



Fachhochschule Südwestfalen  
Hochschule für Technik und Wirtschaft  
*University of Applied Sciences*

# Diplomarbeit

## Pflichtenheftgenerator in Perl/TK

verfasst von  
Alexandros Kechagias

Betreuer: Prof. Dr. Fritz Mehner  
Fachbereich Informatik und Naturwissen-  
schaften

Korreferent: Prof. Dr. Walter Roth  
Fachbereich Informatik und Naturwissen-  
schaften

Lüdenscheid, 10. August 2011

**Eidesstattliche Erklärung**

Ich versichere, dass ich die vorliegende Arbeit selbständig angefertigt und mich fremder Hilfe nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichtem oder unveröffentlichtem Schrifttum entnommen sind, habe ich als solche gekennzeichnet.

---

Alexandros Kechagias  
Lüdenscheid, 10. August 2011

# Inhaltsverzeichnis

<b>1. Einleitung.....</b>	<b>5</b>
1.1. Allgemeines.....	5
1.2. Motivation.....	6
1.3. Übersicht.....	7
<b>2. Technologien und Programme.....</b>	<b>8</b>
2.1 Programmiersprache.....	8
2.2 Entwicklungsumgebung .....	8
2.2.1 Vim .....	8
2.2.2 perl-support.vim .....	9
<b>3. Qualitätssicherung.....</b>	<b>10</b>
3.1 Perl::Critic .....	10
3.2 Perl::Tidy .....	12
<b>4. Datenhaltung .....</b>	<b>13</b>
4.1 XML .....	13
4.2 Perl und XML .....	14
4.3 Implementierung .....	15
<b>5. Exportformate.....</b>	<b>18</b>
5.1 Das ODF-Format.....	18
5.1.1 Die ODT-Datei (Package).....	18
5.1.2 Perl und ODF .....	22
5.1.3 Implementierung .....	24
5.1.4 Generiertes Beispielpflichtenheft .....	27
5.2 Textformat.....	32
<b>7. Benutzeroberfläche.....</b>	<b>35</b>
7.1 Tk.....	35
7.2 Die Pflichtenheftverwaltung .....	37
7.3 Der Pflichtenhefteditor.....	45
7.3.1. Kompakter Aufbau .....	46
7.3.2. Fehlertoleranz und Bedienfreundlichkeit von Dialogen .....	47
<b>8. Ausblick .....</b>	<b>51</b>
<b>9. Anhang.....</b>	<b>53</b>
9.1 Abbildungsverzeichnis .....	53
9.2 Literatur verzeichnis .....	54

9.3 Glossar .....	55
-------------------	----

# 1. Einleitung

## 1.1. Allgemeines

Das Pflichtenheft ist ein wesentlicher Bestandteil eines Projektes und basiert auf den Angaben eines Lastenheftes. Das Lastenheft wird auf Seiten des Auftraggebers formuliert und umfasst laut [DIN69905] die „*vom Auftraggeber festgelegte Gesamtheit der Forderungen an die Lieferungen und Leistungen eines Auftragnehmers innerhalb eines Auftrages*“. Laut der oben genannten DIN-Norm umfasst das Pflichtenheft „*die vom Auftragnehmer erarbeiteten Realisierungsvorhaben aufgrund der Umsetzung des vom Auftraggeber vorgegebenen Lastenheftes*“. Der Aufbau und Umfang eines Pflichtenheftes kann je nach Anforderungen an das Projekt variieren. Am gebräuchlichsten ist hierbei der Aufbau nach *Helmut Balzert*, vgl. [Balz96] .

Bei großen Projekten können die zugehörigen Pflichtenhefte sehr umfangreich werden und die Erstellung der Pflichtenhefte viel Zeit in Anspruch nehmen. Hier kann ein Pflichtenheftgenerator die Produktivität steigern. Ein Pflichtenheftgenerator ist ein Programm, das die computergestützte Generierung von Pflichtenheften ermöglicht. Die Eingabe erfolgt über speziell auf die benötigten Informationen zugeschnittenen Eingabemasken. Eingebaute Hilfsfunktionen wie die Prüfung auf Doppeleinträge verhindern mögliche Fehlerquellen im Projekt. Nachdem das Pflichtenheft einmal angelegt wurde, ist es möglich es dieses in mehrere Dateiformate zu exportieren. Die automatische Formatierung des Exportdokumentes nach einer bewährten Grundstruktur ermöglicht es dem Benutzer sich vollständig auf den Inhalt zu konzentrieren.

Software die dem Benutzer diese Möglichkeiten bietet, ist in der Regel nur in Verbindung mit einer kostspieligen Lizenz erhältlich. Ihr Funktionsumfang liegt weit über den Anforderungen eines Benutzers der "einfach" nur ein Pflichtenheft erstellen möchte. Dies begründet sich darin, dass diese Software für große Projekte und damit in der Regel für Unternehmen zugeschnitten ist. Unternehmen vertreiben diese Art von Software zusammen mit technischer Unterstützung oder sonstigen Dienstleistungen die noch zusätzlich angeboten werden<sup>1</sup>. Meistens handelt es sich dabei um Applikationen deren Quelltext nicht zugänglich und somit eine Anpassung oder Optimierung auf Seiten des Benutzers nicht möglich ist. Es gibt allerdings auch Pflichtenheftgeneratoren die offenen Zugang zu ihren Quelltexten bieten. Erwähnenswert ist in dieser Kategorie die Applikation *rmtoo*. Diese Applikation besitzt allerdings keine grafische Benutzeroberfläche<sup>2</sup> und eine Fehlerüberprüfung ist nur über die Konsole möglich. Außerdem unterstützt *rmtoo* kein Exportformat, dass von gängigen Textverarbeitungsprogrammen wie *OpenOffice.org* oder *LibreOffice* weiterverarbeitet werden könnte.

---

<sup>1</sup> *IBM Rational DOORS* - <http://www-142.ibm.com/software/products/de/de/ratidoor>

<sup>2</sup> *fionatel - rmtoo* - <http://www.fionatel.de/projekte/rmtool/>

Einen weiteren wichtigen Punkt stellt die Datenhaltung des Pflichtenheftgenerators dar. Diese sollte möglichst in einem offenen und austauschfreundlichen Format wie *XML* erfolgen um zu ermöglichen, dass die bereits vorhandenen Pflichtenhefte auch durch andere Software weiterverarbeitet werden kann.

## 1.2. Motivation

Während der Studienzeit wurden in den Modulen *Software Engineering 1* und *2* Kenntnisse zur generellen Erstellung von Pflichtenheften erlernt. Während der praktischen Unterrichtsstunden wurde dabei der Pflichtenheftgenerator *PROCESS*<sup>3</sup> von dem Unternehmen *otris* verwendet. Diese Applikation ist ein kommerzieller Pflichtenheftgenerator mit dem Pflichten- und Lastenhefte während des Praktikums im Modul *Software Engineering* generiert wurden. Dieser Generator stand jedoch während des Praktikums nur als Demoversion zur Verfügung. Die Einschränkungen der Demoversion bestanden darin, dass sich die Applikation nach einer Laufzeit von 15 Minuten selbstständig schloss und pro Textabsatz die Eingabe auf 250 Zeichen beschränkt war. In den späteren Demoversionen von *PROCESS* ging man dazu über, Inhalte von Texten ab einer Länge von 100 Zeichen im exportierten Pflichtenheft abzuschneiden. Die Seiten des exportierten Dokumentes waren versehen mit einem Wasserzeichen der Firma *otris* und dem Hinweis, dass es sich bei der verwendeten Version um eine Demoversion handelte. Bis heute wird keine Demoversion für das Betriebssystem GNU/Linux angeboten, lediglich die Kaufversion für 178 *Euro*, ohne Studentenrabatt. Trotz der genannten Einschränkungen war *PROCESS* sehr benutzerfreundlich gehalten. Die Erstellung von Pflichtenheften war mit Hilfe von Eingabemasken und intuitiven Dialogen angemessen zu bewältigen.

Im späteren Verlauf des Studiums, bot sich im Rahmen einer schriftlichen Ausarbeitung vom Modul Skriptsprachen die Möglichkeit einen Pflichtenheftgenerator zu programmieren. Dieser sollte den Export von Pflichtenheften in einem Dateiformat ermöglichen, dass von gängigen Textverarbeitungsprogrammen unterstützt wird. Die Ausarbeitung wurde in einer Zusammenarbeit von Cagdas Ünlüer, Phouc-Tou Chau und Alexandros Kechagias ( die letztgenannte Person ist der Autor dieser Diplomarbeit ) erstellt. Bereits während der Ausarbeitung stand fest, dass der erstellte Pflichtenheftgenerator um weitere Funktionen erweitert werden sollte um die *Open Source Community* mit diesem Softwarewerkzeug zu unterstützen. Demzufolge lag es Nah dieses reichlich Potenzial bietende Projekt innerhalb einer Diplomarbeit weiterzuentwickeln.

Die in der Skriptsprachenausarbeitung designte grafische Benutzeroberfläche hatte bereits Ähnlichkeit mit der aktuellen Oberfläche. *Moliri* entstand auf Basis dieser Skriptsprachenausarbeitung. Die Motivation dieser Diplomarbeit bestand darin, einen frei verfügbaren Pflichtenheftgenerator mit grafischer Benutzeroberfläche für GNU/Linux zu erstellen, da es eine solche Applikation nicht frei verfügbar gab. Dieser sollte auf offenen Dokumentenstandards wie *XML* oder dem *Open Document Format* aufbauen und somit mit bereits bestehenden

---

<sup>3</sup>otris software AG - *PROCESS* - [https://www.otris.de/cms/Pflichtenheftgenerator\\_otris.html](https://www.otris.de/cms/Pflichtenheftgenerator_otris.html)

*Parsern* oder Textverarbeitungsprogrammen einen leichten Zugang und Portabilität sicherstellen.

### 1.3. Übersicht

Im folgenden Kapitel werden genutzte Technologien und Programme vorgestellt. Es wird auf die genutzte Programmiersprache sowie die Entwicklungsumgebung eingegangen. Dabei wird ein Überblick über die Funktionalität der Entwicklungsumgebung gegeben sowie die verwendeten *Plugins*.

Das dritte Kapitel beschreibt die durchgeführte Qualitätssicherung des Quelltextes. Es wird das Werkzeug zur statischen Quelltextanalyse *Perl::Critic* und seine Konfiguration innerhalb des Projektes besprochen sowie der Quelltextformatierer *Perl::Tidy*.

Im vierten Kapitel folgt die Beschreibung der Datenhaltung. Diese beinhaltet den Aufbau des Projektordners und eine Übersicht über die XML-Module von Perl. Anschließend folgen die Implementierung und der Aufbau der Datenhaltung.

Auf die genutzten Exportformate wie ODF und Textdateien wird im fünften Kapitel eingegangen und diese verglichen. Dies ist der wichtigste Teil der Diplomarbeit; Hier wird auf die Generierung der Pflichtenhefte und die damit verbundenen Module eingegangen. Das Kapitel schließt mit der Präsentation der exportierten Dokumente des ODT- und Text-Formates ab.

Die Benutzeroberfläche der Pflichtenheftverwaltung sowie des Pflichtenhefteditors wird im sechsten Kapitel besprochen. Weiterhin werden noch Besonderheiten von ausgewählten Dialogen sowie die eingesetzten Programmiertechniken zu deren Umsetzung erläutert.

Im letzten Kapitel bietet der Ausblick noch mögliche Erweiterungen und Ergänzungen zu *Moliri* an.

## 2. Technologien und Programme

### 2.1 Programmiersprache

Die Programmiersprache *Perl* in der Version 5.10.1 wurde zur Implementierung dieser Diplomarbeit verwendet. Die Vorteile von *Perl* sind zum einem, eine gute Portabilität auf eine Vielzahl von Plattformen<sup>4</sup>, eine hohe Flexibilität und Erweiterbarkeit durch Module die auf *CPAN* bereitgestellt werden und vor allem, dass *Perl* und viele zugehörige Module freie Software sind, die der *GNU* oder alternativ der *Artistic License* von Larry Wall unterliegen. Weiterhin bietet *Perl* viele Strukturmechanismen die bei größeren Projekten hilfreich sind. Da die zu erstellende Applikation unter *GNU*-Lizenz auf *GNU/Linux* Systemen, speziell unter der Distribution *Ubuntu* lauffähig sein soll, ist die hohe Verbreitung unter diesen Systemen ein ausschlaggebender Punkt zur Wahl von *Perl*.

### 2.2 Entwicklungsumgebung

#### 2.2.1 Vim

Der Texteditor *Vim*<sup>5</sup> in der Version 7.2.330, mit dem zusätzlichen *Plugin perl-support.vim*<sup>6</sup> in der Version 4.11 diente während der Implementierung dieser Diplomarbeit als Entwicklungsumgebung. *Vim* zeichnet sich besonders durch seine Spezialisierung beim Bearbeiten von Texten aus. Der Editor *Vim* unterstützt drei unterschiedliche Bearbeitungsmodi, den Befehlsmodus, den Einfügemodus und den Kommandozeilenmodus.

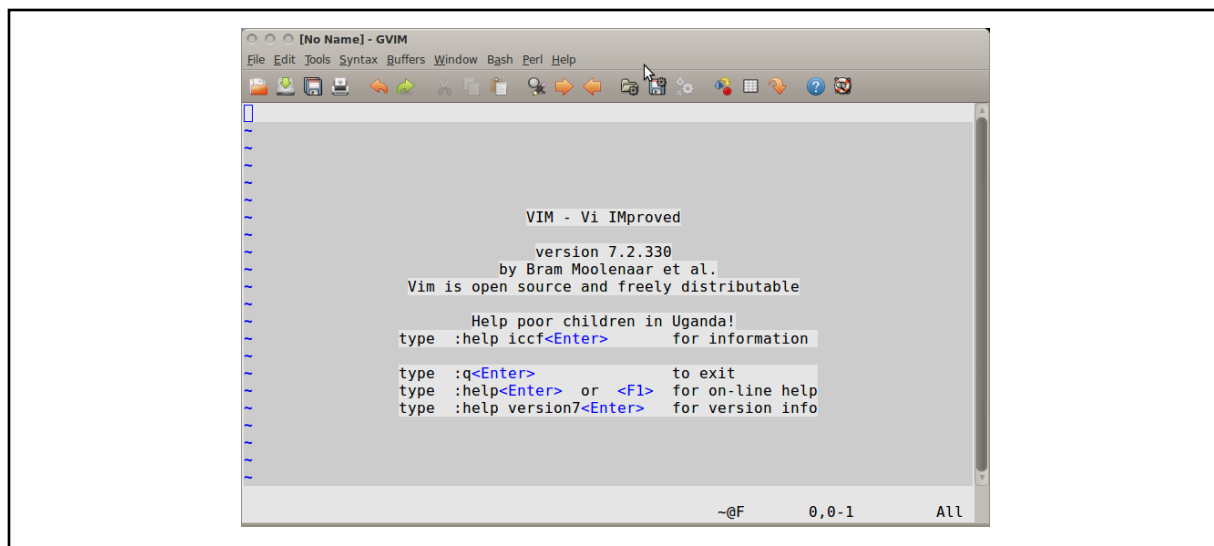


Abbildung 1 : VIM - Vi IMproved

<sup>4</sup>Plattformunterstützung Perl - <http://perldoc.perl.org/perlport.html#Supported-Platforms>

<sup>5</sup>Bram Moolenaar – *Vim* - <http://www.vim.org/download.php>

<sup>6</sup>Fritz Mehner - *perl-support* - [http://vim.sourceforge.net/scripts/script.php?script\\_id=556](http://vim.sourceforge.net/scripts/script.php?script_id=556)



Im **Befehlsmodus** liegt die eigentliche Stärke dieses Editors. Dieser als Standard eingestellte Modus ermöglicht das Ausführen verschiedener Befehle durch einfache Tasteneingabe und ersetzt die in gewöhnlichen Editoren vorkommenden Makros, die meist durch Kombinationen mehrerer Tasten ähnliche Funktionen bieten. Ein klarer Nachteil ist ein höherer Einarbeitungsaufwand, was später mit einer höheren Effizienz bei der Entwicklung belohnt wird. Vor allem die Kombination mehrerer Befehle ermöglicht eine elegante und schnelle Möglichkeit Programmieraufgaben zu lösen. Befehle können miteinander kombiniert werden. Das ermöglicht das Verfassen von komplexen Befehlen um Bearbeitungs-Aufgaben die während der Programmierung aufkommen elegant und schnell zu lösen.

Der **Bearbeitungsmodus** ermöglicht eine normale Eingabe von Texten, wie es aus üblichen Editoren bekannt ist.

Im **Kommandozeilenmodus** können Befehle wie beispielsweise, das Suchen und Ersetzen von beliebigen Textpassagen mit Hilfe von regulären Ausdrücken oder auch einfache Aufgaben wie Speichern und Beenden ausgeführt werden.

## 2.2.2 perl-support.vim

Durch die Verwendung des Plugins *perl-support.vim* wird aus dem vielseitigen Texteditor *Vim* eine Perl-Entwicklungsumgebung.

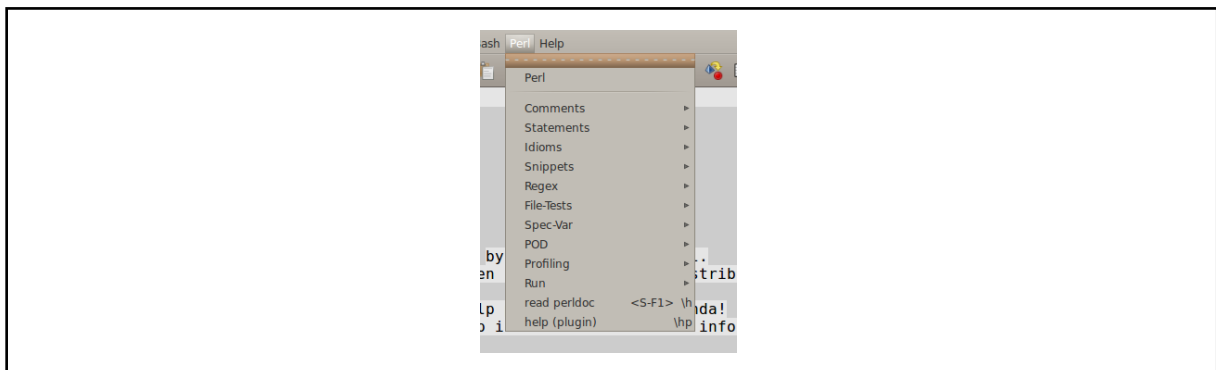


Abbildung 2: perl-support.vim

Häufig wiederkehrende Standardausdrücke und Kontrollstrukturen wie beispielsweise Schleifen, *Filehandler* und Kommentare können mit geringem Aufwand in den Quelltext, mithilfe von Tastenkürzel, eingebunden werden. Weitere nützliche Hilfsmittel sind die Autovervollständigung, der integrierte Zugriff auf die Dokumentation von Ausdrücken aus der Perl-Standardbibliothek, sowie die Ausführung des momentan verwendeten Quelltextes.

### 3. Qualitätssicherung

*Perl* verfolgt die *TIMTOWTDI*-Philosophie (*There Is More Than One Way To Do It*). Dem Programmierer wird kein bestimmter Programmierstil aufgezwungen. Dies erfordert jedoch viel Disziplin bei der Einhaltung von bewährten Standards in der *Perl*-Programmierung und kann sich bei Nichteinhaltung merklich auf die Lesbarkeit und somit auch auf die Wartbarkeit des Programmes auswirken. Um die grundsätzliche Einhaltung dieser Standards zu gewährleisten wurden Hilfsmittel wie *Perl::Critic*<sup>7</sup> und *Perl::Tidy*<sup>8</sup> in den Entwicklungsprozess eingebunden.

#### 3.1 Perl::Critic

**Perl::Critic** ist ein Werkzeug zur statischen Quelltextanalyse. Es analysiert den Quelltext auf Einhaltung von Programmierrichtlinien, die in [Conw05] nachgeschlagen werden können. Es wurden darüber hinaus noch andere bewährte Programmierrichtlinien in diesem Modul berücksichtigt. Das Plugin *perl-support.vim* integriert diese Analyse in *Vim*.

