I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Abkürzungsverzeichnis

API	Application Programming Interface
EPS	Encapsulated PostScript
HTTP	Hypertext Transfer Protocol
IIP	Internet Imaging Protocol
iipsrv	IIPImage Server
JPG / JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
OSI	Open Source Initiative
PDF	Portable Document Format
PNG	Portable Network Graphics
SVN	Apache Subversion
TDD	Test-driven development
TIFF	Tagged Image File Format

1 Einleitung

1.1 dotSource – die ecommerce agentur

Die dotSource GmbH, in welcher der Autor arbeitet, ist im Bereich des E-Commerce tätig. Sie ist spezialisiert auf die Plattformen Intershop – Enfinity und Magento; zwei leistungsfähige Shopsysteme mit vielen Möglichkeiten zur Anpassung und Erweiterung der existierenden Funktionalität. Durch das Beobachten von Trends und Arbeiten mit innovativen Konzepten befasst sich die dotSource aktuell mit dem Thema Mass Customization.

1.2 Wozu dient Mass Customization

Mass Customization ist die Kombination aus Massenproduktion und individuellen Anpassungen. Hierbei wird versucht dem Kunden so viele Gestaltungsmöglichkeiten bei der Individualisierung seines Produktes wie möglich zu geben. Ziel ist es „den individuellen Bedürfnissen der Nachfrager zu genügen.“[1] Ungeachtet dessen ist die Fertigung solcher spezifisch erstellten Produkte keine Einzelfertigung. Dies wird durch die technischen Hilfsmittel der Massenfertigung möglich. Beispielsweise kann der Kunde einer Druckerei eine persönliche Gestaltung für Flyer erstellen und anschließend eine große Menge davon zeitnah erhalten.

Im Zuge der Idee dieser kundenorientierten Produkterstellung entwickelte die dotSource den *Product Designer*. Dieser stellt eine Webapplikation dar, mit welcher Text, Bilder und Farben auf einer Vorlage selbst gestaltet und später über den Shop gekauft werden können. Serverseitig bearbeitet der dsImageServer die Anfragen (sogenannte Requests) nach Bildern. Sie werden verarbeitet und die Ergebnisse in Form von Bildern an den Produktgestalter zurückgegeben.

[1] [Kre05], S. 4

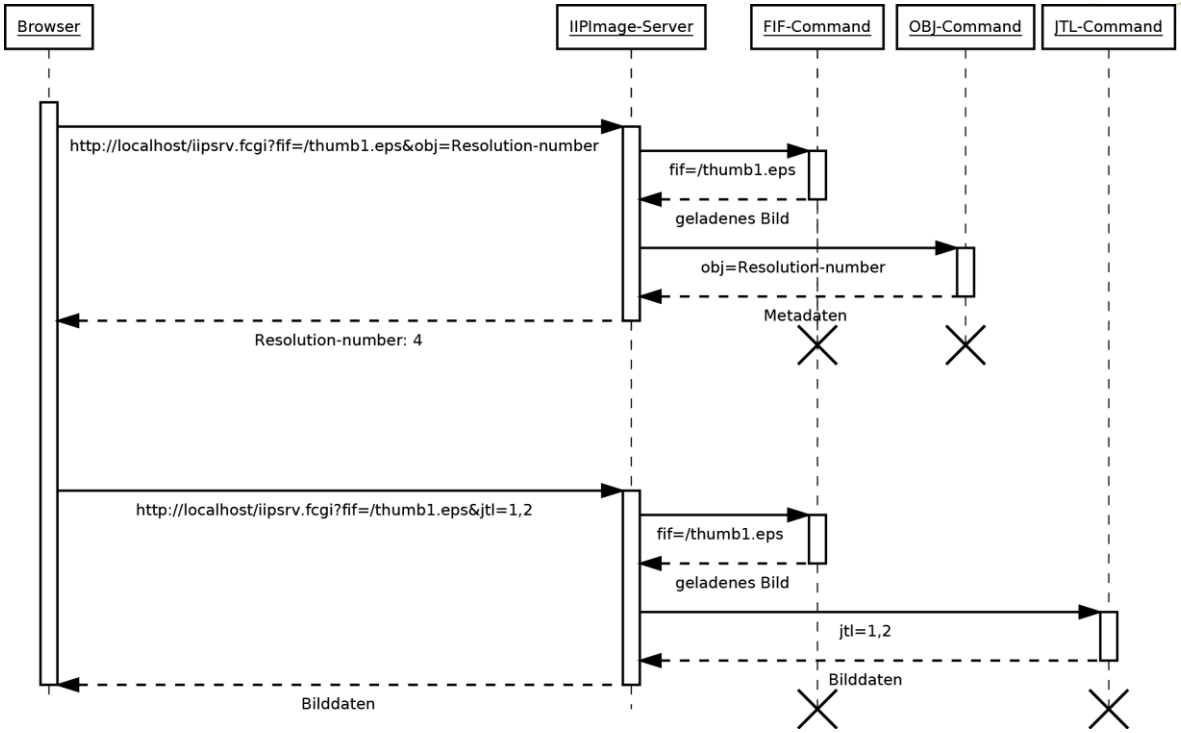
1.3 Was ist der dsImageServer

Der dsImageServer basiert auf der frei verfügbaren Implementierung des Servers IIPImage (iipsrv).[2] Dieser iipsrv stellt eine in der Programmiersprache C++ verfasste Implementierung des Internet Imaging Protocols (IIP) dar.[3] Das Protokoll wiederum definiert verschiedene Operationen. Dazu gehört beispielsweise die Anfrage nach Metadaten des Bildes (OBJ-Befehl) oder nach dessen Kacheln unter Verwendung verschiedener Auflösungsstufen (JTL-Befehl). Der Server implementiert diese Befehle über ein Command-Pattern. Es bedeutet, dass die einzelnen Teilfunktionen innerhalb eines Requests in unterschiedlichen Klassen verarbeitet werden. Der Workflow zweier beispielhafter Anfragen an den Server kann der nachfolgenden Abbildung entnommen werden. Dabei wird zuerst eine Anfrage nach den möglichen Auflösungsstufen eines Bildes gesendet. Anschließend wird eine Anfrage nach einer bestimmten Kachel innerhalb einer der zuvor erfragten Auflösungen verschickt.

[2] [Mac11]

[3] [Pil12]

Abbildung 1: IIPImage Requests



Quelle: eigene Darstellung

Der dsImageServer selbst stellt nun eine Erweiterung der Funktionalitäten des iipsrv dar. Er ergänzt den IIPImage-Server um Funktionalitäten wie:

- Serverseitiges Umfärben von Bildern
- Vektorgrafiken (Encapsulated PostScript Dateien (EPS)) als mögliche Eingabeformate
- Portable Network Graphics (PNG) als neues Bild-Rückgabeformat
- Generierung von Portable Document Format Dateien (PDF) unter Verwendung mehrerer Einzelbilder

Für weiterführende Erklärungen zum IIP-Protokoll und dem IIPImage-Server sei hier auf die Links aus *Anlage A: Hilfreiche Lektüre zur Funktionsweise des dsImageServer* verwiesen. Unter den genannten Verweisen befinden sich die zugehörigen Dokumentationen. Der Funktionsumfang des Servers unter Einbeziehung der hinzu

implementierten Funktionen der dotSource GmbH kann dem Dokument [Mac11] entnommen werden.

1.4 Aufgabenstellung

Das Ziel dieser Arbeit besteht aus drei Teilaufgaben:

1. Bei der Migration wird es darum gehen in den dsImageServer die aktuelle iipsrv Version 0.9.9 zu integrieren. Dabei muss darauf geachtet werden, dass nach Aktualisierung des Servers die ursprünglichen Funktionen weiterhin lauffähig sind. Zur Sicherstellung dessen wird eine maschinelle Testabdeckung des Systems geschaffen.
2. Ein weiterer Teil dieser Arbeit ist die Dokumentationserstellung des dsImageServers. Anschließend wird anhand der erwähnten Abhängigkeiten und Workflows die zeitintensiven Abschnitte des Systems analysiert und auf Optimierungen hin untersucht, bzw. abgeändert.
3. Der iipsrv steht unter einer Open Source Lizenz. Um die Nutzung des dsImageServers rechtlich gültig zu machen muss dieser ebenfalls unter der zugehörigen Lizenz laufen. Nach Absprache der dotSource GmbH und den Entwicklern des iipsrv wurde ein Vertrag geschlossen, welcher die benötigten Rechte an die dotSource definiert. Eine Bedingung des Vertrags ist, dass die Nutzung des iipsrv im Rahmen dotSource eigener Entwicklungen die Offenlegung mindestens eines Teils der Funktionen bedarf. Innerhalb dieser Arbeit wird auf den Open-Source Begriff im Allgemeinen ebenso wie auf bekannte Vertreter im Speziellen eingegangen. Der dsImageServer-Quelltext wird entsprechend den enthaltenen freizugebenden Funktionen angepasst, getestet und dokumentiert.

1.5 Open Source Lizenzmodelle

Open Source Programme bieten dem Nutzer die Möglichkeit Programme frei zu verwenden, sie zu bearbeiten und zu verbreiten. Ebenso besitzen die Lizenzen meist

sogenannte „Copyleft“ Klauseln. Diese sollen sicherstellen, dass Weiterentwicklungen an Open Source Software ebenfalls unter eine solche Lizenz gesetzt wird, damit die freie Nutzung weiterhin ermöglicht wird[4]. Die Grundparadigmen solcher Lizenzen können der nachfolgenden Tabelle entnommen werden. Nach der Open Source Initiative (OSI) besitzt eine Open Source Lizenz die folgenden zehn Merkmale:

Tabelle 1: Open Source Definition

#	Open Source Definition	Beschreibung
1	Free Redistribution	Die Software darf ausschließlich ohne Lizenzkosten weitergegeben werden.
2	Source Code	Der Quellcode muss mit der Software selbst verteilt werden oder nachträglich kostenfrei beziehbar sein.
3	Derived Works	Veränderungen an der Software sind zulässig. Diese können dann ebenfalls unter eine Open Source Lizenz gestellt werden.
4	Integrity of The Author's Source Code	Die Lizenz kann verlangen, dass der Original Quelltext unversehrt bleibt. Trifft dies zu, so sind Änderungen dennoch erlaubt, wenn sie mit Patch-Files geliefert werden oder die Software beispielsweise nach Änderung einen anderen Namen trägt.
5	No Discrimination Against Persons or Groups	Die Lizenz darf niemanden in der Verwendung benachteiligen
6	No Discrimination Against Fields of Endeavor	Die Software darf keine Einsatzfelder vor der Verwendung ausschließen.

[4] [Jae02], S. 30 f.

7	Distribution of License	Die Rechte des Programms müssen auf jeden übergehen, der dieses verwendet.
8	License Must Not Be Specific to a Product	Wird ein Teil aus einem Programm extrahiert, so gelten für diese Teilsoftware ebenfalls sämtliche Rechte des gesamten Programmpakets.
9	License Must Not Restrict Other Software	Die Lizenz darf andere Lizenzen einer Software nicht einschränken. Beispielsweise darf eine OS-Lizenz nicht verlangen, dass alle Teilkomponenten Open-Source sind.
10	License Must Be Technology-Neutral	Die Lizenz darf nicht vorschreiben, wie sie verwendet wird. Z.B. darf sie nicht festlegen, welche Technologie oder Nutzermaske verwendet wird.

Quelle: Mit Änderungen entnommen aus: http://www.opensource.org/docs/osd (Abruf 05.07.2012)

Auf diesen Paradigmen aufbauend sind bis heute weit über 60 Lizenzen entstanden. Die folgenden Erklärungen werden nun einzelne spezielle Aspekte nennen. Sie sollen dem Leser einen kurzen Einblick in bekannte Vertreter von Open Source Lizenzen geben. Der Leser soll einen Überblick erhalten um im Weiteren entscheiden zu können, ob beispielsweise die Quelltext-Offenlegung für ihn in Frage kommt. Für ausführliche Erklärungen, wie auch die Lizenztexte selbst sei auf die Fußnoten sowie die Links unter *Anlage B: Online Versionen bekannter Open Source Lizenzen* verwiesen.

