Todo list

Bachelorarbeit 2

Entwicklung von wiederverwertbaren Oberflächen mit Hilfe von Web Components und Google Polymer

Welche Vor- und Nachteile gibt es bei der Entwicklung von wiederverwertbaren Oberflächen mit reinen Web-Components bzw. Google Polymer?

StudentIn Georg Eschbacher, 1110601005
BetreuerIn Hubert Hölzl

Salzburg, am X. X 2014

Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst, und keine anderen als die angegeben Quellen und Hilfsmittel benutzt habe. Weiters versichere ich hiermit, dass ich die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission weder im In- noch im Ausland vorgelegt und auch nicht veröffentlicht.

Datum Unterschrift

Kurzfassung

Vor- und Zuname:	Vorname NACHNAME
Institution:	FH Salzburg
Studiengang:	Bachelor MultiMediaTechnology
Titel der Bachelorarbeit:	Die Bachelorarbeit und ihre Folgen
Begutachter:	Titel Vorname Nachname

Deutsche Zusammenfassung ...

... zwischen 150 und 300 Worte ...

Schlagwörter: Folgen, Bachelor, Wissenschaftliches Arbeiten

Abstract

English abstract ...

... between 150 and 300 words ...

Keywords: *a few descriptive keywords*

Inhaltsverzeichnis

1 Einführung

Remove Color of Hyperref package (see Bakk.tex comments).

Check version of Polymer

Check version of W3C draft

Update and rewrite Chapter 1 completly

Put connections between paragraphs in Chapter 2

Referencing original papers/books with: autociteX nach autociteY

Look up if there is any component collection available

Nutzwerkanalyse von Web-Components im Allgemeinen

Nutzwerkanalyse von Web-Components im allgemeinen

1.1 Relevanz

Zu Beginn muss geklärt werden, was eine Softwarekomponente im Allgemeinen definiert. 1996 wurde die Softwarekomponente bei der European Conference on Object-Oriented Programming (ECOOP) folgendermaßen definiert (Szyperski, Gruntz und Murer 2002):

> „A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.“

Um dies näher zu erläutern, wird ein gängiger Tätigkeitsbereich eines Softwarearchitekten herangezogen: das Erstellen einer Liste von Komponenten, die in die gesamte Architektur problemlos eingefügt werden kann. Diese Liste gibt dem Entwicklungsteam vor, welche Komponenten das Softwaresystem zum Schluss umfassen wird. In einfachen Worten zusammengefasst sind Softwarekomponenten die Teile, die eine Software zum Ganzen machen (Szyperski, Gruntz und Murer 2002).
Komponentenentwicklung steht für die Herstellung von Komponenten, die eine spezielle Funktion in einer Softwarearchitektur übernehmen. Sämtliche Komponenten sollten immer für sich gekapselt und unabhängig von einander sein. Mehrere Komponenten werden mit Hilfe eines Verbindungsverfahren zusammengeführt beziehungsweise verwendet.
Softwarekomponente haben ihren Ursprung im „Unterprogramm“. Ein „Unterprogramm“, oder auch „Subroutine“ genannt, ist ein Teil einer Software, die von anderen Programmen beziehungsweise Programmteilen aufgerufen werden kann. Eine Subroutine gilt als Ursprung der ersten Einheit für die Softwarewiederverwendung (Wheeler 1952). Programmierer entdeckten, dass sie sich auf die Funktionalität von zuvor geschriebenen Codesegmenten berufen können, ohne sich beispielsweise um ihre Implementierung kümmern zu müssen. Neben der Zeitersparnis, die dadurch entstand, erweiterte diese „Technik “ unser Denken: der Fokus kann auf neue Algorithmen und komplexere Themen gelegt werden. Des Weiteren entwickelten sich auch die Programmiersprachen weiter. Anspruchsvolle Subroutines waren ununterscheidbar von primitiven, atomaren Anweisungen (Szyperski, Gruntz und Murer 2002).
Komponentenbasierte Softwareentwicklung entwickelte sich in einer ununterbrochenen Linie von diesen frühen Anfängen. Moderne Komponenten, von denen die meisten webbasierte Services sind, sind viel größer und viel anspruchsvoller, als früher. Des Weiteren haben die neuen Komponenten

eine dienstorientierte Architektur, was zu höherer Komplexität betreffend der Interaktionsmechanismen führt, als die damaligen Subroutinen (Andresen 2003).
In der gleichen Weise, wie die frühen Subroutinen die Programmierer vom Nachdenken über spezifische Details befreit haben, verschiebt sich durch komponentenbasierte Softwareentwicklung der Schwerpunkt von direkter Programmiersoftware zu komponierten Softwaresystemen, sprich komponentenbasierten Systemen. Auf der Grundlage, dass der Schwerpunkt auf der Integration von Komponenten liegt, basiert die Annahme, dass es genügend Gemeinsamkeiten in großen Softwaresystemen gibt, um die Entwicklung von wiederverwendbaren Komponenten zu rechtfertigen und um dafür diese Gemeinsamkeiten ausnutzen zu können. Heute werden Komponente gesucht, die eine große Sammlung von Funktionen bereitstellen. Beispielsweise wird aus unternehmerischer Sicht nicht nach einem simplen Adressbuch, um die Daten beziehungsweise Kommunikation seiner Kunden zu sichern, sondern eine CRM-System[1] gesucht. Darüberhinaus sollte dieses System auch flexibel sein, d.h. es sollte mit anderen Komponenten erweiterbar sein. Die Auswirkungen, wenn die Kontrolle über die Zerlegung in die jeweiligen Komponenten vorhanden ist beziehungsweise wenn diese Kontrolle nicht vorhanden ist, werden in der Bachelorarbeit näher besprochen(Andresen 2003).
Komponentenbasierte Entwicklung dient der Verwaltung von Komplexität in einem System. Sie versucht diese Verwaltung zu erreichen, indem die Programmiererin und der Programmierer sich vollständig auf die Implementierung der Komponente fokussieren. Das Verknüpfen beziehungsweise Kombinieren von Komponenten sollte nicht mehr Aufgabe des Komponentenentwickler sein. Um diese Aufgabe zu lösen wird ein spezielles Framework oder externe Struktur der Komponente verwendet. Vorteile durch dieses Programmierparadigma sind somit einerseits die Zeitersparnisse und andererseits die erhöhte Qualität der Komponenten. Dadurch, dass bei der komponentenbasierten Entwicklung die Wiederverwendbarkeit von Komponenten im Vordergrund steht, gibt es verschiedene Anwendungsszenarien, die automatisch als Testszenarien dienen(Andresen 2003).