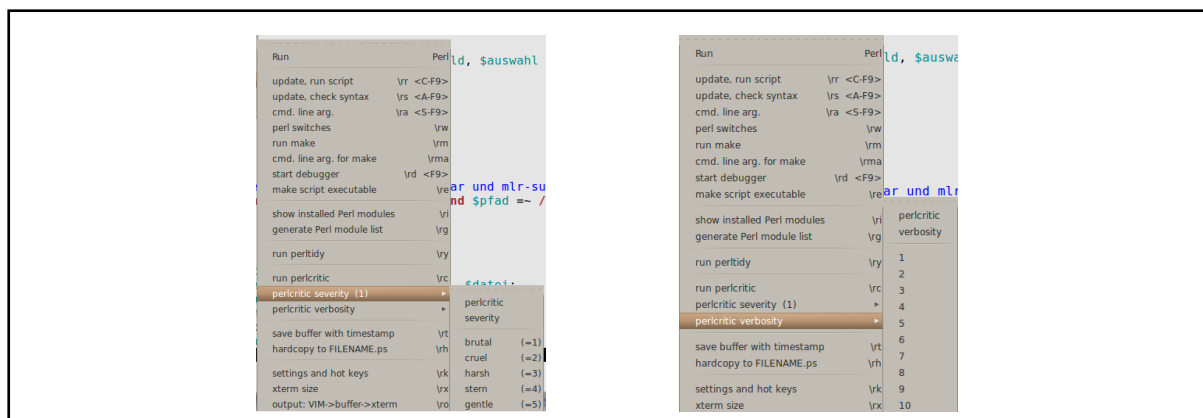


Abbildung 3 : Perl::Critic-Konfiguration in perl-support.vim

Innerhalb des Editors *Vim* ist die Möglichkeit gegeben grundsätzliche Einstellungen vorzunehmen. Unterschieden wird hierbei in fünf Stufen bei der Strenge (*severity*)<sup>9</sup> und zehn Stufen bei der Ausführlichkeit (*verbosity*) der Meldung bei Verstoß gegen eine Richtlinie.

*Perl::Critic* selber beinhaltet diese Richtlinien nicht, sondern analysiert nur den Quelltext. Auf Basis dieser Analyse werden die Richtlinien in den von *Perl::Critic::Policy* abgeleiteten Modulen bestimmt. Die Basisinstallation von *Perl::Critic* umfasst bereits eine große Anzahl dieser Module. Abhängig von der ausgewählten Stufe wird eine Gruppe von Modulen aktiviert. In der fünften Stufe (*gentle*) werden nur die größten Verstöße gemeldet. In der vierten Stufe

<sup>7</sup>Elliot Shank - *Perl::Critic* - <http://search.cpan.org/~elliots/Perl-Critic-1.116/lib/Perl/Critic.pm>

<sup>8</sup>Steve Hancock - *Perl::Tidy* - <http://perltidy.sourceforge.net/>

<sup>9</sup>*Perl::Critic* - Konfiguration - <http://search.cpan.org/~elliots/Perl-Critic-1.116/lib/Perl/Critic.pm#CONFIGURATION>

werden alle Verstöße aus der fünften und aus der vierten Stufe gemeldet. Für alle weiteren Stufen bis hinab zur ersten und damit strengsten Stufe verläuft dieses Verfahren analog.

Die folgende Abbildung zeigt eine solche Auflistung von Richtlinienverstößen innerhalb der Datei *moliri2\_verwaltung.pl*.

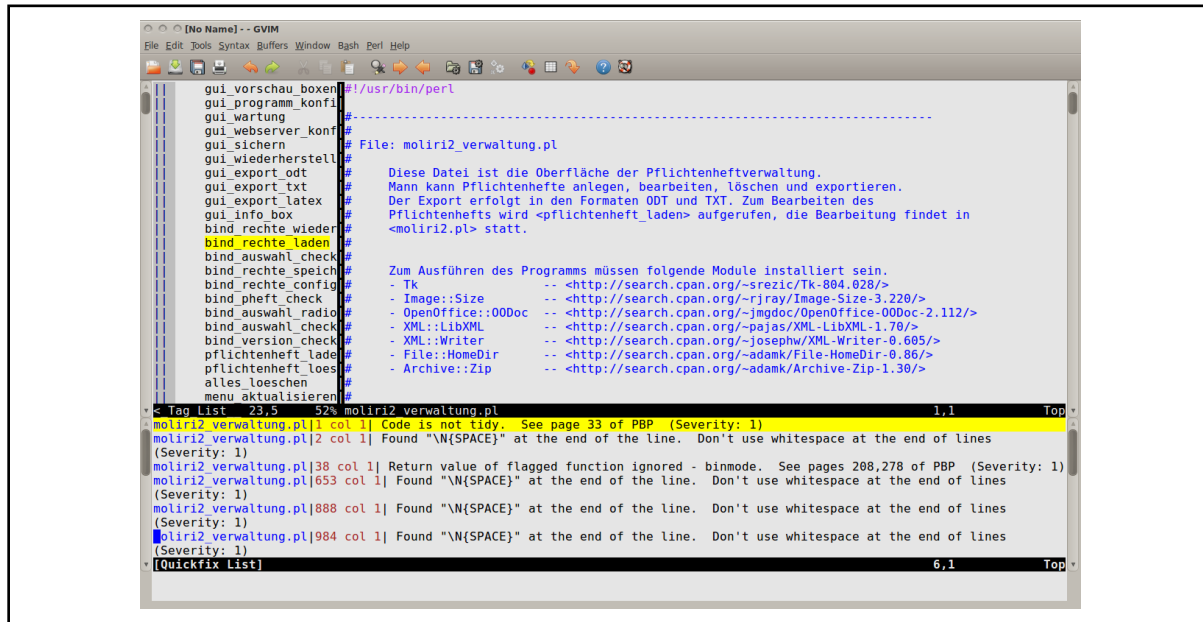


Abbildung 4 : Beispiel von Richtlinienverstößen

Wie hier zu sehen ist, wird der Programmierer benachrichtigt, dass der Quelltext gegen festgelegte Richtlinien verstößt. Wichtig ist hierbei die Art, die Zeilennummer und Stufe in der der Verstoß ausgelöst wird.

Es werden aber auch Verstöße gemeldet die, im Rahmen dieses Projektes, nicht oder nur wenig Sinnvoll sind. Daher ist die Konfiguration von *Perl::Critic* anhand einer Konfigurationsdatei namens *perlcriticrc* notwendig. Diese Datei muss sich im selben Ordner befinden wie die zu überprüfende Datei. In diesem Projekt wurden zwei solcher Konfigurationsdateien erstellt; Eine für die im Hauptordner befindlichen Dateien und eine weitere für die selbstgeschriebenen Klassen im *lib*-Ordner.

Es folgt nun ein Ausschnitt aus einer solchen *perlcriticrc*-Datei aus dem Projektordner:

```
#-----
# Dokumentation wird mit Natural Docs generiert
#-----
[-Documentation::PodSpelling]
[-Documentation::RequirePackageMatchesPodName]
[-Documentation::RequirePodAtEnd]
[-Documentation::RequirePodSections]
```

Eines der Richtlinien von *Perl::Critic* ist, dass der Quelltext dokumentiert sein muss<sup>10111213</sup>. Vorzugsweise in der *Perl* eigenen POD. In diesem Projekt wurde allerdings *Natural Docs*<sup>14</sup> zum dokumentieren bevorzugt. Dies macht jedoch die Überprüfung und Meldung dieser Richtlinien überflüssig und birgt zudem die Gefahr wichtigere Verstöße in der Ausgabe zu übersehen. Demzufolge werden diese Richtlinien mit einem führenden Minus-Zeichen deaktiviert. Entsprechend wurden auch andere solcher Richtlinien ausgeschaltet.

Grundsätzlich wurde in diesem Projekt darauf geachtet die meisten dieser Richtlinien bis hin zu Stufe 1 (*bruta*), soweit möglich, in allen Quelltextdateien einzuhalten.

## 3.2 Perl::Tidy

**Perl::Tidy** ist ein Werkzeug zur automatischen Quelltextformatierung. Eine gleichbleibende Formatierungsart die sich durch alle Quelltextdateien zieht, kann merklich zur Lesbarkeit und somit auch Wartbarkeit des Programms beitragen. Gute Formatierung im Quelltext ist ein Zeichen von professioneller Programmierung und wurde während des gesamten Projektes eingehalten.

---

<sup>10</sup>Documentation::PodSpelling - <http://search.cpan.org/dist/Perl-Critic/lib/Perl/Critic/Policy/Documentation/PodSpelling.pm>

<sup>11</sup>Documentation::RequirePackageMatchesPodName - <http://search.cpan.org/dist/Perl-Critic/lib/Perl/Critic/Policy/Documentation/RequirePackageMatchesPodName.pm>

<sup>12</sup>Documentation::RequirePodAtEnd - <http://search.cpan.org/dist/Perl-Critic/lib/Perl/Critic/Policy/Documentation/RequirePodAtEnd.pm>

<sup>13</sup>Documentation::RequirePodSections - <http://search.cpan.org/dist/Perl-Critic/lib/Perl/Critic/Policy/Documentation/RequirePodSections.pm>

<sup>14</sup>Greg Valure - *Natural Docs* - <http://www.naturaldocs.org/about.html>

## 4. Datenhaltung

Die Datenhaltung von Moliri erfolgt mittels *XML*-Dateien. Die *XML*-Dateien beinhalten die einzelnen Pflichtenheftversionen. Diese Pflichtenheftversionen befinden sich innerhalb einer Ordnerstruktur. Die jeweiligen Unterordner fassen die einzelnen Versionen zu den dazugehörigen Pflichtenheften zusammen. Somit ist die folgende beispielhafte Ordnerstruktur gegeben, die nachfolgend betrachtet werden soll.







 Pflichtenheftordner	2 items	folder
 Brettspiele	2 items	folder
 1.0.xml	17,9 KB	XML document
 2.0.xml	19,9 KB	XML document
 Testheft	1 item	folder
 1.0 pre.xml	6,2 KB	XML document

Abbildung 5 : Pflichtenheftordner - Struktur

Der Pflichtenheftordner beinhaltet zwei Unterordner für Pflichtenhefte mit mehreren zugehörigen Pflichtenheften (vgl. Abb. 5). Die Benennung der Pflichtenhefte, geht aus der Ordnerbezeichnung hervor, somit lautet die Bezeichnung des ersten Pflichtenhefts "Brettspiele" und des zweiten "Testheft". Das Pflichtenheft "Brettspiele" ist in der Version 1.0 und 2.0 vorhanden, das Pflichtenheft "Testheft" in der Version 1.0 pre. Diese Ordnerstruktur hat den Vorteil der leichten Verarbeitung, da das Einfügen von neuen Versionen und Ihre Zuordnung zu den Pflichtenheften auf der Ebene des Dateisystems erfolgt. Zum Einfügen eines neuen Pflichtenheftes muss lediglich ein Ordner mit der Bezeichnung des Pflichtenheftes und eine XML-Datei mit dem Versionsnamen angelegt werden. Somit entstehen keine Metadaten die zusätzlich gepflegt werden müssen.

### 4.1 XML

„XML ist wie die universell einsetzbaren Plastikbehälter von *Tupperware*®. Es gibt wirklich keine bessere Möglichkeit Ihre Nahrung frisch zu halten als mit diesen bunten, luftdichten kleinen Behältern. Es gibt sie in verschiedenen Farben und Formen so, dass Sie sich aussuchen können welche am besten passt.“

Erik T. Ray – O'Reilly: *Learning XML*, 2nd Edition, Kapitel 1.1

Die erweiterbare Auszeichnungssprache *XML* dient zur Darstellung von strukturierten Informationen wie Dokumente, Konfigurationsdateien, Bücher und Vielem mehr. XML ist ein einfaches textbasiertes Format und basiert auf einem älteren Standard, dem SGML-Format.<sup>15</sup>

<sup>15</sup>vgl. <http://www.w3.org/TR/REC-xml/#sec-origin-goals>

Das Ziel beim Entwurf von XML war, dass eine Erstellung und Verarbeitung von Dokumenten mit geringem Aufwand möglich sein sollte. Weiterhin sollte XML für den Benutzer leicht lesbar und verständlich sein<sup>16</sup>. Der XML-Standard ermöglicht es dem Entwickler auf schnelle und einfache Art eine Datenstruktur zu definieren die sehr portabel, gleichzeitig aber auch einfach erweiterbar ist.

## 4.2 Perl und XML

Es gibt eine große Anzahl von *Perl*-Modulen die sich auf die Verarbeitung von XML spezialisiert haben. Diese Module benutzen alle einen Parser. Der Parser ist ein Programm, das die XML-Daten analysiert und in eine verwertbare Form bringt.

Dabei kann unterschieden werden zwischen Modulen deren *Parser* in Perl geschrieben wurde oder Modulen, die eine Schnittstelle, auch Interface genannt, zu einem in beispielsweise der Programmiersprache C geschriebenen Parser bieten. Erstere finden in der Praxis kaum Anwendung, da der einzige Vorteil den sie den binären Parsern gegenüber haben, die einfache Installation und Portabilität, durch die hohe Verbreitung von binären Parsern auf nahezu allen Systemen, nicht mehr von Bedeutung ist. Hinzu kommt, dass dynamische Programmiersprachen die erst einmal interpretiert und dann in Maschinencode umgewandelt werden müssen, wie *Perl* eine ist, generell nicht so eine Performanz aufweisen wie Programme, die bereits im binären Format vorliegen.

Die gebräuchlichsten Parser in der Praxis sind demnach die zuletzt genannten. In der *GNU/Linux* beziehungsweise *Unix*-Welt sind die beiden weit verbreitetsten *expat*<sup>17</sup> und *libxml2*<sup>18</sup>.

**Expat** verarbeitet XML in Form eines Datenstroms (*streams*), dabei werden *Events* ausgelöst wenn XML-Elemente wie zum Beispiel *Tags*, Kommentare oder der Inhalt zwischen diesen Elementen (*content*) verarbeitet werden. Diese Events müssen *Eventhandlern* zugewiesen werden, die dann bestimmen was mit diesen Elementen geschieht. Diese *Handler* sind Funktionen, in Perl auch Subroutinen genannt, die der Programmierer dann implementieren muss.

**GNOMEs Libxml2** bietet verschiedene Möglichkeiten. Es kann XML in Form einer Baumstruktur verarbeiten. Dabei wird das gesamte XML-Dokument, in Form eines *DOM*-Objektes, in den Arbeitsspeicher geladen. Der Zugriff auf die Daten erfolgt dann mit *XPath*. Es kann auch zudem mit *SAX2 XML* in Form eines Datenstroms bearbeiten; Dies funktioniert dann ähnlich wie bei *expat*.

Da beim Arbeiten mit *DOM* das gesamte XML-Dokument auf einmal eingelesen wird, hat *DOM* einen deutlich höheren Speicherverbrauch als *expat*, welches nur die Daten im Speicher behält, die es auch wirklich braucht. Während bei *expat* nur mit *Handlern* und *Events*

<sup>16</sup>vgl. <http://www.w3.org/standards/xml/core>

<sup>17</sup>*expat* - <http://www.libexpat.org/>

<sup>18</sup>GNOME Projekt - *libxml2* - <http://xmlsoft.org/>

auf die Daten zugegriffen werden kann, hat *LibXML2* die deutlich komfortablere und intuitivere Möglichkeit der *XPath*-Ausdrücke.

*XPath*-Ausdrücke beschreiben den Pfad der im *XML*-Dokument zum gewünschten Element führt, was in der Regel ein Knoten ist. Diese Pfade sind den Dateisystempfaden auf unixoiden Betriebssystemen sehr ähnlich.

Das Zielsystem für den Pflichtenheftgenerator ist das Betriebssystem Ubuntu. Aufgrund seiner hohen Verbreitung und da Ubuntu in der Standardinstallation als Desktopoberfläche GNOME anbietet und somit bereits *libxml2* sowie das Perl-Modul *XML::LibXML*<sup>19</sup> standardmäßig auf allen Ubuntu-Systemen installiert ist, fiel die Wahl auf *libxml2* und somit auch auf das Modul *XML::LibXML*.

Das erstellen eines *XML*-Dokumentes ist die einfachere Aufgabe. Dazu hätte prinzipiell auch *XML::LibXML* verwendet werden können, doch die intuitive Schnittstelle sowie die klare Syntax von *XML::Writer*<sup>20</sup> überzeugte bei einer durchgeführten Machbarkeitsstudie und somit wurde das Modul *XML::Writer* im Verlauf der Implementierung verwendet.

### 4.3 Implementierung

Die *XML*-Struktur des Pflichtenheftes ist unkompliziert. Es wurden keine Attribute verwendet; Alle nötigen Informationen befinden sich innerhalb der *Tags*. Die Anordnung der *XML*-Elemente entspricht dem der Benutzeroberfläche. Der Zugriff auf die *XML*-Dokumente erfolgt über die Klasse *MoliriXML.pm*, die sich im *lib*-Ordner (Bibliothekenordner) des Projektes befindet. Der Quelltext dieser Klasse beinhaltet Methoden die das Erstellen, Einlesen und Versionieren von Pflichtenheften ermöglicht.

Bei der **Neuanlage oder Veränderung** eines Pflichtenheftes wird ein Objekt vom Typ *MoliriXML* erstellt, dem der gewünschte Speicherort als Pfad übergeben wird. Ist ein Pflichtenheft bereits existent, wie in dem Fall einer Veränderung, wird das vorhandene Pflichtenheft überschrieben. Von diesem Objekt aus kann auf die Methode *export\_xml* zugegriffen werden. Als Parameter werden die Daten vom Pflichtenheft erwartet. Mit dem übergebenen Pfad wird ein Filehandler erstellt, mit dem Perl auf das Dateisystem zugreifen kann. Nun wird der Klasse *XML::Writer* der Filehandler übergeben und somit das *XML*-Dokument erzeugt.

Als erster Schritt muss die Zeichenkodierung des *XML*-Dokumentes festgelegt werden. Als Zeichenkodierung wird mit der Methode *xmlDecl* der *XML*-Standard UTF-8 angegeben<sup>21</sup>

```
$writer->xmlDecl('UTF-8');
```

<sup>19</sup>Shlomi Fish - *XML::LibXML* <http://search.cpan.org/~shlomif/XML-LibXML/>

<sup>20</sup>Joseph Walton - *XML::Writer* <http://search.cpan.org/~josephw/XML-Writer/>

<sup>21</sup><http://www.w3.org/TR/REC-xml/#charsets>

Als nächstes wird das Wurzelement angelegt.

```
$writer->startTag('Pflichtenheft');
```

Die XML-Ausgabe erfolgt über die Funktionen *startTag*, *characters*, *endTag*

```
$writer->startTag('Titel');
$writer->characters($titel);
$writer->endTag('Titel');
```

Sofern, wie in diesem Beispiel keine weiteren *XML*-Elemente verschachtelt werden und zwischen dem Anfangs- und Endtag nur noch abzuspeichernde Daten enthalten sind, kann auch die vereinfachte Schreibweise von *dataElement* benutzt werden. Diese ist kürzer und trägt ein großes Stück zur Übersichtlichkeit des Quelltextes bei.

```
$writer->startTag('Pflichtenheftdetails');

$writer->dataElement( 'Titel',    $titel );
$writer->dataElement( 'Version', $version );
$writer->dataElement( 'Autor',   $autor );
$writer->dataElement( 'Datum',   $datum );
$writer->dataElement( 'Status',  $status );
$writer->dataElement( 'Kommentar', $kommentar );

$writer->endTag('Pflichtenheftdetails');
```

Das erzeugte XML sieht dann wie folgt aus:

```
<Pflichtenheftdetails>
  <Titel>Brettspiele</Titel>
  <Version>1.0</Version>
  <Autor>Alexandros Kechagias</Autor>
  <Datum>2011-7-20</Datum>
  <Status>in Arbeit</Status>
  <Kommentar>Dieses Pflichtenheft dient nur zu Demonstrationszwecken.</Kommentar>
</Pflichtenheftdetails>
```

Das Wurzel-Element wird mit *end* geschlossen.

```
$writer->end();
```

Zum **Einlesen eines Pflichtenheftes** wird ebenfalls ein Objekt vom Typ *MoliriXML* erstellt, dem der gewünschte Ort des einzulesenden Pflichtenheftes übergeben wird. Von diesem Objekt aus kann auf die Methode *import\_xml* zugegriffen werden. Diese erwartet keine Parameter. Der Rückgabewert dieser Funktion ist das gesamte Pflichtenheft als *DOM*-Objekt.

