TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Masterarbeit

Implementierung von OSS in der Produktion eines eTOD Systems

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines

Master of Science

unter der Leitung von

Univ.Prof. Mag.rer.nat. Dr.rer.nat. Georg Gartner
(E120 - Department für Geodäsie und Geoinformation, Bereich: Kartographie)

Fakultät für Mathematik und Geoinformation

von

Daniel Zamojski

e1025620 (E 033 221)

Eugen Bormann Gasse 2/2

1220 Wien

Wien, im August 2016

Vorname, Nachname

_____

_____

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Ich habe zur Kenntnis genommen, dass ich zur Drucklegung meiner Arbeit unter der Bezeichnung

Masterarbeit

nur mit Bewilligung der Prüfungskommission berechtigt bin.

Ich erkläre weiters unter Eid statt, dass ich meine Masterarbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen selbstständig ausgeführt habe und alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, genannt habe.

Weiters erkläre ich, dass ich dieses Masterarbeitsthema bisher weder im In- noch Ausland (einer Beurteilerin/einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe und dass diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.

Wien, im August 2016 Vorname,Nachname

_____ _____

Vorwort

Die Suche nach einem passenden Thema für eine Diplomarbeit ist meist nicht leicht. Doch Dank der Arbeit in der Frequentis AG konnte ich eine wählen, die nicht nur Spaß bei der Umsetzung gemacht hat, sondern auch verschiedene Fähigkeiten erfordert hat und mir somit erlaubte, mich in einem Bereich weiterzuentwickeln, in dem ich später arbeiten möchte. Daher hat die Arbeit für mich einen besonderen Wert.

Danksagung

Ich möchte mich in aller Form bei meinem Betreuer für die Unterstützung und Offenheit bei dieser Arbeit bedanken. Dank seiner Offenheit und Leitung konnte ich nicht nur meine Kreativität und mein Interesse für dieses Thema ausleben, sondern auch eine Arbeit hervorbringen auf die ich stolz sein kann. Des Weiteren möchte ich meinem derzeitigen Chef und Teamleiter der Firma Frequentis meinen Dank aussprechen, welcher stets dem Bombardement an Fragen standhielt und mich beim Projekt mit Optimismus unterstützte.

Abstract

This Master Thesis investigates the question of if and how the change from Closed Source Software (CSS) to Open Source Software (OSS) affects the costs and quality in an environment of a software industry production. An experiment will be done in a company where a team of developers produces an eTOD ("electronic terrain and obstacle database"), which represents a geoinformationsystem (GIS) to manage and visualize geodata. This experiment covers the whole life cycle of a product in the production, which consists the choice of GIS OSS components, the planing of costs and system architecture as well as the development of the GIS product. For a better understanding the topic will be introduced with the current stand of knowledge which can be find in the literature. The costs and the quality of the GIS product as the outcome of the experiment, will be compared with an pure CSS application. Furthermore the results of the experiment will be discussed and compared with the statements of the literature.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Open Source steht für den frei zugänglichen, modifizierbaren Quellcode einer Software (Open Source Initiative 2007). Definiert vor 30 Jahren, hat sich der Begriff über diese Zeit quer durch alle Softwarebereiche verbreitet, etabliert und wurde somit zu einem wichtigen Bestandteil der Softwareentwicklung. Angetrieben durch den Gedanken Programme unabhängig von Softwarekonzernen nutzen zu können, verbreitet sich die Idee des frei verfügbaren Quellcodes immer weiter (Kuan 2001). Non-Profit-Organisationen wurden gegründet und Dank dem ständigem Wachstum an Internetzugängen und somit Nutzern findet die Open Source Bewegung immer mehr Anhänger (Kuan 2001). Speziell im Bereich der Kartographie und Geoinformation finden sich Produkte wie zum Beispiel das Geodatenmanagementsystem GeoServer (http://geoserver.org/) oder das Kartierungsprogramm Open Layers (http://openlayers.org/), die im Laufe der Zeit zu großeren und stabilen Programmen weiterentwickelt wurden, deren Funktionen und Verwendungsmöglichkeiten durch die dahinterstehende Gemeinschaft stetig erweitert werden und somit weitere Anwendungsbereiche erschließen.

Das Potenzial der Geoinformationssystem-Open Source Software (GIS OSS) Produkte wurde bereits von Universitäten erkannt und in die Bildungspläne mit aufgenommen (Kotwani & Kalyani 2012) (Sakellariou 2016). So wird zum Beispiel im Studiengang der Geodäsie und Geoinformation an der Technischen Universität Wien die Nutzung von GIS OSS gelehrt. Ist GIS OSS einmal in die Lehrpläne integriert, ist der Einzug in die Softwareindustrie erleichtert. Dennoch werden in der Industrie für das Managen und Visualisieren von Geodaten meist kommerzielle Lösungen bevorzugt. Beispiele für solche Lösungen sind Programme von ESRI (http://www.esri.com/), welche Geodatenmanagementsysteme und Kartierungs-Anwendungen umfassen und Luciad (http://www.luciad.com/), welche ebenfalls eine breite Palette an Kartierungsanwendungen anbietet. Aufgrund der Unzugänglichkeit des Quellcodes für den Nutzer werden diese als Closed Source Software (CSS) bezeichnet. Der Grund für die Bevorzugung liegt oft an der Unklarheit über den Aufwand für die Einbindung der OSS Komponenten und Unsicherheiten in den vermuteten Kosten die durch die Wahl der richtigen GIS- OSS sowie für das Zusammenführen der einzelnen GIS OSS zu einem größeren System entstehen können. Denn Fehler sowie Inkompatibilität zwischen den GIS-OSS Komponenten stellen ohne notwendigem Fachwissen ein schwer kalkulierbares Risiko für die Plannung und Umsetzung eines Programms dar. Eben mit jenem Planungs und

Umsetzungs-isiko möchte ich mich befassen um Nutzern, die in Erwägung ziehen auf OSS Programme umzusteigen, ein Rüstzeug für die Abschätzung des Aufwands und der Unsicherheiten zu liefern. Ich sehe in diesem Thema eine hohe Relevanz, da aufgrund der steigenden Anzahl der End-Nutzer, welche von kommerziellen Programmen auf Open Source umsteigen, wie beispielsweise in den Bereichen der öffentlichen Administration (Kovacs *et al.* 2004) und Bildung (Tomazin & Gradisar 2009), OSS präsenter ist als je zuvor und damit auch der Druck auf Unternehmen steigt, dieser Veränderung zu folgen.

1.2 Hintergrund des Problems

Aus der vorhandenen Literatur sind einige Vor- und Nachteile bezogen auf Kosten und Qualität von OSS (Noyes n.d., Renner *et al.* 2005, Höst *et al.* 2011) gegenüber CSS (Rubens 2014, Karels 2003) für die Nutzung und Implementierung in Anwendungen ableitbar. Anhand der Aussagen in der Literatur wird meist OSS als die bessere Wahl in Bezug auf Qualität und Kosten angesehen (Kuan 2001). Auch über die Verwendung und Vermarktung von OSS finden sich diverse Arbeiten. Speziell für die Verwendung in der Softwareindustrie werden verschiedene Verwendungsarten und bereits genutzte Businessmodelle vorgestellt. Dies kann beispielsweise die Weiterentwicklung von OSS sein, um ein verbessertes Produkt auf den Markt zu bringen, kostenpflichtige Hilfestellungen und Unterstützung sowie die Implementierung in größere Projekte um eine Kostenersparnis zu erzielen (Bitzer 2004, Höst *et al.* 2011).

OSS Produkte gewinnen an Beliebtheit (Kuan 2001), sie überschwemmen allerdings nicht den Markt, sondern koexistieren und teilen sich die Nutzer (Bitzer 2004). Dennoch ist zu sagen, dass OSS Produkte als große Konkurrenz für proprietäre Projekte angesehen werden könnten, da diese den Marktpreis erheblich senken (Bessen 2005). Schließlich kann OSS laut (Bitzer 2004) eine günstige Variante für den Endkunden liefern.

Die Produktion von Programmen erfordert jedoch andere Anforderungen, wie Stabilität und Datensicherheit, als ein Nutzer der ein GIS OSS für private Zwecke verwendet. In diesem spielen andere Faktoren´, wie beispielsweise Datenschutz, Applikationssicherheit, Qualitäts- und Interoperabilitätsbestimmungen, Standards und Stabilität des Systems, eine Rolle. Die Fragen, die sich Entwickler und Projektverantwortliche stellen, ist ob auch bei der Erstellung einer GIS Anwendung für das Verwalten und Visualisieren von Geodaten mit einer Kostenreduktion und Qualitätssteigerung zu rechnen ist. Aus der Literatur können hierzu kaum Informationen gesammelt werden, da viel stärker auf den Aufbau einer Applikation,

basierend auf OSS, eingegangen wird (Xia *et al.* 2009) (Delipetrev *et al.* 2014a). Auch fehlen Informationen darüber, ob und mit welchen Gefahren bei der Implementierung von GIS OSS in ein größeres Projekt zu rechnen ist und des Weiteren wie diese gelöst oder umgangen werden können.

Mehr Informationen über die Verwendung von GIS OSS

Die Auflösung der oben genannten Unklarheiten würde helfen zu entscheiden ob ein Wechsel von GIS CSS zu GIS OSS für den jeweiligen Projektverantwortlichen in Betracht zu ziehen ist. Ohne ausreichende Informationen stellt der Wechsel von GIS CSS zu GIS OSS Produkten ein schwerlich kalkulierbares Risiko dar.

Unter dem Vorbehalt der ausreichenden Fachkenntnis, teile ich die Ansicht von (Bitzer 2004) und vertrete die Annahme, dass eine Kostenreduktion und Qualitätssteigerung auch für die Produktion, also von der Planung bis zum Abschluss der Entwicklung, von Geoinformationssystemen unter Zuhilfenahme von GIS OSS zu erzielen ist.

1.3 Ziel der Arbeit

Die vorliegende Arbeit beschäftigt sich mit der Untersuchung folgender Hypothese. Ein proprietäres eTOD („electronic Terrain and Obstacle Database“), welches für das Verwalten und Visualisieren von Geländedaten verwendet wird, kann mittels GIS OSS erstellt werden. Eine mit einer CSS Applikation vergleichbare Qualität der auf OSS basierenden Applikation ist mit Zeit- und Kostenersparnis erzielbar. Die Qualität setzt sich aus folgenden Kriterien zusammen: Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Instandhaltbarkeit (Andreou & Tziakouris 2007). Es wird ein Experiment durchgeführt, bei dem ein Team von 4 Entwickler ohne GIS OSS Erfahrungen ein eTOD planen und entwickeln sollen. Dabei werden GIS-OSS-Komponenten wie GeoServer (http://geoserver.org/), GDAL (http://www.gdal.org/) und OpenLayers (http://openlayers.org/) für die Entwicklung eines eTOD-Projektes verwendet werden

Die Summe der einzelnen Schritte umfasst den ganzen Entwicklungszyklus eines Programm in der Produktion, bestehend im Groben aus der Planung des Projektes, der Wahl der OSS-Komponenten, der Implementierung und Entwicklung, sowie dem Testen der resultierenden Geoinformationsapplikation. Das Experiment ist damit wiederholbar und würde zu gleichen Ergebnissen und Schlussfolgerungen führen, wenn der beschriebene Entwicklungszyklus eingehalten wird sowie Entwickler eingesetzt werden, die in OSS unerfahrenen sind.

Das Ziel ist nach dem Experiment die Aussagen aus der Literatur zu verifizieren unddamit einen relevanten und informativen Beitrag für Fachkräfte aus der Softwareindustrie zu liefern.

1.4 Inhalt

Zu Beginn werden in Kapitel 2 die Grundlagen und Theorie besprochen die aus der Literatur abgeleitet werden. Dabei wird zuerst in Kapitel 2.1 auf die Closed Source Software und anschließend auf die Open Source Software in Kapitel 2.2 eingegangen. Dabei wird auf Definition, Lizenzen und Vor- und Nachteile von OSS näher eingegangen. Im Kapitel 2.2.5 wird zusätzlich der Vergleich und die Wahl der passenden OSS-Komponenten diskutiert. Für das Experiment werden zum einen die benötigten Grundlagen zur Arbeitszeitabschätzung in der Softwareentwicklung und zum anderen die Indikatoren für die Bewertung der Qualität (Kapiteln 2.3.1 und 2.3.2) näher erläutert.

Den Hauptteil bilden die Kapitel Methodik (Kapitel 3) und Durchführung (Kapitel 4). Ersteres beschreibt im allgemeinen den ganzen Prozessablauf des im darauffolgenden Kapitel durchgeführten Experiments. Daher wird auch auf die Ziele und den Vorgang der Evaluierung eingegangen. Letzteres stellt das eigentliche Experiment aufgeteilt in einzelnen Schritten dar. Im Anschluss werden im Kapitel 5 (Resultate und Interpretationen) die Ergebnisse des Experiments diskutiert und mit den Aussagen der vorhanden Literatur verglichen. Zum Schluss findet sich in Kapitel 6 eine Zusammenfassung sowie in Kapitel 7 der Ausblick für die nähere Zukunft.

2 Grundlagen und Theorie

2.1 Closed Source Software (proprietäre Software)

2.1.1 Definition & Lizenzen

Closed Source CS bedeutet, dass der Quellcode eines Programms für den Nutzer weder verwendbar noch einsehbar ist. Die Idee dabei ist die Nutzung des Programms gegen ein Entgelt zu erlauben. Diese Strategie wird meist in proprietären Programmen angewandt. Hierzu ist eine rechtliche Absicherung notwendig. Zwar gilt das Urhebergesetz auch für Programme, jedoch ist der Hersteller nicht gleich der Urheber. Im Normalfall wird daher der Standardlizenzvertrag, auch als „End User Licence Agreement“ (EULA) bezeichnet,verwendet. Der Lizenzvertrag sagt aus

welche Nutzungsrechte dem Nutzer eingeräumt werden. Dieser beinhaltet auch ob und wie oft das Programm vervielfältigt werden darf, und wie viele Nutzer die jeweilige Lizenz verwenden dürfen. Über den Inhalt der Nutzungsrechte kann der Inhaber frei entscheiden. Allerdings sind meist mehrere Entwickler am Werk beteiligt und somit liegen die Nutzungsrechte nicht ausschließlich bei einer Person. Angesichts des frei wählbaren Inhaltes differieren die Verträge sehr stark. Daher werden lediglich zwei spezielle Lizenzarten für CSS vorgestellt, die neben dem Standardlizenzvertrag ebenfalls oft in Verwendung sind. Zum einen handelt es sich um die Shareware Lizenz. Diese wird für Testversionen von gebührenpflichtigen Programmen genutzt und erlaubt eine kostenlose Nutzung des Programms mit der Beschränkung auf eine bestimmte Dauer oder auf bestimmte Funktionen. Dies erlaubt dem Nutzer das Programm auszuprobieren bevor dieser sich zum Kauf entscheidet. Zum anderen ist es die Individuallizenz die meistens zum tragen kommt wenn ein Abnehmer einen Auftrag zur Programmierung oder Individualisierung aufgibt. Bei diesem Vertrag werden die Nutzungsbestimmungen mit dem Abnehmer ausgehandelt.

2.1.2 Vor- und Nachteile von proprietären Programmen

Im Folgenden wird auf die Vor- Nachteile eingegangen, die sich bei der Nutzung von proprietären Programmen gegenüber der Nutzung von OSS ergeben. Eine zusammenfassende Auflistung der beschriebenen Vor- und Nachteile kann in Tabelle 2.1.2.1 eingesehen werden.

Proprietäre Programme werden über eine Lizenz gegen ein Entgelt zur Verfügung gestellt. Das Zahlungsmodell variiert dabei in Abhängigkeit von Firma und Produkt. (Rubens 2014) sieht der Verwendung von OSS kritisch entgegen und liefert ein Liste von Vorteilen die in einem Vergleich mit OSS zu erwähnen sind. Einer davon ist, dass die Programme meist so entwickelt werden, dass der Nutzer keine besonderen Kenntnisse oder Ausbildung braucht wie das bei OSS der Fall ist. Damit wird möglicherweise auch keine speziell für die Betreuung des Programms geschulte Fachkraft gebraucht. Die Installation ist einfach, da der Kunde das Programm als binäre Datei erhält, die nicht konfiguriert und kompiliert werden muss, sondern gleich installiert werden kann (Karels 2003). Handelt es sich um ein fachspezifisches Programm, das einer breiten Kundschaft dient, so werden die Programme auch an der Universität gelehrt und genutzt. Das Programm bildet daher einen gewissen Standard. Des Weiteren ist eine Produkt-bietende Firma bestrebt ein gutes Produkt zu verkaufen um die Kunden zufrieden zu stellen und sie damit stärker an sich zu binden. Die meist 24/7 angebotene Kundenbetreuung sowie die Angebote von Einschulungen sind ebenfalls ein starkes Argument für den Kauf von proprietären Programmen. Hat der Kunde ein Problem, dass er selbst nicht lösen kann, hat er die

Möglichkeit sich über verschiedene Medien an die Kundenbetreuung zu wenden. Sollte es sich bei dem Problem um einen Fehler des Programms handeln, wird auch versucht dieses mit dem nächsten Update zu beheben. Die Dauer der Behebung hängt jedoch von den Ressourcen der Entwickler ab. In Bezug zur Kundenzufriedenheit wird auch eine ausführliche Dokumentation bereitgestellt. Die Weiterentwicklung ist ebenfalls ein wichtiger Punkt. Es werden meist regelmäßig Updates geliefert um das Programm an neue Browser oder Betriebssystem-Versionen anzupassen oder um es mit neuen Funktionen auszustatten.

Betrachtet man nun die Nachteile proprietärer Programme, so ist das Entgelt, das für die genutzten Lizenzen standardmäßig jährlich verlangt wird, ein offensichtliches. Vor allem wird die Gebühr für jedes Computergerät, auf dem das Programm läuft, abgegolten (Karels 2003). Folgt man der Studie von (Raghunathan *et al.* 2005) die einen Kostenvergleich zwischen dem proprietären Betriebssystem Windows und der Open Source Variante Linux aufzeigt, so wäre für einen Zeitraum von 5 Jahren Windows aufgrund der niedrigeren Personalkosten die kostengünstigere Variante. Abhängig von den Vergleichsfaktoren und den Produkten liefert die Literatur unterschiedliche Resultate (Kavanagh 2004a). Es ist zu sagen, dass es sich um eine erworbene Lizenz handelt und daher der Lizenzinhaber nicht Eigentümer des Programms ist, womit auch kein Recht zur Einsicht und Nutzung des Quellcodes des erworbenen lizenzierten Programms besteht.. Wie ebenfalls in (Optimus Information Inc. 2015) beschrieben, ist es damit nicht möglich das Programm und seine Funktionen abzuändern. Der Kunde hat lediglich die Möglichkeiten den Produktanbieter über Fehler oder über den Bedarf von bestimmten Funktionen zu informieren. Jegliche Adaptierung oder Veränderung des Produktes durch den Nutzer würde mit dem Verlust der Hilfestellung durch die Kundenbetreuung einhergeht. Vor allem bei recht neu auf den Markt gekommenen Produkten muss damit gerechnet werden, dass bei der Nutzung des Programms Fehler aufgedeckt werden, welche in kommenden Updates behoben werden. Die Programme werden zwar von der jeweiligen Firma getestet, diese kann aber aufgrund der Anzahl an Tester keine gleichwertige Testabdeckung erzielen wie bei OSS. Dennoch liegt die Haftung für das Produkt beim Hersteller (Karels 2003). Des weiteren ist sicherlich zu erwähnen, dass für Updates sowie auch für den Support zusätzliche Kosten anfallen können (Karels 2003). Es ist auch zu bedenken, dass wenn das proprietäre Produkt nicht mehr lukrativ erscheint, es vom Markt genommen und nicht mehr unterstützt wird. Es muss dann ein Alternative gesucht und in die in einer Firma vorhandenen Programme implementiert werden. Laut (Optimus Information Inc. 2015) ist dieser Wechsel für eine Firma, in der das Produkt bereits eingebunden ist, mit hohen Kosten verbunden und sollte daher ebenfalls bedacht werden.

Vorteile	Nachteile
wenig Wissen erforderlich	Lizenzkosten
Installation einfach	kein Zugriff auf Quellcode
etablierter Standard	nicht adaptierbares Programme
Kundensupport (meist 24/7)	meist kostenpflichtiger Support
angebotene Einschulungen	Gefahr für Einstellung des Supports
seltener Updates	geringe Interoperabilität
an neuste Betriebssysteme angepasst	
Haftung beim Produkthersteller	
gute Dokumentationen	

Tabelle 2.1.2.1: Vor und Nachteile bei der Verwendung von proprietären Programmen

2.2 Open Source Software

2.2.1 Entstehung und Entwicklung von Open Source

Es ist schwierig den tatsächlichen Beginn der Open Source Software Entwicklung festzulegen.Universitäten vertreten seit je die Prinzipien der Zusammenarbeit und Wissensteilung. In den 1960ern waren Akademiker und Forscher die Vorreiter der Entwicklung von Programmen. Da diese für jede Hardware speziell angepasst werden mussten, war eine Zusammenarbeit in Form von freien Quellcodes durchaus sinnvoll. Durch die Anfänge des Internets 1969 wurde diese auch etwas vereinfacht. Mit der fortlaufenden Entwicklung von Computern und Programmen bekamen auch immer mehr Menschen Zugang zu einem Gerät. Neben der Verfügbarkeit von freien Quellcodes begannen Unternehmen gegen Gebühren Programme zu veräußern. Durch den Wunsch weiterhin freien Quellcode zur Verfügung zu haben, entstanden Gruppen und weiters Gemeinschaften, die Quellcodeteile zusammenführten und darüber entschieden, wie dieser in Zukunft weiter entwickelt werden sollte. Die Idee schien der der heutigen Open Source Bewegung sehr ähnlich zu sein. Doch in der Literatur wird erst das veröffentlichte GNU Manifest sowie der Start des GNU Projektes im Jahre 1984 durch Richard Stallman als Geburt der Open Source Software Bewegung definiert (Kuan 2001). Als Grund für die Veröffentlichung des Manifests lieferte Stallman, dass das Copyright die Freiheit eines Nutzers eine Software zu verwenden und zu verändern beeinträchtigen würde. Ein paar Jahre später hatte er daher auch die „General Public License“ für sein Projekt beschrieben. Diese hat neben den geforderten Freiheiten auch das Copyleft, dass die Beibehaltung der Rechte bei Weitergabe sichert. Diese gilt auch bei Veränderung und Erweiterung des Programms. Somit war die Einführung des Copylefts ein wichtigen Schritt für die Beständigkeit von OSS, denn jegliche GPL lizenzierten Projekte mussten wieder mit derselben Lizenz veröffentlicht werden. Damit hat

Stallman der Bewegung nicht nur ein Gesicht verliehen, sondern auch auch den wichtigen Grundstein für Open Source gelegt. Heute gibt es bereits GPLv3, die die dritte Version dieser Lizenz repräsentiert.

2.2.2 Definition

Oft wird „Freie Software“ als Synonym für Open Source verwenden, allerdings ist zu sagen, dass das zwei eigenständige Bewegungen mit unterschiedlichen politischen Zielen sind, obgleich sie an denselben Projekten arbeiten können. Open Source beschreibt den Entwicklungsprozess und die Offenheit des Quellcodes. Freie Software dagegen möchte die grundsätzlichen Freiheiten bewahren, die sich aus Verändern, Verteilen, uneingeschränkter Nutzung und Verfügbarkeit des Quellcodes zusammensetzen.

Die Definition von Open Source laut (Open Source Initiative 2007) spiegelt deutlich das Konzept von Open Source wieder. Der Quellcode soll kostenlos und frei für jedermann, daher ungeachtet welcher Herkunft und Ausbildung, sein und für alle Zwecke und Plattformen ohne Einschränkungen ausführbar sein. Modifikationen und Arbeiten die auf den lizenzierten Quellcode basieren, müssen jedoch unter den gleichen Lizenzbedingungen veröffentlicht werden. Damit ist die freie Nutzung, Veränderbarkeit und Teilbarkeit des Quellcodes auch bei der weiteren Verbreitung gesichert. Die von Stallman eingeführte Anforderung, den Quellcode mit derselben Lizenz zu veröffentlichen, wird als „Copyleft“ bezeichnet (Link 2010). Die Definition erlaubt zwar den Verkauf von Software mit einer Open Source Lizenz, allerdings nur unter Einhaltung der oben genannten Bedingungen. Zurzeit gibt es bereits eine große Anzahl von verschiedensten Open Source Lizenzen. Diese erlauben durch abgeschwächten Copyleft auch proprietäre und des weiteren kommerzielle Softwareentwicklung ohne der Notwendigkeit der Veröffentlichung des Quellcodes. Informationen über die meist genutzten Open Source Lizenzen können dem Kapitel 2.2.3 entnommen werden.

Ein wichtiges Merkmal von OSS ist, dass die Entwickler gleichzeitig auch deren Nutzer sind. Dadurch fließen die Ideen jedes einzelnen Entwicklers in das Projekt ein. Dennoch werden jegliche Ideen geplant und strukturiert umgesetzt. wofür eine besondere Teamstruktur notwendig ist.

Wie in (Bahamdain 2015) beschrieben besteht eine OSS Gemeinschaft aus Kern-Entwicklern, Co-Entwicklern, Fehler-Reportern und den Nutzern. Jede dieser Gruppen hat seine Aufgaben und damit Nutzen für das OSS Projekt. Die Gruppen

werden dabei in einem Zwiebeldiagramm dargestellt. Die Kern-Entwickler stellen laut (Bahamdain 2015) die kleinste Gruppe dar. Sie treiben die Weiterentwicklung der Kern-Funktionen des OSS voran, tätigen die wichtigsten Entscheidungen und bestimmen damit die Pläne für das Produkt. Die Gruppe der Co-Entwickler ist größer und umfasst laut Abbildung 2.2.2.1 auch die Kern-Entwickler. Sie erstellen neue Funktionen für das OSS Produkt oder beheben etwaige Fehler. Die Fehler-Reporter, welche unter anderem auch die vorher genannten Gruppen umfassen, testen das Produkt auf verschiedenen Plattformen und melden jegliche Fehlverhalten des Produkts weiter. Die letzte Gruppe und auch die größte ist die der Nutzer. Sie umfasst die anderen Gruppen, was die getätigte Aussage bestätigt, dass die Entwickler gleichzeitig auch Nutzer sind.

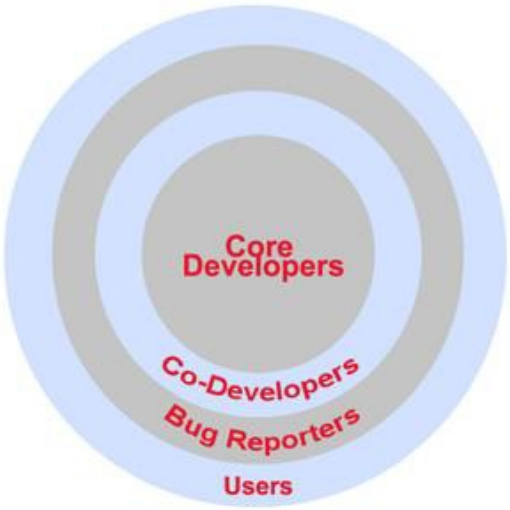

Abbildung 2.2.2.1: Zwiebelmodell der Gemeinschaft eines OSS Produktes (Bahamdain 2015)

2.2.3 OSS Lizenzen

Es gibt eine große Anzahl verschiedener Open Source Lizenzen, die auf die im letzten Kapitel 2.2.2 angeführten Definition von OSS abgestimmt sind. Die bekanntesten sind dabei GPL, LGPL, BSD, MIT und Apache (Kavanagh 2004b) wobei knapp 70% der Open Source Projekte die von der Free Software Foundation (FSF) veröffentlichte General Public License (GPL) verwenden. Diese entspricht der von Stallmans beschriebenen Lizenz für das GNU Projekt. 2007 erschien die dritte Version, welche als GPLv3 bezeichnet wird. Nutzer eines Projektes mit implementierten GPL Programmen haben aufgrund der Einschränkungen durch das Copyleft nur drei Optionen. Das Projekt wird wieder unter denselben Lizenzbedingungen veröffentlicht, der Open Source und proprietäre Quellcode klar von einander getrennt gehalten oder das Projekt wird einfach nicht veröffentlicht (Link 2010).

Laut (Kavanagh 2004b) verwenden 6% der Projekte die Lesser General Public License (LGPL). Diese erlaubt durch ein schwächeres Copyleft die Implementierung von LGPL Programmen ohne den Quellcode veröffentlichen zu müssen. Dies ist beispielsweise der Fall wenn die Software als Service angeboten wird oder wenn die verwendeten LGPL Komponenten nur verlinkt werden (Link 2010).

Lizenzen wie Berkeley Software Distribution (BSD), Apache und die aus Massachusetts Institute of Technology stammende Lizenz MIT werden von 5 % und weniger aller Open Source Projekte verwendet (Kavanagh 2004b). Sie bieten aufgrund des fehlenden Copylefts geringere Einschränkungen. Diese erlauben auch eine Veröffentlichung unter proprietären Lizenzen ohne der Angabe jeglicher Modifikationen oder gar der verwendeten Komponenten (Link 2010).

In der Literatur wird auch öfters von reziproken und nicht reziproken Lizenzen gesprochen. Ersteres umfasst die Lizenzen mit stärkerem Copyleft wie das bei GPL der Fall ist. Nicht reziproke Lizenzen sind im Gegenzug die mit einem schwächerem Copyleft wie die oben genannten LGPL, BSD, MIT und Apache Lizenzen (Kavanagh 2004b) welche für kommerzielle Zwecke einen höheren Nutzen versprechen.

Detailierte Beschreibung für die genannten Lizenzen finden sich auf der Webseite der Open Source Initiative (Open Source Initiative 2007).

2.2.4 Vor- und Nachteile von OS Programmen

In diesem Kapitel wird auf die Vor- Nachteile eingegangen, die sich bei der Nutzung von OSS gegenüber der Nutzung von proprietären Programmen ergeben. Eine zusammenfassende Auflistung der beschriebenen Vor- und Nachteile kann in Tabelle 2.2.4.1 eingesehen werden.

Aus der Definition von Open Source Programmen, die im vorangegangenen Kapitel 2.2.2 dargelegt wurden, geht hervor, dass Entwickler eines OSS gleichzeitig deren Nutzer sind. Dies hat zum Vorteil, dass sie das Wissen aus dem speziellen Fachgebiet mitbringen und daher genau wissen, was die Nutzer aus dem Bereich benötigen oder welche Probleme das Programm lösen muss. Die Nutzung der OSS ist kostenlos was einen nennenswerten Vorteil für Privatnutzer und Firmen gegenüber kommerziellen Programmen darstellt. Damit ist es auch möglich die jeweilige OSS Komponente auszuprobieren ohne das dadurch Anschaffungskosten

entstehen. Es muss jedoch für die Kompilierung des Quellcodes, Installation und Konfiguration sowie für das Erlernen der vom Programm bereit gestellten Funktionen Zeit investiert werden, womit für eine Firma Kosten entstehen können. Bei Problemen oder bei zur Verfügung „Updates“ ist ein erneutes Kompilieren, Installieren und Konfigurieren notwendig (Karels 2003). Dennoch ist zu sagen, dass im Gegensatz zu kommerziellen Programmen diese Kosten nur einmalig zu Beginn entstehen und daher keine zusätzlichen regelmäßigen Abgaben verursacht werden. Neben der geringen Kosten punktet OSS, wie in (Noyes n.d.) beschrieben, auch mit anderen Vorteilen. Da der Quellcode im Gegensatz zu kommerziellen Programmen verfügbar ist, besteht die Möglichkeit das Programm nach Wunsch zu verändern. Somit können fehlende Funktionen einfach selbst erstellt und hinzugefügt werden. Das OSS Produkt erfüllt sehr gut die Anforderungen seiner Nutzer, da die Entwickler gleichzeitig auch Nutzer sind. Somit entspricht das Produkt dem, was die Nutzer sich erwarten oder benötigen. Das führt auch zu einer besseren Interoperabilität. Denn während bei kommerziellen Programmen versucht wird ein eigenes internes Format zu nutzen um den Nutzer stärker an sich zu binden, versucht man bei OSS Entwicklung möglichst viele Formate zu unterstützen und auf bekannte Standards zu setzen. Das wiederum erleichtert die Verbindung und Kommunikation mit anderen Systemen. Dokumentation und Weiterentwicklung des jeweiligen Programms hängen jedoch stark von der Gemeinschaft des OSS ab. Je größer die Gemeinschaft desto schneller werden Fehler gefunden und auch behoben, Übersetzungen für Anleitungen und Hilfe werden schneller fertiggestellt und das Programm wird durch die Weiterentwicklung mit mehr Funktionen ausgestattet. Laut (Renner *et al.* 2005) sind OSS Produkte auch in Bezug auf Sicherheit und Qualität besser als kommerzielle Programme. Da viele Entwickler am Quellcode arbeiten, fällt ein beabsichtigter oder unbeabsichtigter Sicherheitsmangel schneller auf und kann behoben werden, was einen positiven Einfluss auf die Qualität hat.

