



Fachinformatiker für Anwendungsentwicklung
Dokumentation zur schulischen Projektarbeit im Fach P/LZ

Aufbau einer DMZ

in einem mittelständischen Unternehmen

Arbeitsgruppe 9: Rico Krüger, Andreas Biller



Abbildung 1: DMZ zwischen Nord- und Südkorea

Abgabetermin: Berlin, den 25.06.2017



Oberstufenzentrum Informations- und Medizintechnik
Haarlemer Str. 23-27, 12359 Berlin

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Listings	V
Abkürzungsverzeichnis	VI
1 Einleitung	1
1.1 Projektumfeld	1
1.2 Projektziel	1
1.3 Projektbegründung	2
1.4 Projektschnittstellen	2
1.5 Projektabgrenzung	3
2 Projektplanung	3
2.1 Projektphasen	3
2.2 Zeitplanung	4
2.3 Abweichungen vom Projektantrag	4
2.4 Ressourcenplanung	4
2.5 Entwicklungsprozess	5
3 Analysephase	5
3.1 Ist-Analyse	5
3.2 Wirtschaftlichkeitsanalyse	5
3.2.1 „Make or Buy“-Entscheidung	6
3.2.2 Projektkosten	6
3.2.3 Amortisationsdauer	7
3.3 Nutzwertanalyse	7
3.4 Anwendungsfälle	7
3.5 Qualitätsanforderungen	7
3.6 Lastenheft/Fachkonzept	7
3.7 Zwischenstand	7
4 Entwurfsphase	8
4.1 Zielplattform	8
4.2 Architekturdesign	8
4.3 Entwurf der Benutzeroberfläche	9
4.4 Datenmodell	9
4.5 Geschäftslogik	9
4.6 Maßnahmen zur Qualitätssicherung	10

Inhaltsverzeichnis

4.7	Pflichtenheft/Datenverarbeitungskonzept	10
4.8	Zwischenstand	11
5	Implementierungsphase	11
5.1	Implementierung der Datenstrukturen	11
5.2	Implementierung der Benutzeroberfläche	11
5.3	Implementierung der Geschäftslogik	12
5.4	Zwischenstand	12
6	Abnahmephase	12
6.1	Zwischenstand	12
7	Einführungsphase	13
7.1	Zwischenstand	13
8	Dokumentation	13
8.1	Zwischenstand	14
9	Fazit	14
9.1	Soll-/Ist-Vergleich	14
9.2	Lessons Learned	15
9.3	Ausblick	15
	Literaturverzeichnis	16
	Eidesstattliche Erklärung	17
A	Anhang	i
A.1	Detaillierte Zeitplanung	i
A.2	Lastenheft (Auszug)	ii
A.3	Use Case-Diagramm	iii
A.4	Pflichtenheft (Auszug)	iii
A.5	Netzplan	v
A.6	Oberflächenentwürfe	vi
A.7	Screenshots der Anwendung	viii
A.8	Entwicklerdokumentation	x
A.9	Testfall und sein Aufruf auf der Konsole	xii
A.10	Klasse: ComparedNaturalModuleInformation	xiii
A.11	Klassendiagramm	xvi
A.12	Benutzerdokumentation	xvii

Abbildungsverzeichnis

1	DMZ zwischen Nord- und Südkorea	1
2	Vereinfachtes ER-Modell	9
3	Netzplan DMZ Arbeitsgruppe 9	10
4	Use Case-Diagramm	iii
5	Netzplan der DMZ (Arbeitsgruppe 9)	v
6	Liste der Module mit Filtermöglichkeiten	vi
7	Anzeige der Übersichtsseite einzelner Module	vii
8	Anzeige und Filterung der Module nach Tags	vii
9	Anzeige und Filterung der Module nach Tags	viii
10	Liste der Module mit Filtermöglichkeiten	ix
11	Aufruf des Testfalls auf der Konsole	xiii
12	Klassendiagramm	xvi

Tabellenverzeichnis

1	Zeitplanung	4
2	Kostenaufstellung	6
3	Zwischenstand nach der Analysephase	8
4	Entscheidungsmatrix	8
5	Zwischenstand nach der Entwurfsphase	11
6	Zwischenstand nach der Implementierungsphase	12
7	Zwischenstand nach der Abnahmephase	13
8	Zwischenstand nach der Einführungsphase	13
9	Zwischenstand nach der Dokumentation	14
10	Soll-/Ist-Vergleich	14

Listings

Listings/tests.php	xii
Listings/cnmi.php	xiii

Abkürzungsverzeichnis

DMZ	Demilitarisierte Zone
FA54	Klassenbezeichnung am OSZ IMT)
ITS	Informationstechnische Systeme
P/LZ	Projekt/Linux-Zertifizierung
API	Application Programming Interface
CSV	Comma Separated Value
EPK	Ereignisgesteuerte Prozesskette
ERM	Entity-Relationship-Modell
HTML	Hypertext Markup Language
MVC	Model View Controller
PHP	Hypertext Preprocessor
SQL	Structured Query Language
SVN	Subversion
UML	Unified Modeling Language
XML	Extensible Markup Language

1 Einleitung

1.1 Projektumfeld

Unternehmen:

"Das OSZ IMT in der Haarlemer Straße in Berlin-Britz im Bezirk Neukölln ist eines von 36 Oberstufenzentren in Berlin. Es vereint das Berufliche Gymnasium, die Berufsoberschule, die Fachoberschule, die Berufsfachschule, die Fachschule und die Berufsschule. (...) [An ihm] arbeiten etwa 160 Lehrkräfte und nichtpädagogisches Personal in Laboren, Werkstätten, Lernbüros und allgemeinen Unterrichtsräumen. (...) [Es] hat rund 3000 Schüler (...) [und] ist die größte Schule Berlins für Informationstechnik und Deutschlands größte Schule für Medizintechnik."¹ Wir besuchen dort seit 2 bzw. 1.5 Jahren den Unterricht der Klasse Klassenbezeichnung am OSZ IMT) (FA54).

Auftraggeber:

Als angehende Fachinformatiker für Anwendungsentwicklung am OSZ IMT sollen wir nun im Rahmen des Faches Projekt/Linux-Zertifizierung (P/LZ) ein auf mittelständige Unternehmen anwendbares IT-Sicherheitskonzept entwickeln. Dazu werden wir im Verlauf des Projektunterrichtes eine Demilitarisierte Zone (DMZ) unter Verwendung des zuvor in Informationstechnische Systeme (ITS) erlernten Wissens über Netzwerktechnik einrichten. Gleichzeitig erarbeiten wir uns Anhand eines Online-Kurses der Cisco-Networking-Academy die für das Projekt benötigten Grundkenntnisse im Umgang mit Linux. Verantwortlicher Auftraggeber und unser Ansprechpartner für dieses Projekt ist **Herr Ralf Henze**, Netzwerktechniker und Lehrer am OSZ IMT in den Unterrichtsfächern ITS und P/LZ.

1.2 Projektziel

Projekthintergrund:

Neben dem offensichtlichen Ziel dieses Projektes, ein DMZ-Netzwerk unter Linux einzurichten, will es uns als Teil des Berufsschulunterrichtes natürlich vor allem etwas beibringen. So ist die eigentliche Projektarbeit durchzogen von unterschwelligem Langzeitnutzen für unsere berufliche Entwicklung. Das Wissen, wie und wo man jederzeit Befehle nachschlagen kann, die beneidenswerten Möglichkeiten mit grep, pipes und kleinen Tools wie xargs erstaunlich komplizierte Probleme lösen zu können. Auch die bewusst schon fast aufs Niveau der IHK angehobenen Anforderungen an die Projektdokumentation und das Nahelegen, für deren Erstellung mit einer Sprache wie L^AT_EX zu arbeiten, anstelle dies mit gängigen Office Paketen zu tun, waren eine gute Vorbereitung und hervorragende Übung. So konnte Gelerntes durch praktisches Anwenden gefestigt und Neues sinnvoll ausprobiert werden.

Ziel des Projekts:

Die eigentliche Kernaufgabe des Projektes ist die Planung und praktische Umsetzung eines grundlegenden IT-Sicherheitskonzeptes mit Hilfe eines DMZ-Netzwerkes und dessen Absicherung durch das Setzen bzw. Löschen von Firewall-Regeln über ein Shell-Script. Die demilitarisierte Zone soll zwischen

¹Pressemappe, "Porträt des OSZ IMT"?

1 Einleitung

den Windows-Clients des Kunden im internen Netz und den potentiell schädlichen Anfragen der restlichen Welt aus dem externen Netzwerk liegen. Hier steht auch der Windows-Webserver des Kunden, welcher sowohl von Innen (zur Wartung) wie auch von Außen (für Besucher) erreichbar sein muss. Zwei virtuelle Linuxmaschinen sollen als Router zwischen den Netzen konfiguriert werden, wobei der Äußere sowohl das NATen als auch die Funktion der Firewall übernehmen soll. Planung und Umsetzung sollen umfassend Dokumentiert werden. Jedes Gruppenmitglied soll ein Kompetenzportfolio führen, in dem er seine Kenntnisse, Gelerntes und Probleme vor, während und nach den Aufgaben der Projektarbeit sammelt und kritisch analysiert.