Short history of Web-Components before the new technology

Im Zeitalter von Internet, Intranet und Extranet sind viele verschiedene Systeme und Komponenten miteinander zu verzahnen. Web-basierte Lösungen sollen auf Informationen eines Hostrechners zugreifen können, um z.B. mittels eines Unternehmensübergreifenden Extranets diversen Zwischenhändlern transparente Einblicke in die Lagerbestände eines Unternehmens zu ermöglichen. ERP[2]- und CRM-Systeme sollen eingebunden werden, um die Betreuung und den Service für Kunden zu optimieren. Aus diesen Gründen ist eine erweiterbare und flexible Architektur notwendig, die eine schnelle Reaktion auf neue Anforderungen ermöglicht.

überleitung

Das Komponentenmodell, das 2013 in einem Arbeitsentwurf des W3C veröffentlicht wurde und auch „Web-Components“ genannt wird, besteht aus fünf teilen:

Templates beinhalten Markup, das vorerst inaktiv ist, aber bei späterer Verwendung aktiviert werden kann.

Decorators verwenden CSS-Selektoren basierend auf den Templates, um visuelle beziehungsweise verhaltensbezogene Änderungen am Dokument vorzunehmen.

Custom Elements ermöglichen eigene Elemente mit neuen Tag-Namen und neuen Skript-Schnittstellen zu definieren.

1. Ein „Customer Relationship Management System“ bezeichnet ein System, das zur Kundenpflege beziehungsweise Kundenkommunikation dient.
2. Enterprise Resource Planning

Shadow DOM erlaubt es eine DOM-Unterstruktur vollständig zu kapseln, um so zuverlässigere Benutzerschnittstellen der Elemente garantieren zu können.

Imports definieren, wie Templates, Decorators und Custom Elements verpackt und als eine Ressource geladen werden können.

Dieses Komponentenmodell hat das Potenzial, die entwicklung von Web-Applikationen enorm zu vereinfachen und zu beschleunigen. Mit Hilfe von „Custom Elements“ und „Imports “ lassen sich eigene, komplexe HTML-Elemente selbst bauen oder von anderen entwickelte Elemente in der eigenen Applikation oder Website nutzen. Beispielsweise könnte ein eigenes HTML-Element von einer einfachen Überschrift mit fest definiertem Aussehen, über einen Videoplayer, bis hin zu einer kompletten Applikation, alles sein. Vieles, was heute über Javascript-Bibliotheken abgewickelt wird, könnte künftig in Form einzelner Webkomponenten umgesetzt werden. Das verringert Abhängigkeiten und sorgt für mehr Flexibilität. Bis die dafür notwendigen Webstandards aber verabschiedet, in Browsern umgesetzt und diese bei ausreichend Nutzern installiert sind, wird aber noch einige Zeit vergehen. Google hat daher mit Polymer eine Bibliothek entwickelt, die die Nutzung von Webkomponenten schon heute ermöglicht und dazu je nach den im Browser vorhandenen Funktionen die fehlenden Teile ergänzt.
Die persönliche Motivation beziehungsweise Relevanz zu diesem Thema ist durch das praxisnahe Projekt gegeben. Hierbei wird der Fokus auf die Programmierung von zwei wiederverwertbaren Frontend-Oberflächen gelegt. Zum einen wird eine Diagramm-Komponente und zum anderen ein komplexes Menü entwickelt, dass für folgende Projekte weiterverwendet werden kann. Die Hauptanforderung ist die Kompatibilität zu so vielen Browsern wie möglich zu gewährleisten. Des Weiteren soll durch die einmalige Programmierung dieser Komponente und deren darauffolgende Wiederverwendung den Wartungsaufwand möglichst gering zu halten.

1.2 Forschungsfeld und Forschungsfrage

Wie siehts mit der cross-browser Kompatiblität von WebComponents aus? Hilft da Google Polymer?

Aus den bisherigen Überlegungen ergibt sich folgende Forschungsfrage:
Inwiefern sind Web-Components bereits Cross-Browser kompatibel? Ist es mit Hilfe von Google-Polymer möglich, Web-Components bereits in der Produktion von Webseiten einzusetzen?

1.3 Struktur der Arbeit

2 Softwarearchitektur und Softwarekomponenten

Das Kapitel 2 sollte der Leserin und dem Leser verhelfen, den Begriff „Softwarearchitektur“ zu verstehen. Demnach sollen Vorteile von der Verwendung von Softwarearchitektur genannt werden. Des Weiteren werden wichtige Begriffe wie serviceorientierte Architektur beziehungsweise komponentenbasierte Softwarearchitektur näher erläutert. Auch wird in Kapitel 2 nur Architektur behandelt, die sich über die Erstellung, Auslieferung und den Betrieb von Software jeglicher Art erstreckt. Folglich gibt es Berührungspunkte zu anderen Architektur-Arten wie zum Beispiel der Daten- oder Sicherheitsarchitektur, die jedoch nicht in dieser Arbeit behandelt werden.

2.1 Klassische Softwarekomponente

„The characteristic properties of a component are that it:

- is a unit of independent deployment

- is a unit of third-party composition
- has no (externally) observable state“

Dies ist die Definition einer Komponente von Clemens Szyperski im Buch „Component software: Beyond object-oriented programming“(Szyperski, Gruntz und Murer 2002). Diese Definition bedarf weiterer Erläuterung.

A component is a unit of independent deployment
Dieser Punkt der Definition besitzt eine softwaretechnische Implikation. Damit eine Komponente „independent deployable“ also unabhängig auslieferbar ist, muss sie auch so konzipiert beziehungsweise entwickelt werden. Sämtliche Funktionen der Komponente müssen vollständig unabhängig von der Verwendungsumgebung und von anderen Komponenten sein. Des Weiteren muss der Begriff „independent deployable“ als Ganzes betrachtet werden, denn es bedeutet, dass eine Komponente nicht partiell, sondern nur als Ganzes ausgeliefert wird. Ein Beispiel für diesen Kontext wäre, dass ein Dritter keinen Zugriff auf die involvierten Komponenten einer Software hat.

