Introduction

> Simplicity is prerequisite for reliability. - Edsger W. Dijkstra

Der Erfinder der Programmiersprache Clojure, Rich Hickey, ist ohne Zweifel eine Koryphäe auf seinem Gebiet, der Strukturierung von komplexen Systemen. In einer vielbeachteten Keynote aus dem Jahr 2012 geht er auf etymologisch-philosophische Spurensuche nach dem Wort **simplicity** aus Sicht eines Softwareentwicklers.[1] Das Adjektiv *simple* hat demnach seinen Ursprung im lateinischen Wort *simplex*, was soviel wie *einfach* oder *einzeln* bedeutet. In Gegensatz dazu stehen Eigenschaften wie *complex* oder *multiplex*. Qualitative Software ist, so Hickey, vor allem simpel - im Prozess, im Design, in der Struktur und in der Entwicklung.

Bezeichnend für diese Keynote ist, dass sie von Rich Hickey im Rahmen einer Webentwicklerkonferenz gehalten wurde. Software für den Browser war (und ist) seit langer Zeit maßgeblich geprägt von **Komplexität** auf mehreren Ebenen. Im Laufe dieser Bachelorarbeit werden diese Stru

- Das *Document Object Model* als Rückgrad jeder Webapplikation ist hierarchisch strukturiert und deren Elemente damit keinesfalls unabhängig in ihrer Darstellung und Reihenfolge.
- JavaScript als single-threaded Skriptsprache lässt sich schlecht ihn ihrem Verhalten isolieren, ist Fehleranfällig und hatte lange Zeit nur sehr wenig idiomatische Lösungsansätze für komplexe Probleme, wie beispielsweise Asynchronität
- CSS ist geprägt von stetigem Überschreiben vorher definierter Regeln und verletzt damit Simplizität in ihrer Struktur, Design und dem Entwicklungsprozesses auf bester Art und Weise.

Betrachtet man andere populäre Webframeworks dieser Zeit, wie beispielsweise React oder Angular, lässt sich ziemlich schnell ein gemeinsames Designpattern ausmachen, wie Komponenten intern ihre Zuständen verwalten. Der Facebook Entwickler Dan Abramov hat dieses dichotome Pattern systematisch erfasst und in zwei Kategorien eingeteilt.

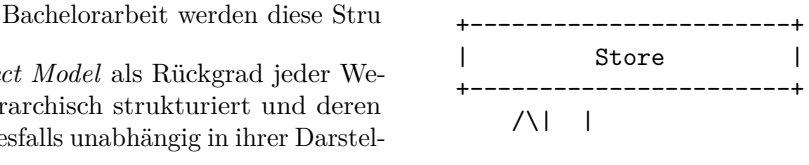
Nach Abramov existieren zum einen Komponenten, die alleine für das **Darstellen von Information** zuständig sind. Diese Komponenten nehmen keinerlei Einfluss auf Informationen oder anderen Komponenten um sich herum. Sie sind im wahrsten Sinne passiv und fremdgesteuert über klar definierte Schnittstellen. Diese Komponenten sind häufig flexibel einsetzbar und hochgradig wiederwerwendbar. Abramov nennt diese Komponenten "Presentational Components".[1] Verortet man diese Art von Komponenten innerhalb des *MVC Pattern*, sind die Komponenten reine *View-Elemente*.

[1] Rails Conf 2012 Keynote: Simplicity Matters by Rich Hickey

Im Gegensatz dazu stehen Komponenten, die für die **Verarbeitung von Informationen** zuständig sind. Diese so genannten "Container Components" haben oft einen internen Zustand, den sie verändern können und an ihre Kindkomponenten weiterreichen können.[1] Im *MVC Pattern* handelt es sich um die *Controller*. In der funktionalen Programmiersprache Elm werden diese Elemente *Updater* genannt, was ihre Funktion noch besser umschreibt.

Ein üblicher eventgesteuerter Webservice setzt sich aus unterschiedlichsten Komponenten zusammen, die wiederum unterschiedlichste Eventlistener & -emitter in sich subsummieren. Diese inhärente Komplexität verlangt geradezu nach einer klaren, deterministischen Struktur des Webservices, die das Zusammenspiel orchestriert. In der Analogie des Orchesters gesprochen, benötigt der Webservice (oder sogar die gesamte Webapplikation) einen Dirigenten, der für die Steuerung verantwortlich ist.

