Duale Hochschule  
Baden-Württemberg  
Mosbach  
Lohrtalweg 10  
74821 Mosbach

Analyse und Vergleich der CSS Präprozessoren LESS, Sass und Stylus im Hinblick auf benötigte Einarbeitungszeit, Praxisnutzen und Vor- bzw. Nachteile gegenüber reinem CSS

Projektarbeit im Rahmen des Studiengangs Onlinemedien

**Sperrvermerk:** Nein

Inhalt

Abbildungsverzeichnis II

Tabellenverzeichnis IV

Abkürzungsverzeichnis IV

1. Einleitung 1
   1. Problemstellung 1
   2. Zielsetzung 2
   3. Vorbereitung – nichtfunktionelle Anforderungen an Quellcode 2
      1. Kriterien 2
      2. Analyse von CSS 4
      3. Funktionelle Anforderungen an CSS Präprozessoren 8
2. Hauptteil 11
   1. Präprozessoren 11
      1. SASS 12
      2. LESS 12
      3. Stylus 13
   2. Prüfung der funktionellen Anforderungen 13
      1. Grafische Oberfläche 13
      2. Geschwindigkeit 14
      3. Tests 14
   3. Entscheidungsfindung 33
3. Schluss 36
   1. Fazit 36
   2. Ausblick 36

Literaturverzeichnis 5

Ehrenwörtliche Erklärung 7

Abbildungsverzeichnis

Abbildung 1 – einfaches CSS 5

Abbildung 2 – Superset Syntax: Quellcode – LESS, SASS, Stylus 14

Abbildung 3 – Superset Syntax: Resultat – LESS, Stylus 15

Abbildung 4 – Superset Syntax: Resultat – SASS 15

Abbildung 5 – minimalistische Syntax: Quellcode – LESS, SASS, Stylus 15

Abbildung 6 – minimalistische Syntax: Ergebnis - Stylus 16

Abbildung 7 – Variablen: Quellcode - LESS 17

Abbildung 8 – Variablen: Quellcode – SASS 17

Abbildung 9 – Variablen: Quellcode - Stylus 18

Abbildung 10 – Variablen: Resultat – LESS 18

Abbildung 11 – Variablen Resultat - SASS, Stylus 19

Abbildung 12 – Mixins: Quellcode - LESS 20

Abbildung 13 – Mixins: Quellcode – SASS 20

Abbildung 14 – Mixins: Quellcode - Stylus 21

Abbildung 15 – Mixins: Resultat – LESS, SASS, Stylus 21

Abbildung 16 – Vererbung: Quellcode - LESS 23

Abbildung 17 – Vererbung: Quellcode – SASS, Stylus 24

Abbildung 18 – Selektorverkettung: Quellcode – LESS, SASS, Stylus 26

Abbildung 19 – Selektorverkettung: Resultat – LESS, SASS, Stylus 26

Tabellenverzeichnis

Tabelle 1 – Geschwindigkeit: Benchmark Ergebnisse 13

Tabelle 2 - Wertung 27

Abkürzungsverzeichnis

CSS ................................................................................. Cascading Stylesheets

SASS ........................................................... Syntactically Awesome Stylesheets

SCSS .................................................................... Sassy Cascading Stylesheets

LESS ................................................................... Leaner Cascading Stylesheets

# Einleitung

“We work with crappy tools on inconsistent platforms run by every kind of device you can imagine, all "standardized" by a toothless and glacially slow committee that mostly holds us back, with a mandate to bend the web into shapes it wasn't meant to take while making sure the product is [sic] works for everyone no matter what, all at a pace of change that many people can't keep up with—and our metric is perfection.”[[1]](#footnote-1)

Bochkariov, Frontend-Entwickler, verdeutlicht in seiner subjektiven Formulierung die Probleme, denen Frontend-Entwickler, die Zielgruppe dieser Arbeit, im Kontext der Entwicklung moderner Webanwendungen gegenüberstehen. Die Motivation dieser Arbeit ist die Erleichterung der Entwicklung von CSS, das heißt den nötigen Zeitaufwand mit einem Beitrag zu effizienteren Prozessen zu minimieren und Frustrationsquellen einzuschränken.

## Problemstellung

„CSS is complicated. It’s not object oriented. It’s not hierarchical. It’s a specificity based cascade applied to a dynamic hierarchical data structure that few people truly comprehend.“*[[2]](#footnote-2)*

Kyle Neath, GitHub Lead Designer und langjähriger Frontend-Entwickler, deutet an: Die Vielfalt der Anforderungen, denen während der Entwicklung von Weboberflächen begegnet wird, zieht aufgrund der funktionsweise von CSS eine mindestens ebenso große Vielfalt verschiedener Lösungsansätze mit weiteren Problemen nach sich.

Dieser Mangel an Konsistenz wirkt sich negativ auf die Lesbarkeit und somit auch auf Wiederverwendbarkeit und Wartbarkeit von Code aus. Die Möglichkeit, kooperativ Problemlösungen zu entwickeln und die Effizienz des Transfers von Projekten zwischen Mitarbeitern kann beeinträchtigt werden.

## Zielsetzung

Präprozessoren bergen Potenzial für einen großen Schritt in Richtung eines effizienteren Entwicklungsprozesses von CSS. Mit dem Ziel, die richtige Wahl zwischen den drei meistverbreiten CSS-Präprozessoren, LESS, SASS und Stylus zu treffen. werden sie im Rahmen dieser Arbeit auf relevante Anforderungen getestet und verglichen.

## Vorbereitung – nichtfunktionelle Anforderungen an Quellcode

Um die zuletzt erwähnten Anforderungen an Präprozessoren greifbar zu machen und zu begründen, müssen zunächst Anforderungen an erstrebenswerten Code gestellt werden.

### Kriterien

Erstrebenswerter Code erfüllt drei Anforderungen: Lesbarkeit, Wartbarkeit und Wiederverwendbarkeit. Die folgenden Abschnitte beschreiben Kriterien für die Bewertung von CSS im Bezug auf das soeben definierte Ideal.

#### Ästhetik

Boswell und Foucher, Informatiker mit langjähriger Erfahrung in der Entwicklung von Software-Großprojekten, haben Richtlinien für lesbaren Code erarbeitet.

Bei visuellen Aspekten des Quellcodes stellt Konsistenz die Basis für Lesbarkeit her. Die unter Lesbarkeit fallenden Kriterien sind die logische Reihenfolge von Bestandteilen, sowie die Gliederung des Codes in Paragraphen und Spalten zur Darstellung von Hierarchien. Das erlaubt die Struktur von Quellcode in seiner Gestalt wider zu spiegeln.