A component is a unit of third-party composition
Hier wird der Begriff „composable“ dahingehend verstanden, dass Komponenten zusammensetzbar sein sollen. In diesem Kontext bedeutet dies, dass eine Applikation aus mehreren Komponenten bestehen kann. Des Weiteren soll auch mit einer Komponente interagiert werden können, was einer klar definierten Schnittstelle bedarf. Nur mit Hilfe dieser Schnittstelle kann garantiert werden, dass die Komponente einerseits vollständig gekapselt von anderen Komponenten ist und andererseits mit der Umgebung interagieren kann. Dies erfordert demnach eine klare Spezifikation, was die Komponente erfordert und was sie bietet.

A component has no (externally observable) state
Eine Komponente sollte keinen (externen) „observable“also feststellbaren Zustand haben. Die Originalkomponente darf nicht von Kopien ihrer selbst unterschiedlich sein. Wenn Komponenten einen feststellbaren Zustand haben dürften, wäre es nicht möglich, zwei „gleiche“ Komponenten mit den gleichen Eigenschaften zu haben. Eine mögliche Ausnahme von dieser Regel sind Attribute, die nicht zur Funktionalität der Komponente beitragen. Ein Beispiel in diesem Kontext wäre die Seriennummer für die Buchhaltung. Dieser spezifische Ausschluss ermöglicht einen zulässigen technischen Einsatz eines Zustands, der kritisch für die Leistung sein könnte. Beispielsweise dafür sei der Cache. Eine Komponente kann einen Zustand mit der Absicht zu cachen verwenden. Ein Cache ist ein Speicher, auf den man ohne Konsequenzen verzichten kann, jedoch möglicherweise reduzierte Leistung in Anspruch nimmt. umschreiben!

In vielen aktuellen Ansätzen sind Komponenten eine schwerwiegende Einheit mit genau einer Instanz in einem System. Beispielsweise könnte ein Datenbankserver eine Komponente darstellen. Oftmals wird der Datenbankserver im Zusammenhang mit der Datenbank als Modul mit einem feststellbaren Zustand angesehen. Dahingehend ist der Datenbankserver ein Beispiel für eine Komponente und die Datenbank ein Beispiel für das Objekt, das von der Komponente verwaltet wird. Es ist wichtig zwischen dem Komponentenkonzept und dem Objektkonzept zu differenzieren, da das Komponentenkonzept in keinster weise den Gebrauch von Zuständen von Objekten fördert beziehungsweise zurückstuft (Szyperski, Gruntz und Murer 2002).

Eine Softwarearchitektur ist die zentrale Grundlage einer skalierbaren Softwaretechnologie und ist für komponentenbasierte Systeme von größter Bedeutung. Nur da, wo eine Gesamtarchitektur mit Wartbarkeit definiert ist, finden Komponenten die Grundlage, die sie benötigen. Folgend werden einige Eckpfeiler einer Komponentenarchitektur genannt (Szyperski, Gruntz und Murer 2002):

- Interaktionen zwischen Komponenten und deren Umfeld sind geregelt

- Die Rollen von Komponenten sind definiert
- Schnittstellen von Komponenten sind standardisiert
- Aspekte der Benutzeroberflächen für Endbenutzer und Assembler sind geregelt

In Kapitel 2.3 auf Seite 6 wird der Begriff der Softwarearchitektur näher erläutert. Des Weiteren werden in Kapitel 2.3.1 auf Seite 8 und Kapitel 2.3.2 auf Seite 9 zwei Aspekte der Softwarearchitektur hinsichtlich der Entwicklung mit Komponenten erklärt.

2.2 Arten von Softwarekomponenten

Verschiedene Arten von Komponenten können entsprechend ihren Aufgabenbereichen klassifiziert werden. Eine übersichtliche Art der Zuordnung von Aufgabenbereichen zu Komponenten kann auf der Basis der Trennung von Zuständigkeiten erfolgen, zum Beispiel auf der Basis einer Schichten-Architektur. Eine Schichtenarchitektur dient der Trennung von Zuständigkeiten und einer losen Kopplung der Komponenten. Sie unterteilt ein Software-System in mehrere horizontale Schichten, wobei das Abstraktionsniveau der einzelnen Schichten von unten nach oben zunimmt. Eine jede Schicht bietet der über ihr liegenden Schicht Schnittstellen an, über die diese auf sie zugreifen kann. Abbildung 1 auf Seite 5 veranschaulicht die bereits genannte Architektur (Andresen 2003).

Abbildung 1: Beispiel einer Schichtenarchitektur

Auf Grundlage der in Abbildung 1 auf Seite 5 veranschaulichten Architektur können Komponenten wie folgt unterteilt werden:

Komponenten der Präsentations-Schicht
Diese Komponenten stellen eine nach außen sichtbare Benutzerschnittstelle dar. Ein Beispiel dieser Schnittstelle ist eine GUI[3]-Komponente (Button, Menü, Slider, und vieles mehr...).

Komponenten der Controlling-Schicht
Diese verarbeiten komplexe Ablauflogik und dienen als Vermittler zwischen Komponenten der Business- und Präsentations-Schicht.Beispielhaft hierfür wäre ein Workflow-Controller. Er koordiniert die Interaktion mit einer oder mehreren Businesskomponenten. Diese Komponente kann zum Beispiel den Ablauf eines Geschäftsprozesses umsetzen.

Komponenten der Business-Schicht
Sie bilden die Geschäftslogik im Sinne autonomer Businesskonzepte ab. Geschäftslogik in diesem Kontext bedeutet, dass die Komponente die Logik besitzt, die die eigentliche Problemstellung löst. Oftmals bezeichnet man hier die sogenannte Entity-Komponente. Sie dient der Perstistierung von Daten beziehungsweise bildet Entitäten auf Komponenten ab (Firma, Produkt, Kunden).

3. Graphical User Interface

Komponenten der Integrations-Schicht
Sie dient der Anbindung an Alt-Systeme, Fremd-Systeme und Datenspeicher. Dies könnten eispielsweise Connector-Komponenten oder Datenzugriffs-Komponenten sein. Connector-Komponenten dienen der Integration eines Fremd-Systems und Datenzugriffs-Komponenten liefert den Datenzugriff für Komponenten der oben genannten Schichten. Bei Datenzugriffs-Komponenten werden die Besonderheiten einer Datenbank berücksichtigt.