MIT

Die MIT Lizenz erlaubt sämtliche Verwendungszwecke wie auch Modifikationen an der Software. Eine Weiterverwendung bedarf nicht zwingend der öffentlichen Bekanntmachung des neuen Quelltextes.[5]

[5] [Lau04], S. 14 f.

New BSD License (3-clause license)

Diese Lizenz ist der MIT Lizenz sehr ähnlich, mit folgender Erweiterung: Bei Verbreitung des Quelltextes oder der Software muss ein Vermerk auf den Urheber und die Lizenz existieren. Wird die Software verändert, so ist diese Namensnennung zu entfernen.[6]

Apache License 2.0

Diese Lizenz nutzt die gleichen Sachverhalte wie die BSD-Lizenz in Bezug auf die Verbreitung der Software (Quelltext und Binärdateien sind erlaubt) ebenso wie die Nennung des Urhebers. Als neue Eigenschaft enthält sie eine Klausel, welche explizit die Patentrechte dem Urheber der Software zugesteht.[7]

MPL-2.0

Sämtliche Teile einer neuen Software, die Quelltext unter der MPL-2.0 Lizenz verwenden, müssen ihren Quellcode ebenfalls weitergeben. Dies bedeutet, dass alle Dateien, welche in irgendeiner Art und Weise MPL-2.0 Source Code enthalten, öffentlich gemacht werden müssen. Alle anderen Dateien sind von der Veröffentlichungspflicht ausgenommen.[8]

Die GPLv3 / LGPL

Programme, welche diese Lizensierung verwenden sind dazu verpflichtet, neue Applikationen ebenfalls unter diese Lizenz zu stellen. Des Weiteren muss beim Verbreiten der Software stets der Quelltext enthalten sein.[9] Eine Sonderform stellt die LGPL Lizenz dar. Diese entspricht weitestgehend der GPL-Lizenz mit einer funktionalen Änderung: Bei Erstellung einer neuen Applikation, welche eine mit der LGPL versehene Bibliothek

---

[6] [Jae02], S. 54 f.

[7] [Wil05]

[8] [Moz12]

[9] [Jae02], S. 31 ff.

verwendet, muss das neue Programm nicht unter die gleiche Lizenz gestellt werden. Diese Erweiterung trifft beispielsweise zu, wenn es sich bei der Bibliotheksnutzung um externe Aufrufe handelt, d.h. dass beide Programme nicht miteinander kompiliert und vertrieben werden.[10]

Der iipsrv steht unter der GPLv3 Lizenz.[11] Damit eine Verwendung ohne Freigabe der Quelltexte dennoch möglich ist, wurde wie zuvor bereits erwähnt ein Sondervertrag geschlossen, welcher der dotSource die Nutzung unter dieser Bedingung ermöglicht. Im Rahmen dieser Arbeit kann auf die einzelnen Sonderklauseln des Vertrages jedoch nicht weiter eingegangen werden.

[10] [Jae02], S. 50 ff.

[11] [Pil12a]

2 Migration und Erweiterung des dsImageServers

2.1 Migration auf die Version 0.9.9

Im Folgenden wird das im dsImageServer verwendete Framework des iipsrv von der Version 0.9.8 auf die Version 0.9.9 angehoben. Dabei gilt es eine fehlerfreie Migration sicherzustellen.

2.1.1 Ziel einer Migration

„Die Erhaltung der ursprünglichen Funktionalität ist das oberste Ziel einer Migration.“[12] Um dies zu gewährleisten muss der Zustand vor- und nach der Änderung des Programms überprüft werden. Die gewünschte Fehlereingrenzung geschieht mittels Regressionstests. Bei diesen Tests werden vor und nach der Änderung an der Applikation die Ausgabedaten bei gleichbleibenden Eingaben miteinander verglichen. Dabei wird zwischen zwei Arten, dem vollen- und selektiven Regressionstest, unterschieden. Letzterer stellt zwar keine 100-prozentige Testabdeckung dar, spezialisiert sich jedoch bei reduziertem Aufwand auf die repräsentativen Anfragen an das System.[13] Im Folgenden werden nun ein Vorgehensmodell und Testverfahren erläutert, welche eine selektive Testabdeckung bei der Migration gewährleisten sollen.

[12] [Sne10], S. 33

[13] [Sne10], S. 164 ff.

2.1.2 **Test-driven development**

Diese Form der Programmumsetzung nutzt den Test als ein zentrales Element in der Entwicklung. In der Literatur wird davon ausgegangen, dass zuerst Testfälle erstellt werden. Diese stellen Features dar, die in kleine Teile zerlegt werden.[14] Anschließend beginnt der eigentliche Test-Entwicklungs-Prozess. Dieser gliedert sich in die folgenden drei Teilbereiche:

1. Test schreiben
2. Lösungssoftware zum aktuellen Test erstellen
3. Refakturierung der Funktion

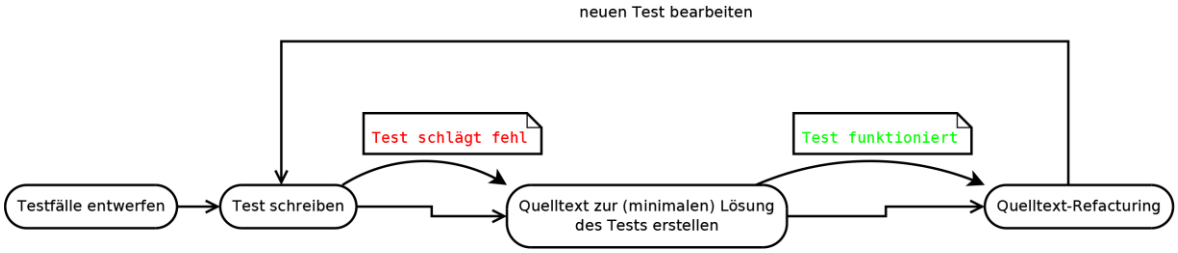
Bei der Abarbeitung der einzelnen Testfälle wird stets von einem Fehler ausgehend zu einer funktionierenden Lösung gearbeitet. (Von einem Fehler in rot zu einer Lösung in grün) D.h. der erstellte Test schlägt zu Beginn zwangsläufig fehl, da die benötigten Funktionen noch nicht existieren. Der anschließend entwickelte Programmcode soll den Testfall nun möglichst schnell erfüllen. Dabei wird kein Wert auf Lesbarkeit und Abstraktion gelegt. Vielmehr soll innerhalb kurzer Zeit (1...5min) ein Erfolgserlebnis in Form eines funktionierenden Tests erhalten werden.[15] Nachdem dieser nun erfolgreich verläuft, wird der zuvor entworfene Programmtext refrakturiert. An dieser Stelle wird er auf Quelltextduplikate, Strukturverbesserungen usw. untersucht und angepasst. Nach Abschluss eines Testfalles wird dieser Zyklus nun solange wiederholt, bis alle Testfälle erfüllt sind. Der nachfolgenden Abbildung kann dieser Zyklus nochmals entnommen werden.

---

[14] [Pic11], S. 40

[15] [Wes06], S. 59

Abbildung 2: TDD Zyklus



Quelle: eigene Darstellung

Ein Vorteil dieser Programmiermethodik ist es, dass sie schnell Erfolgserlebnisse schafft, da zwischen der Lösung einzelner Testfälle nur wenige Minuten vergehen. Es wird versucht stets nur so viel Programmcode zu erstellen, wie durch die Tests gefordert wird. Dadurch ist gewährleistet, dass eine vollständige Testabdeckung der Software vorliegt und diese mittels automatischer Tests jederzeit auf Funktionstüchtigkeit untersucht werden kann. Nennenswerte Programme für automatische Tests sind beispielsweise JUnit (Java), PHPUnit (PHP) und CppUnit (C++).

Ein Kritikpunkt ist, dass bei dieser Entwicklungsmethodik keine Architektur vorgegeben wird. Sie ergibt sich erst aus den einzelnen Testlösungen. Es wird demzufolge nicht vorher eine abstrakte Darstellung der Architektur geschaffen sondern diese entsteht dynamisch aus den erstellten Tests und deren Lösungen während der einzelnen Refakturierungsschritte. Nicht zuletzt hängt bei der Nutzung dieses Verfahrens viel vom Entwickler selbst ab. Zur Erfüllung der zuvor erwähnten Quellcode-Abdeckung muss strikt darauf geachtet werden, dass stets nur der Programmtext zur Lösung des aktuellen Testfalles erstellt wird. Des Weiteren sollte der Programmierer ausschließlich an einem Test zur gleichen Zeit arbeiten. Da durch Sprünge in der Entwicklung der Effekt eines schnellen Erfolges abschwächt.[16] Neue Gedanken während einer Umsetzung können

[16] [Wes06] S. 72

jedoch in Form neuer Testfälle festgehalten werden.

Da die Funktionen des dsImageServer bereits existieren, kann bei der Testerstellung nicht mehr vom TDD als Entwicklungsmethode gesprochen werden. Jedoch können einzelne Aspekte dieses Verfahrens im weiteren Verlauf der Migration dazu verwendet werden, eine nachträgliche automatische Testabdeckung zu erreichen. Diese bereits erwähnten UnitTests werden nun genauer analysiert und später während der Migration integriert.

2.1.3 **UnitTests zur Testunterstützung**

Sie dienen dem automatischen Prüfen von Klassen, Modulen oder Methoden. Automatisch bedeutet, dass der Quelltext eines jeden Tests so entworfen wird, dass mittels eines Befehlsaufrufes die Tests zu jedem Zeitpunkt wieder ausgeführt werden können.[17] Ein Vorteil dieses Testverfahrens besteht darin, dass diese Tests immer zur Verfügung stehen und schnell aufzeigen können, ob ein System nach einer Änderung weiterhin funktioniert. Dadurch wird die Fehlerrate gesenkt.[18]

Im Verlauf dieser Arbeit wird die UnitTest-Implementierung PHPUnit verwendet. Die darin enthaltenen Tests geschehen über sogenannte Assertions (Zusicherungen). Wird einer Assertion ein unerwarteter Wert übergeben, so interpretiert PHPUnit dies als einen Fehler und gibt eine entsprechende Meldung aus. Der Aufruf eines UnitTests geschieht über die Kommandozeile mithilfe des Befehls: *phpunit*. Nachfolgend kann eine der eben genannten Fehlermeldungen eingesehen werden.

[17] [Möh09], S. 384

[18] [Möh09], S. 385

Abbildung 3: PHPUnit - Ausgabe auf Konsole

```
jma@dsImageServer:~/workspace/Migration_IIPSRV_0.9.9/UnitTests/tests$ phpunit --filter VIF
PHPUnit 3.6.10 by Sebastian Bergmann.

Configuration read from /home/jma/workspace/Migration_IIPSRV_0.9.9/UnitTests/tests/phpunit.xml

..F..

Time: 0 seconds, Memory: 8.25Mb

There was 1 failure:

1) VIFTest::testGetWithChangedColour
The two images for >VIF=@clipart#UnitTests#Heart.eps#wid{100}#hei{200}#dpi{1000}  differs in 4 pixels.
Failed asserting that '4' matches expected 0.

/home/jma/workspace/Migration_IIPSRV_0.9.9/UnitTests/tests/AbstractCase.php:138
/home/jma/workspace/Migration_IIPSRV_0.9.9/UnitTests/tests/AbstractCase.php:87
/home/jma/workspace/Migration_IIPSRV_0.9.9/UnitTests/tests/VIFTest.php:84
/home/jma/workspace/Migration_IIPSRV_0.9.9/UnitTests/tests/VIFTest.php:49

FAILURES!
Tests: 5, Assertions: 8, Failures: 1.
```

Quelle: eigene Darstellung

Am unteren Bereich der Abbildung ist ersichtlich, dass fünf Tests mit insgesamt acht Assertions getroffen wurden. Eine davon ist fehlerhaft, woraufhin PHPUnit eine Fehlermeldung mitsamt der Aufrufhierarchie des Fehlers (Stack Trace) ausgibt. Der Aufbau von Testklassen für PHPUnit kann beispielhaft der *Anlage C: UnitTest – Template Klasse* entnommen werden.