Diese Gemeinschaftsabhängigkeit birgt aber auch ein paar Probleme. Angesichts dessen, dass meist viele Entwickler an dem Quellcode des OS Produkts beteiligt sind, werden auch oft neue Versionen veröffentlicht. Zwar muss der Nutzer das Programm nicht mit neuen Versionen aktualisieren aber sollte er dies doch tun, ist er meist gezwungen alle Erweiterungen ebenfalls auf die neueste Version zu aktualisieren. Auch besteht keine Garantie, dass regelmäßig neue Versionen veröffentlicht werden. Wird am Quellcode etwas geändert aber von der Gemeinschaft nicht angenommen, entstehen auch zusätzliche Kosten durch das Zusammenführen der eigenen Änderungen mit der neuen Version (Höst *et al.* 2011). Des Weiteren gibt es keine Kundenbetreuung bei welcher man anrufen kann. Ist die Gemeinschaft jedoch groß genug, besteht die Möglichkeit über diverse Foren Hilfe zu erhalten. Doch auch hier gibt es keine Garantie diese zu bekommen. (Bahamdain 2015) erwähnt als Nachteil die schlechte oder unvollständige Dokumentation aber auch das schlechte Design. Dies ist nicht weiter verwunderlich, da die Entwicklung schnell

vorangetrieben wird, da neue Funktionen eher gebraucht werden. Vorhandene Dokumentationen werden daher selten auf den neuesten Stand gebracht. Das bedeutet jedoch, dass der Nutzer verstärkt auf Probleme stoßen wird, die er selbst zu bewältigen hat. Ein gewisses Grad an Wissen oder Erfahrung ist daher unabdingbar. Laut (Raghunathan *et al.* 2005) können nach einem längerem Zeitraum die Kosten bei der Nutzung von OSS aufgrund von Personalkosten höher sein als bei der Nutzung von kommerziellen Programmen. Bei einem Wechsel von Mitarbeiter muss daher auch die Einschulung mitberechnet werden.

Vorteile	Nachteile
auf Nutzer vom Fachbereich zugeschnitten	steile Lernkurve
kostenlos erhältlich	keine Garantie auf Updates
Zugriff auf Quellcode	niemand für Programm haftbar
Programm adaptierbar	unvollständige Dokumentationen
schnelle Fehlerbehebung	Produktentwicklung abhängig
viele Funktionen	Von Gemeinschaft
hohe Stabilität	
hohe Interoperabilität	

Tabelle 2.2.4.1: Vor- und Nachteile bei der Verwendung von OSS

2.2.5 Vergleich und Wahl von OSS

Bevor für ein bestimmtes OSS Produkt entschieden wird und für die Implementierung in ein Programm eingesetzt werden soll, sollten jene Produkte die für eine Implementierung in Frage kommen, miteinander verglichen werden. Der Grund ist, dass ein bestehendes Programm jederzeit von einem anderen abgelöst werden kann wobei dies stark von der dahinterstehenden Gemeinschaft abhängt. Es ist von entscheidender Bedeutung ein OSS zu implementieren oder zu nutzen, das voraussichtlich die nächsten Jahre unterstützt und weiterentwickelt wird. Somit kann sichergestellt werden, dass bei Auftreten von Problemen die Gemeinschaft an einer Lösung arbeiten wird. Je größer die Gemeinschaft desto schneller werden auch Fehlverhalten des Programms gefunden und des Weiteren behoben.

Zur Vergleichbarkeit der einzelnen OSS Produkte müssen Metriken, also Maße zur Bewertung, eingeführt werden. Die Arbeiten von (Jansen 2014, Rudzki *et al.* 2009, Teixeira *et al.* 2015) beschäftigen sich mit diesem Thema und stellen solche anwendbare Metriken vor. Während (Teixeira *et al.* 2015) eine Metrik definiert, die speziell auf die Wahl von OS Bibliotheken zugeschnitten ist, liefern (Rudzki *et al.* 2009, Jansen 2014) einen allgemeineren Ansatz, welcher die Möglichkeit bietet auch GIS OSS Produkte mit Hilfe einer Metrik zu vergleichen. (Rudzki *et al.* 2009) und (Jansen 2014) verwenden sehr ähnliche Kriterien, wobei ersterer diese einfach

auflistet und letzterer sie in Haupteigenschaften gliedert, was die Auswertung vereinfacht. Dabei werden als die drei Haupteigenschaften Produktivität, Robustheit und Nischenkreation von (Jansen 2014) definiert. Diese Bewertungskriterien bilden die benannte Metrik und wie folgt definiert:

- Produktivität ist als Beitrag des OSS Produkts zu jenem Ökosystems zu verstehen, in dem sich das Produkt befindet.

- Die Robustheit gibt Auskunft darüber wie sich das Ökosystem oder das Projekt bei Änderungen verhält und umfasst auch das Zusammenspiel mit anderen Produkten im Ökosystem.

- Nischenkreation ist die Möglichkeit das Projekt in unterschiedlichem Kontext zu verwenden oder neu entdeckte Nischen abzudecken.

Unter Verwendung dieser Metriken würde eine gute Beurteilung bedeuten, dass das OSS Produkt eine Neigung zu Wachstum und Langlebigkeit zeigt. Dies würde bedeuten, dass das OSS Produkt voraussichtlich über einen längeren Zeitraum weiterentwickelt und unterstützt sowie die Gemeinschaft weiter wachsen wird. Es könnte gesagt werden, dass das OSS Produkt eine gute Wahl für eine Implementierung wäre.

Als Kriterien zur Beurteilung der vorgestellten Metriken schlägt (Jansen 2014) einige Möglichkeiten diese durch Zahlen zu festigen, welche im Folgenden beschrieben werden und in Tabelle 2.2.5.1 strukturiert dargestellt sind.

- Aufschluss über die Produktivität gibt die Anzahl der geöffneten Foren-Themen,Einträge und organisierten Veranstaltungen sowie auch eine Untersuchung, wie oft das Produkt heruntergeladen wird. Die Anzahl der Partnerschaften, seien es Firmen oder Organisationen, aber auch die Anzahl an E-Mails, Fehler-Reporte und Code-Änderungen welche in einem gewissen Zeitraum erfasst werden liefern ebenfalls Informationen zur Produktivität.
- Die Robustheit kann daran gemessen werden wie gut das OSS Projekt mit anderen Projekten verbunden ist. Enge Zusammenarbeit, Kooperationen oder gegenseitige Ergänzungen fördern die Robustheit des Projektes. Auch hier lassen sich Zahlen liefern. Die Anzahl der Partnerschaften und Kooperationen könne recherchiert werden. (Jansen 2014) schlägt des weiteren vor die Anzahl von diesem Projekt anhängige aktive Projekte, Bewertungen, Anzahl der Bewertungen sowie Suchstatistiken für das jeweilige Projekt in Betracht zu

ziehen. Die Anzahl der Seitenaufrufe und der eingetragenen Follower können ebenfalls gute Indikatoren sein.

- Nischenkreation lässt sich schwieriger beziffern, allerdings kann recherchiert werden in welchem Kontext des Produkt verwendet wird und in wie vielen Sprachen die Anleitung und Hilfe übersetzt wurde. Oft gibt es bereits Diagramme, die sämtliche Fachgebiete zeigen die das Produkt nutzt. Eine stärkere Verbreitung über unterschiedliche Themen und Bereiche hinweg ist als positiv zu bewerten.

Produktivität	Robustheit	Nischenkreation
- Anzahl der geöffneten Forenthemen - organisierte Veranstaltungen (Seminare, Workshops,...) - Anzahl der Downloads der OSS - Anzahl der Partnerschaften (Firmen, Organisationen,...) - Anzahl der veröffentlichten E-Mails - Anzahl der Fehler-Reporte - Anzahl der Codeänderungen	- Anzahl der Kooperationspartner - Bewertungen - Anzahl der abhängigen Projektebene - Suchstatistiken - Anzahl der Seitenaufrufe - Anzahl der Follower in sozialen Medien	- Anzahl der unterstützten Sprachen - Verbreiterung über unterschiedliche Fachgebiete - Verwendung von unterschiedlichen Technologien

Tabelle 2.2.5.1: Kriterien zur Beurteilung der Metriken von (Jansen 2014)

Für die Erhebung der Indikatoren gibt es im Web kostenfreie Möglichkeiten. (Teixeira *et al.* 2015) beschäftigt sich mit der Wahl eines OS Frameworks und liefert Beispiele aus der Praxis. Dabei wird die Anzahl der Twitter-Follower, StackOverflow Follower und Nutzermails des jeweiligen OS Frameworks verglichen. Twitter als soziales Medium und StackOverflow als Seite für Fragen und Antworten für Entwickler besitzen eine hohe Anzahl an Nutzern. Eine Erhebung der Zahlen kann daher Aufschluss über die Größe und Aktivität der Gemeinschaft des OS Produktes liefern. Dennoch spricht (Teixeira *et al.* 2015) eine Warnung aus, da nicht jedes Forum die OS Produkte gleich stark repräsentiert. Eine Möglichkeit dem entgegen zu wirken wäre die Miteinbeziehung von mehreren Foren. (Jansen 2014) verwendet SourceForge, Github, BitBucket, Tigris sowie für die Erhebung von Suchstatistiken

für eine bestimmte Periode Google Trends. Github verbindet Projektbezogene Foren mit Hilfsmitteln für die Veröffentlichung und Teilung von Code und Produkten und erfuhr in den letzten Jahren einen hohen Bekanntheitsgrad. Github bietet eine eigene API für etwaige Abfragen der besprochenen Zahlen. SourceForge bietet sogar eine herunterladbare Datenbank. OSS Produkte die bereits eine große Gemeinschaft besitzen, wie das bei Frameworks der Fall ist, erlauben auch einen Vergleich mit der Anzahl von Stellenangeboten in denen das OSS Produkt erwähnt wird. (Teixeira *et al.* 2015) beweist dies unter der Verwendung von Seiten wie Dice, Indeed und Linkedin. Einerseits liefert dies eine gute Schätzung für den Bekanntheitsgrad, andererseits ist es ein gutes Anzeichen für eine weiterwachsende Gemeinschaft des OSS Produktes. Beides deutet auf ein langlebiges OSS Produkt hin. Denn während Firmen durch interne Entwicklung auch die Weiterentwicklung des OSS Produktes vorantreiben, beschäftigen sich Privatpersonen vermehrt mit dem OSS Produkt um am Arbeitsmarkt mit dem Wissen punkten zu können. Beschäftigt ein Unternehmen Mitarbeiter die bereits an OSS Projekten mitentwickelt haben oder bereits Erfahrung mit bestimmten Produkten sammeln konnten, können auch diese befragt werden um die richtige OSS Wahl zu treffen (Anh *et al.* 2012).

Aus der Sicht einer Firma sollten auch immer wieder Untersuchungen durchgeführt werden um festzustellen ob das jeweilige OSS auch in absehbarer Zukunft weiterentwickelt wird. Ist das nicht der Fall, können dann noch rechtzeitig die notwendigen Maßnahmen eingeleitet werden. Bei den durchgeführten Untersuchungen sollten die Ergebnisse festgehalten werden um diese in der Zukunft wiederverwerten zu können. Denn die Anforderungen an ein zu entwickelndes Produkt, in das die OSS Komponente implementiert wird, können sich durch die Wünsche des Kunden ändern, wodurch ein Wechsel zu einer anderen OSS Komponente notwendig ist (Rudzki *et al.* 2009).

2.2.6 GIS OSS

In dieser Arbeit wird speziell auf GIS OSS eingegangen, weshalb im Folgenden auch auf diese etwas näher eingegangen wird.

Open Source fand bereits Einzug in der Kartographie und Geoinformation und wird stark von Universitäten und Forschungsinstituten gefördert (Maurya *et al.* 2015). Eine Liste der bekanntesten GIS OSS findet sich in der Arbeit von (Maurya *et al.* 2015). Eine wichtige Rolle spielt die im Jahre 2006 gegründete Open Source Geospatial Foundation (OSGeo), welche eine gemeinnützige Organisation darstellt, die zum einen rechtliche, finanzielle und organisatorische Unterstützung für OSS Projekte leistet und zum anderen selbst Projekte betreut, wie GDAL, GeoServer, MapServer

und Open Layers (Wisam & Mohammed n.d.). Außerdem stärkt OSGeo den Zusammenhalt der GIS OSS Gemeinschaft und verbindet Entwickler von verschiedenen GIS OSS Projekten indem die OSGeo beispielsweise die FOSS4G Konferenz veranstaltet und das OSGeo Journal publiziert. Die Veranstaltungen werden meist mit dem Kooperationspartner Open Geospatial Consortium (OGC n.d.) durchgeführt, welche eine in 1994 gegründete gemeinnützige Organisation mit mehr als 530 Mitgliedern, bestehend aus Unternehmen, Universitäten und Behörden, dargestellt. OGC setzt Standards im GIS Bereich für eine bessere Interoperabilität zwischen GIS Projekten (Wisam & Mohammed n.d.). Sowohl OSGeo als auch OGC fördern eine gute Zusammenarbeit zwischen den einzelnen GIS OSS Projekten und stärken somit die Nachhaltigkeit des GIS OSS Ökosystems.

Wird das Wachstum von Projekten im GIS Bereich über das letzte Jahrzehnt betrachtet, so zeigt diese sowohl in OSS als auch in proprietären Programmen einen starken Anstieg (Maurya *et al.* 2015). Die GIS Projekte basieren meist auf eine Geodateninfrastruktur (GDI), welche das Verwalten, Bearbeiten, Darstellen und Analysieren von Geodaten umfasst und besteht aus vielen Programmkomponenten, wie beispielsweise Geodatenbanken, Kartierungsdienste und Bearbeitungstools (Steiniger & Hunter 2012). Die Forderung der GIS Nutzer nach höhere Interoperabilität und das Aufkommen neuer Standards, wie die von OGC (OGC n.d.), setzen höhere Anforderungen an GDIs, die durch die große und aktive GIS OSS Gemeinschaft sowie der Unterstützung durch OSGeo umgesetzt und unterstützt wird (Wisam & Mohammed n.d.), sodass GIS OSS neben proprietären Programm in nächster Zukunft erhalten bleibt. Damit steht dem privaten Nutzer und Unternehmen frei zwischen proprietären und OSS Produkten zu wählen, wobei die Vor- und Nachteile, die in Kapitel 2.1.2 und 2.2.4 beschrieben werden, zur Hilfe genommen werden können.

2.3 Vergleich zwischen OSS und CSS

Um einen Vergleich zwischen OSS und CSS zu ermöglichen, sind einige Vergleichsparameter notwendig. Im Folgenden wird daher in Kapitel 2.3.1 auf die Aufwandsabschätzung und in Kapitel 2.3.2 auf die Programm-Bewertungs Indikatoren näher eingegangen.

2.3.1 Aufwandsabschätzung

Studien zeigen, dass Software Projekte in den meisten Fällen im Aufwand unterschätzt werden (Basten & Sunyaev 2011), verursacht kann dies beispielsweise durch ändernde Kundenwünsche, Komplexität des Projektes oder durch den Prozess des Schätzens. Dies führt zu hohen Kosten und geringerer Qualität des Produktes.

Auch bei einer Überschätzung werden Ressourcen verschwendet. Beides möchte man verhindern, allerdings ist der Druck ein Projekt so schnell wie möglich abzuschließen, meistens sehr hoch. Schließlich wählt ein Kunde meist das günstigste Produkt (Jorgensen 2014).

Um eine gute Abschätzung abgegeben zu können, sollten daher eine Metrik herangezogen werden die nicht nur präzise ist, sondern auch dem jeweiligen Projektkontext entspricht und möglichst einfach anzuwenden ist, damit diese tatsächlich verwendet wird (Rammage *et al.* 2010). Bevor mit der Schätzung begonnen werden kann, sollten die Anforderungen für das resultierende Produkt präzise definiert und schriftlich festgehalten werden. Die Anforderungen können sich während der Produktion beispielsweise durch die Hinzunahme von weiteren Kundenwünschen ändern (Khatibi Bardsiri *et al.* 2012). Eine Verfolgung der Änderungen erlaubt ein Controlling mit dessen Hilfe der geschätzte Aufwand und des weiteren die zur Verfügung stehenden Ressourcen stetig angepasst werden können. Damit können die Unsicherheiten der Schätzung verringert werden.

Eine Methode zur Abschätzung des Arbeitsaufwandes eines Programms ist die „Schätzung durch Fachexperten“, bei der Experten aus dem jeweils benötigten Fachbereich gebeten werden eine Schätzung abzugeben, da diese Erfahrung für die zu schätzenden Aufgaben besitzen (Jorgensen 2014). Der Vorteil ist, dass diese Methode auch anwendbar ist, wenn keine historische Daten zur Verfügung stehen. Doch auch hier sind die Anforderungen für das zu schätzende Programm präzise zu definieren damit der Experte eine möglichst genau Schätzung abgeben kann. Bei dieser Art der Schätzung gibt es zwei Möglichkeiten die Berechnung durch zu führen. Die eine Möglichkeit ist ein ähnliches und bereits abgeschlossenes Projekt aus den historischen Daten heranzuziehen und in Abhängigkeit vom Grad der Ähnlichkeit zum dem Projekt, eine Schätzung abzugeben (Jorgensen 2014) (Rammage *et al.* 2010). Die andere Möglichkeit ist das Projekt in kleinere Aufgabenpakete zu separieren und anschließend diese schätzen. Die Praxis ergibt, dass je kleiner die Einheiten, desto genauer kann die Schätzung abgegeben werden. Die Summe der Aufwände aller Einheiten ergibt den Aufwand für das ganze Projekt (Rammage *et al.* 2010) (Basten & Sunyaev 2011). Ändern sich einzelne Pakete während der Produktion so kann direkt auch die Änderung für das ganze Projekt ermittelt werden.

Der Aufwand berechnet sich wie folgt:

$$Effort = HistoricalProductivityFactor * Predictor$$

Abbildung 2.3.1.1: Formel für Aufwandsschätzung nach Experten

(Rammage et al. 2010)

Wie anhand der Formel in Abbildung 2.3.1.1 zu sehen ist, fließen nur zwei Werte ein. Der „*HistoricalProductivityFactor“* wird aus den historische Daten berechnet und hängt von der Leistung des Teams sowie von zusätzlichen Aufwänden ab, die nicht vom Projekt abhängig sind, wie Meetings und Weiterbildungen. Nach Abschluss eines Projektes sollte dieser Wert für das nächste angepasst werden (Rammage *et al.* 2010) (Fairley 2002). Qualität und Komplexität sind schwierig zu messen, haben aber Einfluss auf die Schätzung „*Predictor“* und auf den Produktivitätsfaktor „*HistoricalProductivityFactor“*. Daher fließen diese über die beiden Parameter in die Gleichung mit ein.

Es ist zu sagen, dass bei der Befragung der Experten bei der „Schätzung durch Fachexperten“ keine Frist für den Abschluss des Projektes angegeben werden sollte. Dadurch, dass dem Experten eine Frist genannt wird, wird dieser die Schätzung zu niedrig ansetzen, damit sich das Projekt innerhalb der vorgegebenen Frist ausgeht. Die Folge ist jedoch ein hoher Druck auf die Entwickler und auf das Management (Rammage *et al.* 2010) (Jorgensen 2014). Sollte ein unerwartetes Problem auftreten gibt es auch keinen zeitlichen Puffer. Obwohl dieses Problem bekannt ist, tritt diese Art der Schätzung sehr oft in der Industrie auf.

Um eine gewisse Schwankungsbreite zu erhalten können die Unsicherheiten dieser Methode wie folgt verringert werden. Für die Schätzung kann ein ganzes Team von Experten mit unterschiedlichen Spezialisierungen und Erfahrungsschatz miteinbezogen werden. Des weiteren kann von jedem Experten, wie in der Formel von Abbildung 2.3.1.2 zu sehen, eine Schätzung für den besten („*BestCase“)*, schlechtesten („*WorstCase“)* und wahrscheinlichsten Fall („*MostLikelyCase“)* abgegeben werden. Aufgrund der sich ergebenden Spannweite entfällt der Produktionsfaktor. Die Projektleitung kann dann neben der besseren Schätzung *ExpectedCase* auch eine Standardabweichung *EstStdDev* angeben (Khatibi Bardsiri *et al.* 2012).

$$ExpectedCase = [\, BestCase + (4 * MostLikelyCase) + WorstCase \,] / 6$$
$$EstStdDev = [\, WorstCase - BestCase \,] / 6$$

Abbildung 2.3.1.2: Formel für Aufwandsschätzung nach Experten mit Standardabweichung (Rammage et al. 2010)

Eine erweiterte Methode, die von (Basten & Sunyaev 2011) vorgeschlagen wird, ist die „Delphi Methode“. Dabei wird die Schätzung in mehreren Runden durchgeführt. Die Experten geben meist anonym eine Schätzung mit Begründung ab und erfahren

nach jeder Runde das Ergebnis. Anschließend können diese in der nächsten Runde ihre Schätzung anpassen.

Neben der Expertenmethode gibt es auch eine größere Anzahl an automatischen Verfahren. Ein paar Methoden wie beispielsweise „Analogy Based Estimation (ABE)“, „Artificial Neural Network (ANN)“ und „Clustering“ stellt (Khatibi Bardsiri *et al.* 2012) vor und vergleicht diese miteinander. Diese Methoden berücksichtigen jedoch nich Unternehmens-, Team- und Fachspezifische Einflussfaktoren. Praktische Methoden, wie die Schätzung nach Fachexperten, sind daher meist die bessere Wahl, da sie sich dem Kontext der Schätzung besser angepasst werden können. Da sich in der Praxis der Programmentwicklung die praktischen Methoden weitgehend bewährt haben (Rammage *et al.* 2010) (Jorgensen 2014), wird auf die automatischen Methoden nicht näher eingegangen.

2.3.2 Programm-Bewertungsindikatoren

Für die Bewertung der Qualität einer Applikation sind Indikatoren festzulegen. Diese können zwischen verschiedenen Unternehmen oder gar Entwicklungsteams unterschiedlich festgelegt sein. Die Erfüllung der Kriterien erfordert ein aktives Handeln und ist ein wichtiger Bestandteil des Entwicklungsprozesses. Daher wird dies vom Projektmanagement in die Kostenrechnung miteinbezogen. Die Literatur zeigt, dass eine geringere Qualität der Applikation zu höheren Kosten führt (Boehm *et al.* 1976). Dies ist nicht weiter erstaunlich, da Wartungsarbeiten oder Fehlerbehebungen Personalkosten verursachen, die deutlich höher ausfallen als für andere Bereiche wie Hardwareankauf.

Wird nun nach Qualitätskriterien in der Literatur gesucht, finden sich sehr ähnliche Qualitätskriterienbäume, deren Auswertung mit unterschiedliche Herangehensweisen verbunden sind (Andreou & Tziakouris 2007) (Boehm *et al.* 1976) (Koçak *et al.* n.d.) (Li *et al.* 2010).

Die Arbeit von (Andreou & Tziakouris 2007) liefert eine Liste an Applikationseigenschaften und Subeigenschaften die als Kriterien für die Qualitätsbewertung dienen und sind in der Tabelle 2.3.2.1 angegeben. Die Subeigenschaften werden noch weiter in Attribute getrennt die aufgrund der Menge nicht weiter angeführt werden. Jedem Attribut ist dabei ein Gewicht zugewiesen, das die Auswirkung auf die Gesamtqualität widerspiegelt. Außerdem können die Attribute unterschiedliche Metriken wie Verhältnisse, Listen, Summen oder auch boolesche Werte aufweisen. So könnte beispielsweise für die Interoperabilität eine Liste von

unterstützten Betriebssystemen und Hardware Komponenten verwendet werden. Für die Service Stabilität wäre dagegen die Anzahl der auftretenden Fehler in einem spezifischen Zeitraum eine verwendbarer Wert.

Eigenschaft	Subeigenschaft
Funktionalität	Interoperabilität, Sicherheit, Vollständigkeit
Zuverlässigkeit	Service Stabilität, Ergebnismenge
Benutzbarkeit	Erlernbarkeit, Hilfsmittel, Bedienbarkeit, Erreichbarkeit
Effizienz	Reaktionszeit, System Overhead
Instandhaltbarkeit	Veränderlichkeit, Testbarkeit, Anpassbarkeit

Tabelle 2.3.2.1: Eigenschaften zur Bewertung der Qualität eines Programms (Andreou & Tziakouris 2007)

Die ältere Arbeit von (Boehm *et al.* 1976) liefert einen ähnlichen Baum mit den Eigenschaften Portabilität, Zuverlässigkeit, Effizienz, Menschliche Interaktion, Testbarkeit, Verständlichkeit, Veränderbarkeit. Die Subeigenschaften sind jedoch untereinander verzweigt wodurch eine Subeigenschaft mehreren Eigenschaften zugeordnet werden kann. Dieser Ansatz liefert ein komplexes Gebilde das nicht zu trennen ist und somit die getrennte Bewertung der einzelnen Eigenschaften unmöglich macht.

Da in einigen Softwareunternehmen der Einfluss auf die Umwelt und Nachhaltigkeit in der Qualitätsanalyse eine Rolle spielen, definiert (Koçak *et al.* n.d.) Qualitätskriterien die diesen Bereich ebenfalls mit einschließen. Dafür werden neben den herkömmlichen Qualitätskriterien, die den ersten 4 von (Andreou & Tziakouris 2007) verwendeten Eigenschaften in der Tabelle 2.3.2.1 entsprechen, auch Umweltkriterien definiert. Diese Umweltkriterien werden dann in Ressourcen- und Energieverbrauch unterteilt. Ein geringerer Verbrauch bedeutet auch zugleich geringere Kosten. Den einzelnen Kriterien werden ebenfalls Gewichte zugewiesen.

(Li *et al.* 2010) verwendet einen anderen Ansatz als die bereits vorgestellten Arbeiten. Dabei wird die Qualität von Webanwendungen aus verschiedenen Bereichen mit Hilfe von auftretenden Fehlern, seien diese direkt durchs Testen der

Applikation oder durch das Schreiben vom Log-Dateien entnommen worden, bewertet. Dies erlaubt zwar, im Gegensatz zu den zu oben genannten Arbeiten, nummerische Werte anzugeben um einen direkten Vergleich zu ermöglichen, allerdings sagen diese Art der Qualitätsbewertungen nichts über die Struktur, Instandhaltbarkeit oder den Erweiterungsmöglichkeiten des Systems.

Wie die vorgestellten Arbeiten zeigen, unterscheiden sich diese Qualitätskriterien und Ansätze, obwohl die Arbeiten dasselbe Ziel, die Qualitätsbewertung eines Programms, verfolgen. Die Qualitätskriterien bzw. die zu messenden Eigenschaften sind daher der jeweiligen Situation anzupassen.

2.4 Zusammenfassung der dargestellten Theorie

In Kapitel 2 wurde die Theorie, die für die in Kapitel 4 durchzuführende Untersuchung notwendig ist, dargestellt. Dabei wurde sowohl auf CSS (Kapitel 2.1) also auch OSS (Kapitel 2.2) im Bezug auf Definition, Lizenzen sowie Vor- und Nachteile eingegangen. Da das Hauptaugenmerk der Arbeit auf die Nutzung von GIS OSS liegt, wurde in Kapitel 2.2.6 auch das Thema GIS OSS besprochen. Das Kapitel 2.2.5 zeigt, dass der Vergleich von OSS wissenschaftlich angegangen werden kann. Das Wissen wird während Durchführung der Untersuchung in Kapitel 4.2 eingesetzt.

Zum Schluss wurde noch auf den Vergleich zwischen OSS und CSS eingegangen, wobei speziell auf die Themen Aufwandsschätzung (Kapitel 2.3.1), die für die Aufwandsplanung von Projekten genutzt wird und Programm-Bewertungsindikatoren (Kapitel 2.3.2), die eine Einschätzung auf die Qualität eines Programms erlaubt, eingegangen wurde. Die Aufwandsschätzung kommt in Kapitel 4.3 indem ein geplantes Programm abzuschätzen ist, zum Einsatz. Die Programm-Bewertungsindikatoren werden schließlich in Kapitel 5 (Resultate und Interpretationen) verwendet um das Ergebnis der in der Arbeit durchgeführten Untersuchung zu bewerten. Eine Beschreibung der Untersuchung ist im folgenden Kapitel 3 zu finden.

3 Methodik

3.1 Die Untersuchtung

Das Ziel der Arbeit ist zu prüfen ob sich ein proprietäres Geoinformationssystem unter Verwendung von OSS zeitsparender und damit kostengünstiger erstellen lässt

sowie die Ermittlung der Einflüsse auf die Qualität des Produktes. Um tatsächlich wissenschaftlich verwertbare Ergebnisse zu erhalten, die es erlauben die Aussage aus der Literatur zu beweisen oder zu widerlegen sind im Voraus die Schritte klar zu definieren. Zur Beantwortung der gestellten Frage, wird eine Untersuchung in Form eines Experiments durchgeführt, bei dem ein Entwicklerteam eines Unternehmens ein Geoinformationssystem erstellt.

Zu Beginn ist das Programm zu definieren, das in dem Experiment erstellt werden soll. In diesem Fall handelt es sich bei dem Geoinformationssystem um ein eTOD - „electronic Terrain and Obstacle Database“. Die Wahl des Programms für dieses Projekt lässt sich wie folgt begründen. Zum einen ist das Projekt neuartig, was in der Produktion in einem Unternehmen üblich ist um sich am wirtschaftlichen Markt zu positionieren zu können. Zum anderen weil es ein GIS-Produkt ist und sich daher im einem Bereich befindet, wo viele große OSS Projekte existieren, miteinander konkurrieren und aufgrund des Fortschritts in der Softwareindustrie stark im Wandel ist. Das bedeutet, dass ganze OSS-Ökosysteme entstehen und wieder verschwinden und damit ständiger Veränderungen unterworfen sind. Aufgrund dessen liefert der GIS-Bereich gute Voraussetzungen für das im Rahmen dieser Arbeit durchzuführende Experiment.

Für das Experiment ist noch zu erwähnen, dass der Quellcode der OSS Komponenten, die für das zu entwickelnde Programm verwendet werden, nicht verändert wird, sondern durch Schnittstellen verlinkt wird. Diese Methode der Implementierung von OSS in ein Programm ohne den Quellcode zu verändern, ist die am öfterste verwendete in der Produktion von Programmen (Höst *et al.* 2011) und wird daher in dem geplanten Experiment verwendet. Das Verändern des Quellcodes erschwert das Updaten der Komponenten. Mehr Informationen diesen Nachteil können dem Kapitel 2.2.4, welches die Vor- und Nachteile von OSS beschreibt, entnommen werden.

Bezogen auf Ausgangssituation des Experiments ist zu sagen, dass das Entwicklerteam aus 4 Personen besteht, die jeweils eine andere Erfahrungsstufe besitzen. 2 können als Senior Entwickler bezeichnet werden und die anderen 2 als Junior Entwickler. Jedoch hat keine Personen von diesem Team Erfahrung mit einem ähnlichen Projekt sammeln können und haben nur geringes Wissen über OSS. Dies entspricht auch der Situation die bei einem Team, das einen Wechsel von CSS zu OSS durchführt, zu erwarten ist.