2.3 Softwarearchitektur

Architektur ist nicht ausschließlich eine technologische Angelegenheit, sondern beinhaltet zahlreiche soziale und organisatorische Gesichtspunkte, die den Erfolg einer Architektur und damit eines gesamten Projekts erheblich beeinflussen können. Wenn man bedenkt, dass Architektur in verschiedenen Bereichen ein Thema ist und unterschiedliche Aspekte bei der Erstellung eines Systems umfasst, wird deutlich, warum eine allgemeingültige Definition schwer fällt (Vogel 2009).
Zu Beginn wird die klassische Architektur als Ausgangspunkt verwendet. Eine mögliche Definition der klassischen Architektur bietet das „American Heritage Dictionary[4]“:

> Architecture is:
>
> 1. The art and science of designing and erecting buildings.
> 2. A style and method of design and construction
> 3. Orderly arrangement of parts

Wenn man diese Definition zugrunde legt, ist Architektur sowohl eine Kunst als auch eine Wissenschaft, die sich sowohl mit dem Entwerfen als auch mit dem Bauen von Bauwerken beschäftigt. Sie konzentriert sich nicht nur auf die Planung, sondern erstreckt sich bis hin zu der Realisierung eines Bauwerks. Ferner ist ein Schlüsselergebnis der Architekturtätigkeit das Arrangieren von Teilen des Bauwerks. Laut dieser Definition ist Architektur hiermit nicht nur die Struktur eines Bauwerks, sondern auch die Art und Weise, an etwas heranzugehen. Generell entstehen Architekturen auf Grund von Anforderungen, wie beispielsweise der Wunsch nach einer Behausung und unter Verwendung von vorhandenen Mitteln wie zum Beispiel Werkzeugen. Historisch basiert der eigentliche Entwurf auf dem Prinzip von Versuch und Irrtum. Erst durch die gewonnenen Architektur-Erfahrungen, welche mündlich oder schriftlich weitergegeben wurden, entwickelten sich Architekturstile. Folglich basiert Architektur auf Konzepten beziehungsweise Methoden, die sich in der Vergangenheit bewährt haben (Vogel 2009).

Zum Begriff „Architektur“ in der IT existieren im Gegensatz zur klassischen Architektur unzählige Definitionen[5]. Daran zeigt sich, dass es eine Herausforderung darstellt, eine Definition zu finden, die allgemein anerkannt wird (Shaw und Garlan 1996).

Softwarearchitektur erstreckt sich von der Analyse des Problembereichs eines Systems bis hin zu seiner Realisierung. Sie bewegt sich nicht auf der Abstraktionsebene fein-granularer Strukturen wie Klassen oder Algorithmen, sondern vielmehr auf der Ebene von Systemen, also grob-granularer Strukturen. Oftmals werden bei Projekten keine Aufwände im Zusammenhang mit Architektur bezahlt, was dazu führt, dass es im späteren Verlauf der Entwicklung zu vermeidbaren höheren finanziellen Kosten auf Grund eines erhöhten Wartungsaufwands kommen kann (Vogel 2009).

Symptome mangelhafter Softwarearchitektur
Fatalerweise zeigen sich die Folgen einer mangelhaften Architektur in der IT nicht selten erst

4. Siehe American Heritage Online-Dictionary
5. Das Software Engineering Institute(SEI) der Carnegie-Mellon Universität der Vereinigten Staaten von Amerika hat in der Fachliiteratur über 50 verschieene Definitionen für den Begriff „Softwararchitektur“ ausgemacht.Software Architecture Definitions

mit erheblicher Verzögerung, das heißt, ernste Probleme treten eventuell erst auf, wenn ein System zum ersten Mal produktiv eingesetzt wird oder wenn es bereits im Einsatz ist und für neue Anforderungen angepasst werden muss. Eine Architektur, die ungeplant entstanden ist, sich also unbewusst im Laufe der Zeit entwickelt hat, führt zu erheblichen Problemen während der Erstellung, der Auslieferung und dem Betrieb eines Systems. Folgende Symptome können potentiell auf eine mangelhafte Architektur hindeuten (Vogel 2009):

- Fehlender Gesamtüberblick
- Komplexität ufert aus und ist nicht mehr beherrschbar
- Planbarkeit ist erschwert
- Risikofaktoren frühzeitig erkennen ist kaum möglich
- Wiederverwendung von Wissen und Systembausteinen ist erschwert
- Wartbarkeit ist erschwert
- Integration verläuft nicht reibungslos
- Performanz ist miserabel
- Architektur-Dokumentation ist unzureichend
- Funktionalität beziehungsweise Quelltext ist redundant
- Systembausteine besitzen zahlreiche unnötige Abhängigkeiten untereinander
- Entwicklungszyklen sind sehr lang

Folgen mangelhafter Softwarearchitektur

Diese sind folgende (Vogel 2009):

- Schnittstellen, die schwer zu verwenden beziehungsweise zu warten sind weil sie einen zu großen Umfang besitzen.
- Quelltext, der an zahlreichen Stellen im System angepasst werden muss, wenn Systembausteine, wie beispielsweise Datenbank oder Betriebssystem, geändert werden.
- Klassen, die sehr viele ganz unterschiedliche Verantwortlichkeiten abdecken und deshalb nur schwer wiederzuverwenden sind ("MonsterKlassen).
- Fachklassen, deren Implementierungsdetails im gesamten System bekannt sind.

Vorteile von Architektur

Unabhängig davon, welche Art von System entwickelt wird, legt eine Architektur ausgehend von der Anforderungen an das System immer die Fundamente und damit die tragenden Säulen, jedoch nicht die Details für das zu entwickelnde System fest ((Buschmann 1996) nach (Vogel 2009)). Architektur handelt also von den Fundamenten, ohne auf deren interne Details einzugehen. Folgende Fragen im Hinblick auf ein System werden durch eine Architektur beantwortet:

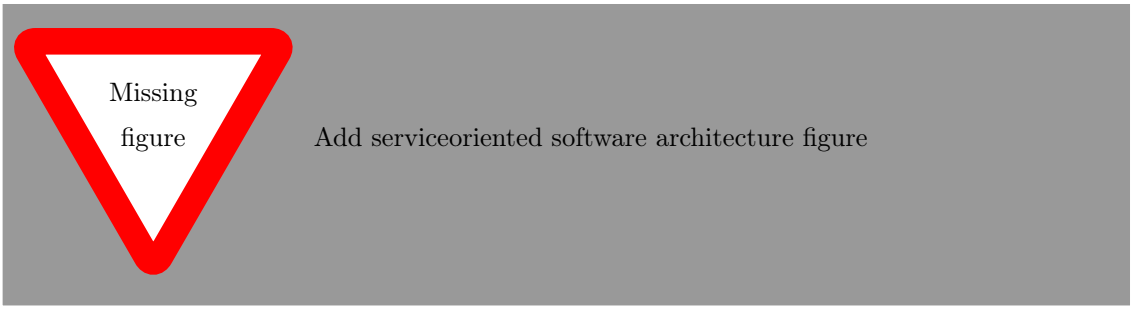
- Auf welche Anforderungen sind Strukturierung und Entscheidungen zurückzuführen?
- Welches sind die wesentlichen logischen und physikalischen Systembausteine?
- Wie stehen die Systembausteine in Beziehung zueinander?
- Welche Verantwortlichkeiten haben die Systembausteine?
- Wie sind die Systembausteine gruppiert beziehungsweise geschichtet?
- Was sind die Festlegungen und Kriterien, nach denen das System in Bausteine aufgeteilt wird?

