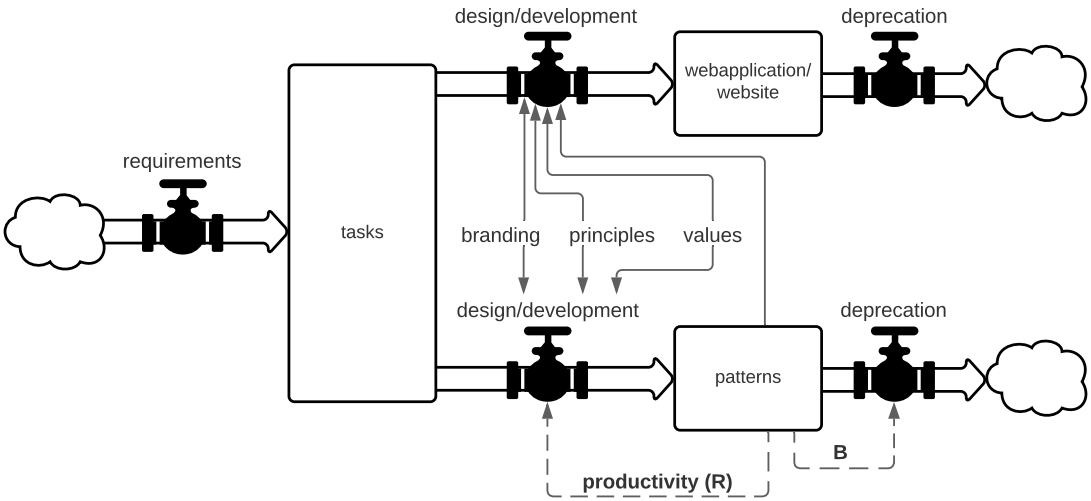

Figure 2.4: Flows and Connections in Design Systems

Figure 2.4 shows the established connections with arrows leading to the flow they are influencing. All solid arrows are representing supporting influences for development/design processes emitted by the elements they are starting. The dashed arrows are indicating specific feedback loops. A feedback loop is present when a changing stock also affects its own inflows or outflows (Meadows, 2008, p. 24). The reinforcing loop (indicated with R) on patterns indicates that the more patterns are existent the more productive other patterns can get created and therefore leads to the *"grow as a constant fraction of itself"* (Meadows, 2008, p. 31). This reinforcing feedback relation also has another important characteristic of nonlinearity. Nonlinear behavior is happening when a dominant loop is shifting to an inverted behavior at some point (Meadows, 2008, p. 94). This is the case for patterns, as it can get more difficult to develop and maintain a collection of particularly many patterns which then is leading to a minimization of productivity of changes to the patterns stock. Therefore a balancing loop (indicated by B) should be established to support deprecations - if possible - under specific stock characteristics. Meadows (2008) describes balancing loops as goal-seeking structures in systems which are on the one hand sources of stability but also sources of resistance of change (Meadows, 2008, p. 30).

3 Characteristics of Micro Frontends

This chapter aims to open up the context of problems and requirements for suitable design systems within micro frontend architectures. The chapter explains the relevant characteristics of micro frontends and continues by discussing popular concepts and approaches for setting up those. The chapter concludes by opening up the problem scope of framework-agnostic design systems in the environment of micro frontends.

3.1 General Characteristics of Micro Frontends

The concept of microservice architecture has been already become a popular approach to implement backend systems. A backend system is divided into smaller services that exist as separate processes and can communicate with each other via lightweight mechanisms, often an HTTP resource API (Fowler and Lewis, 2014, sec. Introduction). Such architecture's particular advantages lie in the targeted independence in the development and deployment of individual services. This means that individual backend teams can work and make changes in system sections isolated from each other. Besides, the smaller services are easier to oversee than monolithic systems and thus promote effective development in growing teams in the long-term.

In single-page applications, this micro architecture can also provide an alternative to the more common monolithic approaches. This concept is currently gaining interest and acceptance and is being disseminated under the term micro frontends (ThoughtWorks, Inc., 2020*a*, p. 8). In micro frontends, the entire frontend is divided into smaller independent frontends that are managed by independent teams. This enables advantages that have a particularly positive impact on team scaling and product scaling.

In the following, the micro frontend key characteristics according to Herrington (2020) are presented and validated through serval other sources like Geers (2020) and Mezzalira (2021) to establish a fundamental understanding of the micro frontend world.

3.1.1 Team Scalability

As the development team scales, it is getting more difficult for team members to overview the whole system, and chances for interference are getting higher (Herrington, 2020). Micro Frontends offer a way to manage system parts independently by smaller teams which break up the complexity of developing features (Geers, 2020, sec. 1). In comparison to a frontend monolithic approach the micro frontend approach allows a vertical team arrangement which

also supports scalability and efficiency for development teams (Geers, 2020, sec. 1.1.1). In a vertical team arrangement, the teams are not grouped by their specific disciplines anymore but by a specific purpose. That means that a team consists out of a small set of frontend and backend developers as well as designers, researchers, and marketers. If a new goal might need to get included in the application a new autonomous cross-functional team can be formed for that specific frontend purpose. These cross-functional teams also have not only the benefit of better scalability but also promote the creation of more creative, effective, and user-orientated solutions as sections 3.1.2 and 4.3.4 will further explore.

3.1.2 Strategical and Tactical Focus

The teams focus on one area without the need to know other application parts Herrington (2020). As already in section 3.1.1 mentioned cross-functional teams have all competencies to develop a specific feature and accomplish a specific goal: The micro frontend architecture is optimized for feature development (Geers, 2020, sec. 1) by enabling teams to autonomously and independently take decisions in choosing, upgrading, and changing their own approaches (Geers, 2020, sec. 1). Therefore, each business domain's technologic restriction is easier to define inside micro frontends, and teams have clear boundaries in which they operate (Mezzalira, 2021, ch. 2). Within these boundaries, the focus of user-centered design increases as the whole micro frontend team takes full responsibility for the final results they create and not only for a specific disciplinary side task (Geers, 2020, sec. 1).

3.1.3 Reusability

According to Herrington (2020) the establishment of contracts between the boundaries and frontend communication mechanisms supports the reusability of any micro frontend through the whole application. Geers (2020) further expands this as parts within a webpage can consist out of various fragments (Geers, 2020, sec. 1.1.1). Fragments are smaller micro frontends that are self-contained and usable in other micro frontends. It might be necessary that a fragment also needs to communicate with the parent page or other fragments on the same page. In this case, a contract of a defined communication procedure needs to be available (Mezzalira, 2021, ch. 2). Contracts help keep the deployment of each fragment independent of other micro frontends and advance the organizational complexity, as section 3.1.5 further describes.

3.1.4 Technology-Agnosticism

Technology-Agnosticism means that every team can autonomously decide on the technology set it would prefer to use to accomplish a specific feature's goal. While micro frontends target that high grade of autonomy from a technical standpoint, teams can still agree on shared practices from an organizational viewpoint. On the one hand, team scalability and solution quality might depend on the use of various technologies. On the other hand, it also helps the whole team use shared practices and technologies to promote knowledge-transfer or team

member shiftings (Mezzalira, 2021, ch. 2). Furthermore, it is important to consider that technology-agnosticism is only realizable to a specific level: In the web, micro frontends still rely on key technologies, protocols, and standards like HTML, CSS, JavaScript, and HTTP. In general technology-agnosticism in micro frontends considers another level of abstraction, which implies mainly using various web frameworks, CSS preprocessors, and paradigms within micro frontends. Therefore the term framework-agnosticism is also often mentioned in the same relation as technology-agnosticism (Geers, 2020, sec. 12.4.1) and will be further investigated in section 4.2.

3.1.5 Complexity

Mirco frontends aim to reduce the complexity of each individual frontend and therefore introduce complexity in the organizational layer of the architecture (Geers, 2020, sec. 1.4.3). Micro frontends' organizational layer ensures that the subsystems are connected to the complete system while also not limiting the gained abstraction in the subsystems themselves. This includes tasks like autonomous deployment mechanisms that allow continuous integration and testing of recent micro frontend systems (Mezzalira, 2021, ch. 2), or implementing failure procedures if a fragment is due to any reason not available (Mezzalira, 2021, ch. 2). Also, as a micro frontend architecture requires a commitment to *"create many small things rather than one large thing"* (Jackson, 2019) more aspects need to get managed: like repositories, pipelines, servers, domains, etc. (Jackson, 2019).

3.1.6 No Standards

The micro frontend architecture has not industry accepted standard and varies in its approaches (cf. section **??**). Therefore decisions need to be taken under the consideration of the specific use cases of a product (Geers, 2020, sec. 2). That requires strong research and the ability to correct system development through serval evaluations. There are various frameworks that help implement specific micro frontend architectures, but these options need to be weighed.

3.2 General Domain of Micro Frontends

This section investigates the general context in that micro frontends operate. As the specific implementation of micro frontends is always dependent on the use cases of a product, this investigation abstracts into a bigger picture with the purpose to show when micro frontends should get considered in architecture development and when they should not. Characteristics like team scalability (cf. section 3.1.1) and strategical/tactical focus (cf. section 3.1.2) have already indicated that micro frontends are an architecture that aims to make project scaling easier (Geers, 2020, sec. 1.4.1). Therefore a project without the need of scaling itself and its development team would not benefit from micro frontends: If a team is small already (and also stays small), it already has the benefits of quick communications and decisions. This circumstance leads to the conclusion that micro frontends are starting to

be beneficial when teams are bigger, and scaling is required. Therefore it can be concluded that micro frontends only make sense for medium to large-sized teams (Mezzalira, 2021, ch. 2). Geers (2020) exemplifies that a good team size would consist out of around 5 people, grounded on the *two-pizza rule* of Jeff Bezos that states that a team is too big when two big pizzas are not enough to satisfy the team. Another characteristic of the context of micro frontends is that micro frontend architectures are more suitable for web applications, even though the architecture is not limited in theory to that context. Geers (2020) exemplifies this by showing that the web is much more capable of complex architectures than native apps published to an app store, as native apps need to be offered as monolithic client systems that can be reviewed by the store owners (Geers, 2020, sec. 1.4.4).

> *"The micro frontend architecture generally addresses the domain of medium- to large-sized web applications, regularly created by more than 10 people."*

4 Design Systems for Micro Frontends

In the previous chapters, the fundamentals of design systems, the characteristics of micro frontends, and utility-first CSS concepts have been introduced. This chapter combines the previous topics by elaborating the requirements for system design in micro frontends and introducing various connections between the topics. Furthermore, it investigates how the technology-agnostic characteristic influences the creation of a proper design system and opens up a discussion regarding the usage of frameworks in micro frontend design. In the next step, a cybernetic investigation into micro frontend design is done to identify the interconnected behaviors inside the system design and take advantage of these discoveries to create more responsive and feedback-looped design systems. Finally, the concept of utility-first CSS is put into relation to evaluate cybernetical enhancement possibilities through utility-first CSS usage.

4.1 An Investigation into the Context

In this section, the general context of design systems in the environment of micro frontends is investigated based on the characteristics of micro frontends (cf. chapter 3) and a stakeholder analysis that aims to treat all potential persons or person groups that are relevant for the product creation in the domain of micro frontends (cf. section 3.2). As the stakeholder analysis method normally addresses specific projects in specific environments to specific times, this investigation will rely on the abstracted stakeholder analysis according to Wittmann (2015).

4.1.1 Stakeholder Analysis

Introduction

This section deals with the potential persons and person groups that are involved in systematic micro frontend design by using an abstracted stakeholder analysis approach (Wittmann, 2015, p. 56) that is based upon the stakeholder map presented by Lobacher (2015).