3.2 Verwendete Qualitätskriterien

Als nächstes sind die Qualitätskriterien zu definieren. Einige Ansätze und Kriterien aus der Literatur wurden bereits in Kapitel 2.3.2 vorgestellt. Da die Qualität direkt nach der Produktion bewertet wird, entfällt eine Bewertung basierend auf den reinen Quantitätsvergleich der auftretenden Fehler während der Nutzung. Speziell bei Innovationsentwicklungen wo viel auch probiert werden muss, entstehen häufig Fehler während der Entwicklung. Aber diese sind Teil des Schaffungsprozesses. Eine Messung der Anzahl der Fehler wäre daher nicht sinnvoll. Auch der Energie- und Hardwareverbrauch, wie in der von (Koçak *et al.* n.d.) vorgestellten Metrik, wird nicht miteinbezogen, da sich das Experiment auf die Programmentwicklung bezieht. Als Bewertungskriterien werden daher die Programmeigenschaften von (Andreou & Tziakouris 2007), welche in Tabelle 2.3.2.1 dargestellt sind, verwendet. Auf eine Gewichtung um eine Gesamtqualität anzugeben wird jedoch verzichtet, da die Gewichtung frei gewählt werden kann. Würde das Experiment mehrmals wiederholt werden, jedoch mit anderen Gewichten für die einzelnen Bewertungskriterien, so wären auch unterschiedliche Ergebnisse zu erwarten. Das bedeutet, dass das Ergebnis durch das setzen der Gewichte beeinflussbar ist und die Ergebnisse nicht reproduzierbar wären.

3.3 Schritte des Experiments

Das Experiment wird in einzelne Schritte unterteilt um einerseits eine klare Struktur zu liefern und andererseits die Möglichkeit zu bieten einzelne Schritte zu analysieren und zu bewerten. Im Folgenden werden kurz die einzelnen Schritte beschrieben und zu den jeweiligen Kapiteln verwiesen. Das Experiment wird in folgende 3 Stufen eingeteilt:

1 Vorbereitung

   Die Vorbereitung für das Projekt umfasst das Definieren und Festlegen des Rahmens des Experimentes und wurde bereits in Kapitel 1.3 und Kapitel 3.1 beschrieben. Für eine Analyse müssen Vergleichsparameter definiert werden. Hierzu wird die Qualität und der Aufwand herangezogen. Die Qualitätskriterien werden in Kapitel 3.2 festgelegt. Die Aufwandsabschätzung ist als Teil der Durchführung zu sehen und daher in Kapitel 4.3 ausgeführt, nachdem die Anforderungen an das Programm evaluiert sind.

2 Durchführung

Die Durchführung entspricht den Schritten in der Produktion. Die Anforderungen für das zu erstellende Programm werden in Kapitel 4.1 analysiert und festgehalten. Dieser Schritt entspricht auch dem in der CSS Entwicklung und wird daher zeitlich nicht aufgezeichnet. Anhand der Anforderungen wird in Kapitel 4.2 eine Analyse gestartet welche OSS Komponenten in Frage kommen. Dazu wird die aus der Literatur bekannt Theorie aus Kapitel 2.2.5 zur Hilfe genommen. Die Zeit für die Suche und Wahl der OSS Komponenten wird aufgezeichnet.

Nachdem die Komponenten festgelegt worden sind, wird die Zeit, die für die Produktion des Programms benötigt wird, in Kapitel 4.3 abgeschätzt. Anschließend wird das Programm schließlich geplant (Kapitel 4.4), entwickelt (Kapitel 4.6 und 4.7) und getestet (Kapitel 4.8).

3 Resultate und InterpretationNachdem die Produktion des Programms abgeschlossen ist, wird in Kapitel 5.1 der benötigte Aufwand mit dem geschätzten verglichen und etwaige Probleme diskutiert. Auf die resultierende Qualität wird in Kapitel 5.2 eingegangen und schließlich auch bewertet.

Des Weiteren wird in Kapitel 5.3 der Aufwand für eine Applikation berechnet, die auf CSS basiert und in Kapitel 5.4 mit dem Aufwand entwickelten Applikation auf Basis von OSS Komponenten verglichen. Am Ende werden in Kapitel 5.5 die Ergebnisse mit Aussagen aus der Literatur verglichen und diskutiert.

Die 3 beschriebenen Schritte aus dem das Experiment besteht, entsprechen den Schritten, die in der Produktion eines Programms innerhalb einer Unternehmens durchgeführt werden. Auch werden in einem Unternehmen die im letzten Schritt erhaltenen Resultate eines Projektes, bei dem ein Programm erstellt worden ist, analysiert um die gewonnen Erkenntnisse für das nächste Projekt zu nutzen (Jorgensen 2014), allerdings wird nicht zusätzlich mit den Aussagen aus der Literatur verglichen. Eine Änderung dieser Schritte würde daher eine Abweichung von den Schritten in der Programm-Produktion bedeuten.

Im folgenden Kapitel 4 wir auf die Durchführung der Untersuchung eingegangen. Da es sich bei der Untersuchung um ein Experiment handelt, wird etwas präziser die Durchführung behandelt, jedoch werden in Kapitel 5 die Ergebnisse und Erkenntnisse des Experiments in einen allgemeineren Rahmen gestellt.

4 Durchführung der Untersuchung

4.1 Programmanforderungen

Bei der Produktion, also der Planung und Entwicklung eines Programms, ist die Klärung der Anforderungen an das Produkt von äußerster Wichtigkeit. Dies geht damit einher, dass das Produkt direkt an die Anforderungen angepasst werden muss (Anh *et al.* 2012). Anforderungen können sich während der Entwicklung aufgrund von Kundenwünschen ändern, womit auch die Planung und Vorgehensweise der Entwickler beeinflusst werden. Diese Änderungen vermögen einen entscheidenden Kostenfaktor darzustellen (Anh *et al.* 2012). Schließlich können Anforderungen dazu führen, dass weitere Informationen, Wissen oder Kompetenzen für die Produktion erforderlich sind.

Diese Arbeit hat die Produktion einer eTOD Applikation zum Inhalt. Damit handelt es sich bei dem Produkt um eine Programm deren Spezifikationen bereits von der „International Civil Aviation Organization“ ICAO (ICAO n.d.) vorgegeben sind. Damit gelten die Anforderungen auch für dieses Projekt und müssen folglich eingehalten werden.

Für dieses Programm lassen sich nun die Anforderungen wie folgt zusammenfassen:

Das Programm soll:

- ein Web Interface zur Verfügung stellen
- eine Möglichkeit bieten Raster Daten (BIL, DTED) hochzuladen
- es ermöglichen Datenmanagement zu betreiben, was in folgende Unterpunkte aufgeteilt wird:
    - Gruppieren von Daten in Datengruppen
    - Setzen und verändern von Metadaten
    - Löschen von Daten und Datengruppen
- Berechnung von Höhenschichtlinien aus Raster Daten
- Datengruppen für externe Applikationen zur Verfügung zu stellen
- die Möglichkeit bieten zu entscheiden welche Daten für externe Applikationen zur Verfügung stehen
- Möglichkeit die Daten herunter zu laden
- der ISO-Norm ISO 19115 folgen (International Organization for Standardization 2009)

- den ICAO Spezifikationen entsprechen die in Annex 4,14 und 15 definiert sind (ICAO ETOD n.d.)
- dem Standards des Open Geo Consortiums OGC konform sein (OGC n.d.)

Zuletzt werden noch die Anforderungen hinzugefügt, die das Unternehmen intern als wichtig erachtet werden und eine höhere kundenfreundliche Nutzbarkeit gewährleistet.

Das Programm soll daher auch...

- eine Karte zur 2D Datenvisualisierung beinhalten
- optional auch eine Karte zur 3D Datenvisualisierung beinhalten
- ein existierendes User Management System verwenden
- eine Anmeldung über ein existierendes firmeneigenes Schnittstelle erlauben

Die Anforderungen werden schließlich zusammengefasst und in einem eigenem System aufgenommen um deren Erfüllung oder Änderung aufzeichnen zu können. Während der Produktion werden dann die eingetragenen Anforderungen verwendet um Tests für die Software zu schreiben. Dieser Vorgang wird in Kapitel 4.8 näher erläutert.

4.2 Analyse und Wahl des OSS

Geoinformationsprogramme wie das eTOD, welches geplant und umgesetzt werden soll, sind sehr aufwendig, Jedoch könnten einige Teile bereits von anderen Firmen, Privatpersonen oder Organisation entwickelt und optimiert worden sein. Es gilt daher sich vor der Planung des Projektes ein Bild zu machen, ob und welche Programme oder Bibliotheken auf dem Markt sind und ob das Einbinden dieser die Kosten der Entwicklung einer neuen eTOD Applikation senken könnte. Dabei lassen sich diese Programme in zwei verschiedene Arten trennen die dann als kommerzielle sowie Open Source Programme bezeichnet werden. Dieses Experiment wird nur OSS nutzen. Die Wahl der OSS Komponenten kann nicht nur große Auswirkungen auf die Produktionskosten haben, sondern aber auch auf die Architektur des Programms und damit auch auf die Planung des Projektes. Zuerst wird ermittelt welche GIS-OSS Produkte zur Verfügung stehen. Mit Hilfe der schriftlich festgehaltenen Anforderungen wird eine Websuche gestartet, die folgende Produkte geliefert hat: MapServer (http://mapserver.org/), GeoServer (http://geoserver.org/), Deegree (http://www.deegree.org/), Mapnik (http://mapnik.org/), MapFish (http://mapfish.org/), 52°North (http://52north.org/) und Zoo Project (http://www.zoo-project.org/). Die Produkte die gefunden werden, sind mit Hilfe der Dokumentationen zu analysieren, wie es auch von (Anh *et al.* 2012) vorgeschlagen wird.

Zoo Projekt bietet einen Web Processing Service (WPS) Server an mit einer Programmierschnittstelle für das Aufrufen der Services. Das erlaubt das Durchführen von Prozessen wie das Berechnen von Höhenschichtlinien aus Rasterdaten, welche ein Teil der Anforderungen darstellt. 52°North liefert eine Anzahl von GIS-Programmkomponenten darunter auch ähnliche Produkte wie das Zoo Projekt. Die anderen genannten Produkte stellen einen ganzen Server zur Verfügung, welche mit einer großen Anzahl an Funktionen und einfacher Benutzung werben. MapServer, Deegree sowie GeoServer sind dabei führend. Diese bieten einen Web Mapping Service (WMS) zur Verbreitung von Rasterdaten übers Web, Web Feature Service (WFS) zur Verbreitung von Vektordaten, aber auch einen WPS an und sind OGC konform. Damit wäre die Anforderung, die Daten extern zur Verfügung zu stellen und dem OGC Standard (OGC n.d.) zu folgen erfüllt. Auch die Speicherung der Daten wäre damit geklärt. Durch eine Vielzahl an Konfigurationsoptionen lässt sich der jeweilige Server nach den Bedürfnissen anpassen. Aufgrund dessen steht die Entscheidung fest einen GIS-Server zu verwenden. Da die vorgestellten Produkte sehr ähnlich sind, sollten diese verglichen werden.

Daher wird jetzt mit Hilfe der in Kapitel 2.2.5 präsentierte Theorie die für das Projekt passenden OSS Komponenten bewertet und gewählt. Dazu wird Google Trends (https://www.google.at/trends), GitHub (https://github.com/) und Open HUB (https://www.openhub.net/) verwendet. Open HUB ist ein Portal, das Auskunft über alle zurzeit bekannten OSS Produkte liefert. Ein Besucher der Seite kann neben der Summe an Quellcodezeilen, dem Datum des zuletzt gelieferten Updates und der Anzahl der Mitarbeiter auch auf weitere wichtige Informationen zugreifen. Die Informationen werden dabei in Aktivität, Gemeinschaft und Projektzusammenfassung gegliedert. Da die Informationen teilweise unvollständig oder nicht aktuell sind, werden in Tabelle 4.2.1 nur die Werte verwendet, die bei allen dreien OSS Produkten angegeben sind. Die Daten wurden am 28.4.2016 aus OpenHUB entnommen. Zu sehen ist, dass GeoServer mit 84 Mitarbeiter die meisten besitzt, während Deegree nur 8 vorweisen kann. Bei Deegree könnte man vermuten, dass die Entwicklung in absehbarer Zeit eingestellt wird oder jünger ist als die Mitbewerber. Da das letzte Update bereits mehr 4 Jahre vom Referenzdatum 28.4.2016 zurück liegt, ist mit ersterem zu rechnen. MapServer liegt mit 33 Mitarbeiter im Mittelfeld, aber das letzte Update ist 4 Tage vom Referenzdatum zurück und ist verglichen mit GeoServer, dessen letztes Update gerade einmal einen Tag zurück liegt, ebenfalls für die OSS Wahl interessant.

OpenHUB	GeoServer	MapServer	Deegree
Aktuelle Mitarbeiter	84	33	8
Letztes update	1 Tag	4 Tage	>4 Jahre
Lizenz	GPLv2	MIT	LGPL

Tabelle 4.2.1: aus OpenHUB entnommene Vergleichswerte für GeoServer, MapServer und Deegree vom 28.4.16

Da die Informationen nicht ausreichen um eine gute Einschätzung abzugeben, werden, wie in (Jansen 2014), Werte aus GitHub in die Bewertung miteinbezogen. Die verwendeten Werte können aus Tabelle 4.2.2 entnommen werden.

Git	GeoServer	MapServer	Deegree
Repositorien	487	218	38
Quellcodezeilen	240.834	470.501	45.332
Themen	6,468	10.393	522

Tabelle 4.2.2: aus GitHub entnommene Vergleichswerte für GeoServer, MapServer und Deegree vom 28.4.16

GitHub ist ein Webseite mit Diensten für Projekt- und Quellcodeverwaltung und sozialer Vernetzung. Das Herzstück von GitHub ist das Versionsverwaltungsprogramm Git. Daher kann GitHub interessante Statistiken über die jeweiligen Projekte liefern. Ein paar dieser quantitativen Werte, die auch in Tabelle 4.2.2 angezeigt sind, spiegeln die Anzahl der Repositorien, der Quellcodezeilen und der diskutierten Themen in GitHub wieder. Bei der Anzahl der Repositorien ist GeoServer mit 487 führend und Deegree bildet das Schlusslicht. Bei der Anzahl der Quellcodezeilen ist jedoch MapServer mit knapp der doppelten Anzahl vor dem GeoServer und damit, bezogen auf den Quellcode, größer. Auch bei den diskutierten Themen ist MapServer führend. Die Werte von Deegree dagegen geben den Hinweis, dass das Produkt sowie die Gemeinschaft hinter diesem Projekt nicht sehr groß ist. Deegree sollte daher als GIS Server nicht in Betracht gezogen werden. Die Werte der beiden führenden GIS Server GeoServer und MapServer reichen aber für eine Entscheidung nicht aus. Laut (Rudzki *et al.* 2009) sollte auch die Reife des OSS Produktes sowie deren mögliche Zukunft, daher ob das Projekt möglicherweise bald eingestellt oder kommerziell wird, untersucht werden. Da die bisherigen Werte nur den momentanen Status widerspiegeln, wird Google Trends für

eine Analyse der Entwicklung der Projekte über die Zeit genutzt. Es wird jedoch nicht direkt die Entwicklung gemessen sondern mit Google Trends lässt sich die Anzahl der Google-Suchanfragen für das jeweilige Projekt in Abhängigkeit der Zeit angeben. In Abbildung 4.2.1 kann das Ergebnis betrachtet werden. Dabei sind die in der Abbildung dargestellten Werte als relativ zu betrachten, wobei der höchste Wert in dem Diagramm mit 100 beziffern wird. Deegree, mit der gelben Farbe gekennzeichnet, bleibt hinsichtlich der Suchanfragen deutlich unter den anderen GIS Server Projekten. Damit wird die Vermutung bestätigt, dass die Entwicklung bald eingestellt werden könnte. Betrachtet man nun die rote Linien, stellvertretend für den MapServer sowie die blaue Linie stellvertretende für den GeoServer, so sind die in den Tabellen angegebenen Werte nun verständlich. Der MapServer ist ein reiferes Produkt und hat bereits seine höchste Trendphase hinter sich. Das erklärt auch die hohe Anzahl an Quellcodezeilen und Themen. Die Interesse über die Zeit scheint drastisch abzusinken, während die für den GeoServer sich in einem gewissen Rahmen hält. Angesicht der deutlich höheren Anzahl von Mitarbeitern beim GeoServer ist mit schnelleren Problembehandlungen und neuen Funktionen zu rechnen. Außerdem ist es sehr wahrscheinlich, dass die Gemeinschaft aktiver ist und damit bei Problemen schneller helfen bzw. antworten wird. Angesichts der positiven Punkte, die für den GeoServer sprechen, wird dieser als GIS Server gewählt.

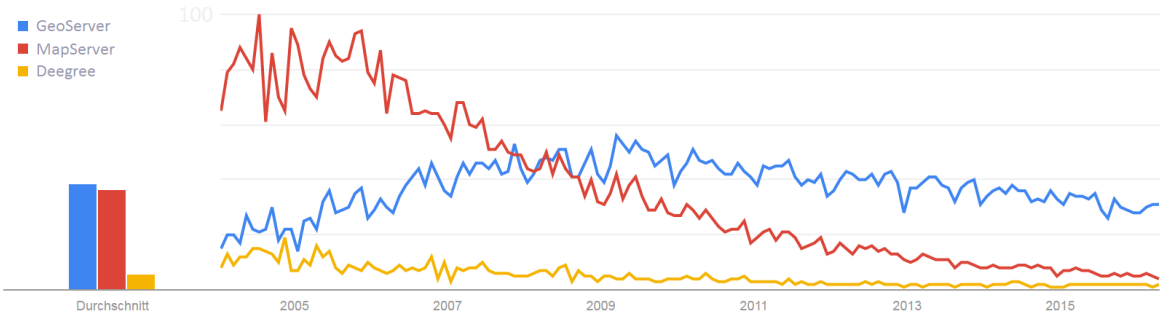

Abbildung 4.2.1: von Google Trends entnommener zeitlicher Verlauf der Interesse für GeoServer, MapServer und Deegree vom 28.4.16

Das bedeutet, dass die Analyse tatsächlich die nötigen Informationen für die Entscheidung liefert. Die zusammengefassten Informationen erlaubten eine einfachere Bewertung der Komponenten im Bezug auf die Weiterentwicklung, Stabilität und des vorherrschenden Ökosystems. Es sichert damit auch die nähere Zukunft für das derzeitige Projekt.

Als nächstes wird nach einer Karte für die Applikation gesucht mit der der Nutzer interagieren kann. In dieser Karte sollen die Daten angezeigt werden, die im eTOD System gespeichert sind und sollte am besten Visualisierungen in 2D und 3D

anbieten. Außerdem soll die Möglichkeit bestehen die Karte in eine Webapplikation einzubinden. Ein Websuche liefert folgende Ergebnisse: Open Layers (http://openlayers.org/), Cesium (https://cesiumjs.org/), MapFish (http://www.mapfish.org/), Mapbox (https://www.mapbox.com/), Leaflet (http://leafletjs.com/). Im Vergleich zu den GIS Servern, kann hier eine deutlich einfachere Entscheidung gefällt werden. Mapfish scheidet aufgrund seiner sehr kleinen Gemeinschaft und limitierten Funktionen aus. Leaflet wird von bekannten Firmen genutzt, da der Vorteil von Leaflet in der Einfachheit liegt. Allerdings sind bei dem geplanten Programm die Funktionen wichtiger. Vor allem könnten in der Zukunft noch weitere Anforderungen an das Programm gestellt werden. In solch einem Fall sollte die Erweiterung der Funktionen einfacher sein um den benötigten Aufwand zu reduzieren. Cesium ist als einzige Bibliothek in der Lage einen 3D Globus oder Karte darzustellen und liefert auch Funktionen 3D Daten anzuzeigen. Daher wird Cesium als Komponente in die Applikation aufgenommen. Für die Wahl der Bibliothek für die 2D Karte bleiben damit noch Open Layers und Mapbox. Mapbox enthält Tools für das einfache Stylen der Karten, ist dafür in seinen Funktionen nicht so umfangreich wie Open Layers. Open Layers dagegen wird auch von GeoServer verwendet und erlaubt aufgrund der breiten Palette an Funktionen auch komplexere GIS Applikationen herzustellen. Des Weiteren ist es länger am Markt und besitzt daher eine größere Gemeinschaft. Aufgrund dessen fällt bei der Wahl der 2D Karte die Entscheidung auf Open Layers.

Neben GeoServer, Cesium and Open Layers werden noch weitere OSS Komponenten benötigt. Der GeoServer ermöglicht den Import von einigen verschiedenen Dateiformaten. Allerdings muss die geplante Applikation aufgrund der Anforderungen auch die Formate DTED und BIL einlesen können, welche vom GeoServer im Moment nicht unterstützt werden. Eine kurze Suche liefert die Geospatial Data Abstraction Library, kurz GDAL, (http://www.gdal.org/). Dieses Kommandozeilenprogramm wird ebenfalls vom GeoServer verwendet und liefert eine breite Palette an Funktionen mit Hauptaugenmerk auf Datenkonvertierung. GDAL unterstützt alle bekannten GIS Datenformaten ungeachtet dessen ob es sich um Raster- oder Vektordaten handelt. Da DTED und BIL zu der Liste der konvertierbaren Datenformate gehören, ist diese Anforderung damit abgedeckt.

Zwar ist laut den Anforderungen neben dem User Management System keine weitere Datenbank geplant, allerdings wird für den Fall auch hier nach der richtigen Komponente gesucht. Sollte es aufgrund von neuen Anforderungen nötig sein, eine Datenbank zu verwenden, muss die System dies ermöglichen. Es stellt sich heraus, dass der GeoServer auch das Nutzen von Datenbanken ermöglicht und zwar sowohl Oracle als auch PostgresSQL Datenbanken. Da PostgresSQL eine breite Palette an Erweiterungen bieten, darunter auch Postgis, welches 2D und 3D Berechnungen an

geometrischen und geographischen Daten erlaubt, würde die Entscheidung auf die flexiblere Lösung PostgresSQL fallen.

Die laut den Dokumentationen bestehende Kompatibilität des GeoServers zu anderen OSS, wie PostgresSQL und Open Layers, zeigt, dass der GeoServer gut in das GIS OSS Ökosystem eingebunden ist. Für Entwickler, die den GeoServer einbinden, stellt diese Kompatibilität ein geringeres Risiko in Bezug auf die Einstellung der Entwicklung durch die OSS Gemeinschaft dar. Der GeoServer unterstützt diverse andere Komponenten und bildet damit einen wichtigen Teil des geplanten Programms. Ein Blick auf die Literatur zeigt auch, dass es GIS Projekte gibt, die auf ähnliche Zusammenstellungen der Komponenten setzen (Delipetrev *et al.* 2014b) (Xia *et al.* 2009). Dennoch ist dies aber keine Garantie, dass keine Probleme auftreten werden, denn die Art der Nutzung der einzelnen Komponenten ist von den Anforderungen an das zu erstellende eTOD Produkt abhängig.

Für das Wählen der GIS-OSS Komponenten wurden 3 Arbeitstage in Anspruch genommen, wobei dies das Suchen, das Lesen der Dokumentationen sowie das Durchführen der Analysen beinhaltet.

4.3 Aufwandsabschätzung für OSS

Bevor mit der Entwicklung begonnen werden kann, ist eine Aufwandsschätzung abzugeben. Dies ist in einem Unternehmen notwendig um die Kosten zu berechnen und Ressourcen einzuteilen. Des Weiteren kann der Kunde informiert werden, wann das Programm voraussichtlich abgeschlossen wird.

In Kapitel 2.3.1 wurde die Theorie der Aufwandsschätzung bereits beschrieben. Für die Aufwandsschätzung des geplanten Programms wird die „Schätzung nach Fachexperten“ angewendet (Rammage *et al.* 2010) (Basten & Sunyaev 2011). Das Projekt wird in kleinere Teilaufgaben bzw. Einheiten aufgeteilt um diese einfacher und genauer schätzen zu können. Die Summe der Aufwände aller Einheiten liefert den Aufwand des Projektes. Für die Schätzung werden 4 Entwickler zusammen gerufen. Nach der gemeinsamen Aufteilung des geplanten Programms in kleinere Einheiten, schätzt jeder Entwickler für sich allein die einzelnen Einheiten. Ähnlich wie bei der Delphi Methode (Basten & Sunyaev 2011) werden die Schätzungen im Team besprochen und etwaige Unklarheiten beseitigt. Nach der Diskussion einigt sich das Team auf eine bestimmte Schätzung. Das Ergebnis der Schätzung kann in Tabelle 4.3.1 betrachtet werden. Die Spalte „Storys“ gibt den Bereich an, welcher dann unter der Spalte „Tasks“ in kleinere Einheiten bzw. Aufgaben geteilt wird. Die Aufteilung der

Bereiche wird auf diese Weise durchgeführt, sodass zusammenhängende Aufgaben jeweils einen Bereich bilden. Unter „Estimations“ sind die Schätzungen der Entwickler zu sehen, die sich zum Teil stark unterscheiden können, da die Experten in verschiedenen Bereichen einen unterschiedlichen Erfahrungsschatz aufweisen. Die Spalte „Results“ sind die Ergebnisse, auf die sich das Team nach der Besprechung der getätigten Schätzungen geeinigt und festgelegt hat. Die letzte Spalte enthält die Summe der Aufwandsschätzung über jeden Bereich. Die Summe der Teilaufwände ergibt einen Gesamtwert von 608 Stunden bzw. 76 Arbeitstage, die mit 8 Arbeitsstunden pro Tag kalkuliert werden. Das Team kann auf historische Projektdaten zugreifen, die dazu verwendet werden einen historischen Produktionsfaktor, entsprechend der in Kapitel 2.3.1 dargestellten Theorie, zu berechnen, welcher mit 0,8 beziffert werden kann. Das bedeutet, dass das Team stets 20% länger gebraucht hat als angenommen. Es ist zu sagen, dass das nicht unbedingt am Team selbst liegen muss. Ungeplante Meetings zwischen verschiedenen Projektteams oder zwischen Entwicklern und dem Management können beispielsweise die Ursache sein.

Storys	Tasks	Estimations (h)				Result	Sum
Project Setup	Maven Setup	8	8	6	5	16	
	Geoserver Configuration (Extensions, Clean-Up, Configuration, GeowebCache)	16	24	16	3	24	
	Javascript-Build	5	8	6	3	8	
	Application-Server Setup	20	16	18	10	16	
	Infrastructural tests	16	8	16	3	16	
	Security	8	8	8	7	8	
	Online Help	8	24	20	4	24	112
UMS	Adapt gui to include new section for etod roles in organisation	4	2	4	2	4	
	Adapt gui to include new section for etod roles in user management	4	2	2	2	4	
	Testing	4	2	4	5	4	
	Enhance usimex	6	6	6	6	6	
	Single sign-on/session valiation	16	16	16	16	16	34
Data Upload/Import	UI Basic Setup (template for all pages)	8	16	10	4	12	
	Forms to specify upload target (grouplayer, dataname)	4	6	4	2	5	
	Styling	10	8	8	7	12	
	Upload/Import Page (MVC - Component, Dataexchange handling)	12	16	12	5	16	
	Fileformat handling	5	4	5	5	5	
	Error handling	10	16	16	8	20	
	Rest-Application (Server-Setup)	24	16	24	5	24	
	Geoserver REST-API Response handling (investigation)	8	20	18	16	20	
	DAO/DTO with Hibernate/Spring	16	18	16	6	18	
	Service-Implementation: Workspace (CRUD)	6	18	16	8	18	
	Service-Implementation: Layer (CRUD)	6	18	16	8	18	
	Service-Implementation: Datasets (CRUD)	6	18	16	8	18	
	Service-Implementation: Style (CRUD)	6	18	16	8	18	
	Service-Implementation: Publish (WMS/WFS/WCS/WPS access) (CRUD)	6	18	16	8	18	
	Implement map (layerswitcher, controls, baselayers)	10	10	10	10	10	232
calculation of countour lines	Implement 2D map view	4	4	4	1	2	
	Implement LayerFactory (OGC conforme layer)	4	16	4	3	8	
	Styling of map components	8	4	8	2	8	
	Customer specific key-configuration (properties on server)	2	4	2	8	4	
	(Async) layer preview (depending on rights) in table depending on viewport (render option)	10	16	10	5	18	
	create WPS calculation prozesses	10	14	10	16	16	
	Filter (WPS) request parameters and make request to geoserver (when data on geoserver)	5	4	5	8	8	
	GUI	8	4	8	2	10	74
define/change datasets	Single workspace for each organisation	6	4	6	8	10	
	Create layergroups (datasets) in workspace by admins	6	4	6	3	6	
	Add layer (data) to layergroups (datasets) by admins	6	4	6	3	6	
	Display layergroups and layers with drag&drop feature for moving data within datasets	8	18	12	10	18	
	Datasets and data UI	4	8	4	7	8	48
meta data add/change	Implement component for modifying datasets and data (editable table)	4	16	8	4	8	
	Providing default styles for display layer on map	8	4	6	4	6	14
expose data	Access to OGC Services through AIMSL	6	6	8	6	6	
	Proxy	12	6	12	10	12	
	SDO through GT db-Layer	16	8	16	8	12	30
visualisation in a web map	Implement obstacle restservice	16	8	16	6	12	
	SDD via WFS Service	8	4	8	5	8	
	Spatial request from client to retrieve obstacle data	8	4	14	3	16	
	Obstacle styling (colorcode height, ObstaclePainter)	8	16	12	3	16	
	Create layer with obstacle data and add to map	6	4	5	5	5	57
Meetings/Design discussions		60	60	60	60	60	60

Sum	661	~83d
Buffer	20,00%	
Final Sum	~793h	~99d

Tabelle 4.3.1: Ergebnis der Schätzung nach Fachexperten mit 3 Experten

Da der Produktionsfaktor bekannt ist, wird dieser auch in die Berechnung miteinbezogen. Bei einer Schätzung von 661 Arbeitsstunden plus einem Puffer von 20% ergibt sich eine SOLL - Dauer von gerundet 793 Stunden bzw. 99 Arbeitstagen. Die IST – Dauer ergibt sich durch die Entwicklung selbst. Der Vergleich wird in Kapitel 5.1 gezogen. Die Aufwandsschätzung für CSS wird nach Abschluss der Programmentwicklung durchgeführt um die Entwicklung mit der Schätzung mental nicht zu beeinflussen.

4.4 Planung und Aufbau

Bevor mit dem Projekt begonnen werden kann, ist dieses vor dem Start zu planen. Diese Planung wird dann herangezogen um die benötigten Ressourcen, bestehend aus Geldmittel und Mitarbeitern einzuteilen. Um dies bewerkstelligen zu können, sind zuerst die Anforderungen an das Produkt zu klären, welche auch in Kapitel 4.1 festgelegt und beschrieben werden. Die Planung umfasst einen größeren Bereich, da sich diese Arbeit aber mit der Implementierung von GIS-OSS in der Produktion eines eTOD Systems beschäftigt, wird im Folgenden nur auf die Planung der Programm-Architektur und deren Komponenten eingegangen. Auf die Dauer der Entwicklung wird im vorhergehenden Kapitel 4.3 eingegangen.

Betrachtet man die Funktionen die das Programm bereit stellen muss, so geht es im Kern darum Geodaten zu erfassen, organisieren, bearbeiten und präsentieren.
Bei einer eTOD Applikation handelt es sich bei den Geodaten um Raster und um Vektordaten Angesichts dessen, dass laut den Anforderungen nur Rasterdaten benötigt werden, nicht jedoch Vektordaten, ist es genau genommen ein Teil von einer eTOD Applikation, die von der ICAO beschrieben wird. Im Folgenden wird aufgrund der Übersichtlichkeit dennoch die Bezeichnung eTOD Applikation oder Programm verwendet.

Eine wichtige Stellung nimmt der GeoServer ein. Dieser kann für das Daten- und Metadaten Management sowie für die Bereitstellung von Daten externe Applikationen in Form von Web Service (WMS,WFS) verwendet werden. Da die Web Service Schnittstelle des GeoServers OGC konform sind, ist damit diese Anforderung erfüllt. Der GeoServer bietet auch Web Processing Services (WPS) an womit Prozesse an vorhandenen Daten durchgeführt werden können. Im Fall der eTOD Applikation, ist es beispielsweise das Berechnen von Höhenschichtlinien aus Raster Daten.

Der GeoServer bietet auch ein eigenes Interface an, jedoch gilt es zu vermeiden, dass der Nutzer des geplanten Programms auf den GeoServer direkt zugreift. Ein Grund dafür ist, dass der GeoServer Inhalte von verschiedenen Nutzern enthält und damit eine höhere Sicherheit erforderlich ist. Ein weiterer Grund ist, dass ein direkter Zugriff die Weiterentwicklung eines angepassten und erweiterten Interfaces erschwert. Das mit dem GeoServer mitgelieferte Interface ist außerdem sehr komplex und enthält eine große Anzahl an Funktionen die nicht gebraucht werden. Schließlich ist zu bedenken, dass es sich bei den Nutzern um keine Spezialisten aus dem Bereich der Kartographie und Geoinformation handelt.