Architektur beinhaltet demnach alle fundamentalen Festlegungen und Vereinbarungen, die zwar durch die fachliche Anforderungen angestoßen worden sind, sie aber nicht direkt umsetzt.

Ein wichtiges Charakteristikum von Architektur ist, dass sie Komplexität überschaubar und handhabbar macht, indem sie nur die wesentlichen Aspekte eines Systems zeigt, ohne zu sehr in die Details zu gehen, und es so ermöglicht, in relativ kurzer Zeit einen Überblick über ein System zu erlangen.

Die Festlegung, was genau die Fundamente und was die Details eines Systems sind, ist subjektiv beziehungsweise kontextabhängig. Gemeint sind in jedem Fall die Dinge, welche sich später nicht ohne Weiteres ändern lassen. Dabei handelt es sich um Strukturen und Entscheidungen, welche für die Entwicklung eines Systems im weiteren Verlauf eine maßgebliche Rolle spielen (Fowler 2005). Beispiele hierfür sind die Festlegung, wie Systembausteine ihre Daten untereinander austauschen oder die Auswahl der Komponentenplattform[6]. Derartige architekturrelevante Festlegungen wirken sich systemweit aus im Unterschied zu architekturirrelevanten Festlegungen (wie beispielsweise eine bestimmte Implementierung einer Funktion), die nur lokale Auswirkungen auf ein System haben (Bredemeyer und Malan 2004).

2.3.1 Serviceorientierte Softwarearchitektur



In diesem Subkapitel wird serviceorientierte Softwarearchitektur mit „SOA“ abgekürzt.

Thomas Erl definiert im Buch „Software-oriented Architecture (Concepts, Tecnology, and Design)“ den Begriff SOA wie folgt : Referenz einfügen

”

- SOA represents an open, agile, extensible, federated, composable architecture comprised of autonomois, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services.
- SOA can establish an abstaction of business logic and technology that may introduce changes to business process modeling and technical architecture, resulting in a loose coupling between these models.
- SOA is an evolution of past platforms, preserving successful characteristics of traditional architectures, and bringing with it distinct principles that foster service-orientation in support of a service-oriented enterprise.
- SOA is an evolution of past platforms, preserving successful characteristics of traditional architectures, and bringing with it distinct principles that foster service-orientation in support of a service-oriented enterprise. “

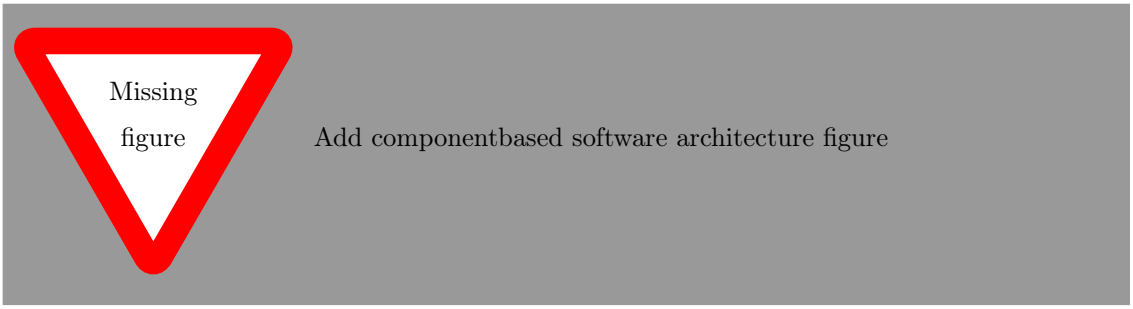
SOA is a form of technology architecture that adheres to the principles of service-orientation. When realized through the Web services technology platform, SOA establishes the potential to support and promote these principles throughout the business process and automation domains of an enterprise.

6. Beispiele für Komponentenplattformen sind JEE, .NET, Adobe AIR und viele mehr...

2.3.2 Komponentenbasierte Softwarearchitektur

Im Gegensatz zu dem bereits definierten Begriff der Softwarearchitektur muss er hinsichtlich der komponentenbasierten Softwarearchitektur wie folgt erweitert werden. Softwarearchitektur erstreckt sich von der Analyse des Problembereichs eines Systems bis hin zu seiner Realisierung. Sie bewegt sich nicht auf der Abstraktionsebene fein-granularer Strukturen wie Klassen oder Algorithmen, sondern vielmehr auf der Ebene von Systemen, also grob-granularer Strukturen.

Sie ist sowohl die Identifikation, als auch die Dokumentation der einerseits statischen Struktur und andererseits der dynamischen Interaktion eines Software Systems, welches sich aus Komponenten und Systemen zusammensetzt. Dabei werden sowohl die Eignschaften der Komponenten und Systeme als auch ihre Abhängigkeiten und Kommunikationsarten mittels spezifischer Sichten beschrieben und modelliert. Softwarearchitektur betrifft alle Artefakte der Software Entwicklung (Andresen 2003).



2.4 Softwarekomponente für den Webbereich

2.5 Konklusion

Dieses Kapitel hilft bei der Beantwortung mehrerer Subfragen des Forschungsfeld. Es wird erklärt wie eine klassische Softwarekomponente definiert ist und in welchen Zusammenhang eine Komponente mit Softwarearchitektur steht. Daraufhin wird erläutert, was eine Softwarearchitektur ist und inwiefern dies für diese Arbeit relevant ist. Demnach werden zwei Aspekte der Softwarearchitektur genauer erläutert, da diese ausschlaggebend für die Entwicklung von Web-Components sind (siehe Kapitel **??** auf Seite **??**).