#### Dokumentation und Semantik

Lesbarkeit beschleunigt die Verständnisbildung für in Software beschriebene Systeme kritisch und spart Zeit bei kostspieligen Wartungsarbeiten.[[3]](#footnote-3) Nach Boswell und Foucher verbessert selbsterklärender Code und Dokumentation nicht offensichtlicher Aspekte die Lesbarkeit und beschleunigt somit die Verständnisbildung. Durch Konsistenz in Semantik und Benennung wird Vorhersehbarkeit gewährleistet und verringert den Zeitaufwand bei Wartungsarbeiten deutlich. Die folgenden Abschnitte beschreiben Kriterien, deren Erfüllung der Semantik von Quellcode ebenfalls zuträglich sein kann.

#### Transparenz

Für Neath heißt Wartbarkeit gemeinsames Verständnis von Code.[[4]](#footnote-4) Diese Beschreibung überschneidet sich mit den Erkenntnissen von „The Pragmatic Bookshelf“-Gründern und Autoren von unter Anderem dem hier zitierten Werk „The Pragmatic Programmer“ Hunt und Thomas. Zugriffstransparenz sollte nach Hunt und Thomas bei dem Einsatz von Bibliotheken, oder in diesem Kontext, CSS-Präprozessoren beachtet werden.

Von Transparenz ist die Rede, wenn sich das Zugriffsschema für eine Funktion bei Modifikation des Systems, der Funktionsweise, nicht ändert. Im Kontext dieser Arbeit bedeutet das, dass der Präprozessorcode dem von einfachem CSS möglichst nahe kommt. Transparenz verbessert sowohl Lesbarkeit als auch Wiederverwendbarkeit von Code.[[5]](#footnote-5)

#### DRY

"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."*[[6]](#footnote-6)*

Das so definierte „DRY“-Prinzip bzw. „Don’t Repeat Yourself“ der Softwareentwicklung, zielt auf die Beseitigung von Redundanz im Quellcode ab.

Dies lässt sich durch die Entkoppelung von zueinander orthogonalen, also logisch unabhängigen Systemen erreichen. Redundanz wird im Idealfall eliminiert, was die Wartbarkeit von Code kritisch verbessert.[[7]](#footnote-7),[[8]](#footnote-8)

Die bis hier beschriebenen Kriterien werden im Folgenden als Basis zur Ermittlung der Defizite von CSS genutzt.

### Analyse von CSS

Um funktionelle Anforderungen für CSS-Präprozessoren zu formulieren werden die herangezogenen Kriterien für guten Code zur Ermittlung von Defizite von einfachem CSS genutzt. Hier ein beispielhafter Codeauszug aus der Arbeit eines Frontend-Entwicklers. Der Auszug ist bereits nach Boswells und Fouchers Richtlinien für lesbaren Code formatiert und durch das von Frontend-Entwickler John W. Long als „Plural Parent Pattern“ bezeichnete Muster zur Modellierung von Eltern-Kind-Beziehungen strukturiert. Hierbei stellt der Selektor des Elternelements .demos die Beziehung zu den enthaltenen .demo Elementen durch Pluralisierung im grammatikalischen Sinne her. .special-demo stellt wiederum eine Unterregel zu .demo dar, „erbt“ also sozusagen ihre Eigenschaften und verändert bzw. erweitert sie in um mindestens ein Attribut.[[9]](#footnote-9)

.demos {

background: #111;

color: #eee;

}

.demo, .special-demo {

background: #eee;

color: #111;

border-radius: 15px;

-moz-border-radius: 15px;

-o-border-radius: 15px;

-webkit-border-radius: 15px;

}

.special-demo {

border: 1px solid #111;

}

Abbildung 1 – einfaches CSS

Trotz der Anwendung sinnvoller Maßnahmen weist CSS öfters, auch in diesem Fall, folgende nicht vermeidbare Defizite in Form von Redundanz auf:

Die Attribut-Wert-Paare einer Regel sind im Kontrast zum Selektor durch Einrückung an der zweiten Spalte ausgerichtet. So formatierte Codeblöcke werden von geschweiften Klammern umschlossen. Klammern und Einrückung enthalten dieselbe, duplizierte Information: Die Gruppierung eines von Attribut-Wert-Paaren einer Regel. Die Redundanz zwischen Doppelpunkten bzw. Semikola und dem der Leerzeichen zur Trennung der Werte bzw. Zeilenumbrüchen zwischen Attributen verhalten sich Analog dazu.

Der zweite Selektor .demo weist auf Abstammung des entsprechenden HTML-Elements vom dem des ersten Selektors .demos hin. Hierbei trägt der zweite Selektor redundante Information in Form des multiplizierten Namens des Ersten in sich.[[10]](#footnote-10)

Die im Vorfeld herausgedeutete Struktur, welche Vererbung von .demo nach .special-demo nachbildet ist bereits bei .demo erkennbar. Das Ziel der Vererbung ist hier also nicht identisch mit dem Kontext der Definition. Die Lesbarkeit des Codes kann aufgrund der Trennung von Ursache und Wirkung beeinträchtigt werden.

Farben werden nicht stellvertretend, sondern wortwörtlich definiert, somit wird eine eindeutige Farbe mehrfach und effektiv definiert.

Es fällt auch die Vermengung zweier zueinander orthogonaler Problemlösungen auf: Die Browser-Präfixe zur Gewährleistung der Browserkompatibilität werden zusammen mit der Zuweisung der Werte des border-radius Attributs eingesetzt.

Diese Beobachtungen widersprechen Hunts und Thomas’ DRY Prinzip und beeinträchtigen nicht nur die Lesbarkeit des Codes, sondern auch die Wartbarkeit. Die gezwungene Duplikation,[[11]](#footnote-11) manuelle Ausarbeitung von Mustern und Vermengung zueinander orthogonaler Systeme stellt sowohl einen vermehrten Zeitaufwand als auch eine Fehlerquelle sowohl bei Implementierung und Wartung von Code dar.

1996 wurde CSS, im Sinne der Trennung von Darstellung und Inhalt, das heißt der Extraktion von Inline-Styles und die Darstellung beschreibende Elemente wie <blink>, vom W3C als Empfehlung deklariert.[[12]](#footnote-12) Obwohl die CSS-Spezifikation aus der „Separation of Concerns“-Philosophie heraus entwickelt wurde, bietet sie bis dato praktisch keine Möglichkeiten zur Abstraktion. Diese werden im Folgenden als Anforderungen Formuliert.