Neben dem GeoServer wird auch die Open Source Bibliothek GDAL verwendet, die für die Konvertierung von Daten notwendig ist, die der GeoServer zur Zeit nicht unterstützt.

Die Daten für das UMS werden in einer Oracle Datenbank gespeichert. Die von der eTOD Applikation zu verwaltenden Daten und Metadaten werden im GeoServer in einem eigenen Depot gespeichert. Eine Alternative wäre dieses Depot mit einer PostgresSQL Datenbank zu ersetzen.

Die Stellung der auf dem Server laufenden GIS OSS GeoServer und GDAL können in der Abbildung 4.4.1 betrachtet werden. Die Abbildung wurde mit der OSS „Dia“ erstellt.

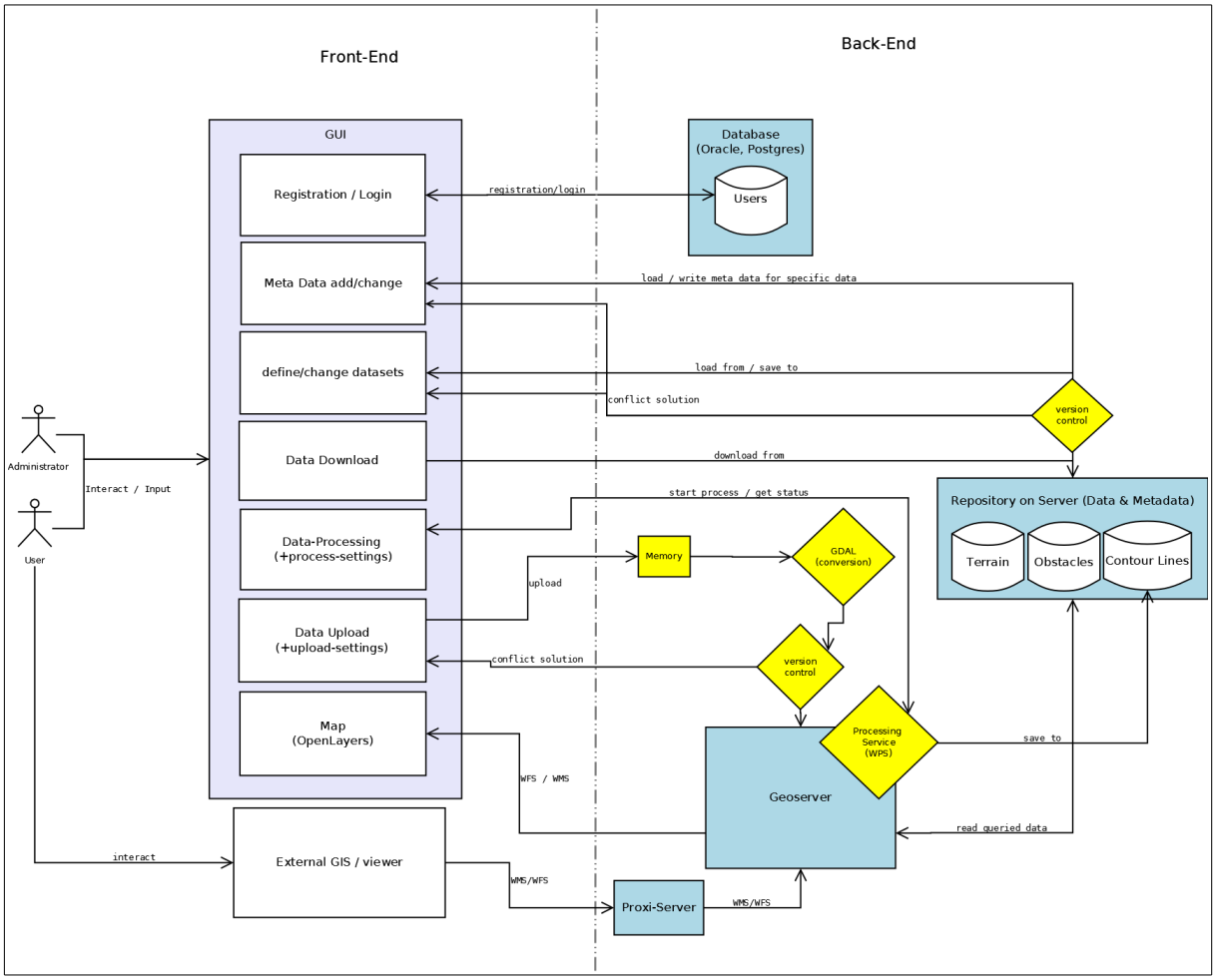

Abbildung 4.4.1: Programmarchitektur der eTOD Applikation

In der Abbildung 4.4.1 ist zu sehen, dass die GIS-OSS Komponenten nicht direkt mit dem Web Interface verbunden sind, sondern mit einem zu entwickelnden Programmteil, das im Folgenden Services bezeichnet wird. Diese Services liegen wie die GIS-OSS Komponenten auf der Serverseite und kommunizieren zwischen diesen und dem Web Interface. Dieser Programmteil soll auch die Versionskontrolle und Datenaufbereitung übernehmen.

Das Web Interface wird über das Gerät des Nutzers in einem Browser seiner Wahl aufgerufen. Die Applikation wird dabei so angepasst, dass sie sowohl auf großen Bildschirmen als auch auf kleinen mobilen Geräten funktionsfähig bleibt. Das

Interface dient zur Interaktion mit dem Nutzer was die Daten-Aufnahme und Ausgabe sowie die Visualisierung der Daten mittels einer Karte mit einschließt.

Wie weiter oben erwähnt bietet der GeoServer Web Services an um Daten für externe Applikationen oder für die verwendete Karte zur Verfügung zu stellen. Aus Gründen der Sicherheit aber auch der Erweiterung der Funktionen, werden die Datenanfragen der Nutzer oder einer externen Applikation von einem Proxy-Server abgefangen, auf Zugriffsrechte geprüft und anschließend zum GeoServer weitergeleitet. Für die Karte die sich in der Webanwendungen befindet und Daten ebenfalls über die Web Services erhält, wird Open Layers verwendet, deren zuletzt veröffentlichte Version die dritte ist.

Im weiteren Verlauf der Arbeit wird der Programmteil, der am Gerät des Nutzers läuft als Client und deren Entwicklung als Frontend Entwicklung bezeichnet. Die Entwicklung des Programmteils am Server wird als Backend Entwicklung bezeichnet.

Im Folgenden entspricht der Aufbau auch der Planung und Umsetzung. Nach der Einrichtung der Projekt- und Entwicklungsumgebung wird zuerst auf die Backend Entwicklung eingegangen, da diese die Funktionalitäten bereitstellt. Außerdem ist im Backend, aufgrund der mangelnden Erfahrung mit den gewählten GISS-OSS mit Problemen zu rechnen.

4.5 Verwendete Software und Projekt Setup

Bevor mit der Entwicklung begonnen werden kann, muss nach der im letzten Kapitel 4.4 besprochenen Planung der Arbeitsplatz für die Entwicklung vorbereitet werden und das Projektsetup, welches das Installieren aller Bibliotheken und zu brauchenden Tools umfasst, durchgeführt werden. Der zeitliche Aufwand für das Projektsetup ist jedoch wegen Installations- und Interoperabilitätsproblemen mit dem genutzten Betriebssystem nicht zu unterschätzen.

Als Entwicklungsumgebung wird IntelliJ IDEA verwendet, welches die Backend und Frontend Entwicklung in demselben Tool erlaubt. Zwar ist es nicht kostenfrei, jedoch ist schnell und komfortabel aufgrund der effizienten Code Vervollständigung und Syntax Kontrolle besitzt. Das begünstigt eine effektivere Programmierung und Fehlersuche.

Verbunden wird IntelliJ mit einem freien Programm für die Versionskontrolle, dem TortoiseSVN. Im Laufe der Programmierung entstehen Code Stücke die obsolet aber möglicherweise später wieder gebraucht werden. Daher sollten diese, sowie Abbilder von den stabilen Versionen, als Sicherung aufgenommen und aufgehoben werden. Konflikte zwischen den Versionen sollten ebenfalls auflösbar sein. Das

optimiert nicht nur das Arbeiten alleine sondern auch im Team. Die Versionskontrolle ist dafür eine gute und einfache Lösung. Eine moderne performante Alternative wäre auch Git. Ein geschlossenes Depot ist jedoch mit weiteren Kosten verbunden und bei diesem Projekt besteht nicht die Notwendigkeit, daher fiel die Wahl auf TortoiseSVN.

Das Projekt wird in IntelliJ mit Maven, einem Projekt-Management Tool, erstellt. Für die Backendentwicklung, die auf Java basiert, wird Apache Tomcat für das Starten und Laufen von Java Applikationen installiert und konfiguriert. Die meist verwendete Java Bibliothek ist zum einen Jersey, welches für die serverseitige Kommunikationsschnittstelle zwischen den Komponenten gebraucht wird und zum anderen die GeoServer-Manager API, die eine vereinfachte Kommunikation mit dem dem GeoServer erlaubt. Für die Verbindung zu einer Datenbank wird Hibernate verwendet. Für die Frontend Entwicklung wird Nodejs sowie npm und Bower als Bibliotheksmanager für Nodejs Projekte genutzt.

Die hauptsächlich verwendeten Frameworks für die Frontend Entwicklung sind AngularJS und Bootstrap. AngularJS basiert auf Javascript und erlaubt dynamische Webapplikationen zu erstellen. Eigene HTML Objekte und Attribute können einfach erstellt werden, was nicht nur die Strukturierung des Codes verbessert sondern auch seine Wiederverwertbarkeit erhöht. Bootstrap hingegen enthält auch HTML und CSS Komponenten und liefert einerseits einen einfachen modernen Stil für das Aussehen der Applikation und ermöglicht des Weiteren eine Webapplikation zu entwickeln die sich auf die Größe des Bildschirms anpasst. Somit kann die Applikation sowohl auf einem Stand-PC als auch auf einem mobilen Gerät wie einem Tablet oder Smartphone genutzt werden. Sowohl AngularJS als auch Bootstrap besitzen zudem eine große und aktive Gemeinschaft, die eine große Zahl an Bibliotheken und Plug-ins hervorbrachte. Für die clientseitige Kommunikationsschnittstelle wird die Javascript Bibliothek Restangular verwendet. JSHint, eine weitere Bibliothek, hilft den Code auf etwaige Fehler oder Nichteinhaltung von Konventionen zu prüfen.

Da jede Funktion der Applikation getestet werden muss, wird auch für dies etwas eingerichtet. Protractor bietet sich als end-to-end Test-Framework für AngularJS an. Zusammen mit der Applikation BrowserStack, das dem Testen auf verschiedenen Browsern und Fenstergrößen dient, lässt sich die geplante Applikation dann browserübergreifend testen. BrowserStack liefert auch die Möglichkeit die Browserversionen zu ändern und erstellt bei jedem durchgeführten Test Log-Dateien und Screenshots zur Verfügung um die Resultate auch manuell überprüfen zu können.

Das Projekt Setup ist damit abgeschlossen. Die Links zu allen verwendeten Programmen und Bibliotheken können im Anhang 8.1 und 8.2 gefunden werden.

4.6 Backend Entwicklung

Die Backend Entwicklung umfasst den ganzen Teil der Entwicklung des Programmteils der auf dem Server laufen wird. Dies beinhaltet die Services (Kapitel 4.6.2), die für die Kommunikation zwischen den einzelnen Teilen verantwortlich sind, den GeoServer (Kapitel 4.6.1), der das Daten Management und die Web Services bereitstellt sowie GDAL (Kapitel 4.6.2.3), das für die Datenkonvertierung zuständig ist. Diese Einbindung dieser Komponenten sowie aufkommende Probleme werden im Folgenden genauer behandelt.

4.6.1 GeoServer

4.6.1.1 Funktionen des GeoServers

Die wichtigste OSS Komponente im Backend stellt der GeoServer dar. Der GeoServer ist ein von der Open Source Geospatial Foundation ins Leben gerufenes Projekt, das in Java programmiert und unter der GNU General Public Lizenz veröffentlicht ist. Es stellt einen Server dar, der speziell geographische Daten über Web Services in einem Netzwerk zur Verfügung stellen kann, die dann in weiterer Folge für Visualisierungen und Applikationen verwendet werden können. Dabei werden Offene Standards eingehalten, die vom OGC definiert sind. Diese verbessern die von der dahinter stehenden Gemeinschaft gewünschten Interoperabilität mit anderen Programmen. Zur Verwendung der Web Services muss eine Anfrage mittels HTTP gesendet werden, die zum einen die Adresse des GeoServers und zum anderen die Parameter, die der GeoServer zur Bearbeitung der Anfrage benötigt, enthalten. Die Web Services werden in vier verschiedene Arten geteilt, die Web Mapping Service WMS, Web Feature Service WFS, Web Coverage Service WCS und Web Processing Service WPS genannt werden. Während der WMS Bilddateien ausgibt, liefert der WFS Vektor Daten. WCS liefert zwar ebenfalls Bilddateien allerdings kann dieser Raum-Zeit abhängige Eigenschaften darstellen. Die durch einen WCS erhaltenen Daten lassen sich dann auch für etwaige Analysen maschinell lesen. WPS erlaubt über eine Anfrage Berechnungen an den Daten durchzuführen. Sind mehrere Berechnungen notwendig lassen sich diese auch zu einer Kette koppeln. Das Ergebnis kann entweder im Speicherdepot des GeoServer gesichert werden oder durch eine weitere Anfrage wieder ausgegeben werden. Je nachdem welche Einstellungen getroffen werden, werden die Prozesse synchron oder asynchron durchgeführt. Im ersten Fall wird nach dem Absenden der Anfrage auf das berechnete Ergebnis gewartet. Im asynchronen Fall wird beim Starten eines

Prozesses eine Identifikationsnummer zurückgegeben. Diese kann bei einer weiteren Anfrage als Parameter verwendet werden um Auskunft darüber zu bekommen wie der Status der jeweiligen Berechnung ist. Wenn der Status die Information liefert, dass der Prozess abgeschlossen ist, können die resultierenden Daten abgefragt werden. Bei allen Arten der Web Services lassen sich Parameter angeben um die Ausgabe zu verändern. Das heißt, dass der GeoServer die Daten konvertieren bzw. transformieren kann, abhängig von den gesendeten Werten für die Parameter. Somit ist es möglich bestimmte Bildausschnitte zu definieren und diese in einem beliebigen Koordinatenreferenzsystem und Format, das unterstützt wird, auszugeben. Wird einer der mitgelieferten Werte für die Anfrage aufgrund eines Fehlers nicht angenommen, wird statt den Daten eine XML Datei mit weiteren Informationen zurückgegeben. Neben der Bereitstellung der Daten, bietet der GeoServer auch Möglichkeiten die Daten und ihre Metadaten, die als XML Dateien im Speicherdepot des GeoServer abgelegt werden, zu adaptieren. Dafür steht ein Interface zur Verfügung welches alle Funktionen und deren Einstellungen beinhaltet, sei es für das Importieren und Exportieren von Raster oder Vektor Daten, das Setzen und Verändern von Metadaten, das Erstellen von Nutzerrollen für die Einschränkung von Rechten oder das Bearbeiten der Einstellungen welche die Nutzung des Arbeitsspeichers, Prozessoren und Festplattenspeicher betreffen.

Beim Importieren von Datensätzen erscheint jeder Datensatz als sogenannter Layer im Interface auf, welcher als eine Ebene zu verstehen ist. Bei eingehenden Web Service Anfragen, die mehrere Daten anfordern, werden die gewünschten Ebenen aufeinander gestapelt. Die Reihenfolge hängt vom zuständigen Parameter der Anfrage ab, der die Liste der angeforderten Layer enthält. Der GeoServer bietet auch eine Art Datencontainer wo mehrere Layer zusammengefasst werden. Diese Datencontainer werden als Layergruppen bezeichnet.

Des Weiteren bietet das Interface auch die Option den Daten ein Design zuzuweisen und sie zu visualisieren. Das Design wird sowohl für die Raster als als auch für Vektor Daten mit Hilfe des Styled Layer Descriptors SLD erstellt. SLD ist eine auf XML basierte Aufzeichnungssprache mit welcher Größe, Farbe und Beschriftung der Daten verändert werden kann. Für die Zwischenspeicherung und das Teilen der Daten in Kacheln ist das GIS-OSS Produkt namens GeoWebCache integriert.

Um die Funktionen und Einstellungen über eine andere Applikation aufzurufen, kann die REST Schnittstelle des GeoServers verwendet werden. Diese erfolgt über HTTP Anfragen welche in Kapitel 4.6.2.1 gemeinsam mit den Services erklärt werden.

Um den Funktionsumfang des GeoServer zu erweitern, können auch Plug-ins verwendet werden die bereits entwickelt worden sind und auf der Webseite von GeoServer download-bereit stehen. Diese sind einfach einzubinden und können beispielsweise bestimmte Datenbanken einbinden oder GDAL in den GeoServer integrieren um die Palette der unterstützten Datenformate zu vergrößern. Details zu GDAL können aus dem Kapitel 4.6.2.3 entnommen werden.

Aufgrund der steigenden Anzahl von Funktionen und somit auch der Komplexität wurden sowohl für den GeoServer also auch für die Plug-ins Dokumentationen und Anleitungen, die den Einstieg wesentlich erleichtern, erstellt. Allerdings sind diese nicht vollständig oder beschreiben eine ältere Version des GeoServers. In solchen Fällen muss mehr Zeit in die Recherche investiert werden.

4.6.1.2 Datenspeicherung und Zugriff

Der GeoServer legt die Metadaten der Layer als XML Dateien im Speicherdepot des GeoServers ab. Die XML Dateien besitzen ein vom GeoServer vordefiniertes Schema. Sollen nun die Metadaten eines Layer gelesen oder verändert werden, ist, wie in Kapitel 4.6.2.1.2 beschrieben, eine Anfrage zu senden. Beim Lesen wird die XML Datei für den jeweiligen Layer zurückgegeben und beim Verändern muss eine XML Datei mit entsprechenden Schema an den GeoServer gesendet werden. Stimmt das Schema nicht überein, wird ein Fehler zurückgegeben. Die Layerdaten, seien es Raster- oder Vektordaten werden ebenfalls im Speicherdepot des GeoServer abgelegt. Es gibt ein Plug-in für den GeoServer, das es erlaubt eine Verbindung zu einer Datenbank herzustellen und die dort liegenden Daten zu verwenden. Dies würde die Speicherung als XML obsolet machen. Laut (Knörchen *et al.* 2015) sollte sich die Speicherung dann aber nur auf die Metadaten beschränken, da die Speicherung der Layer in der Datenbank zu einer Geschwindigkeitsreduktion der Abfragen führt.

Für das zu entwickelnde Programm wird eine Lösung gewählt die keine Datenbank benötigt. Der Grund ist, dass die Java API GeoServer-Manager Funktionen bereitstellt um Metadaten der einzelnen Layer zu adaptieren. Dies soll den Aufwand selbst die XML Dateien zu generieren und mögliche zurückkommende Fehler zu behandeln ersparen.

Bei der Entwicklung wird jedoch klar, dass diese API nicht alle Funktionen des GeoServers abdeckt bzw. nicht so funktionieren wie erwartet. Die Funktionslücken waren vor allem im Bereich der Metadaten sehr auffällig. Folgende 3 Probleme verursachen einen deutlich höheren Aufwand mit dem nicht gerechnet wurde:

- Bei den Layern kann der Name nicht abgeändert werden.
- Bei den Metadaten der Layergruppen kann nur der Titel geändert werden.
- Selbstdefinierte Metadaten werden als ungültige XML gespeichert.

Die drei beschriebenen Probleme haben Auswirkungen auf die Logik der Kommunikation zwischen Client und Backend. Es muss eine eigene Lösung gefunden werden, die in das bereits entwickelte System eingebunden werden kann. Um diese Lösung zu finden ist daher ein höherer Aufwand nötig.

Der Layer- oder Layergruppenname ist der Index für die Suche nach einem Datensatz im GeoServer. Da der Name nicht geändert werden kann, muss der Titel des Datensatzes verwendet werden. Dieser wird aber von der Suchmaschine des GeoServers nicht beachtet. Das bedeutet wenn der Nutzer einen neuen Namen setzt, wird der Datensatz nicht mehr gefunden. Es könnte eine eigene Funktion geschrieben werden, die alle Datensätze abfragt, die Titeln herausfiltert und diese dann mit dem gesuchten Text vergleicht. Angesichts der Datenmenge verschlechtert diese Option die Performance der Applikation beträchtlich. Daher wird eine andere Lösung verwendet und im Folgenden beschrieben. Für den Namen wird eine einzigartige Identifikationsnummer gesetzt, welche für den Nutzer nicht sichtbar ist. Der Titel ist dagegen vom Nutzer frei wählbar. Der Client, also der Teil der Applikation, der im Browser läuft, erhält jedoch beides. Wie in Abbildung 4.7.2.2 in Kapitel 4.7.2 zu sehen, wird dem Nutzer zwar das Feld „Name“ angezeigt, in Wirklichkeit ist dies der Titel, der veränderbar ist. Sollte der Nutzer am Client die Metadaten ändern, schickt der Client neben den veränderten Metadaten auch die eindeutige Identifikationsnummer zurück, die es erlaubt direkt den richtigen Datensatz zu adaptieren.

Bei den Layergruppen wird dasselbe Prinzip verwendet. Da bei den Layergruppen nur der Titel veränderbar ist, werden noch zusätzliche Funktionen für das Adaptieren der restlichen Metadaten gebraucht und erstellt.

Der GeoServer speichert nicht alle Metadaten die gefordert sind, wie beispielsweise die Genauigkeiten des Layers oder wann und von wem der Layer erstellt worden ist. Daher müssen diese durch eigens definierte XML Tags in das Layer-Metadaten-Schema, das vom GeoServer vordefiniert ist, hinzugefügt werden. Die GeoServer-Manager API besitzt Funktionen die das ermöglichen sollte, allerdings werden neudefinierte Metadaten Tags mit einer falschen Syntax geschrieben. Somit werden sie vom GeoServer nicht angenommen. Auch hier müssen daher eigene Funktionen

erstellt werden in denen die benötigten XML Tags erstellt werden und in das vorhanden Schema eingebunden werden.

4.6.1.3 Zugriffs-Einschränkungen/Sicherheit

Der GeoServer bietet zwar auch ein User Management System mit verschiedenen Rollen und Rollengruppen an. Allerdings soll ein eigenes User Management System verwendet werden, das nicht vom GeoServer abhängt.

Des Weiteren sollen laut den Anforderungen Layergruppen für externe Applikationen über Services zugänglich sein. Der GeoServer bietet WMS, WFS und WPS an, welche im nächsten Kapitel 4.6.1.4 erläutert werden. Allerdings wird noch ein Proxy-Server zwischengeschaltet um die Anfragen abzufangen und nach anschließender Validierung entweder mit einer Fehlermeldung zurückzuweisen oder diese weiter zum GeoServer zu leiten. Da nur ganze Layergruppen veröffentlicht werden sollen und auch nur wenn der Nutzer diese über die Applikation als zu veröffentlichende markiert hat, werden automatisch Fehlermeldungen zurück gegeben, wenn einzelne Layer abgefragt werden.

4.6.1.4 Web Services (WMS, WFS und WPS)

4.6.1.4.1 WMS und WFS

Um Daten für externe Applikation sowie für die Karte die am Client angezeigt wird zur Verfügung zu stellen, liefert der GeoServer einen Web Mapping Service (WMS) und einen Web Feature Service (WFS) an. Während der WMS die Übermittlung von Rasterdaten erlaubt, ist es mit den WFS möglich Vektordaten zu übermitteln. Beide folgen den OGC Standards. Das Einrichten dieser Services kann mit einem geringeren Aufwand konfiguriert und eingesetzt werden als erwartet.

4.6.1.4.2 Höhenschichtlinien-Berechnung (WPS)

Für Berechnungen an Raster oder Vektordaten bietet der GeoServer auch Web Processing Services (WPS) an. Dabei werden die Berechnungen als Prozesse bezeichnet, da sie einerseits meist viele Ressourcen und damit Rechenzeit benötigen und andererseits die Möglichkeit bieten sie zu einer Kette zusammen zu führen. Wird so eine Kette in Gang gestartet bedeutet das, dass die Ergebnisse eines Prozesses anschließend als Eingangsdaten für den nächsten Prozess fungieren. Der WPS wird für die Berechnung der Höhenschichtlinien, die im nächsten Kapitel näher erläutert werden genutzt. Da jedoch die GeoServer Manager API keine Funktionen für das Starten der WPS Prozesse des GeoServers bietet, müssen diese von Grund auf selbst erstellt werden.

Die Höhenschichtlinienberechnung wird durch eine Anfrage vom Client an die Services im Backend gestartet. Dazu muss der Nutzer einige Parameter wie den Abstand zwischen den Höhenschichtlinien setzen und bestätigen. Auf das Interface der Höhenschichtlinienberechnung wird in Kapitel 4.7.4 eingegangen. Der Berechnungsvorgang wird als WPS Prozesskette implementiert. Der erste Prozess ist die Berechnung der Linien und der zweite Prozess ist das Einlesen in das Datendepot des GeoServers. Zusätzlich wird noch ein Style für die Höhenschichtlinien definiert und diesen zugewiesen, um diese auf der Karte als solche erkenntlich zu machen. In Abbildung 4.6.1.1 sind die Ergebnisse in der Karte der Webapplikation zu sehen. Dabei ist im Hintergrund der Rasterdatensatz, ein eingefärbtes Digitales Geländemodell DGM, zu sehen aus dem die Höhenschichtlinien berechnet worden sind. Durch das Styling wird das Intervall abhängig vom Maßstab geändert und die passende Beschriftung.

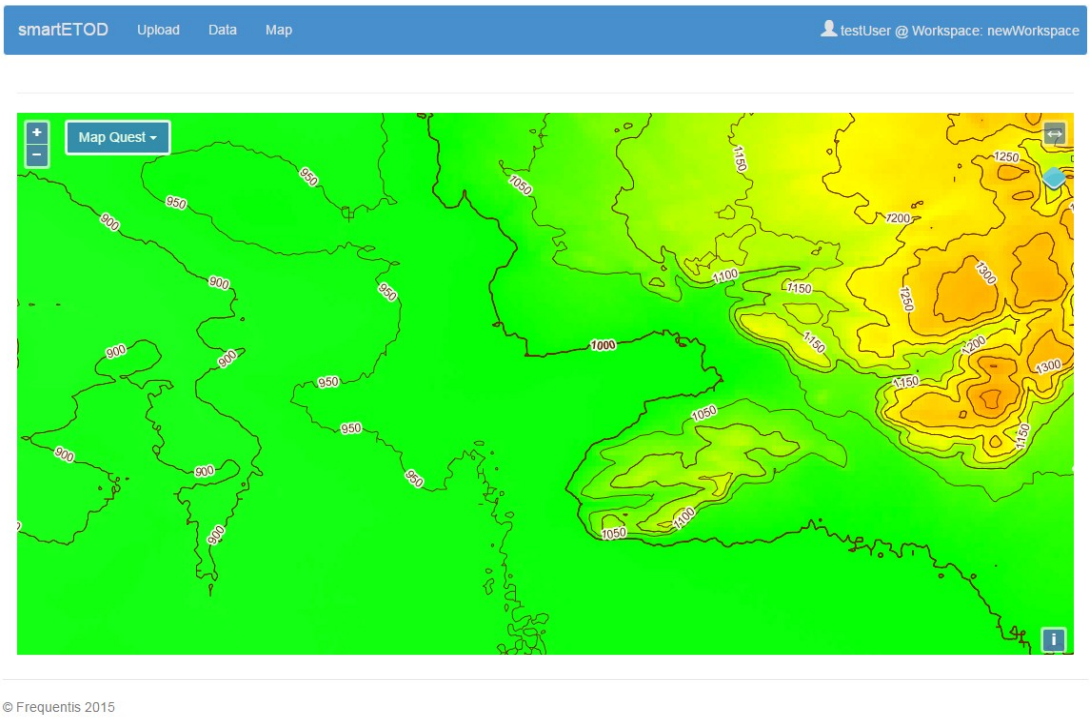

Abbildung 4.6.1.1: Berechnete und gestylte Höhenschichtlinien auf einem gestylten DGM

Beim Testen sind Probleme bei der Berechnung aufgetreten. Bei flachen Bereichen entstehen aufgrund des verwendeten Algorithmus des GeoServers eigenartige Höhenschichtlinien, wie in Abbildung 4.6.1.2 zu sehen. Da dieses Problem bereits in der GeoServer-Gemeinschaft bekannt ist und die Berechnung für eine große Nutzerzahl wichtig erscheint, wird das Problem innerhalb kurzer Zeit behoben. Damit

musste nur auf das nächste Update gewartet werden, das innerhalb der geplanten Entwicklungszeit des eTOD Produktes erscheint.

Ein weiteres Problem ist, dass bei großen Rasterdaten die Berechnung deutlich länger dauert oder nicht abgeschlossen wird. Um den Grund herauszufinden müsste der Quellcode des GeoServer untersucht werden. Da jedoch die OSS Komponente GDAL bereits implementiert ist und ebenfalls die Funktion zur Berechnung von Höhenschichtlinien bietet, wird diese getestet. Das Resultat ist, dass die Berechnung deutlich schneller durchgeführt wird als über die Prozesse des GeoServers. Der Umstieg von der Nutzung des WPS zu GDAL erforderte nur einen geringen zusätzlichen Aufwand.

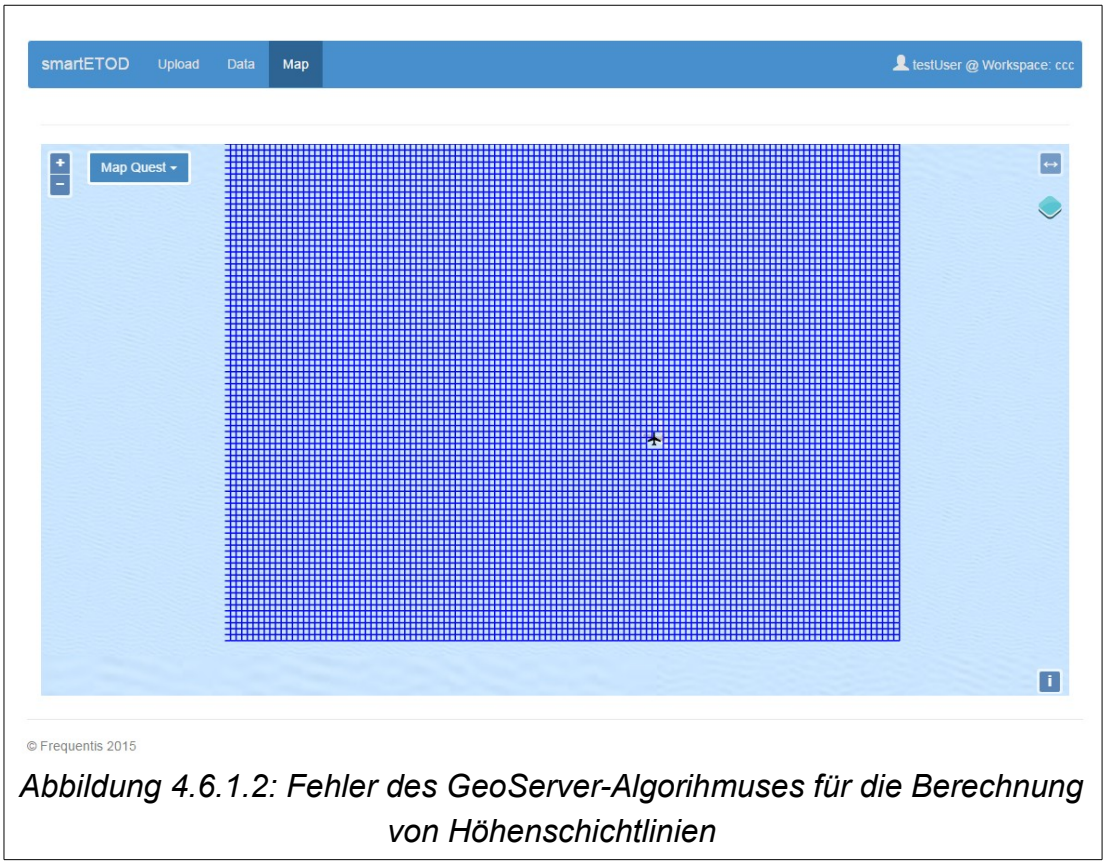

Abbildung 4.6.1.2: Fehler des GeoServer-Algorihmuses für die Berechnung von Höhenschichtlinien

4.6.2 Services

Die Services sind in Java programmiert und können als Ansammlung an Funktionen angesehen werden, die vom Client durch eine Anfrage gestartet werden können. Welche Anfrage jeweils gesendet wird, hängt von der Interaktion zwischen Nutzer und Client ab. Diese Funktionen verarbeiten die Anfragen und senden, wenn erforderlich, Anfragen an die anderen Backend Komponenten. Meistens liefern diese Komponenten Daten zurück, die dann anschließend von den Services aufbereitet und in eine Form gebracht werden die vom Client verwendet werden können und

leitet sie schließlich zum Client weiter. Die hier grob dargestellte Kommunikation verläuft über REST-Dienste, welches in folgendem Kapitel 4.6.2.1 näher erläutert werden.