1.3 Projektbegründung

Nutzen des Projekts:

Neben dem bereits mehrfach erwähnten Lerneffekt für uns als Schüler, sowohl in den Grundlagen der IT-Sicherheit, des Arbeitens auf dem Linux-Filesystem mit Hilfe der CLI, wie auch der Wiederholung der Befehle zur Konfiguration von Netzwerken und Schnittstellen in einer neuen leicht anderen Syntax, liegt der Projektnutzen wohl vor Allem auf dem Verstehen der Arbeitsweise von Access-Control-Listen, der Bedeutung der drei Chains sowie eines besseren Einblicks in die Welt der Linux-Distributionen, deren Stärken und Schwächen sowie deren Konfiguration. Und da das Projekt den Auftraggeber faktisch nichts kostet, uns aber fachlich weiter bringt, ist dessen Durchführung für beide Seiten ein Win-Win-Geschäft.

Motivation:

Grundlegende Motivation ist wohl für jeden Bereiligten an diesem Projekt seine ganz eigene Sache. Der Auftraggeber ist daran interessiert, ein fertiges, funktionierendes System zu erhalten, welches seine Wünsche und Anforderungen erfüllt, aber er und auch wir können darüber hinaus uns und uns gegenseitig an greifbaren Indikatoren bezüglich unserer Fachkompetenz bewerten. Wir stellen uns somit einer solchen Aufgabe, um etwas neues zu lernen, etwas zu wiederholen und uns zu verbessern. Oder einfach, weil wir es können. Manchmal auch, um uns auf eine Zertifizierung vorzubereiten.

1.4 Projektschnittstellen

Technisch gesehen interagieren in unserem Projekt zwei oder mehrere Windows-Rechner, welche über das Labornetzwerk des Raumes 3.1.01 verbunden sind. Auf beiden läuft jeweils eine Linux Debian Distribution in einer virtuellen Umgebung durch den VMWare Player. Die Schnittstellen der virtuellen Linuxdistributionen wiederum sind über den Bridged Modus in den Netzwerkeinstellungen des VMWare Players mit einer der physikalischen Netzwerkschnittstelle des Host-PCs verbunden. Über das Labornetz kann Verbindung zu den Rechnern der anderen Gruppen aufgenommen werden.

Die Unterrichtszeit für das Projekt, sowie die Infrastruktur (Pro Gruppe 2 Rechner + benötigte Peripherie, 2 virtuelle Maschinen und alle sonst benötigten Ressourcen, Zugang zum Internet und ins Labornetz) und alles weitere wird uns im Rahmen des P/LZ-Unterrichtes zur Verfügung gestellt.

Dank der theoretischen Natur des Projektes sind die einzigen Benutzer unseres Projektes wir, evtl.

2 Projektplanung

unsere Mitschüler während des Erfahrungsaustausches untereinander, sowie unser Auftraggeber, Herr Henze, der sich immer wieder über den aktuellen Stand informiert und auch die finale Abnahme des Projektes übernimmt.

Zur finalen Abnahme durch den Kunden sollen sowohl die Funktionalität der Firewall-Regeln nachweislich testbar sein, als auch die Projektdokumentation inkl. einer Kopie des verwendeten Firewall-Scriptes, den tabellarisch erfassten Testresultaten sowie je eines Kompetenzportfolios pro Gruppenmitglied zur Abgabe vorliegen.

1.5 Projektabgrenzung

Was dieses Projekt nicht bietet:

Dieses Projekt will auf keinen Fall den Anspruch erheben, durch die verwendeten Techniken ein Netzwerk oder System perfekt und allumfassend vor unbefugtem Eindringen schützen zu können. Es vermittelt nur Einblicke in die Grundlagen der Netzwerktechnik und IT-Sicherheit. Ein perfektes und vor allen schädlichen Einflüssen geschütztes System kann es nicht geben. Weiterführende Informationen zur Verbesserung der Systemsicherheit können aber der im Quellverzeichnis angegebenen Literatur entnommen werden.

2 Projektplanung

Da unser Projekt über die Dauer eines ganzen Schuljahres angelegt ist und wir die Unterrichtszeit zum Teil mit dem Erlernen von Fertigkeiten im Umgang mit Linux verbringen werden, muss der Ablauf genau geplant werden. Im folgenden erläutern wir die einzelnen Projektphasen, welche Ressourcen genutzt wurden und wann die Durchführung von der Planung abgewichen ist.

2.1 Projektphasen

Im Rahmen des P/LZ Unterrichts erhalten wir in jeder Schulwoche meist Freitags für je zwei Blöcke a 90 Minuten Zugang zum Labor 3.1.01 am OSZ IMT in Berlin. Das Schuljahr umfasst 14 Schulwochen in denen das Projekt durchgeführt werden muss. Außerhalb der Schulzeit können wir Private Ressourcen nutzen und planen pro Schulwoche jeweils 6 Stunden Freizeit am Wochenende als zusätzliche Pufferzeit ein. Die 42 Laborstunden und die Pufferzeit von 84 Stunden ergeben eine Gesamtzeit von 126 Stunden bis zur Projektabgabe.

Wir gehen davon aus die grundlegende Planung und Analyse in den ersten beiden Schulwochen durchzuführen, die nächsten drei Schulwochen sollte das Netzwerk entworfen und erstellt werden. Anschließend wollen wir mit der Implementierung der Firewall beginnen, wofür wir ca. vier Schulwochen einplanen. Die Restliche Schulzeit wird für die Erstellung der Dokumentation und eine Stunde für die Abnahme durch den Kunden verplant. Je nach Bedarf kann die Pufferzeit zu weiterer Recherche zuhause genutzt werden.

2.2 Zeitplanung

Tabelle 1 zeigt unsere Zeitplanung für die einzelnen Projektphasen:

Projektphase	Geplante Zeit
Analysephase	6 h
Entwurfsphase	9 h
Implementierungsphase	12 h
Abnahmetest der Fachabteilung	1 h
Erstellen der Dokumentation	14 h
Pufferzeit	84 h
Gesamt	126 h

Tabelle 1: Zeitplanung

2.3 Abweichungen vom Projektantrag

Aufgrund unserer Unerfahrenheit im Umgang mit \LaTeX gestaltet sich die Erstellung der Projektdokumentation leider schwieriger als vermutet. Zudem konnten die Funktionstests an unserer Firewall nicht bis zum Ende des letzten Unterrichtsblockes abgeschlossen werden, worauf Herr Krüger viel Zeit damit verbracht hat, eine zweite Testumgebung für unser Firewall-Script mit Windows Server 2016 zu virtualisieren, deren Installation und Konfiguration im Anhang dokumentiert wurde. Deshalb erbaten wir eine kurzzeitige Verlängerung der Abgabefrist und konnten nur die ebenfalls im Anhang zu findende Schritt-für-Schritt Anleitung einsenden, welche während des Unterrichtes erstellt und benutzt wurde.

2.4 Ressourcenplanung

Für die Durchführung im Labor werden benötigt: 2 Rechner mit Windows (und einem Benutzeraccount mit Adminrechten), die Software VMWare Player, eine Distribution von Debian für die virtuelle Maschine, Zugang zum Labornetz, ein Webserver und ein Editor zum Bearbeiten von HTML, Zugang zum Internet für Recherche, Software zum Festhalten der Ergebnisse, Software zum Durchführen von Tests. Zusätzlich bedarf es der Unterstützung durch fachkundige Mitschüler wie den Herren Habekost, Schernekau und Mahnke sowie Hilfe durch Herrn Henze bei schwereren Problemen.

Für die Arbeit außerhalb der Schule haben wir zur Recherche und für weitere Versuche sowohl Rechner mit Ubuntu 14.04 als auch Rechner mit Windows 7 und 10 und eigene Heimnetzwerke mit Internetanbindung. Auch die benötigte Software sowie \LaTeX und Editoren um die Dokumentation anzufertigen sind vorhanden. Dank einer während des Projektes angelegten Schritt-für-Schritt Anleitung zum Einrichten des Netzwerks, sowie der Möglichkeit virtuelle Maschinen zu kopieren bzw. das Versuchnetzwerk selbst zu virtualisieren, kann auch zuhause gearbeitet werden.

2.5 Entwicklungsprozess

Um unser Projekt durchzuführen benutzen wir einen auf dem Wasserfallmodell basierenden Entwicklungsprozess und den üblichen Stufen Anforderung, Entwurf, Implementation, Überprüfung und Wartung.

3 Analysephase

Im Nachfolgenden verzichten wir auf einen Großteil der üblichen Berechnungen zur Wirtschaftlichkeit des Projektes, da dieses zum Großteil unserer fachlichen Kompetenzbildung dienen soll. Darüber hinaus wäre für ein fiktives mittelständisches Unternehmen ein bereits existierendes Produkt sowohl vom zu erwartenden Arbeitsaufwand wie auch finanziell deutlich günstiger. Es wird daher lediglich eine beispielhafte Kostenberechnung für die Umsetzung der Planung durch uns erstellt und dafür ein größeres Augenmerk auf Anforderungen und Nutzen des Projekts gelegt.