2.1.4 **Vorbereitung: Erstellung der UnitTests**

Die Migration wird durch selektive Regressionstests abgedeckt. Dazu erstellt der Autor dieser Arbeit UnitTests um die Richtigkeit der aktualisierten Serverversion sicherzustellen. Zur Nutzung von PHPUnit werden die Bibliotheken *php5-curl*, *phpunit* und *ImageMagick* benötigt. Die Kommunikation geschieht über das Hypertext Transfer Protocol (HTTP) in Verbindung mit GET-Parametern der URL. Die UnitTests richten sich dabei nach den Rückgabedaten. Es wird also ausschließlich geprüft, ob bei einer Anfrage an den Server ein bestimmtes Ergebnis erhalten wird. Zu prüfende Rückgabewerte sind dabei:

- Zeilenweise Ausgabe von Informationen (Server- oder Bildinformationen)
- Rückgabewerte in Form von Bilddaten
- Rückgabedaten als JavaScript Object Notation (JSON)

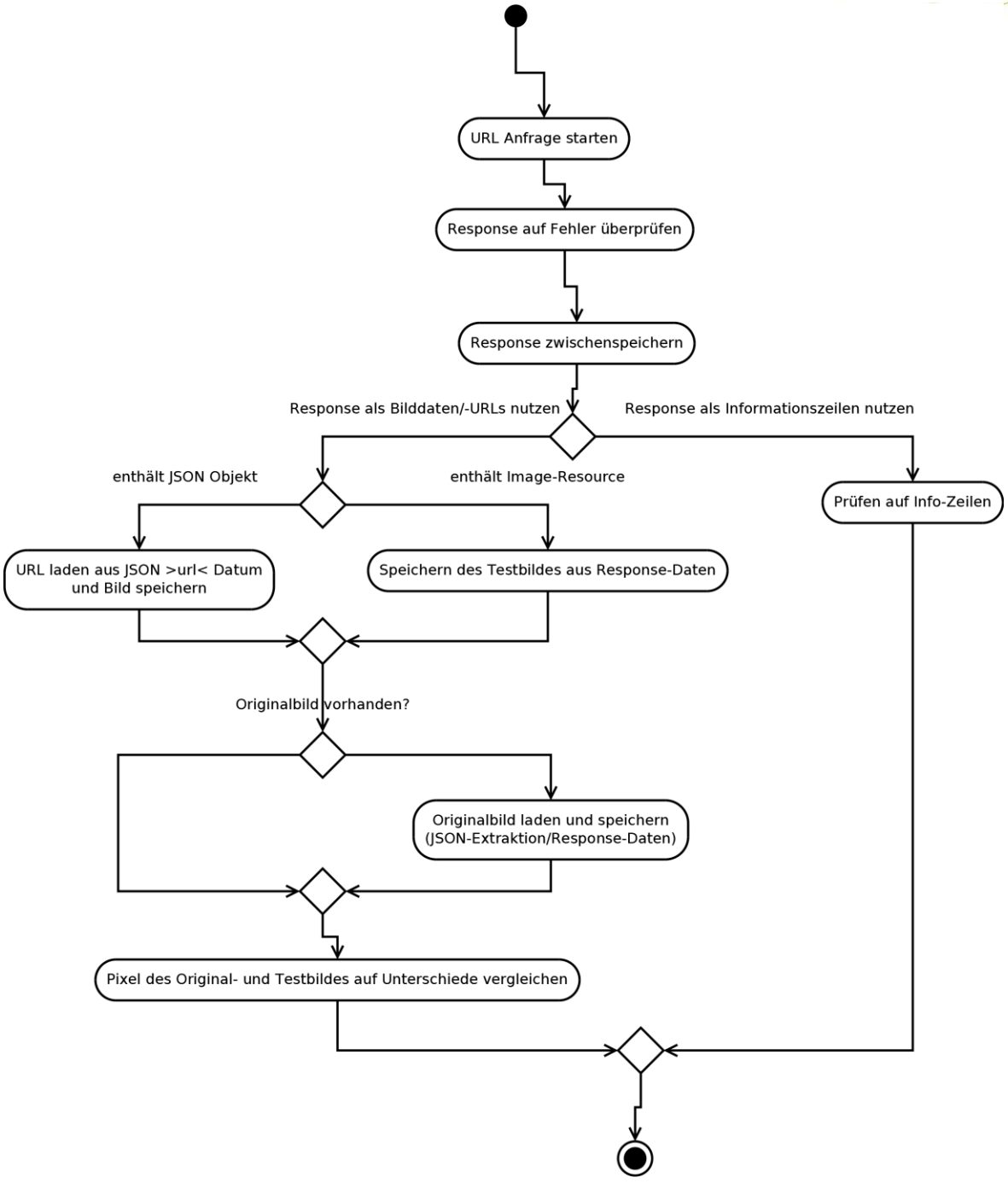
Fehler werden auf drei Rückgabedaten geprüft. Der Response beginnt mit:

- Error – Der dsImageServer hat einen Fehler gemeldet.
- Internal Server Error – Der dsImageServer-Prozess ist abgestürzt. Apache sendet eine Fehlermeldung.
- Der JSON-Response enthält den Eintrag: status => error

In jedem dieser drei Fälle kann die Logdatei des dsImageServers zur Fehleranalyse hinzugezogen werden.

Bei mehrmaligem Anfragen einer URL soll stets das gleiche Bild zurückgeliefert werden. Demzufolge ändern sich die Pixeldaten der Bilder nicht. Dieser Umstand wird genutzt um bei erstmaliger Anfrage nach einem Bild dieses abzuspeichern und im Weiteren mit der zu testenden Anfrage auf einen anderen dsImageServer zu vergleichen. Der Vergleich geschieht über das Command-Line-Tool von ImageMagick (*compare*), einer Bibliothek zur Manipulation von Bildern. Dabei werden die beiden Bilder (Original und Testbild) miteinander auf Pixelunterschiede untersucht. Der nachfolgenden Abbildung können die drei möglichen Response-Typen und deren Verwendung eingesehen werden.

Abbildung 4: UnitTest - Verwendung beim dsImageServer



Quelle: eigene Darstellung

2.1.5 **Durchführung**

Nach Erstellung der UnitTests wird nun die eigentliche Migration durchgeführt. Dazu müssen vom Autor die unterschiedlichen Quelltextstände miteinander verglichen und für

die neue Serverversion entsprechend verwendet werden. Zusätzlich müssen die Makefiles neu erstellt und angepasst werden.

Zum Compilieren der neuen Version (dsImageServer 1.1.0) werden zwei neue Bibliotheken benötigt. Diese sind zuvor über die Paketverwaltung zu installieren:

- *liblcms1-dev*
- *libmemcached-dev*

Die Migration selbst findet in zwei Teilschritten statt:

1. Quelltextunterschiede zwischen dem dsImageServer und der neuen iipsrv Version analysieren und zusammenführen.
2. Erstellen der Makefiles für iipsrv 0.9.9 und Ergänzung der dsImageServer Einträge.

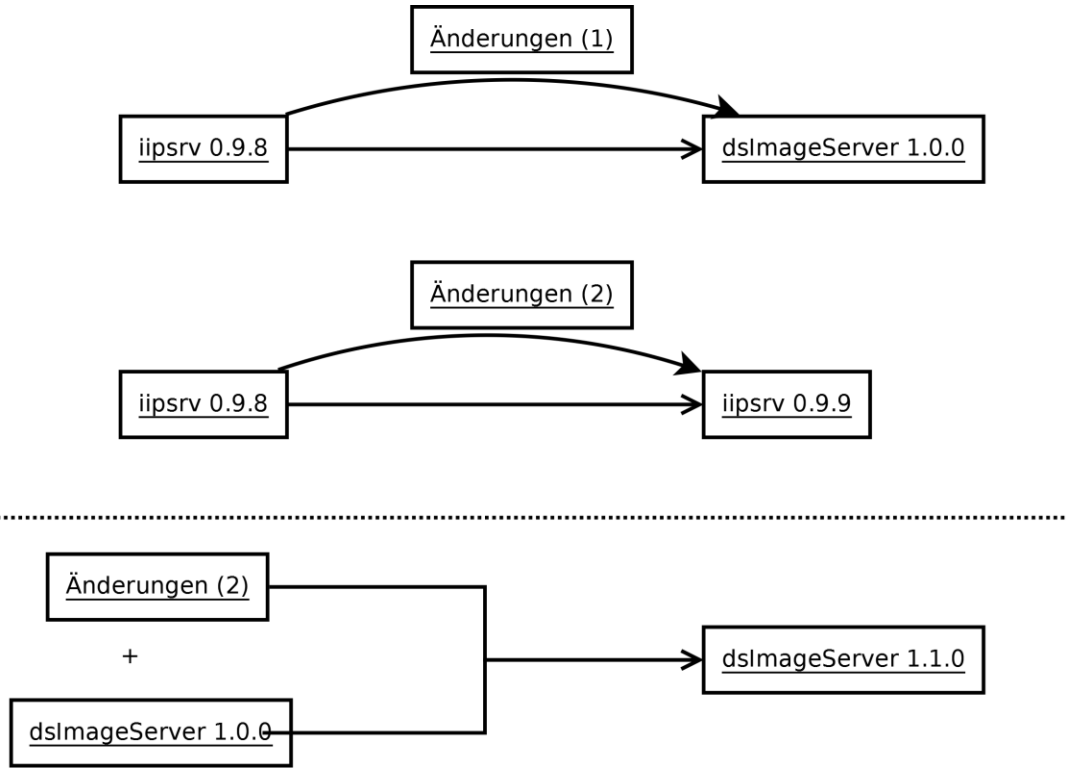
Bei der bereits existierenden Erweiterung des original iipsrv um die dsImageServer Komponenten wurden neue C++-Klassen erstellt und ebenfalls Änderungen in den vorhandenen Quelltexten getätigt. Bei der Migration muss nun darauf geachtet werden, dass:

a. Alle neuen Quelltexte enthalten (Änderungen iipsrv 0.9.8 zu iipsrv 0.9.9)
b. Die vorherigen Änderungen des dsImageServers weiterhin vorhanden (iipsrv 0.9.8 zu dsImageServer 1.0.0) sind und
c. Im Weiteren die Funktionslogik durch die beiden ersten Punkte nicht beeinträchtigt wird. (dsImageServer 1.0.0 zu dsImageServer 1.1.0)

Dazu werden die einzelnen Differenzen im Quelltextstand mittels Versionsverwaltung-Tool Apache Subversion (SVN) ermittelt, analysiert und innerhalb der Migration als neue Version (dsImageServer 1.1.0) abgespeichert (revisioniert). In *Anlage D: Migration – geänderte Dateien des dsImageServers* ist die Liste der veränderten Dateien des dsImageServers am vorhergehenden Stand des iipsrv einsehbar. Diese erstellte Liste dient dazu, die gewünschten Änderungen der dotSource Erweiterungen (iipsrv 0.9.8 zu dsImageServer 1.0.0) zu finden und in die neue Version (iipsrv 0.9.9 zu dsImageServer

1.1.0) einfließen zu lassen. Quelltextzeilen, welche sich bei der Erstellung des dsImageServers und ebenfalls dem Versionssprung vom iipsrv 0.9.8 auf iipsrv 0.9.9 ergeben haben, sind einer gesonderten Analyse zu unterziehen. Die Nutzung der einzelnen Änderungen am Quelltext über die einzelnen Versionen kannn der *Abbildung 6: GNU – Autotools* nochmals strukturiert entnommen werden.