Neben der Hauptaufgabe der Kommunikation spielt die Datenverarbeitung ebenfalls eine wichtige Rolle in den Services. Denn obwohl der GeoServer die Datenkonvertierungsbibliothek GDAL verwendet, unterstützt der GeoServer nur bestimmte Datenformate. Daher ist die direkte Nutzung von GDAL notwendig. Die Implementierung und Nutzung wird in Kapitel 4.6.2.3 beschrieben. Bevor bestimmte Daten aber verarbeitet werden, wird eine vorangestellte Versionskontrolle durchgeführt. Diese wird verwendet um sicherzugehen, dass nur die aktuellsten Daten von einem Nutzer verändert werden. Auf die Umsetzung dieser Versionskontrolle wird in Kapitel 4.6.2.2 näher eingegangen.

Zu guter Letzt werden die Services auch für die Speicherung des Status eines oder mehrerer gestarteter und laufender Prozesse eines Nutzers gebraucht, da die Prozesse eine längere Prozessierungszeit benötigen. Diese Speicherung wird als Caching bezeichnet und wird im Kapitel 4.6.2.4 näher erläutert.

4.6.2.1 REST Web Services

REST steht für „Representational State Transfer“ und beschreibt einen Architekturstil bei einer Maschine zu Maschine Kommunikation via „Hypertext Transfer Protocol“ (HTTP). Es funktioniert wie folgt. Eine Maschine verhält sich passiv und wartet auf Anfragen die über HTTP an diese gesendet werden. Die andere Maschine verhält sich dagegen aktiv. Das heißt, dass die aktive Maschine die Anfragen sendet sobald etwas benötigt oder durchgeführt werden soll. Die passive Maschine verarbeitet die Anfrage und liefert dann eine Antwort zurück. Daher erhält die aktive Maschine nur den Datenzustand zur Zeit der Abfrage. Werden beispielsweise Daten auf der passiven Maschine geändert, so muss die aktive Maschine eine weitere Anfrage schicken um diese Änderung zu erhalten.
Betrachtet man nun den vorliegenden Fall der eTOD Applikation, so ist der Client als die aktive Maschine, während die Services als die passive Maschine zu sehen. Das bedeutet, dass sobald ein Nutzer der Applikation Daten betrachten oder verändern möchte, muss der Client eine Anfrage an die Services senden. Die Services stellen für jede aufrufbare Funktion eine eigene Adresse bereit, die als „Uniform Resource Identifier“, kurz URI, bezeichnet wird. Über diese Adresse kann der Client die jeweilige Funktion über eine Anfrage in Form einer HTTP Nachricht ansprechen. Die Nachrichten müssen jedoch alle Informationen beinhalten, die für die Ausführung notwendig sind. Die REST Anfragen besitzen keine besondere Struktur. Sie sind einfache HTTP Nachrichten die die vorgegebenen Methoden GET, POST, PUT und

DELETE verwenden. Sollte eine Anfrage eine falsche URI oder Methode verwenden, verarbeiten die Services die Anfrage nicht und der Client erhält eine Fehlermeldung.

Die Umsetzung der Services erfolgte in Java unter der Verwendung der Bibliothek Jersey die die Erstellung von REST Web Services erlaubt. Da die Services sowohl mit dem Client als auch mit dem GeoServer kommunizieren, wird auf die speziell in den nächsten beiden Kapiteln 4.6.2.1.1 und 4.6.2.1.2 eingegangen.

4.6.2.1.1 Kommunikation mit dem Client

Wie bereits im vorangegangenen Kapitel beschrieben, warten die Services auf Anfragen vom Client. Aufgrund der unterschiedlichen Funktionen die über das Interface des Clients dem Nutzer angeboten werden, sind auch die Services diesen angepasst. Möchte ein Nutzer Daten auf den Server laden so werden einerseits die Daten und andererseits die Metadaten, die nötig sind um die Daten an den GeoServer zu senden und somit einen Layer zu erstellen, geliefert. Die Metadaten werden dabei über den Client untersucht um festzustellen ob diese valide sind. Der Name wird jedoch über eine eigene HTTP Anfrage von den Services kontrolliert, sobald der Nutzer den Namen im Web Interface angibt. Grund dafür ist, dass der Name kontrolliert werden sollte bevor die Daten samt Metadaten an den Server geschickt werden, da dies bei größeren Daten oder unzureichender Internetverbindung mehr Zeit beanspruchen kann.

Beim Daten Management, also beim Bearbeiten der Daten, funktioniert es sehr ähnlich. Doch gibt es hier einen Unterschied. Wenn ein Nutzer Metadaten verändern möchte, kann nicht davon ausgegangen werden, dass die Daten die der Nutzer über das Web Interface sieht, aktuell sind. Es ist nämlich zu bedenken, dass das Programm von mehreren Nutzern gleichzeitig genutzt werden kann. Das bedeutet, dass mehrere Clients zeitgleich eine Anfrage an die Services senden können, die ebenfalls zeitgleich ausgeführt werden. Dies bedeutet aber auch, dass in demselben Moment mehrere Nutzer Daten verändern können. Um die Konflikte zwischen den Versionen von unterschiedlichen Nutzern zu vermeiden ist in den Services eine Versionskontrolle eingebaut. Detaillierter auf die Versionskontrolle wird in 4.6.2.2 eingegangen. Sollten bei der Versionskontrolle Fehler auftreten, wird die bearbeitete Version des jeweiligen Nutzers nicht übernommen und der Nutzer wird darüber informiert, dass es eine neue Version zur Verfügung steht. Dabei wird auch die neueste Version der Metadaten für den jeweiligen Layer nochmals geladen und im Interface angezeigt.

4.6.2.1.2 Kommunikation mit dem GeoServer

Die Kommunikation mit dem GeoServer erfolgt ebenfalls auf Basis von REST mittels HTTP Nachrichten mit der entsprechenden Methode (Kapitel 4.6.2.1). Allerdings fungieren diesmal die Services als die aktive Einheit und der GeoServer als die passive Einheit die auf die Anfragen wartet. Die REST Schnittstelle beim GeoServer ist bereits implementiert und muss nicht programmiert werden.

Verwendet wird die Schnittstelle für verschiedene Bereiche. Daten können hochgeladen werden, Metadaten gesetzt oder verändert, sowie Prozesse gestartet werden. Die Metadaten werden in einem vordefinierten XML Schema im GeoServer gespeichert und die gesendeten Daten müssen diesem geforderten Schema entsprechen. Ist das der Fall werden die Daten übernommen bzw. verändert. Entsprechen sie diesem Schema nicht, wird eine Fehlermeldung zurückgegeben. Für die vereinfachte Nutzung der REST Schnittstelle des GeoServers wird die Java API GeoServer-Manager verwendet. Diese bietet Funktionen, die XML Kommandos mit Hilfe der mitgegebenen Parameter erstellen und an den GeoServer senden. Da die API nicht alle Funktionalitäten des GeoServers abdeckt, müssen diese um eigens entwickelten Funktionen erweitert werden. Das schließt zum einen das Speichern und Verändern bestimmter Metadaten und zum anderen das Starten und Kontrollieren von Prozessen mit ein.

Bei den Metadaten ist es beispielsweise nicht möglich die Namen von Layern oder Layergruppen zu ändern. Bei den Layergruppen lässt sich sogar nur der EPSG Code, also das Koordinatenreferenzsystem, verändern. Für die Verwendung des WPS des GeoServers bietet die API überhaupt keine Möglichkeit. Daher müssen dafür ebenfalls Funktionen erstellt werden. Dies erfordert einen zusätzlichen Aufwand mit dem nicht gerechnet wurde. Die Lösungsansätze für diese Probleme werden in Kapitel 4.6.1.2 dargelegt.

Neben der REST Schnittstelle werden auch der Web Mapping und Web Feature Service des GeoServers verwendet um die eigentlichen Daten, die für die Karte oder externe Applikationen gebraucht werden, abzufragen. Im Falle eines Zugriffs durch eine externe Applikation wird die Abfrage über einen Proxy Server an den GeoServer gesendet und schließlich dort verarbeitet.

4.6.2.2 Versionskontrolle

Aufgrund dessen, dass die REST Abfragen nur den momentanen Stand der Daten oder Metadaten abfragen können und der Nutzer diesen Zustand über das Interface vor Augen hat bis die nächste Abfrage durchgeführt wird, kann zu der Situation

führen, dass zwischen den Abfragen eins Nutzer der an den Metadaten arbeitet ein anderen Nutzer die Metadaten bereits verändert hat. Dies könnte zu dem folgenden Problem führen:

Nutzer A und Nutzer B starten die Applikation und bekommen den momentanen Stand der Daten und Metadaten die am Server gespeichert sind. Nutzer A und B beginnen beide einen bestimmten Datensatz zu ändern. Nutzer A speichert die Änderungen während Nutzer B noch daran arbeitet. Dies unterbricht den Nutzer B nicht, da dieser an dem letzten Stand des Datensatzes, der geladen wurde, arbeitet. Möchte der Nutzer B nun den bearbeiteten Datensatz ebenfalls speichern, so würde er die Daten die bereits von Nutzer A verändert worden sind überspielen ohne zu wissen, dass Nutzer A eine neue Version gespeichert hat.

Da der GeoServer keine Versionskontrolle besitzt muss das Problem anders umgangen werden. Die Lösung die hier eingesetzt wird, ist ein Zeitstempel für die letzte Änderung. Dieser Zeitstempel wird als eigener Metadaten-Tag in den Metadaten des jeweiligen Layers im GeoServer eingetragen. Im Kapitel 4.6.1.2 wird beschrieben wie neue Metadaten-Tags im GeoServer gespeichert werden können. Dieser Zeitstempel wird auch dem Client geliefert. Werden die Metadaten eines Layers verändert so wird kontrolliert ob der Zeitstempel, den der Nutzer besitzt, dem des Layers im GeoServer entspricht. Ist das der Fall, wird die Änderung akzeptiert, ansonsten wird eine Fehlermeldung an den Client geliefert samt dem neuen Stand der Daten.

4.6.2.3 Einbindung von GDAL

Da die eTOD Applikation Datenformate einbinden soll, die der GeoServer nicht unterstützt, wird die Open Source Bibliothek GDAL zur Konvertierung der Daten verwendet. Zwar gibt es ein GDAL Plug-in für den GeoServer, das auf der gleichnamigen Bibliothek basiert, jedoch unterstützt dieses Plug-in nicht alle benötigten Datenformate. Werden die Formate betrachtet, die laut den Anforderungen von dem geplanten eTOD System unterstützt werden sollen, so beschränkt sich dies auf BIL und DTED Formate. Während das angebotenen Plug-in die Nutzung von DTED Dateien erlaubt, müssen Daten im BIL Format über andere Wege konvertiert werden.
Die Implementierung der GDAL Bibliothek in die Services ist unkompliziert und ermöglicht das Aufrufen der GDAL Funktionen aus Java heraus. Dazu ist zuerst GDAL als Kommandozeilenprogramm herunterzuladen und auf dem Server zu installieren. Die Pfade der zu installierenden GDAL Version müssen in den Umgebungsvariablen des Betriebssystems gesetzt werden. Nun ist es möglich die Befehle aus Java an GDAL zu senden und etwaige zurückkommende Meldungen

beispielsweise in eine Log Datei zu speichern oder falls es sich um eine Fehlermeldung handelt, weiter an den Client zu senden um diesen von etwaigen Problemen zu informieren. Bei der Installation kann es zu Problemen kommen, wie im Falle der eTOD Applikationsentwicklung. Zum einen können die Pfade in den Umgebungsvariablen des Betriebssystems falsch gesetzt sein und zum anderen können Konflikte mit anderen Versionen von GDAL auftreten falls andere vorhanden sind.

Möchte ein Nutzer nun Daten in die eTOD Applikation hochladen, die vom GeoServer nicht unterstützt werden, werden die Daten nach dem Hochladen vom Client zum Server in ein GeoTiff konvertiert. Dieses Format wird vom GeoServer unterstützt. Der Befehl für die Konvertierung einer BIL Datei sieht wie folgt aus (Abbildung 4.6.2.1):

```
gdal_translate -co "COMPRESS=LZW" D:\testdata.bil D:\testdata.tif
```

Abbildung 4.6.2.1: gdal_translate wrid für die Konvertierung der BIL Daten verwendet

„gdal_translate“ ist die verwendete GDAL Funktion und die anderen Angaben bilden die Parameter für die Funktion. „-co“ erlaubt die Angabe einer oder mehrerer Ausgabe Optionen. In diesem Fall wird die Art der Kompression angegeben. Die beiden letzten Parameter sind die Pfade zu der zu verwendeten Datei und zu der neuen Datei.
Für die Konvertierung von DTED Dateien wird eine andere Funktion verwendet. Ähnlich wie bei „gdal_translate“ ist es möglich die Daten von einem DTED Format in ein GeoTiff Format umzuwandeln, allerdings besteht auch die Möglichkeit kleinere Datensätze zu einem größeren zusammenzufügen. Dies geschieht durch die Angabe von mehreren Pfaden zu den zu einlesenden Dateien.

```
gdalwarp D:\e008\n43.dt0 D:\e008\n44.dt0 D:\e009\n43.dt0 D:\e009\n44.dt0 D:\testdata.tif
```

Abbildung 4.6.2.2: gdalwarp wird für die Konvertierung und Zusammenfügen der DTED Daten verwendet

In Abbildung 4.6.2.2 ist ein Beispiel des Befehls zu sehen. „gdalwarp“ ist wieder die verwendete GDAL Funktion. Die ersten vier Pfade nach der Funktion sind die DTED Daten die verwendet werden sollen. Die Anzahl der angegeben Pfade darf dabei variiert werden. Für die eTOD Applikation wird die Angabe daher angepasst anhängig davon wie viele Dateien vom Client zum Server gesendet wurden. Der letzte und in diesem Fall der fünfte Pfad führt zu der zu erstellenden Datei.

4.6.2.4 Cache für Prozesse für die Berechnungen der Höhenschichtlinien

Wie bereits in den Anforderungen in Kapitel 4.1 beschrieben, ist die zu entwickelnde GIS Applikation mit einem UMS zu verbinden. Jeder Nutzer hat die Möglichkeit Höhenschichtlinien zu berechnen. Der Vorgang wird in Kapitel 4.6.1.4.2 erklärt. Da diese Berechnungen eine längere Prozessierungsdauer benötigen, muss eine Lösung gefunden werden um nur die Prozesse dem Nutzer zu zeigen, der sie gestartet hat. Dafür wird eine Cache erstellt, also eine Art Speicher, der für jeden begonnen Prozess einen Eintrag mit der Information, welcher Nutzer diesen gestartet hat sowie einem Status, der darüber informieren soll, ob der Prozess abschlossen ist, noch läuft oder aufgrund eines Fehlers abgebrochen werden musste. Die Liste dieser Prozesse wird dem jeweiligen Nutzer im Web Interface angezeigt, welche kontinuierlich auf dem neuesten Stand gehalten wird. In Abbildung 4.7.3.1 im Kapitel 4.7.3 ist unten die Anzeige der Prozesse zu sehen. Dank der Speicherung der Cache-Prozessliste am Backend kann der Nutzer auch die Web Applikation jederzeit schließen und starten ohne, dass die Information verloren geht.

4.7 Frontend Entwicklung

Während das Backend alle serverseitigen Komponenten umfasst, bildet das Frontend alle Komponenten die auf dem Gerät des Nutzers laufen. Neben dem Web Interface mit dem der Nutzer interagiert beinhaltet dieses auch die Kommunikation mit dem Server sowie die Verarbeitung der Daten die von diesem Aufgrund von Anfragen zur Verfügung gestellt werden. Da es sich bei dem zu entwickelnden eTOD Programm um eine Webapplikation handelt, benötigt der Nutzer lediglich einen Browser und muss keine Inhalte installieren. Da der Browser sowohl auf einem Mobilen Gerät als auch auf einem Standgerät laufen kann, wird auch die Darstellung an die Bildschirmgröße angepasst.

Die Applikation besteht aus mehreren Unterseiten, die jeweils gewisse Funktionsbereiche widerspiegeln. Die Unterseiten könne über die Navigation erreicht werden. Im Folgenden wird auf jede Unterseite einzeln eingegangen. Alle beinhalten Funktionen die eine gewisse Interaktion mit dem Backend erfordern, sei es das Abfragen von Daten oder die Nutzung von Funktionen. Um Fehler im Backend zu reduzieren, werden jegliche Eingaben des Nutzers bereits am Client kontrolliert.

Der Aufwand ist deutlich geringer als im Backend, da im Backend einige unerwartete Probleme auftreten. Die verwendeten Framework im Frontend sind gut dokumentiert

und haben wie beispielsweise Open Layers eine große Gemeinschaft die bei Problemen gerne und schnell hilft.

4.7.1 Startseite und Workspaceerstellung

Nachdem sich der Nutzer in die Applikation eingeloggt hat, kommt dieser auf die in Abbildung 4.7.1.1 gezeigt Startseite. Neben der Navigation, die auf die Unterseiten leitet und immer zu sehen ist, enthält die Startseite auch die Möglichkeit Workspaces zu erstellen und bereits erstellte zu wählen. Ein Workspace ist ein vom GeoServer definierter Container für hochzuladende Daten. In der etod Applikation sind diese Container, sowie deren Daten, nur für die jeweilige Nutzergruppe zugänglich, die sie erstellt hat. Somit ist es möglich die Dienste mehreren Kunden bereit zu stellen. Sobald der Nutzer ein Workspace gewählt hat, wird dieser mit dem Namen des Nutzers in der Navigationsleiste angegeben.

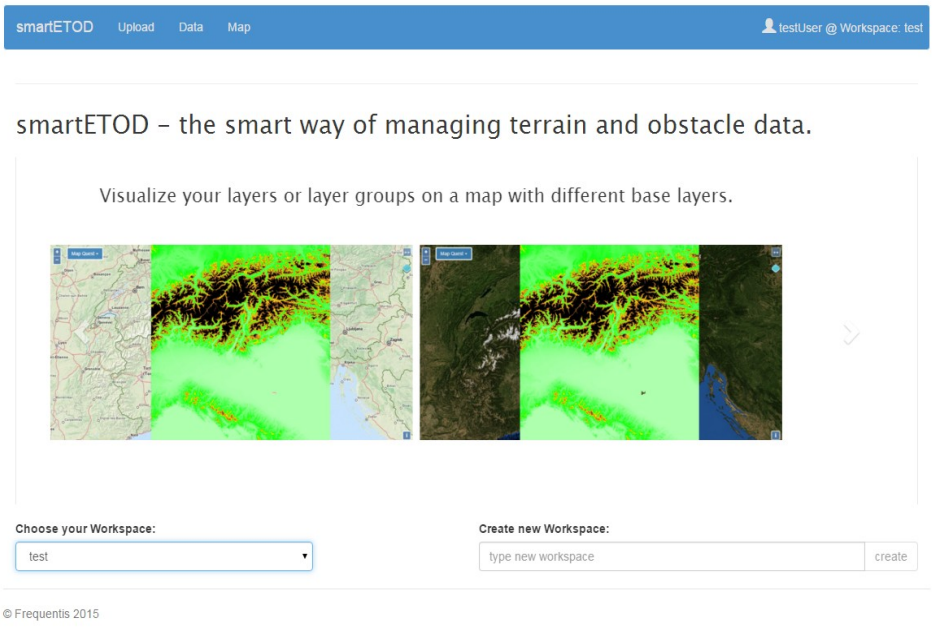

Abbildung 4.7.1.1: Startseite der etod Applikation mit wechselnden Einführungsbildern und Felder für das Erstellen oder Wählen von Workspaces

4.7.2 Hochladen von Daten

Um den Workspace mit Daten zu füllen muss auf die Unterseite „Upload“ gewechselt werden, die in Abbildung 4.7.2.1 zu sehen ist. Diese Seite ermöglicht dem Nutzer, also Rasterdaten im Format von BIL oder DTED hochzuladen. Da das Paradigma

unterstützt wird, dass nur die Inhalte sichtbar sein sollen, welche gerade gebraucht werden, sieht der Nutzer zu Beginn nur das Aufklappmenü für das Datenformat. Klickt der Nutzer auf die blaue Informations-Schaltfläche, bekommt dieser eine Liste der unterstützten Datenendungen. Nach der Auswahl der Daten werden weitere Felder sichtbar. Der Nutzer kann nun die grundlegenden Metadaten setzen, die für das Erstellen des sogenannten Layers am GeoServer notwendig sind. Wie in Abbildung 4.7.2.2 zu sehen, werden die Nutzereingaben validiert. Bei Fehlern wird das Feld markiert und der Nutzer informiert.

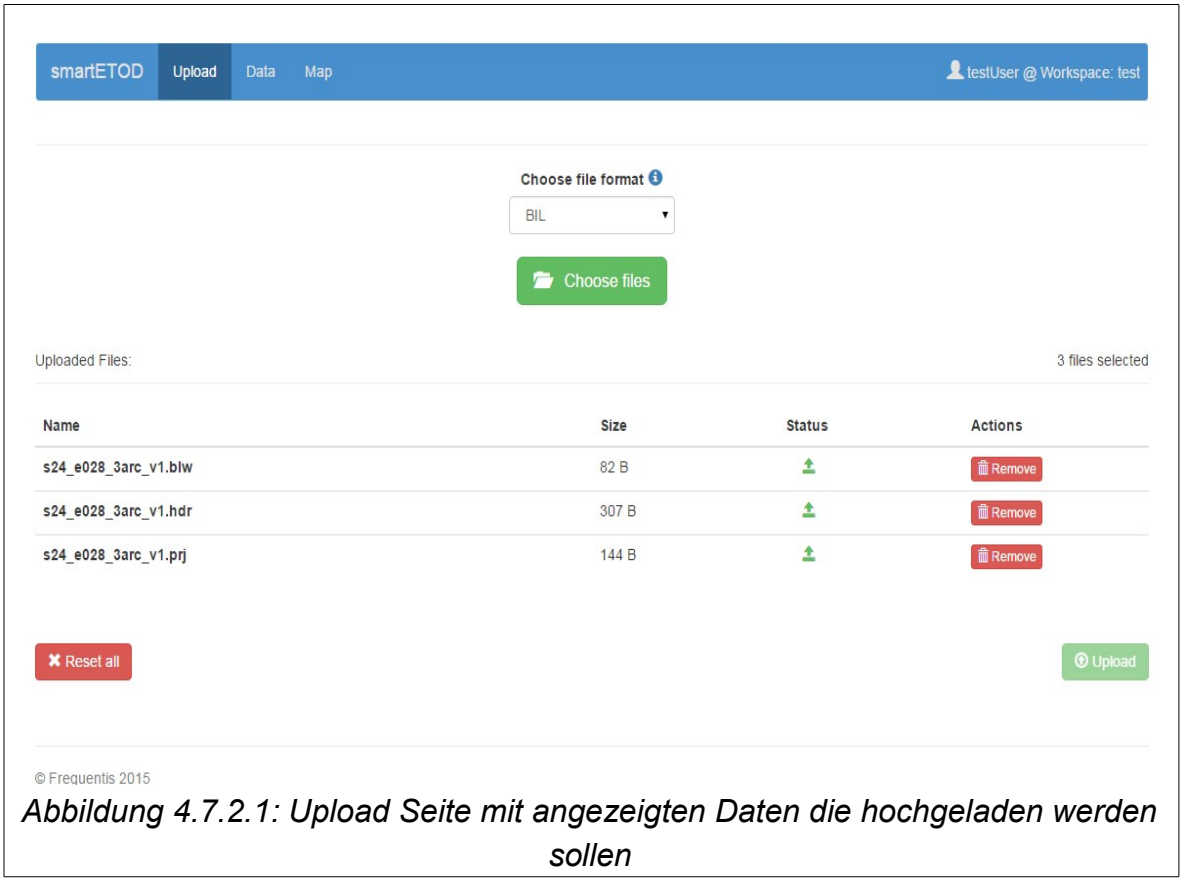

Abbildung 4.7.2.1: Upload Seite mit angezeigten Daten die hochgeladen werden sollen

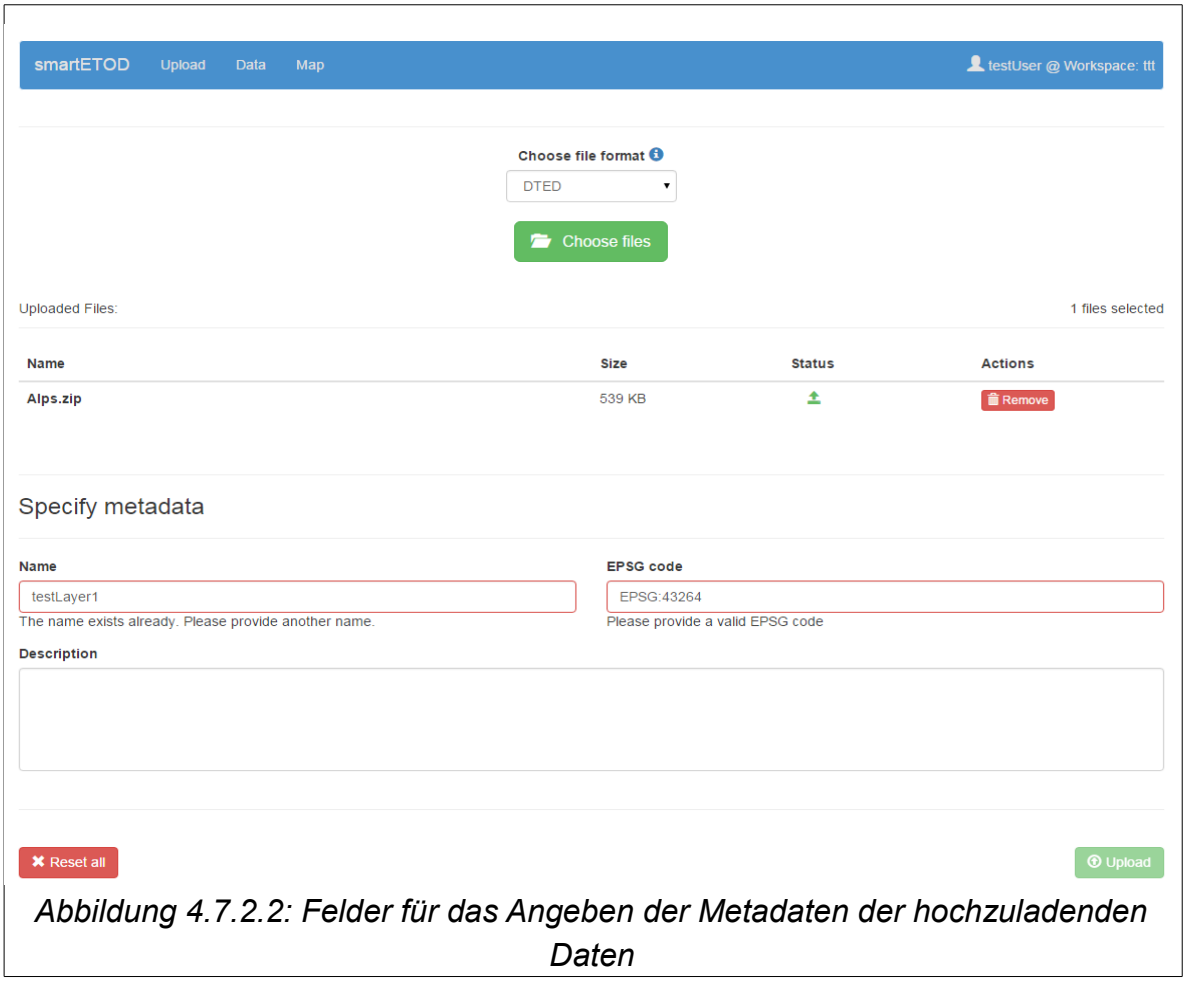

Abbildung 4.7.2.2: Felder für das Angeben der Metadaten der hochzuladenden Daten

4.7.3 Daten- und Metadatenmanagement

Beim Klicken des Reiters „Data“ in der Navigationsleiste öffnet sich die Unterseite für das Daten- und Metadatenmanagement, welche die Layer und Layergruppen, die im System gespeichert sind, sowie deren Metadaten darstellt und Möglichkeiten bietet die Metadaten zu verändern. Abbildung 4.7.3.1 zeigt die beschriebene Seite. Weiter oben in der Abbildung sind zwei Reiter mit der Bezeichnung „Layers“ und „Layergroups“ zu sehen. Beide dieser Unterseiten sehen sich vom Aufbau her ähnlich.

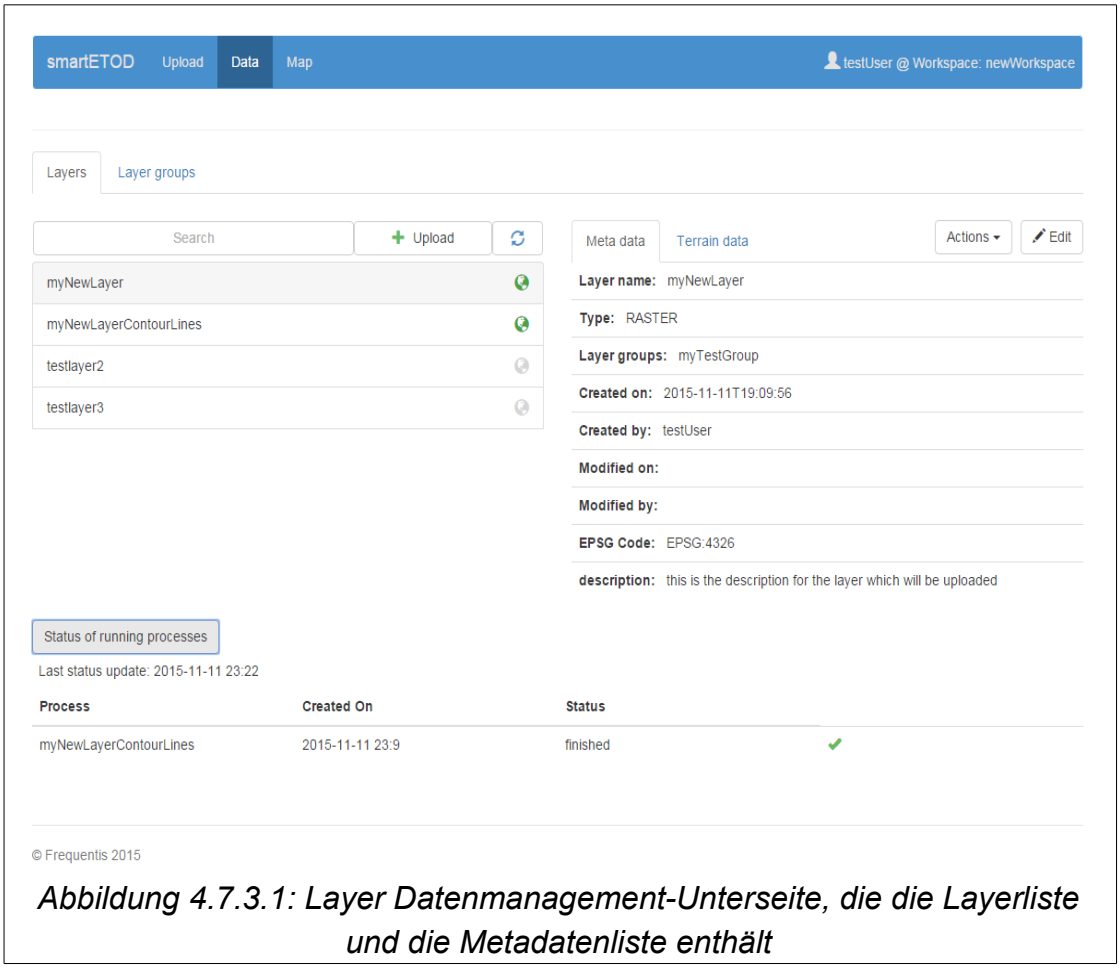

Abbildung 4.7.3.1: Layer Datenmanagement-Unterseite, die die Layerliste und die Metadatenliste enthält

In Abbildung 4.7.3.1 ist der Reiter „Layers“ aktiv. Auf der linken Hälfte der Unterseite ist die Liste der Layer zu sehen, die über die Suchleiste durchsucht werden kann. Daneben ist eine Schaltfläche mit der Beschriftung „Upload“ die direkt zu der in Kapitel 4.7.2 vorgestellten Upload-Unterseite führt. Jeder Layer in der Layerliste enthält eine Globus-Symbol. Wird dieser gedrückt, wird der Layer zur Karte hinzugefügt. Sobald in der Liste auf einen Layer geklickt wird, erscheinen auf der rechten Hälfte der Seite die Metadaten. Über den angezeigten Metadaten finden sich auch zwei weitere Reiter sowie zwei Schaltflächen. Die Reiter teilen die Metadaten auf. Die Abbildung 8.4.2 im Anhang 8.4 zeigt dabei den Inhalt des zweiten Reiters. Unter „Actions“ finden sich alle Aktionen die mit dem Layer durchgeführt werden können. Im momentanen Status der Applikation können nur die Höhenschichtlinien berechnen werden. Das Web Interface zu dieser Berechnung wird in Kapitel 4.7.4 beschrieben. Wird auf die Schaltfläche „Edit“ gedrückt, können die Metadaten, wie in Abbildung 4.7.3.2 zu sehen, verändert werden.