### Funktionelle Anforderungen an CSS Präprozessoren

LESS, SASS und Stylus verfügen über eine Vielzahl von Funktionen, im Rahmen dieser Arbeit können jedoch nicht alle untersucht werden. Wir beschränken uns auf diese, die den an CSS festgestellten Defizite am effektivsten entgegen wirken. Trotz der bis jetzt sehr theoretischen Abhandlungen muss das Ziel des tatsächlichen Einsatzes verfolgt werden. Als erstes im Sinne der Integrationsfähigkeit der Präprozessoren gestellte Anforderungen.

#### Verfügbarkeit einer grafischen Oberfläche

Die erste Bedingung für den Einsatz von CSS-PPs ist das Vorhandensein einer grafischen Oberfläche. Frontend Entwickler ohne Erfahrung mit der Kommandozeile, über die Präprozessoren normalerweise bedient werden können, erfahren somit einen erleichterten Einstieg in die Arbeit mit dem Präprozessor der Wahl. So wird Flexibilität bei der Zuweisung von Mitwirkenden zu Projekten gewährleistet und dem Transfer letzterer effizient gehalten.

#### Geschwindingkeit

Die Dauer der Verarbeitung von Präprozessorcode zu CSS ist laut Frontendentwicklerin Jo Liss entscheidend für die Ergonomie. Die Umwandlung dauert bei kleineren Stylesheets so lang, dass der Unterschied zu CSS praktisch nicht wahrgenommen wird. Bei Umfangreicheren Softwarelösungen soll die Umwandlungsdauer möglichst gering bleiben, um die Arbeit mit Präprozessoren im Vergleich zur Entwicklung mit einfachem CSS nicht unnötig mit langen Reaktionszeiten zu erschweren.[[13]](#footnote-13)

#### Syntaktische Flexibilität - Übermenge

„Superset“ (engl. Übermenge) beschreibt das Verhältnis zweier Mengen zueinander. Die Syntax des idealen Präprozessors soll eine Übermenge von einfachem CSS darstellen. Letzteres soll nach der Verarbeitung durch einen CSS-Präprozessor, wenn nicht anders bezweckt, funktionell unverändert ausgegeben werden.

Diese Art der Transparenz gewährleistet die einfache Wiederverwendung von bestehendem CSS und eliminiert die Hürde, eine neue Syntax zur ersten Verwendung von Präprozessoren lernen zu müssen. Das erlaubt, die Einbindung der Präprozessoren in den Workflow inkrementell durchzuführen.

#### Syntaktische Flexibilität – Minimalismus

Die Untersuchung von einfachem, möglichst lesbar formatiertem CSS hat Redundanz zwischen Weissraum und Zeichen zur Markierung logischer Einheiten ergeben.

Ein idealer Präprozessor soll im Sinne von DRY die Möglichkeit geben, diese Form der Redundanz zu reduzieren, wenn nicht sogar ganz zu beseitigen.

#### Variablen

Der CSS3-Standard sieht die Verwendung von Variablen nicht vor, im Kontext der Entwicklung mit PPs stellen sie vielmehr ein Werkzeug dar, um Werte zentral zu definieren und wieder aufzurufen.

Die Verwendung von aussagekräftig benannten Variablen resultiert in wart- und lesbarerem Code. Tatsächliche Farbwerte wie #34DDDD bspw. beinhalten im Vergleich zu sinnvoll benannten Variablen wie Turquoise meist keinerlei Aussagekraft und müssen einzeln geändert werden.

#### Mixins

Mixins erlauben die Isolation sowie Kapselung von Eigenschaften zu flexiblen, wiederverwendbaren Bündeln und stellen wie bspw. in Python und Ruby ein Mittel zur Vererbung von Teilen verschiedener Klassen dar. Die Extraktion nicht verwandter Unterprobleme vergrößert die Orthogonalität des Codes.[[14]](#footnote-14) Der Nutzen von Mixins ergibt sich in Anbetracht der Streuung der Probleme, die bei der Entwicklung von CSS berücksichtigt werden müssen.

Die Modularisierung von Code als auch die Integration von einheitlichen, gut wartbaren und trotzdem flexiblen Rückfalllösungen werden durch Mixins deutlich vereinfacht. Ein gutes Beispiel hierfür ist das Versehen eines experimentellen CSS3-Attributs mit den entsprechenden Browser-Präfixen wie –moz. Das untergeordnete Problem der Browserkompatibilität wird extrahiert und mit dem Aufruf des entsprechenden Mixins behandelt. So verringeren Mixins Redundanz im Quellcode und erlauben es, die Verwendung eines externen, passiven Codegenerators zu vermeiden.

Die Lösung der Browser-Präfix-Problematik dient lediglich zur Demonstration. Eine potenziell bessere Lösung stellt die Verwendung eines weiteren, spezialisierten Präprozessors wie Autoprefixer dar. Letzterer wendet automatisch Präfixe unter Berücksichtigung der definierten Zielbrowser und deren Unterstützung für entsprechende CSS-Eigenschaften an.

#### Nachvollziehbarere Vererbung

Im Rahmen der Untersuchung von einfachem CSS wurde eine Struktur beobachtet, die die Vererbung der Eigenschaften einer Regel erlaubt. Vererbung soll jedoch im Rahmen der erbenden Regel definiert werden können, sodass Ursache und Wirkung beieinander liegen.

#### Selektorverkettung

Um Neaths Erläuterung aufzugreifen: CSS funktioniert als eine auf Spezifität basierende Kaskade, steht also konzeptionell in keinem Bezug zur Baumstruktur von HTML. Das Kaskadenprinzip beschreibt die Anwendung der Regeln auf die Knoten, bzw. Verzweigungen des Document Object Models (DOM).

Trotz dieser Verschiedenheit sind im CSS-Quellcode oft Einflüsse von HTML zu erkennen. Selektoren tragen oft Information zu ihrem Kontext in sich. Augenscheinlich wird dies anhand der Tatsache, dass die Konnektoren wie ~,> und + das Verhalten der Kaskade beschreiben. CSS bietet keine Möglichkeit, die sich daraus ergebenden Muster effizient und redundanzfrei zu beschreiben.

Selektorverkettung erlaubt zum einen, eine HTML ähnlichere und so einfacher zu kontextualisierende visuelle Struktur zu entwickeln, und zum anderen die Reduktion redundanter Information aufgrund des Bezugs zu so erweiterten Selektoren.

Ein Problem der Verkettung von CSS-Selektoren ist die sich somit ergebende Spezifität bzw. „Priorität“ dieser Regeln. Eine Lösung hierfür ist, Selektoren nicht logisch sondern lexikalisch miteinander zu verknüpfen, was dieselben Vorteile bringt, dabei die Priorität der Regel niedrig hält, Wiederverwendbarkeit sicherstellt und auch die Geschwindigkeit des Stylesheets erhöht.[[15]](#footnote-15) Die lexikalische Verkettung erfordert, dass die entsprechenden Klassen bzw. IDs in HTML zugewiesen werden.