Die gewünschten Elemente aus der XML-Datei werden über XPath-Pfade erreicht. In dem folgenden Beispiel wird das Einlesen der Pflichtenheftdetails demonstriert.



```
my $titel      = $dom->findnodes('//Pflichtenheft/Pflichtenheftdetails/Titel');
my $version   = $dom->findnodes('//Pflichtenheft/Pflichtenheftdetails/Version');
my $autor     = $dom->findnodes('//Pflichtenheft/Pflichtenheftdetails/Autor');
my $datum     = $dom->findnodes('//Pflichtenheft/Pflichtenheftdetails/Datum');
my $status    = $dom->findnodes('//Pflichtenheft/Pflichtenheftdetails/Status');
my $kommentar = $dom->findnodes('//Pflichtenheft/Pflichtenheftdetails/Kommentar');
```

Die Methode *findnodes* gibt ein Element vom Typ *XML::LibXML::NodeList* zurück. In diesem Fall ist es nur ein einzelnes XML-Element da hier keine weitere Verschachtelung vorliegt. Der Inhalt dieses Elements kann nun über die Methode *string\_value* ausgelesen werden.

```
$pflichtenheft{'details'} = {
    'titel'      => $titel->string_value(),
    'version'   => $version->string_value(),
    'autor'     => $autor->string_value(),
    'datum'     => $datum->string_value(),
    'status'    => $status->string_value(),
    'kommentar' => $kommentar->string_value()
};
```

## 5. Exportformate

Der Export ist eine der wichtigsten Funktionalitäten dieser Diplomarbeit. Als Exportformat waren *ODF*, *TeX* und das Textformat vorgesehen.

### 5.1 Das ODF-Format

Das *ODF*-Format ist ein offenes *XML*-basiertes Dokumenten-Format für Textverarbeitungssoftware und es soll für Dokumente benutzt werden die Text, Tabellenkalkulationen, Diagramme und grafische Elemente verwenden. Das Dateiformat vereinfacht die Umwandlungen in andere Formate durch Nutzung und Wiederverwendung von vorhandenen Standards.<sup>22</sup> Es ist ein international genormter quelloffener Standard für Dateiformate von Bürodokumenten wie z.B. Texte, Tabellendokumente, Präsentationen, Zeichnungen, Bilder und Diagramme. Dabei wurde darauf geachtet, dass das verwendete *XML* den *XML*-Spezifikationen vom *W3C* entspricht. Der Inhalt und die Formatierung wurden dabei strikt getrennt, so dass sie unabhängig voneinander bearbeitet werden können.<sup>23</sup>

Als Basis für dieses Projekt diente der *OASIS Open Document Format for Office Applications (OpenDocument) v1.0*<sup>24</sup>

#### 5.1.1 Die ODT-Datei (Package)

Aus der Not heraus, dass *XMLs* Architekturschema nicht das Einbinden von binären Objekten unterstützt und auch keine Möglichkeiten bietet, die Daten zu komprimieren, ist das Dateiformat *ODT* entstanden. *ODT*, wird in der *OpenDocument*-Spezifikation<sup>24</sup> auch als *Package* bezeichnet. Es ermöglicht dem Entwickler alle notwendigen *XML*-Daten mit ihren dazugehörigen binären Daten, wie *OLE*-Objekte, in ein *ZIP*-Archiv zu verpacken; Optional auch zu komprimieren, da unkomprimierte *XML*-Dateien sehr groß werden können.

Eine *ODT*-Datei hat laut der *OASIS Open Document Spezifikation*<sup>24</sup> folgenden Aufbau:

<i>/META-INF/Manifest.xml</i>	Diese Datei enthält eine Auflistung aller Dateien zusammen mit ihrem Dateityp. Sollte eine dieser Dateien verschlüsselt sein, muss die benötigte Information zum entschlüsseln sich in <i>Manifest.xml</i> befinden. Damit sind die Informationen, wie die Art der Verschlüsselung und alle Informationen die Textverarbeitungssoftware bräuchte um Mithilfe des Passwortes diese Datei zu entschlüsseln.
<i>/Thumbnails/thumbnail.png</i>	Ein Miniaturbild der ersten Seite des Dokuments. Dieses sollte beim ersten Speichern erstellt werden. Um maximale Kompatibilität zu allen Dateimanagern zu gewährleisten darf es keine

<sup>22</sup>vgl. *OASIS* - [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=office#odf11](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office#odf11)

<sup>23</sup>vgl. *OASIS* - <http://www.oasis-open.org/committees/office/charter.php>

<sup>24</sup>[PDF] *ODF-Spezifikationen* - <http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf>

Effekte, umschließende Rahmen oder Ränder haben. Sollte die Datei verschlüsselt sein, darf das Bild keinen Inhalt der Datei preisgeben und kenntlich machen, dass Sie verschlüsselt ist. Die Bilddatei muss mit dem *Thumbnail Managing Standard*<sup>25</sup> konform sein.

*/Pictures/000001201.png*

Sollten Bilddateien in die Datei importiert werden, werden sie im Unterverzeichnis unter Pictures gespeichert. Dies ist bei Moliri jedoch nicht der Fall, Bilddaten werden hier nur verlinkt, da bei einem Import die Textverarbeitungssoftware ansonsten einen Fehler des Dokumentes meldet.

*/mimetype*

Dies ist eine Datenkette, die am Anfang eines jeden ODT-Dokumentes stehen sollte ("SHOULD"<sup>26</sup>), es sei denn es liegen gute Gründe vor, von dieser Empfehlung nicht einzuhalten. Das ODT-Dokument sollte folgenden Aufbau haben:

```

0000000: 504b 0304 1400 0008 0000 b2b6 f23e 5ec6  PK.....>^
0000010: 320c 2700 0000 2700 0000 0800 0000 6d69  2.'...'.mi
0000020: 6d65 7479 7065 6170 706c 6963 6174 696f  metypeapplicatio
0000030: 6e2f 766e 642e 6f61 7369 732e 6f70 656e  n/vnd.oasis.open
0000040: 646f 6375 6d65 6e74 2e74 6578 7450 4b03  document.textPK.

```

**Abbildung 6** : Hex-Ansicht eines ODT-Dokumentes mit Vim

Es sei an dieser Stelle darauf hingewiesen, dass die Zeilennummern zur Basis 16, also Hexadezimal sind. Somit ist  $10_{16} = 16_{10}$ ,  $20_{16} = 32_{10}$ ,  $30_{16} = 48_{10}$ ,  $40_{16} = 64_{10}$ .

**Position  $0_{10}$ :** Wie in ZIP-Dateien üblich sollte auch hier an erster Stelle die Zeichenkette "PK" stehen.

**Position  $30_{10}$ :** Alle ODF-Dokumente sollten an dieser Stelle die Zeichenkette "mimetype" haben

**Position  $38_{10}$ :** Hier kommt die Datenkette, deren Inhalt abhängig vom Typ des ODF-Dokumentes ist. Im Falle von Moliri lautet die zugehörige Datenkette "application/vnd.oasis.opendocument.text".

*/content.xml*

Inhaltsbereich der ODT-Datei. Der Inhaltsbereich enthält den textuellen Inhalt der für den Benutzer sichtbar ist.

*/styles.xml*

Formatierungsbereich des ODT-Dokumentes. Der Formatierungsbereich bestimmt wie der textuelle Inhalt dargestellt wird.

<sup>25</sup>Thumbnail Managing Standard - <http://jens.triq.net/thumbnail-spec/index.html>

<sup>26</sup>[Textdatei]RFC2119 - SHOULD - <http://www.ietf.org/rfc/rfc2119.txt>

<i>/meta.xml</i>	Der Metabereich enthält allgemeine Informationen über das Dokument wie die Anzahl der Seiten oder den Namen des Urhebers.
<i>/settings.xml</i>	Konfigurationen bezüglich der Textverarbeitungssoftware, haben keine Bedeutung für das Dokument selber, sondern nur für die Software die es erstellt. Dies sind Informationen wie beispielsweise die Größe des Fensters oder die Druckereinstellungen.

### 5.1.1.1 content.xml

Im folgenden Kapitel wird näher auf die Datei *content.xml* eingegangen. Das Wurzel-Element ist immer das Element `<office:document-content>`. Optional werden mit dem Element `<office:script>` auch Skripte eingebunden; Dies geschieht durch einen Verweis auf eine externe Datei. Im Rahmen der Implementierung von Moliri war kein einbinden von Skripten notwendig.

Im Abschnitt `<office:font-face-decls>` werden die in dem Dokument verwendeten Schriftarten angegeben. In diesem Beispiel die Schriftart *Times New Roman*. Das folgende XML-Element ist ein Beispiel einer solchen Angabe:

```
<style:font-face style:name="Times New Roman" svg:font-family="'Times New Roman'"
  style:font-family-generic="roman" style:font-pitch="variable" />
```

Im Abschnitt `<office:automatic-styles>` werden alle Formatierungen definiert die automatisch Strukturen innerhalb des Dokumentes zugewiesen werden. Diese Strukturen können Paragraphen oder normaler Text sein. Je nach Textverarbeitungssoftware haben diese bei ihrer Erstellung unterschiedliche Voreinstellungswerte, die direkt nach der Erstellung zugewiesen werden. Das nachstehende Beispiel zeigt eine automatisch erzeugte Formatierung:

```
<style:style style:family="paragraph" style:name="P7" style:parent-style-name="Standard">
  <style:paragraph-properties fo:text-align="start" style:justify-single-word="false" />
</style:style>
```

Im Abschnitt `<office:body>` befindet sich der vom Benutzer eingetippte Inhalt des Dokumentes. Dieser Bereich kann unter anderem aus Überschriften, normalen Text oder Listen bestehen. Es folgen einige Beispiele:

Normaler Text wird in Dokumenten als Paragraf dargestellt, dem eine bestimmte Formatierungsangabe mitgegeben wird. In diesem Fall ist es die Formatierungsart "*Standard*".

```
<text:p text:style-name="Standard">Der Benutzer kann seine Kennung anfordern.</text:p>
```

### 5.1.1.2 styles.xml

Das Wurzel-Element für die Formatierungen ist immer `<office:document-styles>`. Sollte es Abweichungen zu den automatischen Formatierungsangaben innerhalb von `<office:automatic-styles>` geben werden diese Formatierungen in `styles.xml` abgespeichert.

### 5.1.1.3 meta.xml

Das Wurzel-Element für die *Meta*-Daten ist immer `<office:document-meta>`. In dieser Datei werden allgemeine Information über das Dokument gespeichert. Das folgende Beispiel zeigt die Metadaten eines mit *Moliri* generierten ODT-Dokumentes.

```
<?xml version="1.0" encoding="utf-8"?>
<office:document-meta grddl:transformation="http://docs.oasis-
open.org/office/1.2/xslt/odf2rdf.xsl"
office:version="1.2" xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:grddl="http://www.w3.org/2003/g/data-view#"
xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0"
xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
xmlns:ooo="http://openoffice.org/2004/office" xmlns:xlink="http://www.w3.org/1999/xlink">
  <office:meta>
    <meta:initial-creator>Alexandros Kechagias</meta:initial-creator>
    <meta:creation-date>2011-06-12T17:33:40</meta:creation-date>
    <meta:generator>LibreOffice/3.3$Linux LibreOffice_project/330m19$Build-
202</meta:generator>
    <dc:date>2011-07-19T00:53:36</dc:date>
    <dc:creator>Alexandros Kechagias</dc:creator>
    <meta:editing-duration>PT8H29M44S</meta:editing-duration>
    <meta:editing-cycles>57</meta:editing-cycles>
    <meta:document-statistic meta:character-count="191" meta:image-count="0" meta:object-
count="0"
    meta:page-count="1" meta:paragraph-count="8" meta:table-count="0" meta:word-count="16" />
    <dc:title>Brettspiele 1.0</dc:title>
  </office:meta>
</office:document-meta>
```

<code>&lt;meta:initial-creator&gt;</code>	Dies ist der Name der Person die das Dokument angelegt hat.
<code>&lt;meta:creation-date&gt;</code>	Das Datum an dem Dokument erstellt wurde.
<code>&lt;meta:generator&gt;</code>	Der Generator des Dokumentes. Da <i>Moliris</i> Template in LibreOffice erstellt wurde, wird auch das hier angezeigt.
<code>&lt;dc:date&gt;</code>	Datum der letzten Änderung.
<code>&lt;dc:creator&gt;</code>	Name der Person die das Dokument zuletzt geändert hat.
<code>&lt;meta:editing-duration&gt;</code>	Dauer die im Dokument mit Änderung verbracht wurde.
<code>&lt;meta:editing-cycles&gt;</code>	Anzahl der Änderungen.
<code>&lt;meta:document-statistic&gt;</code>	Diverse statistische Informationen über das Dokument.

<dc:title> Der Titel des Dokumentes.

### 5.1.2 Perl und ODF

Zur Auswahl standen die Module **oolib-perl**<sup>27</sup> und **OpenOffice::OODoc**<sup>28</sup>. Obwohl das zuerst genannte Modul nicht auf CPAN zur Verfügung steht, sind dennoch beide für den Benutzer komfortabel über *Ubuntus Paketverwaltung*<sup>29</sup> installierbar.

Das Modul **oolib-perl** wurde aufgrund der verständlichen Schnittstelle und der unproblematischen Installation in Erwägung gezogen. Folgende Gründe führten jedoch dazu sich gegen dieses Modul zu entscheiden:

<i>ODF-Standard</i>	Das Modul unterstützt nicht den <i>ODF-Standard</i> , da der Export noch im mittlerweile nicht mehr aktuellen <i>OpenOffice.org XML</i> <sup>30</sup> -Format erfolgt. Wie das heutige <i>ODF-Format</i> war auch <i>OpenOffice.org XML</i> , wie der Name bereits vermuten lässt, ein <i>XML-basiertes Format</i> . Es wurde von <i>Sun Microsystems</i> für <i>OpenOffice.org 1.0</i> eingeführt und war der Vorreiter des heutigen <i>ODF-Standards</i> . Es wird zwar noch von allen Office-Anwendungen unterstützt, die auch den <i>ODF-Standard</i> unterstützen, ist jedoch nicht mehr das Standardformat von aktuellen Textverarbeitungsprogrammen. Der Vorteil von diesem Format wäre hingegen die Unterstützung von älteren Programmversionen wie z.B. <i>OpenOffice.org 1.0</i> oder <i>StarOffice 6.0</i> . Da aber diese Programmversionen heutzutage immer mehr an Bedeutung verlieren, wird davon ausgegangen, dass mittels <i>Moliri</i> generierte Pflichtenhefte nur mit Programmen geöffnet werden die den aktuellen Standard unterstützen. Desweiteren wird das ältere <i>OpenOffice.org XML-Format</i> nicht mehr weitergepflegt oder erweitert, sondern alle Entwicklungsarbeit in den neueren <i>ODF-Standard</i> investiert.
<i>Funktionsumfang</i>	Dokumente können nur minimal formatiert werden und erweiterte Funktionalität wie etwa das Einfügen von Bildern, Fuß- oder Kopfzeilen ist nicht möglich.
<i>Aktualität</i>	Die letzte Aktualisierung des Moduls, am 5. Februar 2007, liegt mittlerweile über vier Jahre zurück. Die Wahrscheinlichkeit, dass das Modul bald auf aktuellen Systemen nicht mehr uneingeschränkt verwendbar ist, ist nicht zu vernachlässigen. Obwohl am <i>OpenOffice.org XML Standard</i> vorrausichtlich keine Änderungen mehr stattfinden werden, unterliegt <i>Perl</i> und die von <i>oolib</i> benötigten Module einer ständigen Weiterentwicklung.

<sup>27</sup> *OpenOffice.org Utility Library* - <http://oolib.sourceforge.net/>

<sup>28</sup> Jean-Marie Gouarné - *OpenOffice::OODoc* - <http://search.cpan.org/dist/OpenOffice-OODoc/>

<sup>29</sup> *Advanced Packaging Tool (APT)* - <http://wiki.debian.org/Apt>

<sup>30</sup> [PDF] SUN - *OpenOffice.org XML Spezifikation* - [http://xml.openoffice.org/xml\\_specification.pdf](http://xml.openoffice.org/xml_specification.pdf)

Aus den oben genannten Gründen fiel die Wahl auf **OpenOffice::OODoc**, da es das einzige Modul zum Zeitpunkt der Machbarkeitsstudie war womit die Generierung von ODF-konformen Dokumenten ermöglicht wurde. Als Nachteile des Moduls seien seine sehr hohe Komplexität und die daraus folgende Einarbeitungszeit genannt. Alles was über das einfache Einfügen von Texten hinaus geht und Feineinstellungen erfordert ist nur mit einem hohen Zeitaufwand zu realisieren, da hierzu, wie auch in dieser Diplomarbeit, eine intensive Einarbeitung in die umfangreiche ODF-Spezifikation nötig ist. Außerdem unterstützt **OpenOffice::OODoc** nicht die aktuellste ODF-Spezifikation<sup>31</sup> (1.2) sondern nur die Version 1.0<sup>32 33</sup>. Das hat jedoch keine Auswirkung auf die Kompatibilität zu aktuellen Textverarbeitungsprogrammen.

**OpenOffice::OODoc** kann mit allen ODF-Formaten arbeiten. Es können Dokumente vollständig neu erstellt oder bearbeitet werden. Im Folgenden wird eine kurze Übersicht über die Module von **OpenOffice::OODoc** aufgezeigt.

<b>OpenOffice::OODoc::File</b>	Dieses Modul sollte selten Verwendung finden. Es ist zuständig für alle Operationen die den direkten Zugriff auf die XML-Dateien eines ODF-Dokumentes erfordern.
<b>OpenOffice::OODoc::XPath</b>	Wird nur benutzt wenn Objekte aus dem Dokument direkt als XML-Elemente behandelt werden müssen.
<b>OpenOffice::OODoc::Text</b>	Alle Operationen mit Objekten die etwas mit Texterstellung oder -manipulation zu tun haben, sich also auf den textuellen Inhalt des Dokumentes beziehen. Dies können unter anderem Tabellen, Paragraphen und Überschriften sein. Alle Operationen dieses Modules beschränken sich bis auf einige Ausnahmen auf die <i>content.xml</i> .
<b>OpenOffice::OODoc::Styles</b>	Alle Operationen mit Objekten die etwas mit Formatierungsangaben in Verbindung stehen, sich also auf die Darstellung von Inhalt konzentrieren. Alle Operationen dieses Modules beschränken sich bis auf ein paar Ausnahmen auf die <i>styles.xml</i> .
<b>OpenOffice::OODoc::Meta</b>	Verändert die Meta-Daten. Alle Operationen dieses Modules beschränken sich bis auf ein paar Ausnahmen auf die <i>meta.xml</i> .
<b>OpenOffice::OODoc::Image</b>	Bilddateien können mit diesem Modul zum ODF-Dokument hinzugefügt werden. Dabei werden größten-

<sup>31</sup>[PDF]OASIS - ODF-Spezifikation 1.2 - <http://docs.oasis-open.org/office/v1.2/cs01/OpenDocument-v1.2-cs01.pdf>

<sup>32</sup><http://search.cpan.org/~jmgdoc/OpenOffice-OODoc/OODoc/Intro.pod#Overview>

<sup>33</sup>[PDF]OASIS - ODF-Spezifikation 1.0 - <http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf>

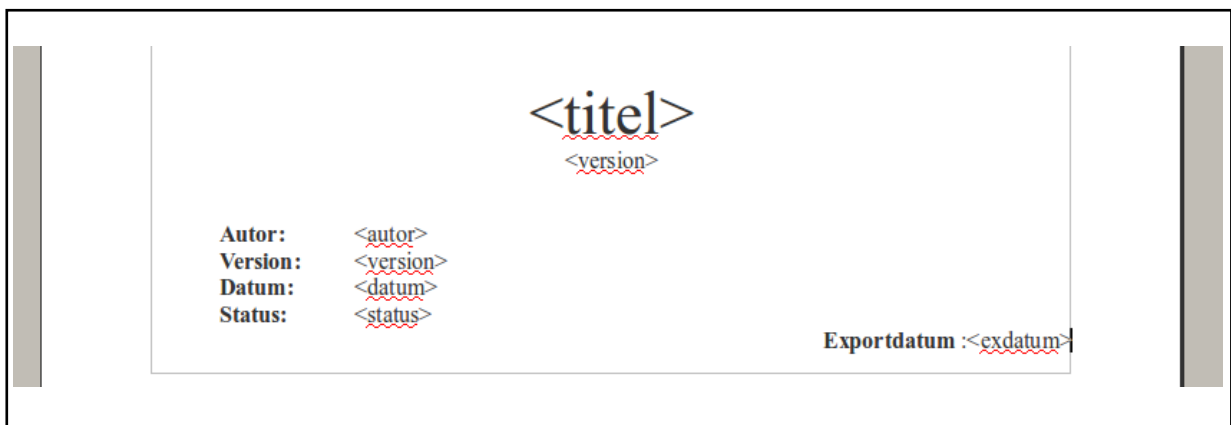
teils Einstellungen bezüglich der Größe und Position des Bildes innerhalb des Dokumentes angeben.