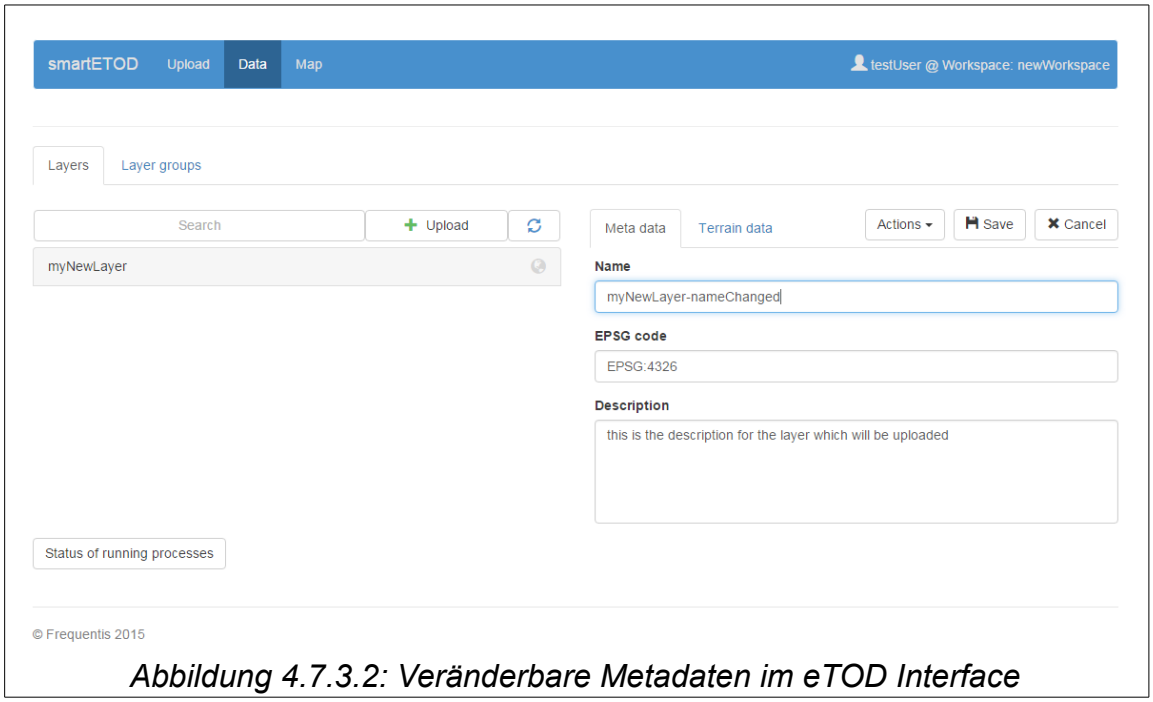

Abbildung 4.7.3.2: Veränderbare Metadaten im eTOD Interface

In Abbildung 8.4.3 im Anhang zeigt die erscheinende Unterseite wenn der Reiter „Layergroups“ aktiv ist. Statt der „Upload“ - Schaltfläche befindet sich nun die „Create“ Schaltfläche, welche auf die Möglichkeit eine Layergruppe zu erstellen hindeutet. Das sich dabei öffnende Fenster mit dem auszufüllenden Formular ist in Abbildung 4.7.3.3 zu sehen. Die Layer für die Gruppen können ausgewählt und ein Name vergeben werden. Sobald der Nutzer das Formular abschickt, wird die Anfrage von den Services im Backend angenommen, validiert und zum GeoServer zur Prozessierung weitergeleitet. Anschließend erscheint die Layergruppe, wie bei den Layern, in der Liste. Die Metadaten der Layergruppe können ebenfalls verändert werden. Wird, wie auf der Layer Unterseite, auf die „Edit“ Schaltfläche gedrückt, lassen sich die Metadaten verändern. Abbildung 4.7.4.1 zeigt wie das Nutzer Interface nach Betätigung der Schaltfläche aussieht. Neben den Feldern „Name“, „EPSG Code“ und „Description“, die auch bei den Layern zu sehen sind, sind auch die Layer, die in der Layergruppe sind, aufgelistet sowie eine Schaltfläche um die Layer hinzuzufügen oder zu entfernen. Der Schieberegler neben der Bezeichnung „Visible“ dient zur Festlegung ob die Layergruppe und damit die Liste der Layer über WMS abfragbar sind. Auf der rechten Seite außerdem noch zwei weitere Schaltflächen, die für das Speichern der Änderungen sowie das Abbrechen des Editierungsvorgangs.

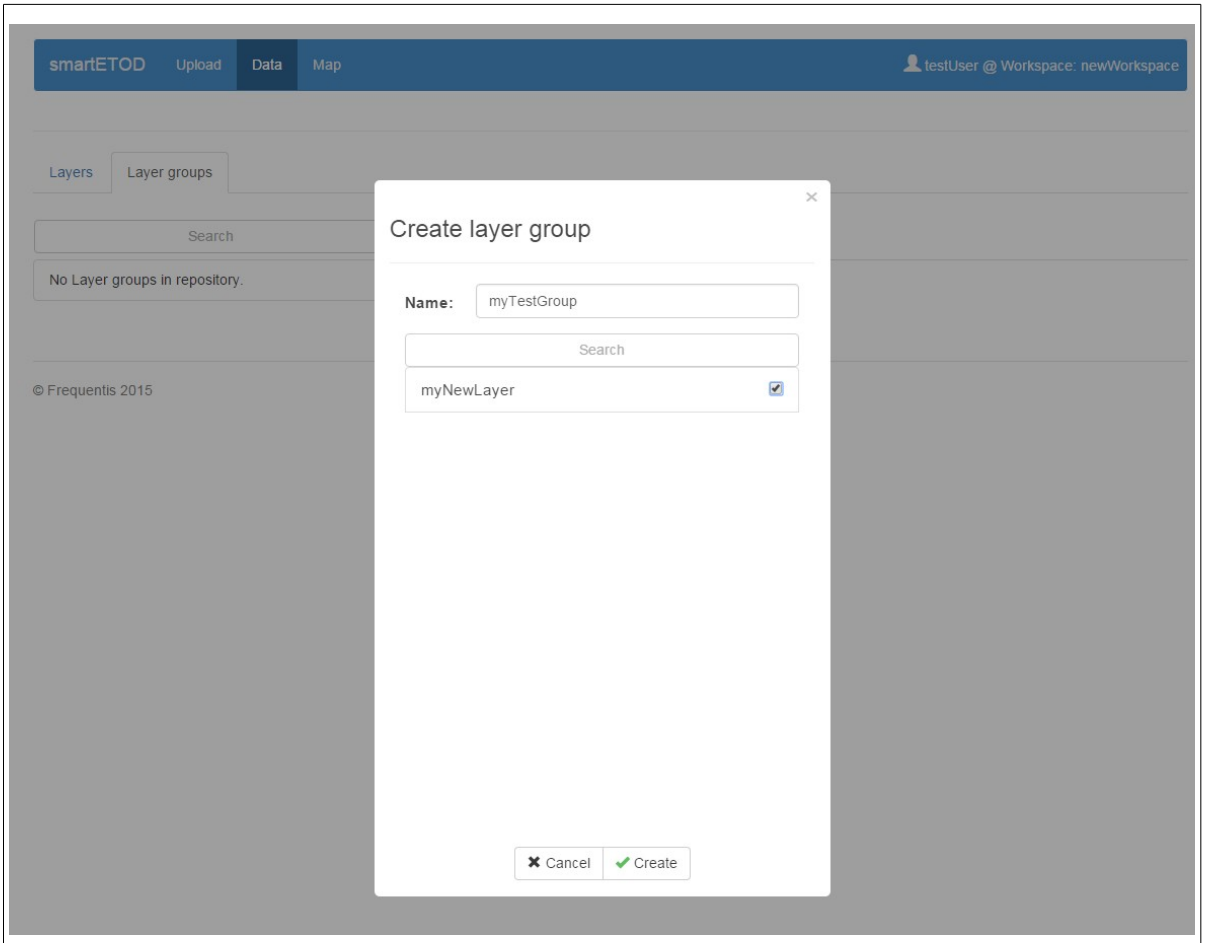

Abbildung 4.7.3.3: Fenster für die Erstellung von Layergruppen

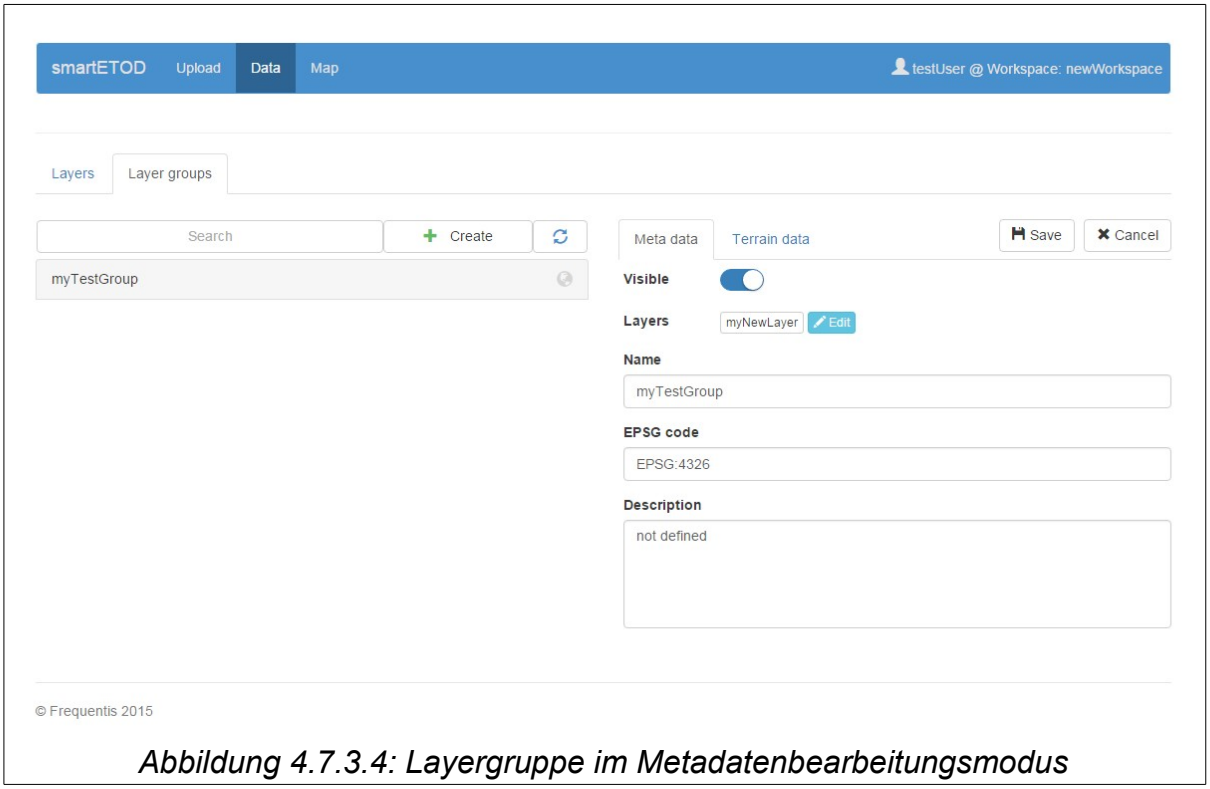

Abbildung 4.7.3.4: Layergruppe im Metadatenbearbeitungsmodus

4.7.4 Höhenschichtlinien berechnen

Für die Berechnung der Höhenschichtlinien, wie sie in Kapitel 4.6.1.4.2 beschrieben wird, ist auch ein Nutzer Interface zur Verfügung zu stellen, durch das der Nutzer die für die Berechnung erforderlichen Eingaben tätigen kann. Da diese Höhenschichtlinien auf einzelne Layer berechnet werden, wird das Nutzer Interface in die Layer Metadaten Unterseite gelegt. Der Nutzer erreicht es über die „Actions“ Schaltfläche, die in Abbildung 4.7.3.1 zu sehen ist. Nach der Betätigung der Schaltfläche öffnet sich ein kleineres Fenster. Neben dem Namen und der Beschreibung, lässt sich auch das Intervall für die Höhenschichtlinien setzen. Drückt der Nutzer nach der Eingabe auf die in Abbildung 4.7.4.1 zu sehende Schaltfläche „Submit“, werden die Daten an die Services im Backend gesendet, wo, wie in die Kapitel 4.6.1.4.2 beschrieben, der Prozess für die Berechnung gestartet wird sowie im Cache (Kapitel 4.6.2.4) gespeichert wird, damit der Nutzer jederzeit den Status der gestarteten Berechnung im Nutzer Interface abfragen und kontrollieren (Abbildung 4.7.3.1) kann. Das Resultat jeder Berechnung ist ist eine Ansammlung an Linien, die als Vektorlayer im GeoSserver gespeichert werden. Werden diese Daten für die Karte oder für externe Applikationen benötigt können sie mit dem WFS abgefragt werden.

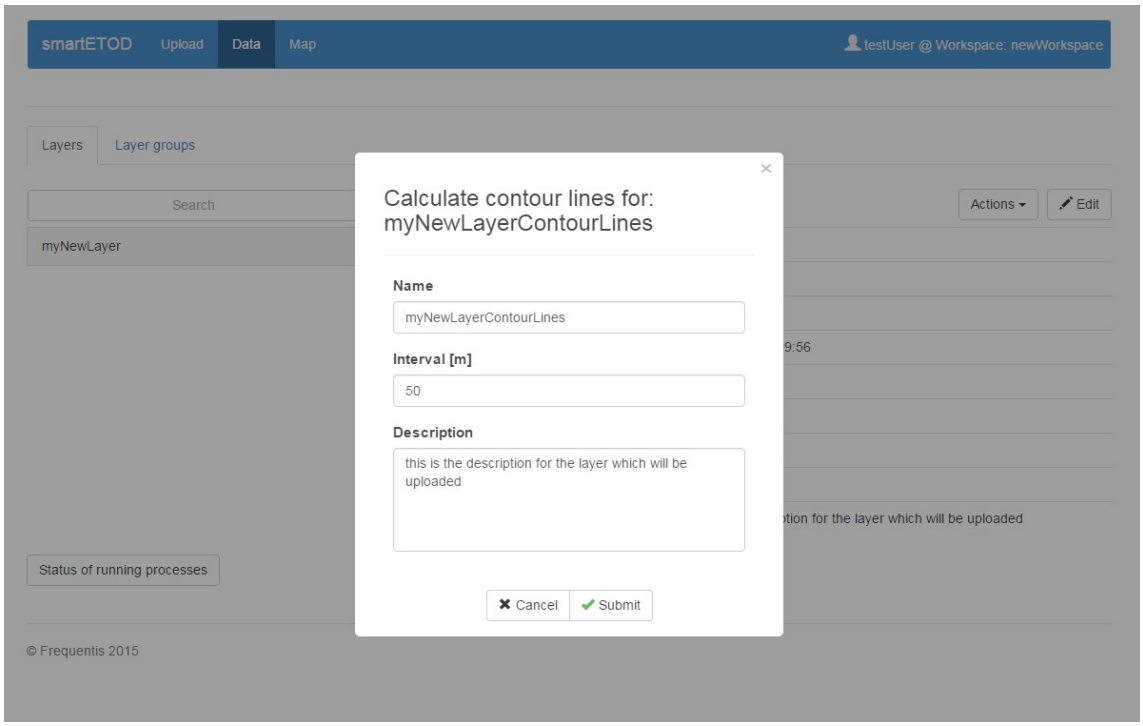

Abbildung 4.7.4.1: Nutzer Interface für die Berechnung der Höhenschichtlinien

4.7.5 Download von Daten

Der Herunterladen der Daten kann wie auch die Höhenschichtlinien Berechnung (Kapitel 4.7.4) über die „Actions“ Schaltfläche auf der Layer Metadaten Unterseite, die in Abbildung 4.7.3.1 zu sehen ist, durchgeführt werden. Mit dem WMS oder WFS könnten die Layer direkt vom GeoServer abgefragt werden und mit dem Browser, in dem der Nutzer die Applikation bedient, die Daten auf das Gerät des Nutzers herunterladen. Allerdings werden die Originaldaten, die der Nutzer in das eTOD System über die Upload Seite (Kapitel 4.7.2) hochlädt, in GeoTiff umgewandelt (Kapitel 4.6.2.3) und damit nicht mehr Originalzustand. Der Grund für die Konvertierung ist, dass der GeoServer diese Datenformate nicht unterstützt. Daher wird das eTOD System auf diese Weise angepasst, sodass die Originaldaten zusätzlich in einem eigenen Depot gespeichert und aufgehoben werden. Außerdem werden die Services im Backend erweitert um Funktionen bereitzustellen, um diese Daten zu lesen und an den Client weiterzugeben. Diese Erweiterung um den Download der Originaldaten zu ermöglichen lieferte einen Aufwand der nicht geplant gewesen ist.

4.7.6 Karte / Datendarstellung

Die Karte ist ein wichtiger Bestandteil des zu entwickelten GIS Produktes. In dieser können alle Daten, die in das eTOD System hoch geladen werden, angezeigt werden. Für alle Rasterdaten sowie für die Höhenschichtlinien in Form von Vektordaten werden eigene Designs erstellt und direkt bei der Speicherung der Daten in den GeoServer zugewiesen. Das Styling wird, wie in Kapitel 4.6.1.1 beschrieben, mit Hilfe von der Darstellungsbeschreibungssprache SLD festgelegt. Diese erlauben, nicht nur die Farben festzulegen sondern auch Beschriftungen und Symbole. Für die Rasterdaten wird eine Farbskalierung verwendet, die von grün als niedrigste Klasse, über orange für die mittleren Höhen, bis hin zu braun als Repräsentant für die höchste Klasse. Die Vektordaten sind ausschließlich Höhenschichtlinien, daher wird jede Linie mit einer Höhe versehen. Die Linien werden alle dunkelbraun koloriert. Linien, die durch 500 teilbar sind, werden etwas dicker dargestellt um dem Nutzer einen leichteren Überblick für die Höhenunterschiede zu geben. Mit SLD lassen sich auch maßstabsabhängige Style festlegen. Damit lässt sich das Intervall der Höhenschichtlinien beim heraus zoomen aus der Karte vergrößern, da sonst die Karte auf Grund von zu vielen dicht beieinander liegenden Linien nicht mehr lesbar wird. In Abbildung 4.6.1.1 des Kapitels 4.6.1.4.2 werden ein Raster und ein Vektor übereinander liegend gezeigt. Die Daten werden mittels WMS für die Rasterdaten sowie WFS für die Höhenschichtlinien , die der GeoServer bereits zur Verfügung stellt, abgefragt. Damit muss keine eigene Schnittstelle dafür entwickelt werden. Es ist auch zu beachten, dass die Daten in einer bestimmten Projektion, der EPSG:3857, erforderlich sind,

damit diese auf der Karte angezeigt werden können. Der GeoServer bietet auch dafür eine Konvertierung. Daher kann die gewünschte Projektion über WMS oder WFS Abfrage festgelegt werden.

Laut den Anforderungen ist für die etod Applikation sowohl eine 2D als auch eine 3D Visualisierung einzubauen. Wie bereits in der Planung im Kapitel 4.4 beschrieben wird Open Layers für die 2D und Cesium für die 3D Karte verwendet. Open Layers ist bereits ein reiferes OSS Produkt mit einer großen Gemeinschaft die für ein robustes Produkt sorgt. Die Implementierung stellte sich daher problemlos dar. Auch Cesium ist ohne größeren Aufwand einzubinden. Problematisch ist jedoch die Synchronisation der beiden Produkte. Wenn der Nutzer einen Layer betrachtet und währenddessen zwischen 2D und 3D wechselt, sollten die Daten in beiden Kartierungsframeworks angezeigt werden, allerdings fällt bei den Tests, bei denen verschiedene Datensätze herangezogen werden, auf, dass dabei Probleme in der Visualisierung entstehen. In Abbildung 4.7.6.1 ist ist in weißer Farbe ein Testvektordatensatz zu sehen, der das Pentagon in den Vereinigten Staaten repräsentiert. Es ist ein Fünfeck mit einem ausgeschnittenen Fünfeck in der Mitte. Wird nun auf die 3D Karte von Cesium gewechselt so wird der Datensatz zwar wie erwartet auf einem Globus in 3D dargestellt, allerdings ist nun das ausgeschnittene Fünfeck in der Mitte, wie in Abbildung 4.7.6.2 zu entnehmen, nicht mehr vorhanden. Dieser Fehler tritt jedoch nicht auf, wenn keine Synchronisation verwendet wird. Das würde bedeuten, dass der Nutzer bei jedem Wechsel zwischen 2D und 3D die Daten wieder vom GeoServer laden müsste. Aus der Sicht der Benutzerfreundlichkeit ist dies keine gute Lösung.

Da Cesium die Möglichkeit bietet, die Daten auch in 2D darzustellen, indem aus der Vogelperspektive auf die 3D Daten geblickt wird, wird diese Option getestet. Allerdings zeigten die Tests, dass Cesium sehr langsam ist. Beim Starten ist mit einer lange Initialisierungszeit zu rechnen und während dem Navigieren auf der Karte ruckelt diese. Auf mobilen Geräten ist auf Grund dieses Ruckeln deutlich stärker. Angesicht der genannten Probleme muss nochmals eine Entscheidung gefällt werden. Da die 3D Karte laut den Anforderungen optional ist und Aufwand bereits größer ist als erwartet, wird die Entscheidung gefällt, dass nur Open Layers 3 verwendet wird.

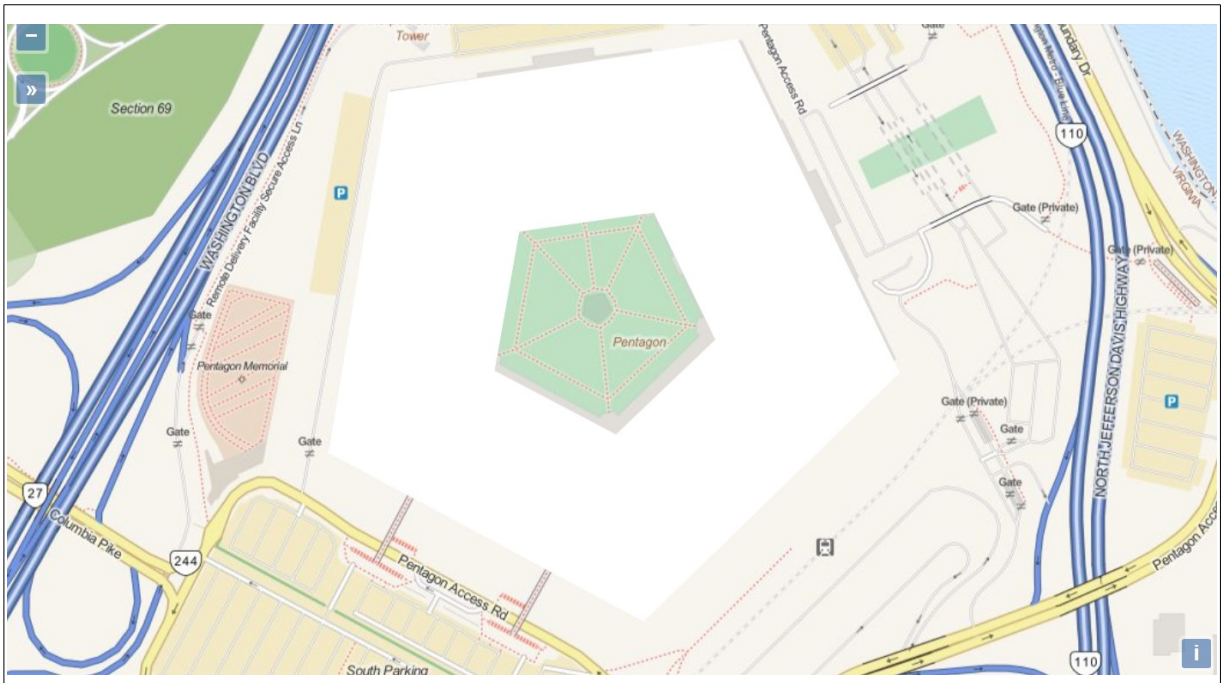

Abbildung 4.7.6.1: 2D Karte von Openlayers 3 mit einem Testvektordatensatz der das Pentagon darstellt

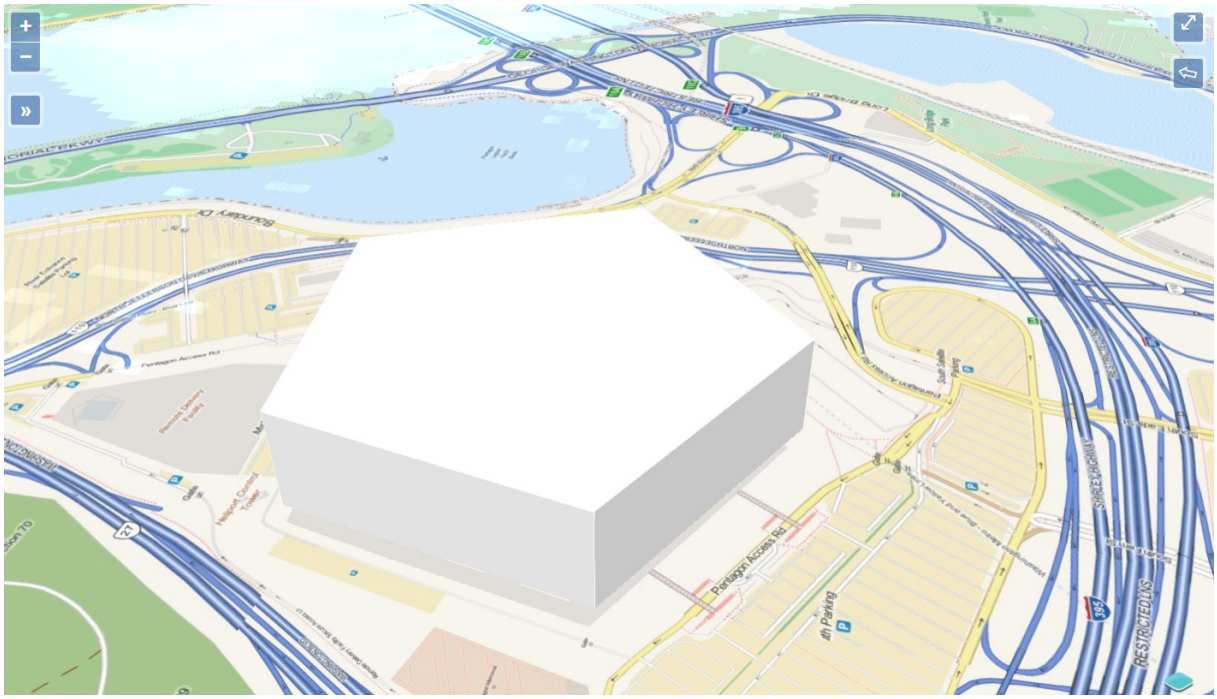

Abbildung 4.7.6.2: 3D Karte von Cesium Karte mit einem Testvektordatensatz der das Pentagon darstellt

4.8 Testen der Applikation

Jede Applikation die erstellt wird, muss auch getestet werden. Dies ist ein meist umfangreicher Prozess der nicht zu unterschätzen ist. Je mehr ein Programm an

Funktionen anbietet, desto größer ist auch der Aufwand. Es ist ebenfalls zu bedenken, dass es eine Reihe an Browser gibt und eine noch größere Anzahl an Versionen der jeweiligen Browser. Damit ergibt sich eine große Anzahl an Tests,auf die nicht verzichtet werden darf. Die Test werden nämlich geschrieben um klar darzulegen ob und wie die gegebenen Anforderungen, die in Kapitel 4.1 behandelt werden, erfüllt sind.

Für das Testen werden zwei verschiedene Arten von Tests verwendet. Zum einen manuelle Tests und zum anderen automatische Tests. Bei den manuellen Tests sitzt ein Tester vor der Applikation und prüft die von der Applikation bereitgestellten Funktionen. Für gewöhnlich ist der Tester jemand der an der Entwicklung des Programms nicht beteiligt ist um Voreingenommenheit zu vermeiden und eine neue Perspektive auf das Programm mitbringt. Die automatischen Tests dagegen können programmierte Klick und Tipp Sequenzen, die einen Nutzer simulieren, sein, wenn es sich um das Nutzerinterface handelt. Es können aber auch Kontrollen von Ausgabe- bzw. Rückgabewerte sein, wenn es sich um einzelne Codesegmente handelt.

Im Falle des entwickelten eTOD Programms werden beide Testarten durchgeführt. Bei den manuellen Tests versucht der Tester unerwünschte Fehlverhalten des Programms zu detektieren. Das Hauptaugenmerk wird jedoch auf die automatischen Tests gelegt. Sobald diese programmiert sind, ist es wesentlich zeitsparender das in Entwicklung stehende Programm zu testen. Dabei geht es um das Kontrollieren ob die bereits eingebauten Funktionen tatsächlich so funktionieren, wie sollen. Für das Testen wird das auf Javascript basierende Gerüst namens Protractor verwendet, das dafür konzipiert ist AngularJS Anwendungen zu testen. Das ebenfalls genutzte Tool Browserstack erlaubt über verschiedene Browser und Browserversionen sowie auf unterschiedlichen simulierten Geräten das Programm zu starten und die erstellten Tests an diesen anzuwenden.

Da es sich um GIS handelt, spielt die Karte eine wichtige Rolle und muss ebenso getestet werden. Allerdings gestaltet sich die Umsetzung etwas schwieriger. Im Gegensatz zu einem Textfeld wo der Text überprüft werden muss, kann eine Karte komplexe Visualisierungen enthalten. Daten können unvollständig sein oder falsch angezeigt werden. Diesen Probleme können nur bei bestimmten Zoomstufen oder Bildschirmgrößen auftreten. Das Design kann suboptimal definiert sein, sodass Textangaben verschoben, falsch skaliert sind oder sich gegenseitig überschneiden. Es ist daher eine Lösung zu wählen die möglichst automatisch durchgeführt werden kann und die genannten Faktoren berücksichtigt.

Um die Karte zu testen wird daher ein Ansatz gewählt der eine Kombination aus Bildschirmfotographierung und Pixel-Matching darstellt. Dabei wird eine Interaktion, die getestet werden soll, auf der Karte ausgeführt und anschließend ein Bildschirmfoto geschossen. Beim ersten Durchgang, wird das Bild vom Tester evaluiert. Wenn es korrekt ist, wird das Foto in einen vom Testprogramm einsehbaren Ordner verfrachtet und dient für alle zukünftigen Durchläufe als Referenz. Das bedeutet, dass bei allen weiteren Durchläufen das Referenzfoto und das neu erstellte Bildschirmfoto über eine Pixel-Matching verglichen wird. Falls eines der Pixel nicht dem Referenzpixel entspricht, versagt der Test. Diese Methode funktioniert sehr gut, allerdings ist der Aufwand die Referenzfotos zu erstellen und zu kontrollieren sehr hoch, denn die Referenzfotos müssen für jede zu testende Bildschirmgröße sowie für jeden Browser erstellt werden.