# Hauptteil

Bis zu diesem Punkt lag der Fokus dieser Arbeit auf allgemeinen Kriterien für objektiv erstrebenswerten Quellcode, im Folgenden liegt der Fokus auf den Namensgebenden Technologien, CSS-Präprozessoren und deren Nutzen bzw. Defizite.

## Präprozessoren

Präprozessoren dienen zur Verarbeitung von Code, das Resultat wird als Eingabe für ein weiteres System genutzt. In diesem ist Fall das die Layout-Engine von Webbrowsern.

„Active code generators are used each time their results are required. The result is a throw-away—it can always be reproduced by the code generator.“*[[16]](#footnote-16)*

CSS-Präprozessoren können, im Gegensatz zu passiven Codegeneratoren, deren Ausgabe als Basis für weitere Entwicklungen verwendet wird, auf sich ändernde Anforderungen reagieren. Sie sind also per Definition ein effektives Werkzeug zur Erleichterung von Wartungsarbeiten, weil sie die Verknüpfung zwischen Eingabe und Ausgabe erhalten. CSS-Präprozessoren können den Entwicklungsprozess deutlich erleichtern, nehmen dem Entwickler jedoch nicht die tatsächliche Arbeit ab und erweitern den CSS-Standard in keiner Weise.

CSS-Präprozessoren sind eine keine allzu junge Technologie mehr. Bis Dato gibt es jedoch kein Verzeichnis für verschiedene Präprozessoren, ein Großteil von ihnen ist dabei mithilfe der Plattform „GitHub“ entwickelt. Die vier am häufigsten erwähnten und von anderen Entwicklern, durch „Github-Starring“ wertgeschätzten Projekte sind LESS, SASS und Stylus. Myth erscheint als weitere Option, ist jedoch zeitlich deutlich im Rückstand und wird von CSS-Tricks Autor und Frontend-Entwickler Chris Coyer sowie Rodney Rehm, Informatiker bei Telekom, in seinem Grundkonzept kritisiert.

### SASS

Die Erstveröffentlichung des von Natalie Weizenbaum auf Basis von Ruby entwickelten CSS-Präprozessors SASS fand im Jahr 2007 statt.

SASS unterstützt zwei Syntaxen: Erstens die ältere SASS Syntax, die aufgrund ihrer Orientierung an der Ruby-Syntax mit der CSS Syntax inkompatibel ist. Zweitens, seit Version drei, die SCSS Syntax, welche ein Superset von CSS darstellt. Im Kontext dieser Arbeit, wird die SASS-Syntax aufgrund ihrer Inkompatibilität mit einfachem CSS außen vor gelassen, mit SASS ist im folgenden also immer SASS in der SCSS Syntax gemeint.[[17]](#footnote-17)

### LESS

Im Jahr 2009 wurde LESS von den Entwicklern [Alexis Sellier](http://en.wikipedia.org/w/index.php?title=Alexis_Sellier&action=edit&redlink=1) und [Dmitry Fadeyev](http://en.wikipedia.org/w/index.php?title=Dmitry_Fadeyev&action=edit&redlink=1) veröffentlicht. Der bis dato populärste Präprozessor LESS basiert auf JavaScript und ist bspw. über den „Node Package Manager“ (NPM) zu beziehen.

### Stylus

Der jüngste der drei Präprozessoren ist Stylus, von Web-Dienstleister Learnboost entwickelt und 2011 unter dem vielversprechenden Leitsatz „Expressive, dynamic, robust CSS”[[18]](#footnote-18) ebenfalls als Node-Modul veröffentlicht.[[19]](#footnote-19)

## Prüfung der funktionellen Anforderungen

Im folgenden werden die Kandidaten LESS, SASS und Stylus auf Erfüllung der funktionellen Anforderungen geprüft.

### Grafische Oberfläche

Für den entsprechenden Präprozessor soll auf Windows 7 bzw. MacOSX 10.7 und neuer Software zur Anwendung mit einer grafischen Oberfläche zur Verfügung stehen.

Es gibt mehrere Produkte von Drittherstellern, die die Arbeit mit CSS-Präprozessoren durch eine grafische Oberfläche erleichtern können.

Entwickler Subash Pathak bietet die Software „Prepros“ zur einfachen Verwendung von, aber nicht beschränkt auf LESS, SASS und Stylus an. Prepros erlaubt die automatische Umwandlung verschiedener Template- und Präprozessorsprachen für Windows.[[20]](#footnote-20)

Für Mac OSX stellt „CodeKit“ von Incident57 dieselben und noch deutlich mehr Werkzeuge zur Code- und Workflowoptimierung bereit.[[21]](#footnote-21)

Auch wenn Linux-Nutzer üblicherweise solide Kenntnisse über das Terminal besitzen, bietet das von unter Anderen Ethan Lai entwickelte Tool „Koala“ die Möglichkeit, SASS und LESS, aber nicht Stylus mithilfe einer grafischen Oberfläche zu verarbeiten.

„Codepen“ erlaubt die Umwandlung aller drei Präprozessoren und ist neben „LESS2CSS“, „SASSmeister“ und „Stylus Online“ eine Plattform für Experimente mit Präprozessoren.

Für „Internet Information Services“ (IIS) Server bietet DotLESS eine Integrationslöung für LESS. Über die Paketverwaltung von Ruby und NodeJS sind Gems bzw. Module für alle drei Präprozessoren verfügbar.

### Geschwindigkeit

Im folgenden ein Auszug der Ergebnisse des von Jo Liss durchgeführten Präprozessor Benchmarks.

|  |  |
| --- | --- |
| Compiler | Compiledauer in Sekunden |
| LESS | 0.5 |
| SASS | 4.9 |
| SASS mit warmem Cache | 2.5 |
| libSASS | 0.2s |
| Stylus | 1.7 |

Tabelle 1 – Geschwindigkeit: Benchmark Ergebnisse

Verarbeitet wird eine einfache CSS-Datei einer Größe von 200KB, was den Vergleich zu dem CSS einer großen Webanwendung erlaubt. Die C# Implementierung von SASS, „libSASS“ arbeitet hier am schnellsten, ist jedoch nicht auf dem aktuellen Entwicklungsstand von SASS. Später folgt Stylus und zuletzt SASS, mit warmem Cache, das heißt ab dem zweiten Compilevorgang derselben Datei.[[22]](#footnote-22)

### Tests

Die Tests prüfen LESS v1.7.4, SASS v3.4.1 (SCSS-Syntax) und Stylus v0.47.3 in Hinsicht auf Syntax und Interpretation von Kernfeatures. Die Tests sind unter <https://bitbucket.org/JanWirth/preprocessor-tests>öffentlich zugänglich.