According to the ISO stakeholders are defined as follows:

> *"Individual or organization having a right, share, claim or interest in a system or in its possession of characteristics that meet their needs and expectations"* (ISO 9241-210:2019-07(en), 2019)

And the stakeholder analysis is defined according to the DIN as follows:

> *"Analysis of the project participants regarding their influence on the project and their attitude (positive or negative) towards the project."* (DIN 69901-5:2009-01, 2009, translated from german)

Wittmann (2015) highlights in his stakeholder analysis for the creation of a procedure model the circumstance that the stakeholder analysis, in general, is used in the context of a certain project. He also shows that it is possible to shift this method into another, more general, abstraction level where the scope is located in *"in a sense of a meta-level"* and therefore allows his model to be applicable into multiple contexts (Wittmann, 2015, p. 56). This approach can also be projected from procedure model development into an investigation of a specific type of system in a specific type of technical environment as the upcoming analysis demonstrates.

The analysis is done in two steps. In the first step, a rough stakeholder map is modeled that introduces the stakeholders and their relation to the context. In the case of Wittmann (2015) his procedure model is the center of the map and around it, the stakeholders are placed as labeled circles under consideration of the following properties size and distance (Wittmann, 2015, p. 57). In relation to the design system these properties shall be defined as follows:

- **Center**: The design system
- **Size**: The larger the circle, the more responsibility and influence the stakeholder has for the success of the system
- **Distance**: The closer the circle is to the center, the more the stakeholder interacts with the system

In the second step, the rough stakeholder map is refined and the stakeholders are described in more detail.

Furthermore, it is necessary to define *"the success of the system"* to be able to evaluate the level of responsibility of a stakeholder. Therefore the goals of a design system (cf. section 2.1.2) need to be considered in conjunction with the characteristics of micro frontends (cf. chapter 3); summarized as follows:

> *The design system needs to support and serve the development of diverse micro frontend purposes, team-workflows and various selected technology stacks.*

Rough Analysis

In figure 4.1 the three key sources of stakeholder needs are presented. According to Couldwell (2020) the business, the team and the user needs are necessary to consider to address all relevant stakeholders (Couldwell, 2020, p. 30-34). For a rough stakeholder mapping it is therefore helpful to model the following stakeholder groups:

- **The Product Team**
  Has the responsibility to develop the final product by using the design system. There-

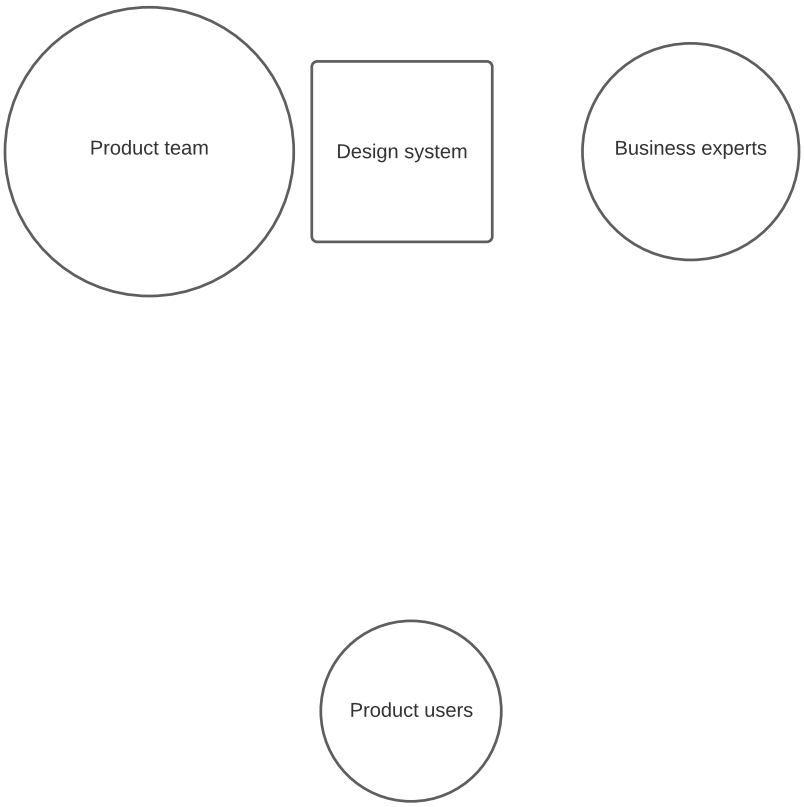

Figure 4.1: Rough Stakeholder Analysis

fore it has a close relationship to the design system. It is also the department that will benefit the most from the tools of the design system and consists of the engineers and designers (Couldwell, 2020, p. 31)

- **The Business Experts**
  These are the persons which take care of the company and the product's economic success. They include persons like product managers, leaders, as well as heads of departments and have an interest in the return of investment for creating a design system. Therefore the initial acceptance of this group is necessary to create and maintain the system. As the design system also aims to unify the company in regards to language and product understanding (Couldwell, 2020, p.31) this group also frequently comes in touch with the system.

- **The Product Users**
  These are the people that are using the final product for a specific purpose. They do not directly interact with the design system but the results created through the design system. Also while the primary aim of a design system is to support the purpose of a product this mission can only be successful if it influencing the developed product in

a positive way. Therefore it is relevant that the users are also experiencing benefits through the systematic design.

Refined Analysis

As the rough map in figure 4.1 has introduced the main stakeholder groups of a design system the next step is to refine the stakeholders of a design system in the context of micro frontends. In figure 4.2 the refined stakeholder map is modeled and a new notion is introduced to highlight the autonomous interests of the micro teams in the microarchitecture. Stakeholders are grouped by a labeled rectangle that represents organizational relations. The preferred team-organizational approach, when setting up a micro frontend architecture, is the alignment of the teams according to essential parts of the product itself (Geers, 2020, sec. 13.1). That concludes that not only the frontend team is divided regarding the subdomains of each micro frontend, but also the whole company structure gets aligned into the subdomains. Therefore the micro teams shown in figure 4.2 are not called micro frontend teams as these teams are not only responsible for the frontend of a specific subdomain of the product but also for the backend and business cases. In the map two micro teams are displayed and shall be recognized as follows:

- **Micro Team X**
  Illustrates any micro team out of the total quantity of N micro teams in the architecture.
- **Micro Team N**
  Illustrates every other micro team of the quantity N that is not X

Although the possible types of stakeholders are the same for every micro team it is important to consider the different teams in the stakeholder map. They all consist, in general, out of the same property values, but due to the nature of the micro frontend architecture, they will follow different purposes by working in different workflows which is an important characteristic to all team stakeholders of the design system. Based on the stakeholder map in figure 4.2 a specific description of each stakeholder type is presented in the following and sorted by importance for the success of the design system:

- **Design System Manager**
  The design system manager (or also the management department) is the stakeholder which has most important for the design system and therefore has the most influence as well as the most responsibility for the design system. The stakeholder manages that the design system achieves the goals mentioned in section 2.1.2 in quality and reliability. It is also possible to introduce other design system responsibilities like a dedicated pattern creation team or an own product team for the design system. In this thesis, the focus is set to a decentralized approach due to the investigated benefits of the cybernetical enlightenment presented in section 4.3. Therefore the primary tasks of the design system manager are to keep the design system clean and beneficial for all teams, by first reviewing the system, the micro frontends, and feedback of stakeholders and then evaluating general improvements. The design system manager

is also the initial initiative when the design system is started to get built and the primary instance for setting up the general system behaviors and connections, so definement of principles, values, and branding as well as the development of patterns and code is influencing all teams. The manager also has a high interest in cross-team knowledge transfer. The manager has his skillset rooted in design and development (Couldwell, 2020, p. 115).

- **Micro Frontend Architect**
  The micro frontend architect takes the most technological responsibilities of a traditional frontend lead developer. This includes setting up the general micro frontend environment in its coded properties as well as deployment guidelines. He can also be the person which sets up general deployment procedures but due to the aimed autonomy of each micro frontend, it is more suitable to introduce a specific development operations expert to each micro team. The micro frontend architect mostly aims to establish a robust ecosystem where the micro frontends are living and able to autonomously can get updated. The architect also manages that the performance and quality of the final composition are guaranteed and that fallbacks are available when services or frontends are not working.

- **Frontend Developer**
  Frontend developers *"care about a unified code, version control, consistency, performance, organization, efficiency, and naming conventions."* (Couldwell, 2020, p. 31). The design system offers tools like guidelines and usable patterns to them which need to help by the mentioned aspects regularly. Also, the frontend developers can be considered as one of the primary user groups of the design system, while they also influence the development of it. This influence can happen directly by implementing solutions into the design system, or by giving feedback and requirements to other responsibles. In the context of their tasks, they also benefit from being able to use the most suitable or preferred technologies in a manageable environment.

- **Designer**
  Designers *"care about aesthetics, brand, typography, color, user experience, and shipping beautiful products, which stay true to their original version."* (Couldwell, 2020, p. 31). Typically designers take care of the planning of a part of the product that fits the user's needs. Designers do not like to be limited in creativity, while also prefer an exact reproduction of the designs they deliver to developers (Couldwell, 2020, p. 31). The designers are like the frontend developers also one of the primary users of the design system, while they also might work on design system improvements.

- **Backend Developer**
  The backend developers share the aspects mentioned for the frontend developers. The difference in the context of design systems is not only that they are working on serverside-functionalities, but also that the design system primarily addresses the visual language of a product. This does not mean that backend developers might not benefit from the design system. For example, they might still have interests in the values of a product, so they are able to align their focus also. Communication with

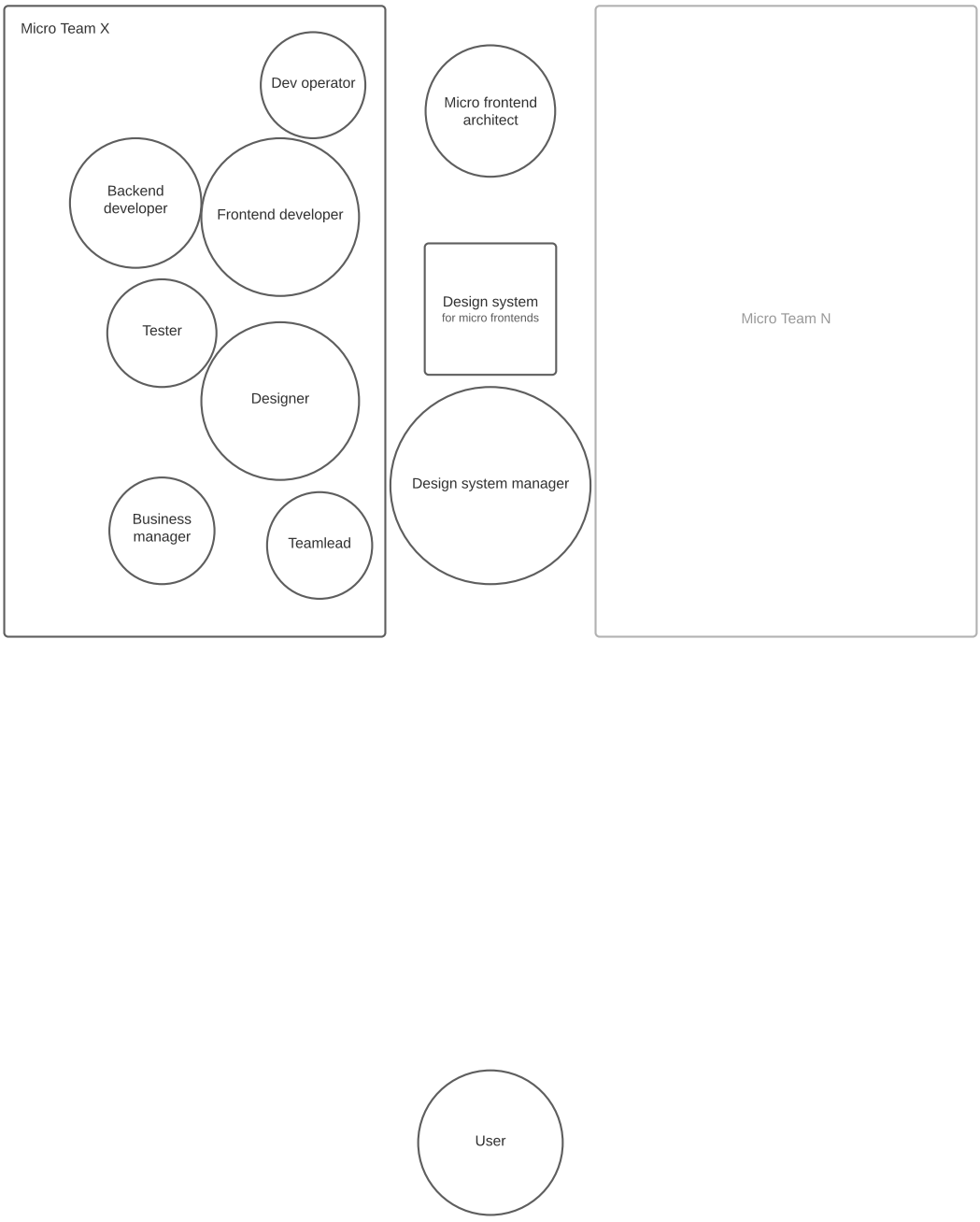

Figure 4.2: Refined Stakeholder Analysis

frontend developers is also important and staying consistent with naming conventions cross-disciplinary help.