Auf die passende Seite referenzieren

3 Web-Components

HTML Templates W3C Draft (18.März.2014) https://dvcs.w3.org/hg/webcomponents/raw-file/tip/spec/templates/index.html

Um Web-Components besser verstehen zu können, wird in diesem Kapitel zu Beginn eine kurze Übersicht über die Geschichte von Web-Bibliotheken gezeigt.

2005 Veröffentlichung von Dojo Toolkit[7] mit der innovativen Idee von Widgets. Mit ein paar Zeilen Code konnten Entwickler komplexe Elemente, wie beispielsweise einen Graph oder eine Dialog-Box in ihrer Website hinzufügen.

7. Mehr Information zu Dojo Toolkit unter http://dojotoolkit.org/

2006 jQuery[8] stellt Entwicklern die Funktion zur Verfügung Plugins zu entwickeln, die später wiederverwendet werden können.

2008 Veröffentlichung von jQuery UI[9], was vordefinierte Widgets und Effekte mit sich bringt.

2009 Erstveröffentlichung von AngularJS[10], ein Framework mit Direktiven.

2011 Erstveröffentlichung von React[11]. Diese Bibliothek gibt den Entwicklern die Fähigkeit, das User Interface ihrer Website zu bauen, ohne dabei auf andere Frameworks, die auf der Seite benutzt werden, achten zu müssen

2013 Veröffentlichung des Entwurfs von Web-Components, jedoch mit schlechter Browser Unterstützung

Mit der Veröffentlichung von Dojo Toolkit sagen Entwickler die Vorteile von wiederverwendbaren Modulen. Wenn man zurzeit Plugins auf einer Website erwähnt, denken die meisten Entwickler von jQuery Plugins, da sie beinahe überall Verwendung finden und ein großes Spektrum von Funktionen bieten. Mit den Veröffentlichungen von AngularJS und React wurde gezeigt, in welche Richtung sich Web-Anwendungen bewegen. Sie zeigen, dass es nicht nur um visuelle Elemente geht, sondern auch um Elemente, die eine komplexe Logik besitzen.

Ähnlich zu HTML5 ist Web-Components ein Sammelbegriff für mehrere Features:

Shadow DOM (ausführliche Erklärung siehe Kapitel 3.2.4 auf Seite 14) erlaubt es das DOM und CSS zu kapseln

HTML Templates (ausführliche Erklärung siehe Kapitel 3.2.1 auf Seite 12) sind ein Weg, um den DOM zu klonen und somit den Klon wiederzuverwenden

Custom Elements (ausführliche Erklärung siehe Kapitel 3.2.3 auf Seite 14) können einerseits neue Elemente definieren, oder bereits bestehende Elemente erweitern. Dies bedeutet, dass ein Entwickler beispielsweise den HTMl `<input>`-Tag dahingehend erweitern kann, dass dieser nur das Format von Kreditkartennummern unterstützt. Ein Beispiel für die Definition eines neuen Elements wäre ein Element, dass sämtliche Felder, die für die Bezahlung mit einer Kreditkarte notwendig sind, bereitstellt.

HTML Imports (ausführliche Erklärung siehe Kapitel 3.2.5 auf Seite 14) sind dazu da, um externe HTML-Dateien in die bestehende Website zu integrieren, ohne dabei den Code kopieren zu müssen. Sie können beispielsweise dazu verwendet werden, um Web-Components in eine Website zu integrieren.

Decorators (ausführliche Erklärung siehe Kapitel 3.2.2 auf Seite 14) sind Elemente, die nach dem „Decorator programming pattern“ benannt sind. Durch dieses Pattern ist es möglich Elemente um zusätzliche Funktionalitäten zur Laufzeit erweitern zu können.

Obwohl „Web-Components“ für viele Entwickler noch kein Begriff ist, wird es bereits vom Browser automatisch verwendet. Beispiele dafür sind der Datepicker oder das `<video>`-Elemen. Abbildung 2 auf Seite 11 zeigt die Datepicker-Komponente und Abbildung 3 auf Seite 11 zeigt den dazugehörigen Source-Code. Dieser Code zeigt, dass sämtliche Kontrollbuttons des Datepickers vor dem Entwickler „versteckt“, also im Shadow DOM liegen.

8. Mehr Information zu jQuery unter http://jquery.com/
9. Mehr Information zu jQuery UI unter http://jqueryui.com/
10. Mehr Information zu AngularJS unter http://angularjs.org/
11. Mehr Information zu Facebook React unter http://facebook.github.io/react/

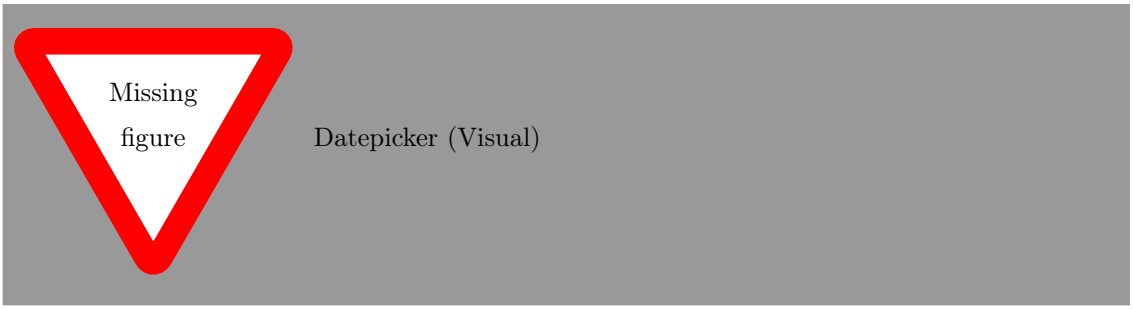


Abbildung 2: Beispiel einer Schichtenarchitektur

Abbildung 3: Beispiel einer Schichtenarchitektur

Warum Web-Components?
Javascript Widgets und Plugins sind fragmentiert, weil sie auf diversen unterschiedlichen Bibliotheken und Frameworks basieren, die möglicherweise nicht miteinander funktionieren. Web-Components versuchen einen gewissen Standard in Widgets und Plugins zu bringen. Das Problem der nicht miteinander funktionierenden Plugins versucht Web-Components mit Kapselung zu lösen. Durch die Lösung dieses Problems ist die Wiederverwendbarkeit von Komponenten garantiert, da es sämtliche Interferenzenzwischen Plugins löst. Web-Components können des Weiteren viel mehr als nur UI-Komponenten sein. Eine Bibliothek könnte bereits eine Komponente darstellen, die eine gewisse Funktionalität bereitstellt.

richtiges Wort?