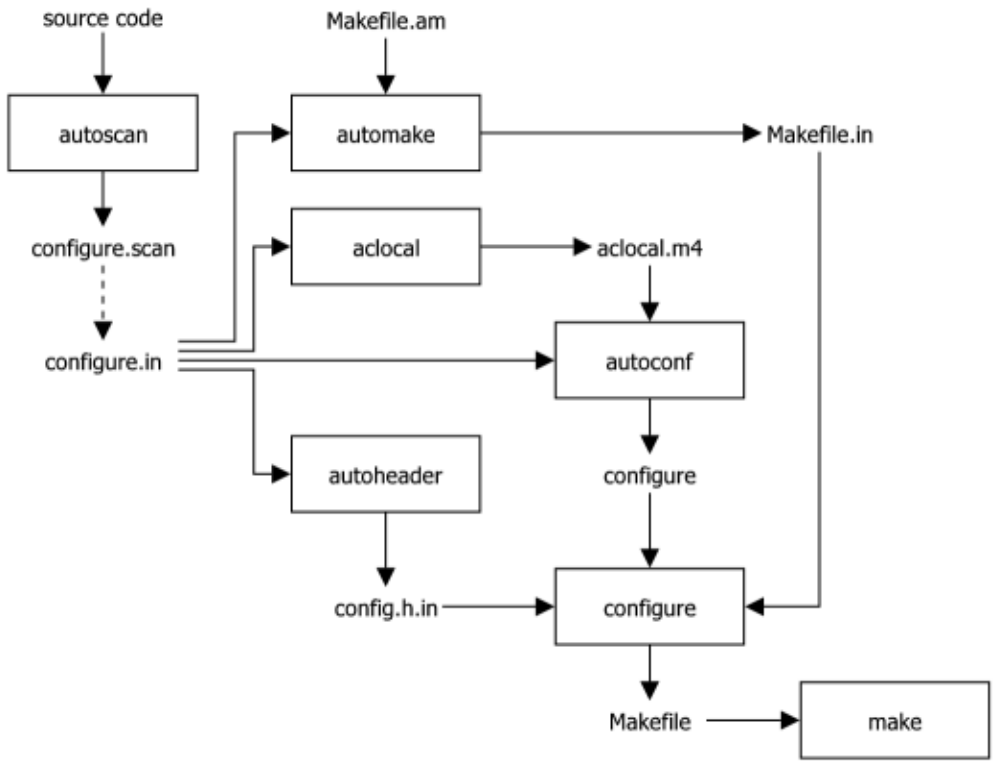
Abbildung 5: Verwendung von Änderungen bei der Migration



Quelle: eigene Darstellung

Eine weitere Teilaufgabe der Migration ist die Anpassung der Makefiles. Die Original Makefiles des iipsrv können über die GNU Autotools *(automake, autoconf*, etc.) auf dem zu verwendenden System generiert werden. Nach Ausführung dieser Kommandozeilen-Tools werden daraus Linux distributionsspezifische Makefiles erstellt. Die Reihenfolge der Aufrufe wird skizzenhaft in der nachfolgenden Abbildung dargestellt. Ebenso sind die zugehörigen projektspezifischen Kommandozeilen-Eingaben festgehalten. Diese sind der *Anlage E: Build des IIPImage Server mithilfe der GNU Autotools* zu entnehmen.

Abbildung 6: GNU – Autotools



Quelle: http://www.kitty.in.th/wp-content/uploads/2004/01/autotools.png (Abruf: 25.05.2012)

Während der Migration werden die Änderungen, welche bereits in der Vorgängerversion des dsImageServers in den Makefiles enthalten sind, eingefügt. Diese Anpassungen beschränken sich in der Regel auf das Einbinden neuer Bibliotheken (*ImageMagick, libXerces, etc.*) und das Aufzählen der im dsImageServer hinzugefügten Quelldateien (*VIF.cc, dsColorAPI.h,* etc.). Die Dokumentation der auszuführenden Befehle und Änderungen an diesen Dateien ist in der *Anlage F: Änderungen des dsImageServers an der Datei: ./src/Makefile* dokumentiert.

2.1.6 **Probleme**

Wie bereits erwähnt geschieht der Kompiliervorgang des Servers über die GNU-Autotools. Leider ist die Erweiterung der verwendeten Dateien (*configure.in*) nicht allen Bedürfnissen entsprechend. Beispielsweise kann mit dem Befehl *autoconf* auf

Vorhandensein von verschiedenen Bibliotheken geprüft werden, bevor die entsprechende Makefile erstellt wird. Leider können dabei nur prozedurale Funktionen und keine Klassenmethoden berücksichtigt werden.[19] Das Überprüfen einer C++ Funktion wird jedoch z.B. zur Einbindung der *libXerces* benötigt. Eigenentwicklungen von Include-Tests sind möglich, aufgrund der vier abzuändernden Zeilen der dsImageServer-Makefile, ist diese jedoch dem Aufwand-Nutzen nicht förderlich.

Es ist innerhalb der UnitTest Entwicklung nicht möglich die Komponenten unabhängig voneinander zu testen. Der VIF Befehl beispielsweise benötigt stets einen Nachfolgebefehl, welcher nach Bilddaten (*JTL*, *PTL*) fragt. Somit wird stets ein VIF Command in Verbindung mit anderen Befehlen gesendet. Es entsteht somit eine Abhängigkeit, die nicht feingranularer auseinandergenommen werden kann.

Eine weitere Einschränkung der UnitTests besteht in der automatischen Ausführung. Bei Rückgabedaten in Form von Bildern werden die Unterschiede in den Pixeln der originalen und zu testenden Bilder verglichen. Ändert sich nun bewusst etwas an einem Command (beispielsweise durch ein neues Update des iipsrv), so dass der Server Bilder mit anderen Pixeldaten zurückgibt, wird der UnitTest zwischen dem alten und neuen Bild stets Pixelfehler angeben. In diesem Fall muss manuell nach der richtigen Erstellung des Bildes geschaut werden.

2.1.7 **Ergebnis**

Die Migration verlief ohne größere Probleme. Durch die Anpassungen am Original iipsrv mussten die Quelltexte zwar miteinander verglichen werden, die Änderungen an diesen Dateien waren aber eher geringfügig. Durch die Anwendung der UnitTests konnte die Funktionstüchtigkeit des dsImageServers nach der Migration erfolgreich geprüft werden. Des Weiteren ist mit diesen Tests gewährleistet, dass auch bei nachträglichen

[19] [Stö07], S. 97

Änderungen stets die Funktionstüchtigkeit des Servers überprüft werden kann. Dieses stellt eine Unterstützung für den Entwickler über einen längeren Zeitraum dar.

2.2 Analyse und Optimierungen

2.2.1 Einleitung

Die zweite Aufgabe dieser Arbeit bezieht sich auf die Analyse und Optimierung von einzelnen Anfragen an den dsImageServer. Da Requests an den Server in der Live-Umgebung nach erstmaligem Aufruf gecached werden, wird sich die folgende Analyse auf die Initialanfragen zu einer Bildressource beschränken.

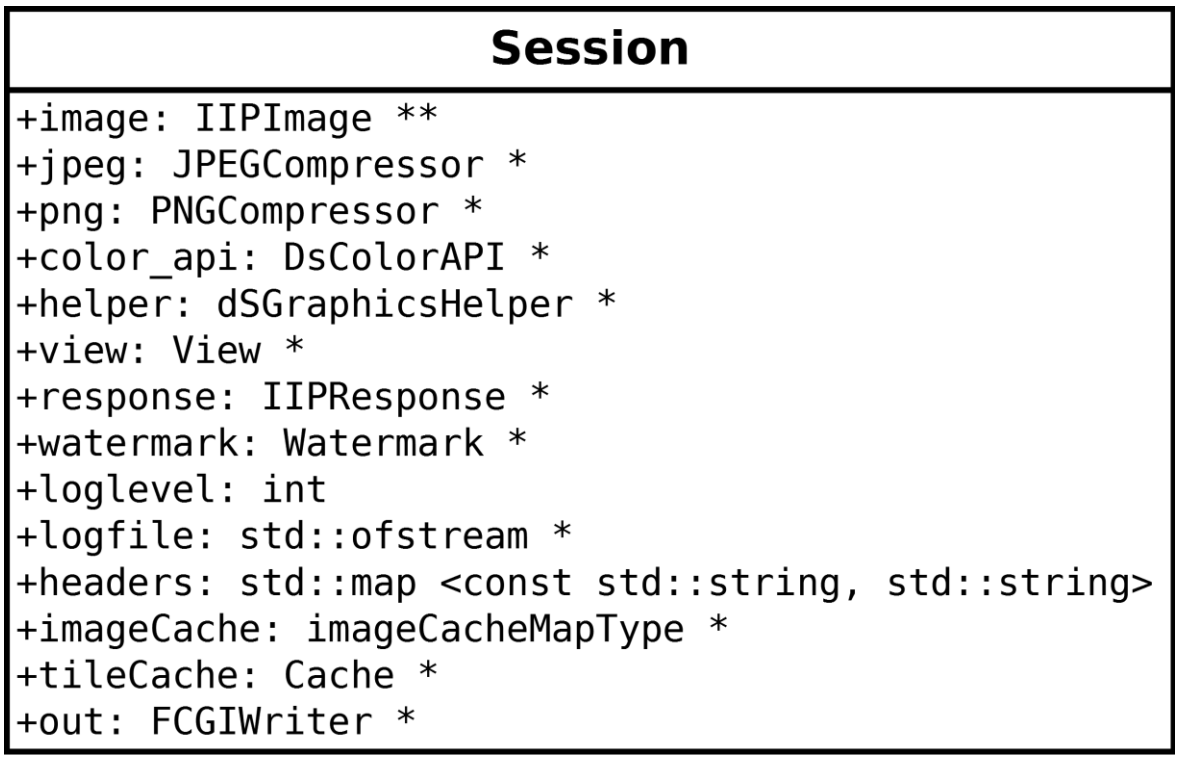
2.2.2 Dokumentation

Die folgende Dokumentation des Systems wird sich auf die Zusammenarbeit der einzelnen Klassen innerhalb einer Anfrage beschränken. Dabei werden Abhängigkeiten und Aufrufreihenfolgen mittels ausgewählter Klassendiagramme dargestellt. Eine ausführliche Klassendokumentation der einzelnen Methoden und Variablen existiert bereits und kann der entsprechenden dsImageServer Dokumentation entnommen werden. Diese befindet sich am Quelltext im Ordner */doc* und */doxygen-doc*. Die nun folgende Dokumentation wird im Rahmen dieser Arbeit neu erstellt.

Die Befehlsabarbeitung beginnt in der Datei *Main.cc*. Diese führt zu Beginn benötigte Initialisierungen wie das Setzen eines Wasserzeichens (Klasse *Watermark*) oder Integrieren des Memcache-Servers durch. Anschließend verbleibt die Instanz im Wartemodus bis eine neuer Anfrage ankommt. Tritt der Fall auf, so wird dieser in seine Teilanfragen an die Klasse *Task* weitergereicht. Diese erstellt eine Session Variable und reicht die Requestdaten jedem Command weiter. Innerhalb der Sessionstruktur werden alle global zu verwendenden Variablen festgehalten. Die Klasse *Task* reicht diese Struktur an alle aufrufenden Commands weiter. *Session* dient in der Regel dazu die initialisierten

Instanzen der Bildkompressoren, Hilfsklassen und Response-Daten (*IIPResponse*) festzuhalten. Die folgende Abbildung zeigt die enthaltenen Daten der Struktur.