#### *OpenOffice::OODoc::Document*

Dieses Modul ist die am häufigsten genutzte Programmierschnittstelle. Es beinhaltet die Funktionalität der Module *XPath*, *Text*, *Styles* und *Image* die je nach bearbeiteten Objekt und aufgerufener Funktion Verwendung finden.

### 5.1.3 Implementierung

Die Implementierung wurde innerhalb der Klasse *MoliriODT.pm* umgesetzt. Zuerst wurde mit *LibreOffice* die Vorlage erstellt auf deren Basis das Pflichtenheft generiert wird.



**Abbildung 7 :** Vorlage des generierten Pflichtenheftes

Diese Vorlage beinhaltet Platzhalter für den Titel, Autor, Version, Datum und den Status des generierten Pflichtenheftes. Zusätzlich enthält die Vorlage das Exportdatum und eine Fußzeile. Die Platzhalter sind durch spitze Klammern eingeschlossen und werden beim Generieren des Objektes durch die jeweiligen Benutzerangaben ersetzt. Desweiteren wurden *Feldbefehle* für den Titel, Autor und die aktuelle Seitenzahl des Dokumentes eingefügt. *Feldbefehle* enthalten Information die automatisch aktualisiert werden. Das folgende Beispiel zeigt wie über Modul *OpenOffice::OODoc::Meta* die Daten manuell vor dem Beginn der Generierung zugewiesen werden.

```
my $meta = odfMeta( file => $ausgabe );

my $titel  = encode_utf8( $ref_pflcht('details'){ 'titel' } );
my $version = encode_utf8( $ref_pflcht('details'){ 'version' } );
my $autor  = encode_utf8( $ref_pflcht('details'){ 'autor' } );
# Metadaten ändern
$meta->initial_creator( $autor );
$meta->title( encode_utf8( $titel.' '.$version ) );
$meta->save( $ausgabe );
```



*odfMeta()* ist die Kurzform von *OpenOffice::OODoc::Meta->new()*. Mit den Funktionen *initial\_creator()* und *title()* wird der Zugriff auf die Metadaten bezüglich des Autors und der Dokumentbezeichnung ermöglicht. Diese ändern die XML-Elemente *<meta:initial-creator>* und *<dc:title>* der *meta.xml* in den gewünschten Inhalt. Anschließend greifen die Feldbefehle auf diese Elemente zu und zeigen deren Inhalt im resultierenden Dokument.

Der nächste Ausschnitt aus dem Quelltext soll veranschaulichen wie die Platzhalter mit der Funktion *selectElementsByContent()* durch die gewünschten Werte ersetzt werden.

```
# Exportdatum
my @loc = localtime;
my $jahr = $loc[5]+1900;
my $monat = $loc[4]+1;
my $tag = $loc[3];
my $exdatum = "$jahr-$monat-$tag";

$doc->selectElementByContent( '<titel>', $elemente[0] );
$doc->selectElementByContent( '<version>', $elemente[1] );
$doc->selectElementByContent( '<autor>', $elemente[2] );
$doc->selectElementByContent( '<datum>', $elemente[3] );
$doc->selectElementByContent( '<status>', $elemente[4] );
$doc->selectElementByContent( '<exdatum>', $exdatum );
```

Zunächst wird das aktuelle Datum aus der Perlfunktion *localtime* extrahiert und zur Anzeige aufbereitet wird. Die Funktion *selectElementByContent()* durchsucht das gesamte Dokument nach dem erstmaligen Vorkommen der oben genannten Platzhalter. So wird beispielsweise *<status>* oder *<exdatum>* ersetzt durch die Pflichtenheftdetails bzw. durch das Exportdatum.

Alle anderen Teile des Dokumentes werden vollständig vom *OpenOffice::OODoc* generiert. Dies geschieht über Paragraphen die dem Dokument über die Funktion *appendParagraph()* hinzugefügt werden. Diesen Paragraphen können mit Formatierungsangaben bezüglich ihres textuellen Inhalts und der Ausrichtung im Dokument versehen werden. Es folgt ein Beispiel eines solchen Paragraphen aus der Klasse *MoliriODT.pm* :

```
$doc->appendParagraph(
    text => "Ein bisschen Inhalt" ,
    style => 'einschub_inhalt',
);
```

In dem folgenden Beispiel wird dem Paragrafen mit dem Parameter "*style*" eine Formatierungsart übergeben und mit dem Parameter "*text*" der textuelle Inhalt. Die Formatierungsangabe "*einschub\_inhalt*" legt Einschubtiefe fest. Die Definition der besagten Formatierungsart lautet demnach:

```
$doc->createStyle(
    'einschub_inhalt',
    family    => 'paragraph',
    properties => { 'fo:margin-left' => '2cm' }
);
```

Hier wurde zuerst eine Bezeichnung für die Formatierungsart vergeben und anschließend die Art der Formatierung. Die Eigenschaften der Formatierung werden mit dem Parameter "*properties*" zugewiesen.

Ein weiteres Beispiel veranschaulicht, dass der Inhalt von Paragraphen auch komplexer werden kann. Dieser Fall tritt z.B. ein wenn Text innerhalb von Paragraphen nicht einheitlich formatiert ist, wie dies teilweise bei fetter Schrift der Fall ist.

```
my $para = $doc->appendParagraph(
    text => q{},
    style => 'einschub_typ',
);
# Kennung
$doc->extendText( $para, "Name: " , 'fett' );
$doc->extendText( $para, encode_utf8( "Löwenherz" ), 'default' );
```

In diesem Beispiel muss zuerst der Paragraf mit dem gewünschten Einschub angelegt werden. Anschließend wird sein textueller Inhalt mit der Funktion *extendText()* um die Zeichenkette "*Name:*" erweitert und als letzter Parameter die Formatierungsart "*fett*" übergeben. Um zu verhindern, dass jede weitere an diesen Paragrafen angehängte Zeichenkette die gleiche Formatierung hat, muss die nachfolgende Zeichenkette wieder auf die Standardformatierung des Dokumentes umgestellt werden. Dies wird mit der Formatierungsart "*default*" erreicht.

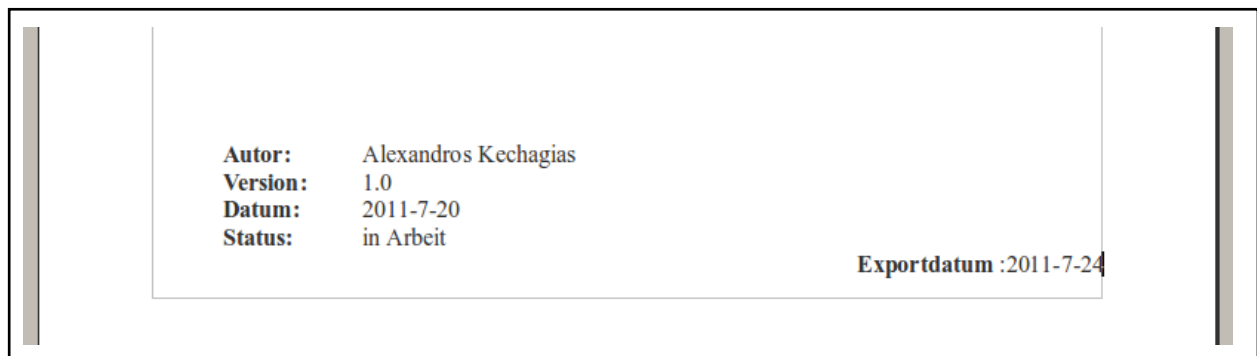
Das Resultat eines mit *Moliri* generierten Pflichtenheftes wird in den folgenden Abbildungen exemplarisch präsentiert<sup>34</sup>.

#### 5.1.4 Generiertes Beispielpflichtenheft



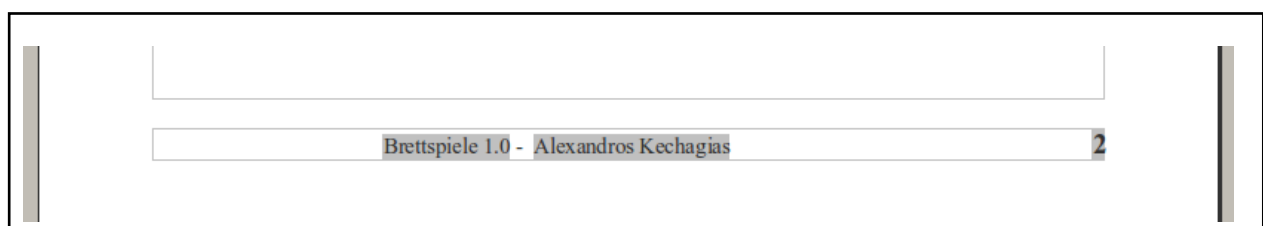
**Abbildung 8** : Oberer Bereich des Titelblattes

Der Titel des Pflichtenheftes wird in einer Schriftgröße von 32 Pixeln an erster Stelle des Dokumentes eingefügt. Darunter folgt die Versionsnummer in 12 Pixeln.



**Abbildung 9** : Unterer Bereich des Titelblattes

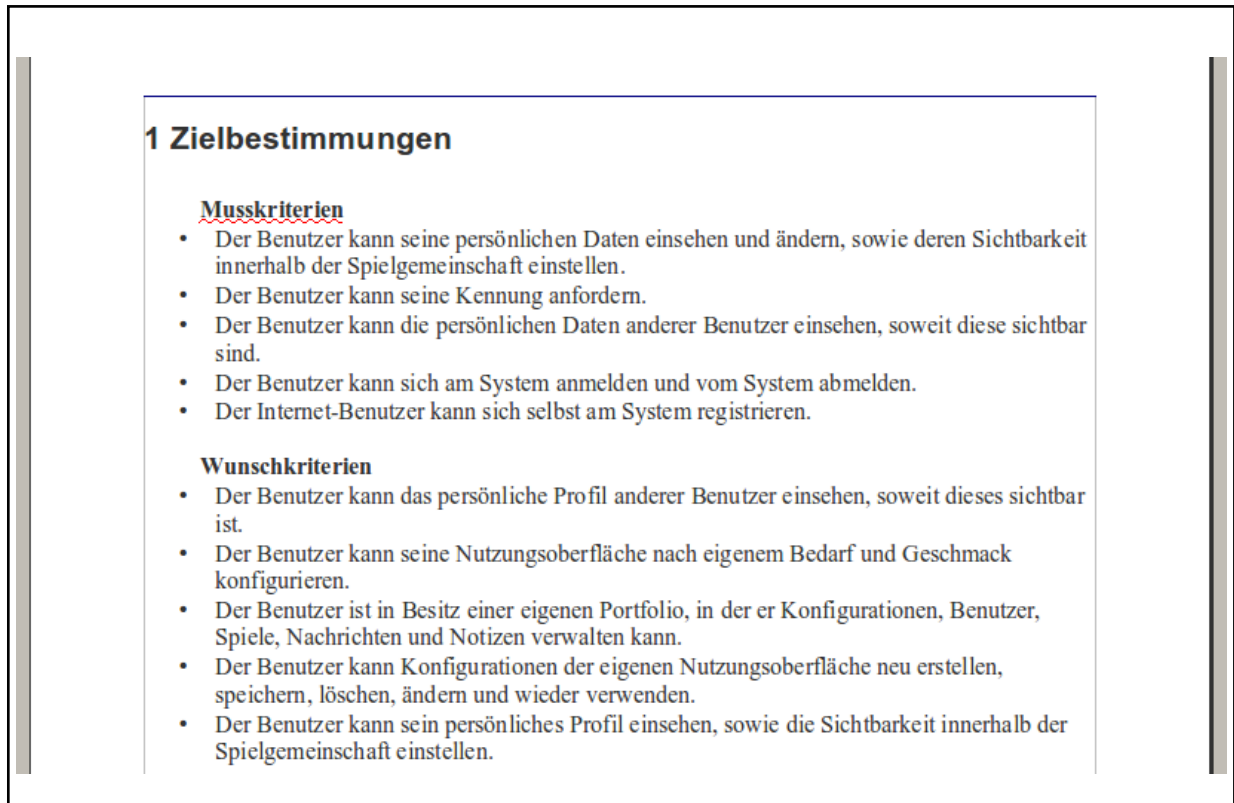
Der untere Bereich des Titelblattes besteht aus den Pflichtenheftdetails wie Autor, Version, Datum, Status und desweiteren das Exportdatum in rechtsbündiger Ausrichtung.



**Abbildung 10** : Fußleiste

<sup>34</sup>[PDF] Stefan K. Baur - *Beispielpflichtenheft* - <http://www.stefan-baur.de/downloads/Pflichtenheft.pdf>

Die Fußzeile besteht aus dem Dokumententitel, der Versionsnummer und dem Namen des Autors. Der farblich unterlegte Feldbereich ist beim Drucken oder beim Export in ein anderes Format nicht mehr sichtbar. Die farbliche Markierung eines Feldbefehles wird von *LibreOffice* und *OpenOffice.org* nur verwendet um eine Abhebung der Felbereiche vom normalen Text zu erreichen.



**Abbildung 11 : Zielbestimmungen**

Die Zielbestimmungen sind aufgeteilt in der Reihenfolge Musskriterien, Wunschkriterien und Abgrenzungskriterien. Die einzelnen Kriterien werden als Aufzählung mit einem führenden Punkt als Aufzählungssymbol generiert.



Abbildung 12 : Ergänzungen

Dies ist der Bereich der Ergänzungen, in dem Text und auch Bilder eingefügt werden können.

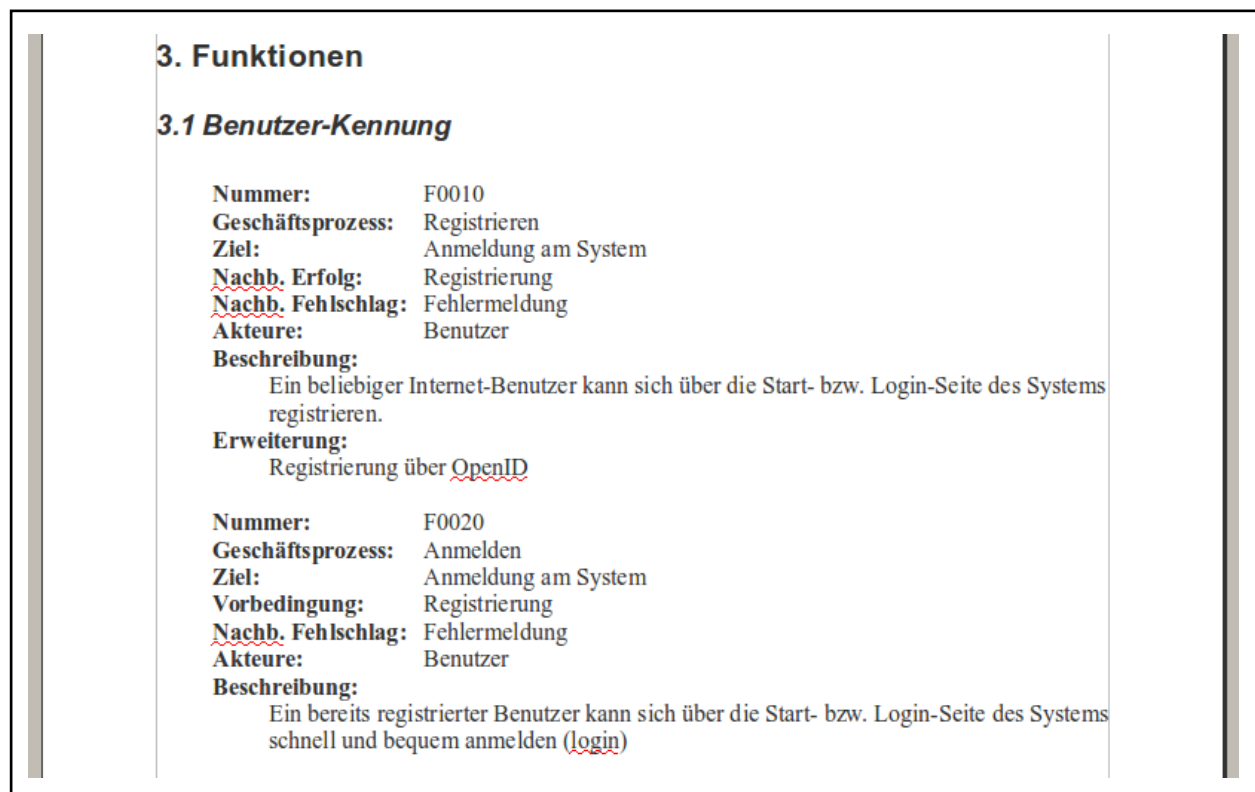


Abbildung 13 : Funktionen

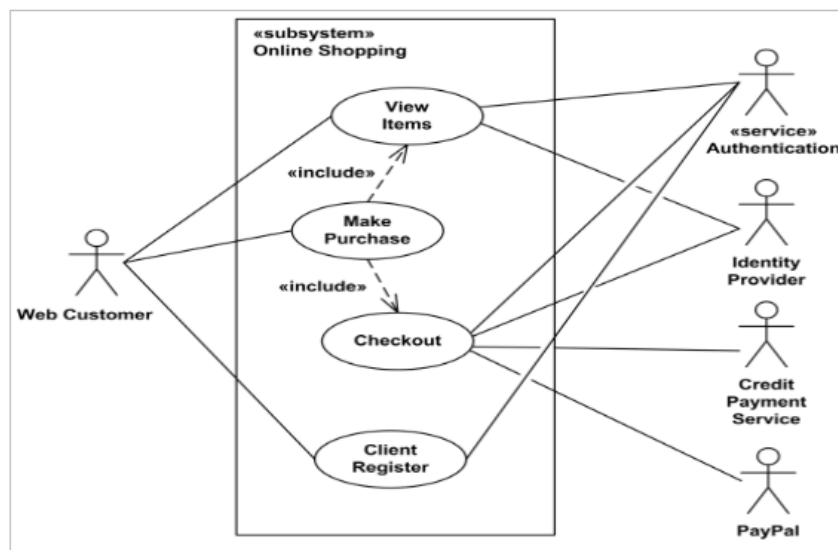
In *Abbildung 13* ist der Bereich der Funktionen zu sehen. Dieser besitzt nur eine Kapitelbezeichnung wenn diese in der Oberfläche angegeben ist. Es werden nur die Einträge ins Pflichtenheft exportiert die auch in der Benutzeroberfläche angegeben wurden. Dies ist an dem Eintrag "Nachb. Erfolg" zu sehen, der nicht in Funktion "F0020" angezeigt wird. Die Bereiche der Daten, Leistungen, GUI, Teilprodukte, Testfälle und Glossar haben einen ähnlichen Aufbau.

<b>6. Qualität</b>				
<b>Produktqualität</b>	<b>Sehr Gut</b>	<b>Gut</b>	<b>Normal</b>	<b>Nicht Relevant</b>
<b>Funktionalität</b>				
Angemessenheit	X			
Richtigkeit	X			
Interoperabilität				X
Ordnungsmässigkeit				X
<b>Zuverlässigkeit</b>				
Reife	X			
Fehlertoleranz			X	
<b>Änderbarkeit</b>				
Analysierbarkeit		X		
Stabilität		X		

Abbildung 14 : Qualitätsanforderungen

Für jede Produkthanforderung wird eine eigene Tabellenzeile generiert in der die dazugehörige Relevanz in der jeweiligen Spalte durch ein "X" gekennzeichnet wird.

### 3. Produktübersicht



Online-Shopping

Der Kunde hat die Möglichkeit sich zu registrieren oder sich zuerst sich die Produktpalette anzuschauen. Die Registrierung muss durch die Service Authentication genehmigt werden.

**Abbildung 15** : Produktübersicht mit eingefügter Bilddatei

Hier wird der Bereich Produktübersicht mit eingefügter Bilddatei gezeigt. Überschreiten die Abmessungen des eingefügten Bildes<sup>35</sup> die Seitenbreite und -höhe, so werden diese automatisch an die Standardseitengröße angepasst. An dieser Stelle sei erwähnt, dass die richtige Bildgröße nicht von *OpenOffice::OODoc* automatisch ermittelt wird sondern mithilfe des Modules *Image::Size* errechnet werden muss.

<sup>35</sup><http://www.uml-diagrams.org/examples/use-case-example-online-shopping.png>

## 5.2 Textformat

Der Export ins Textformat wird in der Klasse *MoliriTXT.pm* realisiert. Zum Export in die Textdatei wird der Text zuerst in *UTF-8* kodiert und anschließend über ein Perl *File-Handle* in die gewünschte Datei exportiert. Die Darstellung des generierten Pflichtenheftes sollte sich möglichst an die Vorgaben eines *ODT*-Dokumentes anlehnen. Da nicht alle Texteditoren über einen automatischen Zeilenumbruch verfügen wurden mithilfe des Moduls *Text::Wrap* Zeilenumbrüche erzeugt. Der Zeilenumbruch erfolgt nach 80 Zeichen, somit ist die Textdatei auf Bildschirmen mit einer Auflösung von 1024 Pixeln in Ihrer vollen Breite sichtbar. Da die Darstellung von Bilddateien nicht in Textdateien möglich ist, werden die Bilddateien und ihre Beschreibungen nicht exportiert.