Unterstützung von Web-Components
Zur Zeit ist die Hauptproblem von Web-Components die mangelhafte Browserunterstützung. Kein einziger Browser unterstützt diesen Standard zu 100%. Es gibt bereits mehrere Möglichkeiten beziehungsweise Polyfills[12], um dennoch Web-Components nutzen zu können. Beispiele hierfür sind:

- Polyfill-Webcomponents[13]
- Polymer-Project[14]
- X-tags[15]

Obwohl es mehrere Polyfills bezüglich Web-Componnts gibt, ist es egal, auf welcher Basis man seine Web-Components programmiert, denn durch die Standardisierung ist die Interkompatibilität der

12. Ein Polyfill ist ein Browser-Fallback, um Funktionen, die in modernen Browsern verfügbar sind, auch in alten Browsern verfügbar zu machen.
13. Mehr Information zu Polyfill-Webcomponents unter http://github.com/timoxley/polyfill-webcomponents
14. Mehr Information zu Polymer unter http://www.polymer-project.org/
15. Mehr Information zu X-tags unter http://x-tags.org/

einzelnen Komponenten gegeben. Diese Arbeit beschränkt sich hauptsächlich auf die Entwicklung von Web-Components mit Hilfe des Standards beziehungsweise der Polyfill-Bibliothek Polymer. Durch die Benutzung von dem Polyfill Polymer funktionieren Web-Component in allen „evergreen"-Browsern[16] und Internet-Explorer 10 und neuer. Des Weiteren funktionieren sie dadurch auf mobilen Endgeräten, wo iOS6+, Chrome Mobile, Firefox Mobile, oder Android 4.4 oder höher vorhanden ist. Auch mit Hilfe von Polyfill bieten sowohl der Internet-Explorer 9 und niedriger, als auch Android-Browser 4.3 oder niedriger keine Unterstützung für Web-Components. Dies bedeutet, dass Web-Components zur Zeit noch für die Verwendung im Web bereit sind, außer man hat als Zielgruppe ausschließlich eine Plattform, die unterstützt wird.

Web-Components Alternativen Die folgenden Alternativen von Web-Components benutzen ähnliche Patterns um die gewünschte DOM-Abstrahierung zu erreichen.

React benutzt seinen eigenen „Virtual DOM"und versucht in keinster Hinsicht Web-Components zu simulieren. Folglich ist die Browser-Unterstützung von React besser, als jene der Web-Components. Ab Internet-Explorer 8 werden sämtliche Browser vollständig unterstützt. Zur Zeit wird diese Technologie in Facebooks und Instagrams Kommentarsystemen eingesetzt.

AngularJS besitzt diverse Interferenzen zu Web-Components, jedoch versucht auch diese Technologie nicht Web-Components zu simulieren, um bessere Browser-Unterstützung zu bieten (Internet Explorer 8+). Die genaue Unterscheidung bezüglich AngularJS-Direktiven und Web-Components werden in Kapitel 3.3 auf Seite 14 erklärt.

3.1 Relevanz von Web-Components hinsichtlich der Forschungsfrage

3.2 W3C Web-Components Standard

3.2.1 Templates

Laut W3C sind Templates

referenz zu W3C adden

> „a method for declaring inert DOM subtress in HTML and manipulating them to instantiate document fragments with identical contents. "

Somit sind Templates eine Methode um inaktive DOM-Subtrees im HTML zu deklarieren und manipulieren, um so sämtliche identische Dokumentfragmente mit identischem Inhalt zu instanzieren.
In Web-Applikationen muss man oft den gleichen Subtree von Elementen öfters benutzen, mit den passenden Inhalt füllen und es zum Maintree hinzufügen. Man hat zum Beispiel eine Liste von Artikel, die man mit mehreren `<li>` -Tags in das Dokument einfügen will. Des Weiteren kann jeder `<li>`-Tag weitere Elemente, wie beispielsweise einen Link, ein Bild, einen Paragraphen, etc., enthalten. Bis jetzt bot HTML keine native Möglichkeit an, eine solche Aufgabenstellung zu lösen. Mehrere Beispiele, wie Entwickler diese Aufgabe lösten, sind:

1. **Versteckte Elemente**
   Dies galt unter den Entwicklern als die einfachste, aber auch als die ineffizienteste Methode. Man gab sein Markup irgendwo in das DOM (meist mit der CSS-Eigenschaft `display: none;` am Parent-Element des Markups) und wann immer es gebraucht wurde, wurde es geklont, mit diversen Daten gefüllt und schlussendlich in das DOM eingefügt. Dies beinhaltet diverse Nachteile. Sämtliche Ressourcen, die im Markup verwendet wurden, sind bei Seitenaufruf geladen worden. Die Gesamtperformanz des Browsers wurde durch die vielzählige DOM-Traversierung beeinträchtigt.

16. Ein „Evergreen"-Browser ist ein Web-Browser, der sich automatisch beim Start updatet.

2. **String-Templates**
   Man versuchte die Probleme der Methode mit „versteckten Elementen“ zu beseitigen, indem man in einem `<script>`-Tag ein Template mit einem String definiert. Dadurch, dass diese Methode hinsichtlich Laufzeit-Parsen von `innerHtml` geht, können XSS-Attacks ermöglicht werden . Folglich wird ein Beispiel gezeigt, wo ein Template mit Hilfe eines Strings in einem `<script>`-Tag definiert wird:

Information von: http://ejohn.org micro-templating

```
1    <script type="text/html" id="test_Showcase">
2    </script>
```

Listing 1: String-Template

Folgend wird an Hand eines Beispiels, wo eine Liste von Autos benutzt wird, der neue Standard zu Templates erläutert werden:

```
1  <template id="carTemplate">
2    <li>
3      <span class="carBrand"></span>
4      <span class="carName"></span>
5    </li>
6  </template>
```

Listing 2: Web-Components Template-Standard

Ein Template, wie das aus der Listing 2 auf Seite 13, kann sowohl im `<head>`- als auch im `<body>`-Tag definiert werden. Das Template, inklusive Subtree, ist inaktiv. Dies bedeutet, dass wenn sich ein `<img>`-Tag mit einer validen Quelle in diesem Template befinden würde, würde der Browser dieses Bild nicht laden. Des Weiteren ist es nicht möglich via JavaScript ein Element des Templates zu selektieren.

```
1    document.querySelectorAll('.carBrand').length; // length ist 0
```