5 Resultate und Interpretation

Nach der Durchführung des im Kapitel 4 beschriebenen Experiments, werden nun die Resultate analysiert und bewertet. In Kapitel 5.1 wird der geschätzte Aufwand dem Benötigten gegenüber gestellt. Anschließend wird die Qualität des eTOD Programms, das in Rahmen des Experiments entwickelt wurde, in Kapitel 5.2 besprochen. Des Weiteren wird in Kapitel 5.3 der Aufwand des auf Basis von GIS OSS erstellten eTOD Produktes mit dem eines auf reinem CSS beruhenden Produktes verglichen. In Kapitel 5.4 werden die erhaltenen Resultate interpretiert und bewertet. Mit den erhaltenen Erkenntnissen und der gesammelten Erfahrung im Rahmen des Experiments, können auch die Aussagen aus der Literatur verifiziert werden. Dies wird in Kapitel 5.5 durchgeführt. Zuletzt werden in Kapitel 5.6 Verbesserungsvorschläge für die GIS OSS Gemeinschaft sowie für die Entwickler geliefert um die GIS OSS Produkte einfacher in die Produktion einer CSS zu implementieren.

5.1 Analyse und Vergleich des SOLL-IST Aufwandes

Zu Beginn des Experiments, bevor mit der Entwicklung des eTOD Programms begonnen wurde, wurde das Projekt in kleinere Einheiten geteilt und von den Entwicklern gemäß der Methode „Schätzung nach Fachexperten“ geschätzt und anschließend im Team besprochen um schließlich für jede Einheit die Stunden festzulegen, die für das Abschließen der Entwicklung benötigt werden (Kapitel 4.3). Das Ergebnis ist in Tabelle 4.3.1 zu sehen. Die Werte der Einheiten werden über die

einzelnen Bereiche aufsummiert sowie mit dem in Kapitel 4.3 festgelegten Puffer von 20% versehen und in Tabelle 5.1.1 in der Spalte „hours estimated“ aufgelistet.

Während des Experiments, hat jeder Entwickler die benötigten Stunden für jede Einheit in einem Projektmanagementtool festgehalten. Diese werden ebenfalls über die einzelnen Bereiche aufsummiert und in Tabelle 5.1.1 in der zweiten Spalte „hours needed“ angeführt.

Die Differenz zwischen den geschätzten und den benötigten Stunden wird in der Einheit Stunden in der dritten Spalte „Difference (h)“ und in Tagen in der letzten angegeben.

	hours estimated	hours needed	Difference (h)	Difference (days)	rise
Project Setup	134	56	78	10	-58%
UMS	41	55	-14	-2	35%
Data Upload/Import	278	240	39	5	-14%
calculation of countour lines	89	190	-101	-13	114%
define/change datasets	58	322	-264	-33	458%
meta data add/change	17	122	-106	-13	628%
expose data	36	51	-15	-2	41%
data visualisation in web map	68	233	-165	-21	241%
Meetings/Design Discussions	72	55	17	2	-24%
Sum	793h	1324h	-531h	-66d	67%

Tabelle 5.1.1: Gegenüberstellung der geschätzten und benötigten Stunden

Betrachtet man das Ergebnis am Ende von Tabelle 5.1.1, welches die Summe der Stunden oder Tage über alle Bereiche angibt, so ist deutlich zu erkennen, dass das Experiment wesentlich mehr Zeit beansprucht hat als angenommen. Der Unterschiede beträgt circa 530h bzw. 66 Tage, was einen Zuwachs des Aufwandes von 67% bedeutet. Werden die einzelnen Bereiche betrachtet, so ist zu sehen, dass genau in jenen der Aufwand größer war, wo bei der Implementierung der GIS OSS Produkten Probleme entstanden sind. Im Folgenden wird näher auf die in Tabelle 5.1.1 zu sehenden Ergebnisse und deren Ursachen eingegangen.

Weniger Zeit als erwartet benötigten zum einen das Projekt Setup, aufgrund der guten Dokumentationen und Nutzerfreundlichkeit der GIS OSS und zum anderen das Entwickeln der Tools für das Hochladen von Daten. Durch das REST Interface des GeoServer ist es einfach Daten als Layer in den GeoServer zu speichern. Der Umweg, die BIL und DTED Dateien vorher mit GDAL zu GeoTIFF zu konvertieren

(Kapitel 4.6.2.3), weil der GeoServer diese Formate nicht unterstützt, war zwar nicht geplant, allerdings konnte die Implementierung, nachdem GDAL bereits im Projekt Setup installiert und konfiguriert war, ohne Komplikationen durchgeführt werden. Die Meetings nahmen zwar ebenfalls weniger Zeit ein, allerdings ist dies auf die Tatsache zurückzuführen, dass bei aufkommenden GIS OSS Problemen sich eher die Frage stellte ob und welche Lösungen existieren als die nach den besten Lösungen. Meistens boten sich nur wenige Möglichkeiten, sodass diese schnell besprochen und über diese entscheiden werden konnte. Die benötigte Zeit für das Suchen und das Testen der Möglichkeiten wurde jedoch in den anderen Bereiche eingetragen, da sie als Teil des Entwicklungsprogresses gesehen wird.

Mehr Aufwand als geschätzt benötigte zum einen, mit einem Mehraufwand von 35%, das UMS, welches die Nutzerrechte und Nutzerrollen speichert und die Ermittlung von einer Lösung für die Abbildung dieser Rechte auf die Nutzung der Daten im GeoServer erforderte. Zum anderen das Verfügbarmachen der Daten für externe Applikationen und der vom Client verwendeten Karte. Der Bereich hat über 40% mehr an Aufwand benötigt als erwartet. Zwar stellt der GeoServer WMS und WFS zur Verfügung, die es anderen Applikationen erlaubt den OGC Standards folgend die Layer und Layergruppen, die sich im GeoServer befinden, abzufragen, allerdings liefert der GeoServer keine Möglichkeit diese Funktion nur für Layregruppen, aber nicht für Layer zu erlauben. Daher mussten auch hier Änderungen in der Form eines Proxy-Servers vorgenommen werden. Der Proxy-Server fängt jegliche Abfragen ab und entscheidet darüber ob diese zum GeoServer weitergeleitet werden oder eine Fehlermeldung an den Abfragesteller zurück gesendet wird (Kapitel 4.6.1.3).

Die in Tabelle 5.1.1 rot markierten Bereiche haben einen nicht zu vernachlässigen höheren Aufwand benötigt als ursprünglich angenommen. Solche Werte können aus Kostengründen ein Projekt zum Scheitern bringen. Es sind genau die Bereiche betroffen, in denen die meisten Probleme mit der GIS OSS auftraten. Im Folgenden werden auf die einzelnen Bereiche näher eingegangen.

Beginnend mit der Berechnung der Höhenschichtenlinien, die mehr als das doppelte an Zeit benötigt hat, bildet sie mit einem Anstieg des Aufwands von 114% noch das unterste Ende der Skala der in Abbildung 5.1.1 rot markierten extremeren Werte. Der Algorithmus für die Berechnung der Höhenschichtlinien hat in den Bereichen gleichbleibender Höhen ungewollte Artefakte erstellt. Außerdem war die Berechnung, also der Prozess im GeoServer, sehr langsam, sodass der Nutzer lange für die Ergebnisse warten musste. Daher war ein Wechsel zu GDAL, welches ebenfalls die Berechnung von Höhenschichtlinien anbietet, erforderlich.

Für die Begutachtung können die Layer und Layergruppen in der Karte angezeigt werden. Die Probleme bei dem Versuch die Kartierungsbibliotheken Open Layers 3 für die 2D und Cesium für die 3D Visualisierung durch eine Datensynchronisierung gemeinsam in der eTOD Applikation zu verwenden, schlug fehl und erforderte 241% mehr Zeit als vermutet.

Der Bereich „Hinzufügen und Adaptieren von Metadaten“ führt die Skala mit einem benötigten Aufwand, der knapp 630% größer als der geschätzte ist. In diesem Bereich kamen die meisten Probleme auf und das schlägt sich auch im Aufwand nieder. Die GeoServer Manager API, die für die Kommunikation mit dem GeoServer zuständig ist, hatte zum Zeitpunkt der Entwicklung einige unvollständige und fehlerhafte Funktionen bereitgestellt. Bei den Layern konnte der Name nicht geändert werden während bei den Layergruppen einzig und allein nur der Titel gesetzt werden konnte. Hinzu kommt, dass für das Definieren von eigenen Metadaten die API falsche XML Daten erstellt hat, die anschließend nach dem Übermitteln der Daten an den GeoServer vom diesem entfernt wurden. Bei der Wahl der GIS OSS Produkte konnten diese fundamentalen Probleme nicht festgestellt werden, da die Funktionen sowohl in der Dokumentation als auch im Quellcode bereit gestellt und gelistet waren.

Der Bereich für das Definieren und Verändern von Datensätzen, also den Layergruppen, nahm rund 460% mehr Zeit in Anspruch als vorgesehen war. Ein Grund war, dass die Sichtbarkeit von den einzelnen Layergruppen für externe Applikationen in der eTOD Web Applikation veränderbar sein sollten. Der GeoServer bot das zwar an, jedoch musste der Proxy-Server diesen Wert einsehen können um zu entscheiden ob externe Applikationen auf die jeweilige Layergruppe zugreifen dürfen oder nicht.

Zusammenfassend ist zu sagen, dass der tatsächliche aufgebrachten Aufwände deutlich über die geschätzten Werte hinaus gehen. Doch Angesichts der durch die GIS OSS hervorgerufenen schwerwiegenden Probleme, die teilweise fundamentale Änderungen an der Applikation erforderten, ist dies nicht verwunderlich. Der Zusammenhang zeigt jedoch, dass Erfahrung und Wissen hinsichtlich der verwendeten GIS OSS eine bessere Schätzung erlauben würden.

5.2 Analyse der Qualität

Für der Analyse der Qualität werden, wie in Kapitel 3.2 beschrieben, die Eigenschaften von (Andreou & Tziakouris 2007) herangezogen. Diese sind in Tabelle

2.3.2.1 dargestellt und im Kapitel 2.3.2 näher erläutert. Im Folgenden wird die Bewertung separat für die einzelnen Eigenschaften getätigt und am Ende des Kapitels zusammengefasst. Bewertet wird dabei das aus dem Experiment hervorgebrachte Programm und nicht die einzelnen OSS Produkte, allerdings wird auf deren Zusammenhänge eingegangen.

Funktionalität

Das Produkt erfüllt alle Anforderungen, die in Kapitel 4.1 gestellt wurden. Damit ist das Programm in seinen Funktionen vollständig. Deren einwandfreie Funktionalität wurde mit den manuellen und automatischen Tests bestätigt und kontrolliert. Aufgrund dessen, dass bereits die verwendete GIS OSS, wie beispielsweise der GeoServer, eine hohe Interoperabilität gewährleistet, konnte diese auf das resultierende Produkt übertragen werden. Das eTOD System bietet, dank dem GeoServer, OGC Standard konforme WMS und WFS an. Damit ist es anderen Systemen möglich auf die Daten, die das eTOD bereits stellt, zuzugreifen. Hierbei muss auch auf die Sicherheit geachtet werden, da mehrere Nutzer die Daten in der Applikation verwalten. Durch die Verwendung des Proxys für die Anfragen externer Applikationen sowie durch das Nutzermanagement, das mehrere Nutzerrollen und Nutzergruppen erlaubt, ist die Sicherheit gegeben, dass die Daten vor Unberechtigten geschützt sind. Hinsichtlich der Funktionalität ist das auf GIS OSS basierende Programm daher die bessere Wahl.

Zuverlässigkeit

Die Zuverlässigkeit umfasst die Stabilität des Programms sowie die Wiederherstellbarkeit. Es kann gesagt werden, dass das Programm eine hohe Stabilität aufweist, welche wie folgt zu erklären ist. GeoServer, GDAL sowie Open Layers haben eine große dahinterstehende Gemeinschaft, welche die Systeme stetig verbessert um stabile und einwandfrei funktionierende Produkte zu ermöglichen, da die OSS Entwickler auch ebenfalls Nutzer sind und davon profitieren. Sollte ein Nutzer ein Problem entdecken und teilt diesen der Gemeinschaft mit, wird diese versuchen das Problem möglichst schnell zu lösen. Da die verwendeten GIS OSS einen großen Teil des eTOD ausmachen und die Kernfunktionen des Applikation enthalten, ist die Wahrscheinlichkeit, dass bei der Verwendung des Programms Fehler auftreten, geringer. Bei dem Quellcode des eTOD dagegen, welcher selbst entwickelt werden musste, ist die Wahrscheinlichkeit höher, dass sich darin noch unentdeckte Fehler befinden. Zwar behandelt das erstellte Programm jegliche Fehler die auftreten können um Fehlfunktionen aber vor allem auch einen Systemausfall zu verhindern, jedoch können trotz ausgiebigen Testens noch Probleme auftreten, die nicht vorhergesehen werden konnten. Zusammenfassend ist zu sagen, dass aufgrund der OSS das System als Ganzes stabiler ist als ein selbsterstelltes CSS

Programm. Die Wiederherstellbarkeit ist beim eTOD ebenfalls gegeben. Sollte es zu einem Systemausfall kommen, sind die Metadaten und Daten separat als XML oder GeoTiff gespeichert und gehen nicht verloren. Die originalen Daten, die für den Download gespeichert werden müssen, sind ebenfalls in einem eigenen Speicherdepot. Das System selbst kann, nach Behebung eines Problems einfach wieder gestartet werden. In der Zuverlässigkeit punktet damit das auf GIS OSS basierende Programm.

Benutzbarkeit

Die Benutzbarkeit hängt stark von dem Nutzerinterface ab, das für die Bedienung der Applikation erstellt worden ist. Allerdings ist das Interface eines Programms, das unter Verzicht auf GIS OSS erstellt worden ist, das einer CSS Applikation sehr ähnlich. Beim durchgeführten Experiment ist die Benutzerfreundlichkeit gegeben, da das Prinzip verfolgt wurde, dass nur das sichtbar sein sollte, das auch in der jeweiligen Situation notwendig ist. Dennoch ist zu sagen, dass das optionale Ziel eine 3D Karte zu verwenden aufgrund von Kompatibilitäts- und Performanceproblemen (Kapitel 4.7.6) im Rahmen des Experiments nicht erreicht werden konnte. Die Möglichkeit zwischen einer 2D und einer 3D Ansicht zu wechseln würde eine bessere Benutzerfreundlichkeit bieten. Die Verwendung eines proprietären Kartierungsprogramms würde eine Verbesserung zeigen. Der Wechsel zwischen der 2D und 3D Ansicht gehört als Standardfunktion zu den proprietären GIS Lösungen dazu. Aufgrund der für die Nutzer bereitgestellten kundenspezifischen Lösungen ist auch die Performance bei dem jeweils genutzten Gerät optimiert. Doch neben der Benutzerfreundlichkeit fallen auch noch andere Subeigenschaften in den Bereich der Benutzbarkeit, wie die Erreichbarkeit der Applikation und die bereitgestellten Hilfsmittel für die Bedienung. Eine gute Erreichbarkeit ist gegeben, da das eTOD über jedes Gerät, das einen Internetbrowser verfügt, gestartet werden kann. Auch ein Wechsel des Gerätes stellt kein Problem für den Nutzer dar, da die Daten auf einem Server gespeichert sind und damit unabhängig vom verwendeten Gerät ist. Hilfsmittel für die Applikation findet der Nutzer in Form von Online-Dokumentationen, die der Nutzer direkt aus der Applikation aufrufen kann. Beides ist jedoch auch ohne GIS OSS durchführbar und hat damit denselben Einfluss auf die Benutzbarkeit. Damit liefert die Nutzung von einer auf CSS basierenden Applikation im Bereich der Benutzerfreundlich einen leichten Vorteil.

Effizienz

Da OSS Produkte von und für eine breite Masse an Nutzern konzipiert sind, ist der System Overhead, also der Quellcode der nicht effektiv vom Programm genutzt wird, größer als bei selbst erstellten CSS Programmen, die für die jeweiligen Anforderungen optimal angepasst sind. Beispielsweise werden nur ein Teil der

Funktionen des GeoServer verwendet, dennoch muss das GIS OSS als ganzes verwendet werden, da es bereits kompiliert ist. Zwar könnte das genutzte GIS OSS Produkt in seinen Funktionen adaptiert werden, allerdings müsste dazu der Quellcode verändert werden. Größerer Overhead bedeutet auch eine geringere Reaktionszeit auf Anfragen vom Nutzer, da bei jeder Anfrage mehr Quellcode durchlaufen werden muss. Auch im Rahmen des Experiments ist dieses Problem aufgetreten. Die Berechnung der Höhenschichtlinien, welche in Kapitel 4.6.1.4.2 näher erläutert wird, war mit Hilfe des vom GeoServer bereitgestellten WPS langsamer als durch die direkte Nutzung von GDAL. Auf CSS basierte Programme punkten daher im Bereich der Effizienz.

Instandhaltbarkeit

Der Quellcode der GIS OSS Komponenten wurde nicht geändert, dadurch war es einfach ein neues Update einzuspielen ohne Änderungen am ganzen Programm durchzuführen. Sobald ein Update zur Verfügung stand, konnte das Entwicklerteam entscheiden wann das laufende Programm abgebrochen und das Update eingespielt werden sollte. Der selbst geschriebene Quellcode kann jederzeit erweitert werden ohne Einfluss auf die GIS OSS Komponenten. Aufgrund der beiden genannten Punkte kann von einer guter Veränderlichkeit und Anpassbarkeit gesprochen werden. Die Testbarkeit ist wie bei einem Programm, dass auf CSS basiert, gegeben. Der GeoServer, der die meisten Berechnungen im eTOD System durchführt, bietet jedoch zusätzlich Monitoring Tools an mit diesem der Ressourcenverbrauch des GeoServer observiert werden kann. Damit ist die Instandhaltbarkeit des eTOD, aufgrund der Nutzung von GIS OSS kostengünstiger und einfacher (Anh *et al.* 2012).

Zusammenfassend kann gesagt werden, dass ein auf GIS OSS basierendes Programm Vorteile in der Funktionalität, Zuverlässigkeit und Instandhaltbarkeit liefert, während die Benutzbarkeit und Effizienz Bereiche sind in denen CSS ein besseres Ergebnis liefern würde.

5.3 Analyse und Vergleich des Arbeitsaufwandes für GIS CSS mit GIS OSS

Eine Frage die sich noch stellt, ist ob der benötigte Aufwand für das auf GIS OSS basierte Programm nun größer ist als das eines selbsterstellten CSS Programms. Um dies festzustellen muss noch der Aufwand eines solchen Programms ermittelt werden. Es bieten sich zwei Möglichkeiten dazu an. Zum einen kann eine Aufwandsanalyse für die einzelnen GIS Produkte durchgeführt werden und zum anderen können die Kosten der Nutzung von proprietären Produkten berechnet

werden. Jedoch werden proprietäre Produkte in den meisten Fällen über Lizenzen verkauft und das Entgelt meist jährlich abgegolten. Um einen klaren Vergleich darzustellen, müssten daher die Kosten des im Rahmen des Experiments entwickelten Projektes, das auf OSS basiert, über einen längeren Zeitraum, wie ein paar Jahre, beobachtet werden. Die Kosten die entstehen würden, würden durch die notwendigen Updates der Programme verursacht werden, die auch tiefer greifende Veränderungen in den verwendeten GIS OSS Produkten mit sich bringen und damit auch Änderungen an den selbsterstellten Komponenten notwendig machen würden, da durch die Updates unerwartete Fehler auftreten könnten.

Bei proprietären Produkten entstehen zwar Lizenzkosten, allerdings werden viele mögliche Fehler durch den hohen Kundenkontakt vermieden. Sollten dennoch Probleme beim Update auftreten, kann der Kundensupport benachrichtigt werden.

Aufgrund des erforderlich langen Beobachtungszeitraums, kann nur ein theoretischer Wert angegeben werden. Dazu werden die jährlichen Lizenzkosten der zu benutzenden CSS benötigt. Allerdings war es im Rahmen dieser Arbeit nicht möglich Angebote für die proprietären Produkte einzuholen. Daher wird nur auf die Aufwandsschätzung eingegangen, die sich durch die Entwicklung eines eTOD unter Verzicht der Nutzung von GIS OSS Komponenten ergibt.

Für die Aufwandsschätzung wird, wie bei der Wahl des GIS Servers in Kapitel 4.2, OpenHUB verwendet. Dieses Portal scannt den Quellcode des im GitHub Portals befindlichen GIS OSS Projektes und liefert die Summe der Quellcodezeilen. Da das OSS Projekt sehr groß ist und viele Funktionen beinhaltet, die im eTOD Projekt nicht in Verwendung sind, sind diese Module aus der Berechnung der Summe von Quellcodezeilen auszuschließen. OpenHUB bietet dafür ein Tool an. Damit werden alle Ordner und Dateien, die in GitHub zur Verfügung stehen, angezeigt. Durch das Markieren können diese aus der Berechnung ausgeschlossen werden. Die Resultate für die genutzten GIS OSS kann in Tabelle 5.3.1 betrachtet werden. Entfernt werden Quellcode-Module wie Dokumentationen, Konfigurationsdateien, Beispiele, Vorlagen sowie Dateien, die Funktionen enthalten, die nicht benötigt werden.

	Lines of Code	Effort (person-year)	ignored files
GeoServer	296.773	93	12.704
Open Layers 3	36.067	9	1.784
GDAL	55.754	14	910
Sum	388.594	116	15.398

Tabelle 5.3.1: geschätzter Aufwand für die CSS Entwicklung der GIS OSS Komponenten

Wie in Tabelle 5.3.1 zu sehen, ist allein der Aufwand für die Entwicklung einer CSS Version der aufgelisteten GIS OSS sehr hoch und übersteigt bereits die des programmierten eTODs, das mit 1324 Arbeitsstunden beziffert wurde (Kapitel 5.1), bei weitem. Es muss jedoch noch darauf hingewiesen werden, dass noch Aufwände durch das für das Zusammenfügen der Komponenten sowie für das Testen anfallen, diese sind jedoch im Vergleich zu den Zahlen in der Tabelle 5.3.1 vernachlässigbar.

In Anbetracht dieser Zahlen kann die Aussage getätigt werden, dass die Entwicklung unter Verwendung von GIS OSS günstiger ausfällt, als eine reine CSS Entwicklung. Diese Aussage wird in Kapitel 5.5 mit der vorhanden Literatur verglichen.

5.4 Interpretation der Ergebnisse der Analysen

Im Folgenden werden die Resultate der Analysen aus den vorhergehenden Kapiteln herangezogen, zusammengefasst und interpretiert.

Der Aufwand für die Entwicklung des in Kapitel 4.3 dargestellten eTOD Systems, das auf GIS OSS basiert, wurde der Aufwand nach der Auswertung in Kapitel 5.1 mit 1324 Arbeitsstunden beziffert. Dies war deutlich über dem geplanten Wert von 793 Arbeitsstunden (Kapitel 4.3). Werden die Ursachen für diese Verzögerung betrachtet, die in Kapitel 5.1 dargestellt werden, so ist festzustellen, dass ein detaillierteres Wissen über die GIS OSS Produkte eine höchstwahrscheinlich bessere Schätzungsabgabe ermöglicht hätte als jene die in Kapitel 4.3 für das eTOD durchgeführt worden ist. Es muss jedoch erwähnt werden, dass reines Wissen über GIS OSS zu wenig für eine gute Schätzung sind. Zwar existieren zahlreiche Dokumentationen und Beispiele, die die Grundlagen der GIS OSS Komponenten erläutern und eine Liste von bereitgestellten Funktionen wiedergeben, allerdings können diese fehlerhaft und unvollständig sein, wie sich das auch im Rahmen des Experiments für die GeoServer Manager API gezeigt hat (Kapitel 4.6.1.2). Das Risiko auf solche Probleme zu stoßen ist bei OSS immer vorhanden (Rudzki *et al.* 2009).

Für die Minimierung des Risikos ist daher auch die Erfahrung mit der zu verwendeten GIS OSS wichtig und zwar sowohl in der Planung des Projektes, wo der Aufwand und damit die Kosten für das Projekt definiert werden müssen, als auch in der Durchführung, wo ein reicher Erfahrungsschatz dabei helfen könnte Probleme in der Implementierung zu lösen. In dem im Rahmen dieser Arbeit durchgeführten Experiment waren, wie in Kapitel 3.1 beschrieben, 4 Entwickler beteiligt. Alle hatten kaum Erfahrung mit der verwendeten GIS OSS. Bei der durchgeführte Schätzung nach Fachexperten (Kapitel 4.3) fehlten damit Experten aus dem Bereich des GIS OSS. Für Unternehmen die einen Wechsel von GIS CSS zu GIS OSS in Erwägung ziehen bedeutet dies, dass das benötigte Wissen und die Erfahrung, wenn diese nicht bereits vorhanden ist, ins Team geholt werden sollte. Dies kann, wie von (Kavanagh 2004a) vorgeschlagen, in Form von neuen Mitarbeitern oder Weiterbildungen umgesetzt werden. Wenn es sich um kleinere Projekte handelt, könnten auch Berater eine mögliche Option sein.

Die hier mitgenommene Erkenntnis, dass bei einem Wechsel von GIS CSS auf GIS OSS, das Wissen und die Erfahrung der Projektbeteiligten entscheidend sein kann, dass ein Projekt erfolgreich abgeschlossen wird oder nicht, kann auch auf Projekte übertragen werden, die nicht aus dem Bereich der Geoinformation oder Kartographie kommen. Begründen lässt sich dies dadurch, dass die Situation des Projektes und der Entwickler beim Wechsel von CSS auf OSS die gleiche ist. Lediglich das zu erstellende Programm sowie die zu verwendeten OSS Komponenten würden variieren.

Auch die Qualität des resultierenden Programms ist beim einem Wechsel von GIS CSS zu GIS OSS zu berücksichtigen. Zu Beginn in Kapitel 1.3 wurde die Frage gestellt ob die Qualität eines Programms, welches unter der Verwendung von OSS erstellt worden ist, mit dem eines CSS Programms vergleichbar wäre. In Kapitel 5.2 wurde die Qualität des im Experiment erstellten eTOD analysiert. Das Ergebnis beantwortet die Frage mit einem klaren Ja. Werden die einzelnen Qualitätseigenschaften, die von (Andreou & Tziakouris 2007) übernommen worden sind, im Detail betrachtet, so kann anhand der Experiments geschlossen werden, dass CSS Produkte sowie auf OSS basierende Produkte von der Qualität her zwar sehr ähnlich sind, jedoch sich diese in ein paar Eigenschaft etwas unterscheiden. Wie bereits in Kapitel 5.3 beschrieben, hat das resultierende Programm, dank der Verwendung von GIS OSS, eine breitere Palette an Funktionen, deren Funktionalität aufgrund der hohen Menge an Tester in der GIS OSS Gemeinschaft eine hohe Stabilität und Zuverlässigkeit aufweisen. Die Komponenten lassen sich ohne Probleme updaten solange nicht der Quellcode geändert wird. Somit sind die Instandhaltungskosten geringer. Auf der anderen Seite erreicht die Applikation aufgrund des System Overheads, der mit hohen Anzahl an Funktionen kommt, nicht

die Effizienz einer CSS Applikation. Die Benutzerfreundlichkeit ist zwar bei einem auf CSS basierenden Produkt besser, allerdings können manchmal die GIS OSS Nutzeroberflächen durch eigene Nutzeroberflächen, wie die des GeoServers, ersetzt werden. Dies gilt jedoch nicht für Web Komponenten wie beispielsweise Open Layers 3.

Wie schon eine der Aufwandsberechnung und Entwicklung, können auch hier die gewonnenen Erkenntnisse hinsichtlich der Qualität von auf GIS OSS basierenden Programmen auf Produkte, die nicht aus dem Kartographie- und Geoinformationsbereich kommen, verwendet werden. Etwas näher auf dieses Thema wird in Kapitel 5.5 eingegangen.

Angesichts der genannten Punkte, ist zusammenfassend zu sagen, dass der Wechsel von CSS zu OSS einen Vorteil im Bezug auf den benötigten Aufwand für ein Programm liefert. Des Weiteren kann mit Hilfe von OSS ein gute Programmqualität erzielt werden, die mit einem reinen CSS Programm durchaus vergleichbar ist. Zu beachten ist jedoch, dass der Wechsel ein gewisses Knowhow und Erfahrung erfordert um ein Projekt erfolgreich zu planen und zu entwickeln. Zum Schluss ist noch zu sagen, dass ein durchgeführtes Experiment mit der in Kapitel 3 definierten Ausgangssituation auf dieselbe Schlussfolgerung schließen lassen würde, wie das in dieser Arbeit.

5.5 Vergleich mit Aussagen aus der Literatur

In Kapitel 1.2 zu Beginn der Arbeit sowie in Kapitel 2.2.4, in welchem die Vor- und Nachteile von OSS beschrieben wurden, wurden zahlreiche Aussagen geäußert, die die Nutzung des OSS betreffen. Im Folgenden werden diese mit den aus dem Experiment gewonnenen Erkenntnissen (Kapitel 5.4) verglichen und diskutiert.

Bevor mit der Entwicklung des Programms begonnen werden konnte, mussten die zu verwendeten OSS Komponenten gewählt werden. Laut (Teixeira *et al.* 2015) und (Jansen 2014) sollte es möglich sein bei Betrachtung von verschiedenen OSS Produkten, die ähnliche Funktionalitäten aufweisen, mit der Zuhilfenahme von Foren und der Erhebung von Suchstatistiken zu entscheiden, welche Produkte eine große aktive Gemeinschaft besitzen, die am jeweiligen Produkt weiterentwickeln wird. Für diese Evaluierung wurde in Kapitel 4.2, wie bei (Jansen 2014), GitHub und Google Trends herangezogen und lieferte tatsächlich die für die Entscheidung erforderlichen Informationen. Damit konnte das Risiko, dass in naher Zukunft keine neuen fehlerbehebenden Updates mehr zur für Verfügung stehen, minimiert werden.

Wie wichtig diese Updates sind, zeigte sich während des durchgeführten Experiments. Beim Testen der Höhenschichtlinienberechnung in Kapitel 4.6.1.4.2 wurden eigenartige Anomalien in den Höhenschichtlinien in den Bereichen von flachen Gelände entdeckt (Abbildung 4.6.1.2). Eine Websuche ergab, dass das Problem bereits bekannt war und die OSS Gemeinschaft bereits daran arbeitete. Bevor das Experiment abgeschlossen wurde, kam bereits ein Update, welches das Problem gelöst hat. Laut (Renner *et al.* 2005) ist die Qualität von OSS besser als die der kommerziellen Programme, da mehr Entwickler am Quellcode arbeiten und damit Fehler schneller gefunden sowie behoben werden können. Im Rahmen des durchgeführten Experiment kann daher die Aussage von (Renner *et al.* 2005) bestätigt werden. Da der Quellcode nicht geändert wurde, konnte die jeweilige Komponente problemlos upgedatet werden. Damit ist auch neben der schnellen Fehlerbehebung, welche ein qualitatives Endprodukt begünstigt, auch die von (Anh *et al.* 2012) postulierte einfache und kostengünstige Instandhaltung bestätigt. Innerhalb der Arbeit wurde jedoch auch ein Gegenbeispiel für gute Qualität bei OSS entdeckt. Die GeoServer Manager API zeigte Mängel in fundamentalen Funktionen auf. Die API wurde erstellt um die Kommunikation mit dem GeoServer zu erleichtern, allerdings konnten einige der Funktionen, aufgrund von Fehlern im Quellcode, nicht eingesetzt werden. Die Dokumentation war nicht auf dem neusten Stand, was (Bahamdain 2015) als Nachteil von OSS im Allgemeinen sieht. Meldungen bezüglich der Fehler konnten schließlich in einem Forum gefunden werden. Das Problem ist seit 2014 bekannt (GeoServer Manager API n.d.) allerdings wurde nicht daran weitergearbeitet. Damit könnte meiner Meinung nach die Aussage von (Renner *et al.* 2005) in der Hinsicht präzisiert werden, dass die Qualität von OSS gegenüber kommerziellen Programmen nur besser ist, wenn die Gemeinschaft groß und aktiv genug ist, diese Qualität zu erhalten oder zu fördern. Damit bleibt, wie von (Rudzki *et al.* 2009) bestätigt, bei der Wahl der OSS immer ein gewisses Risiko, welches durch eine gute Evaluierung der OSS Komponenten minimiert werden kann.