Das Resultat eines solchen Pflichtenheftes wird in den folgenden Abbildungen demonstriert.

```

Titel:      Brettspiele
Version:    2.0
Autor:      Alexandros Kechagias
Datum:      2011-7-20
Status:     in Arbeit
|
Exportdatum: 2011-7-28

```

Abbildung 16 : Textdatei - Kopf

Wie in *Abbildung 16* dargestellt, besteht der Anfang der Textdatei aus Titel, Version, Autor, Datum, Status und Exportdatum.

```

#####
#      13. Ergänzungen      #
#####

Das System verfügt über ein austauschbares Sprachmodul, indem alle Texte
zum Dialog mit den Benutzern vorliegen und jeweils über eine
Identifikationsnummer angesprochen werden.

Um das System um eine Sprache erweitern zu können, muss das Sprachmodul
mit der neuen Sprache analog zum bereits existierenden Sprachmodul erstellt
werden.

```

Abbildung 17 : Textdatei - Eränzungen

Wie in *Abbildung 17* zu sehen, wurde zur Hervorhebung der Überschriften ein Rahmen aus Rauten erstellt.



```
#####
#          4. Funktionen          #
#####

-- 4.1 Benutzer-Kennung --

Nummer:                                F0010

Geschäftsprozess:  Registrieren
Ziel:              Anmeldung am System
Nachb. Erfolg:    Registrierung
Nachb. Fehlschlag: Fehlermeldung
Akteure:          Benutzer
Beschreibung:
    Ein beliebiger Internet-Benutzer kann sich über die Start- bzw.
    Login-Seite des Systems registrieren.
Erweiterung:
    Registrierung über OpenID

Nummer:                                F0020

Geschäftsprozess:  Anmelden
Ziel:              Anmeldung am System
Vorbedingung:     Registrierung
Nachb. Fehlschlag: Fehlermeldung
Akteure:          Benutzer
Beschreibung:
    Ein bereits registrierter Benutzer kann sich über die Start- bzw.
    Login-Seite des Systems schnell und bequem anmelden (login)
```

Abbildung 18 : Textdatei – Funktionen

```
#####
#          10. Technische Umgebung          #
#####

    Das Produkt ist weitgehend unabhängig vom Betriebssystem, sofern folgende
    Produktumgebung vorhanden ist.

--Produktumgebung--
    Clientseite:
    - www-Browser der neuesten Generation: Internet Explorer 6 und Mozilla 1.3
      (keine textbasierten Browser)
    Serverseite:
    - PHP (mind. Version 4.0.5)
    - MySQL-Datenbank
    - SMTP-Server (eMail-fähig)

--Software--
    Clientseite:
    - Internetfähiger Rechner
    - Rechner, der die Ansprüche der o.g. Server-Software erfüllt
    - Ausreichend Rechen- und Festplattenkapazität
```

Abbildung 19 : Textdatei - Technische Umgebung

Wie in *Abbildung 19* zu sehen ist, wurde zur Hervorhebung der Unterüberschriften ein Rahmen aus Rauten erstellt.

#####					
# 7. Qualität #					
#####					
Produktqualität	Sehr Gut	Gut	Normal	Nicht	Relevant
Funktionalität					
Angemessenheit	X				
Richtigkeit	X				
Interoperabilität					X
Ordnungsmässigkeit					X
Zuverlässigkeit					
Reife	X				
Fehlertoleranz			X		
Änderbarkeit					
Analysierbarkeit		X			
Stabilität		X			
Übertragbarkeit					
Anpassbarkeit			X		
Konformität					X
Austauschbarkeit		X			

Abbildung 20 : Textdatei - Qualität

#####	
# 1. Zielbestimmungen #	
#####	
--Musskriterien--	
<ul style="list-style-type: none"> <li>* Der Benutzer kann seine persönlichen Daten einsehen und ändern, sowie deren Sichtbarkeit innerhalb der Spielgemeinschaft einstellen.</li> <li>* Der Benutzer kann seine Kennung anfordern.</li> <li>* Der Benutzer kann die persönlichen Daten anderer Benutzer einsehen, soweit diese sichtbar sind.</li> <li>* Der Benutzer kann sich am System anmelden und vom System abmelden.</li> <li>* Der Internet-Benutzer kann sich selbst am System registrieren.</li> </ul>	
--Wunschkriterien--	
<ul style="list-style-type: none"> <li>* Der Benutzer kann das persönliche Profil anderer Benutzer einsehen, soweit dieses sichtbar ist.</li> <li>* Der Benutzer kann seine Nutzungsoberfläche nach eigenem Bedarf und Geschmack konfigurieren.</li> <li>* Der Benutzer ist in Besitz einer eigenen Portfolio, in der er Konfigurationen, Benutzer, Spiele, Nachrichten und Notizen verwalten kann.</li> <li>* Der Benutzer kann Konfigurationen der eigenen Nutzungsoberfläche neu erstellen, speichern, löschen, ändern und wieder verwenden.</li> <li>* Der Benutzer kann sein persönliches Profil einsehen, sowie die Sichtbarkeit innerhalb der Spielgemeinschaft einstellen.</li> </ul>	

Abbildung 21 : Textdatei – Zielbestimmungen

## 7. Benutzeroberfläche

Die Benutzeroberfläche wurde mit dem *Widget-Toolkit Tk*<sup>36</sup> erstellt. Neben Tk gibt es noch andere *Widget-Toolkits* wie z.B. *wxPerl*<sup>37</sup> oder *Gtk2-Perl*<sup>38</sup>. Da aber ein großer Teil der Benutzeroberfläche bereits mit Tk geschrieben und das komplette Neuschreiben sehr viel Einarbeitungszeit kosten würde, fiel die Entscheidung auf Tk.

### 7.1 Tk

Ursprünglich wurde Tk von John K. Ousterhout für die Skriptsprache Tcl entwickelt. Die erste Version von Tk erschien 1991 und war aufgrund seiner Einfachheit ein großer Erfolg. In den folgenden Jahren wurden tausende von Tcl/Tk Programme geschrieben und Tks Schnittstelle wurde für Programmiersprachen wie Eiffel, Modula-3, Prolog, Python und Scheme zugänglich gemacht. 1994 machte Malcom Beattie mit seinem TkPerl möglich, Tk in Perlprogrammen zu benutzen. Die anschließende Zusammenarbeit mit Nick Ing-Simmons führte dazu, dass auf beidseitigem Einverständnis die Arbeit von Nick Ing-Simmons fortgeführt wurde<sup>39</sup>. Heute ist Perl/Tk neben wxPerl und Gtk2-Perl einer der am meisten verwendeten *Widget-Toolkits* für Perl. Nicht zuletzt das Buch *Mastering Perl/Tk* ( siehe [LiWa02] ) sorgt dafür, daß Tk noch heute von vielen Programmierern verwendet wird.

Tks ereignisgesteuerte Architektur<sup>40</sup> bietet dem Benutzer, wie andere *Widget-Toolkits* auch, die Möglichkeit mit dem Programm über die grafische Ebene zu kommunizieren, indem es ihm über Ereignisse den Programmablauf steuern lässt. Es können beliebige Interaktionen mit der Benutzeroberfläche sein, wie das Drücken eines Steuerelementes z.B. eines Buttons. Sie sagen dem Programm was der Benutzer als nächstes machen will. Ereignisse müssen vom Programmierer einer Funktionalität zugewiesen werden die die Aufgaben des Benutzers löst. Diese ereignisgesteuerte Architektur wird in Tk durch *bindings*<sup>41</sup> realisiert. Die *bindings* verknüpfen ein Ereignis mit einer Rückruffunktion, in Perl ist eine solche Funktion eine Subroutine.

Dabei müssen diese Steuerelemente zur Bewältigung dieser Aufgaben beitragen, indem ihre Anordnung und Bezeichnung möglichst intuitiv vom Benutzer auf seine Aufgabenstellung bezogen werden kann.

In Tk erfolgt der Aufbau der Programmoberfläche über *Widgets*, die in einer hierarchischen Struktur angeordnet sind. An der höchsten Stelle dieser Struktur steht das *MainWindow*. Das *MainWindow* dient als Container dem alle anderen *Widgets* untergeordnet werden. Der Container trägt in Tk die Bezeichnung *Eltern-Widget* und die *Widgets* die er beinhaltet werden als Kinder-Widgets bezeichnet. Nach diesem Prinzip der Verschachtelung ist die gesamte Ober-

<sup>36</sup>Slaven Rezić - Tk - <http://search.cpan.org/dist/Tk/pod/gencmd>

<sup>37</sup>Mark Dootson - wxPerl - <http://search.cpan.org/dist/Wx/>

<sup>38</sup>Torsten Schönfeld - gtk2-perl - <http://gtk2-perl.sourceforge.net/>

<sup>39</sup>vgl. [LiWa02] Chapter 1.2.2 The Coming of Tcl/Tk

<sup>40</sup>vgl. [LiWa02] Chapter 15: Anatomy of the MainLoop, 1. Absatz

<sup>41</sup>Slaven Rezić - Tk::bind - <http://search.cpan.org/~srezic/Tk-804.029/pod/bind.pod>

fläche aufgebaut. Jedes Widget bis auf *MainWindow* hat auch ein *Eltern-Widget* von dem es abstammt. Die Positionierung und Sichtbarkeit der *Kinder-Widgets* innerhalb Ihrer *Eltern-Widgets*, wird durch den *Geometry-Manager* übernommen.

## 7.2 Die Pflichtenheftverwaltung

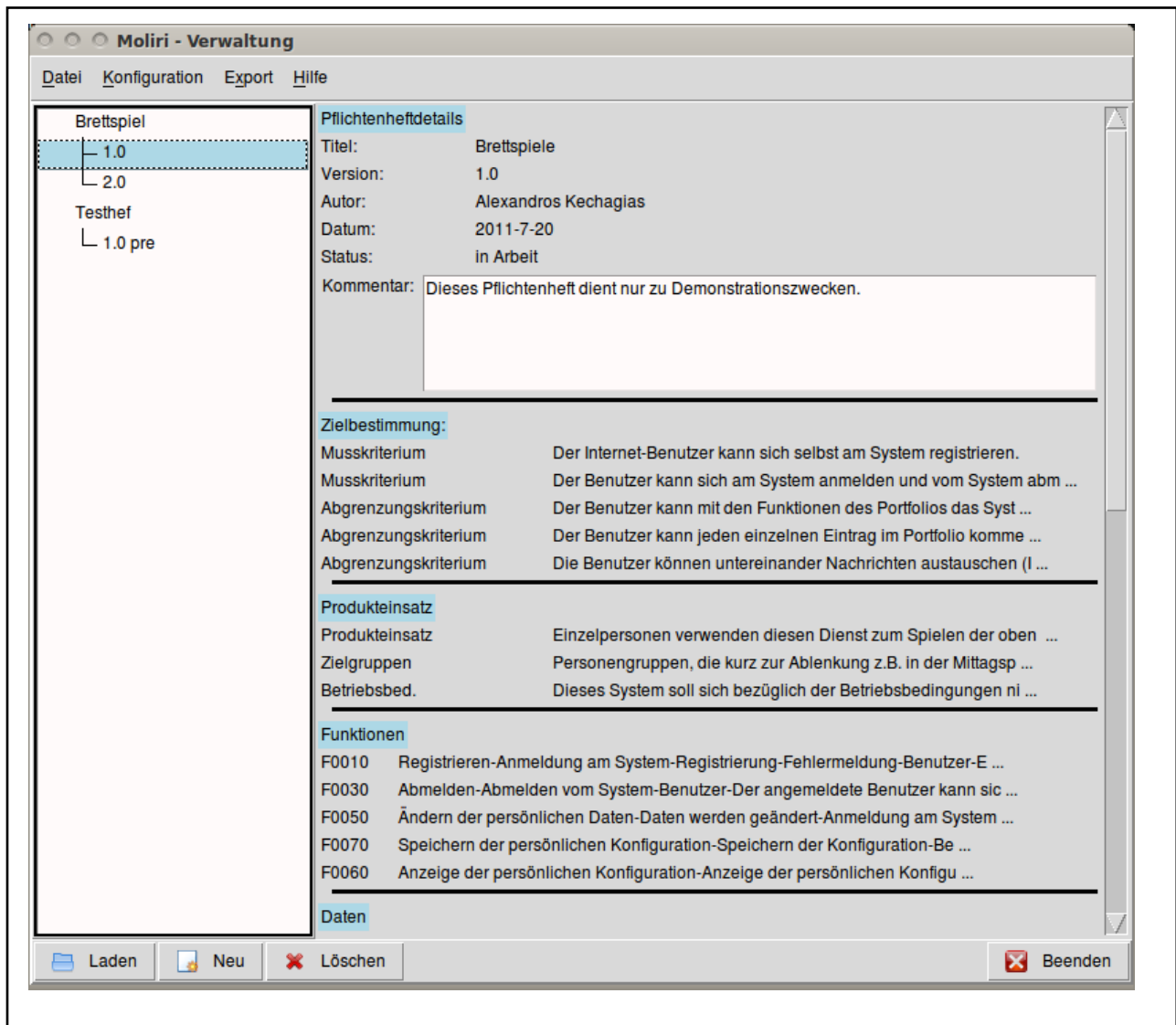
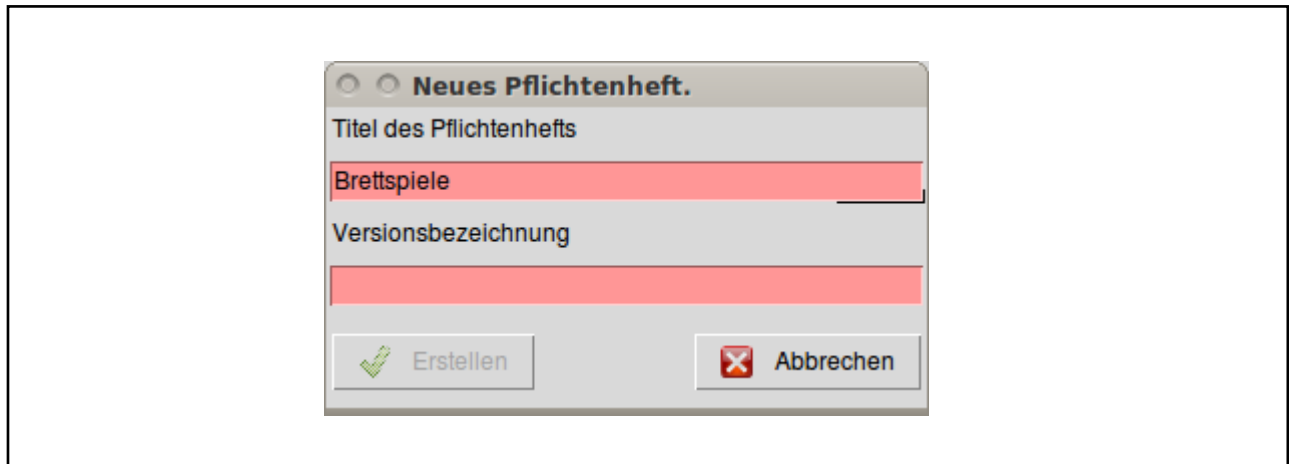


Abbildung 22 : Moliri-Verwaltung

Die Pflichtenheftverwaltung beinhaltet eine Übersicht aller Pflichtenhefte in Form einer Baumstruktur. Durch das Klicken mit der Maus auf die einzelnen Einträge der Baumstruktur, wird eine Vorschau des markierten Pflichtenheftes auf der Benutzeroberfläche erstellt. Die angezeigten Einträge von Datensätzen unbestimmter Anzahl wie z.B. Funktionen oder Ziele wurden aus Gründen der Übersichtlichkeit auf fünf Einträge begrenzt. Außerdem tragen die blau hinterlegten Kapitelüberschriften sowie die schwarzen Balken zur Übersichtlichkeit bei.



**Abbildung 23** : Moliri - Neues Pflichtenheft bei Fehler in der Texteingabe

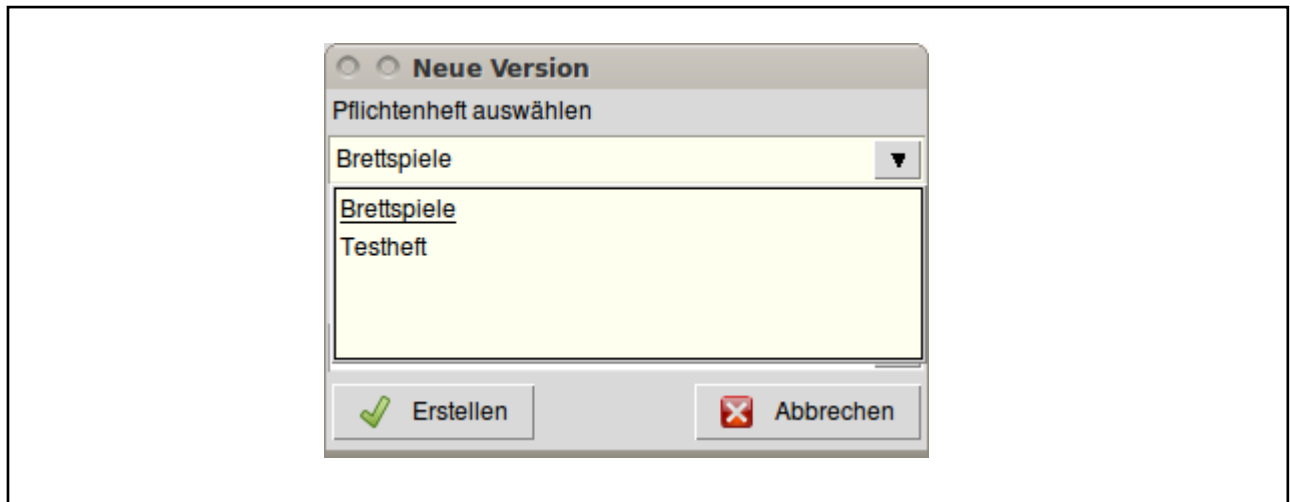
Da ein Projektordner eine Vielzahl von Pflichtenheften enthalten kann und somit der Benutzer nur schwer die Übersicht über bereits vorhandene Pflichtenhefte behält, wurden fehlertolerante Dialoge implementiert. In diesen Dialogen werden fehlerhafte oder unvollständige Eingaben, dem Benutzer sofort durch farbliches Hinterlegen des betroffenen Eingabefeldes gemeldet. In diesem Fall ist im oberen Eingabefeld der Pflichtenhefttitel "Brettspiele" bereits vergeben und im unteren Eingabefeld keine Versionsbezeichnung angegeben. Somit wird auch der Erstellen-Button deaktiviert. Dieses Verfahren sorgt für eine *Fehlertoleranz*<sup>42</sup> die es dem Benutzer bereits während der Eingabe ermöglicht seine Fehler zu korrigieren, wodurch ein fließender Arbeitsablauf gewährleistet wird.



**Abbildung 24** : Moliri - Neues Pflichtenheft

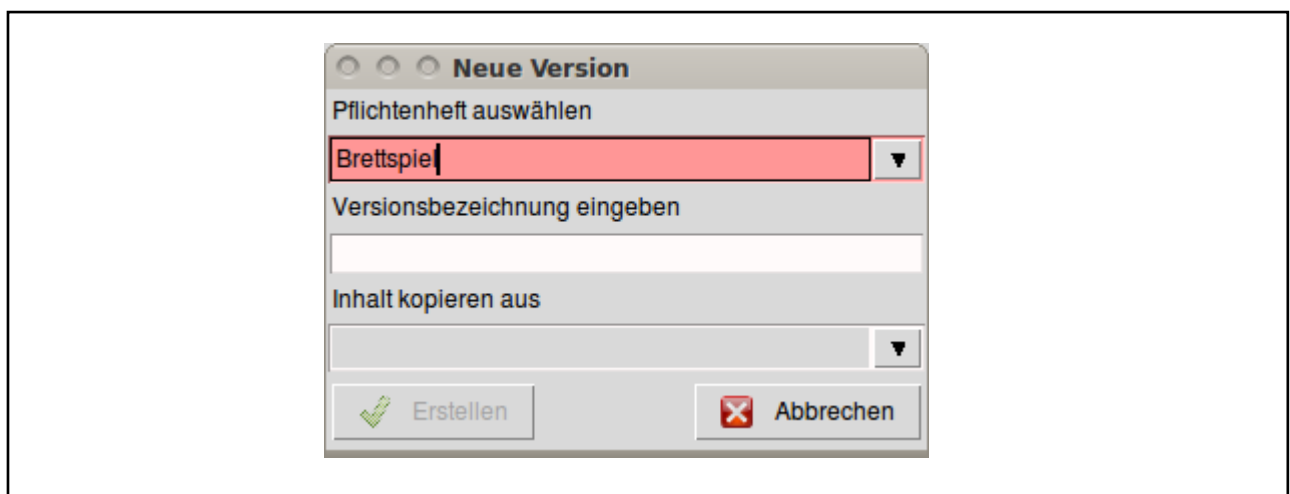
Bei Erfüllung aller Bedingungen sind die benötigten Eingabefelder mit einem weißen Hintergrund hinterlegt und der Benutzer kann sein gewünschtes Pflichtenheft erstellen in dem er auf den nun auch aktivierten Erstellen-Button klickt.