- **User**
  The user has an interest in a product that is fulfilling the needs in a unified and

consistent way. The user is not interacting directly with the design system, but to remember: every design system's primary goal is to support the purpose of a specific product. Therefore the user is an important indicator when it comes to the final assessment of the success of the design system. Observations that are taken from the user can be related to the design system.

- **Development Operations Expert**
  As already mentioned in the design system manager paragraph it is recommended to have custom deployment procedures in each micro frontend. As the deployment of a micro frontend will not live in its own ecosystem but rather in a shared ecosystem, the expert needs to have requirements and guidelines that help to set up compatible pipelines and configurations. A DevOps expert therefore is interested in systematic guidances of deploying the design system and developing a infrastrucutre that is well suited for the elements of micro frontends.

- **Teamlead**
  The teamleads primary interests are grounded in efficient and predictable development times of the micro frontend releases. They are often the key decision takers especially when it comes to investing in researches and new development tools. They are organizing the team tactically and therefore due to the close team relation are also close to the teams working habits like systematic designing and developing.

- **Tester**
  The tester has a special interest in finding errors and ensuring the quality of created solutions. Besides the manual testing of new implementations, the tester also sets up automatic tests to ensure that one implementation does not break another implementation. Testing prohibits that bugs, wrong or incomplete implementations are reaching the release to the users. There are many topics in that the tester can take advantage of a design system. For example a tester needs to be able to compare the created results against the principles of the design system, like performance or appearances.

- **Business Manager**
  Based on Coulwells statement regarding product managers, business managers *"care about how fast the team can ship products, and the impact the work has on business goals, sales reports, and analytics."* (Couldwell, 2020, p. 31). They bear the main responsibility of the whole business and are key deciders when it comes to scaling, department budgeting, and financial alignment.

4.1.2 Environmental Constraints

Normally the context of use refers to the users, goals, tasks, tools, and the physical as well as the social environment of a certain product (ISO/IEC/IEEE 9241-11:2018(en), 2018). As section 4.1.1 already presents certain dimensions, other environmental properties are still missed. Likewise, the stakeholder analysis is done abstractly the environment of the design system also can only be elaborated abstractly, as no certain product is considered. Therefore the environmental constraints are mostly based upon the domain and characteristics of micro frontends (cf. chapter 3) aswell as the general goals of design systems (cf. chapter

2). Furthermore, according to Geers (2020) it is *"important to provide ways for people to exchange knowledge between teams"* in micro frontend design, while also highlighting the autonomy of each micro team.

4.2 About Framework-Agnosticism

The usage of frameworks in web application development has upsides and downsides. On the one hand, a framework delivers features and foundations into the development of an application, and on the other hand, it constraints the development team to a specific predesigned foundation that is not designed specifically to the goals and requirements of the final product, while also bringing weight to the overall deployed codebase. Due to that circumstance, teams need to decide which framework does fulfill the product's vision the best by comparing the advantages and disadvantages of several relevant frameworks. The micro frontend architecture allows the use of several frameworks within a system, where each subsystem addresses a specific purpose and is capable of selecting its own technology-stack. As the design system is the element applied to each micro frontend development process, it is important to consider establishing a framework-agnostic characteristic.

In computer science, the term agnostic is defined as in the following:

> *"Agnostic: Denoting or relating to hardware or software that is compatible with many types of platform or operating system."* (Oxford University Press, 2020)

This definition concludes that the adjective framework-agnostic describes an entity as compatible with many types of frameworks:

> *"Framework-agnostic design system: A design system that is compatible with many types of frameworks."*

This section deals with framework-agnosticism and expands the relevance in the domain of micro frontend design.

4.2.1 The Relation between Design Systems and Frameworks

Frameworks are commonly used in various disciplines and domains and therefore also appear in different variants, which can be defined and summarized as in the following:

> *"Framework: a basic structure underlying a system, concept, or text."* (Oxford University Press, 2020)

In the domain of web development, frameworks are commonly established and used as a system of concepts, functions, and oftentimes also code to constraint an application into an environmental structure that promotes the application's development.

Every design system sets up an environment of systematical creation for solutions and purpose achievement. Through the establishment of principles, patterns, and the design

language, a design system lays the foundation for the user interface creation of a specific product and therefore also fits into the definition of a framework. In monolithic frontends, such a systematic design framework (design system) can be created constrained to the application's selected technology stack, or otherwise, a suitable technology stack might also get selected upon the design systems framework. Therefore design systems and other application frameworks rarely interferer with each other, but in the domain of micro frontends, technology stacks are allowed to shift more dynamic and freely as micro frontends aim for high autonomy with the characteristic of technology-agnosticism (cf. 3.1.4). This challenge of serving different types of micro frontends is modeled in figure 4.3 where the design system takes influence on every micro frontend while every micro frontend is also based upon different application frameworks.

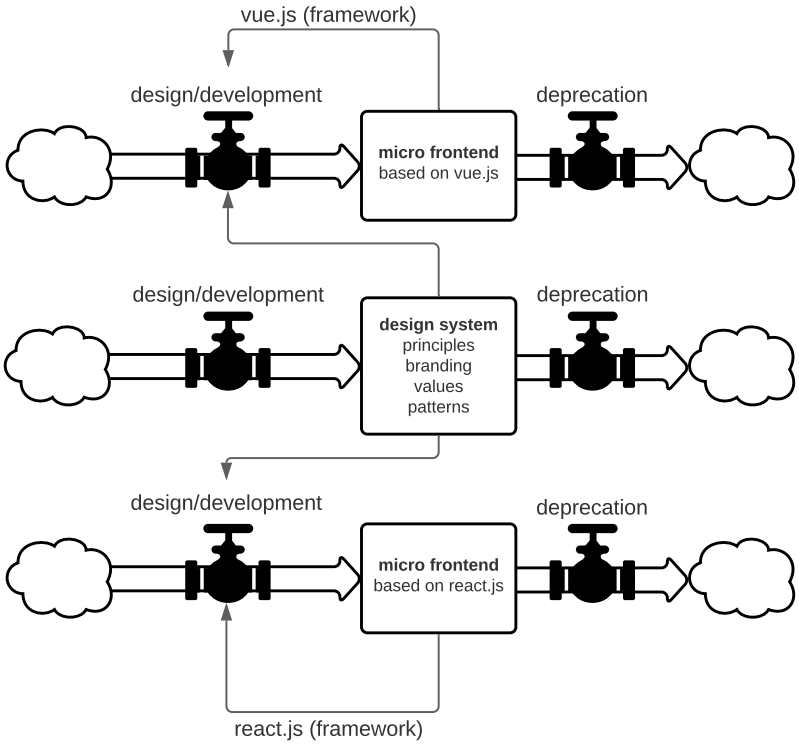

Figure 4.3: Frameworks and Design Systems Relation

While figure 4.3 already indicates that a framework directly influences the development of a frontend in combination with the design system, it does not explain why its influence is also rooted in the frontend itself. This is due to the reason that once an application framework like vue.js or react.js is introduced to the development process, it also requires the frontend to contain and keep the specific structure it relies on. Therefore the design system needs to be compatible with each framework in the micro frontend architecture. If a framework like vue.js or react.js gets applied to the development process of the design system's patterns, it will interfere with the other front-end frameworks. Therefore the design

system would be unusable for diverse technology stacks and couldn't be seen as framework-agnostic. As a well-prepared set of patterns is crucial to the quality of every design systems (Couldwell, 2020, p. 142), a framework-agnostic approach is needed and will be expanded in the upcoming section 4.2.2.

4.2.2 From Frameworks to Compilers

> *"Frameworks are not tools for organizing your code, they are tools for organizing your mind."* (Harris, 2019, Founder of Svelte)

While most web frameworks affect the final size and performance of an application due to the code they are including Harris (2019) highlights the idea that a web framework does not necessarily need to be binded to the final codebase of a deployed product but rather can live in the stage of code development through the use of compilers. This idea can be projected to pattern development by compiling finished patterns to a web standard (for example JavaScript instructions, as by Svelte, or webcomponents) that is agnostic to other frameworks.

4.3 Cybernetically Enhanced Design Systems

This section aims to highlight the cybernetical properties of design systems in the context of micro frontend design. The discipline of cybernetics was first introduced by Norbert Wiener who worked during the second world war on an anti-aircraft predictor and published in 1948 the book *Cybernetics; or, Control and Communication in the Animal and the Machine* that gained world-wide attention (Friis et al., 2009, ch. 20). Although cybernetics deals intensively with the behavioral complexity of systems (Ashby, 1957, p. 1), definitions regarding cybernetics vary (Friis et al., 2009, ch. 20). The systematic micro frontend design is due to its environmental requirements a discipline which enables some kind of complexity for unforeseeable design possibilities while also targeting the general goals of system design (cf. sec. 2.1.2). This means for example that the autonomous nature of micro frontends, on the one hand, allows developers to be more creative and flexible in taking decisions. On the other hand, this case demands a global systematic design approach that is applicable to many unforeseeable varieties. While this cognition implies risks like threatening the product's consistency or obstructing the share of knowledge between micro teams it also holds the chance to level up design systems to a layer where they become the environments of cross-discipline and cross-team thinking, promoting creativity and intelligence of the sum of all involved members. With this idea in mind, the upcoming subsections introduce the science of cybernetics in projection to design systems for micro frontends. Furthermore, the section ends by investigating the possibility to level up system design with technology in a cybernetic and systematic adaptable occurrence.

4.3.1 Introduction to Cybernetics

Based on the Greek word for *"steersman"*, cybernetics was first defined by Norbert Wiener and Arturo Rosenblueth to describe the field of regulation, control, and communication of information processing systems (Wiener, 1948, p. 11). This work is now considered the origin of cybernetics and has influenced many fields of study until today. Cybernetics is an extensive science that can be applied in various forms as long as information-processing systems are the subject of investigation. In cybernetics an information-processing system is not defined by technologies or representations but simply seen into a looping circular sequence that allows processing an input regarding a goal, then sensor the output, and compare the output against the intended goal, with the ability to adjust its own behavior based on feedback generated from this comparison. This fundamental characteristic is known as the negative feedback loop which is shown in figure 4.4 and illustrated by Jackson (2019).

