



Abschlussprüfung Sommer 2016

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Parsen eines Schemas in eine Baumstruktur

und zergliedern eines Datenstroms anhand dieses Schemas

Abgabetermin: Nürnberg, den 15.05.2016

Prüfungsbewerber:

René Ederer
Identnummer 4135980
Steinmetzstr. 2
90431 Nürnberg



Ausbildungsbetrieb:

PHOENIX GROUP IT GMBH
Sportplatzstr. 30
90765 Fürth

Ausbildende:

Frau Birgit Günther

Projektbetreuer:

Herr Marco Kemmer

Inhaltsverzeichnis

Inhaltsverzeichnis	I
Abbildungsverzeichnis	IV
Tabellenverzeichnis	V
Listings	VI
Abkürzungsverzeichnis	VII
1 Projektdefinition	1
1.1 Auftraggeber	1
1.2 Projektumfeld	1
1.2.1 1920Schemas	1
1.2.2 Datenströme	2
1.3 Projektziel	2
1.4 Projektbegründung	2
1.5 Projektschnittstellen	3
2 Projektplanung	3
2.1 Projektphasen	3
2.2 Ressourcenplanung	3
2.3 Entwicklungsprozess	4
3 Analysephase	4
3.1 Ist-Analyse	4
3.2 Wirtschaftlichkeitsanalyse	4
3.2.1 „Make or Buy“-Entscheidung	4
3.2.2 Projektkosten	4
3.2.3 Amortisationsdauer	5
3.3 Anwendungsfälle	6
3.4 Qualitätsanforderungen	6
3.5 Lastenheft/Fachkonzept	6
3.6 Zwischenstand	6
4 Entwurfsphase - Zergliedern des Datenstroms	7
4.1 Zielplattform	7
4.2 Aufbau der Schemadateien	7
4.3 Architekturdesign	8
4.4 Entwurf der Benutzeroberfläche	8
4.5 Geschäftslogik	8

4.6	Maßnahmen zur Qualitätssicherung	9
4.7	Pflichtenheft/Datenverarbeitungskonzept	9
4.8	Zwischenstand	9
5	Implementierungsphase - Zergliedern des Datenstroms	9
5.1	Implementierung der Benutzeroberfläche	9
5.2	Implementierung der Geschäftslogik	10
5.2.1	Parse des Schemas in eine Baumstruktur	10
5.2.2	Grundschema der Methoden von AbstractNode	10
5.3	Zwischenstand	10
6	Entwurfsphase - Schema speichern und automatisch auswählen	10
6.1	Entwurf eines Bedienkonzeptes	10
6.2	Entwurf des XML-Schemas	11
7	Implementierung - Schema speichern	11
7.1	Erstellung des XSD-Datei	11
8	Abnahmephase	12
8.1	Abnahmetest	12
8.2	Zwischenstand	12
9	Einführungsphase	12
10	Dokumentation	12
10.1	Zwischenstand	13
11	Fazit	13
11.1	Soll-/Ist-Vergleich	13
11.2	Lessons Learned	13
11.3	Ausblick	14
	Eidesstattliche Erklärung	15
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	Datenbankmodell	v
A.6	Oberflächenentwürfe	vii
A.7	Screenshots der Anwendung	ix
A.8	Entwicklerdokumentation	xi
A.9	Testfall und sein Aufruf auf der Konsole	xiii

Inhaltsverzeichnis

A.10	Klasse: ComparedNaturalModuleInformation	xiv
A.11	Klassendiagramm	xvii
A.12	Benutzerdokumentation	xviii

Abbildungsverzeichnis

1	Use Case-Diagramm	iii
2	Datenbankmodell	vi
3	Liste der Module mit Filtermöglichkeiten	vii
4	Anzeige der Übersichtsseite einzelner Module	viii
5	Anzeige und Filterung der Module nach Tags	viii
6	Anzeige und Filterung der Module nach Tags	ix
7	Liste der Module mit Filtermöglichkeiten	x
8	Aufruf des Testfalls auf der Konsole	xiv
9	Klassendiagramm	xvii