Abbildung 7: Dokumentation - Session Struct

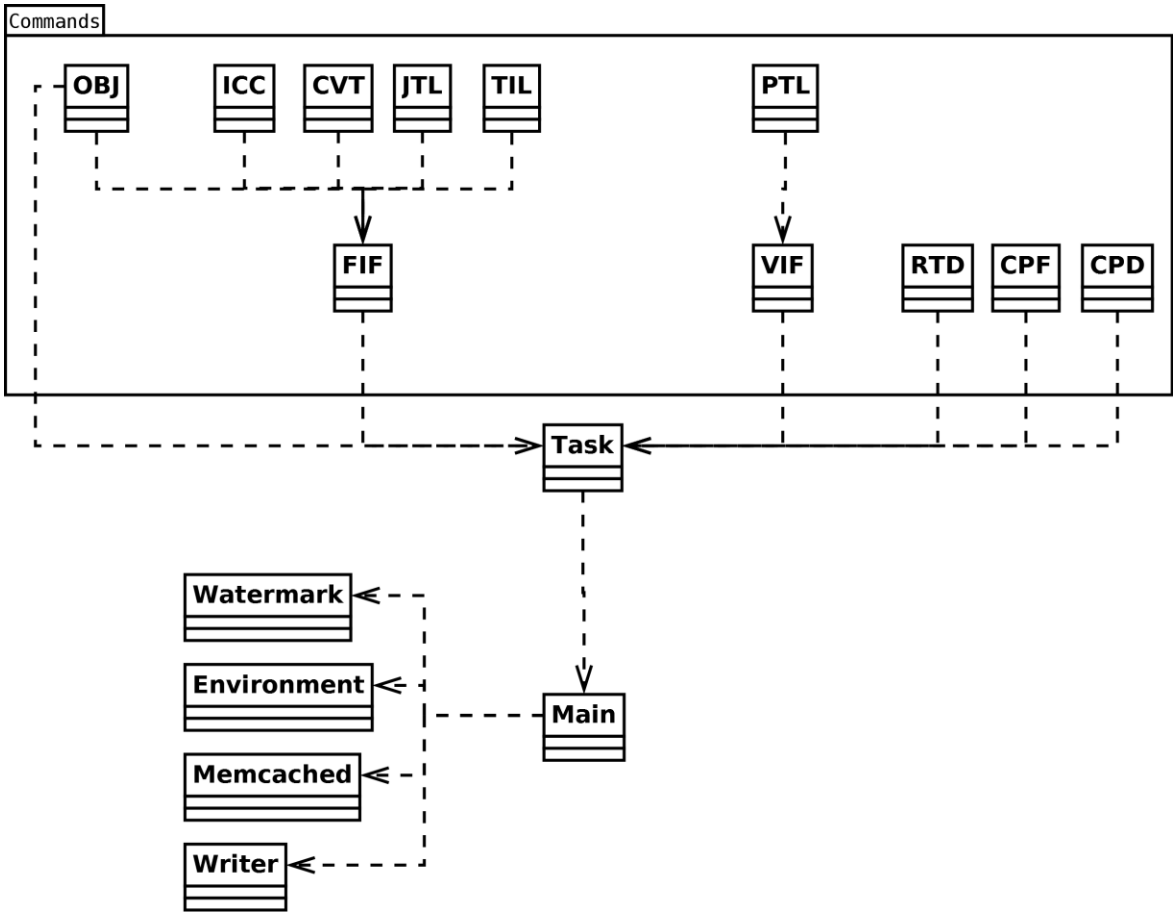


Quelle: eigene Darstellung

Wie bereits in Kapitel *1.3 Was ist der dsImageServer* erläutert werden Anfragen an diesen mittels Command-Pattern abgearbeitet. Die Klasse Task nimmt dabei die Anfragen entgegen und reicht sie an die einzelnen Instanzen der Commands weiter (i.d.R. Klassennamen mit drei Buchstaben). Der nachfolgenden Abbildung können die einzelnen verwendeten Commands entnommen werden. Die entsprechenden Commands sind dabei zumeist abhängig von Bildressourcen. Da diese in den Klassen *FIF* (iipsrv) und *VIF* (dsImageServer) generiert werden, sind diese Commands vielen Befehlen vorangestellt. Die Abbildung ist so gestaltet, dass die linken Commands vom iipsrv und die auf der rechten Seite vom dsImageServer hinzugefügt wurden. Die beiden Bilderstellungs-Klassen generieren eine allgemeinlesbare Bildressource, welche aus verschiedenen Bildtypen generiert werden kann. *FIF* unterstützt dabei nur die Verwendung von Raster-Bilddaten,

wohingegen *VIF* zusätzlich Vektordaten in Form von EPS Dateien verwenden kann. Es sei hier noch angemerkt, dass alle Bildformate, welche mit FIF erstellt werden können, die Möglichkeit besitzen dies mittels VIF zu tun.

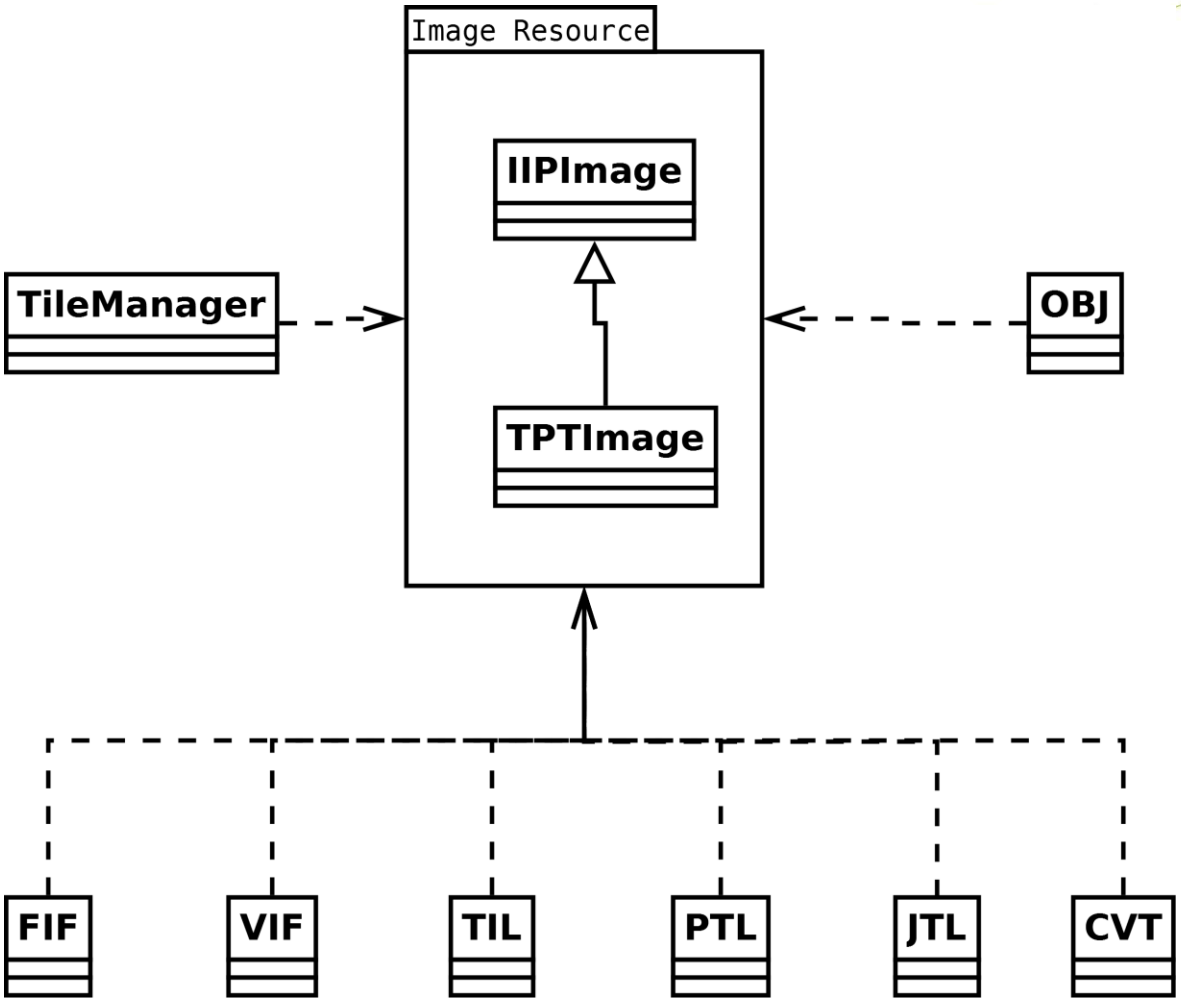
Abbildung 8: Dokumentation - Commands und deren Abhängigkeiten



Quelle: eigene Darstellung

Wurde nun von einen der beiden Commands *FIF* oder *VIF* eine Bildressource erstellt, so wird diese innerhalb der Klasse *IIPImage* abgelegt. Alle nun dargestellten Commands nutzen diese Bildressource.

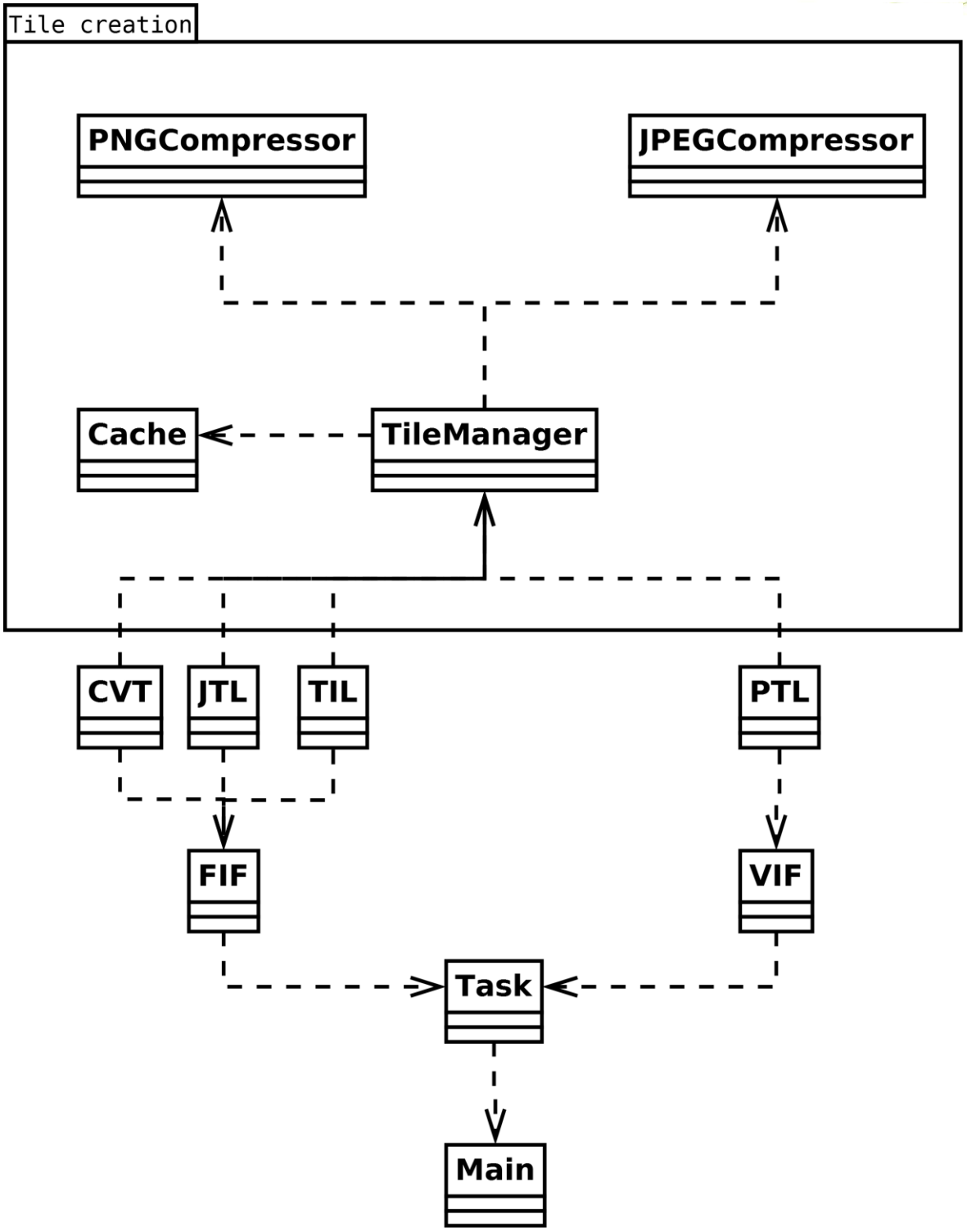
Abbildung 9: Dokumentation - Erstellung und Nutzung der Bildressource



Quelle: eigene Darstellung

Durch die Umsetzung der dsImageServer Funktionalitäten, wurde eine weitere Möglichkeit der Bildauslieferung an den Browser erstellt. Dieser kann nun Bilder im Joint Photographic Experts Group (JPEG) und PNG Format erhalten. Dementsprechend werden die beiden Klassen *JPEGCompressor* und *PNGCompressor* dazu verwendet die Bildressource in das entsprechende Format zu bringen. In der Klasse *TileManager* werden die gewandelten Bilddaten in Kacheln geteilt und an die einzelnen Commands weitergereicht. Der nachfolgenden Darstellung können die Abhängigkeiten der Klassen nochmals entnommen werden.