<sup>42</sup>[Balz04] Kapitel 9 Ergonomie & Barrierefreiheit, *Abschnitt Fehlertoleranz*



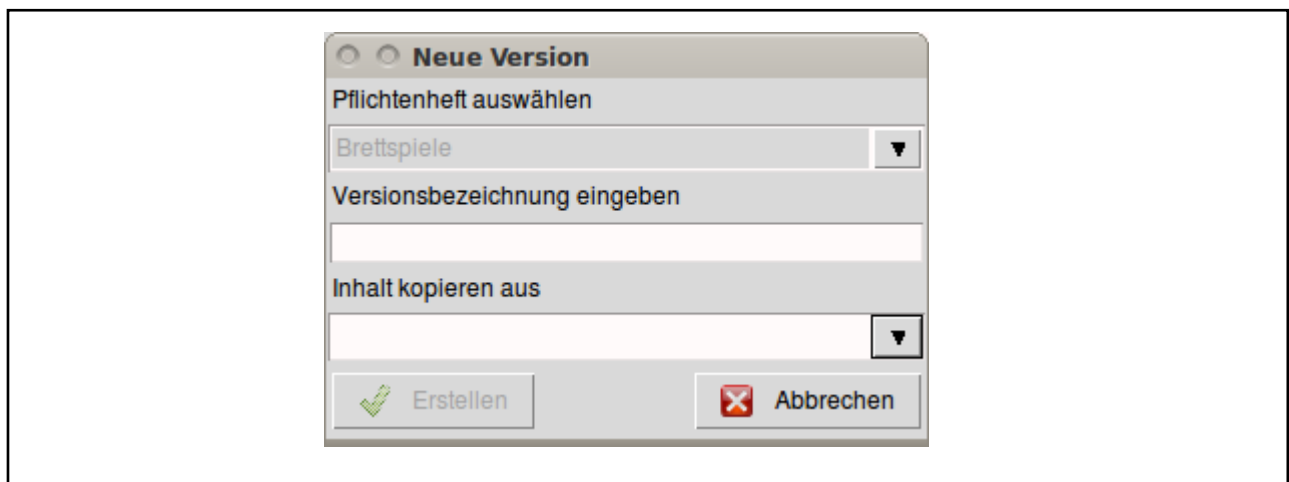
**Abbildung 25** : Moliri - Neue Version bei Auswahl des Pflichtenheftes

Sollte ein neues Pflichtenheft erstellt werden, wird dem Benutzer eine Auswahl bereits vorhandener Pflichtenhefte in Form eines Dropdown-Menüs präsentiert. Hier ist es dem Benutzer möglich aus einer Übersichtsliste aller vorhandenen Pflichtenhefte, das gewünschte auszuwählen.



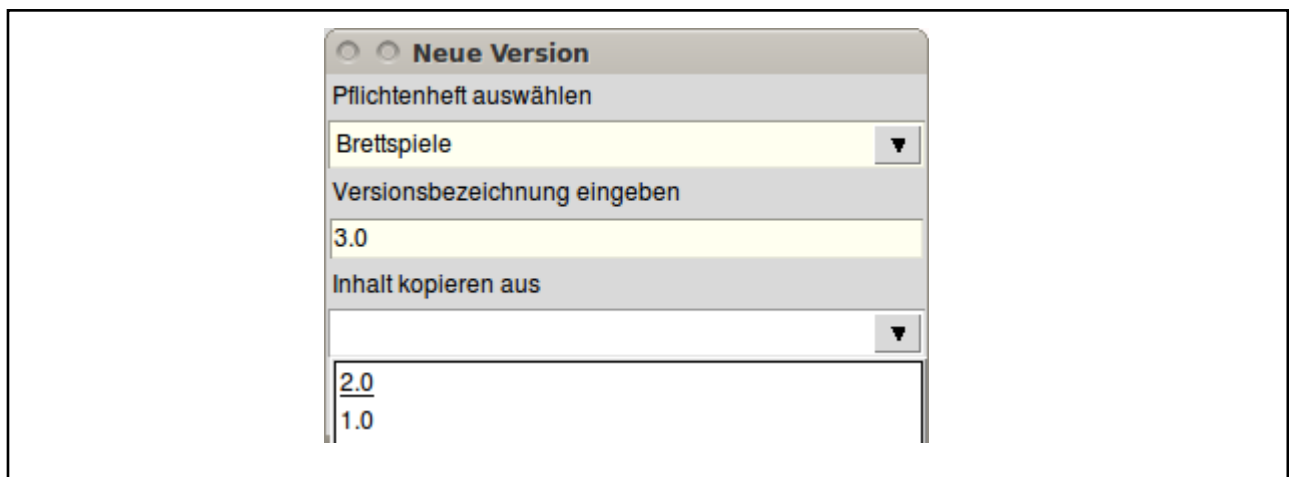
**Abbildung 26** : Moliri - Neue Version bei fehlerhafter Eingabe

Der Benutzer kann zusätzlich, wie im Dialog zur Erstellung eines neuen Pflichtenheftes, selber die Eingabe vornehmen. Wie in *Abbildung 26* veranschaulicht wird, werden auch hier fehlertolerante Dialoge verwendet. Da in diesem Fall eine neue Version zu einem Pflichtenheft angelegt wird, muss das Pflichtenheft bereits vorhanden sein. Auch hier wird bei fehlerhafter Eingabe der Erstellen-Button deaktiviert. Da bei fehlerhafter Eingabe des Pflichtenheftes kein Inhalt zum Kopieren ausgewählt werden kann, wird auch dieses Feld automatisch deaktiviert.



**Abbildung 27 :** Moliri - Neue Version Pflichtenheft bereits ausgewählt

Sollte der Benutzer bereits ein Pflichtenheft innerhalb der Baumstruktur markiert haben, wird zur weiteren Vereinfachung, das Pflichtenheft im Dialog ausgewählt. Der Benutzer muss nur noch die Versionsbezeichnung und den zu kopierenden Inhalt auswählen.



**Abbildung 28 :** Moliri - Neue Version Inhalt auswählen

Nach erfolgreich ausgewähltem Pflichtenheft kann nun optional der Inhalt aus einem anderen Pflichtenheft in das neue kopiert werden. Das Dropdown-Menü von "Inhalt kopieren aus" wird hierbei je nach Auswahl des Pflichtenheftes dynamisch erzeugt und enthält nur die Versionen des ausgewählten Pflichtenheftes. Wenn alle Einträge bis auf den Optionalen korrekt angegeben wurden, wird auch der Erstellen-Button wieder aktiviert.



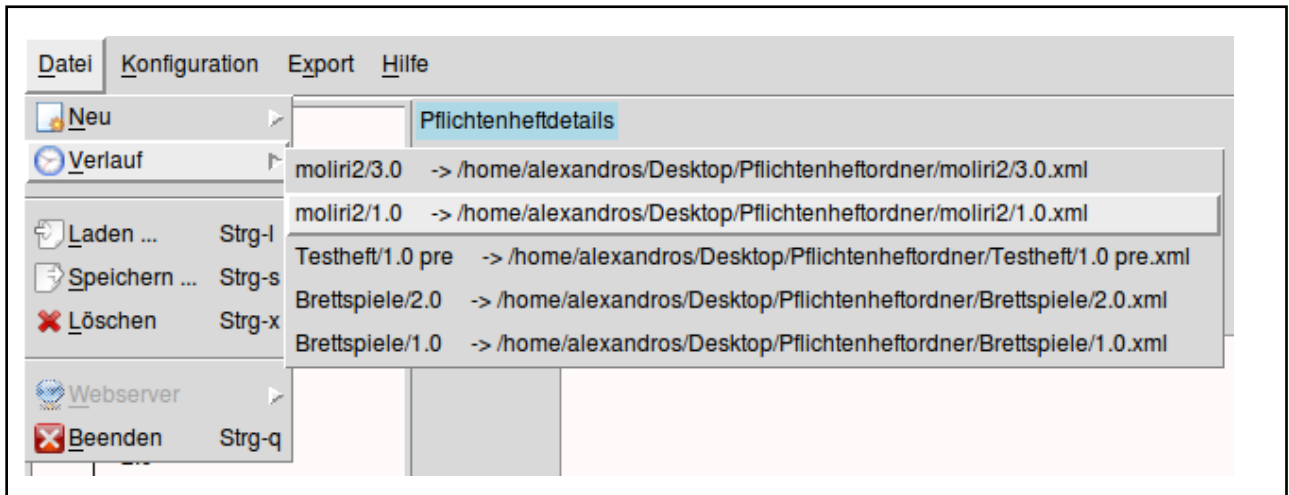


Abbildung 29 : Moliri - Verlauf

Alle aufgerufenen Pflichtenhefte werden in der Datei *history* gespeichert, diese befindet sich innerhalb des Projektordners. Der Verlauf speichert maximal 10 Pflichtenhefte und zeigt nur Pflichtenhefte an, die existieren und beschreibbar sind. Dies wird nach jedem Löschen oder Neuanlegen eines Pflichtenheftes, sowie beim Programmstart überprüft.

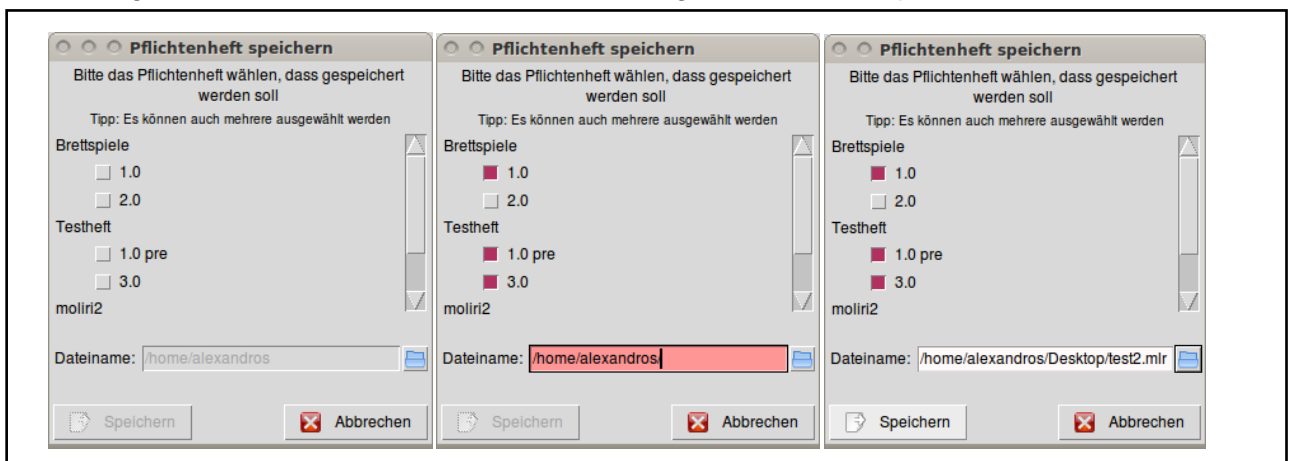


Abbildung 30 : Moliri - Pflichtenheft speichern

Das abspeichern von Pflichtenheften erfolgt über Moliris Speichern-Dialog. Dazu werden die Pflichtenhefte aus der Baumansicht der Pflichtenheftverwaltung ausgelesen und als Auswahlfelder angezeigt. Aus diesen Auswahlfeldern kann sich der Benutzer seine gewünschten Pflichtenhefte aussuchen. Erst nach der Auswahl eines Pflichtenheftes wird die Pfadeingabe aktiviert. Da das Heimverzeichnis nicht auf jedem System in `"/home/<Benutzername>"` abgespeichert sein muss, wird zur Ermittlung des Heimverzeichnisses das Modul *File::HomeDir*<sup>43</sup> verwendet. Erst wenn alle Eingaben gültig sind, wird der Speichern-Button aktiviert.

<sup>43</sup>Adam Kennedy - *File::Homedir* - <http://search.cpan.org/~adamk/File-HomeDir/>

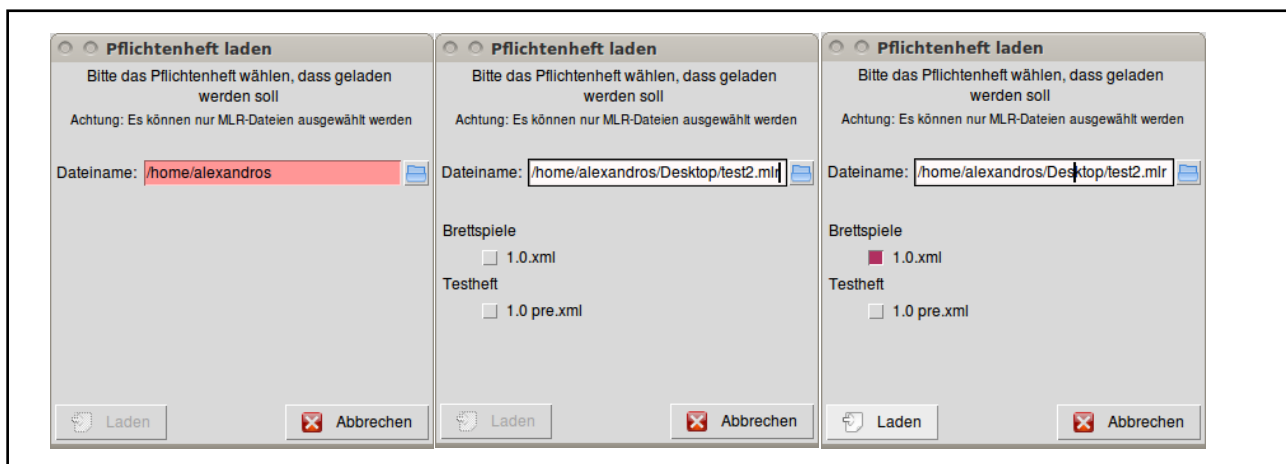


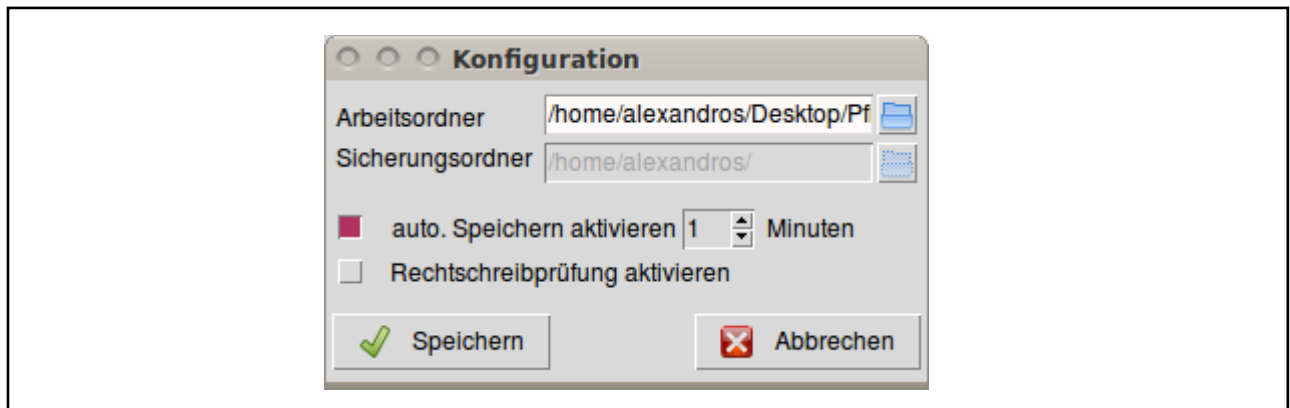
Abbildung 31 : Moliri Pflichtenheft laden

Das Laden eines Pflichtenheftes funktioniert ähnlich wie das Speichern. Mit dem Unterschied, dass die Auswahlfelder dynamisch erzeugt werden, wenn eine *mlr*-Datei im Pfad angegeben wurde. Sollte vom Benutzer am angegebenen Pfad nicht die ODT-Erweiterung angegeben werden, wird diese vom Programm automatisch angehängt.



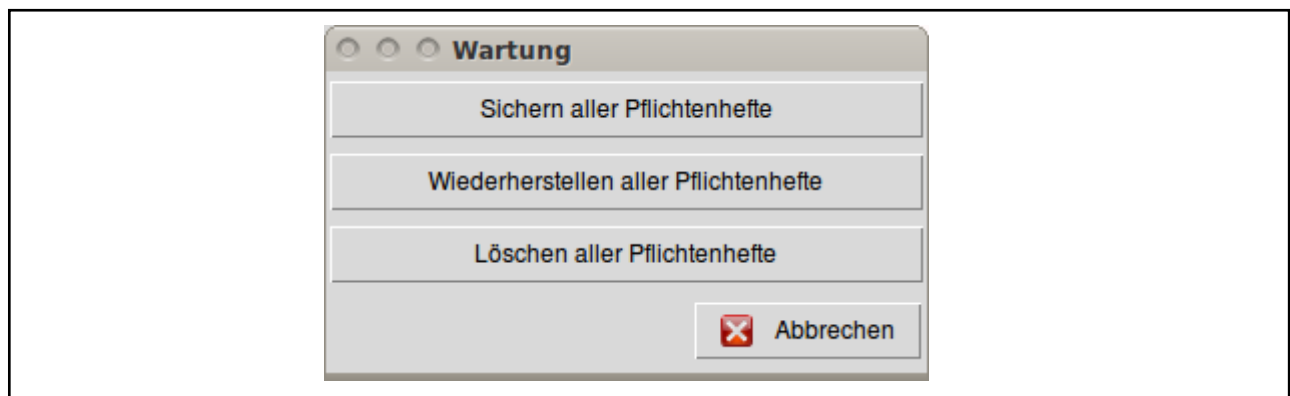
Abbildung 32 : Moliri - Pflichtenhefte exportieren

Die Export-Dialoge verhalten sich ähnlich wie die Dialoge zum Laden und Speichern. Der Unterschied liegt hierbei in der Auswahl der Pflichtenhefte. Da nur ein Pflichtenheft zum Export angegeben werden darf, wurde das Auswahlfeld durch ein Optionsfeld ersetzt. Das erlaubt nur das Auswählen von einzelnen Einträgen.



**Abbildung 33 :** Moliri - Konfiguration

Wie in allen Dialogen von Moliri werden auch im Konfigurationsdialog alle Eingaben auf ihre Gültigkeit überprüft. Der Pfad des Arbeitsordners wird zusätzlich beim Start der Anwendung überprüft. Sollte dieser nicht vorhanden sein, wird der zu diesem Dialog weitergeleitet, um einen Arbeitsordner auszuwählen.



**Abbildung 34 :** Moliri - Wartung

Die Wartungsfunktionen zur Erhöhung der Benutzerfreundlichkeit implementiert. Sie liefern keine neuen Funktionalitäten sondern sie sollen Aufgaben vereinfachen die alle Pflichtenhefte betreffen um eine möglichst vollständige Funktionalität aus der Benutzeroberfläche heraus zu gewährleisten.