Der in den Tests verwendete Quellcode ist im Sinne der Lesbarkeit primär auf Ähnlichkeit zur regulären CSS-Syntax, sekundär auf Aussagekraft bzw. Semantik, und tertiär auf minimalistische Anwendung ausgelegt. Die Validität der resultierenden Stylesheets wird außer Acht gelassen, da mit dem Fokus auf Lesbarkeit stellenweise invalide Attribute eingesetzt werden.

Funktionell identische Resultate werden, mit Ausnahme der Ergebnisse des ersten Tests zu einem Codeblock zusammengefasst.

Die Bewertungskriterien sind zum einen die von Boswell und Foucher entwickelten Regeln für lesbaren Code, das DRY-Prinzip und transparente Integration.

#### Superset Syntax

Die Umwandlung von validem CSS3 durch den Präprozessor soll in funktional identischem CSS resultieren.

##### Quellcode

@media screen and (min-width: 1200px){

.selector{

color: #112233;

margin: 1px 5em 0 0;

-webkit-transform: translateX(30px)

}

}

Abbildung 2 – Superset Syntax: Quellcode – LESS, SASS, Stylus

##### Resultat

@media screen and (min-width: 1200px){

.selector{

color: #112233;

margin: 1px 5em 0 0;

-webkit-transform: translateX(30px)

}

}

Abbildung 3 – Superset Syntax: Resultat – LESS, Stylus

@media screen and (min-width: 1200px){

.selector{

color: #112233;

margin: 1px 5em 0 0;

-webkit-transform: translateX(30px)}}

Abbildung 4 – Superset Syntax: Resultat – SASS

##### Analyse

Das Resultat ist funktional identisch mit dem Quellcode.

Es fallen zwei Details auf: SASS formatiert die ausgegebenen Regeln kompakter durch Verbergen eines Zeilenumbruchs von schließenden Semikola und Stylus komprimiert HEX-Farbcodes von sechs auf drei stellen, wenn möglich.

Bewertung

Ausdrücke der CSS3 Syntax lassen sich auch in LESS, SCSS und Stylus beschreiben.

#### Minimalistische Syntax

Der Präprozessor muss durch Weißraum strukturierten Quellcode interpretieren und korrekt ausgeben können, ohne, dass Einstellungen verändert werden müssen.

##### Quellcode

.selector

color: #112233;

margin: 1px 5em 0 0;

-webkit-transform: translateX(30px)

Abbildung 5 – minimalistische Syntax: Quellcode – LESS, SASS, Stylus

##### Resultat

.selector {

color: #123;

margin: 1px 5em 0 0;

-webkit-transform: translateX(30px);

}

Abbildung 6 – minimalistische Syntax: Ergebnis - Stylus

Die Verarbeitung derartig strukturierten Quellcodes resultiert in folgenden Fehlern bei SASS Line 3: Invalid CSS after " font-size 0.3em": expected "{", was "" und bei LESS Unrecognised input in file [...]/minimalism.less line no. 3.

##### Analyse

Trotz der Tatsache, dass SASS auf Ruby basiert, welche eine der populärsten Sprachen mit der Einrückungs-Syntax darstellt, ist dieses Feature in SCSS ebenso wenig wie in LESS gegeben, beide erlauben keinerlei Reduzierung der Redundanten Zeichen im Quellcode. SASS bietet zwar die zu SCSS alternative SASS Syntax, welche jedoch kein CSS-Superset darstellt, weil sie die geschweiften Klammern und Semikola komplett aus der Syntax verbannt.

Stylus hingegen erlaubt es, CSS ohne durch lesbare Formatierung redundante Zeichen zu beschreiben.

#### Variablen

Ein Symbol, dem ein Wert zugeordnet wird, soll bei einem Aufruf in der Ausgabe eben des Werts resultieren.

##### Quellcode

@var1: default;

@var2: default;

.selector{

var1: @var1;

}

#block-one {

#before-var-reassignment-inside {

var1: @var1;

}

@var1: changed-by-block-one;

}

#block-two {

@var2: changed-by-block-two;

#after-var-reassignment-inside {

var2: @var2;

}

}

.completely-different-block {

var2: @var2;

}

Abbildung 7 – Variablen: Quellcode - LESS

$var1: default;

$var2: default;

.selector{

var1: $var1;

}

#block-one {

#before-var-reassignment-inside {

var1: $var1;

}

$var1: changed-by-block-one;

}

#block-two {

$var2: changed-by-block-two;

#after-var-reassignment-inside {

var2: $var2;

}

}

.completely-different-block {

var2: $var2;

}

Abbildung 8 – Variablen: Quellcode – SASS

var1= default;

$var2= default;

.selector{

var1: var1;

}

#block-one {

#before-var-reassignment-inside {

var1: var1;

}

var1= changed-by-block-one;

}

#block-two {

$var2= changed-by-block-two;

#after-var-reassignment-inside {

var2: $var2;

}

}

.completely-different-block {

var2: $var2;

}

Abbildung 9 – Variablen: Quellcode - Stylus

Abb. 9 zeigt, dass Präfixe von Variablen bei Stylus im Gegensatz zu LESS und SASS optional sind.

.selector {

var1: default;

}

#block-one #before-var-reassignment-inside {

var1: changed-by-block-one;

}

#block-two #after-var-reassignment-inside {

var2: changed-by-block-two;

}

.completely-different-block {

var2: default;

}

Abbildung 10 – Variablen: Resultat – LESS

.selector {

var1: default;

}

#block-one #before-var-reassignment-inside {

var1: default;

}

#block-two #after-var-reassignment-inside {

var2: changed-by-block-two;

}

.completely-different-block {

var2: default;

}

Abbildung 11 – Variablen Resultat - SASS, Stylus

##### Analyse

Alle drei Präprozessoren unterstützen die Verwendung von Variablen. Allerdings unterscheidet sich hier sowohl die Syntax, als auch die Interpretation bei einer Zuweisung.

Die Sichtbarkeit, also die „Reichweite“ einer Variable ist bei SASS immer lokal, wenn nicht mit !global ausgezeichnet. Bei der Neuzuweisung innerhalb eines Blocks überschattet Stylus hingegen den Wert für darauffolgende Aufrufe und LESS für alle Aufrufe innerhalb des Blocks sowie für tiefere Ebenen.

##### Bewertung

SASS unterstütze lokale Variablen bis v3.4.0 nicht, was eine von Shaw und Wulf, beide renommiert für ihre Arbeit im Feld des Software-Engineerings, wörtlich übersetzt, eine als „schädlich“ bezeichnete Verhaltensweise beseitigt.[[23]](#footnote-23) Variablen sind seitdem, wenn nicht mit !global markiert, immer lokal.