Store



die Änderungen des Zustands vorhersehbar aber auch zurückverfolgbar macht.

Microservices

In a nutshell a microservice is a small, autonomous service that works together with other services seamlessly.[2, p. 2] While the concept of microservices lacks a formal standardization there are certain defacto standardized ideas emergine from this pattern. The purpose of this section is to match those ideas against the manifestations of web components.

First of all, from a technical perspective, a microservice reinforces the *Single Responsibility Principle* defined by Robert C. Martin: "Gather together those things that change for the same reason and separate those things that change for different reasons."[3] An a way this principle tackles another often cited design principle of the *seperation of concerns*. Web Components incorporate this principle in multiple ways while still remaining flexible.

Most obvious ist the gathering of all related code under the umbrellar of a single HTML tag. Grouping together HTML, JS and CSS Code in a safe, sandboxed environment exposes the possibility to build more cohesive and understandable services. In the typical global nature of web development those three pillars are separated. This circumstance left the developer switching back and forth between code bases trying to find a tricky way to glue related parts together.

Secondly, the sub-standard *custom elements* introduces so called lifecycle methods and a getter/setter interface

exposing the functionality to the developer. Event handling, for example, can be registered in place which is much more declarative than assigning event listeners from the outside. Of course, this events can be pushed down to nested tags, allowing an increasingly granular system design. This approach will be explained further in the upcoming sections.

The concept of microservices incooperates not only a technical perspective. Microservice patterns are a product of real-world usage.[2, p. 1] In a real world we typical have to deal with the so called *Conway's Law*:

> "organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations". [4]

Following this logic any company, whether its web-related or not, should be devided in units grouped around a destinct business service to optimise the workflow. Fowler and Lewis outlines this approach as an "alignment of business capabilities"[5, p. 2] While this kind of structure may be true for companies like Google or Amazon, there is a vast majority of companies developing for the web which are grouped around tasks.[6] A very common structure is formed by the technology stack: UX Designers, Frontend- & Backend Developers.

The microservice approach models all those functions into business-related units. Web components are one (but important) way to tie up those diciplines as one component can host a single independent business service. Combined with a flexible backend service these components can be huge gain over the cumbersome functional organizational approach.

Macroperspektive / Composition

http://alistair.cockburn.us/Hexagonal+architecture

MVC Pattern

Progressive Enhancement

Chapter about progressive enhancement

title: Browsernative Microservices subtitle: Modulare Webarchitekturen durch neue W3C-Spezifikationen author: Jan Peteler, FH Würzburg-Schweinfurt, jan.peteler@student.fhws.de date: Januar 2017 lang: de-DE abstract: Webapplikationen lassen sich oft nur mit zusätzlichen Abstraktionsebenen wie beispielsweise "Frameworks" verwirklichen. Neue Browserstandards ermöglichen eine native Servicearchitektur, die direkt auf der Browser-Engine aufsetzt. Diese Charakteristiken sollen systematisch erfasst werden. ...

[1] D. Abramov, "Presentational and Container Components – Medium." 2015 [Online]. Available: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0. [Accessed: 01-Dec-2016]

[2] S. Newman, *Building microservices*. O'Reilly Media, Inc., 2016 [Online]. Available: http://www.ebook.de/de/product/22539693/sam_newmann_building_microservices.html

[3] R. C. Martin, "The single responsibility principle." [Online]. Available: http://programmer.97things.oreilly.com/wiki/index.php/The_Single_Responsibility_Principle

[4] M. E. Conway, "How do committees invent?" 1968 [Online]. Available: http://www.melconway.com/Home/Committees_Paper.html

[5] M. Fowler and J. Lewis, "Microservices: Nur ein weiteres konzept in der softwarearchitektur oder mehr?" *OBJEKTspektrum*, Jan. 2015.

[6] B. Issa, "The way of the web." Polymer Summit 2016, Oct-2016 [Online]. Available: https://www.youtube.com/watch?v=8ZTFEhPBJEE