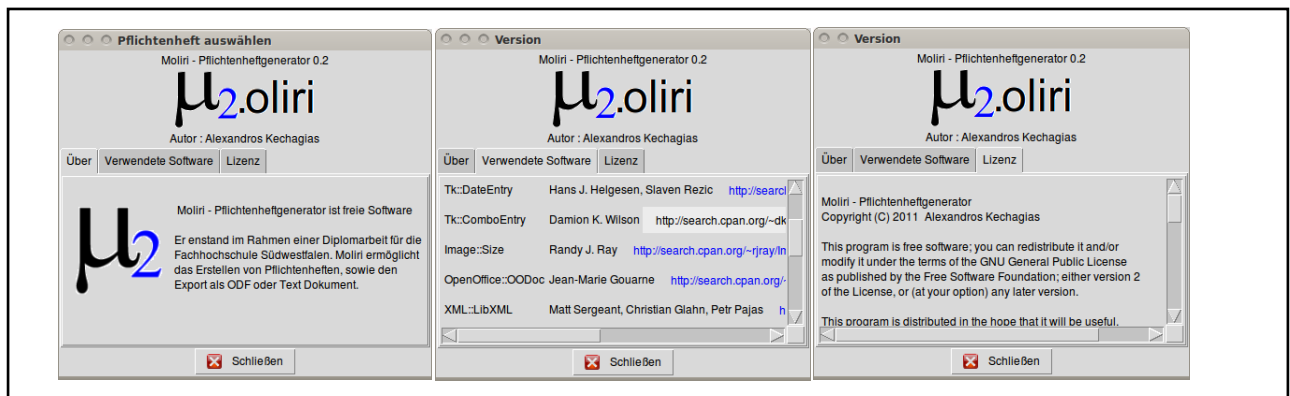


Abbildung 35 : Moliri - Version

Im Versionsdialog wird der Pflichtenheftgenerator prägnant beschrieben, die verwendete Software inklusive Weblinks und die Lizenz angegeben. Der Abschnitt der verwendeten Software, bietet Links zu Webadressen, diese wurden mit Buttons realisiert, die den Konsolenbefehl *gnome-open*<sup>44</sup> mit der Webadresse starten. Der Befehl öffnet den übergebenen Pfad mit dem Standardprogramm der *GNOME-Oberfläche* für Dateien dieses Typs.

<sup>44</sup>Manpage - *gnome-open* - <http://manpages.ubuntu.com/manpages/lucid/man1/gnome-open.1.html>

### 7.3 Der Pflichtenhefteditor

**Morili - Pflichtenheftgenerator**

**Datei**

Titel: Brettspiele  
 Autor: Alexandros Kechagias  
 Datum: 2011-7-20  
 Version: 1.0  
 Status: ☒ in Arbeit ☐ akzeptiert ☐ freigegeben

Kommentar: Dieses Pflichtenheft dient nur zu Demonstrationszwecken.

**1. Zielbestimmung** | Neues Ziel | Bearbeiten | Löschen

Typ	Beschreibung
Wunschkriterium	Der Benutzer kann das persönliche Profil anderer Benutzer einsehen, soweit dies
Abgrenzungskriterium	Der Benutzer kann jeden einzelnen Eintrag im Portfolio kommentieren und bewert
Musskriterium	Der Benutzer kann seine persönlichen Daten einsehen und ändern, sowie deren
Wunschkriterium	Der Benutzer kann seine Nutzungsoberfläche nach eigenem Bedarf und Geschma
Wunschkriterium	Der Benutzer ist in Besitz einer eigenen Portfolio, in der er Konfigurationen, Benut
Musskriterium	Der Benutzer kann seine Kennung anfordern.
Abgrenzungskriterium	Die Benutzer können untereinander Nachrichten austauschen (Instant-Messaging
Wunschkriterium	Der Benutzer kann Konfigurationen der eigenen Nutzungsoberfläche neu erstellen
Musskriterium	Der Benutzer kann die persönlichen Daten anderer Benutzer einsehen, soweit die
Musskriterium	Der Benutzer kann sich am System anmelden und vom System abmelden.
Musskriterium	Der Internet-Benutzer kann sich selbst am System registrieren.
Abgrenzungskriterium	Der Benutzer kann mit den Funktionen des Portfolios das System durchsuchen, di
Wunschkriterium	Der Benutzer kann sein persönliches Profil einsehen, sowie die Sichtbarkeit inner

12. Teilprodukte  
 13. Ergänzungen  
 14. Testfälle  
 15. Glossar

Projektansicht | vorheriger Schritt | nächster Schritt

Abbildung 36 : Moliri - Pflichtenhefteditor

Die Editor-Oberfläche besteht im oberen Abschnitt aus den Pflichtenheftdetails. Der linke Abschnitt hat zweierlei Funktionalität. Zum einen trägt er durch seine farbliche Hervorhebung des aktuell befindlichen Schrittes zur Übersichtlichkeit bei und dient zum anderen auch zur Navigation. Die Arbeitsfläche befindet sich im mittleren Bereich, und kann entweder durch den Navigationsbereich oder die Buttons *nächster Schritt* und *vorheriger Schritt* gewechselt werden. Jeder der 15 Schritte beinhaltet eine eigene Ansicht die auf Abruf angezeigt wird. Durch diese Funktionsweise konnte eine relativ komplexe Benutzeroberfläche auf möglichst kompakte Weise dargestellt werden, ohne dass der Benutzer umständlich über mehrere Dialogebenen hinweg arbeiten muss.

### 7.3.1. Kompakter Aufbau

Im Gegensatz zu anderen *Widget*-Bibliotheken wie *Nokias Qt* mit seinem *QtStackwidget*<sup>45</sup> oder *Java Swing* mit seinem *CardLayout*<sup>46</sup> bietet *Tk* diese Funktionalität in keinem seiner *Widgets* an. Um eine ähnliche Funktionalität in *Tk* zu erreichen wurde mithilfe von *Tks* gegebenen Mechanismen eine Lösung erarbeitet.

Nachdem in *Tk* ein *Widget* erstellt wurde, wird dies noch nicht automatisch auf der Oberfläche angezeigt. Die Platzierung und Sichtbarkeit der *Widgets* muss explizit mit einem *Geometry-Manager* erfolgen. Dazu wird abhängig vom *Geometry-Manager* eine der folgenden Funktionen aufrufen.

<i>pack</i>	Die Anordnung der <i>Widgets</i> ist ähnlich dem eines Schiebepuzzles. Die <i>Widgets</i> können sich nicht überschneiden und bleiben innerhalb Ihres Containers.
<i>grid</i>	Die <i>Widgets</i> werden wie in einer Tabelle in Reihen und Zeilen angeordnet.
<i>place</i>	Die Anordnung erfolgt durch eine absolute Positionierung der <i>Widgets</i> innerhalb ihres Containers.
<i>form</i>	Die <i>Widgets</i> werden an andere bestehende <i>Widgets</i> anhand bestimmter Regeln angehängen

Sollten mehrere *Geometry-Manager* gleichzeitig verwendet werden kann es bei überschneidenden Positionierungsangaben von *Widgets* zu einer *race condition*<sup>47</sup> führen. Hierbei konkurrieren zwei *Geometry-Manager* um die angegebene Position. Da *Tk* in so einem Fall keine Priorisierung vorsieht, versuchen beide *Geometry-Manager* ihre *Widgets* an dieselbe Stelle zu positionieren und ändern gegenseitig wiederholt die Positionierung. Dies kann zu einer sehr hohen CPU-Last bis hin zur Blockierung des Programms führen.

Das folgende gekürzte Beispiel der Funktion *show\_frame* des Pflichtenhefteditors soll die Implementierung einer solchen Funktionalität veranschaulichen.

```
sub show_frame {
    my ($frm) = @_;

    # Durchlaufe alle Frames
    foreach ( 0 .. 14 ) {
        if ( $_ != $frm ) {      # Wenn nicht hervorzuhebender Frame
            $frame_stack[$_]->packForget(); # aus dem geometry-manager löschen
        }
        else {
            $frame_stack[$_]
                ->pack( -side => 'left', -fill => 'both', -expand => '1' );
        }
    }
    return;
} # ----- end of subroutine show_frame -----
```

<sup>45</sup>Nokia - *QtStackwidget* - <http://doc.qt.nokia.com/latest/qstackedwidget.html>

<sup>46</sup>Oracle - *CardLayout* - <http://download.oracle.com/javase/7/docs/api/java/awt/CardLayout.html>

<sup>47</sup>[LiWa02] Chapter 2. *Geometry Managment*, 4. Absatz

Nachdem die Oberfläche mithilfe eines *Geometry-Managers* aufgebaut wurde ist es nun möglich mittels der Funktion *packForget* einzelne Widgets aus dem *Geometry-Manager* zu entfernen, also für den Benutzer nicht sichtbar zu machen. Sollte ein Eltern-Widget aus dem *Geometry-Manager* entfernt werden, wirkt sich das aufgrund der hierarchischen Struktur von *Tk* auf alle seine *Kinder-Widgets* aus. Dieses Verhalten wurde für die Diplomarbeit genutzt, indem die Oberflächen der einzelnen Schritte innerhalb solcher *Container-Widgets*, auch *Frames* genannt, platziert wurden. Nun wird das gewünschte *Frame* über den *Geometry-Manager* auf der Oberfläche angezeigt indem alle Anderen mittels *packForget* versteckt werden.

### 7.3.2. Fehlertoleranz und Bedienfreundlichkeit von Dialogen

Im Pflichtenhefteditor wie auch schon in der Pflichtenheftverwaltung wurde versucht dem Benutzer bereits während der Eingabe auf Fehler hinzuweisen. Zusätzlich dazu wurde in den Dialogen der Editoroberfläche eine automatische Vervollständigung und Vorschau auf bereits vorhandene Einträge implementiert. Bei Pflichtenheftabschnitten wie den Funktionen oder Testfällen können diese Einträge je nach Umfang des Pflichtenheftes eine kaum noch überschaubare Anzahl annehmen. Für die Eingabefelder solcher Dialoge, deren Inhalt als eindeutiger Bezeichner eines Datensatzes dient, sind diese Funktionen unerlässlich.

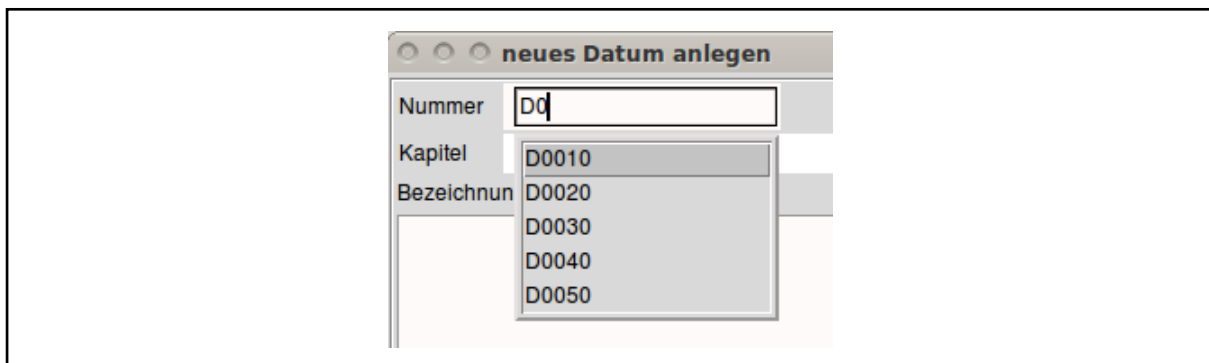


Abbildung 37 : Moliri-Pflichtenhefteditor - Vorschau

Wie in *Abbildung 37* zu sehen wird dem Benutzer bereits während der Eingabe eine Liste vorhandener Nummern gezeigt. Diese Funktion ist kein Bestandteil von *Tk*, sie wurde über das Fremdmodul *Tk::MatchEntry*<sup>48</sup> eingebunden.

<sup>48</sup> Wolfgang - *Tk::MatchEntry* - *Hommel* - <http://search.cpan.org/~whom/Tk-MatchEntry-0.4>

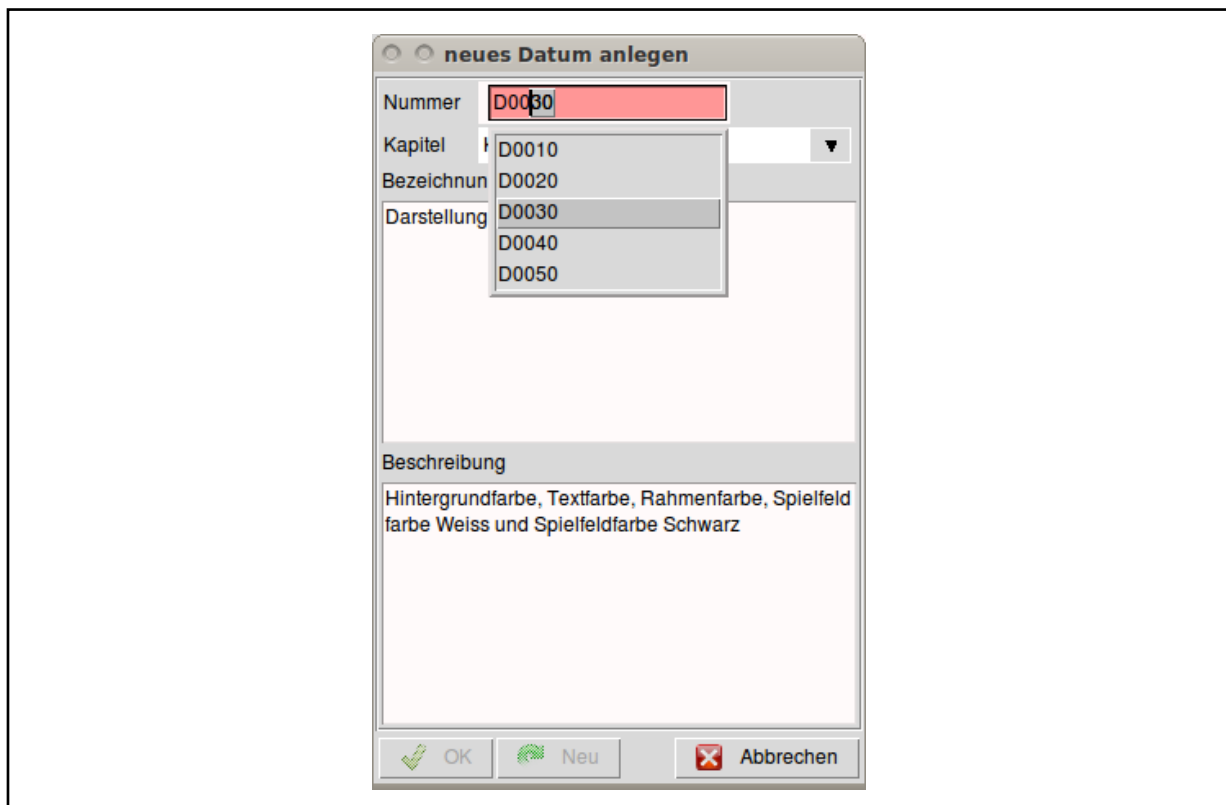


Abbildung 38 : Moliri-Pflichtenhefteditor - neues Datum anlegen - Dialog

Sollte die gewünschte Nummer bereits vergeben sein, wird das Eingabefeld farblich hervorgehoben und die Buttons zum Bestätigen der Eingabe deaktiviert. Außerdem wird der unter dieser Nummer bestehende Eintrag zur Ansicht in den Dialog geladen, so kann der Benutzer die bereits vorhandenen Einträge durchsuchen. Diese Funktionalität wurde außerdem noch für die Dialoge der Daten vom Typ *Funktionen*, *Daten*, *Leistungen*, *GUI*, *Testfälle* und *Glossar* implementiert. In den Dialogen zum Bearbeiten von Einträgen erfolgt die farbliche Hervorhebung nur wenn ein Eintrag nicht vorhanden ist.

Zur Umsetzung einer solchen Funktionalität musste mit *bindings* gearbeitet werden. Die *bindings* die herfür erforderlich sind werden in den nächsten Quelltextabschnitt vorgestellt.

```
$entry_nummer->bind(
    '<KeyRelease>',
    [
        \&dat_bind_nummer_type,
        {
            nummer           => \$nummer,
            text_bezeic       => \$text_bezeic,
            text_beschr       => \$text_beschr,
            kapitel           => \$kapitel,
            button_ok         => \$button_ok,
            button_neu        => \$button_neu,
            button_uebernehmen => \$button_uebernehmen,
        }
    ]
);
```

Zuerst muss das *binding* definiert werden. Hierfür wird das *binding* einem *Widget* zugewiesen, in diesem Fall dem Eingabefeld von *Nummer*.



Anschließend muss das Ereignis angegeben werden, bei dem das *binding* in Kraft tritt. Im diesem Beispiel wäre das beim loslassen einer gedrückten Taste. Weitere mögliche Ereignisse wären:

ButtonPress, Button	Expose	Map
ButtonRelease	FocusIn	Motion
Circulate	FocusOut	Property
Colormap	Gravity	Reparent
Configure	KeyPress, Key	Unmap
Destroy	KeyRelease	Visibility
Enter	Leave	Activate
Deactivate		

Im Anschluss wird dem binding die Rückruffunktion *dat\_bind\_nummer\_type* mit den nötigen Parametern übergeben. Die Parameter sind alle Eingabefelder und Buttons die von diesem *binding* verändert werden müssen.

```
sub dat_bind_nummer_type {
    my ( $w, $parameter_ref ) = @_;
    my $num          = ${$parameter_ref}{ 'nummer' };
    my $text_bezeic   = ${$parameter_ref}{ 'text_bezeic' };
    my $text_beschr   = ${$parameter_ref}{ 'text_beschr' };
    my $kapitel       = ${$parameter_ref}{ 'kapitel' };
    my $button_ok      = ${$parameter_ref}{ 'button_ok' };
    my $button_neu     = ${$parameter_ref}{ 'button_neu' };
    my $button_uebernehmen = ${$parameter_ref}{ 'button_uebernehmen' };

    my $nummer = ${$num};

    if ( $nummer ) {
        if ( $daten{$nummer} ) {

            # wenn Eintrag im Hash dann auf jeden Fall Einträge laden
            ${$text_bezeic}->Contents( $daten{$nummer}[0] );
            ${$text_beschr}->Contents( $daten{$nummer}[1] );
            $kapitel = suche_id( $nummer, \%daten_kapitel );
            ${$button_uebernehmen}->configure( -state => 'normal' );

            # Wenn OK-Button gemapped dann befindet man sich im
            # "Neu anlegen"-Modus. Die Buttons und Eingaben
            # deaktivieren um zu verdeutlichen, dass die Zahl bereits
            # vergeben ist.
            if ( ${$button_ok}->ismapped() ) {
                $w->configure( -background => '#ff9696' );
                ${$button_ok}->configure( -state => 'disabled' );
                ${$button_neu}->configure( -state => 'disabled' );
            }
        }
        else {

            #Ansonsten alle Felder leeren und
            #Ok/Neu Buttons auswählbar machen
            ${$text_bezeic}->Contents($EMPTY);
            ${$text_beschr}->Contents($EMPTY);
            $kapitel = $EMPTY;
            $w->configure( -background => 'snow1' );
            ${$button_ok}->configure( -state => 'normal' );
            ${$button_neu}->configure( -state => 'normal' );

            #Übernehmen-Button deaktivieren, da im Bearbeitungs-Modus
            #nur vorhandene Einträge bearbeitet werden. Rein Theoretisch
            #könnte man auch mit dem Bearbeiten-Button Einträge anlegen,
            #jedoch würde das nur unnötig für Verwirrung sorgen.
            ${$button_uebernehmen}->configure( -state => 'disabled' );
        }
    }
}
```

```
}  
else {  
  
    #Wenn keine Nummer angegeben, Alle Buttons deaktivieren.  
    ${button_ok}->configure( -state => 'disabled' );  
    ${button_neu}->configure( -state => 'disabled' );  
    ${button_uebernehmen}->configure( -state => 'disabled' );  
}  
  
return;  
}
```

Wie an diesem Beispiel zu sehen ist, wird nach jedem Tastenanschlag überprüft ob ein Eintrag bereits vorhanden ist. Ist der Eintrag vorhanden werden die Eingabefelder des Dialoges mit den Daten des vorhandenen Eintrages gefüllt. Abhängig davon ob der Dialog zum Bearbeiten oder Neuanlegen eines Eintrages dient, werden entsprechenden Buttons aktiviert oder deaktiviert beziehungsweise der Hintergrund des Eingabefeldes rot oder weiß gefärbt.

## 8. Ausblick

Der **Export in das LaTeX<sup>49</sup>-Format** würde nicht nur sehr hochwertige Dokumente sondern eine Vielzahl weiterer Konvertierungsmöglichkeiten in andere Formate eröffnen. Ein guter Ansatz dafür wäre das Modul *Template::Toolkit*<sup>50</sup>. Es ermöglicht die Generierung von Dokumenten aus speziellen Vorlagen heraus. Dabei handelt es sich um funktionale Vorlagen, die weit mehr als nur statische Vorlagen sind. Nach dem Einbinden der Vorlage in das *Template::Toolkit* ist es möglich der Vorlage die Daten in Form von *Hashes* und *Felder* zu übergeben. Zum Einstieg existiert eine gute Anleitung von Richard Hardwick vom *PraTeX-Journal* von 2010<sup>51</sup>. Die LaTeX-Vorlagen für die Pflichtenhefte können entweder selbst erstellt oder von diversen Webadressen bezogen werden<sup>52</sup>. Mithilfe dieser Vorlagen ist das Erstellen einer LaTeX-Pflichtenheftvorlage für das *Template::Toolkit* bereits mit Grundwissen von LaTeX möglich. Ähnlich wie Perls CPAN bietet LaTeX auch über CTAN<sup>53</sup> eine Ansammlung von LaTeX-bezogenen Downloads wie Skripte oder Vorlagen.