Variablen überschatten auch bei LESS und Stylus globale Variablen im Kontext eines Blocks, statt sie zu überschreiben, allerdings geschieht das bei LESS rückwirkend für „frühere“ Aufrufe im selben Block. Dies widerspricht nicht nur der Logik der Leserichtung von Quellcode, sondern lässt auch an der logischen Integrität von LESS zweifeln, denn eine Neuzuweisung im globalen Kontext wirkt sich nur auf darauffolgende Aufrufe aus.

Stylus erlaubt die Verwendung von Variablen ohne Präfix. Die Implementierung von Variablen ist hier also als transparent zu bezeichnen. Aufgrund dessen wird Quellcode ermöglicht, der CSS ähnlicher ist, als die Quellcodes der anderen beiden Präprozessoren. Die Verwenung von Variablen bei Stylus lässt erfolgt Transparent. Diese Standardkonformität erhöht die Lesbarkeit des Codes.[[24]](#footnote-24)

#### Mixins

Die per Mixin definierten Attribut-Wert-Paare sollen beim Aufruf an der entsprechenden Position nach der Interpolation mit dem Parametern ausgegeben werden.

##### Quellcode

.border-radius(@radius) {

-webkit-border-radius: @radius;

-moz-border-radius: @radius;

-ms-border-radius: @radius;

border-radius: @radius;

}

.box { .border-radius(10px); }

Abbildung 12 – Mixins: Quellcode - LESS

@mixin border-radius($radius) {

-webkit-border-radius: $radius;

-moz-border-radius: $radius;

-ms-border-radius: $radius;

border-radius: $radius;

}

.box { @include border-radius(10px); }

Abbildung 13 – Mixins: Quellcode – SASS

border-radius(radius) {

-webkit-border-radius: radius;

-moz-border-radius: radius;

-ms-border-radius: radius;

border-radius: radius;

}

.box { border-radius: 10px; }

Abbildung 14 – Mixins: Quellcode - Stylus

##### Resultat

.box {

-webkit-border-radius: 10px;

-moz-border-radius: 10px;

-ms-border-radius: 10px;

border-radius: 10px;

}

Abbildung 15 – Mixins: Resultat – LESS, SASS, Stylus

##### Analyse

Alle drei Präprozessoren erlauben die Einbindung von Codeblöcken unter der Berücksichtigung von Parametern. Die Definition bei SASS und Stylus geschieht über das @mixin Schlüsselwort, gefolgt von einem, die optionalen Parameter umschließendes Klammerpaar.

LESS kehrt das Mixin-Prinzip um und betrachtet ganze Regeln als wiederverwendbare Block-Mixins. Die Ausgabe einer Regel kann hier, wie bei SASS und Stylus durch Anhängen eines Klammerpaares an den Selektor verhindert werden. Die Anwendung als Mixin erfolgt identisch zu der Verwendung von Mixins bei SASS, nämlich durch das @include Schlüsselwort.

In Bezug auf die Anwendung erlaubt Stylus hingegen, das hier beispielhaft eingesetzte Attribut border-radius zu überschreiben und ist somit wieder als transparent zu bezeichnen.

##### Bewertung

Die Untersuchungen des Mixin-Features der drei Präprozessoren haben konzeptionelle und funktonielle sowie syntaktische Unterschiede ergeben, die sich verschieden auf den Quellcode auswirken beeinträchtigen können.

LESS verwischt die Grenzen zwischen Mixins und Regeln – das ist problematisch, da das Anwenden von einfachen Regeln als Mixin nichts weiter als Duplikation bewirkt. Das sinnvollere Vorgehen ist hier die Verwendung des im nächsten Test untersuchten @extend Features. Mixins bei LESS sind vielseitig, doch diese Vielseitigkeit stellt effektiv nur eine Gefährdung für DRY CSS als Endprodukt dar.

Stylus und SASS bieten das Mixin-Feature mit identischer Funktionalität. SASS setzt jedoch voraus, dass Mixins bei Definition und Verwendung klar gekennzeichnet sind. Die Tatsache, dass Mixins, an dem Punkt, an dem sie aufgerufen werden, eine erweiterte Funktion tragen ist jedoch weder bei der Erstellung noch Wartung des entsprechenden Codeabschnitts für den Leser wirklich relevant. Das im Kontext einer Regel bearbeitete Problem ist die Definition von Styles, nicht die technische Umsetzung unter Berücksichtigung der Browser-Kompatibilität.

Stylus unterscheidet sich hier, wie auch bei Variablen, durch Transparenz. Ohne die Modifikation der Syntax wird hier klare und einfache Entkoppelung des Subproblems ermöglicht. Diese nahtlose Integration sichert Wiederverwendbarkeit und Lesbarkeit.[[25]](#footnote-25) Mixins sollten in Stylus nur dann transparent eingesetzt werden, wenn sie Parameter erhalten. Bei transparenten Mixins ohne Parameter empfiehlt der Autor das Anfügen eines Klammerpaares als wichtigen Hinweis auf die erweiterte Funktion, bzw. als Unterscheidungsmerkmal zu unvollständigen CSS-Definitionen.

#### Nachvollziehbarere Vererbung

Ein idealer CSS-Präprozessor bietet die Möglichkeit, Vererbung im Rahmen der erbenden Regel zu definieren.

##### Quellcode

.message {

border: 1px solid #ccc;

padding: 10px;

color: #333;

}

.success {

&:extend(.message);

border-color: green;

}

.error {

&:extend(.message);

border-color: red;

}

.warning {

&:extend(.message);

border-color: yellow;

}

Abbildung 16 – Vererbung: Quellcode - LESS

.message {

border: 1px solid #ccc;

padding: 10px;

color: #333;

}

.success {

@extend .message;

border-color: green;

}

.error {

@extend .message;

border-color: red;

}

.warning {

@extend .message;

border-color: yellow;

}

Abbildung 17 – Vererbung: Quellcode – SASS, Stylus

##### Resultat

.message,

.success,

.error,

.warning {

border: 1px solid #ccc;

padding: 10px;

color: #333;

}

.success {

border-color: #008000;

}

.error {

border-color: #f00;

}

.warning {

border-color: #ff0;

}

##### Analyse

Alle drei Präprozessoren erlauben es, Vererbung im Kontext des erbenden Elements zu definieren. Die Syntax von LESS sieht die Verwendung des &:extend(.descendant) Pseudoselektors innerhalb eines Blocks vor, um Vererbung darzustellten. Bei SASS und Stylus wird Vererbung mit @extend .descendant dargestellt.