3.1 Ist-Analyse

Was ist vorhanden:

Im Labor sind für jedes Gruppenmitglied vorhanden: ein Bildschirmarbeitsplatz, Windows 7, Adminrechte, zwei physikalische Netzwerkinterfaces, Anschluß an Labornetzwerk und Internet, die Software VMWare Player, Debian Images auf einem Netzlaufwerk.

Was ist zu erstellen:

Zuerst muss nun von jeder Gruppe ein Netzplan erstellt werden. Dann gilt es, die Debian 7 (Wheezy) Linux-Images in virtuellen Maschinen auf beiden Rechnern mit Hilfe des VMWare Players aufzusetzen. Diese werden zu einem Outside- und einem Inside-Router konfiguriert und die geplanten Netzwerk- und Routingeneinstellungen müssen sowohl an den virtuellen wie auch physikalischen Schnittstellen durchgeführt werden. Auf dem Rechner des Outside-Routers muss ein Webserver eingerichtet werden, wofür NAT und Port-Forwarding nötig sind. Zwischendurch wird es immer wieder der gezielten Recherche bedürfen. Um schließlich Zugriffe von außen zu regulieren, muss eine Firewall mit entsprechenden Regeln erstellt werden, die per Skript an- und abschaltbar ist. Die Funktionalität muss getestet werden und Projekt und Tests sind zu dokumentieren. Unser Lernfortschritt ist in einem Kompetenzportfolio niederzuschreiben. Gleichzeitig sind Laborübungen und Tests zu Linux-Kenntnissen zu absolvieren.

3.2 Wirtschaftlichkeitsanalyse

Wie bereits Anfänglich erwähnt, lohnt sich das Projekt für ein fiktives mittelständisches Unternehmen nur bedingt.

3.2.1 „Make or Buy“-Entscheidung

Die Kosten für eine qualifizierte Kraft zur ständigen Wartung des Servers, die durch Dauerbetrieb anfallenden Stromkosten sowie die zusätzlichen Hardwarekosten bei einem zukünftigen Upscaling übersteigen bei weitem die Kosten für einen fachkundig und sicher Administrierten Server bei einem seriösen Hosting-Anbieter.

Da unsere Empfehlung an den Kunden ein Produkt eines anderen Anbieters wäre, wird das Projekt nur zu unserem Nutzen und der Erfahrung willen, die wir damit gewinnen, umgesetzt.

3.2.2 Projektkosten

Da es sich nur um ein fiktives Projekt handelt, verzichten wir auf eine detaillierte Berechnung mit Stromkosten innerhalb des Labors, den Gehältern der Lehrkräfte oder etwaiger Lizenzgebühren. Wir beschränken uns auf eine fiktive Beispielrechnung mit unserem Stundenlohn während der Projektdauer.

Beispielrechnung (verkürzt) Die realen Kosten für die Durchführung des Projekts setzen sich sowohl aus Personal-, als auch aus Ressourcenkosten zusammen. Wir rechnen hier lediglich mit dem fiktiven Gehalt eines Auszubildenden im zweiten Lehrjahr von ca. 800 € Brutto pro Monat.

$$3 \cdot 800 \text{ €/Monat} \div 13 \div 40 \text{ h/Monat} \approx 4,62 \text{ €/h} \quad (1)$$

Es ergibt sich also ein Stundenlohn von

4 Entwurfsphase

4.1 Zielplattform

- Beschreibung der Kriterien zur Auswahl der Zielplattform (u. a. Programmiersprache, Datenbank, Client/Server, Hardware).

4.2 Architekturdesign

- Beschreibung und Begründung der gewählten Anwendungsarchitektur (z. B. [MVC](#)).
- Ggfs. Bewertung und Auswahl von verwendeten Frameworks sowie ggfs. eine kurze Einführung in die Funktionsweise des verwendeten Frameworks.

4 Entwurfsphase

Beispiel Anhand der Entscheidungsmatrix in Tabelle 4 wurde für die Implementierung der Anwendung das [PHP-Framework Symfony](#)² ausgewählt.

Eigenschaft	Gewichtung	Akelos	CakePHP	Symfony	Eigenentwicklung
Dokumentation	5	4	3	5	0
Reengineerung	3	4	2	5	3
Generierung	3	5	5	5	2
Testfälle	2	3	2	3	3
Standardaufgaben	4	3	3	3	0
Gesamt:	17	65	52	73	21
Nutzwert:		3,82	3,06	4,29	1,24

Tabelle 2: Entscheidungsmatrix

4.3 Entwurf der Benutzeroberfläche

- Entscheidung für die gewählte Benutzeroberfläche (z. B. GUI, Webinterface).
- Beschreibung des visuellen Entwurfs der konkreten Oberfläche (z. B. Mockups, Menüführung).
- Ggfs. Erläuterung von angewendeten Richtlinien zur Usability und Verweis auf Corporate Design.

Beispiel Beispielentwürfe finden sich im Anhang [A.6: Oberflächenentwürfe](#) auf Seite vi.

4.4 Datenmodell

- Entwurf/Beschreibung der Datenstrukturen (z. B. [ERM](#) und/oder Tabellenmodell, [XML-Schemas](#)) mit kurzer Beschreibung der wichtigsten (!) verwendeten Entitäten.

Beispiel In [Abbildung 2](#) wird ein Entity-Relationship-Modell ([ERM](#)) dargestellt, welches lediglich Entitäten, Relationen und die dazugehörigen Kardinalitäten enthält.

4.5 Geschäftslogik

- Modellierung und Beschreibung der wichtigsten (!) Bereiche der Geschäftslogik (z. B. mit Komponenten-, Klassen-, Sequenz-, Datenflussdiagramm, Programmablaufplan, Struktogramm, Ereignisgesteuerte Prozesskette ([EPK](#))).
- Wie wird die erstellte Anwendung in den Arbeitsfluss des Unternehmens integriert?

²Vgl. [SENSIO LABS](#) [2010].

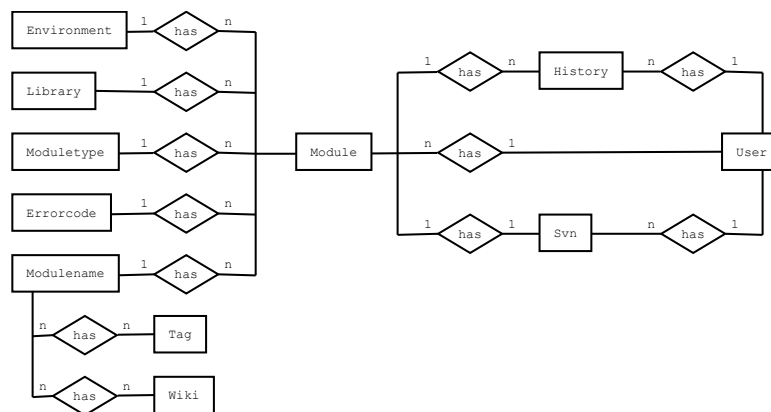


Abbildung 2: Vereinfachtes ER-Modell

Netzplan Ein Klassendiagramm, welches die Klassen der Anwendung und deren Beziehungen untereinander darstellt kann im Anhang [A.5: Netzplan](#) auf Seite [v](#) eingesehen werden.

Abbildung 3 zeigt die grundsätzliche IP-Adressverteilung in den geplanten Netzwerken. Unser Konzept teilt sich grundsätzlich in das Labornetz (hier symbolisch für den Rest der Welt), das interne, sichere Netz (mit den Windows-Clients unseres Kunden) und das als Pufferzone zwischen diesen beiden Netzen eingerichtete **DMZ**-Netzwerk (mit dem Kundenwebserver).

4.6 Maßnahmen zur Qualitätssicherung

- Welche Maßnahmen werden ergriffen, um die Qualität des Projektergebnisses (siehe Kapitel [3.5: Qualitätsanforderungen](#)) zu sichern (z. B. automatische Tests, Anwendertests)?
- Ggfs. Definition von Testfällen und deren Durchführung (durch Programme/Benutzer).

4.7 Pflichtenheft/Datenverarbeitungskonzept

- Auszüge aus dem Pflichtenheft/Datenverarbeitungskonzept, wenn es im Rahmen des Projekts erstellt wurde.

Beispiel Ein Beispiel für das auf dem Lastenheft (siehe Kapitel [3.6: Lastenheft/Fachkonzept](#)) aufbauende Pflichtenheft ist im Anhang [A.4: Pflichtenheft \(Auszug\)](#) auf Seite [iii](#) zu finden.

4.8 Zwischenstand

Tabelle [5](#) zeigt den Zwischenstand nach der Entwurfsphase.