Abbildung 10: Dokumentation – Kachelerstellung



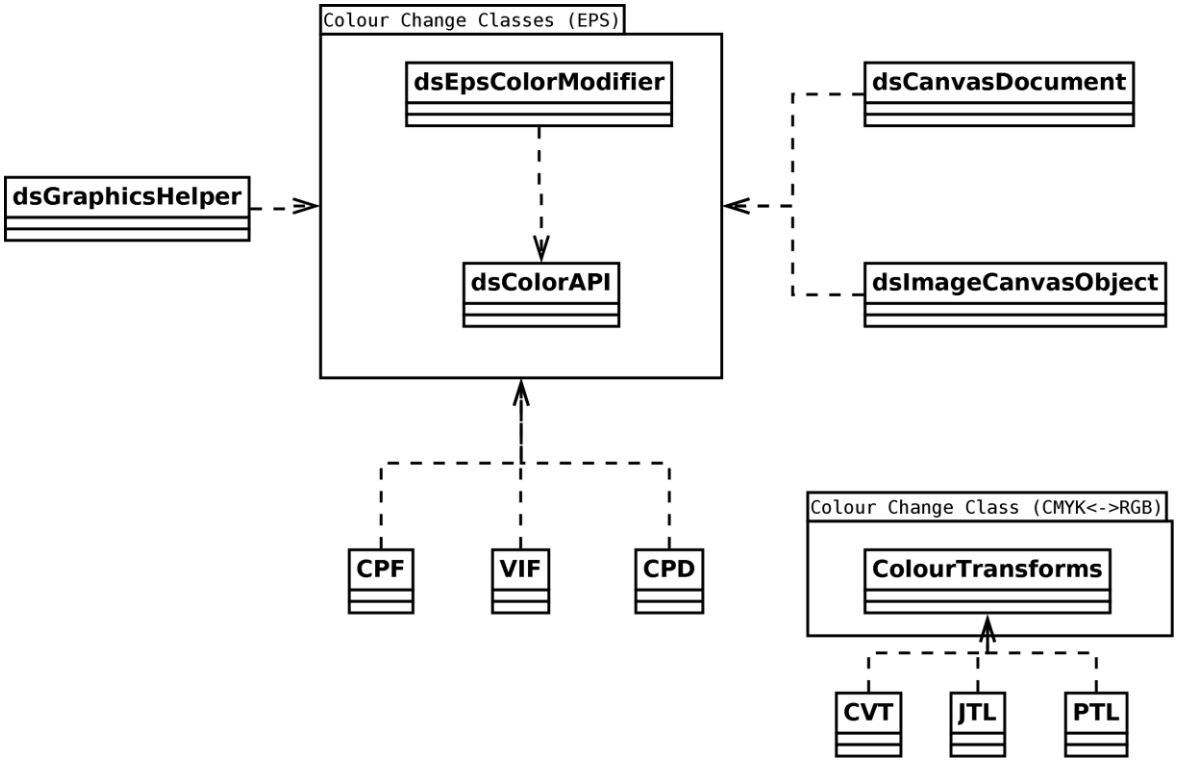
Quelle: eigene Darstellung

Eine weitere erwähnenswerte Funktion ist das Umfärben von Bildern. Dies geschieht auf zwei Arten:

- Färben mittels *ColourTransforms* (iipsrv)
- Umfärben der EPS Dateien in der Klasse *dsEpsColorModifier*

Letztere nimmt Umfärbe-Anforderungen entgegen und schreibt die neuen Farben in die zugehörige EPS Datei. Da dieses Dateiformat aus Klartext besteht, werden die Farbänderungen direkt mittels Zeichenketten-Operationen (Strings) in die Datei geschrieben. Die nachfolgende Abbildung stellt nun die Verwendungen der beiden Umfärbe-Verfahren dar.

Abbildung 11: Dokumentation - Farbänderungen durchführen



Quelle: eigene Darstellung

Nachdem für die einzelnen Klassen des dsImageServers Abhängigkeiten und Nutzungen erläutert wurden, werden nun die wichtigsten Commands auf zeitintensive Codeabschnitte untersucht.

2.2.3 Verwendete Bibliotheken

Innerhalb des dsImageSevers werden verschiedene zusätzliche Bibliotheken verwendet. Die wichtigsten Vertreter, welche in der nachfolgenden Analyse eine Rolle spielen, werden in der nachfolgenden Tabelle genannt. Die Aufgaben der Bibliotheken beziehen sich in der Regel auf die Verarbeitung von Bildressourcen. Die jeweiligen Aufgabenbeschreibungen sind dem Quelltext des dsImageServer entnommen.

Tabelle 2: Analyse - verwendete Bibliotheken

Bibliothek	Beschreibung
Ghostscript	Konvertierung der Bilder von Vektorformaten in Pixelformate.
ImageMagick / Magick++	Wandlung von Bildformaten und –daten innerhalb einer Pixeldatei. Die C++ Implementierung trägt den Namen Magick++.
PDFlib	Erstellen von PDFs aus Text- und Bildressourcen.
VIPS	Erstellung von Tagged Image File Format Kacheln (TIFF-Kacheln).

Quelle: eigene Darstellung

2.2.4 Analyse der zeitintensivsten Befehle

Die Zeitanalyse geschieht in zwei Testverfahren. Durch Messen des Zeitverbrauchs von:

1.) einem gesamten Requests.
2.) einer Teilfunktion (z.B. Methode oder Kommandozeilenaufruf) innerhalb des Servers.

Die Zeitmessungen werden mithilfe von sogenannten Profiler-Klassen erstellt. Diese stellen jeweils eine Methode zum Starten und Stoppen der aktuellen Zeit bereit.

Anschließend wird die errechnete Zeitdifferenz in ms zurückgegeben. Die genannten Klassen werden im Rahmen dieser Arbeit vom Autor erstellt.

Für die Messung gesamter Requests werden die Anfragen der UnitTests (während der Migration erstellt) verwendet. Eine Übersicht der zeitintensivsten Befehle kann der nachfolgenden Tabelle entnommen werden.

Tabelle 3: Analyse - Langsame Requests

Request	**Zeitbedarf**	**Kurzbeschreibung**
VIF=@clipart/UnitTests/Heart.eps/wid{100}/hei{100}/dpi{300}/scl{2}/col{PANTONE+188+C%7CPANTONE+109+C}/ses{UnitTests}&PTL=0,0,0	00:00:19.041	Rückgabe eines Gesamtbildes als PNG aus einer EPS Datei mit einer Umfärbung, Skalierung und Änderung der DPI-Werte sowie den Dimensionen (x, y)
VIF=@clipart/UnitTests/Heart.eps/wid{100}/scl{2}&PTL=0,0,0	00:00:03.328	Einfache Anfrage an eine Vektorgrafik mit Änderung der Dimensionen (scl, wid)
VIF=@clipart/UnitTests/Heart.eps/col{PANTONE+188+C%7CPANTONE+293+C}&PTL=0,0,0	00:00:02.524	Einfache Anfrage an eine Vektorgrafik mit einer Umfärbung.
CPF=UnitTests_CPF_WithClipart/vid{1}/hei{600}	00:00:02.195	Generierung einer Vorschau PNG über die Erstellung der zugehörigen PDF. Enthalten ist hier ein einfaches EPS-Bild.
CPF=UnitTests_CPF_AllTogether/vid{1}/hei{600}	00:00:02.112	Zusätzlich zur obigen CPF-Anfrage wird hier noch ein kurzer Schriftzug erstellt.

CPD=UnitTests_CPD_AllTogether	00:00:01.555	Erstellung einer PDF. Enthalten sind: ein Text und ein EPS-Bild.

Quelle: eigene Darstellung

Da die Zeitmessungen auf einem lokalen System durchgeführt werden, sind die absoluten Zeitwerte als Richtwert anzusehen. Durch die Tabelle ist ersichtlich, dass die folgenden Befehle am zeitaufwendigsten sind:

- VIF
- CPD
- CPF

Diese Befehle werden anschließend in einer genaueren Analyse untersucht. Es werden innerhalb des Servers einzelne Funktionsaufrufe mittels Zeitmessung versehen. Dazu wird den Server Quelltexten eine weitete Klasse *Profiler* hinzugefügt. Der Quellcode dieser Klasse kann unter *Anlage G: Zeitmessung mittels Profiler.h* eingesehen werden. Es stellt sich heraus, dass die externen Aufrufe zur Manipulation und Generierung der Bilder die zeitintensivsten Befehle darstellen. Diese beschränken sich meist auf das Anfragen einzelner Bildattribute.

Die langwierigsten Befehle des VIF-Commands können der nachfolgenden Tabelle entnommen werden. Die Prozentwerte sind gemessen am kompletten Zeitbedarf des Requests. Sie entsprechen dem Mittelwert sämtlicher VIF-Anfragen. Die Sortierung richtet sich nach der Aufrufreihenfolge des VIF-Commands.

Tabelle 4: Analyse - VIF-Command – zeitintensive Methoden

Methode	Befehl	Dauer in %	Beschreibung
dsGraphicsHelper:: isVectorFormat	Kommandozeile: identify (ImageMagick)	19	Auslesen der Bild-Metadaten und Überprüfung ob Vektor- (EPS) oder Rasterbild (JPG, TIF, …)
VIF:: updateDpiToRender	Kommandozeile: identify	12	Abfragen der aktuellen Abmessungen des Bildes und Neuberechnung über Anfrage-Parameter (wid, hei, scl).
dsGraphicsHelper:: convertCmykEPStoRgbPNG	Kommandozeile: gs (Ghostscript)	24	Mittels Ghostscript wird die angefragte EPS in eine PNG gewandelt.
dsGraphicsHelper:: convertToRgbImage	Kommandozeile: identify, convert (ImageMagick)	10	Enthält das Bild ein CMYK-Farbschema, so wird es zu RGB konvertiert.
VIF:: processVirtualImage (Nach Prüfung auf existierende .tif)	Objekt: Image (ImageMagick), Kommandozeile: convert	2	Wenn das Zwischenbild (PNG) bereits im RGB Format ist, wird hiermit geprüft, ob es Fehler enthält. Diese werden im Bedarfsfall behoben.

VIF:: createImageServerTifTiles	Objekt: VImage (VIPS)	32	Erstellung von TIFF-Kacheln in unterschiedlichen Auflösungen.

Quelle: eigene Darstellung

Folgende Verbesserungsvorschläge sind für die Umsetzung zu prüfen:

- Das C++ Application Programming Interface (API) von ImageMagick bietet die Möglichkeit ein Bild einmalig einzulesen und anschließend unterschiedliche Operationen sowie Abfragen z.B. über das Format anzuwenden. Dadurch müssten die externen Aufrufe der ImageMagick Kommandozeilen-Tools (*identify, convert*) nicht mehr verwendet werden. Zur Nutzung der ImageMagick C++ API stehen die Klassen des Namespaces *Magick* (zu finden im Includepath: <*ImageMagick/Magick++*>) zur Verfügung.
- Für die Generierung von Bildern werden unterschiedliche Bildbibliotheken verwendet (Vektor- in Rasterdaten mittels Ghostscript, TIFF-Kachelgenerierung mithilfe von VIPS). Es ist zu prüfen, ob diese Funktionalitäten ebenfalls mit ImageMagick erreicht werden können.
- Überflüssige Zwischenkonvertierungen sind auf Relevanz zu prüfen und evtl. zu entfernen.

Weitere für die Optimierung relevante Commands sind *CPD* und *CPF*. Diese erstellen aus einer generierten XML-Datei eine PDF. CPD liefert diese aus, wogegen CPF aus der PDF anschließend eine PNG generiert. Somit sind beide Funktionen ähnlichen Aufbaus und werden in der nachfolgenden Tabelle gemeinsam betrachtet.

Tabelle 5: Analyse - CPD/CPF-Command – zeitintensive Methoden

Methode	Befehl	Dauer in %	Beschreibung
dsCanvasDocument:: initPDFDocument	PDFlib.load_iccprofile()	10	ICC-Profil des Hintergrundes laden.
dsImageCanvasObject:: addToPDF	dsgraphicsHelper:: isVectorFormat -> Kommandozeile: identify	20	Operationen auf den enthaltenen Bildern als Vektor/Rasterbilder werden hier unterschieden.
dsTextCanvasObject:: addToPDF	PDFlib.load_iccprofile()	15	Zweimaliges initialisieren einer PDF-Objekts für ein Textelement.
CPD:: run	PDFlib.load_iccprofile()	10	Finale PDF wird mit dem CMYK-Profil versehen.
dsCanvasDocument:: applyClippingMask	Kommandozeile: identify (ImageMagick)	0…20	Größe der zu verwendenden Maske erfragen.
dsCanvasDocument:: applyClippingMask	dsGraphicsHelper:: convertCmykEPStoRgbPNG -> Kommandozeile: gs (Ghostscript)	0…20	Maske in das PNG-Format wandeln. (nur für CPF)
dsGraphicsHelper:: convertPostScriptToPDFX	Kommandozeile: gs	20…40	Einzelne EPS Datei in PDF wandeln.