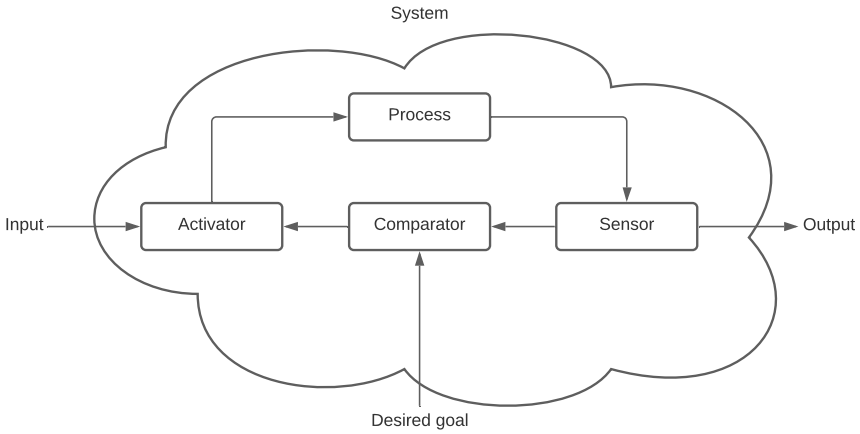

Figure 4.4: A Closed-Loop Feedback System by Jackson (2019)

The model shows that a cybernetic system works with two defined parameters: the *input* and the *desired goal*. As long as the system has these parameters it is able to work effectively (Jackson, 2019, ch. 6) and self-maintained through the interconnections shown in the diagram. In the following the elements of the system that are crucial (Jackson, 2019, ch. 6) in cybernetics are described:

- **Sensor**: Detects changes like to the current output, but also other optional parameters
- **Comparator**: Compares the current outcome and the desired goal (desired outcome)

- **Activator**: Reacts to the discrepancy feedbacked by the comparator by taking decisions that influence the current outcome regarding the intended goal.

Looping through the elements enables the system to continuously register movements away from the goal and regarding the goal, which allows the system to automatically correct itself (Jackson, 2019, ch. 6). As an example of such a feedback-based system a heating system with a thermostat often gets mentioned [(Jackson, 2019, ch. 6), (Meadows, 2008, p. 36)], where the thermostat is set to a specific desired temperature, senses the environmental temperature, and based on the feedback of the comparison of these both temperatures adjusts the heating-output. It is important to mention that the discussed model of feedback systems is just the basic concept and that more complex systems can rely on various types of feedback loops that are influencing the behavior of the system in varity. These types of systems are also considered under Ashby's Law and in *second-order cybernetics* that are discussed in the next sections.

4.3.2 Ashby's Law

In the fifties, the system theorist W. Ross Ashby published the book *"An Introduction to Cybernetics"* which gained international interest in cybernetics through Ashby's description of the law of requisite variety. Furthermore, Ashby (1957) highlights two other principles that are relevant to cybernetics, first the black-box theory and then the negative-feedback loop. As the negative feedback loop has been already presented in the previous introduction it will be skipped in this section, but all three principles are forming a framework of cybernetics that is applicable to all possible types of machines, *"whether electronic, mechanical, neural, or economic, i.e."* (Jackson, 2019, ch. 6).

The Black-Box

In figure 4.5 the concept of the black box according to Ashby (1957) is shown. Essentially the black box is a system where internal mechanisms are not available to the evaluation of the system's behavior. Taking advantage of the black box theory can help understand complex systems, while a traditional analysis would fail due to the high amount of elements and connections occurring. When viewing a system in a black box the experimenter is not breaking down elements in the system itself to evaluate its behavior but emitting various types of inputs into the system. After the input has been processed by the system the output is compared with the previous input. Other examples of inputs follow the procedure until the experimenter system is able to conclude behaviors of the black box system.

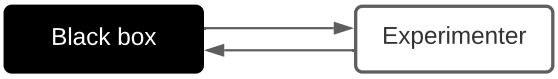

Figure 4.5: Black-Box System and Experimenter System (Ashby, 1957, p. 87)

Furthermore, Ashby (1957) highlights, that through this process of emitting an input and comparing the result a cybernetic feedback system is established, which is capable to identify behaviors based on the variety of the emitted inputs.

The Law of Requisite Variety

Ashby's law is until today one of the fundamental achievements of cybernetics and states that only *"variety can destroy variety"* (Ashby, 1957, p. 207). Variety hereby can be described as the sum of distinct states in which a system can exist which would be for a simple switch the variety of *on* and *off* (Krishnamurthy and Saran, 2007, ch. 2). According to Ashby, this essentially means that a controller must be at least as complex as the problem it is designed to control: it requires complexity to handle complexity and solve a problem. Prof. Peter Kruse, therefore, summarizes the law as follows:

> *"Wherever we have a highly complex dynamic problem system, we need at least an as complex dynamic solution system; if we don't have an equivalent complexity, we are not capable of solving it [the problem system]."*(Kruse, 2007, translated from the german transcript)

4.3.3 Second-Order Cybernetics

In section 4.3.1 and section 4.3.2 the first fundamental principles of cybernetics have been introduced. Between the sixtieth and seventieth the ideas of Heinz von Foerster which he himself summarized under the phrase *cybernetics of cybernetics* gained attention (Jackson, 2019, ch. 6) and created a movement within cyberneticians called the *second-order cybernetics* (Heylighen and Joslyn, 2001, p. 3). The first principles from the fifties and sixties, which were mostly concerned with circular causal processes, e.g., control, negative feedback, computing, adaptation, then became summarized under the term *first-order cybernetics* and introduced two topics of cybernetics. The idea of second-order cybernetics was to take advantage of cybernetical methods and principles by applying them onto the cybernetical observations themselves, meaning that the observers and cybernetic systems also form a cybernetic system (Heylighen and Joslyn, 2001, p. 3-4). Von Foester defines the orders of cybernetics as follows:

> *"I submit that the cybernetics of observed systems we may consider to be first order cybernetics; while second order cybernetics is the cybernetics of observing systems."* (Jackson, 2019, quoted in ch. 6)

This second-order thinking also highlighted the fact that various observer systems, due to their own properties and behaviors, perceive information differently and therefore judge the behavior of an observed system in variety. Furthermore, when various observers act with a system it is important to *"pay attention to the circular processes in which observers interact with what they observe"* (Jackson, 2019, ch. 6) by also assume that general objectivity can not be specified in advance. It is much more important to establish languages that act as connections between the observers and through these connections disagreements and

agreements can be communicated to evaluate a general consensus between all parties in regards to the observed system. In relation to Ashby's law Kruse (2007) shares the idea that interconnections of different cybernetic systems within a system are leading to disturbances within the system which then allows the different parties to *"become creative"* in introducing solutions that are in general applicable to the system.

4.3.4 Thinking Cybernetical in Micro Frontend Design

In chapter 2 the general concepts of design systems based on various sources have been introduced, showing that a design system is strongly considered to be a tool for successful system design handed to the product team. Theoretical design system views can often neglect cybernetic principles, especially those of the second-order, by focusing more on the non-living elements of the system than the behaviors that are in existence through human involvement. While in practice, however, things like circular correction processes or varieties are present during system design, the human origin of these processes is often not considered as part of the design system itself. As in section 4.1.1 a stakeholder analysis of design systems for micro frontends has taken place, introducing various types of persons that are important to be considered when it comes to establishing such a system, this section aims to highlight the need for human involvement as an element of the design system itself, based upon cybernetic principles. Furthermore, in regards to micro frontend design, an advanced cybernetic approach is modeled that considers the possible disturbances, interconnections, and interactions between the design system and its influential stakeholders based upon second-order cybernetics. Interactors are also part of the system and are present in their own variety. There are designers, developers

The Interactor's Relative Awareness

Based on the design system model in figure 2.4 and the feedback loop in figure 4.4 it can be emphasized that a human, normally seen as a user or creator, not only interacts with the design system but also is an element of the system itself. Figure 4.6 demonstrates this circumstance by evaluating the negative-feedback loop in the context of design systems, setting the human interacting with the system into the system's boundaries.

Especially in micro frontend design, it is important to highlight, that a design system needs to be capable to be adaptable to every occurring micro frontend environment. This adaption is on the one hand dependent on the framework-agnostic property but on the other hand also on many other varieties. Design systems, as presented in chapter 2, themselves are initially simple sets of rules and pattern collections that cannot perform any task on their own and certainly cannot adapt themselves to shifting environments. This can be demonstrated by the negative feedback loop in figure 4.6 that indicates that the interactor contains all cybernetic properties. There are no strong mechanisms in design systems, besides the human, that are able to sensor, compare and decide to shift systems behaviors and outputs regarding selected goals. There might be tools, like linters or accessibility checkers, which can help a human to work more efficiently in for example sensoring and comparing

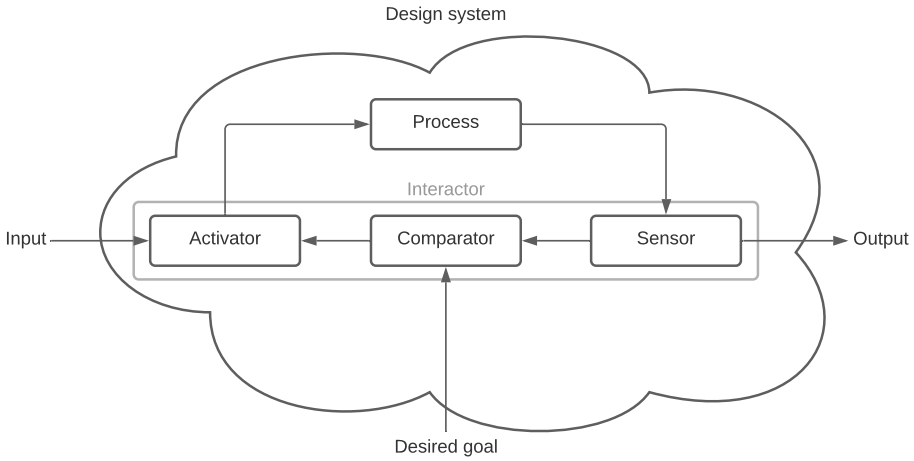

Figure 4.6: The interactor's roles inside the design system

outputs, but the responsibility of extracting and evaluating new sensoring properties or decisions would be still onto the human. Next, it is important to note that the interactor can be any stakeholder highlighted in section 4.1.1, from any possible micro team of the quantity N, interacting with the design system. This circumstance shows the relevance of second-order cybernetics (cf. section 4.3.3), as each interactor relies on his own senses and mind that lead to subjective occurrences inside the design systems.

By considering Ashby's law of requisite variety the introduction quote from chapter 1 is represented again:

> *"As application development becomes increasingly dynamic and complex, it's a challenge to achieve the effective delivery of accessible and usable products that are consistent in style."* (ThoughtWorks, Inc., 2019, p. 8)

This quote has been mentioned by ThoughtWorks, Inc. (2019) with the aim to highlight the problem field that design systems address to offer solutions to. Furthermore, micro frontends also seem to follow systematic concepts with the aim of reducing the necessary frontend complexity to the developers of a product, by dividing it into organizational sections. This approach can be mapped to the self-organizational characteristic of hierarchical systems (Meadows, 2008, p. 85). In order to create *"highly functional systems, hierarchy must balance welfare, freedoms, and responsibilities of the subsystem and total system"*(Meadows, 2008, p. 85). Traditional design systems offer two approaches in getting developed. One approach is considered as the *centralized model*, where a specific design system development team is the main creator of the system, while the other is classified as the *distributed model* (Kholmatova, 2017, p. 157), where the creation responsibility is dependent on everyone within the team. In the world of micro frontends, it also needs to be considered that the *"dynamic complexity"* of the whole product increases even more due to the autonomy and general domain (cf. section 3.2) even further. Therefore, according to Ashby's law, it is

necessary to establish a design system that is in minimum as complex as the presented problem field of micro frontend design itself.