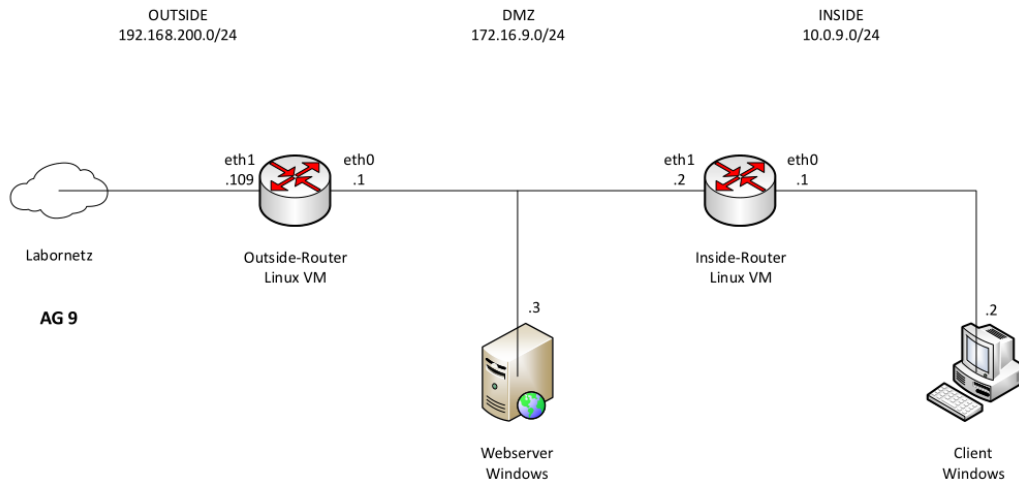


Abbildung 3: Netzplan DMZ Arbeitsgruppe 9

5 Implementierungsphase

5.1 Implementierung der Datenstrukturen

- Beschreibung der angelegten Datenbank (z. B. Generierung von [SQL](#) aus Modellierungswerkzeug oder händisches Anlegen), [XML](#)-Schemas usw..

Vorgang	Geplant	Tatsächlich	Differenz
1. Prozessentwurf	2 h	3 h	+1 h
2. Datenbankentwurf	3 h	5 h	+2 h
3. Erstellen von Datenverarbeitungskonzepten	4 h	4 h	
4. Benutzeroberflächen entwerfen und abstimmen	2 h	1 h	-1 h
5. Erstellen eines UML-Komponentendiagramms	4 h	2 h	-2 h
6. Erstellen des Pflichtenhefts	4 h	4 h	

Tabelle 3: Zwischenstand nach der Entwurfsphase

5.2 Implementierung der Benutzeroberfläche

- Beschreibung der Implementierung der Benutzeroberfläche, falls dies separat zur Implementierung der Geschäftslogik erfolgt (z. B. bei [HTML](#)-Oberflächen und Stylesheets).
- Ggfs. Beschreibung des Corporate Designs und dessen Umsetzung in der Anwendung.
- Screenshots der Anwendung

Beispiel Screenshots der Anwendung in der Entwicklungsphase mit Dummy-Daten befinden sich im Anhang [A.7: Screenshots der Anwendung](#) auf Seite [viii](#).

5.3 Implementierung der Geschäftslogik

- Beschreibung des Vorgehens bei der Umsetzung/Programmierung der entworfenen Anwendung.
- Ggfs. interessante Funktionen/Algorithmen im Detail vorstellen, verwendete Entwurfsmuster zeigen.
- Quelltextbeispiele zeigen.
- Hinweis: Wie in Kapitel 1: [Einleitung](#) zitiert, wird nicht ein lauffähiges Programm bewertet, sondern die Projektdurchführung. Dennoch würde ich immer Quelltextausschnitte zeigen, da sonst Zweifel an der tatsächlichen Leistung des Prüflings aufkommen können.

Beispiel Die Klasse `ComparedNaturalModuleInformation` findet sich im Anhang [A.10: Klasse: ComparedNaturalModuleInformation](#) auf Seite [xiii](#).

5.4 Zwischenstand

Tabelle 6 zeigt den Zwischenstand nach der Implementierungsphase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Anlegen der Datenbank	1 h	1 h	
2. Umsetzung der HTML-Oberflächen und Stylesheets	4 h	3 h	-1 h
3. Programmierung der PHP-Module für die Funktionen	23 h	23 h	
4. Nächtlichen Batchjob einrichten	1 h	1 h	

Tabelle 4: Zwischenstand nach der Implementierungsphase

6 Abnahmephase

- Welche Tests (z. B. Unit-, Integrations-, Systemtests) wurden durchgeführt und welche Ergebnisse haben sie geliefert (z. B. Logs von Unit Tests, Testprotokolle der Anwender)?
- Wurde die Anwendung offiziell abgenommen?

Beispiel Ein Auszug eines Unit Tests befindet sich im Anhang [A.9: Testfall und sein Aufruf auf der Konsole](#) auf Seite [xii](#). Dort ist auch der Aufruf des Tests auf der Konsole des Webserverns zu sehen.

6.1 Zwischenstand

Tabelle 7 zeigt den Zwischenstand nach der Abnahmephase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Abnahmetest der Fachabteilung	1 h	1 h	

Tabelle 5: Zwischenstand nach der Abnahmephase

7 Einführungsphase

- Welche Schritte waren zum Deployment der Anwendung nötig und wie wurden sie durchgeführt (automatisiert/manuell)?
- Wurden ggfs. Altdaten migriert und wenn ja, wie?
- Wurden Benutzerschulungen durchgeführt und wenn ja, Wie wurden sie vorbereitet?

7.1 Zwischenstand

Tabelle 8 zeigt den Zwischenstand nach der Einführungsphase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Einführung/Benutzerschulung	1 h	1 h	

Tabelle 6: Zwischenstand nach der Einführungsphase

8 Dokumentation

- Wie wurde die Anwendung für die Benutzer/Administratoren/Entwickler dokumentiert (z. B. Benutzerhandbuch, [API-Dokumentation](#))?
- Hinweis: Je nach Zielgruppe gelten bestimmte Anforderungen für die Dokumentation (z. B. keine IT-Fachbegriffe in einer Anwenderdokumentation verwenden, aber auf jeden Fall in einer Dokumentation für den IT-Bereich).

Beispiel Ein Ausschnitt aus der erstellten Benutzerdokumentation befindet sich im Anhang [A.12: Benutzerdokumentation](#) auf Seite [xvii](#). Die Entwicklerdokumentation wurde mittels PHPDoc³ automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse findet sich im Anhang [A.8: Entwicklerdokumentation](#) auf Seite [x](#).

8.1 Zwischenstand

Tabelle 9 zeigt den Zwischenstand nach der Dokumentation.

Vorgang	Geplant	Tatsächlich	Differenz
1. Erstellen der Benutzerdokumentation	2 h	2 h	
2. Erstellen der Projektdokumentation	6 h	8 h	+2 h
3. Programmdokumentation	1 h	1 h	

Tabelle 7: Zwischenstand nach der Dokumentation

9 Fazit

9.1 Soll-/Ist-Vergleich

- Wurde das Projektziel erreicht und wenn nein, warum nicht?
- Ist der Auftraggeber mit dem Projektergebnis zufrieden und wenn nein, warum nicht?
- Wurde die Projektplanung (Zeit, Kosten, Personal, Sachmittel) eingehalten oder haben sich Abweichungen ergeben und wenn ja, warum?
- Hinweis: Die Projektplanung muss nicht strikt eingehalten werden. Vielmehr sind Abweichungen sogar als normal anzusehen. Sie müssen nur vernünftig begründet werden (z. B. durch Änderungen an den Anforderungen, unter-/überschätzter Aufwand).

³Vgl. PHPDOC.ORG [2010]

9 Fazit

Beispiel (verkürzt) Wie in Tabelle 10 zu erkennen ist, konnte die Zeitplanung bis auf wenige Ausnahmen eingehalten werden.

Phase	Geplant	Tatsächlich	Differenz
Entwurfsphase	19 h	19 h	
Analysephase	9 h	10 h	+1 h
Implementierungsphase	29 h	28 h	-1 h
Abnahmetest der Fachabteilung	1 h	1 h	
Einführungsphase	1 h	1 h	
Erstellen der Dokumentation	9 h	11 h	+2 h
Pufferzeit	2 h	0 h	-2 h
Gesamt	70 h	70 h	

Tabelle 8: Soll-/Ist-Vergleich

9.2 Lessons Learned

- Was hat der Prüfling bei der Durchführung des Projekts gelernt (z. B. Zeitplanung, Vorteile der eingesetzten Frameworks, Änderungen der Anforderungen)?

9.3 Ausblick

- Wie wird sich das Projekt in Zukunft weiterentwickeln (z. B. geplante Erweiterungen)?

Literaturverzeichnis

ISO/IEC 9126-1 2001

ISO/IEC 9126-1: *Software-Engineering – Qualität von Software-Produkten – Teil 1: Qualitätsmodell*. Juni 2001

phpdoc.org 2010

PHPDOC.ORG: *phpDocumentor-Website*. Version: 2010. <http://www.phpdoc.org/>, Abruf: 20.04.2010

Sensio Labs 2010

SENSIO LABS: *Symfony - Open-Source PHP Web Framework*. Version: 2010. <http://www.symfony-project.org/>, Abruf: 20.04.2010

Eidesstattliche Erklärung

Wir, Rico Krüger und Andreas Biller, versichern hiermit, dass wir unsere **Dokumentation zur schulischen Projektarbeit im Fach P/LZ** mit dem Thema

Aufbau einer DMZ in einem mittelständischen Unternehmen

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben, wobei wir alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet haben. Die Arbeit wurde bisher keinem anderen Lehrer vorgelegt und auch nicht veröffentlicht.