Obwohl man Dokumente im ODF-Format mittels der Textverarbeitungssoftware in das PDF-Format, setzt dies jedoch die Installation eines Textverarbeitungsprogramms oder die Benutzung eines Konverters vom Benutzer voraus. Der **Direktexport ins PDF-Format** wäre sicherlich eine sinnvolle Bereicherung von *Moliris* Funktionalität. Dies könnte über einen Konverter wie *odt2pdf*<sup>54</sup> aus dem Hauptprogramm heraus geschehen ohne dass der Benutzer selber eingreifen müsste.

Eine weitere sinnvolle Funktionalität ist die **Rechtschreibprüfung** innerhalb der Textboxen und Eingabefelder. Dies kann entweder während der Eingabe geschehen oder auf Kommando des Benutzers. Die zuletzt genannte Alternative wäre unproblematisch zu implementieren über die Überprüfung mit einem Kommando, dass das momentan aktive Feld oder wahlweise auch das komplette Dokument auf Rechtschreibfehler prüft.

Aufgrund eines Fehlers im Modul *OpenOffice::OODoc* führt das Abspeichern von Bildern innerhalb von ODT-Dateien zum Absturz von Textverarbeitungsprogrammen wie *OpenOffice.org* oder *LibreOffice*. Das einbinden von Bildern innerhalb des Dokumentes ist im Moment nur über den absoluten Pfad zur Bilddatei möglich. Das hat den Nachteil, dass generierte Dokumente die Bilder enthalten, nicht einfach auf andere Systeme transportiert werden können.

Das automatische Speichern von Dokumenten ist zurzeit noch instabil. Das liegt vor allem daran, dass Tk keine Threadsicherheit<sup>55</sup> bietet. Das automatische Speichern während der Arbeit im Pflichtenhefteditor wurde mit Perls *threads* realisiert. *Threads* sind Funktionen die

<sup>49</sup>The LaTeX3 project - <http://www.latex-project.org/latex3.html>

<sup>50</sup>Andy Wardley - *Template::Toolkit* - <http://search.cpan.org/~abw/Template-Toolkit/>

<sup>51</sup>[PDF]Richard Hardwick - *The PraTeX Journal*, 2011, No. 1 - <http://www.tug.org/pracjourn/2010-1/hardwick/hardwick.pdf>

<sup>52</sup>Stefan Baur - *TeX-Pflichtenheftvorlage* - <http://www.stefan-baur.de/cs.se.pflichtenheft.beispiel.html>

<sup>53</sup>Jim Hefferon - CTAN - <http://ctan.org/starter.html>

<sup>54</sup>*odt2pdf* - <http://sourceforge.net/projects/odt2pdf/>

<sup>55</sup>tcl and threads - <http://wiki.tcl.tk/1339>

Parallel zum eigentlichen Hauptprogramm ausgeführt werden. In *Moliri* kann dies unter Umständen zu Programmabstürzen führen. Die Benutzung der Klasse *Tk::After*<sup>56</sup> wäre in diesem Fall vermutlich die beste Lösung. Diese beinhalten die Funktion *repeat*, die von jedem Widget in *Tk* aufgerufen werden kann. Diese erlaubt es Befehle nach einer angegebenen Zeit auszuführen. Der Funktionsaufruf würde wie folgt aussehen:

```
$mw->repeat($time, &speichern);
```

Wobei *\$time* durch die Zeit in Millisekunden ersetzt wird und *speichern* die aufzurufende Funktion darstellt. Eine andere Möglichkeit wäre alle Tk-Zugehörigen Funktionen aus der *thread*-Funktion *speichern* zu entfernen. Dies könnte dadurch erreicht werden indem auf eine Variable verwiesen wird, die den Inhalt des jeweiligen *Widgets* bereithält. Dies ist bei *Widgets* von Typ *Tk::Entry* durch die Option "-textvariable" möglich. Bei *Widgets* vom Typ *Tk::Text* dürfte sich dies jedoch schwieriger gestalten, da der Inhalt eines solchen *Widgets* nur über Funktionalitäten von *Tk* auszulesen ist.

Eine zusätzliche Anbindung zu einer Datenbank zum zentralen Zugriff auf die Pflichtenhefte wäre vom großen Nutzen. Zum Arbeiten mit Datenbanken bietet sich das Modul *DBI*<sup>57</sup> an. Unabhängig vom Datenbanktyp ermöglicht es eine einheitliche Schnittstelle zum Zugriff auf die Datenbank. Bei gleichem Datenbankmodell wäre im Idealfall ein Wechsel der Datenbank nur mit dem Wechsel des Datenbanktreibermoduls verbunden. *CPAN* bietet Datenbanktreibermodule für eine Vielzahl von Datenbanken<sup>58</sup> an, die sogenannten *DBD*-Module. Im Falle von *Firebird* könnte das *DBD*-Modul *DBD::Firebird*<sup>59</sup> die richtige Wahl darstellen. Damit ließe sich der Zugriff innerhalb eines lokalen Netzwerkes umsetzen. Der direkte Zugriff zur Datenbank über das Internet hingegen wäre keine optimale Lösung, da *Firebirds* Netzwerkprotokoll von der Authentifizierung bis hin zum Datenempfang eine hohe Latenzzeit<sup>60</sup> erfordert. Tests haben ergeben<sup>61</sup>, dass sich das im Netzwerkprotokoll von *Firebird* 2.5 im Vergleich zur Version 2.0 verbessert hat, dort sollen die Latenzzeiten bis zu 60% zurückgegangen sein. Von daher stellt der direkte Zugriff auch über das Internet eine Möglichkeit dar, die in Betracht zu ziehen wäre; Zumindest ab der *Firebird* Version 2.5. Bei älteren Versionen müsste der Zugriff über eine Zwischenanwendung erfolgen, die einen direkten Zugriff zur Datenbank ermöglicht und die Latenzzeiten somit senkt. Im Falle von *Moliri* wäre das *RPC*-Verfahren aufgrund der reichhaltigen Anzahl von *RPC*-Modulen auf *CPAN* sowie der relativ unkomplizierten Umsetzung angebracht.

<sup>56</sup>Slaven Rezić - *Tk::After* - <http://search.cpan.org/~srezic/Tk-804.029/pod/after.pod>

<sup>57</sup>Tim Bunce - *Perl DBI* - <http://dbi.perl.org/>

<sup>58</sup>Liste von *DBD-Modulen* - <http://search.cpan.org/search?query=DBD%3A%3A&mode=module>

<sup>59</sup>Popa Marius Adrian - *DBD::Firebird* - <http://search.cpan.org/~mariuz/DBD-Firebird/>

<sup>60</sup>Firebird FAQ - *Is it possible to use Firebird over the internet?* - <http://www.firebirdfaq.org/faq53/>

<sup>61</sup>Adriano dos Santos Fernandes - *Influence of network latency in the Firebird wire protocol* - <http://asfernandes.blogspot.com/2009/07/network-latency-influence-on-firebird.html>

## 9. Anhang

### 9.1 Abbildungsverzeichnis

Abbildung 1 : VIM - Vi IMproved	8
Abbildung 3: perl-support.vim	9
Abbildung 4 : Perl::Critic-Konfiguration in perl-support.vim	10
Abbildung 5 : Beispiel von Richtlinienverstößen	11
Abbildung 6 : Pflichtenheftordner - Struktur	13
Abbildung 7 : Hex-Ansicht eines ODT-Dokumentes mit Vim	19
Abbildung 8 : Vorlage des generierten Pflichtenheftes	24
Abbildung 9 : Oberer Bereich des Titelblattes	27
Abbildung 10 : Unterer Bereich des Titelblattes	27
Abbildung 11 : Fußleiste	27
Abbildung 12 : Zielbestimmungen	28
Abbildung 13 : Ergänzungen	29
Abbildung 14 : Funktionen	29
Abbildung 15 : Qualitätsanforderungen	30
Abbildung 16 : Produktübersicht mit eingefügter Bilddatei	31
Abbildung 17 : Textdatei - Kopf	32
Abbildung 18 : Textdatei - Eränzungen	32
Abbildung 19 : Textdatei – Funktionen	33
Abbildung 20 : Textdatei - Technische Umgebung	33
Abbildung 21 : Textdatei - Qualität	34
Abbildung 22 : Textdatei – Zielbestimmungen	34
Abbildung 23 : Moliri - Verwaltung	37
Abbildung 24 : Moliri - Neues Pflichtenheft bei Fehler in der Texteingabe	38
Abbildung 25 : Moliri - Neues Pflichtenheft	38
Abbildung 26 : Moliri - Neue Version bei Auswahl des Pflichtenheftes	39
Abbildung 27 : Moliri - Neue Version bei fehlerhafter Eingabe	39
Abbildung 28 : Moliri - Neue Version Pflichtenheft bereits ausgewählt	40
Abbildung 29 : Moliri - Neue Version Inhalt auswählen	40
Abbildung 30 : Moliri - Verlauf	41
Abbildung 31 : Moliri - Pflichtenheft speichern	41
Abbildung 32 : Moliri - Pflichtenheft laden	42
Abbildung 33 : Moliri - Pflichtenhefte exportieren	42
Abbildung 34 : Moliri - Konfiguration	43
Abbildung 35 : Moliri - Wartung	43
Abbildung 36 : Moliri - Version	44
Abbildung 37 : Moliri - Pflichtenhefteditor	45
Abbildung 38 : Moliri - Pflichtenhefteditor - Vorschau	47
Abbildung 39 : Moliri - Pflichtenhefteditor - neues Datum anlegen - Dialog	48

## 9.2 Literaturverzeichnis

### **[Balz96]**

Balzert, Helmut: "Lehrbuch der Software-Technik" (1. Auflage 1996), Spektrum Akademischer Verlag GmbH

### **[Conw05]**

Conway, Damian: "Perl Best Practices" (1. Auflage 2005), O'Reilly Media, Inc

### **[Balz04]**

Balzer, Heide: "Webdesign & Web-Ergonomie" (1. Auflage 2004), W3L GmbH

### **[DIN69905]**

DIN69905: "Projektabwicklung, Begriffe", Beuth Verlag GmbH

### **[LiWa02]**

Lidie, Steve und Walsh, Nancy: "Mastering Perl/Tk" (1. Auflage 2002), O'Reilly & Associates, Inc

## 9.3 Glossar

### A

- **Applikation:** Software die verwendet wird um eine meist umfangreiche Aufgabe zu erfüllen.
- **Aufzählungssymbol:** Symbol dass sich am Anfang einer Liste bzw. Auflistung befindet.

### B

- **Baumstruktur:** Eine hierarchische Anordnung von Einträgen.
- **Benutzeroberfläche:** Schnittstelle zwischen Benutzer und dem Programm, die es einem Benutzer erlaubt über grafische Symbole mit dem System zu interagieren.
- **Benutzer:** Anwender einer Software.
- **Button:** Bedienelement von grafischen Benutzeroberflächen.

### C

- **CPAN:** Akronym für **C**omprehensive **P**erl **A**rchive **N**etwork. CPAN ist ein weltweit gespiegeltes Online-Archiv für Perl-Module, -Anwendungen und Dokumentationen.
- **CPU:** Akronym für **C**entral **P**rocessing **U**nit. Die CPU ist die zentrale Verarbeitungseinheit in einem Computer.

### D

- **Datenbank:** Eine digitale Datenhaltung, mit dem Zweck große Mengen von Daten dauerhaft zu speichern.
- **Datenbankmodell:** Bestimmt auf welche Art die Daten in der Datenbank abgespeichert werden.
- **Datenhaltung:** Die Art in der Daten von Programm gespeichert werden.
- **DIN:** Akronym für **D**eutsches **I**nstitut für **N**ormung. Das Deutsche Institut für Normung ist eine Normungsorganisation der Bundesrepublik Deutschland.
- **DOM:** Akronym für **D**ocument **O**bject **M**odel. DOM ist eine Spezifikation einer Schnittstelle für den Zugriff auf XML-Dokumente.
- **Dropdown-Menü:** Bedienelement einer grafischen Benutzeroberfläche bei dem durch Interaktion des Benutzers eine Liste Auflistung von Daten sichtbar wird.

## E

- **Editoren:** Programme zum Bearbeiten von Dokumenten.
- **Eiffel:** Eine Objektorientierte Programmiersprache.
- **Entwicklungsumgebung:** Oder auch IDE. Ist eine Zusammenstellung mehrerer Programme zum Entwickeln von Software.
- **Event:** Ereignis zum Steuern des Programmflusses.

## F

- **Feldbefehle:** Dienen als Platzhalter für dynamisch erzeugten Inhalt innerhalb von Textverarbeitungssoftware.
- **Filehandler:** Struktur einer Programmiersprache zum Zugriff auf Dateien des Betriebssystems.

## G

- **GNOME:** Akronym für GNU Network Object Model Environment.
- **GNU/Linux:** Betriebssystem, dass auf dem Linux-Kernel und den GNU-Werkzeugen basiert.
- **GNU:** Rekursives Akronym für **G**nu's **n**ot **U**nix.
- **Gtk2-Perl:** Schnittstelle für Perl zu GNOMEs Bedienelementen.
- **GUI:** Akronym für **G**raphical **U**ser **I**nterface. Grafische Schnittstelle zwischen Benutzer und dem Programm (siehe Benutzeroberfläche).

## H

- **Handler:** Als Teil einer ereignisgesteuerten Architektur führt es übergebene Funktionen zu bestimmten Bedingungen aus.
- **Heimverzeichnis:** Der Ort an dem üblicherweise persönliche und benutzerspezifische Daten des Benutzers abgelegt werden und Programme ihre Konfigurationen speichern können.

## I

- **Implementierung:** Andere Bezeichnung für Umsetzung der Programmieraufgaben.

## J

- **Java:** Objektorientierte Programmiersprache.



## K

- **Kommentare:** Teile eines Quelltextes die nicht vom Compiler oder Interpreter berücksichtigt werden.

## L

- **Latenzzeit:** Die Zeit die zwischen zwei Aktionen verstreicht.
- **LibreOffice:** Quelloffene Textverarbeitungssoftware.

## M

- **Maschinencode:** Bezeichnet eine Abfolge von Anweisungen, die ein Prozessor eines datenverarbeitenden Systems direkt ausführen kann.
- **Metadaten:** Enthalten Informationen über andere Daten.
- **Miniaturbild:** Vorschau des Dokumenteninhaltes als Bilddatei.
- **mlr:** Moliri-Eigenes Dateiformat
- **Modula-3:** Imperative und objektorientierte Programmiersprache.
- **Modul:** Bezeichnet in Perl eine funktionale Einheit oder Klasse

## N

- **Netzwerkprotokoll:** Legt fest auf welche Art und Weise Programme im Netzwerk miteinander kommunizieren.
- **Nokia:** Telekommunikationskonzern mit Hauptsitz in Finnland.

## O

- **OASIS:** Akronym für **O**rganization for the **A**dvancement of **S**tructured **I**nformation **S**tandards. Internationale, nicht-gewinnorientierte Organisation, die sich mit der Weiterentwicklung von E-Business- und Webservice-Standards beschäftigt.
- **OLE-Objekte:** Akronym für **O**bject **L**inking and **E**mbelling. Elemente eines Verbunddokumentes. Bei einem Word-Dokument mit einem eingebundenen Excel-Diagramm, ist das Excel-Diagramm das OLE-Objekt.
- **Open Source Community:** Eine Gemeinschaft in der Quelltexte und allgemeine Informationen frei verfügbar sind.
- **OpenOffice.org:** Quelloffene Textverarbeitungssoftware.

## P

- **Packetverwaltung:** Ermöglicht die komfortable Verwaltung von Programmen in Packetform. Somit können die Aktualisierung und Installation von Programmen weitestgehend automatisiert werden.

- **Parser:** Übernimmt die Aufbereitung von Daten zur Weiterverarbeitung.
- **Pixel:** Einheit des Bildschirmrasters.
- **Plugins:** Optionale Erweiterung für Software.
- **POD:** Akronym für **P**lain **O**ld **D**ocument. Auszeichnungssprache zur Dokumentation von Perl-Quelltexten.
- **Portabilität:** Die Eigenschaft einer Software auf mehreren Systemen, die sich hinsichtlich ihres Betriebssystems und/oder Konfiguration unterscheiden, lauffähig zu sein.
- **Prolog:** Eine logische Programmiersprache.
- **Python:** Eine dynamische, objektorientierte Programmiersprache.

## Q

- **Qt:** Software die Bibliotheken zum entwickeln anderer Software bietet.
- **Quelltext:** In einer Programmiersprache geschriebener Text, aus dem der Compiler Maschinencode generieren kann.

## R

- **RPC:** Akronym für **R**emote **P**rocedure **C**all. Ermöglicht es Funktionen auf fremden Rechnern auszuführen und sie dabei als eigene systemeigene Funktionen zu behandeln.
- **Rückruffunktion:** Wird als Teil einer ereignisgesteuerten Architektur vom Handler zu bestimmten Bedingungen ausgeführt.

## S

- **SAX2:** Rekursives Akronym für **S**imple **A**PI for **X**ML. Standard zum Beschreiben einer Programmschnittstelle die auf XML zugreift.
- **Scheme:** Programmiersprache mit mehreren Paradigmen.
- **Schleifen:** Kontrollstruktur die einen Programmblock wiederholt.
- **Standardprogramm:** Vom Benutzer festgelegtes Programm zum öffnen eines bestimmten Dateityps.
- **StarOffice:** Textverarbeitungssoftware basierend auf den Quelltexten von OpenOffice.org. Die Entwicklung wurde im April 2011 eingestellt.
- **Steuerelemente:** Element einer grafischen Benutzeroberfläche, dass die Interaktion mit dem Benutzer erlaubt, wie z.B. Buttons oder Scrollbalken.
- **Swing:** Bibliotheken zum erstellen grafischer Benutzeroberflächen mit der Programmiersprache Java.
- **Syntax:** System von Regeln, nach denen Konstruktionen gebildet werden.
- **Systeme:** Im Rahmen dieser Diplomarbeit ein Synonym für Betriebssystem.

## T

- **Tcl:** Stark vereinfachte Skriptsprache.
- **TeX:** Textsatzsystem mit eingebauter Makrosprache zum Erstellen von hochwertigen Textdokumenten.
- **Textverarbeitungssoftware:** Software zum Erstellen und Bearbeiten von Texten, meistens aus dem Bürobereich wie OpenOffice.org oder Microsoft Word 2007.
- **TIMTOWTDI:** Akronym für **T**here is **m**ore than **o**ne **w**ay to **d**o **i**t. In Zusammenhang mit Perl bedeutet dies, dass es mehrere Wege gibt ein Problem oder eine Aufgabe zu lösen. Bei gleichem Problem ist eine unterschiedliche Syntax erlaubt.

## U

- **Ubuntu:** Die im Desktopbereich am weitesten verbreitetste GNU/Linux-Distribution.
- **Unix:** Mehrbenutzer-Betriebssystem, Vorgänger von GNU/Linux.
- **UTF-8:** Eine Kodierung für Unicodezeichen die bis zu 8 Byte zur Darstellung eines Zeichens erlaubt.

## W

- **W3C:** Akronym für **W**orld **W**ide **W**eb **C**onsortium. Gremium zur Standardisierung der World Wide Web betreffenden Techniken.
- **Weblinks:** Weblink ist ein elektronischer Verweis auf ein sich im Netzwerk befindliches Dokument. Meistens eine Internetseite.
- **Widget:** In Zusammenhang mit dieser Diplomarbeit ein Element eines Widget-Toolkits wie Tk
- **Widget-Toolkit:** Bibliotheken zum Programmieren einer grafischen Benutzeroberfläche.
- **wxPerl:** Bibliothek zum Entwickeln grafischer Benutzeroberflächen.

## X

- **XML:** Eine Auszeichnungssprache zur Darstellung von hierarchisch angeordneten Daten.
- **XPath:** Eine Abfragesprache zum gezielten Zugriff auf Teile eines XML-Dokumentes.

## Z

- **ZIP-Archiv:** Ein Format zum komprimieren von Daten.
- **Zwischenanwendung:** Eine Anwendung die Daten zwischen Anwendungen weiterleitet bzw. vermittelt.