##### Bewertung

Die nötige Verwendung eines Pseudo-Selektors bei LESS bietet keine sinnvolle Information, sondern deutet auf das im Vergleich zur einfacheren Schreibweise bei SASS und Stylus. Alle drei Präprozessoren erfüllen die Anforderung.

#### Selektorverkettung

Die Möglichkeit zur redundanzfreien logischen und lexikalischen Erweiterung eines Selektors muss gegeben sein. Die Zuweisung von Attribut-Wert Paaren zu dem sich so ergebenden Selektors ist zur Darstellung einer kompletten Regel unabdingbar.

##### Quellcode

.selector {

color: black;

&-submodule {

background: black;

color: white;

}

.parent & {

opacity: 0.8;

}

}

Abbildung 18 – Selektorverkettung: Quellcode – LESS, SASS, Stylus

##### Resultat

.selector {

color: #000;

}

.selector-submodule {

background: #000;

color: #fff;

}

.parent .selector {

opacity: 0.8;

}

Abbildung 19 – Selektorverkettung: Resultat – LESS, SASS, Stylus

##### Analyse

Abgesehen von der Formatierung der Ausgabe bieten alle drei Präprozessoren praktisch identische Funktionalität. Das &-Zeichen dient als Referenz zum Elternselektors und kann zur einfacheren Bildung verschiedener Muster dienen.

Des weiteren erlaubt SASS[[26]](#footnote-26) die Zuweisung des Elternselektors zu einer Variable zur weiteren Evaluation mithilfe von SASSScript,

Bewertung

Die effiziente Verkettung von Selektoren bietet ein Werkzeug zur Reduktion redundanter Information im Quellcode, was Lesbarkeit und Wartbarkeit verbessert. Der Einsatz von Selektorverkettung in dieser Form gewährleistet, Tragbarkeit zwischen den CSS-Präprozessoren, da sie alle die identische Syntax verwenden, die Abwärtskompatibilität mit CSS wird damit jedoch gebrochen.

## Entscheidungsfindung

Wie bereits zu Beginn der Formulierung nicht-funktioneller Kriterien erwähnt, verfügen alle drei Präprozessoren über viele, in dieser Arbeit nicht berücksichtigte Funktionalitäten. Bei der Untersuchung des Umgangs mit Variablen wurde am Beispiel von SASS verdeutlicht, dass sich die Technologien unter ständiger Weiterentwicklung befinden. Diese beiden Aspekte gilt es zu berücksichtigen, da die Testergebnisse ihre Aussagekraft in Zukunft verlieren werden.

Nach der Untersuchung der funktionellen Anforderungen folgt eine tabellarische Gegenüberstellung der Ergebnisse mit dem Ziel, aus dem Vergleich Schlüsse zur Empfehlung bei der Wahl des optimalen CSS-Präprozessors auszusprechen.

Die Gegebene Notenskala orientiert sich am System der Schulnoten, stellt jedoch lediglich ein Vergleichsmaß dar. Bei kompletter Erfüllung der Anforderungen wird die Note „1“ Vergeben, bei Mängeln, welche in den Bewertungsabschnitten beschrieben sind, die Noten „2“ bis „4“. Die Noten „5“ und „6“ werden bei Nichterfüllung mit möglichen Alternativen bzw. gänzlicher Nichterfüllung vergeben.

|  |  |  |  |
| --- | --- | --- | --- |
| Anforderung | LESS | SASS | Stylus |
| Grafische Oberfläche | 1 | 1 | 2 |
| Geschwindigkeit | 1 | 3 | 2 |
| Superset Syntax | 1 | 1 | 1 |
| Minimalistische Syntax | 6 | 5 | 1 |
| Variablen | 3 | 1 | 1 |
| Mixins | 3 | 2 | 1 |
| Einfacherere Vererbung | 2 | 1 | 1 |
| Selektorverkettung | 1 | 1 | 1 |
| Durchschnitt | 2,25 | 1,875 | 1,25 |

Tabelle 2 - Wertung

Beim Vergleich der Ergebnisse zeichnet sich Stylus positiv ab. Dies begründet sich durch die Erfüllung der vom DRY Prinizip gestellten Anforderungen. Stylus erlaubt die transparente, also nahtlose Integration einer Abstraktionsebene in den Entwicklungsprozess von CSS und implementiert die in diesem Kontext wichtigsten Funktionen logisch und effizient, ohne dabei die CSS Syntax in einem mehr als nötigen Umfang zu modifizieren. Trotz der Transparenz die sich konsequent durch diesen Präprozessor zieht, erlaubt er durch die minimale Syntax CSS im Rahmen der Möglichkeiten vieler Defizite zu entledigen.

Bei der Wahl des passenden Prärpozessors sind nicht nur die Ergebnisse des Vergleichs alleinig als Basis heranzuziehen. Hinzu kommen bspw. das Ökosystem, die Umgebung des Projekts, oder, bei Einsatz in kleinem Rahmen auch *persönliche Vorlieben.*

# Schluss

## Fazit

CSS-Präprozessoren lassen sich, so wie viele Informationstechnologien als Zweischneidiges Schwert bezeichnen, das man aufgrund seiner Natur zumindest nicht falsch herum halten kann.

Im Rahmen dieser Arbeit liegen die Ergebnisse zugunsten von Stylus. Doch unabhängig davon, auf welchen Präprozessor die Wahl fällt, stellt die Anwendung zumindest einer dieser Technologien aufgrund der Deckungsgleichheit ihrer Funktionen einen Effizienzgewinn für den Entwicklungsprozess von CSS dar.

## Ausblick

„CSS preprocessors are here to stay, if you work in web development and haven’t touched them yet I highly recommend you get into them.“[[27]](#footnote-27)

Durch die Natur der CSS-Präprozessoren, die außerhalb der Zielplattform Webbrowser liegt, können sie sich deutlich schneller weiterentwickeln, als die tatsächliche Implementierung diverser Webstandards. Sie bieten nicht nur einen effizienteren Umgang mit CSS, sondern auch eine Plattform zur Entwicklung von Lösungen, die CSS auf neue Arten und Weise nutzen.

Zum Testen der Präprozessoren bietet Codepen einen einfachen Einstieg und volle Unterstützung der Syntaxen.

Literaturverzeichnis

*In alphabetischer Reihenfolge*

Bochkariov, Bulat. *Why are front end developers so high in demand at startups if front end development is relatively easier than other fields of engineering?* 11. Dezember 2013. http://www.quora.com/Startups/Why-are-front-end-developers-so-high-in-demand-at-startups-if-front-end-development-is-relatively-easier-than-other-fields-of-engineering (Zugriff am 01. August 2014).