Berlin, den 25.06.2017

ANDREAS BILLER, RICO KRÜGER

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase	9 h
1. Analyse des Ist-Zustands	3 h
1.1. Fachgespräch mit der EDV-Abteilung	1 h
1.2. Prozessanalyse	2 h
2. „Make or buy“-Entscheidung und Wirtschaftlichkeitsanalyse	1 h
3. Erstellen eines „Use-Case“-Diagramms	2 h
4. Erstellen des Lastenhefts mit der EDV-Abteilung	3 h
Entwurfsphase	19 h
1. Prozessentwurf	2 h
2. Datenbankentwurf	3 h
2.1. ER-Modell erstellen	2 h
2.2. Konkretes Tabellenmodell erstellen	1 h
3. Erstellen von Datenverarbeitungskonzepten	4 h
3.1. Verarbeitung der CSV-Daten	1 h
3.2. Verarbeitung der SVN-Daten	1 h
3.3. Verarbeitung der Sourcen der Programme	2 h
4. Benutzeroberflächen entwerfen und abstimmen	2 h
5. Erstellen eines UML-Komponentendiagramms der Anwendung	4 h
6. Erstellen des Pflichtenhefts	4 h
Implementierungsphase	29 h
1. Anlegen der Datenbank	1 h
2. Umsetzung der HTML-Oberflächen und Stylesheets	4 h
3. Programmierung der PHP-Module für die Funktionen	23 h
3.1. Import der Modulinformationen aus CSV-Dateien	2 h
3.2. Parsen der Modulquelltexte	3 h
3.3. Import der SVN-Daten	2 h
3.4. Vergleichen zweier Umgebungen	4 h
3.5. Abrufen der von einem zu wählenden Benutzer geänderten Module	3 h
3.6. Erstellen einer Liste der Module unter unterschiedlichen Aspekten	5 h
3.7. Anzeigen einer Liste mit den Modulen und geparsen Metadaten	3 h
3.8. Erstellen einer Übersichtsseite für ein einzelnes Modul	1 h
4. Nächtlichen Batchjob einrichten	1 h
Abnahmetest der Fachabteilung	1 h
1. Abnahmetest der Fachabteilung	1 h
Einführungsphase	1 h
1. Einführung/Benutzerschulung	1 h
Erstellen der Dokumentation	9 h
1. Erstellen der Benutzerdokumentation	2 h
2. Erstellen der Projektdokumentation	6 h
3. Programmdokumentation	1 h
3.1. Generierung durch PHPdoc	1 h
Pufferzeit	2 h
1. Puffer	2 h
Gesamt	70 h

A.2 Lastenheft (Auszug)

Es folgt ein Auszug aus dem Lastenheft mit Fokus auf die Anforderungen:

Die Anwendung muss folgende Anforderungen erfüllen:

1. Verarbeitung der Moduldaten

- 1.1. Die Anwendung muss die von Subversion und einem externen Programm bereitgestellten Informationen (z.B. Source-Benutzer, -Datum, Hash) verarbeiten.
- 1.2. Auslesen der Beschreibung und der Stichwörter aus dem Sourcecode.

2. Darstellung der Daten

- 2.1. Die Anwendung muss eine Liste aller Module erzeugen inkl. Source-Benutzer und -Datum, letztem Commit-Benutzer und -Datum für alle drei Umgebungen.
- 2.2. Verknüpfen der Module mit externen Tools wie z.B. Wiki-Einträgen zu den Modulen oder dem Sourcecode in Subversion.
- 2.3. Die Sourcen der Umgebungen müssen verglichen und eine schnelle Übersicht zur Einhaltung des allgemeinen Entwicklungsprozesses gegeben werden.
- 2.4. Dieser Vergleich muss auf die von einem bestimmten Benutzer bearbeiteten Module eingeschränkt werden können.
- 2.5. Die Anwendung muss in dieser Liste auch Module anzeigen, die nach einer Bearbeitung durch den gesuchten Benutzer durch jemand anderen bearbeitet wurden.
- 2.6. Abweichungen sollen kenntlich gemacht werden.
- 2.7. Anzeigen einer Übersichtsseite für ein Modul mit allen relevanten Informationen zu diesem.

3. Sonstige Anforderungen

- 3.1. Die Anwendung muss ohne das Installieren einer zusätzlichen Software über einen Webbrowser im Intranet erreichbar sein.
- 3.2. Die Daten der Anwendung müssen jede Nacht bzw. nach jedem [SVN](#)-Commit automatisch aktualisiert werden.
- 3.3. Es muss ermittelt werden, ob Änderungen auf der Produktionsumgebung vorgenommen wurden, die nicht von einer anderen Umgebung kopiert wurden. Diese Modulliste soll als Mahnung per E-Mail an alle Entwickler geschickt werden (Peer Pressure).
- 3.4. Die Anwendung soll jederzeit erreichbar sein.
- 3.5. Da sich die Entwickler auf die Anwendung verlassen, muss diese korrekte Daten liefern und darf keinen Interpretationsspielraum lassen.
- 3.6. Die Anwendung muss so flexibel sein, dass sie bei Änderungen im Entwicklungsprozess einfach angepasst werden kann.

A.3 Use Case-Diagramm

Use Case-Diagramme und weitere UML-Diagramme kann man auch direkt mit L^AT_EX zeichnen, siehe z. B. <http://metauml.sourceforge.net/old/usecase-diagram.html>.

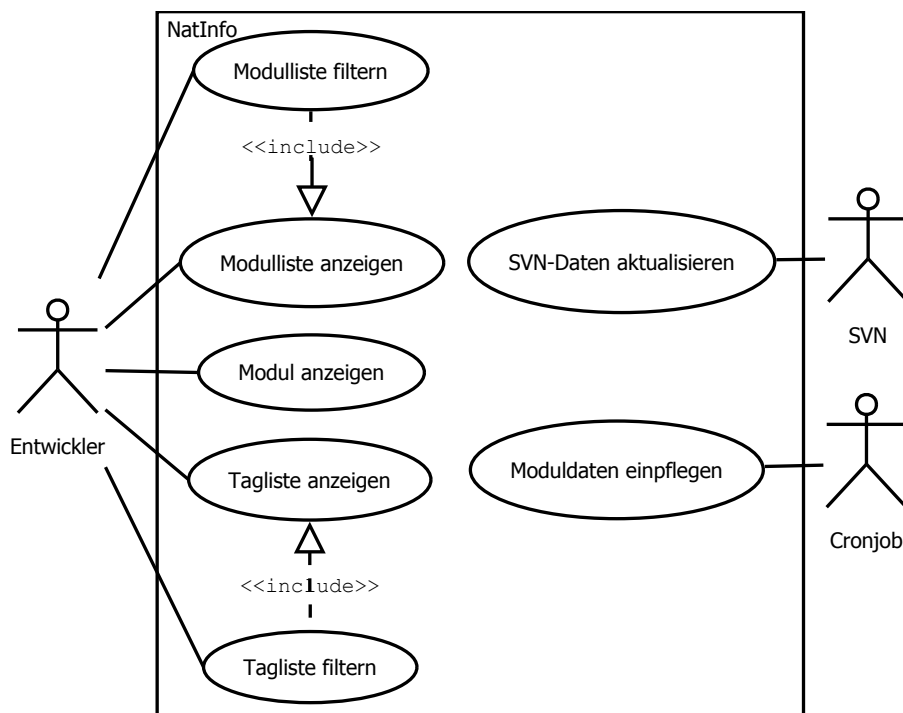


Abbildung 4: Use Case-Diagramm

A.4 Pflichtenheft (Auszug)

Zielbestimmung

1. Musskriterien

1.1. Modul-Liste: Zeigt eine filterbare Liste der Module mit den dazugehörigen Kerninformationen sowie Symbolen zur Einhaltung des Entwicklungsprozesses an

- In der Liste wird der Name, die Bibliothek und Daten zum Source und Kompilat eines Moduls angezeigt.
- Ebenfalls wird der Status des Moduls hinsichtlich Source und Kompilat angezeigt. Dazu gibt es unterschiedliche Status-Zeichen, welche symbolisieren in wie weit der Entwicklungsprozess eingehalten wurde bzw. welche Schritte als nächstes getan werden müssen. So gibt es z. B. Zeichen für das Einhalten oder Verletzen des Prozesses oder den Hinweis auf den nächsten zu tätigenden Schritt.
- Weiterhin werden die Benutzer und Zeitpunkte der aktuellen Version der Sourcen und Kompilate angezeigt. Dazu kann vorher ausgewählt werden, von welcher Umgebung diese Daten gelesen werden sollen.