Tabellenverzeichnis

1	Beispiel für ein 1920Schema	1
2	Zeitplanung	3
3	Kostenaufstellung	5
4	Zwischenstand nach der Analysephase	6
5	Beispiel für ein Schema	7
6	Zwischenstand nach der Entwurfsphase	9
7	Zwischenstand nach der Implementierungsphase	11
8	Zwischenstand nach der Abnahmephase	12
9	Zwischenstand nach der Dokumentation	13
10	Soll-/Ist-Vergleich	13

Listings

Listings/tests.php	xiii
Listings/cnmi.php	xiv

Abkürzungsverzeichnis

GUI	Graphical User Interface
Regex	Regular Expression
SVN	Subversion
UML	Unified Modeling Language
WPF	Windows Presentation Foundation

1 Projektdefinition

1.1 Auftraggeber

Die Phoenix Pharmahandel GmbH & Co. KG. und ihre Tochtergesellschaften sind europaweit unter dem Namen “Phoenix group” mit etwa 30000 Mitarbeitern im Pharmagroßhandel tätig. Ihre Haupttätigkeiten sind der Großhandel mit Pharmaprodukten und die Belieferung von Apotheken mit Medikamenten.

Auftraggeber des Projektes ist die Phoenix group IT GmbH. Sie hat etwa 200 Mitarbeiter und unterstützt die Phoenix group durch die Bereitstellung von IT-Dienstleistungen.

1.2 Projektumfeld

Die Phoenix group IT ist weiter unterteilt in die Abteilungen Inbound, Outbound und Warehouse. Das Projekt findet in der Abteilung Warehouse statt, in der überwiegend mit COBOL gearbeitet wird.

Phoenix verwendet ein firmeneigenes Dateiformat als Schnittstelle zu verschiedenen Programmen und Diensten. Es wird im Folgenden 1920Schema genannt und spielt eine zentrale Rolle für das Projekt.

1.2.1 1920Schemas

1920Schemas sind Textdateien, die einen Satz hierarchisch gegliederter Variablen beschreiben. Phoenix nutzt diese Schemas unter anderem als Schnittstelle, um Daten vom Mainframe zum Lagerrechner zu schicken, als Vorlage für Copybooks¹ und als Schnittstelle zu SSORT².

Beispiel für ein 1920Schema

Level	Name	Typ	Bytes	Wiederholzahl	Kommentar
01	Daten				Level und Name sind Pflichtangaben
03	Personendaten				Level gibt die Hierarchiestufe an
05	Vorname	C	5		fünf Bytes vom Typ char
05	Nachname	C	4		
R03	Gesamter-Name	C	9		redefiniert Personendaten
03	Bestellungen			2	Ein “Array”, Länge 2
05	Artikelnr	N	4		vier Byte lange Artikelnummer

Tabelle 1: Beispiel für ein 1920Schema

¹COBOL-Datei, in der eine Variablenstruktur definiert wird

²IBM-Programm, zeigt Copybooks an

1.2.2 Datenströme

Anhand von 1920Schemas werden Datenströme zergliedert und erhalten dadurch eine Bedeutung. Datenströme können prinzipiell beliebige Zeichen³ enthalten, nur die Anzahl muss ausreichen, um jeder Schema-Variablen einen Wert zuzuweisen. Das Schema aus Tabelle 1 beschreibt einen 17 Byte langen Datenstrom (5+4+2*4), der so aussehen könnte:

Beispiel für einen Datenstrom

VN~~~NN~~12345678

1.3 Projektziel

Es soll ein Programm geschrieben werden, das nach Eingabe von Datenstrom und 1920Schema den Datenstrom dem Schema entsprechend zergliedert und anzeigt.

Gewünschte Ausgabe (mit den Beispielwerten der Abschnitte 1.2.1 und 1.2.2)

01 Daten

03 Personendaten

05 Vorname=VN~~~

05 Nachname=NN~~

R03 Gesamter-Name=VN~~~NN~~

03 Bestellungen(1)

05 Artikelnr=1234

03 Bestellungen(2)

05 Artikelnr=5678

1.4 Projektbegründung

Bei Kundenreklamationen, Änderungen an Programmen und Neuentwicklungen stehen die Warehouse-Programmierer oft vor den Problemen:

- Wert einer Schemadatei-Variablen in einem Datenstrom finden.
- Datenstrom-Bytes einer Schemadatei-Variablen zuordnen.