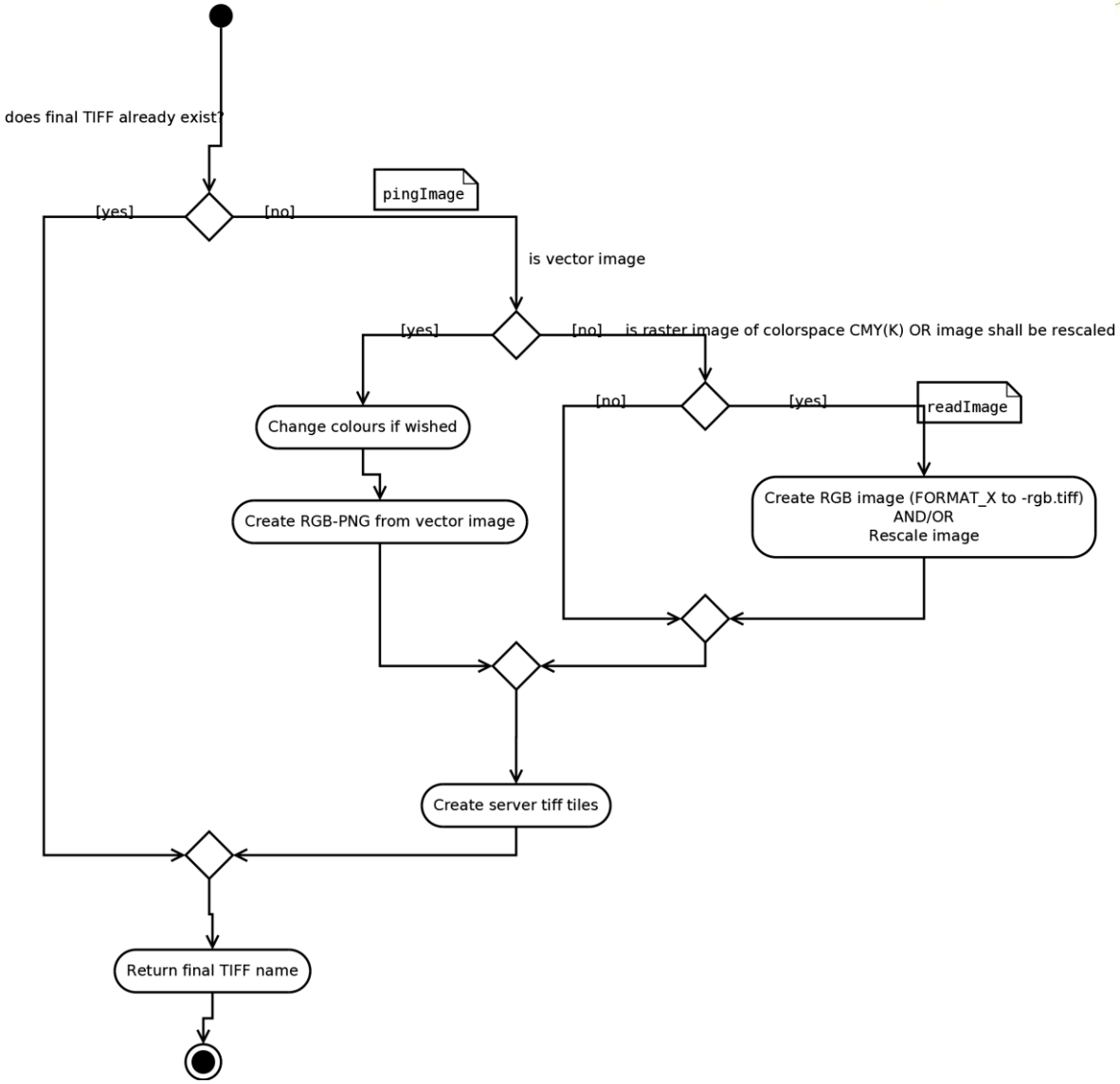
Quelle: eigene Darstellung

Aus der Tabelle ist ersichtlich, dass das Laden der ICC-Profile einen großen Anteil an den Abarbeitungszeiten hat. Das entsprechende Profil, welches geladen wird ist das ISO Coated v2 der ECI (European Color Initiative). Im Folgenden wird geprüft, ob die ICC-Profile zwischengespeichert werden können, um Mehrfachanfragen zu umgehen.

2.2.5 **Optimierung des dsImageServers**

Die Grundstruktur bei der Erstellung einer Image-Resource während des VIF-Commands wird umgestellt. Es wird nun versucht ein Image-Objekt durch den gesamten Request hindurch zu verwenden. Dadurch wird initial ein Aufwand benötigt um die Ressource einzulesen. Anschließend werden für alle weiteren Anfragen an das Bild (z.B. Geometrie oder Farbpalette) die bereits geladenen Objektdaten genutzt (siehe nachfolgende Abbildung). Durch diese Änderung können die Kommandozeilen-Anfragen *identify* und *convert* über die C++ API entfernt werden.

Abbildung 12: VIF – Bildpräparationsprozess



Quelle: eigene Darstellung

Eine weitere Optimierung am VIF-Command ist die Einsparung mehrerer Konvertierungen. Diese sind im Folgenden:

- Bei bestimmten angefragten TIFF Bildern kann es vorkommen, dass die ImageMagick Bibliothek beim Lesen und Schreiben Warnungen wirft. Diese Meldung beim Lesen wurde zuvor dazu verwendet um ein Zwischenbild zu generieren, welches keinen Fehler mehr enthält. Diese Konvertierung wurde entfernt, da sich herausstellte, dass Bilder mit Warnungen bei der weiteren

Verarbeitung keine Probleme bereiten. Es muss im Quelltext ausschließlich darauf geachtet werden, dass die Warnung: *WarningCoder* abgefangen wird.

- Bei Anfrage eines Rasterbildes, welches vom CMYK- in den RGB- Farbraum umgewandelt und ebenfalls reskaliert werden soll, wurden in der alten dsImageServer Version zwei Zwischenbilder angelegt. Durch die Verwendung eines Bildobjektes innerhalb des Commands werden beide Änderungen zunächst in das Bild-Objekt geschrieben und anschließend gemeinsam als ein einzelnes neues Zwischenbild gespeichert.

2.2.6 **Probleme bei der Optimierung**

Alle Bildanfragen des dsImageSevers mittels ImageMagick zu realisieren ist nicht möglich:

- Zum Einen, da die Bibliothek für bestimmte Konvertierungen selbst andere Tools wie Ghostscript benutzt (durch sogenannte delegates),
- Zum Anderen, da die Konvertierung von Vektor- in Rasterbilder keine Funktion darstellt, für die ImageMagick ursprünglich gedacht war (siehe http://www.imagemagick.org/Usage/formats/#vector Abruf: 05.07.2012). Somit würde das Ausbauen von VIPS Performanceverluste nach sich ziehen.

Eine weitere Funktion, welche im Rahmen dieser Arbeit nicht optimiert wurde ist die der PDF-Generierung (CPD und CPF) aus *Tabelle 5: Analyse - CPD/CPF-Command – zeitintensive Methoden*. Einsehbar ist, dass das Laden der ICC Profile viel Zeit in Anspruch nimmt und bei jedem Hinzufügen eines neuen Elementes zur PDF erneut geladen wird. Eine Optimierungsmöglichkeit würde darin bestehen, das geladene Profil innerhalb eines PDFlib Objektes zu halten und während des Requests stets nur eine Kopie an die neuen Objekte weiterzureichen. Der zu erwartende Performancegewinn hängt dabei signifikant von der Anzahl der enthaltenen Elemente (Bilder und Texte) ab. Leider wurde der Kopier-Konstruktor, welcher zur Datenvorhaltung benötigt würde von der Bibliothek privat gesetzt. Nachzulesen in den ChangeLogs der PDFLib v.8.0.4: (2010-02-18 (bug #2644) -

http://www.pdflib.com/fileadmin/pdflib/changelog/changes-PDFlib-8.0.4.txt - Abruf am: 05.07.2012).

Die angesprochenen Probleme stellen gleichzeitig wirkungsvolle weitere Optimierungen dar. Aus genannten Gründen wurden sie jedoch im Rahmen dieser Arbeit nicht umgesetzt. Nachfolgend werden nun die erreichten Ergebnisse der Optimierungen vorgestellt.

2.2.7 Ergebnisse der Optimierung

Durch die Restrukturierung des VIF-Commands wird nun bei Anfrage von Vektorbildern ein Performancegewinn von durchschnittlich 20…40% erhalten. Für die nachfolgende Übersicht wurden die in *Tabelle 3: Analyse - Langsame Requests* genannten Anfragen mit altem- und neuem Codestand mehrfach angefragt und der mittlere Performancegewinn festgehalten.

Tabelle 6: Optimierung - Erhaltene VIF Geschwindigkeitssteigerungen

Request	**alter Zeitbedarf**	**neuer Zeitbedarf**	**Performance-gewinn in %**
VIF=@clipart/UnitTests/Heart.eps/wid{100}/hei{100}/dpi{300}/scl{2}/col{PANTONE+188+C%7CPANTONE+109+C}/ses{UnitTests}&PTL=0,0,0	00:00:18.355	00:00:13.053	29
VIF=@clipart/UnitTests/Heart.eps/wid{100}/scl{2}&PTL=0,0,0	00:00:03.399	00:00:2.608	23
VIF=@clipart/UnitTests/Heart.eps/col{PANTONE+188+C%7CPANTONE+293+C}&PTL=0,0,0	00:00:02.522	00:00:01.565	38

Quelle: eigene Darstellung

Obwohl ausschließlich VIF abgeändert wurde, stellt dieser Command doch einen wesentlichen Anteil an der Nutzung des dsImageServers dar. Der optimierte Server trägt nun die Versionsnummer 1.2.0.

2.3 Erstellung der GPL Version

2.3.1 Enthaltene Klassen

Die GPL Version des dsImageServers enthält eine Untermenge der implementierten Funktionen des von der dotSource verwendeten Servers dar. Dabei sind die Funktionen des iipsrv vollständig integriert. Die enthaltenen Funktionen des dsImageServers sind dabei:

- dsGraphicsHelper
- PTL
- VIF

2.3.2 Umsetzung der Open Source Version

Die Erstellung dieser Server Version basiert auf der aktuellsten dsImageServer Version 1.2.0. Dazu werden alle Dateien des Original Servers kopiert und sukzessiv nicht benötigte Klassen entfernt. Im Folgenden werden detaillierte Änderungen an den zuvor genannten enthaltenen Klassen des dsImageServer genannt:

Tabelle 7: GPL Version - enthaltene dsImageServer Klassen

Klasse	Änderungen
dsGraphicsHelper	Alle Funktionen, die in Zusammenhang mit der PDF-Erstellung von CPF und CPD stehen werden entfernt. Dazu gehören beispielsweise *convertPostScriptToPDFX()*.

PTL	Die komplette Funktionalität des Commands bleibt erhalten. Es werden keinerlei Änderungen an der Klasse vorgenommen.
VIF	Innerhalb dieses Commands wird nur eine Funktion entfernt: Das Umfärben von EPS-Dateien. Dazu sind die entsprechenden Methoden zu entfernen.

Quelle: eigene Darstellung

Zur Sicherung der Funktionstüchtigkeit kommen abermals die bereits zur Migrations-Aufgabe erstellten UnitTests zum Einsatz. Weitere Änderungen müssen für die Open Source Version nicht vorgenommen werden.

2.3.3 **Ergebnis**

Durch den Einsatz der vom Autor programmierten UnitTests stellte die Lösung und Qualitätssicherung dieser Aufgabe keine weiteren Probleme dar. Die Version wurde im SVN gespeichert und kann nun zur Veröffentlichung freigegeben werden.