- Es kann eine Filterung nach allen angezeigten Daten vorgenommen werden. Die Daten zu den Sourcen sind historisiert. Durch die Filterung ist es möglich, auch Module zu finden, die in der Zwischenzeit schon von einem anderen Benutzer editiert wurden.
- 1.2. Tag-Liste: Bietet die Möglichkeit die Module anhand von Tags zu filtern.
- Es sollen die Tags angezeigt werden, nach denen bereits gefiltert wird und die, die noch der Filterung hinzugefügt werden könnten, ohne dass die Ergebnisliste leer wird.
 - Zusätzlich sollen die Module angezeigt werden, die den Filterkriterien entsprechen. Sollten die Filterkriterien leer sein, werden nur die Module angezeigt, welche mit einem Tag versehen sind.
- 1.3. Import der Moduldaten aus einer bereitgestellten [CSV](#)-Datei
- Es wird täglich eine Datei mit den Daten der aktuellen Module erstellt. Diese Datei wird (durch einen Cronjob) automatisch nachts importiert.
 - Dabei wird für jedes importierte Modul ein Zeitstempel aktualisiert, damit festgestellt werden kann, wenn ein Modul gelöscht wurde.
 - Die Datei enthält die Namen der Umgebung, der Bibliothek und des Moduls, den Programmtyp, den Benutzer und Zeitpunkt des Sourcecodes sowie des Kompilats und den Hash des Sourcecodes.
 - Sollte sich ein Modul verändert haben, werden die entsprechenden Daten in der Datenbank aktualisiert. Die Veränderungen am Source werden dabei aber nicht ersetzt, sondern historisiert.
- 1.4. Import der Informationen aus SVN. Durch einen „post-commit-hook“ wird nach jedem Einchecken eines Moduls ein [PHP](#)-Script auf der Konsole aufgerufen, welches die Informationen, die vom SVN-Kommandozeilentool geliefert werden, an NatInfo übergibt.
- 1.5. Parsen der Sourcen
- Die Sourcen der Entwicklungsumgebung werden nach Tags, Links zu Artikeln im Wiki und Programmbeschreibungen durchsucht.
 - Diese Daten werden dann entsprechend angelegt, aktualisiert oder nicht mehr gesetzte Tags/Wikiartikel entfernt.
- 1.6. Sonstiges
- Das Programm läuft als Webanwendung im Intranet.
 - Die Anwendung soll möglichst leicht erweiterbar sein und auch von anderen Entwicklungsprozessen ausgehen können.
 - Eine Konfiguration soll möglichst in zentralen Konfigurationsdateien erfolgen.

Produkteinsatz

1. Anwendungsbereiche

Die Webanwendung dient als Anlaufstelle für die Entwicklung. Dort sind alle Informationen

für die Module an einer Stelle gesammelt. Vorher getrennte Anwendungen werden ersetzt bzw. verlinkt.

2. Zielgruppen

NatInfo wird lediglich von den Natural-Entwicklern in der EDV-Abteilung genutzt.

3. Betriebsbedingungen

Die nötigen Betriebsbedingungen, also der Webserver, die Datenbank, die Versionsverwaltung, das Wiki und der nächtliche Export sind bereits vorhanden und konfiguriert. Durch einen täglichen Cronjob werden entsprechende Daten aktualisiert, die Webanwendung ist jederzeit aus dem Intranet heraus erreichbar.

A.5 Netzplan

Der Netzplan unserer [DMZ](#)

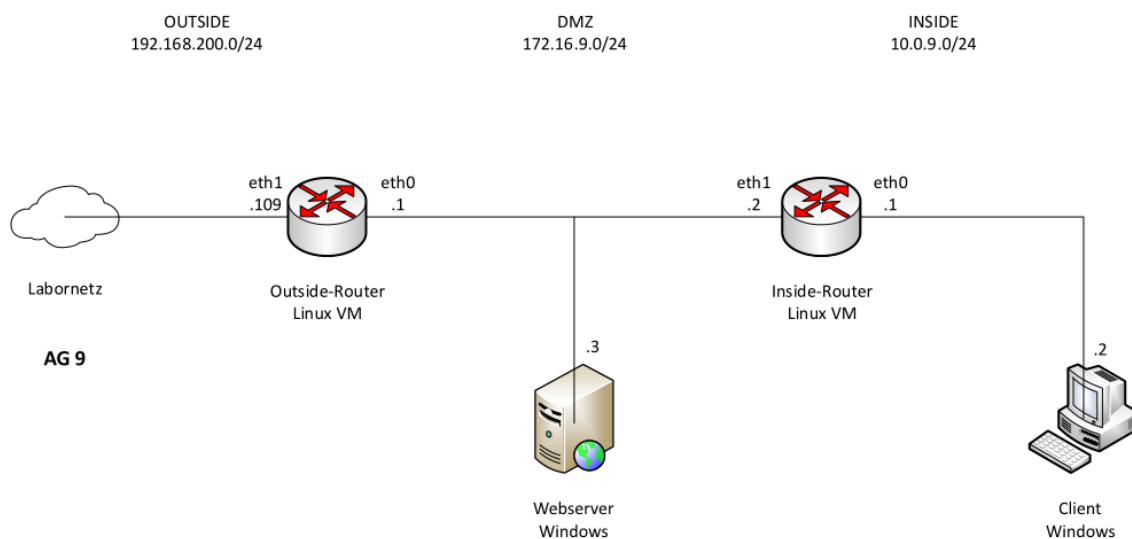


Abbildung 5: Netzplan der DMZ (Arbeitsgruppe 9)

