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

1.2 Forschungsfeld und Forschungsfrage

Wird zum Schluss geschrieben

Warum werden Web-Components als die Zukunft des Webs gesehen?

Welche Vor- und Nachteile haben Web-Components, die mit Google Polymer umgesetzt wurden, gegenüber Web-Components, die mit den W3C-Standards umgesetzt wurden? Um die Forschungsfrage der Arbeit beantworten zu können, müssen zuerst einige Subfragen beantwortet werden, die die Hauptforschungsfrage mit sich bringt und das Forschungsfeld definieren.
Wie ist eine klassische Softwarekomponente definiert? Im Zusammenhang mit der Definition einer Softwarekomponente wird der Begriff „Softwarearchitektur“ genannt, welcher demnach auch geklärt werden muss. Was unterscheidet klassische Softwarekomponenten von Web-Komponenten? Welche Voraussetzungen haben Web-Komponenten? Welche Probleme werden durch Web-Komponenten gelöst? Das Subfragen des Forschungsfelds werden mit Hilfe von Literatur beantwortet. Danach wird mit Hilfe eines Praxisbeispiels gezeigt, inwiefern die zuvor beantworteten Fragen mit der Entwicklung von Web-Components zusammenhängen.

1.3 Struktur der Arbeit

2 Softwarearchitektur und Softwarekomponente

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 Architektur näher erläutert. Des Weiteren wird in Kapitel 2 nur nur Architektur behandelt, die sich über die Erstellung, Auslieferung und den Betrieb von Software jeglicher Art erstreckt. Folglich gibt es Berührungspunkte zu anderen Architektur Disziplinen wie zum Beispiel der Datenarchitektur. Architekturen anderer Architektur-Disziplinen werden in dieser Arbeit jedoch nicht behandelt.

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, haben 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 2.2 auf Seite 3 wird der Begriff der Softwarearchitektur näher erläutert. Des Weiteren werden in Kapitel 2.2.1 auf Seite 5 und Kapitel 2.2.2 auf Seite 6 zwei Aspekte der Softwarearchitektur hinsichtlich der Entwicklung mit Komponenten erklärt.

2.2 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 verschieden 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[1]“:

> 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 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. Daran zeigt sich, dass es eine Herausforderung darstellt, eine Definition zu finden, die allgemein anerkannt wird. Bewusstes Architektur-Denken in der Software-Entwicklung an sich ist erst circa dreißig Jahre alt ((Shaw und Garlan 1996)). Da es nicht unmittelbar ersichtlich ist, dass wirklich jedes Software-System eine Architektur benötigt und diese auch in sich trägt, führt es dazu, dass Architektur im Zusammenhang mit Software schwer greifbar ist. Trotzdem werden

1. Siehe American Heritage Online-Dictionary

Entwickler, wenn auch oft unbemerkt, in ihrer täglichen Arbeit mit Architektur konfrontiert, weil diese implizit immer ein Aspekt von Software ist und sich nicht eliminieren, allenfalls ignorieren lässt (Vogel 2009).

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 (Vogel 2009).

Softwarearchitektur aus Kundenperspektive ist oftmals sehr schwer zu verstehen, da für den Kunden eine Softwarearchitektur keinen kommerziellen Nutzen für ein Projekt bringt. Deshalb ist der Kunde selten dazu bereit, ohne Weiteres Extra-Aufwände im Zusammenhang mit Architektur mitzutragen. Es gibt nur wenige Möglichkeiten, wie mit dieser Herausforderung umgegangen werden kann. Eine Möglichkeit wäre die Kundenseite schon früh auf die mittelfristig eigentlich vermeidbaren höheren finanziellen Kosten auf Grund eines erhöhten Wartungsaufwands hinzuweisen, die durch eine Vernachlässigung von Architektur verursacht werden (Vogel 2009).

Symptome mangelhafter Softwarearchitektur

Fatalerweise zeigen sich die Folgen einer mangelhaften Architektur in der IT nicht selten erst 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:

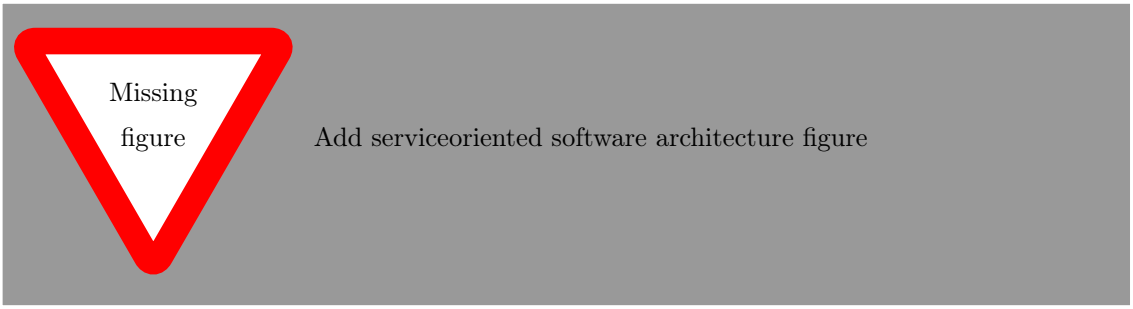
- Auf welche Anforderungen sind Strukturierung und Entscheidungen zurückzuführen?
- Welches sind die wesentlichen logischen und physikalischen Systembausteine?
- Wie stehen die Systembausteine in Beziehung zueinander?
- Welche Verantwortlichkeiten haben die Systembausteine?
- Wie sind die Systembausteine 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.

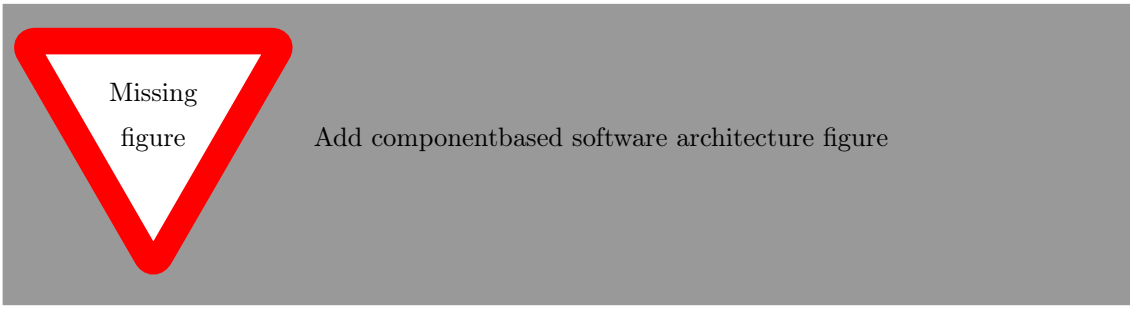
Die Festlegung, was genau die Fundamente und was die Details eines Systems sind, ist subjektiv beziehungsweise kontextabhängig. Gemeint sind in jedem Fall die Dinge, welche sich später nicht ohne Weiteres ändern lassen. Dabei handelt es sich um Strukturen und Entscheidungen, welche für die Entwicklung 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[2]. 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).

2.2.1 Serviceorientierte Softwarearchitektur



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

2.2.2 Komponentenbasierte Softwarearchitektur



2.3 Softwarekomponente für den Webbereich

2.4 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

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.*

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.