Although figure 4.6 shows that the variety of the individual interactors already affects the global behavior of system design in variety an organizational approach counterfeiting the problem-variety is still needed, as the variety of the problem environment consists within black-boxed interconnection (cf. section 4.3.2) while intended design systematic connections are not yet evaluated. As Kruse (2007) suggests:

> *"If our world is now becoming more and more complex due to interconnectedness, one can say: the only solution we have is complexity through interconnectedness;"*(Kruse, 2007, translated from the german transcript)

For the scope of this work, the human behavioral interconnections will be continued to be considered within a black-box system as investigating these interconnections further would open another field of science. In regards to second-order cybernetics it is already shown that there is a remarkable influence of each interactor in systems and figure 4.6 also supports this statement. So the question which gets open up in the following is orientated regarding the mentioned statement of Kruse (2007): *Where and how are connections initiatable in design systems, to promote consistent and efficient design executed by various humans within micro frontends?*

Interactors Connections

One key concept of the micro frontend approach is the usage of operational delegation and specification of the team, which is a characteristic of self-organized systems (see above). According to Conway (1968) a organizational structure has direct impact on the structure of the system it designs, which concludes that when the final micro frontend system shall get designed in a specific aimed structure the team behind its creation needs also get organized into a similar structure:

> *"Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure."* (Conway, 1968)

This statement is also considered and known as Conway's law and promotes one of the key manifestations in microservice and micro frontend design (cf. 3.1.2). Furthermore, as the alignment of team strucutre and system strucutre is crucial to the prevention of tensionpoints (Newman, 2015, ch. 10) in the final product a design system also needs to be aligned regarding the very same principle, as it's primary purpose is to promote the goals of the final product. In other words the design system and the final product are living in an almost same organizational enviroment during their creation. By recognizing the connection between the design system, the product and the team structure, we ensure that the system we are trying to build makes sense to the organization we are building it for (Newman, 2015, ch. 10).

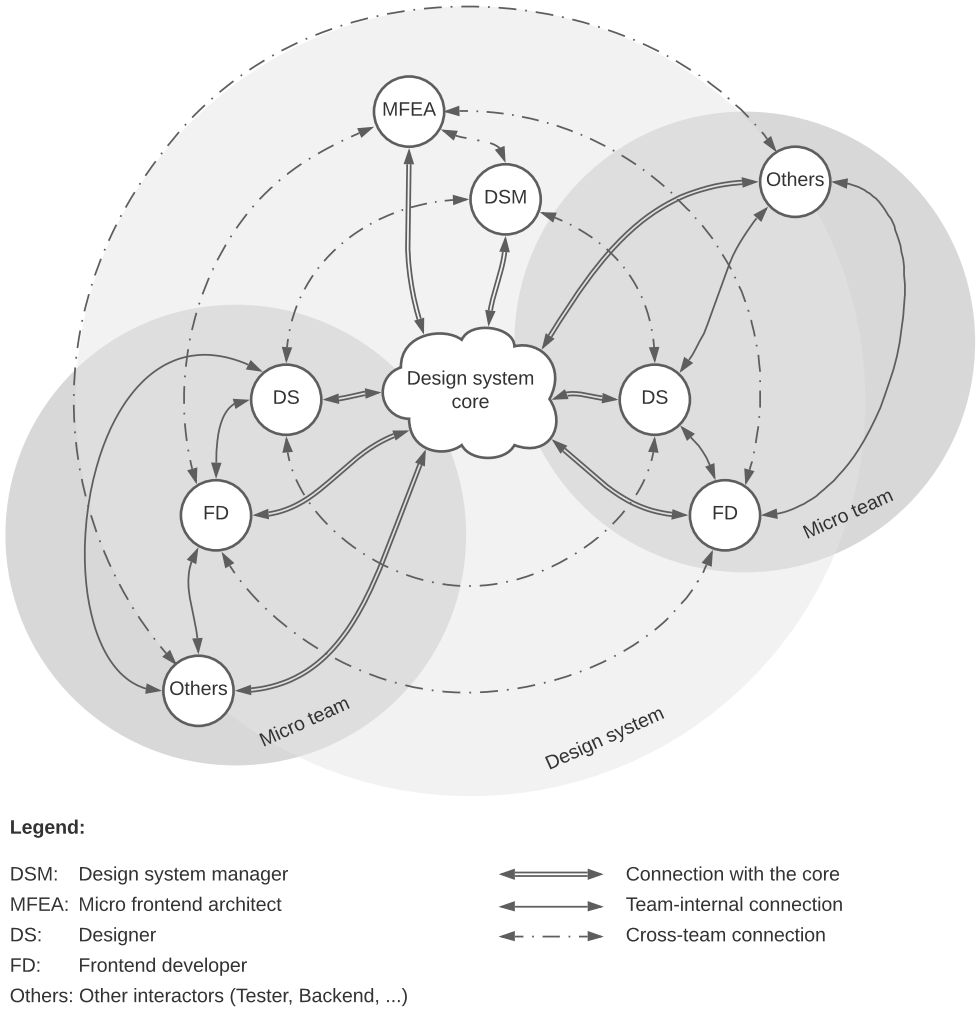

Figure 4.7: Interactors Connections within the Design System

Figure 4.7 shows the organizational influence of the interactors with possible interconnections to counteract the variety of dynamic design awareness within the design system. The design system cor in the center of the illustration represents the fundamental design system model from figure 2.4 of chapter 2 containing the flows and connections between patterns, principles, values and branding. Furthermore, the design system, illustrated in figure 4.7, considers the system emerging between the interactor(-systems) and the fundamental design system presented in figure 4.6 creating a second-order design system in regards to second-order cybernetics (cf. section 4.3.3). By applying Conway's law it is possible to structure the system by structuring the variety of interactors. This structuring is possible through the presented interconnections, to produce a system behavior that primarily targets a consistent design awareness between all interacting parties:

> *"System structure is the source of system behavior. System behavior reveals itself as a series of events over time."*(Meadows, 2008, p. 89)

The key parameters, that are influenced by humans, within the modeled second-order design system are the activator, comparator, and sensor, which can be concluded from 4.6. Therefore especially in these areas, a mechanism is needed to ensure more consistent and right evolving judgment between the different parties. But identifying the modeled connections is only the first condition that promotes intended behaviors, as the connections themselves need to be also used by the interactors by flowing different types of information efficiently and purposive effectively. Efficiently means that the processes operating onto the connections demand the least possible amount of work to the interactor, as the design system exists to promote the works of the interactors. Purposive effectively means that the connections need to serve a goal that has an effect on more consistent awareness within the activator, comparator, and/or sensor. It is also important to highlight that these connections are still allowed, even required, to emit disturbances between the connected interactors, as disturbances are the driving forces of the cybernetic evolvement of a system, as systems that are not disturbing are stability-oriented and not dynamic (Kruse, 2007) to encounter the problem-variety. The following describes the elements of the interactors feedback-response loop, that are most susceptible for variety:

- **Activator**
  In a microarchitecture, it is a far more common activity to make decisions in a pool of more options compared to a monolithic approach (Newman, 2015, ch. 12). An interactor also needs to consider his decisions in regards to other interactors as it might affect them.
- **Sensor**
  Interactors might sense their results differently. But a rapid awareness of broken results is crucial in a central design system that might affect other frontends and stakeholders (Mezzalira, 2021, ch. 4). Sensoring mechanisms should promote consistent awareness.
- **Comparator**
  Depended on the individual standards, interactors might compare the results regarding different levels of intended results. Standardizations, like principles, values, and patterns are already key essences of design systems. Connections between the interactors could ensure that these elements are applied within every loop-cycle.

With theses descriptions the organization in figure 4.7 can be expanded:

1. Every interactor is connected to the design system core, as everyone is creating its own awareness of the second-order design system, also influencing the overall design systems evolution to some degree, as well as evaluating results through the design systems process.
2. Most interactors live in a cross-functional team environment as this is the most popular shift within micro frontend architectures (Geers, 2020, sec. 1.1.1), to enable team scaleability (cf. section 3.1.1). Within these team environments, every stakeholder

is strongly related with the team members in pursuing the micro frontends success. When some new feature is implemented in the frontend all members need to work together in close relation.

3. As the micro teams normally follow similar and coupled aimings without inferring each other they are more isolated to every other design system's interactor outside their own micro frontend team. In this general field, every discipline (design, development, ...) pursues diverse goals. For example, while the designer(-s) of one team agree internally on pursuing their goal with the most suitable solution for their case, another team aiming to introduce a similar solution affecting the same part of the design system might implement it in another variety. Following the pattern of this example in other disciplines is can be seen that varieties obviously collide on the discipline level in cross-team interactions. By promoting the principle of delegation and hierarchy (Meadows, 2008, p. 85) a general solution process can be divided into potential levels of variety-collision, which is abstractly visualized based on the in figure 4.8. These levels show that the interactor's variety mostly affects interactors that operate within the same variety. Therefore the cross-team relations in figure 4.7 are organized radial to their discipline-colleagues in the global system.

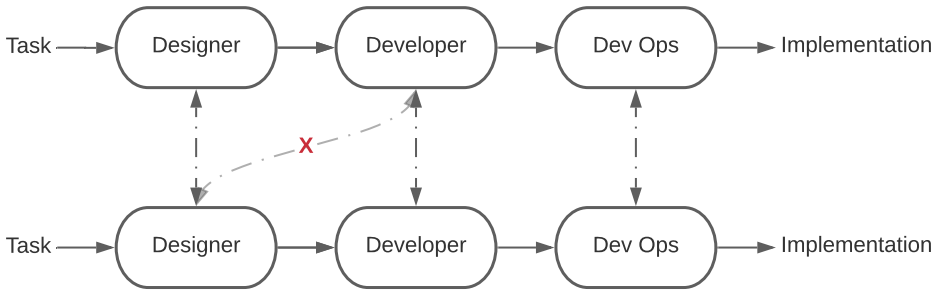

Figure 4.8: Levels of Cross-Team Relations

Creating a solution starts with the design the designer handsoff to the developer. After the developer has implemented the design, the development operator deploys it. The top flow represents one team, while the bottom flow another team. Disciplines might interfere with the solution they create, but a designer, for example, will rarely influence a developer who is not in the same micro team.

So as the radial connections between same-disiplinary interactors should be established to encounter varieties the next section highlights a way for useful cross-team communications. Before that it also shall be metntioned the behavior of the design system manager(-s) and micro frontend architect(-s). The design system manager takes the strongest responsibility role of the design system. Therefore such person needs to be strongly familiar with the design process of the final product aswell as the design system itself. As designers influence the behavior of the final product by designing the product and the design system, it makes sense to connect them at the first state as shown in figure 4.7. Although it would also make sense to connect the design system manager with other stakeholders like the frontend developers, the model is limited to the visual connections due to abstract clarity. The

design system manager also needs to take care that the design system is compatible with the general frontend architecture, therefore the relation to the micro frontend architect is also relevant and important. Furthermore, the micro frontend architect on the next radial level is strongly considered with all other frontend developers and creating the architecture is highly dependent on the micro team goals. Frontend developers are the best interface for connecting with the architect. Interactors defined as "others" are other interactors modeled in the stakeholder analysis. They follow the same paradigm of connectedness as designers and developers, but due to the clarity of the model, they have been summarized.

Commons with Technology

In the previous section, the interconnections within second-order design systems have been introduced. As the organization of cross-team connections has been modeled the next important step is defining and describing these connections further. Connected elements need to interact with each other, otherwise, the connections themselves are dead and do not promote any dynamic complex system behavior that works against the problem-variety within the system. Although one primary aim of micro frontends is to reduce the coupling of frontends as well as possible, to promote autonomy within each micro frontend, this section shows the need for establishing commons between teams. Commons, are properties that are assigned globally to disciplines within the design system. Without common procedures, common goals, and common thinking most cross-team connections won't be able to exist. Commons also enable the flow of knowledge between various interactors (Geers, 2020, sec. 1.4.3), which also leads to feedback orientated behavior in cybernetics. One general trap that occurs with commons is the chance of abuse by other users. on the one hand, if one user breaks something, the others might suffer that from that abuse. On the other hand, the success of one common directly and positively affects everyone relying on that common (Meadows, 2008, p. 191).