A.6 Oberflächenentwürfe

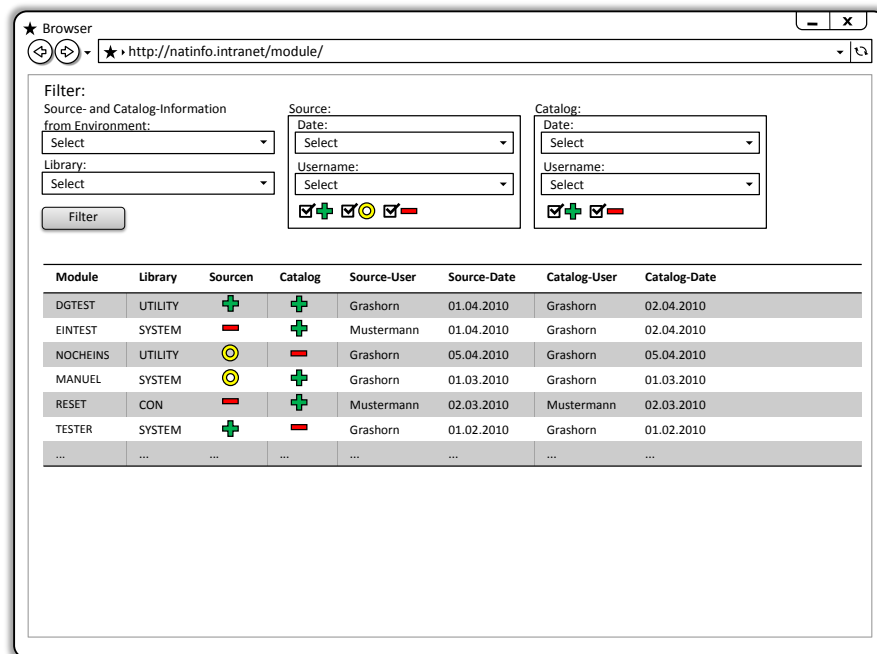


Abbildung 6: Liste der Module mit Filtermöglichkeiten

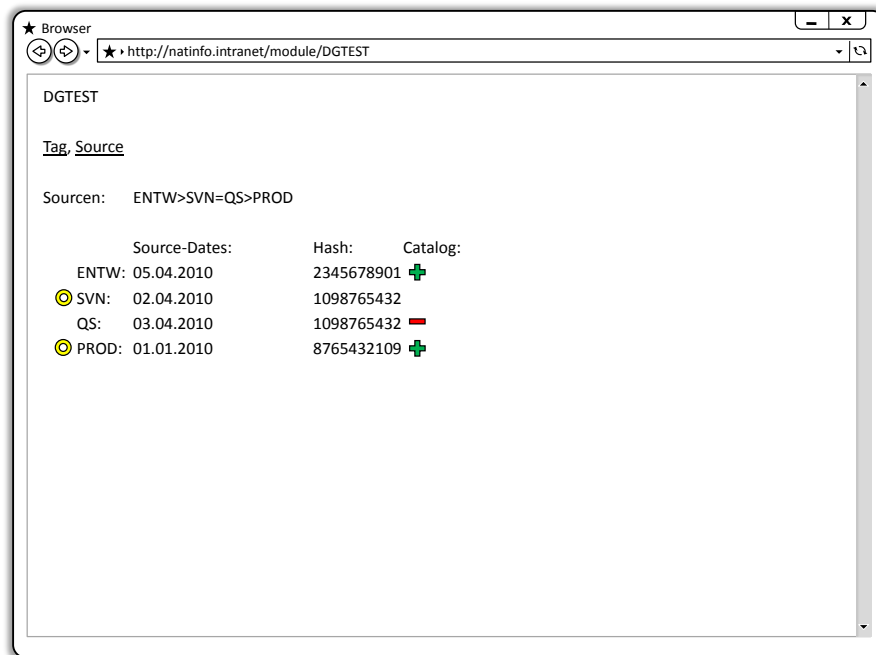


Abbildung 7: Anzeige der Übersichtsseite einzelner Module

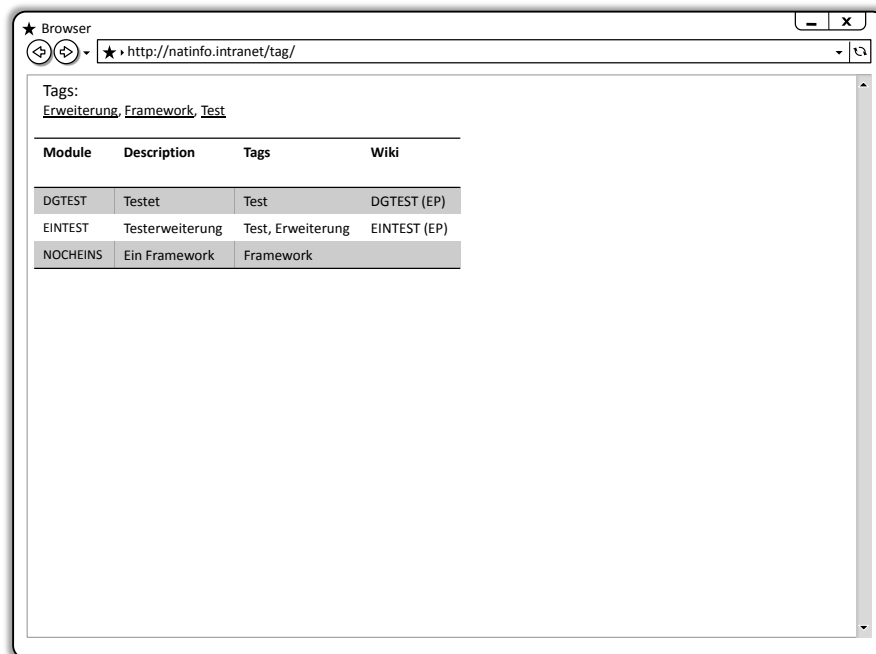


Abbildung 8: Anzeige und Filterung der Module nach Tags

A.7 Screenshots der Anwendung



Tags

Project, Test

Modulename	Description	Tags	Wiki
DGTEST	Macht einen ganz tollen Tab.	HGP	SMTAB_(EP), b
MALWAS		HGP, Test	
HDRGE		HGP, Project	
WURAM		HGP, Test	
PAMIU		HGP	

Abbildung 9: Anzeige und Filterung der Module nach Tags



Modules

Environment	ENTW
Library	Select
Catalog user	Select
Catalog date	Select
Source user	Select
Source date	Select
Reset Filter	











Name	Library	Source	Catalog	Source-User	Source-Date	Catalog-User	Catalog-Date
SMTAB	UTILITY			MACKE	01.04.2010 13:00	MACKE	01.04.2010 13:00
DGTAB	CON			GRASHORN	01.04.2010 13:00	GRASHORN	01.04.2010 13:00
DGTEST	SUP			GRASHORN	05.04.2010 13:00	GRASHORN	05.04.2010 13:00
OHNETAG	CON			GRASHORN	05.04.2010 13:00	GRASHORN	01.04.2010 15:12
OHNEWIKI	CON			GRASHORN	05.04.2010 13:00	MACKE	01.04.2010 15:12

Abbildung 10: Liste der Module mit Filtermöglichkeiten

A.8 Entwicklerdokumentation

lib-model

[class tree: lib-model] [index: lib-model] [all elements]

Packages:
lib-model

Files:
Naturalmodulename.php

Classes:
Naturalmodulename

Class: Naturalmodulename

Source Location: /Naturalmodulename.php

Class Overview

```
BaseNaturalmodulename
|
--Naturalmodulename
```

Subclass for representing a row from the 'NaturalModulename' table.

Methods

- [__construct](#)
- [getNaturalTags](#)
- [getNaturalWikis](#)
- [loadNaturalModuleInformation](#)
- [__toString](#)

Class Details

[line 10]
Subclass for representing a row from the 'NaturalModulename' table.

Adds some business logic to the base.

[\[Top \]](#)

Class Methods

constructor [__construct](#) [line 56]

```
Naturalmodulename __construct( )
```

Initializes internal state of Naturalmodulename object.

Tags:
see: parent::__construct()
access: public

[\[Top \]](#)

method [getNaturalTags](#) [line 68]

```
array getNaturalTags( )
```

Returns an Array of NaturalTags connected with this Modulename.

Tags:

return: Array of NaturalTags
access: public

[\[Top \]](#)

method getNaturalWikis [line 83]

```
array getNaturalWikis( )
```

Returns an Array of NaturalWikis connected with this Modulename.

Tags:

return: Array of NaturalWikis
access: public

[\[Top \]](#)

method loadNaturalModuleInformation [line 17]

```
ComparedNaturalModuleInformation  
loadNaturalModuleInformation( )
```

Gets the ComparedNaturalModuleInformation for this NaturalModulename.

Tags:

access: public

[\[Top \]](#)

method __toString [line 47]

```
string __toString( )
```

Returns the name of this NaturalModulename.

Tags:

access: public

[\[Top \]](#)

Documentation generated on Thu, 22 Apr 2010 08:14:01 +0200 by [phpDocumentor 1.4.2](#)

A.9 Testfall und sein Aufruf auf der Konsole

```

1 <?php
2 include(dirname(__FILE__).'/../bootstrap/Propel.php');
3
4 $t = new lime_test(13);
5
6 $t->comment('Empty Information');
7 $emptyComparedInformation = new ComparedNaturalModuleInformation(array());
8 $t->is($emptyComparedInformation->getCatalogSign(), ComparedNaturalModuleInformation::EMPTY_SIGN, '
    Has no catalog sign');
9 $t->is($emptyComparedInformation->getSourceSign(), ComparedNaturalModuleInformation::SIGN_CREATE, '
    Source has to be created');
10
11 $t->comment('Perfect Module');
12 $criteria = new Criteria();
13 $criteria->add(NaturalmodulenamePeer::NAME, 'SMTAB');
14 $moduleName = NaturalmodulenamePeer::doSelectOne($criteria);
15 $t->is($moduleName->getName(), 'SMTAB', 'Right modulename selected');
16 $comparedInformation = $moduleName->loadNaturalModuleInformation();
17 $t->is($comparedInformation->getSourceSign(), ComparedNaturalModuleInformation::SIGN_OK, 'Source sign
    shines global');
18 $t->is($comparedInformation->getCatalogSign(), ComparedNaturalModuleInformation::SIGN_OK, 'Catalog sign
    shines global');
19 $infos = $comparedInformation->getNaturalModuleInformations();
20 foreach($infos as $info)
21 {
22     $env = $info->getEnvironmentName();
23     $t->is($info->getSourceSign(), ComparedNaturalModuleInformation::SIGN_OK, 'Source sign shines at ' . $env);
24     if ($env != 'SVNENTW')
25     {
26         $t->is($info->getCatalogSign(), ComparedNaturalModuleInformation::SIGN_OK, 'Catalog sign shines at ' .
            $info->getEnvironmentName());
27     }
28     else
29     {
30         $t->is($info->getCatalogSign(), ComparedNaturalModuleInformation::EMPTY_SIGN, 'Catalog sign is empty
            at ' . $info->getEnvironmentName());
31     }
32 }
33 ?>

```

```

ao-suse-ws1.ao-dom.alte-oldenburger.de - PuTTY
ao-suse-ws1:/srv/www/symfony/natural # ./symfony test:unit ComparedNaturalModuleInformation
1..13
# Empty Information
ok 1 - Has no catalog sign
ok 2 - Source has to be created
# Perfect Module
ok 3 - Right modulename selected
ok 4 - Source sign shines global
ok 5 - Catalog sign shines global
ok 6 - Source sign shines at ENTW
ok 7 - Catalog sign shines at ENTW
ok 8 - Source sign shines at QS
ok 9 - Catalog sign shines at QS
ok 10 - Source sign shines at PROD
ok 11 - Catalog sign shines at PROD
ok 12 - Source sign shines at SVNENTW
ok 13 - Catalog sign is empty at SVNENTW
# Looks like everything went fine.
ao-suse-ws1:/srv/www/symfony/natural #