Gegenwärtig zählen die Programmierer die passende Anzahl von Bytes in Schema und Datenstrom ab, einige erfahrene kennen die wichtigsten Schemadateien auch teilweise auswendig.

³Ein Byte entspricht einem Zeichen. Die Angaben in 1920Schemas haben die Einheit Byte, im Zusammenhang mit dem Datenstrom ist der Begriff Zeichen aber manchmal anschaulicher. Eine Unterscheidung der beiden Begriffe ist für das Projekt unwesentlich.

1.5 Projektschnittstellen

Benutzer des Programms sind die Programmierer der Phoenix Group IT GmbH, hauptsächlich die aus der Abteilung Warehouse. 1920Parser soll nicht unmittelbar mit anderen Systemen interagieren. Vorgesehen ist, dass die Benutzer die notwendigen Angaben in eine Eingabeaske hineinkopieren.

Projektgenehmigung und die Bereitstellung von Ressourcen erfolgt durch die Ausbildende Frau Birgit Günther, die Projektbetreuung und die Abnahme des Programms durch Herrn Marco Kemmer. Herr Kemmer arbeitet in der Abteilung Warehouse als COBOL-Entwickler, er kommuniziert die Kundenwünsche und will das Programm in Zukunft auch selbst nutzen.

2 Projektplanung

2.1 Projektphasen

Für das Projekt standen 70 Stunden zur Verfügung. Es fand im Zeitraum vom 11.04.2016 - 15.05.2016 statt.

Beispiel Tabelle 2 zeigt ein Beispiel für eine grobe Zeitplanung.

Projektphase	Geplante Zeit
Analysephase	6 h
Entwurfsphase	8 h
Implementierungsphase	44 h
Abnahmetest der Fachabteilung	2 h
Erstellen der Dokumentation	10 h
Gesamt	70 h

Tabelle 2: Zeitplanung

Eine detailliertere Zeitplanung findet sich im Anhang [A.1: Detaillierte Zeitplanung](#) auf Seite i.

2.2 Ressourcenplanung

Windows 7, Visual Studio Professional 2010, Microsoft Visio 2010, TexMaker, XMLSpy 2007, OpenText HostExplorer, Büro, Büro mit PC mit Verbindung zum Mainframe, Internetverbindung, Projektbetreuer, Strom, Kaffee

2.3 Entwicklungsprozess

Es wurde ein agiler Entwicklungsprozess angewendet, der an Extreme Programming angelehnt war. Der im Projekt angewandte Prozess beinhaltete die Extreme Programming-Praktiken, testgetriebene Entwicklung, häufige Kundeneinbeziehung, häufiges Refaktorisieren, kurze Iterationen und einfaches Design.

3 Analysephase

3.1 Ist-Analyse

Die Ist-Analyse fand durch Befragung von Herrn Kemmer statt. Herr Kemmer ist sowohl Projektbetreuer als auch potentieller Benutzer und Auftraggeber des Projektes. Der Autor bereitete eine Liste mit Fragen vor.

Bei der Befragung stellte sich heraus, dass die Entwickler der Abteilung Lager bei Kundenreklamationen und Programmänderungen oft Logdateien analysieren müssen, welchen Umständen genau das Programm benötigt wird, woher die Schemas stammten, Datenstrom verbrachten, wofür sie verwendet werden, wie oft sie verwendet werden, von wie vielen, wie hoch er die Zeitersparnis schätzt, wie die Entwickler bisher arbeiten ohne das Programm. Der Autor erfragte genau die Anforderungen und ordnete sie in Muss-, Soll-, und Kann-Kriterien.

3.2 Wirtschaftlichkeitsanalyse

Das Programm soll den Entwicklern das bisher notwendige, fehleranfällige Abzählen von Zeichen in Schema und Datenstrom abnehmen. Das Projekt verspricht dadurch nicht nur, den Entwicklern Zeit zu sparen, sondern auch Zählfehler wirkungsvoll zu verhindern.