Listing 3: Beispiel-Selektor eines Elements in einem Template, das nicht aktiven DOM ist

Abbildung 4: Visualisierung des DOM eines inaktiven Templates

In Abbildung 4 auf Seite 13 wird gezeigt, dass das Template ein Dokument-Fragment ist. Dies bedeutet dass es ein eigenständiges Dokument ist und unabhängig vom ursprünglichen Dokument existiert. Folglich bedeutet dies, dass sämtliche `<script>`, `<form>`, `<img>`-Tags etc. nicht verwendet werden.

```
1  var template = document.getElementById('carTemplate');
2  template.content.querySelector(".carBrand").length; // length ist 1
3
4  var car = template.content.cloneNode(true);
5  car.querySelector(".carBrand").innerHTML = "Seat";
6  car.querySelector(".carName").innerHTML = "Ibiza";
```

```
7
8 document.getElementById("carList").appendChild(car);
```

Listing 4: Verwendung des Templates 2 auf Seite 13

Listing 4 auf Seite 13 basiert auf dem in Listing 2 auf Seite 13 definierten Template. Zuerst wird sich in Zeile 1 der Listing 4 das vorher definierte Template in die Variable `template` geholt. Daraufhin wird der gesamte Knoten in Zeile 4 mit Hilfe einer `deep-copy` geklont und des Weiteren mit Daten befüllt. Damit das mit Daten befüllte Listenelement auch sichtbar wird, wird es in Zeile 8 in das aktive DOM eingefügt.

3.2.2 Decorators

3.2.3 Custom Elements

3.2.4 Shadow DOM

3.2.5 HTML Imports

3.2.6 Browser Unterstützung

3.3 Google Polymer

Unterscheidung zwischen predefined Elements, self-written custom Elements und Benutzung von Elementen

Nesting von Elements bzw. Reuse beispielhaft zeigen

Ein Beispiel für ein vordefiniertes Element der Google-Polymer Bibliothek wäre das `<polymer-ajax>`-Element. Es erscheint in erster Linie als nicht sehr nützlich, jedoch versucht es, einen Standard für Entwickler bereitzustellen, um Ajax-requests zu erstellen beziehungsweise abzuwickeln. Dieses Element ist ähnlich zu folgender Funktion: `\$.ajax()`[17]. Der Unterschied zwischen den beiden Möglichkeiten, einen Ajax-Request abzuwickeln, ist, dass die `\$.ajax()`-Methode Abhängigkeiten besitzt, wohingegen die `<polymer-ajax>`-Methode vollkommen unabhängig ist.

17. Mehr Information zur jQuery.ajax-Funktion unter http://api.jquery.com/jQuery.ajax/

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

6 Repräsentative Literaturliste

Andresen, Andreas. 2003. *Komponentenbasierte softwareentwicklung: mit mda, uml und xml.* München und Wien: Hanser. ISBN: 3446222820.

Brajnik, Giorgio. 2013. *Proceedings of the 10th international cross-disciplinary conference on web accessibility.* [S.l.]: ACM. ISBN: 978-1-4503-1844-0.

Bredemeyer, Dana, und Ruth Malan. 2004. *Software architecture action guide.* `http://www.ruthmalan.com`. [Online, 30.03.2014].

Buckett, Chris. 2014. *Web components in action.* Manning Pubns Co. ISBN: 9781617291944.

Buschmann, Frank. 1996. *Pattern-oriented software architecture.* Chichester u. a.: Wiley. ISBN: 0471958697.

Dimitri Glazkov, Erik Arvidsson, Daniel Freedman, Steve Orvell, Scott J. Miles, Frankie Fu, Eric Bidelman, Shans, Rafaelw, Yvonne Yip, MattsmcNulty, Addy Osmani. 2013. *Polymer project.* `http://www.polymer-project.org/`.

Dominic Cooney, Dimitri Glazkov. 2013. *Introduction to web components.* `http://www.w3.org/TR/components-intro/`.

Eric Bidelman. 18.5.2013. *Web components.* `http://www.youtube.com/watch?v=fqULJBBEVQE`.

Fowler, Martin. 2005. *The new methodology.* `http://martinfowler.com/articles/newMethodology.html`. [Online, 30.03.2014].

Grunske, Lars, Ralf Reussner und František Plášil. 2010. *Component-based software engineering: 13th international symposium, cbse 2010, prague, czech republic, june 23-25, 2010 ; proceedings.* Bd. 6092. LNCS sublibrary. SL 2, Programming and software engineering. Berlin: Springer. ISBN: 978-3-642-13237-7.

Melzer, Ingo. 2010. Service-orientierte architekturen mit web services: konzepte - standards - praxis. *Service-orientierte Architekturen mit Web Services.*

O'Reilly, Tim. 2005. *What is web 2.0 (design patterns and business models für the next generation of software).* `http://oreilly.com/web2/archive/what-is-web-20.html`. [Online, 30.03.2014].

Pfeiffer, Daniel, und Axel Winkelmann. 2007. Ansätze zur wiederverwendung von software im rahmen der software-industrialisierung am beispiel von softwarekomponenten, service-orientierten architekturen und modellgetriebenen architekturen. *Wirtschaftsinformatik* 49 (3): 208–216. ISSN: 0937-6429, doi:`10.1007/s11576-007-0051-4`.

Shaw, Mary, und David Garlan. 1996. *Software architecture: perspectives on an emerging discipline.* Upper Saddle River und N.J: Prentice Hall. ISBN: 0131829572.

Shawn Lawton Henry. 2005. *Essential components of web accessibility.* `http://www.w3.org/WAI/intro/components.php`.

Szyperski, Clemens, Dominik Gruntz und Stephan Murer. 2002. *Component software: beyond object-oriented programming.* 2nd ed. New York und London: ACM / Addison-Wesley. ISBN: 0201745720, `http://www.sei.cmu.edu/productlines/frame_report/comp_dev.htm`.

Vogel, Oliver. 2009. *Software-architektur: grundlagen - konzepte - praxis.* 2. Aufl. Heidelberg: Spektrum, Akad. Verl. ISBN: 9783827419330.

Wheeler, David. 1952. *The use of sub-routines in programmes.* ACM. doi:10.1145/609784.609816.

Xia Cai. 2000. *Software engineering conference, 2000. apsec 2000. proceedings. seventh asia-pacific.* IEEE. ISBN: 0-7695-0915-0.