3 Fazit

Die vorliegende Arbeit stellte das erste Projekt bei der dotSource dar, in welchem der Autor mit der Programmiersprache C++ umzugehen hatte. Das Einarbeiten in die Besonderheiten der Sprache mit Speicherverwaltung, Pointern, und vor allem der Zeitoptimierung im Sinne von Referenznutzung schufen interessante Fragen und Lösungsansätze bei der Arbeit mit dem dsImageServer. Ebenso der Kompiliervorgang mit Einbindung von Bibliotheken, velinkten Dateien usw. stelle eine weitere neu zu erlernende Aufgabe dar. Durch die im Studium und den bereits bewältigten praktischen Arbeiten konnte der Autor die ihm gestellten Aufgaben mit definierten Ergebnissen abschließen:

- Mittels der Migration ist der dsImageServer auf eine einheitliche Version mit dem aktuellen iipsrv gebracht worden.
- Einer der am stärksten verwendeten Commands (*VIF*) arbeitet nun bis zu 40% schneller.
- Dank der Erstellung der GPL-Version des Servers sind die Nutzungsbedingungen zur Verwendung des dsImageServers erfüllt.

Die Arbeit zeigte bezüglich der Analyse ebenfalls Grenzen in der Änderung vorhandener Softwareprojekte auf. Es musste entschieden werden, welche Änderungen an der Software im gegebenen Zeitrahmen dieser Arbeit umsetzbar waren und einen gewünschten Mehrwert schufen. Somit spielte das eigene Zeitmanagement bei der Bearbeitung der Aufgaben eine weitere Rolle. Mithilfe der erstellten Dokumentationen und UnitTests wurden des Weiteren Möglichkeiten geschaffen sich als Entwickler von zukünftigen Anpassungen schneller in den dsImageServer einzuarbeiten und dessen Funktionalität zu überprüfen.

Anlagenverzeichnis

Anlage A: Hilfreiche Lektüre zur Funktionsweise des dsImageServer

Buchtitel	Beschreibung
Stöhr, F.: „Die GNU Autotools – Leitfaden für die Softwaredistribution“, ISBN: 978-3-936546-48-4	Die Befehle autoconf, automake und libtool, etc. werden zum Erstellen der Makefile des original iipsrv verwendet. Dieses Buch zeigt die meisten verwendeten Befehle und Makros auf.

Quelle: eigene Darstellung

URL	Beschreibung
IIPImage http://iipimage.sourceforge.net/	Nennen des Funktionsumfangs und
Internet Imaging Protocol - Version 1.0.5: http://iipimage.sourceforge.net/IIPv105.pdf	Erklärung der standardisierten Befehle und die benötigte URL-Syntax für Anfragen.
The Open Source Definition | Open Source Initiative http://www.opensource.org/docs/osd	Die englische Originaldefinition von Open Source Lizenzen.
Open Source Licenses | Open Source Initiative http://www.opensource.org/licenses/index.html	Vollständige und kategorisierte Liste aller Open Source Lizenzen

Quelle: eigene Darstellung

Anlage B: Online Versionen bekannter Open Source Lizenzen

Lizenz	URL
MIT	http://www.opensource.org/licenses/mit-license.html
BSD 3-Clause (New BSD License, Modified BSD License)	http://www.opensource.org/licenses/BSD-3-Clause
Apache License, Version 2.0	http://www.apache.org/licenses/LICENSE-2.0.html
Mozilla Public License (MPL-2.0)	http://www.mozilla.org/MPL/2.0/
GPL v3.0 / LGPL v3.0	http://www.gnu.de/documents/gpl-3.0.en.html http://www.gnu.de/documents/lgpl-3.0.en.html

Quelle: eigene Darstellung

Anlage C: UnitTest – Template Klasse

```
<?php

class ExampleTestCase extends PHPUnit_Framework_TestCase
{
	/**
	 * Automatischer Aufruf dieser Funktion vor jeder Testausführung.
	 */
	public function setUp()
	{
		parent::setUp();
	}


	/**
	 * Innerhalb einer solchen Funktion beginnend mit test...() werden Assertions
	 * getroffen und auf der Kommandozeile ausgegeben.
	 */
	public function testWeekend()
	{
		$this->assertContains(date('D'), array('Sat', 'Sun'));
	}


	/**
	 * Automatischer Aufruf dieser Funktion nach jeder Testausführung.
	 */
	public function tearDown()
	{
		parent::tearDown();
	}
}
```

Quelle: eigene Darstellung

Anlage D: Migration – geänderte Dateien des dsImageServers

./ - root Ordner

Datei	Beschreibung
AUTHORS	-
ChangeLog	-
configure	Automatisch erstellter Konfigurationsskript
configure.in	Automatisch erstellter Konfigurationsskript
COPYING	-
INSTALL	-
NEWS	-
README	-
TODO	-

Quelle: eigene Darstellung

./src - Ordner

Datei	Beschreibung
Cache	Kachelcache innerhalb einer dsImageServer Instanz
CVT	Auslieferung eines gesamten Bildes
DeepZoom	Schnittstelle zum Deepzoom Framework
Environment	Definierte Konstanten für Server
IIPResponse	Rückgabe der erstellten bilddaten
JTL	Erzeugung JPEG komprimierter Kacheln
Main	Main-Loop
Makefile	-
Makefile.am	-

Makefile.in	-
OBJ	Rückgabe von Bildinformationen
RawTile	Einzelne Kacheldaten beinhaltende Objekte
SPECTRA	Schnittstelle zum SCPECTRA Framework
Task	Weiterleiten der Anfragen zu den einzelnen Commands
TIL	Kachelerzeugung
TileManager	Kachelerzeugung
Zoomify	Schnittstelle zum Zoomify Framework

Quelle: eigene Darstellung

Anlage E: Build des IIPImage Server mithilfe der GNU Autotools

Einzelaufruf der Build Prozesse

```
libtoolize          # Erzeugen neuer libtool Konfigurationen
aclocal             # Erzeugen neuer aclocal.m4
automake            # Erzeugen neuer Makefile.in
autoconf            # Erzeugen neuer configure
./configure         # Erzeugen der neuen Makefile
make                # Finales Bauen des fcgi plugins
```

Quelle: eigene Darstellung

Funktionsbündelung der GNU Autotools mit autoreconf

```
autoreconf --install --force; ./configure; make    # Initial
autoreconf; ./configure; make                      # Weitere Builds
```

Quelle: eigene Darstellung

Erklärungen der einzelnen Funktionsaufrufe

(Entnommen aus den zugehörigen Manpages: *man <Befehl>*)

Befehl	**Beschreibung**
libtoolize	Erstellung unterstützender Dateien zur Nutzung von Bibliotheken innerhalb des Projekts.
aclocal	Generierung von Standard-Makros zur Späteren Verwendung für *autoconf*.
automake	Generierung der Templatedatei Makefile.in.
autoconf	Erzeugen des Konfigurationsskriptes (./configure).
./configure	Erzeugen der Makefile.
Make	Kompilieren der systemspezifischen Makefile.

Quelle: eigene Darstellung

Anlage F: Änderungen des dsImageServers an der Datei:

./src/Makefile

```
CFLAGS = -g -O2 -Wall `Magick++-config --cflags`
CPPFLAGS = `Magick++-config --cppflags`
CXXFLAGS = -g -O2 `pkg-config vipsCC-7.20 --cflags --libs` `Magick++-config --cxxflags` -Ipdflib/include
LDFLAGS = -L/usr/lib -Wl,-Bsymbolic-functions -L/usr/lib/X11 -Lpdflib/lib `Magick++-config --ldflags`
LIBS = -lnsl -lz -lm  -lmemcached -lfcgi  -ljpeg -ltiff -lz -lm /usr/lib/libxerces-c.so -lpdf `Magick++-config --libs`

dsImageServer_fcgi_SOURCES = \
        …
        PNGCompressor.h
        PNGCompressor.cc
        VIF.cc
        dsAbstractObject.h
        …                                   # Alle hinzugefügten neuen Klassen/Header

include ./$(DEPDIR)/VIF.Po                  # Alle hinzugefügten .Po Dateien
…

am_dsImageServer_fcgi_OBJECTS =
        PTL.$(OBJEXT)
        …                                   # Alle neuen zu linkenden .o Dateien
```

Quelle: eigene Darstellung

Anlage G: Zeitmessung mittels Profiler.h

```
#ifndef PROFILER_H
#define PROFILER_H


#include <sys/time.h>
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>

using namespace std;


class Profiler {
public:
	/**
	 * Start timer.
	 */
	void start()
	{
		gettimeofday(&_start, NULL);
	}

	/**
	 * Stop timer and save timeintervall with given message.
	 * @param string message
	 */
	void stop(string message)
	{
		gettimeofday(&_end, NULL);

		// Log intervall.
		ofstream outfile;
		double elapsedTime;

		elapsedTime = (_end.tv_sec - _start.tv_sec) * 1000.0;      // sec to ms
		elapsedTime += (_end.tv_usec - _start.tv_usec) / 1000.0;   // us to ms
```

```
                outfile.open ("/var/dsImageServer/debug/profiler.log", ios_base::app);
                outfile << elapsedTime << " ms ::: " << message << endl;

                outfile.close();
        }


        /**
         * Write a single line of information.
         */
        void writeLine(string message)
        {
                ofstream outfile;

                outfile.open ("/var/dsImageServer/debug/profiler.log", ios_base::app);
                outfile << message << endl;

                outfile.close();
        }


protected:
        timeval _start;
        timeval _end;
};


#endif
```

Quelle: eigene Darstellung

Literaturverzeichnis

[Jae02] Jaeger, T., Metzger A.: „Open Source Software – Rechtliche Rahmenbedingungen der Freien Software", 1. Auflage, München, Verlag C.H. Beck oHG, 2002

[Kre05] Kreuzer, M.: „Die praktische Relevanz von Mass Customization – Die individuellen Bedürfnisse des Kunden – oder der Kunde als Lemming?", 1. Auflage, Stuttgart, Haupt Verlag, 2005

[Lau04] Laurent, A. M. S.: „Understanding Open Source and Free Software Licensing", 1. Auflage, Sebastopol, O'Reilly Media, Inc., 2004

[Möh09] Möhrke, C.: „Besser PHP programmieren – Handbuch professioneller PHP-Techniken", 3. Auflage, Bonn, Galileo Press, 2009

[Pic11] Pichler, R., Roock, S.: „Agile Entwicklungspraktiken mit Scrum", 1. Auflage, Heidelberg, dpunkt.verlag GmbH, 2011

[Sne10] Sneed, H., Wolf. E., Heilmann, H.: „Softwaremigration in der Praxis – Übertragung alter Softwaresysteme in eine moderne Umgebung", 1. Auflage, Heidelberg, dpunkt.verlag GmbH, 2010

[Stö07] Stöhr, F.: „Die GNU Autotools – Leitfaden für die Softwaredistribution", 1. Auflage, Böblingen, C&L Computer und Literaturverlag, 2007

[Wes06] Westphal, F.: „Testgetriebene Entwicklung mit JUnit & FIT – Wie Software änderbar bleibt", 1. Auflage, Heidelberg, dpunkt.verlag GmbH, 2006

[Pil12] Pillay, R.: „IIPImage >> Server" , 2012, http://iipimage.sourceforge.net/documentation/server/, Abruf: 24.07.2012

[Pil12a] Pillay, R.: „IIPImage >> Download", 2012, http://iipimage.sourceforge.net/download/, Abruf: 24.07.2012

[Mac11] Mackiol, J.: „dsImageServer", dotSource GmbH, 2011, https://svn.dotsource.de/php/dotsource/dsImageServer/trunk/doc/documentation.xhtml, Abruf: 05.07.2012

[Moz12] „MPL 2.0 FAQ", http://www.mozilla.org/MPL/2.0/FAQ.html, Abruf: 05.07.2012

[Osi12] Open Source Initiative: „Open Source Initiative - OSI", 2012, http://www.opensource.org/, Abruf: 05.07.2012

[Wil12] Wilson, R.: „The Apache License (v2) – An Overview", OSS Watch, 2012, http://www.oss-watch.ac.uk/resources/apache2.xml, Abruf: 05.07.2012

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine Bachelorarbeit mit dem Thema:

   Erweiterung einer Imageservertechnologie am Beispiel des dotSource Produktgestalters auf Basis einer Open-Source-Software

   ohne fremde Hilfe angefertigt habe,
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und
3. dass ich meine Bachelorarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

_____ _____

Ort, Datum Unterschrift