3.2.1 „Make or Buy“-Entscheidung

Die Anforderungen sind so speziell, dass fast auszuschließen ist, dass es außerhalb von Phoenix ein Programm gibt, das die Anforderungen erfüllt. Zu bemerken ist aber, dass der Mainframe von Phoenix mit Schemadateien arbeitet und vielleicht Ähnliches leistet, was das zu erstellende Programm leisten soll. Der Autor fragte deshalb bei seinem Auftraggeber nach, ob der Mainframe-Quelltext anzusehen ist. Herr Kemmer antwortete, dass das zwar möglich, aber sehr aufwändig sei. Da nicht sicher war, dass die Ansicht des Quelltextes die Programmerstellung erleichtern würde, wurde auf eine weitere Verfolgung dieser Idee verzichtet. Es wurde entschieden, das Programm selbst neu zu schreiben.

3.2.2 Projektkosten

Im Rahmen des Projektes fallen Kosten für Entwicklung und Abnahmetest an.

Beispielrechnung (verkürzt) Die Kosten für die Durchführung des Projekts setzen sich sowohl aus Personal-, als auch aus Ressourcenkosten zusammen. Der Projektersteller ist Umschüler und erhält deshalb von seinem Ausbildungsbetrieb keine Vergütung.

$$7,7 \text{ h/Tag} \cdot 220 \text{ Tage/Jahr} = 1694 \text{ h/Jahr} \quad (1)$$

$$0 \text{ €/Monat} \cdot 13,3 \text{ Monate/Jahr} = 0 \text{ €/Jahr} \quad (2)$$

$$\frac{0 \text{ €/Jahr}}{1694 \text{ h/Jahr}} = 0,00 \text{ €/h} \quad (3)$$

Dadurch ergibt sich also ein Stundenlohn von 0,00 €. Die Durchführungszeit des Projekts beträgt 70 Stunden. Für die Nutzung von Ressourcen⁴ wird ein pauschaler Stundensatz von 12 € angenommen. Für die anderen Mitarbeiter wird pauschal ein Stundenlohn von 23 € angenommen. Eine Aufstellung der Kosten befindet sich in Tabelle 3 und sie betragen insgesamt 1015,00 €.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	70 h	0,00 € + 12 € = 12,00 €	840,00 €
Fachgespräch	3 h	23 € + 12 € = 35 €	105,00 €
Abnahmetest	2 h	23 € + 12 € = 35 €	70,00 €
			1015,00 €

Tabelle 3: Kostenaufstellung

3.2.3 Amortisationsdauer

Es wird davon ausgegangen, dass ausschließlich Entwickler der Abteilung Lager das Programm nutzen werden. Nach Einschätzung von Herrn Kemmer arbeitet die Hälfte der 20 Entwickler regelmäßig mit Schemadateien und dass das Programm jedem täglich 10 Minuten einsparen kann. Bei einer Einsparung von 10 Minuten am Tag für 10 Entwickler an 220 Arbeitstagen im Jahr ergibt sich eine gesamte Zeiteinsparung von

$$10 \cdot 220 \text{ Tage/Jahr} \cdot 10 \text{ min/Tag} = 22000 \text{ min/Jahr} \approx 366,67 \text{ h/Jahr} \quad (4)$$

Dadurch ergibt sich eine jährliche Einsparung von

$$366,67 \text{ h} \cdot (23 + 12) \text{ €/h} = 12833,45 \text{ €} \quad (5)$$

Die Amortisationsdauer beträgt also $\frac{1015,00 \text{ €}}{12833,45 \text{ €/Jahr}} \approx 0,08 \text{ Jahre} \approx 1 \text{ Monat}$.

⁴Räumlichkeiten, Arbeitsplatzrechner etc.

3.3 Anwendungsfälle

Der mit Abstand wichtigste Anwendungsfall ist, dass das Programm nach Angabe von Datenstrom und Schema den Datenstrom entsprechend dem Schema zergliedert anzeigt. Der Kunde nannte noch einige Wünsche zur Funktionalität. Ein Usecase-Diagramm ist im Anhang beigelegt.

3.4 Qualitätsanforderungen

Schemas müssen frei angebbbar sein und Datenströme richtig zergliedert werden. Beim Abnahmetest soll dies anhand der 5 wichtigsten Schemadateien überprüft werden. Performance ist ziemlich egal, das Programm soll aber flüssig benutzbar sein (Zergliederung von Datenstrom und Anzeige in unter 5 Sekunden). Weil die Benutzer Profis sind, ist es nicht unbedingt notwendig, für alles eine Eingabemaske bereitzustellen. Es genügt auch, wenn eine Einstellung durch Editieren einer Konfigurations-Datei geändert werden kann.