Boswell, Dustin, und Trevor Foucher. *The Art of Readable Code.* 1st Edition. Sebastopol : O’Reilly Media, Inc. , 2012.

Coyier, Chris. „Mixins for Rem Font Sizing.“ *CSS-Tricks.* 06. Mai 2013. http://css-tricks.com/snippets/css/less-mixin-for-rem-font-sizing/ (Zugriff am 12. August 2014).

—. *Poll Results: Popularity of CSS Preprocessors.* 11. Juni 2012. http://css-tricks.com/poll-results-popularity-of-css-preprocessors/ (Zugriff am 6. August 2014).

Flanagan, David. *JavaScript: The Definitive Guide.* 6th Edition. Sebastopol: O'Reilly Media, Inc., 2011.

Hunt, Andrew, und David Thomas. *The Pragmatic Programmer - From Journeyman to Master.* 2nd Print. Reading, Massachusetts: Addison-Wesley, 2000.

Imms, Daniel. *CSS Preprocessors Are here to Stay.* 16. Märt 2014. http://www.growingwiththeweb.com/2014/03/css-preprocessors-are-here-to-stay.html (Zugriff am 03. August 2014).

Incident57. *CodeKit.* 30. Juli 2014. https://incident57.com/codekit/ (Zugriff am 25. August 2014).

LearnBoost. *Stylus.* 2011. http://learnboost.github.io/stylus/ (Zugriff am 25. August 2014).

Liss, Jo. *CSS Preprocessor Benchmark.* 27. Januar 2014. (Zugriff am 30. August 2014).

Long, John Wesley. „Modular CSS Naming Conventions.“ *The Sass Way.* 31. August 2013. http://thesassway.com/advanced/modular-css-naming-conventions (Zugriff am 16. August 2014).

Meyer, Eric. *CSS - The Definitive Guide.* Sebastopol, CA: O'Reilly Media Inc., 2006.

N.N. *Sass - Syntactically Awesome Stylesheets.* 13. August 2011. http://sass-lang.com/ (Zugriff am 13. November 2011).

—. „Sass Changelog.“ *Sass.* 22. August 2014. http://sass-lang.com/documentation/file.SASS\_CHANGELOG.html (Zugriff am 25. August 2014).

Neath, Kyle. *Knyle Style Sheets.* 5. Dezember 2011. http://warpspire.com/posts/kss/ (Zugriff am 7. August 2014).

Pathak, Subash. *Prepros.* 15. Juni 2014. http://alphapixels.com/prepros/ (Zugriff am 25. August 2014).

Snook, Jonathan. *Scalable and Modular Architecture for CSS: A Flexible Guide to Developing Sites Small and Large.* 2nd Edition. Snook.ca Web Development, Incorporated, 2012, 2012.

Tjortjis, Christos, Nicolas Gold, Paul Layzell, und Keith Bennett. „From System Comprehension to Program Comprehension.“ *Researchgate.* 2003. https://www.google.de/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CCAQFjAA&url=http%3A%2F%2Fwww.researchgate.net%2Fpublication%2F3974549\_From\_system\_comprehension\_to\_program\_comprehension%2Ffile%2Fe0b4951b6fd2f00ee1.pdf&ei=hIfvU7-SBsTo7AbrhIG4DA&usg=AFQjCNF-oGrqdZsa6fady135d8HuWNIZKw&sig2=gCh9HOv9IKxEV-xQYCm0TQ.

Verschiedene. „[feature request] variable scope definition #473.“ *GitHub.* 14. Dezember 2014. https://github.com/sass/sass/issues/473 (Zugriff am 12. August 2014).

—. *Stylus (Stylesheet-Sprache).* 16. Juli 2014. http://de.wikipedia.org/wiki/Stylus\_(Stylesheet-Sprache) (Zugriff am 6. August 2014).

Wulf, William, und Mary Shaw. „“Global Variable Considered Harmful".“ *ACM SIGPLAN Notices*, 1. Februar 1973: 28–34.

Ehrenwörtliche Erklärung

Name

Vorname

Ich erkläre mit meiner Unterschrift, dass ich die Projektarbeit selbstständig und ohne fremde Hilfe angefertigt habe. Ich habe alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht und mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient.

Die Projektarbeit hat in dieser oder in ähnlicher Form noch keiner anderen öffentlich-rechtlichen oder staatlichen Prüfungsstelle vorgelegen.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ort / Datum Unterschrift

1. (Bochkariov 2013) [↑](#footnote-ref-1)
2. (Neath 2011) [↑](#footnote-ref-2)
3. Vgl. (Tjortjis, et al. 2003) [↑](#footnote-ref-3)
4. Vgl. (Neath 2011) [↑](#footnote-ref-4)
5. Vgl. (Hunt und Thomas 2000) S. 39 [↑](#footnote-ref-5)
6. (Hunt und Thomas 2000) S. 27 [↑](#footnote-ref-6)
7. Vgl. (Boswell und Foucher 2012) [↑](#footnote-ref-7)
8. Vgl. (Hunt und Thomas 2000) [↑](#footnote-ref-8)
9. Vgl. (Long 2013) [↑](#footnote-ref-9)
10. Vgl. (Long 2013) [↑](#footnote-ref-10)
11. Vgl. (Hunt und Thomas 2000) S. [↑](#footnote-ref-11)
12. Vgl. (Meyer 2006) S. 3 [↑](#footnote-ref-12)
13. Vgl. (Liss 2014) [↑](#footnote-ref-13)
14. Vgl. (Boswell und Foucher 2012) [↑](#footnote-ref-14)
15. Vgl. (Snook 2012) S. 54 [↑](#footnote-ref-15)
16. (Hunt und Thomas 2000) S. 103 [↑](#footnote-ref-16)
17. (N.N., Sass Changelog 2014) [↑](#footnote-ref-17)
18. (LearnBoost 2011) [↑](#footnote-ref-18)
19. Vgl. (Verschiedene, Stylus (Stylesheet-Sprache) 2014) [↑](#footnote-ref-19)
20. Vgl. (Pathak 2014) [↑](#footnote-ref-20)
21. Vgl. (Incident57 2014) [↑](#footnote-ref-21)
22. Vgl. (Liss 2014) [↑](#footnote-ref-22)
23. Vgl. (Wulf und Shaw 1973) [↑](#footnote-ref-23)
24. (Boswell und Foucher 2012) [↑](#footnote-ref-24)
25. Vgl. (Boswell und Foucher 2012) S. 110 [↑](#footnote-ref-25)
26. Vgl. (N.N., Sass Changelog 2014) [↑](#footnote-ref-26)
27. (Imms 2014) [↑](#footnote-ref-27)