Nach (Anh *et al.* 2012) sollte das Lesen der Dokumentationen und Beispiele für kleinere OSS Produkte reichen, doch bei für die Entwicklung wichtigeren und größeren Produkten sollte auch der Quellcode betrachtet werden. Dies wurde im Rahmen des Experiments auch so gehandhabt, allerdings muss hier gesagt werden, dass dies, wie es sich auch beim Experiment gezeigt hat, nicht ausreichend sein kann. Dazu kann wieder auf das Problem mit der GeoServer Manager API verwiesen werden. Laut der zur Verfügung gestandenen Dokumentation, waren die Funktionen für den Gebrauch offen. Bei der Untersuchung des Codes sowie etwaiger Kommentare konnte ebenfalls nichts festgestellt werden, sondern erst bei direkten Anwendungen dieser Funktionen während der Entwicklung. Um das Risiko während der OSS Wahl zu minimieren, müsste die Qualität des Quellcodes untersucht

werden, indem die Funktionen auf ihre Funktionalität geprüft werden. Dies ist allerdings sehr zeitaufwendig. Dennoch meint (Rudzki *et al.* 2009), dass der Fokus auf die Qualität gesetzt werden sollte. Möglicherweise hätte das eTOD Projekt etwas Entwicklungszeit einsparen können, wenn für die Evaluierung mehr Zeit zur Verfügung gestanden wäre. Laut (Anh *et al.* 2012) wird ohnehin meist der Aufwand für die Evaluierungsphase unterschätzt. Für einen Wechsel von CSS auf OSS sowie für weitere Experimente sollte daher mehr Aufwand für diese Phase eingeplant werden.

Auch der Aufwand für die Entwicklung des eTOD auf Basis von OSS wurde unterschätzt. Der Grund war der Mangel an fachspezifischen Wissen und Erfahrung im Bereich der verwendeten OSS. Laut (Karels 2003) unterschätzen viele Firmen, dass die Integration der Komponenten in ein Programm sehr viel Zeit und damit Kosten erfordern kann. Dies bestätigt nun auch das in dieser Arbeit durchgeführte Experiment. Das eTOD benötigte deutlich mehr Zeit als ursprünglich bei der Planung angenommen. Neben den Entwicklungskosten kommen noch Ausbildung und Akquirierungskosten um das notwendige OSS Wissen in das beteiligte Team zu erhalten hinzu (Kavanagh 2004a). Allerdings ist zu sagen, dass der Aufwand für die eTOD Entwicklung auf Basis von OSS, trotz der fehlenden Erfahrung, geringer Ausgefallen ist als eine eTOD Entwicklung auf Basis von CSS. Dies bestätigt die Aussage von (Kavanagh 2004a), dass die durch die Verwendung von OSS entstandenen Kosten in fast allen Fällen deutlich geringer bis gleich groß sind wie bei einer reinen CSS Entwicklung.

Abschließend kann gesagt werden, dass die Erkenntnisse aus dem durchgeführten Experiment den meisten Aussagen aus der Literatur die Bestätigung liefert und somit gut in das theoretische Bild für die Nutzung von OSS passt.

5.6 Einsatz- und Gestaltungsempfehlungen

Aufgrund der im Rahmen dieser Arbeit gewonnen Erfahrung möchte ich ein paar Gestaltungsempfehlungen für die GIS OSS Gemeinschaft darlegen.

Ein größeres Problem stellte die Synchronisation der beiden GIS OSS Komponenten Open Layers 3, welche für die Visualisierung von 2D Daten im Web konzipiert ist und Cesium, welche für die Darstellung von 3D Daten ebenfalls für Web Applikationen entwickelt worden ist. Zwar bietet Cesium die Option die Daten auch in einer vogelperspektivischen Form zu betrachten, allerdings ist die Performance bei weitem nicht die Gleiche wie bei einer optimierten 2D Karte. Daher wäre eine Möglichkeit zwischen den beiden GIS OSS zu wechseln ideal um das beste aus beiden

Komponenten zu holen, vor allem im Bereich der Benutzerfreundlichkeit. Eine optimale Lösung wäre eine GIS OSS Komponente zu entwickeln, die die Synchronisation managen kann. Daher kann der Nutzer entscheiden ob dieser die Komponente implementieren möchte oder nicht. Sollte er sie nicht benötigen, würde auch kein unnötiger System Overhead entstehen.

Die nächste Gestaltungsempfehlung betrifft die GeoServer Manager API, welche für die vereinfachte Kommunikation mit dem GeoServer verwendet wird. Zur Zeit des durchgeführten Experiments waren jedoch einige Funktionen aufgrund von Fehler nicht anwendbar. Sobald Fehlfunktionen festgestellt werden, sollten diese in den Dokumentationen festgehalten werden oder zumindest als Kommentar bei der jeweiligen Funktion hinzugefügt werden. Zwar werden solche Probleme von Fehlerreportern in den Foren der jeweiligen GIS OSS Gemeinschaft gemeldet, allerdings geht diese in der Masse an Meldungen unter. Ein neuer Nutzer, der das GIS OSS für eine mögliche Implementierung untersucht, wird die Foren aufgrund der dafür benötigten Zeit nicht auf etwaige Fehlermeldungen durchforsten, sondern die Dokumentationen, Tutorien sowie den Quellcode näher betrachten (Anh *et al.* 2012).

Als letztes möchte ich noch eine Einsatzempfehlung äußern. Es sollten möglichst OSS Produkte gewählt werden die für eine spezielle Aufgabe konzipiert sind, da der Quellcodeumfang und somit auch der System Overhead klein bleibt. OSS Produkte, die viele Funktionen bereitstellen, kommen mit der Gefahr eines großen System Overhead. Damit wird die zu programmierende Applikation deutlich größer ohne Beachtung dessen ob alle Funktionen genutzt werden oder nicht. Ein größerer System Overhead bedeutet aber auch, dass eine Anfrage oder Berechnung mehr Code durchlaufen muss, was eine längere Prozessierungszeit nachzieht. Der Nutzer muss daher länger auf Resultate warten. Laufen mehrere Prozessierungen zeitgleich, bedeutet dies auch, dass über die Zeit mehr Ressourcen des Servers von dem laufenden Programm in Anspruch genommen werden müssen.

6 Zusammenfassung

Das Ziel der vorliegenden Arbeit war die Evaluierung der Hypothese, dass ein proprietäres GIS mittels GIS OSS erstellt werden kann und aufgrund deren Nutzung eine zu einem reinen CSS Produkt vergleichbaren Qualität mit Zeit- und Kostenersparnis erzielbar ist. Die Qualität wurde anhand der Kriterien von (Andreou & Tziakouris 2007), bestehend aus Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Instandhaltbarkeit analysiert und mit der für das Verständnis des GIS OSS Themas notwendige Wissen und Kapitel 2 beschrieben. Zur Evaluierung dieser

Hypothese wurde ein Experiment in einem Unternehmen durchgeführt, an dem 4 Entwickler, die keine GIS OSS Erfahrung besaßen, ein eTOD entwickelt hatten. Das Experiment, wie in Kapitel 3 beschrieben, umfasste, wie es bei der Produktion in einem Software-Unternehmen üblich ist, die Planung des Aufwandes und der Programmarchitektur, die Wahl der einzubindenden Programmkomponenten sowie die Entwicklung.

Die Ergebnisse dieser Arbeit lieferten neue Erkenntnisse im Bereich der GIS OSS Verwendung in der Produktion. Sie zeigen, dass die Entwicklung eines Programms, das mit Hilfe von GIS OSS erstellt wird, deutlich weniger Aufwand erfordert als eine reine CSS Applikation (Kapitel 5.3) (Kavanagh 2004a). Der Unterschied in dem durchgeführten Experiment betragen mehr als 115 Jahre an Arbeitszeit. Das Ergebnis des Experiments war in qualitativer Hinsicht mit der CSS Applikationen vergleichbar, jedoch punktete das auf GIS OSS basierende Programm in anderen Bereichen, wie in der Funktionalität, Zuverlässigkeit und Instandhaltbarkeit (Anh *et al.* 2012), ein reines CSS Programm dagegen zeigt seine Stärken in der Benutzbarkeit und Effizienz (Kapitel 5.2). Damit wäre die zu Beginn aufgestellte Hypothese (Kapitel 1.3) evaluiert.

Aus dieser Arbeit kann jedoch auch mitgenommen werden, dass bei einem Wechsel von GIS CSS zu GIS OSS Wissen und Erfahrung in Bereich der GIS OSS entscheidend für das erfolgreiche Abschließen eines Projektes ist. Das durchgeführte Experiment wurde deutlich unterschätzt, denn der benötigte Aufwand war 67% größer als geplant. Grund waren zwei verschiedene Faktoren. Zum einen fehlte bei der Schätzung, die nach der aus der Literatur entnommenen Theorie durchgeführt wurde (Jorgensen 2014) (Rammage *et al.* 2010), die nötige Expertise im Bereich der GIS OSS. Zum anderen wurden unvorhergesehene Probleme, wie unvollständige Dokumentationen, nicht funktionsfähige OSS Komponenten und Kompatibilitätsprobleme zwischen den gewählten Komponenten, angetroffen (Karels 2003). Wissen und Erfahrung in den verwendeten Komponenten würden diese beiden Einflussfaktoren entschärfen. Daher sollte im Entwicklungsteam eines Projektes, das einem Wechsel von GIS CSS zu GIS OSS unterzogen werden soll, die fehlende Expertise durch Weiterbildungen oder neuen Mitarbeitern hinzu geholt werden (Kavanagh 2004a).

Des Weiteren wurden die in der Arbeit gefunden Erkenntnisse mit den Aussagen aus der Literatur verglichen. Dass OSS eine gute Qualität besitzen kann, wie von (Renner *et al.* 2005) postuliert wird, konnte mit Hilfe des Experiments bestätigt werden. Schnelle Fehlerbehebungen und kostengünstige Instandhaltung, dank der Nutzung von OSS, konnten ebenfalls gezeigt werden (Anh *et al.* 2012). Allerdings

legten die Ergebnisse des Experiments auch dar, dass es durchaus auch OSS schlechter Qualität gibt, die als solche nicht gleich durch die Dokumentationen und Beschreibungen zu erkennen sind, sondern erst bei der Nutzung während der Entwicklung. Das zeigt, dass die OSS Evaluierungsphase nicht zu vernachlässigen ist (Anh *et al.* 2012) (Rudzki *et al.* 2009).

Der Vergleich der Ergebnisse legt offen, dass die gefunden Erkenntnisse nicht nur für den Bereich der Kartographie und Geoinformation gilt, sondern für alle Bereiche in denen die Verwendung von OSS in der Produktion möglich ist. Das bedeutet, dass die Ergebnisse eines Experiments, das unter der in Kapitel 3 definierten Ausgangssituation und Methodik folgend durchgeführt wurde, auf dieselbe Schlussfolgerung schließen lassen würden

7 Ausblick

Mit der wachsenden OSS Bewegung (Kuan 2001) kann davon ausgegangen werden, dass auch die Interesse für die Verwendung von OSS steigt. Besonders in der Software Industrie, die aufgrund des schnellen Fortschritts einem ständigen Wandel unterworfen ist und daher nach Möglichkeiten sucht Kosten zu reduzieren und qualitativ bessere Produkte auf dem Markt zu bringen (Bitzer 2004) (Höst *et al.* 2011), ist das in der Arbeit behandelte Thema von Relevanz. Möglichkeiten für die Optimierung der Prozesse, wie die Auswahl der richtigen OSS Komponenten und die Planung des Aufwandes für die Auswahlphase, wären Themen die noch zu untersuchen gilt. Auch das Auffinden neuer Möglichkeiten der OSS Implementierung um die Kompatibilität zwischen den einzelnen OSS Komponenten zu verbessern, könnte neue Erfahrungen bringen. Die Ergebnisse könnten nicht nur Möglichkeiten für die Kostensenkung ergeben sondern auch das Verständnis für diese Thematik erweitern. Diese Arbeit trug einen Teil des Fundaments für dieses Verständnis bei und wie angedeutet, bietet die Verwendung von OSS noch ein breites Spektrum an Bereichen, die durchaus für weitere Experimente und Arbeiten geeignet wären und für die Software Industrie von Nutzen sein könnten.

8 Anhang

8.1 Verwendete Programme

IntelliJ IDEA (https://www.jetbrains.com/idea/)

TortoiseSVN (http://tortoisesvn.net/index.de.html)

Apache Tomcat (http://tomcat.apache.org/)

GeoServer (http://geoserver.org/)

GeoWebCache (http://geowebcache.org/)

PostgreSQL (http://www.postgresql.org/)

8.2 Verwendete Bibliotheken

Frontend:

Angular (https://angularjs.org/)

Bootstrap (http://getbootstrap.com/)

Bower (http://bower.io/)

BrowserStack (https://www.browserstack.com/)

generator-gulp-angular (https://github.com/Swiip/generator-gulp-angular) XXXXXXX

Gulp (http://gulpjs.com/)

Jasmin (http://jasmine.github.io/)

JSHint (http://jshint.com/)

Maven (https://maven.apache.org/)

npm (https://www.npmjs.com/)

Node (https://nodejs.org/en/)

Protractor (https://angular.github.io/protractor/)

Restangular (https://github.com/mgonto/restangular)

Backend:

Hibernate (http://hibernate.org/)

Jersey (https://jersey.java.net/)

GeoServer-Manager (https://github.com/geosolutions-it/geoserver-manager/)

GDAL (http://www.gdal.org/)

GeoTools (http://geotools.org/)

8.3 Analysewerkzeuge

Open Hub https://www.openhub.net/

Google Trends https://www.google.com/trends/

Stack Overflow http://stackoverflow.com/

8.4 Weitere Bilder der Applikation

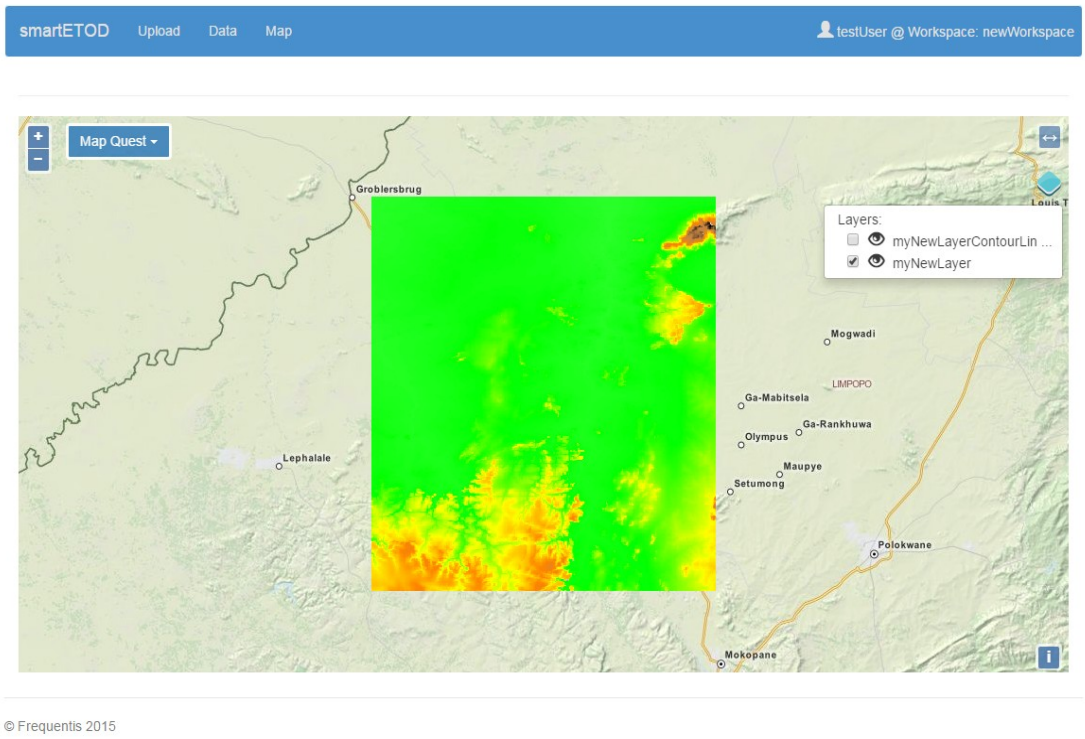

Abbildung 8.4.1: gestyltes DEM auf der Open Layers 3 Karte in der etod Applikation

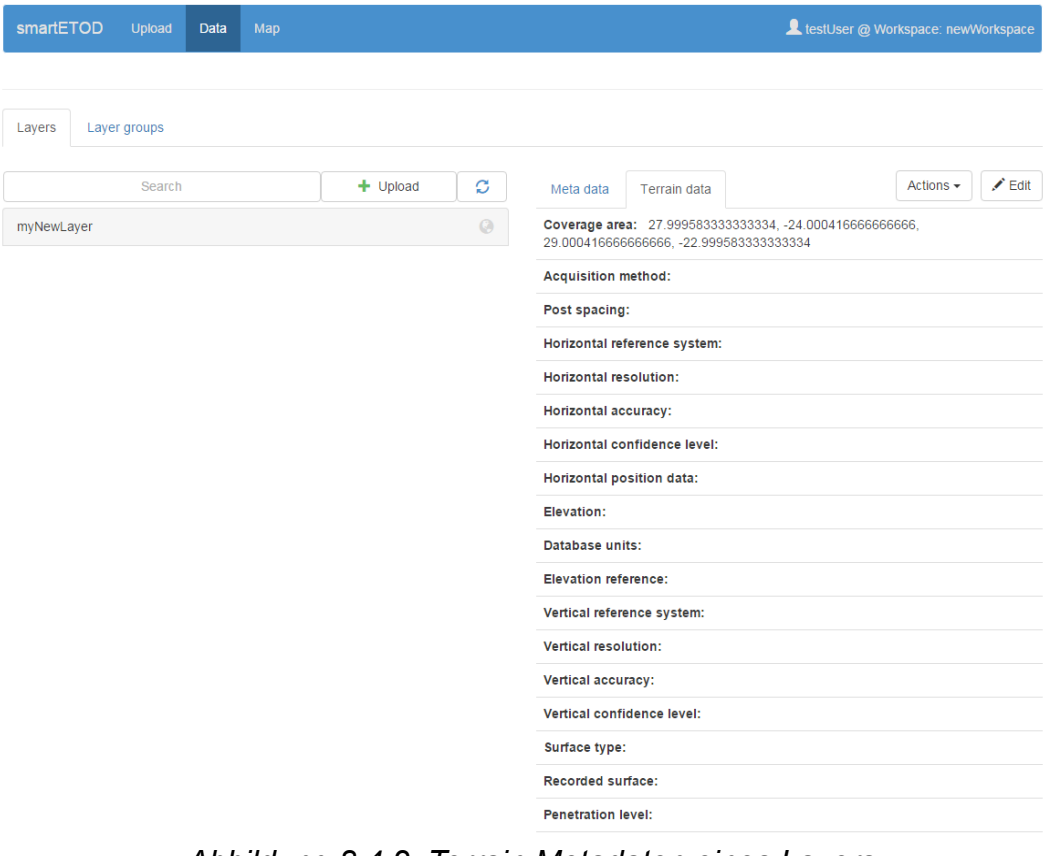

Abbildung 8.4.2: Terrain Metadaten eines Layers

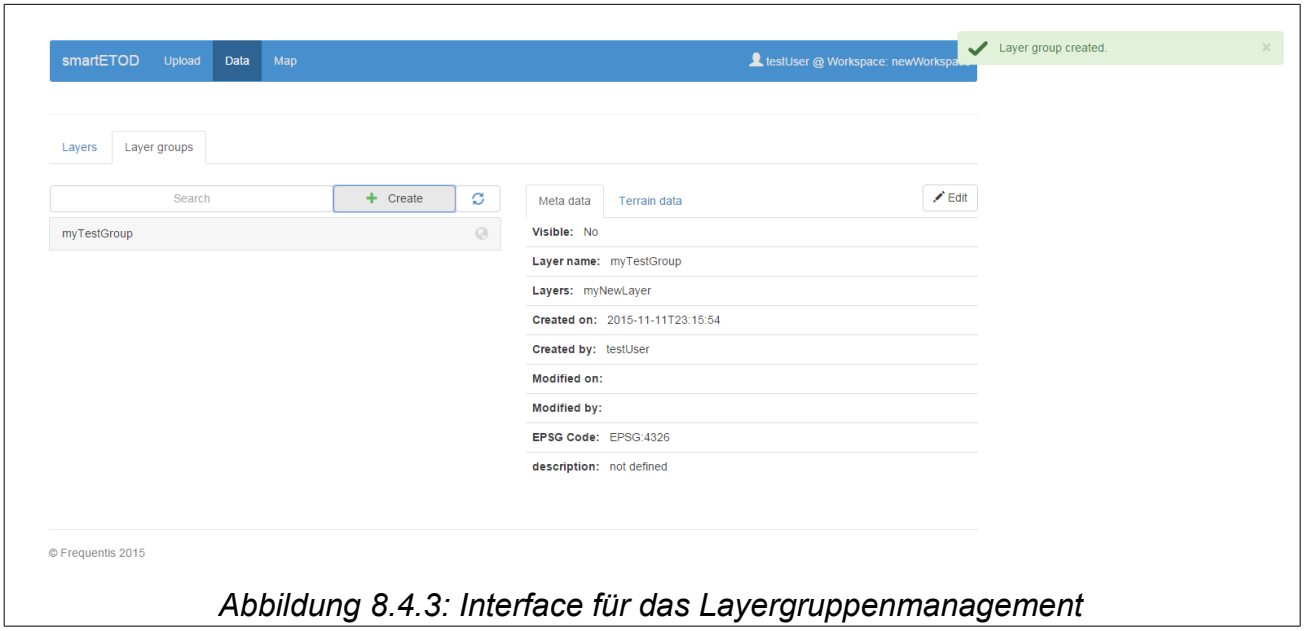

Abbildung 8.4.3: Interface für das Layergruppenmanagement

9 Literaturverzeichnis

Andreou, A.S. & Tziakouris, M. 2007. A quality framework for developing and evaluating original software components. *Information and Software Technology*, **49**, 122–141, doi: 10.1016/j.infsof.2006.03.007.

Anh, N.D., Cruzes, D.S., Conradi, R., Höst, M., Franch, X. & Ayala, C. 2012. Collaborative Resolution of Requirements Mismatches When Adopting Open Source Components. *In*: Regnell, B. & Damian, D. (eds) *Requirements Engineering: Foundation for Software Quality*. Springer Berlin Heidelberg, Lecture Notes in Computer Science, **7195**, 77–93.

Bahamdain SS (2015) Open Source Software (OSS) Quality Assurance: A Survey Paper. Procedia Computer Science 56:459–464.

Basten, D. & Sunyaev, A. 2011. Guidelines for Software Development Effort Estimation. *Computer*, **44**, 88–90, doi: 10.1109/MC.2011.315.

Bessen, J.E. 2005. *Open Source Software: Free Provision Of Complex Public Goods*. Rochester, NY, Social Science Research Network, SSRN Scholarly Paper **ID 588763**.

Bitzer, J. 2004. Commercial versus open source software: the role of product heterogeneity in competition. *Economic Systems*, **28**, 369–381, doi: 10.1016/j.ecosys.2005.01.001.

Boehm, B.W., Brown, J.R. & Lipow, M. 1976. Quantitative evaluation of software quality. *In*: *In ICSE '76: Proceedings of the 2nd International Conference on Software Engineering*. 592–605.

Delipetrev, B., Jonoski, A. & Solomatine, D.P. 2014a. Development of a web application for water resources based on open source software. *Computers & Geosciences*, **62**, 35–42, doi: 10.1016/j.cageo.2013.09.012.

Delipetrev, B., Jonoski, A. & Solomatine, D.P. 2014b. Development of a web application for water resources based on open source software. *Computers & Geosciences*, **62**, 35–42, doi: 10.1016/j.cageo.2013.09.012.

Fairley, D. 2002. Making accurate estimates [software development estimation]. *IEEE Software*, **19**, 61–63, doi: 10.1109/MS.2002.1049392.

GeoServer Manager API. n.d. *GitHub*. World Wide Web Address: https://github.com/geosolutions-it/geoserver-manager.

Höst, M., Orucevic-Alagic, A. & Runeson, P. 2011. Usage of Open Source in Commercial Software Product Development - Findings from a Focus Group Meeting. *In*: *Product-Focused Software Process Improvement/Lecture Notes In Computer Science*. 143–155., doi: 10.1007/978-3-642-21843-9_13.

ICAO. n.d. World Wide Web Address: http://www.icao.int/Pages/default.aspx.

ICAO ETOD. n.d. World Wide Web Address: http://gis.icao.int/icaoetod/.

International Organization for Standardization. 2009. *ISO 191152*. Schwitzerland.

Jansen S (2014) Measuring the health of open source software ecosystems: Beyond the scope of project health. Information and Software Technology 56:1508–1519.

Jorgensen, M. 2014. What We Do and Don't Know about Software Development Effort Estimation. *IEEE Software*, **31**, 37–40, doi: 10.1109/MS.2014.49.

Karels, M.J. 2003. Commercializing Open Source Software. *Queue*, **1**, 40:46–40:55, doi: 10.1145/945074.945125.

Kavanagh, P. 2004a. 3 - Open Source: The Good, the Bad, and the Ugly. *In*: Kavanagh, P. (ed.) *Open Source Software*. Burlington, Digital Press, 41–65.

Kavanagh, P. 2004b. 13 - Licensing. *In*: Kavanagh, P. (ed.) *Open Source Software*. Burlington, Digital Press, 297–305.

Khatibi Bardsiri, V., Jawawi, D.N.A., Hashim, S.Z.M. & Khatibi, E. 2012. Increasing the accuracy of software development effort estimation using projects clustering. *IET Software*, **6**, 461–473, doi: 10.1049/iet-sen.2011.0210.

Knörchen, A., Ketzler, G. & Schneider, C. 2015. Implementation of a near-real time cross-border web-mapping platform on airborne particulate matter (PM) concentration with open-source software. *Computers & Geosciences*, **74**, 13–26, doi: 10.1016/j.cageo.2014.10.003.

Koçak, S.A., Alptekin, G.I. & Bener, A.B. n.d. *Evaluation of Software Product Quality Attributes and Environmental Attributes Using ANP Decision Framework*.

Kotwani, G. & Kalyani, P. 2012. Open Source Software (OSS): Realistic Implementation of OSS in School Education. *Trends in Information Management*, **7**.

Kovacs, G.L., Drozdik, S., Zuliani, P. & Succi, G. 2004. Open source software and open data standards in public administration. *In*: *Computational Cybernetics, 2004. ICCC 2004. Second IEEE International Conference on*. 421–428., doi: 10.1109/ICCCYB.2004.1437766.

Kuan, J.W. 2001. *Open Source Software as Consumer Integration Into Production*. Rochester, NY, Social Science Research Network, SSRN Scholarly Paper **ID 259648**.

Li, Z., Alaeddine, N. & Tian, J. 2010. Multi-faceted quality and defect measurement for web software and source contents. *Journal of Systems and Software*, **83**, 18–28, doi: 10.1016/j.jss.2009.04.055.

Link, C. 2010. Patterns for the Commercial Use of Open Source: Legal and Licensing Aspects. *In*: *Proceedings of the 15th European Conference on Pattern Languages of Programs*. New York, NY, USA, ACM, EuroPLoP '10, 7:1–7:10., doi: 10.1145/2328909.2328918.

Maurya, S.P., Ohri, A. & Mishra, S. 2015. Open Source GIS: A Review. *In*: *ResearchGate* – National Conference on Open Source GIS: Opportunities and Challenges.

Noyes, K. n.d. 10 Reasons Open Source Is Good for Business. *PCWorld*. World Wide Web Address: http://www.pcworld.com/article/209891/10_reasons_open_source_is_good_for_business.html.

OGC. n.d. World Wide Web Address: http://www.opengeospatial.org/docs/is.

Open Source Initiative. 2007. World Wide Web Address: https://opensource.org/.

Optimus Information Inc. 2015. Open-Source vs. Proprietary Software Pros and Cons.

Raghunathan, S., Prasad, A., Mishra, B.K. & Chang, H. 2005. Open source versus closed source: software quality in monopoly and competitive markets. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, **35**, 903–918, doi: 10.1109/TSMCA.2005.853493.

Rammage, R., Lei, H., Claus, M. & Baer, D. 2010. Expert software development estimation with uncertainty correction. *In*: *2010 2nd International Conference on Software Engineering and Data Mining (SEDM)* – 2010 2nd International Conference on Software Engineering and Data Mining (SEDM). 624–630.

Renner, T., Vetter, M., Rex, S. & Kett, H. 2005. *Open Source Software: Einsatzpotenziale Und Wirtschaftlichkeit*. Fraunhofer-Institut für Arbeitswirtschaft und Organisation IAO (ed.). Fraunhofer IRB Verlag.

Rubens, P. 2014. 7 Reasons Not to Use Open Source Software. *CIO*. World Wide Web Address: http://www.cio.com/article/2378859/open-source-tools/7-reasons-not-to-use-open-source-software.html.

Rudzki, J., Kiviluoma, K., Poikonen, T. & Hammouda, I. 2009. Evaluating Quality of Open Source Components for Reuse-Intensive Commercial Solutions. *In*: *35th Euromicro Conference on Software Engineering and Advanced Applications, 2009. SEAA '09* – 35th Euromicro Conference on Software Engineering and Advanced Applications, 2009. SEAA '09. 11–19., doi: 10.1109/SEAA.2009.30.

Sakellariou, P. 2016. FREE AND OPEN SOURCE SOFTWARE IN COMPUTER EDUCATION: EXPLORING THE CURRENT SITUATION IN GREEK SECONDARY SCHOOLS. *In*: *ResearchGate* – Technologies Contect Pilsen 2016.

Steiniger, S. & Hunter, A.J.S. 2012. Free and Open Source GIS Software for Building a Spatial Data Infrastructure. *In*: Bocher, E. & Neteler, M. (eds) *Geospatial Free and Open Source Software in the 21st Century*. Berlin, Heidelberg, Springer Berlin Heidelberg, 247–261.

Teixeira L, Xambre AR, Alvelos H, et al (2015) Selecting an Open-source Framework: A Practical Case Based on Software Development for Sensory Analysis. Procedia Computer Science 64:1057–1064.

Tomazin, M. & Gradisar, M. 2009. Cost-benefit analysis of free/open source software in education. *In*: *Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces, 2009. ITI '09* – Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces, 2009. ITI '09. 473–478., doi: 10.1109/ITI.2009.5196129.

Wisam, D. & Mohammed, E. n.d. *Free and Open Source GIS: An Overview on the Recent Evolution of Projects, Standards and Communities*.

Xia, D., Xie, X. & Xu, Y. 2009. Web GIS server solutions using open-source software. *In*: *2009 IEEE International Workshop on Open-Source Software for Scientific Computation (OSSC)* – 2009 IEEE International Workshop on Open-source Software for Scientific Computation (OSSC). 135–138., doi: 10.1109/OSSC.2009.5416738.

10 Abbildungsverzeichnis

11 Tabellenverzeichnis

12 Abkürzungsverzeichnis

API	Application Programming Interface
BIL	Band Interleaved by Line
BSD	Berkeley Software Distribution
bzw.	beziehungsweise
CS	Closed Source
CSS	Closed Source Software
DGM	Digitales Höhenmodell
DTED	Digital Terrain Elevation Data
ETOD	Electronic Terrain and Obstacle Data
FSF	Free Software Foundation
GPL	General Public License
LGPL	Lesser General Public License
MIT	Massachusetts Institute of Technology
OGC	Open Geospatial Consortium
OS	Open Source
OSS	Open Source Software
GDAL	Geospatial Data Abstraction Library
GIS	Geoinformationssystem
HTTP	Hypertext Transfer Protocol
ICAO	International Civil Aviation Organization
REST	Representational State Transfer
SLD	Styled Layer Descriptor
UMS	User Management System
URI	Uniform Resource Identifier
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Mapping Service
WPS	Web Processing Service
XML	Extensible Markup Language