To achieve effective commons between teams that connect each member with its disciplinary colleagues in other teams a good way is to first rely on shared design system principles as well as shared technologies. Shared technologies might first appear to counter the autonomy and framework-agnosticism of the micro frontends itself. But as section 4.2 already highlighted framework-agnostic components can be based upon frameworks and still be characterized as framework-agnostic aslong as they are able to live in diverse ecosystems without introducing significant overheads. The argument behind using common frameworks when it comes to global actions, like creating components for the design systems pattern library is the chance to connect disciplines across various teams and by that pursuing a shared goal that adapts into each indivdual frontend context. The technology becomes the language and enviroment of discussion to create sultions for highly complex and dynamic problem fields in micro frontend design. How such technologies can be used therefore will be demonstrated and evaluated in the next upcoming chapter by using Svelte, as the global technology to create components and Tailwind CSS based Utility-First CSS classes as a global approach of styling.

5 Evaluation with Svelte and Tailwind CSS

This forthcoming chapter evaluates the properties of design systems in micro frontend architectures described in the previous chapter. Based on the second-order design system model from figure 4.7, the chapter examines the extent to which technological connections between cross-team designers and developers can be implemented. Technological choices for these evaluations are the compiler Svelte, for JavaScript component-pattern generation, and the framework Tailwind CSS, for systematic CSS design. The goal of the evaluation is to test the practical feasibility of implementing cross-team discipline connections in framework-agnostic characteristics in the environment of micro frontends. The chapter first starts by introducing the utility-first approach on that Tailwind CSS relies on. Subsequently, an introduction to Svelte components is given. The chapter ends by using both technologies to offer a framework-agnostic approach that enables connections between the micro frontend teams.

5.1 Utility-First CSS with Tailwind CSS

There are various CSS methodologies web developers apply as a relational philosophy between CSS and HTML today (Godbolt, 2016, p. 27-28). Oftentimes these methodologies need to get applied correctly by developers to stay consistent within the codebase. In comparison to methodologies, CSS frameworks deliver preset CSS classes to the code-base to deliver consistent UI design, but still, to advance and use the framework design a consistent CSS methodology oftentimes needs to be executed to handle common challenges of CSS. This section introduces the CSS framework Tailwind CSS which introduces its own utility-first methodology with predefined utility classes. First, the variety of common challenges in CSS creation is presented. Then the structuring potential of the Tailwind CSS approach gets discussed in conjunction with the previously presented challenges.

5.1.1 The Variety of CSS Styling

Based on the presented CSS issues of Godbolt (2016) the following challenges frequently occur in CSS design (Godbolt, 2016, p. 38):

- **Specificity**
  Dealing with different manifestations of specificity increases the complexity of the CSS code.

- **Resetting Styles**
  Switching between various overwriting within nested definitions duplicates and pollutes the CSS code.
- **Location Dependence**
  If styles are nested or scoped into a parent class, shifting markup elements without this parent is not possible.
- **Multiple Inheritance**
  It can happen that one markup-section is styled by various sources of inherited parent styles, which may lead to resetting styles or location dependence
- **Further Nesting**
  To overwrite inherited styles another nested selector may need to get introduced, only to ensure the needed specificity

Other styling related challenges that occur in larger modularized web applications are exemplified by Klimm (2020) as follows (Klimm, 2020, p. 24-26):

- **Redundant CSS Definitions with Scoped Styling**
  When component styles are created often times a scoped CSS approach is used, where each component scopes its style definitions with a specific parent selector.
- **Component Post-Modularization**
  When components need to get split up into smaller pieces, the markup, the logic, and the CSS need to get extracted and shifted. The more complex the styles are the harder it is to identify the necessary CSS declarations in the codebase.
- **Unused CSS Classes**
  As web applications evolve overtime to do the stylings and CSS too. By removing or editing the markup it can happen, the more complex the CSS is, that unnecessary declarations remain in the codebase. This also reinforces the whole CSS complexity over time by influencing further CSS challenges.
- **Property Hell - Component Style Attributes**
  When it comes to the development of components there can be observed that smaller style adjustments, for example, colors, lead to the introduction of specific attributes, which then make the codebase in the logic section as well as style section bigger and more complex.

As it has already appeared in the previous listings, many of the aspects presented are connected to each other. This means that the complexity of one aspect influences also the complexity of other aspects, which in turn all together lead to increasingly dynamic complexity in the CSS creation of web applications, as it is modeled in figure 5.1. In conjunction with the variety that has been presented in section 4.3, the creation and maintenance of CSS in the design system of a micro frontend architecture can prove itself as highly complex. To solve the challenge of CSS creation and maintenance within an approach that can be used and discussed globally by the design system interactors, while also being framework-agnostic

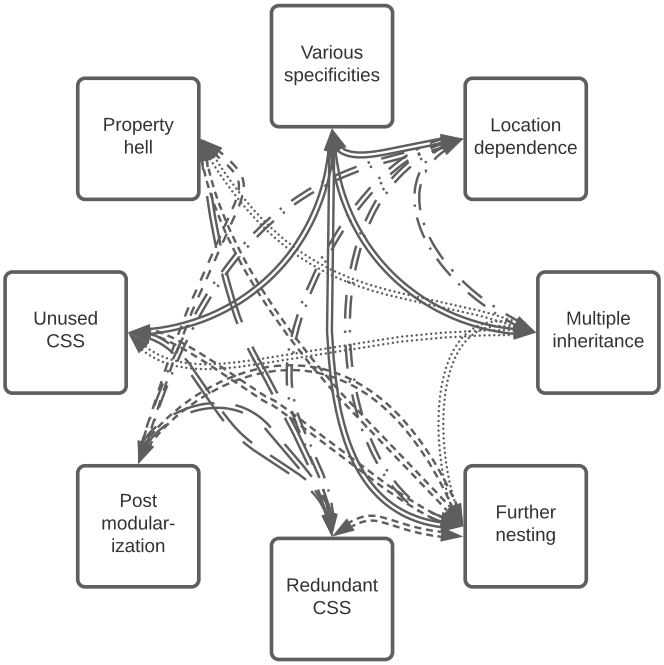

Figure 5.1: The Variety of CSS Styling

to the micro frontends where it gets applied, utility-first CSS might be a methodology that is worth to consider in managing styling.

5.1.2 The Potential of Tailwind CSS in Micro Frontend Design

In the last years, the framework Tailwind CSS has gained significant popularity within the web community (Greif and Benitte, 2020*a*) and now leads according to the latest study of Greif and Benitte (2020*a*) the charts in types of developer satisfaction and interest. Unlike traditional CSS frameworks, like Bootstrap or Bulma, Tailwind CSS follows a constraint-based approach (Tailwind CSS, 2021, /#constraint-based), where utility classes represent elementary CSS style definitions. The design of the UI then gets defined by the developer by not predominantly creating new semantic classes, or using such and overwriting properties through specifities, but by using multiple atomic utility classes directly in the markup. With this approach Tailwind CSS aims up to support custom and consistent UI design by introducing an organization and setup of utilities that promote systematic design (Tailwind CSS, 2021, /#constraint-based).

Cybernetic Benefits

> *"When a constraint exists advantage can usually be taken of it."* (Ashby, 1957, par. 7/14)

According to Ashby (1957) especially in environments of high variety, constraints are important to estimate occurrences better. Constraints can be manifested in different variants, like natural laws, objective properties, or predictive behaviors. Another type is the *"machine as constraint"* (Ashby, 1957, par. 7/19), where an observer is able to observe a sequence of information through something like a protocol that enables the observer to *"recode"* the whole into two simple statements:

1. A statement of transformation
2. A statement of the actual input given

This means that the protocol establishes constraints that structure the view of the processes inside the machine to the observer in a more comprehensible approach. In section 4.3.4 the aim to use technology to establish commons had been addressed, from where it can be concluded that creating commons with technology also means to establish technologic constraints inside the design system. With Tailwind CSS a type of utility-class protocol can be introduced to the design system, meaning that through Tailwind CSS an advanced low-level CSS methodlogy with its reinforcing behaviors can be used to manage the variety of CSS (see above) and the connecting of cross-team discipline layers (cf. section 4.3.4). From a cybernetic perspective Tailwind CSS is due to various properties a beneficial technology that adapts to cross-team system design:

- **Customizability and Traceability**
  The utility-classes are generated based on a JavaScript configuration file. This allows to set up specific and consistent properties. The usage of these generated classes is, due to the good documentation, easy to learn (Dayan, 2019).
- **Consistent Usage is Forced**
  Tailwind CSS requires the developers to use the exact utilities to achieve style goals. Unlike other methodologies, like BEM[1], the developer can not deviate from the specifications, as then no effect would be sesnored. With BEM an inexperienced developer has a higher chance to deviate from the naming conventions. This forces developers to deal with learning Tailwind CSS conventions on a regular basis, which leads to stronger cross-team connections inside the design system.
- **Developer Expierence**
  Tailwind CSS has a high level of satisfaction for developers (Greif and Benitte, 2020*a*), which promotes the activity of the interconnections between micro teams.

Another cicumstance that can be handeled well with Tailwind CSS can be elaborated from Conway's law:

[1] BEM: Block-Element Modifier, is a popular CSS methology

> *"Because the design which occurs first is almost never the best possible, the prevailing system concept may need to change. Therefore, flexibility of organization is important to effective design."* (Conway, 1968, p. 31)

When it comes to style the UI over time evaluations happen, but as also show by Klimm (2020) and Godbolt (2016) it can be difficult for developers to be confident in changing or removing old styles. Therefore unused CSS remains in the markup or more CSS selectors with higher specifities, as a predictability of the side effects in the complex connectedness (cf. figure 5.1) is less available the larger the codebase is. Since Tailwind CSS is used directly within the markup due to its atomic approach, evaluations can get are very good assessed in regards to their impact. This is because changes are implemented directly in the markup and not in the general CSS classes, so developers do not to investigate further beside the section they are editing. With normal CSS classes they would need to check if a change of the class leads to undiesired changes in other parts of the frontend.

The Possibility of Framework-Agnosticism

One challenge that comes with every CSS framework that developers might want to introduce globally inside the micro frontend architecture is the question regarding the framework-agnosticism. In the chapter 4 it has been highlighted how the autonomy of each micro frontend is dependent on the technologies it is required to use. Therefore the important requirement of framework-agnosticism should also not suffer from the goal of establishing cross-team connections. As section 4.2 already has highlighted a design system itself can be considered as a framework to the developer and designer, while also framework-agnosticism does not mean to renounce the use of frameworks, but the aim that the final implementations can work in various environments, where other frameworks are dominating the frontend development. This means, when it comes to the introduction of Tailwind CSS, inside the design system for micro frontends, developers need to be able to integrate the design system into their own autonomous frontend, without limitations. Furthermore, the developers need only be required to use Tailwind CSS, when it comes to creating patterns. This does not mean that developers should not be able to use Tailwind CSS for everything else inside their frontend, but they should not be required to.

To establish Tailwind CSS in a framework-agnostic manner, the configuration needs to live inside the design system and with every update of the system, it needs to get deployed through a mechanism that generates the used utility classes inside a general CSS file the frontends can import. With such an approach, the frontends do not need to set up Tailwind CSS by themselves to make patterns work, but to just ensure that the global CSS file is available. Furthermore, depending on the micro frontend architecture this can get simplified even further, by introducing the design systems CSS file directly into the application shell (the middleware of micro frontends)

5.2 Designing Svelte Components with Tailwind CSS

> *"As usual React and Vue lead the pack, but Svelte is quickly establishing itself as a very serious contender for the front-end crown."* (Greif and Benitte, 2020*b*)

Recently, Svelte has gained more and more attention within the web community and currently also leads the satisfaction ratio close before the popular frameworks Vue.js and React.js (Greif and Benitte, 2020*b*). Svelte is not another single-page application framework that relies on familiar approaches like Vue.js or React.js, as it can be more considered as a compiler that creates components based on vanilla JavaScript instructions. With this advantage, Svelte can be an efficient framework-agnostic solution to create components that are able to live in any environment without introducing significant overhead. On the web, the bundle size of a framework usually has to be taken into account, so some specific functions cannot be brought into the framework (because they are too rarely used) in order to not pollute the size unnecessarily. With Svelte the deployed tools are compiled in the build to performant javascript instructions, which allows much more features, because each time only the part is deployed, which is also used and nothing more. For example, svelte has been able to create a powerful tool for transitions that generate transitions individually, which would not have been possible in a normal framework. Besides the compiled svelte component format, svelte can alternatively be compiled to standardized web components. The advantage of this approach would be the already more frequent use of such components in web apps, but some features of svelte are also lost and so svelte components themselves offer the greatest degree of possibilities. Svelte has mechanisms for reactivity, state management, and event management, which provides common features that you are used to from Vue or React in a framework-agnostic way. Since there is already some work on the use of web components, this evaluation will focus on Svelte's own component format in the following.