3.5 Lastenheft/Fachkonzept

- Auszüge aus dem Lastenheft/Fachkonzept, wenn es im Rahmen des Projekts erstellt wurde.
- Mögliche Inhalte: Funktionen des Programms (Muss/Soll/Wunsch), User Stories, Benutzerrollen

Beispiel Ein Beispiel für ein Lastenheft findet sich im Anhang A.2: Lastenheft (Auszug) auf Seite ii.

3.6 Zwischenstand

Tabelle 4 zeigt den Zwischenstand nach der Analysephase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Analyse des Ist-Zustands	3 h	2 h	-1 h
2. Wirtschaftlichkeitsprüfung und Amortisationsrechnung	1 h	2 h	+1 h
3. Erstellen des Lastenhefts	2 h	2 h	

Tabelle 4: Zwischenstand nach der Analysephase

4 Entwurfsphase - Zergliedern des Datenstroms

4.1 Zielplattform

Das Programm soll auf den Entwicklerrechnern der Phoenix unter Windows 7 laufen. Die Wahl der Programmiersprache wurde zunächst auf die bei Phoenix eingesetzten Sprachen COBOL, C++ und C# eingegrenzt. COBOL schied als Programmiersprache für ein Windows-Tool aus, aber C++ mit Qt und C# waren beide geeignet. Die hohe Performance, die C++ verspricht, wurde für 1920Parser aber nicht wirklich benötigt. Die Wahl fiel auf C# aufgrund von dessen Garbage Collection, Linq und gutem GUI-Designer.

4.2 Aufbau der Schemadateien

Zur Analyse des Aufbaus der 1920Schemas fragte der Autor Herrn Kemmer nach den Namen der 10 wichtigsten Schemadateien und ludt diese unter Verwendung von OpenText HostExplorer 2014 vom Mainframe herunter.

Ein 1920Schema kann zum Beispiel so aussehen:

Level	Name	Typ	Bytes	Wiederholzahl	Kommentar
01	Daten				Level und Name sind Pflichtangaben
03	Personendaten				Level gibt die Hierarchiestufe an
05	Vorname	C	5		fünf Bytes vom Typ char
05	Nachname	C	4		
R03	Gesamter-Name	C	9		redefiniert Personendaten
03	Bestellungen			2	Ein "Array", Länge 2
05	Artikelnr	N	4		vier Byte lange Artikelnummer

Tabelle 5: Beispiel für ein Schema

Level und Name sind Pflichtangaben, alle anderen Felder dürfen leer sein. Typ und Byteanzahl treten nur zusammen auf, solche Variablen sind Wertvariablen (sie definieren Bytes aus dem Datenstrom). Variablen ohne Typ und Bytezahl sind Gruppenvariablen. Zusätzlich zu diesen Variablenzeilen enthalten 1920Schemas Zeilen mit Metainformationen zum Schema. Die Metainformationen sind für 1920Parser nicht relevant und werden ignoriert.

Es ergaben sich unerwartet Schwierigkeiten beim Feld Kommentar. In der Schemadatei IOVK92 gab es eine Variable, deren Kommentar-Feld mit "0" begann. Es musste eine Regel festgelegt werden, damit diese 0 nicht als Wiederholzahl interpretiert wird, sondern als Kommentar-Beginn. Es gab Ideen, 0 als Wiederholzahl zu verbieten oder die untereinander-stehende Anordnung der Variablenfelder auszunutzen. Falsche Annahmen hätten zu Fehlfunktion des Programms geführt, es war deshalb nötig, Rücksprache

mit dem Auftraggeber zu halten. Auf Herrn Kemmers Vorschlag und nach erneuter Prüfung der Schema-dateien wurde festgelegt, dass ein Kommentar mit mindestens 2 Leerzeichen vom vorhergehenden Feld getrennt sein muss.

4.3 Architekturdesign

Während der Projekterstellung wurde auf Kohärenz der Klassen geachtet und gegebenenfalls refaktori-riert (zum Beispiel stand die Funktionalität zum Lesen und Schreiben der XML-Config ursprünglich auch in der Klasse Schema, wurde später aber in die neu erschaffene Klasse SchemaManager ausge-lagert). Im Projektverlauf entstand eine 3-Schichten-Architektur mit den Klassen 1920ParserView und SaveSchemaView als Teil der Präsentationsschicht, den Klassen AbstractNode, GroupNode, ValueNode und Schema auf Logikschicht und SchemaManager und SchemaConfig als Teil der Datenhaltungsschicht. Klassen der Präsentationsschicht und Klassen Datenhaltungsschicht kennen sich nicht und deren Objekte kommunizieren auch nicht direkt miteinander.