```

Abbildung 11: Aufruf des Testfalls auf der Konsole

A.10 Klasse: ComparedNaturalModuleInformation

Kommentare und simple Getter/Setter werden nicht angezeigt.

```

1 <?php
2 class ComparedNaturalModuleInformation
3 {
4     const EMPTY_SIGN = 0;
5     const SIGN_OK = 1;
6     const SIGN_NEXT_STEP = 2;
7     const SIGN_CREATE = 3;
8     const SIGN_CREATE_AND_NEXT_STEP = 4;
9     const SIGN_ERROR = 5;
10
11     private $naturalModuleInformations = array();
12
13     public static function environments()
14     {
15         return array("ENTW", "SVNENTW", "QS", "PROD");
16     }
17
18     public static function signOrder()
19     {
20         return array(self::SIGN_ERROR, self::SIGN_NEXT_STEP, self::SIGN_CREATE_AND_NEXT_STEP, self::SIGN_CREATE, self::SIGN_OK);
21     }
22
23     public function __construct(array $naturalInformations)
24     {
25         $this->allocateModulesToEnvironments($naturalInformations);

```

A Anhang

```

26     $this->allocateEmptyModulesToMissingEnvironments();
27     $this->determineSourceSignsForAllEnvironments();
28 }
29
30 private function allocateModulesToEnvironments(array $naturalInformations)
31 {
32     foreach ($naturalInformations as $naturalInformation)
33     {
34         $env = $naturalInformation->getEnvironmentName();
35         if (in_array($env, self::environments()))
36         {
37             $this->naturalModuleInformations[array_search($env, self::environments())] = $naturalInformation;
38         }
39     }
40 }
41
42 private function allocateEmptyModulesToMissingEnvironments()
43 {
44     if (array_key_exists(0, $this->naturalModuleInformations))
45     {
46         $this->naturalModuleInformations[0]->setSourceSign(self::SIGN_OK);
47     }
48
49     for ($i = 0; $i < count(self::environments()); $i++)
50     {
51         if (!array_key_exists($i, $this->naturalModuleInformations))
52         {
53             $environments = self::environments();
54             $this->naturalModuleInformations[$i] = new EmptyNaturalModuleInformation($environments[$i]);
55             $this->naturalModuleInformations[$i]->setSourceSign(self::SIGN_CREATE);
56         }
57     }
58 }
59
60 public function determineSourceSignsForAllEnvironments()
61 {
62     for ($i = 1; $i < count(self::environments()); $i++)
63     {
64         $currentInformation = $this->naturalModuleInformations[$i];
65         $previousInformation = $this->naturalModuleInformations[$i - 1];
66         if ($currentInformation->getSourceSign() <> self::SIGN_CREATE)
67         {
68             if ($previousInformation->getSourceSign() <> self::SIGN_CREATE)
69             {
70                 if ($currentInformation->getHash() <> $previousInformation->getHash())
71                 {
72                     if ($currentInformation->getSourceDate('YmdHis') > $previousInformation->getSourceDate('YmdHis'))
73                     {
74                         $currentInformation->setSourceSign(self::SIGN_ERROR);
75                     }
76                 }
77             }
78         }
79     }
80 }

```

A Anhang

```

76         else
77         {
78             $currentInformation->setSourceSign(self::SIGN_NEXT_STEP);
79         }
80     }
81     else
82     {
83         $currentInformation->setSourceSign(self::SIGN_OK);
84     }
85 }
86 else
87 {
88     $currentInformation->setSourceSign(self::SIGN_ERROR);
89 }
90 }
91 elseif ($previousInformation->getSourceSign() <> self::SIGN_CREATE && $previousInformation->
    getSourceSign() <> self::SIGN_CREATE_AND_NEXT_STEP)
92 {
93     $currentInformation->setSourceSign(self::SIGN_CREATE_AND_NEXT_STEP);
94 }
95 }
96 }
97
98 private function containsSourceSign($sign)
99 {
100     foreach($this->naturalModuleInformations as $information)
101     {
102         if($information->getSourceSign() == $sign)
103         {
104             return true;
105         }
106     }
107     return false ;
108 }
109
110 private function containsCatalogSign($sign)
111 {
112     foreach($this->naturalModuleInformations as $information)
113     {
114         if($information->getCatalogSign() == $sign)
115         {
116             return true;
117         }
118     }
119     return false ;
120 }
121 }
122 ?>

```

A.11 Klassendiagramm

Klassendiagramme und weitere UML-Diagramme kann man auch direkt mit \LaTeX zeichnen, siehe z. B. <http://metauml.sourceforge.net/old/class-diagram.html>.

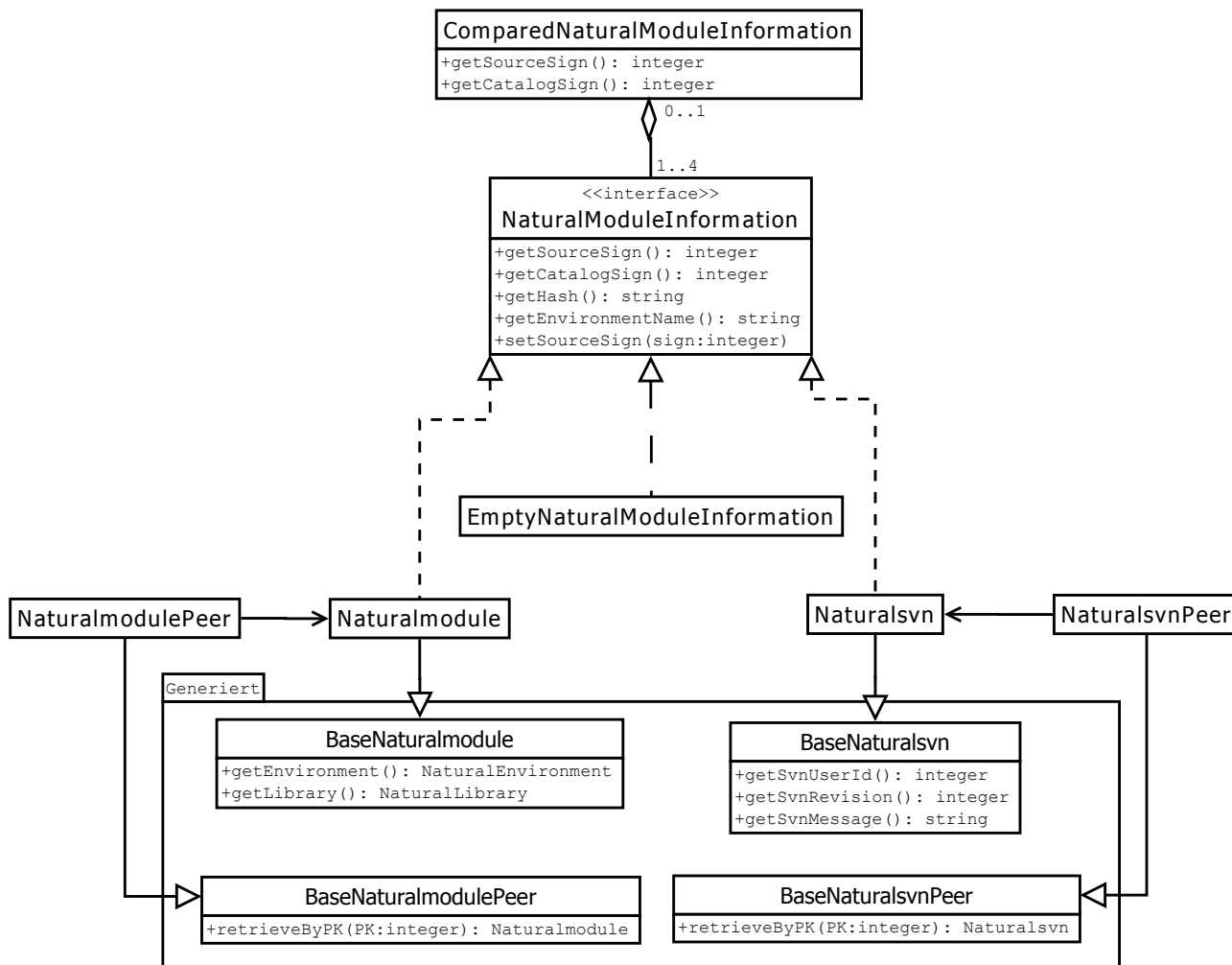







Abbildung 12: Klassendiagramm

A.12 Benutzerdokumentation

Ausschnitt aus der Benutzerdokumentation:

Symbol	Bedeutung global	Bedeutung einzeln
	Alle Module weisen den gleichen Stand auf.	Das Modul ist auf dem gleichen Stand wie das Modul auf der vorherigen Umgebung.
	Es existieren keine Module (fachlich nicht möglich).	Weder auf der aktuellen noch auf der vorherigen Umgebung sind Module angelegt. Es kann also auch nichts übertragen werden.
	Ein Modul muss durch das Übertragen von der vorherigen Umgebung erstellt werden.	Das Modul der vorherigen Umgebung kann übertragen werden, auf dieser Umgebung ist noch kein Modul vorhanden.
	Auf einer vorherigen Umgebung gibt es ein Modul, welches übertragen werden kann, um das nächste zu aktualisieren.	Das Modul der vorherigen Umgebung kann übertragen werden um dieses zu aktualisieren.
	Ein Modul auf einer Umgebung wurde entgegen des Entwicklungsprozesses gespeichert.	Das aktuelle Modul ist neuer als das Modul auf der vorherigen Umgebung oder die vorherige Umgebung wurde übersprungen.