5.2.1 Set Up Approach for the Framework-Agnostic Pattern Library

In the following, an approach of creating a framework-agnostic Svelte pattern library with Tailwind CSS is elaborated. The goal of this section is to evaluate the feasibility of using Svelte and Tailwind CSS within a design system for micro frontends. While there are code examples presented the final repository also has been published to GitHub [2].

Introduction

A common Svelte environment can get generated by using the official Svelte template by using the degit *npx degit sveltejs/template name*. When has been executed some things need to get changed, as the template in general addresses more an app implementation than a

[2] `https://github.com/netzfluencer/svelte-tailwind`

library implementation. While there are serval guides on how to build components, compile and register them to the browsers window object in another web application[3].

This means, that the registration happens in the Library-owned *main.js* file, which then gets compiled and referenced in the micro frontends HTML by using script tags. With this approach, all components are getting registered to the micro frontends window object (by the compiled main.js file of the svelte pattern library). Although using the window object has relevance due to the possibility of including a cached JavaScript file to the frontend, importing only the required components before transpilation into the micro frontends codebase has also relevance, depending on use cases. Therefore the following implementation aims up to validate how compiled Svelte components are usable as ES modules to support module tree-shaking and deliver another approach of injecting Svelte components into common applications.

Installing Dependencies and Adjusting the Rollup

Based on the guide of Dhanaraj (2020) Tailwind CSS can get introduced to the Svelte template environment. But as this is not an app but a library there is one difference: Here the Tailwind styles are not declared in App.svelte, but in an own Tailwind.svelte file, which is only responsible for the tailwind styles and nothing else:

Code Example 5.1: Tailwind.svelte

```
1 <style global>
2   /* purgecss start ignore */
3   @tailwind base;
4   @tailwind components;
5   /* purgecss end ignore */
6   @tailwind utilities;
7 </style>
```

Also worth mentioning is, that Tailwind CSS uses a PostCSS Plugin called PurgeCSS[4], which in the build process removes all utility classes that have been generated by Tailwind but have not been used anywhere in the codebase. After importing the Svelte file in main.js (cf. code example 5.4) the compiler will extract the used utilities into the svelte-common bundle.css file.

Another change that needs to get done is switching the JavaScript format to *ESM (ECMAScript modules)*, so desired components are after compilation importable for other applications/micro frontends (cf. code example 5.2, line 6).

Code Example 5.2: rollup.config.js

```
1 /* ... */
2 export default {
3   input: 'src/main.js',
```

[3] https://github.com/lingtalfi/TheBar/blob/master/discussions/inject-svelte-in-existing-app.md (Last accessed 31 January 2021)

[4] https://purgecss.com/

```
  output: {
    sourcemap: true,
    format: 'esm',
    name: 'app',
    file: 'public/build/bundle.js'
  },
  /* ... */
}
```

Creating a Component

Next, a button component gets created, which can receive and display a label-property, dispatches a click event when clicked, and uses utility classes for styling.

Code Example 5.3: Btn.svelte

```
<script>
  import { createEventDispatcher } from 'svelte'
  const dispatch = createEventDispatcher()
  export let label
</script>

<button class="bg-gray-800 rounded-lg text-gray-100" on:click={()
    => dispatch('click')}>{label}(Svelte)</button>
```

Furthermore, the component needs to get imported and exported in the main.js file:

Code Example 5.4: main.js

```
import Tailwind from './Tailwind.svelte';
import Btn from './components/Btn.svelte';

export { Btn }
```

5.2.2 Using a Compiled Svelte Component in a Vue-Based Frontend

After executing the command *yarn build* within the svelte-tailwindcss repository the bundle.js and the bundle.css files are getting generated, which can be used in a micro frontend. In this case, a Vue 3 single-page application with Vite is used for demonstration purposes. Furthermore, an efficient Vue wrapper is build to more easily use svelte components in Vue. The Vue app itself has been generated based on the official Vue documentation[5]. The full repository of the upcoming implementation is also published to GitHub[6].

To display the svelte components the bundle.css needs to be referenced inside *index.html*. It would also be possible to import the CSS file directly into *main.js* of the Vite project, but the advantage of referencing the file in the index.html is the cross-sharing of the same

[5] https://v3.vuejs.org/guide/installation.html
[6] https://github.com/netzfluencer/vue-vite-svelte

code in various micro frontends when the file is hosted on a globally accessible provider (like a CDN). In the following the file is referenced in *index.html* to prove the possibility of referencing a globally accessible and cacheable file, while the file itself is currently manually moved into the Vite project itself. This circumstance is just existent to promote rapid prototyping and proofing the general concept. In the real world, solid deployment pipelines should get configured. Furthermore, a deeper discussion regarding the sense of using a cached bundle.css will be opened in the next section 5.2.3.

Code Example 5.5: index.html

```
<!-- ... -->
  <head>
    <!-- ... -->
    <link rel="stylesheet" href="/bundle.css">
    <!-- ... -->
  </head>
<!-- ... -->
```

After referencing the bundle.css the created button component is usable within the Vite project.

Code Example 5.6: HelloWorld.vue

```
<template>
  <h1>{{ msg }}</h1>
  <button @click="count++">count is: {{ count }}</button>
  <div ref="svelteComponent" />
</template>

<script setup>
import { defineProps, ref, onMounted, watch } from 'vue'
import { Btn } from '../../bundle'

defineProps({
  msg: String
})

const svelteComponent = ref(null)
const count = ref(0)
let Button = null

onMounted(() => {
  Button = new Btn({
    target: svelteComponent.value,
    props: {
      name: count.value
    }
  })
})

watch(count, (n) => {
  Button.$set({name: n})
```

```
30 })
31 </script>
```

The lines in code example 5.6, that are relevant for injecting the Svelte component are:

- Line 4: The DOM element where the Svelte component shall get mounted in.
- Line 9: The import of the compiled button, in the real world the file could come from the library itself as a NPM dependency.
- Line 15: The Vue reference initialization.
- Line 17: A global variable where the component instance get referenced, to access properties globally inside the setup (eg. line 29).
- Lines 20-25: The component instantiation, property assignment, and mounting
- Line 29: Binding changes of count to the component.

As already mentioned, it makes sense to build a wrapper component inside Vue, that is able to reproduce the mentioned behaviors of the list above for every other Svelte component. Such wrapper component can get approached as follows:

Code Example 5.7: SvelteWrapper.vue

```
1  <template>
2    <div ref="sc" />
3  </template>
4
5  <script>
6  let C = null
7
8  export default {
9    props: {
10     module: {
11       type: Function,
12       required: true
13     },
14     props: {
15       type: Object,
16       default: null
17     },
18     handlers: {
19       type: Object,
20       default: null
21     }
22   },
23   mounted() {
24     const M = this.module
25     C = new M ({
26       target: this.$refs.sc,
27       props: this.props
28     })
```

```
29      if (this.handlers) {
30        Object.entries(this.handlers).forEach(h => {
31          C.$on(h[0], h[1])
32        });
33      }
34    },
35    watch: {
36      props: {
37        deep: true,
38        handler: (n) => C.$set(n)
39      }
40    },
41    beforeUnmount() {
42      C.$destroy()
43    }
44  }
45  </script>
```

The principles are similar to code example 5.6, but this time properties are mapped automatically by their occurring. Also, a mechanism for handling the events has been introduced and when the wrapper gets unmounted, it also automatically unmounts the Svelte component. Variable names have been shorted to characters representing:

- C: component
- M: Module/Class constructor
- h: handler

Using the wrapper with a Svelte component can be achieved like shown in the following:

Code Example 5.8: HelloWorld.vue with SvelteWrapper

```
1  <template>
2    <h1>{{ msg }}</h1>
3    <button class="bg-gray-800 text-yellow-500" @click="count++"
       >count is: {{ count }}</button>
4
5    <SvelteWrapper :module="BtnClass" :props="{label: count}" :
       handlers="{click: () => count++}" />
6  </template>
7
8  <script setup>
9  import { defineProps, ref } from 'vue'
10 import { Btn } from '../../bundle'
11 import SvelteWrapper from './SvelteWrapper.vue'
12 const BtnClass = Btn
13 defineProps({
14   msg: String
15 })
16 const count = ref(0)
17 </script>
```

5.2.3 Managing Tailwind CSS

The more components are available inside the design system, the larger is the number of utilities required by the design system. This might be challenging, as not every frontend needs every component and therefore, on the one hand, it would be more performant to not include the code of unused components. On the other hand, having one file that can be cached and shared among all frontends is an advantage, as using the same components in multiple micro frontends wouldn't require loading the same CSS multiple times. Furthermore, setting all frontends to the same base of browser resettings (removing default browser styles) and general adjustments is a required benefit when it comes to sharing components in micro frontends. Nonetheless, it is important to keep the autonomy of each frontend as high as possible while also preventing issues that might be produced by a non-micro-frontend specific element. As the autonomy of micro frontends also enables the teams to schedule their own deployments a mechanism is needed to ensure that once a frontend is deployed its required resources also do not change anymore while in production. One common way within web applications is the usage of cache busting, where module bundlers assigning for each version a unique hash id to the filename, which could be in the case of bundle.css look like bundle.a73b90.css. This on the one hand allows browsers to use only as a long cached version of the file until references change to a new file with a new id. Depending on the regularity and diversity of the autonomously managed deployments it can therefore lead to the circumstance that a significant amount of frontends use diverse versions of the pattern libraries bundle.css relativizing the advantage of central cached files that a shared among multiple frontends. By considering this behavior, it therefore can be as efficient to directly handle the CSS within each frontend itself, where the CSS management can get based on two types of approaches, as presented in the following.

Importing the Bundle

By importing the bundle.css directly into the codebase of a micro frontend the module bundler can insert the required utility classes into the final deployable CSS. This approach ensures that Tailwind CSS is not required to be installed in the micro frontend itself. But if Tailwind CSS is installed and also used inside the frontend, then chances are high that utilities are declared multiple times inside the final CSS file outputs: One declaration from the bundle.css and one from the frontend.

Code Example 5.9: main.js of the Vite project

```
1 // ...
2 import './bundle.css'
3 // ...
```

Nonetheless, even if Tailwind CSS is used in a frontend itself, it can still be an efficient and less complex approach to start the development with the import of the bundle.css (cf. quote of Godbolt (2016)), to later on improve the CSS management according to the next presented approach.

> *"Instead of starting off your project with a large suite of tools and a sizable starting page weight, consider simplicity and leanness as an asset. Don't give that asset up unless the benefits outweigh the added complexity and weight."* (Godbolt, 2016, p. 46)

Custom Tailwind CSS Setup

Another approach, that is not framework-agnostic but relevant when a frontend also decides to use Tailwind CSS is using a Tailwind configuration that is based on the configuration of the design system. Following such an approach means to include the Svelte components of the design systems repository, which should be available in *node_modules*, to the scope of the PurgeCSS configuration. Furthermore, the CSS deployment of the svelte library needs to get adjusted, so it does not bundle the tailwind classes within a deployment (cf. code example 5.10)

Code Example 5.10: tailwind.config.js in Svelte Library

```
1 module.exports = {
2     // ...
3     ...!process.env.ROLLUP_WATCH ? { prefix: 'twignore-' } : {},
4     // ...
5 }
```

This ensures that the built bundle.css does not hold any Tailwind related declarations anymore, but other selectors like possible custom classes of components. The bundle.css then can get imported like shown in the code example 5.9 and the missing utilities will be handled by the local Tailwind installation, so no utility class occurs twice.