4.4 Entwurf der Benutzeroberfläche

Der Kunde hatte keine Vorgaben bezüglich der Benutzeroberfläche gemacht. Nachdem als Programmier-sprache C# feststand, kamen für die Graphical User Interface (GUI) nur Plattformen aus dem .NET-Framework in Betracht. Da Benutzer Schemas speichern können sollen, wäre ein Webinterface (ASP.NET) mit zusätzlichem Aufwand verbunden gewesen (Benutzerverwaltung/Datenbank). Windows Presentation Foundation (WPF) und Winforms waren beide geeignet, der Autor entschied sich für Winforms, weil er damit mehr Erfahrung hatte.

4.5 Geschäftslogik

Die hierarchische Aufbau von 1920Schemas lässt sich gut mit einer rekursiven Baumstruktur im Pro-gramm abbilden. Für die Baumstruktur wurde die abstrakte Klasse AbstractNode entworfen. Von dieser Klasse erben die Klassen GroupNode und ValuNode, die Gruppen- und WerteVariablen darstellen. Group-Node soll ein Attribut children vom Typ List<AbstractNode> erhalten, mit dem es auf seine KindKnoten verweist. GroupNodes erlauben durch ihre rekursive Definition eine beliebig tiefe Verschachtelung von AbstractNodes.

Ein Klassendiagramm, welches die Klassen der Anwendung und deren Beziehungen untereinander dar-stellt kann im Anhang [A.11: Klassendiagramm](#) auf Seite xvii eingesehen werden.

4.6 Maßnahmen zur Qualitätssicherung

Das korrekte Parsen von 1920Schemas und die richtige Zergliederung des Datenstroms sind zentral für 1920Parser. Um sicherzugehen, dass dieser Programmteil funktioniert, wurden daher die Methoden der Klassen Schema, Abstract-, Group- und ValueNode testgetrieben entwickelt.

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 3.5: [Lastenheft/Fachkonzept](#)) aufbauende Pflichtenheft ist im Anhang [A.4: Pflichtenheft \(Auszug\)](#) auf Seite iii zu finden.

4.8 Zwischenstand

Tabelle 6 zeigt den Zwischenstand nach der Entwurfsphase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Analyse des Aufbaus der Schemadateien	2 h	2 h	
2. Entwurf der Klassen für die Baumstruktur	4 h	2 h	-2 h
3. Erstellen des Pflichtenhefts	2 h	2 h	

Tabelle 6: Zwischenstand nach der Entwurfsphase

5 Implementierungsphase - Zergliedern des Datenstroms

Die Implementierung begann mit dem Anlegen eines neuen SVN-Repositories mit der vorgegebenen Verzeichnisstruktur (branch, tag, trunk) und dem Erstellen einer neuen Solution in Microsoft Visual Studio Professional 2010 mit einem C# Winforms Projekt. Der Solution wurde ein NUnit-Testprojekt für die Unit-Tests hinzugefügt.

5.1 Implementierung der Benutzeroberfläche

Visual Studio erstellte mit Neuanlage des Winform-Projekts automatisch eine Form-Klasse, die ein Fenster darstellt. Die GUI wurde so einfach wie möglich gehalten: 3 Textfelder mit Überschrift, jeweils für Datenstrom, Schema und Ergebnis. Es wurde noch ein Button zum Starten der Verarbeitung eingefügt. Dieser wurde im Laufe der Implementierung entfernt und die Verarbeitung bei jeder Eingabe in Schema- oder Datenstrom-Textfeld gestartet.

5.2 Implementierung der Geschäftslogik

5.2.1 Parsen des Schemas in eine Baumstruktur

Parsen einer Variablenzeile zu einer AbstractNode Jede Variablenzeile im Schema wird zu einer Group- oder ValueNode. Dazu wurde in der Klasse Schema die Methode ParseLine geschrieben. Die Methode splittet die Angaben einer Variablenzeile mit Hilfe einer Regular Expression (**Regex**) in ihre Felder auf. Die **Regex** wurde auf <https://regex101.com> angepasst und getestet, bis sie richtig funktionierte.