6 Conclusion

This work has developed a basis for the systematic design of micro frontends and shown how system behaviors behave in the design process in their interconnectedness to the system essences. Furthermore, it was shown that the different micro frontend stakeholders influence the dynamics and complexity of the design system, leading to variety in the created design solutions. Based on cybernetic insights and the research into the general characteristics of micro frontends a second-order design system has been introduced, which considers the cybernetic system that is formed by the common design system, the micro teams, and the interacting stakeholders and their relations. This second-order design system is an essential insight, as the interactors of the design system are not only interacting with the system but also part of the system itself. To consolidate this property with the aim of introducing new advantages for consistent frontend design Conway's law has been considered to introduce an advanced organizational structure of the second-order design system to counteract the problem variability and thus, according to Ashby's law, support the implementation of a dynamic solution system. However, these cross-organizational team connections must exist for higher purposes than simple undefined theoretical connections, so it is important to introduce common topics across all teams without limiting the local autonomy of the individual teams. Since the commons initially imply a form of coupling, it was therefore considered how appropriate commons can be established for the development of the design system itself. In this regard, technology was considered as a way to establish a common dependency among design system creators from different teams and to create a kind of cross-team language within disciplines. However, if technologies such as frameworks or compilers are to be used for the cross-team creation of the design system, the subsequent deployment must have a framework-agnostic property in order not to interfere with the individual micro frontends and to provide a high degree of compatibility and flexibility. A framework-agnostic design system may rely on frameworks and other technologies but may not require a particular setup of these technologies in the respective frontends after its own deployment.

With these findings, an evaluation of the developed concepts and models was carried out in conjunction with Tailwind CSS and Svelte. Tailwind CSS was chosen due to its utility-first CSS approach to investigate the use of atomic CSS classes in a design system. Since Svelte components act independently without a framework, after compilation (based on JavaScript instructions), the use of this technology in the context of framework-agnostic design systems was also examined. With the creation of Svelte components based on Tailwind CSS, different approaches to integration became apparent. Overall, it could be shown that the creation of framework-agnostic components with Svelte and Tailwind CSS is feasible, although several particularities have to be considered. For example, Svelte components should be wrapped

by wrappers of the local technology stack in the frontend in which they are regularly used to enable efficient handling.

Tailwind CSS can be integrated into frontends in a different ways, these options should be considered in relation to individual contexts. On the one hand, a static bundle CSS file can be generated by the Svelte compiler, which contains the used utilities of the design system. This file can then be referenced by the frontends. However, the disadvantage here is that the different frontends may use different versions of the library at different times. This means that central hosting of such a file can lead to conflicts that can only be managed suboptimally by cache busting. An alternative option, which is also framework-agnostic, is to import the CSS directly into the frontend and integrate it into the micro frontend using a local bundler like webpack or rollup. However, there is another possibility of integration, which can not considered as longer framework-agnostic, where Tailwind CSS is installed in the frontend itself and the setup is chosen to include the design system components in the consideration of the utility classes and not to receive them from the bundle of the design system. This approach makes sense if a frontend also needs to frequently use Tailwind CSS for its own frontend design prevents the multiple declaration utility classes.

Based on this work, fundamental conclusions can be drawn about design systems in the context of micro frontends. As a result, further properties of other facets can now be investigated and processed. Thus, based on the cybernetic insights, frameworks for more specific procedural models for the implementation of design systems could be developed, or other framework-agnostic technology approaches could be elaborated. Another potential possibility for further investigations is to evaluate the use of the presented approaches with Svelte and Tailwind CSS and to apply them to real projects. In addition, a further evaluation of the cross-team connections would be relevant, in which other disciplines such as backend or dev ops could also be taken into account. With increasing attention to the topics referenced in this thesis, there are numerous opportunities to expand on the results presented.

List of Code

List of Figures

Bibliography

Ashby, W. R. (1957), *An Introduction to Cybernetics*, Chapman Hall Ltd, London. PDF Version (1999) last accessed on 10 January 2021 - http://pespmc1.vub.ac.be/books/IntroCyb.pdf.

Bertalanffy, L. (1968), *General System Theory: Foundations, Development, Applications*, George Braziller, New York. ISBN 9780807604533.

Clark, J. (2019), 'Elevating the experience of your design system'. Talk - Last accessed 22 December 2020.
URL: *https://www.rethinkhq.com/videos/elevating-the-experience-of-your-design-system-jess-clark*

Conway, M. (1968), 'How do committees invent?'. Last accessed 4 January 2021.
URL: *http://www.melconway.com/Home/pdf/committees.pdf*

Couldwell, A. (2020), *Laying the Foundations*, black & white edn. ISBN 9798633126808.

Dayan, S. (2019), 'Redesigning our docs – part 4 – building a scalable css architecture'. Last accessed 29 January 2021.
URL: *https://www.algolia.com/blog/engineering/redesigning-our-docs-part-4-building-a-scalable-css-architecture/*

Dhanaraj, C. (2020), 'Tailwind / svelte demo integration'. Last accessed 31 January 2021.
URL: *https://github.com/chrisdhanaraj/svelte-tailwind-integration*

DIN 69901-5:2009-01 (2009), 'Project management - project management systems - part 5: Concepts'.

Fowler, M. and Lewis, J. (2014), 'Microservices'. Last accessed 22 December 2020.
URL: *https://martinfowler.com/articles/microservices.html*

Friis, J. K. B. O., Hendricks, V. F. and Pedersen, S. A. (2009), *A Companion to the Philosophy of Technology*, Wiley-Blackwell, White River Junction, VT 05001. ISBN 9781405146012.

Frost, B. (2016), *Atomic Design*, Frost, Brad, Pittsburgh, Pennsylvania. ISBN 9780998296616 - ePub.

Geers, M. (2020), *Micro Frontends in Action*, Manning Publications Co., Shelter Island, NY 11964. ISBN 9781617296871 - ePub.

Github (2020), 'Github star filtering search'. Last accessed 23 December 2020.
URL: *https://github.com/search?q=stars%3A%3E100&type=repositories*

Godbolt, M. (2016), *Frontend Architecture for Design Systems*, 1 edn, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA. ISBN 9781491926789.

Greif, S. and Benitte, R. (2020*a*), 'The state of css 2020 - css frameworks'. Last accessed 24 December 2020.
URL: *https://2020.stateofcss.com/en-US/technologies/css-frameworks/*

Greif, S. and Benitte, R. (2020*b*), 'The state of js 2020 - frontend frameworks'. Last accessed 29 January 2021.
URL: *https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/*

Harris, R. (2019), 'Rethinking reactivity'. Talk - Last accessed 29 December 2020.
URL: *https://www.youtube.com/watch?v=AdNJ3fydeao*

Herrington, J. (2020), 'Micro-frontends: What, why (and why not) and how'. Last accessed 4 January 2021.
URL: *https://jherr2020.medium.com/micro-frontends-what-why-and-why-not-and-how-997acb2bd674*

Heylighen, F. and Joslyn, C. (2001), 'Cybernetics and second-order cybernetics'. Last accessed 15 January 2021.
URL: *http://pespmc1.vub.ac.be/Papers/Cybernetics-EPST.pdf*

Immich, T. (2019), 'Viel system, aber wenig design? wie design systeme adaptiver werden können, um guter ux nicht eher im weg zu stehen'. Last accessed 22 December 2020.
URL: *https://dl.gi.de/handle/20.500.12116/24470*

ISO 9241-210:2019-07(en) (2019), 'Ergonomics of human-system interaction - part 210: Human-centred design for interactive systems'.

ISO/IEC/IEEE 15026-1:2019(en) (2019), 'Systems and software engineering - system life cycle processes'.

ISO/IEC/IEEE 9241-11:2018(en) (2018), 'Ergonomics of human-system interaction - part 11: Usability: Definitions and concepts'.

Jackson, M. C. (2019), *Critical Systems Thinking and the Management of Complexity*, Wiley, Hoboken, New Jersey. ISBN 9781119118374 - Ebook.

Kholmatova, A. (2017), *Design Systems*, Smashing Media AG, Freiburg, Germany. ISBN 9783945749586.

Klimm, M. (2020), 'Erstellung einer isolierten vue-komponenten bibliothek im zusammenspiel mit dem utility-first-framework tailwind css'. A project documentation submitted to the TH Köln - University of applied science, URL last accessed 27 January 2021.
URL: *https://raw.githubusercontent.com/netzfluencer/SCI$_V$ue − Components − Library−Tailwindcss/master/02$_D$okumentation/PP2020$_K$limm−Vue−Component−Lib.pdf*

Krishnamurthy, N. and Saran, A. (2007), *Building software*, Auerbach Publications, New York. ISBN 9781000654462.

Kruse, P. (2007), 'Transcript of an interview with prof. peter kruse'. Transcript - Last accessed 14 January 2021.
URL: *https://gist.github.com/wolfhesse/c935bba4ae25667f51e7*

Lobacher, P. (2015), 'Kanban grundlagen - stakeholder workshop'. Course - Last accessed 6 January 2020.
URL: *https://www.linkedin.com/learning/kanban-grundlagen/stakeholder-workshop*

Meadows, D. H. (2008), *Thinking in Systems*, 1 edn, Chelsea Green Publishing, White River Junction, VT 05001. ISBN 9781603580557.

Mezzalira, L. (2021), *Building Micro Frontends*, 1 edn, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. ISBN 9781492082996 - Early Access Ebook.

Newman, S. (2015), *Building Microservices*, 1 edn, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. ISBN 9781491950357 - Ebook.

Oxford University Press (2020), 'Oxford dictionary'. Last accessed 1 January 2021.
URL: *https://www.lexico.com/en/*

Scheithauer, A. (2017), 'Was ist kein system?'. Last accessed 22 December 2020.
URL: *https://www.oose.de/blogpost/was-ist-kein-system/*

Suarez, M., Anne, J., Sylor-Miller, K., Mounter, D. and Stanfield, R. (n.d.), *Design Systems Handbook*, Invision. PDF Download - Last accessed 04 December 2020.
URL: *https://www.designbetter.co/design-systems-handbook*

Tailwind CSS (2021), 'Website of tailwind css'. Last accessed 28 January 2021.
URL: *https://tailwindcss.com*

ThoughtWorks, Inc. (2019), 'Technology radar vol. 21'. Last accessed 22 December 2020.
URL: *https://assets.thoughtworks.com/assets/technology-radar-vol-21-en.pdf*

ThoughtWorks, Inc. (2020*a*), 'Technology radar vol. 22'. Last accessed 22 December 2020.
URL: *https://assets.thoughtworks.com/assets/technology-radar-vol-22-en.pdf*

ThoughtWorks, Inc. (2020*b*), 'Technology radar vol. 23'. Last accessed 22 December 2020.
URL: *https://assets.thoughtworks.com/assets/technology-radar-vol-22-en.pdf*

WHATWG (2020), 'Custom elements'. Last accessed 24 December 2020.
URL: *https://html.spec.whatwg.org/multipage/custom-elements.html*

Wiener, N. (1948), *Cybernetics; or, Control and Communication in the Animal and the Machine*, 2 edn, The MIT Press, Cambridge, Massachusetts. ISBN 0262230070.

Wittmann, K. (2015), 'Entwicklung eines vorgehensmodells für die imagefilmproduktion auf der basis agiler vorgehensmodelle und techniken in der softwareentwicklung'. Bachelor Thesis submitted to the TH Köln - University of applied science.