Eine Schwierigkeit des Algorithmus war, dass zu jedem Zeitpunkt in mehrere Richtungen gegangen werden muss. Zum einen muss die nächste Zeile des Schemas in einen Knoten verwandelt werden. Ein Knoten kann sich aber wiederholen. Hier hat der Autor einige Zeit verbraucht mit rekursiven Ideen.

Zur Lösung des Problems führte die Idee, einen Knoten nach dessen Erstellung zu Kopieren.

Der Algorithmus zum Parsen des Schemas nutzt einen Stack. Der Stack enthält zu jedem Zeitpunkt die Vorfahren des gerade bearbeiteten Knotens, mit seinem Elternknoten oben und dem Wurzelknoten unten.

5.2.2 Grundschema der Methoden von AbstractNode

Group- und ValueNode erben von AbstractNode Methoden. Die Implementierung all dieser Methoden ähnelt sich, sie folgt für Group- und ValueNodes immer diesem Prinzip: ValueNode macht eine Aktion und gibt danach einen Wert zurück. GroupNode macht eine Aktion und ruft danach für jedes seiner Kinder die gleiche Methode erneut auf (Rekursion). Aus den Rückgabewerten der Kinder wird ein Wert akkumuliert und dieser zurückgegeben. Die von einer GroupNode angestoßenen Rekursionen enden bei ValueNodes und GroupNodes ohne Kinderknoten, die Abbruchbedingung der Rekursion ist polymorph. Es ist kein Zufall, dass alle Methoden rekursiv sind, denn auch die Klassenstruktur ist rekursiv (GroupNodes können auf GroupNodes verweisen).

5.3 Zwischenstand

Tabelle 7 zeigt den Zwischenstand nach der Implementierungsphase.

6 Entwurfsphase - Schema speichern und automatisch auswählen

6.1 Entwurf eines Bedienkonzeptes

Um die GUI übersichtlich zu halten, und weil in Zukunft eventuell noch Auswahlmöglichkeiten für andere Anwendungsfälle notwendig werden könnten, wurde entschieden, ein Menü zu erstellen. Bei Auswahl des

Vorgang	Geplant	Tatsächlich	Differenz
1. Implementierung der Klassen	1 h	0,5	-0,5 h
2. Schreiben einer Methode zum Vergleichen von Nodes	0 h	2 h	+2 h
3. Erstellen von Tests	5 h	9 h	+4 h
4. Entwickeln einer Regular Expression zum Parsen von Schemazeilen	1 h	3 h	+2 h
5. Schreiben einer Methode zum Kopieren von Nodes	0 h	1 h	+1 h
6. Parsen der Schema-Datei in eine Baumstruktur	14 h	17 h	+3 h
7. Durchlaufen des Baumes und Zuweisen von Werten	12 h	2 h	-10 h
8. Ausgabe des Baumes als String	5 h	1 h	-4 h
9. Erstellen der Benutzeroberfläche	6 h	1 h	-5 h

Tabelle 7: Zwischenstand nach der Implementierungsphase

Menüpunktes "Schema speichern" soll sich ein Dialog öffnen, in dem der Name des Schemas und ein Textfeld zur Angabe eines Datenstrom-Identifikationstextes⁵ eingegeben werden. Beim Klicken auf den Button "Schema speichern" wird ein neuer Eintrag in die XML-Datei vorgenommen und eine Textdatei mit dem Inhalt des Schema-Textfeldes im Programm-Unterordner "schemas" gespeichert. Neben der Überschrift des Schema-Textfeldes soll eine Combobox eingefügt werden, die die Dateien des schemas-Ordners anzeigt, und über die ein Schema ausgewählt werden kann.

6.2 Entwurf des XML-Schemas

Es gibt ein Microsoft-Tool namens xsd.exe, das aus XSD-Schemas C#-Klassen generieren kann. Um das auszunutzen wurde entschieden, die Einstellungen, die notwendigen Einstellungen als XML-Datei vorzunehmen. Das hatte außerdem den Vorteil, dass die Benutzer die Einstellungen direkt editieren können.

7 Implementierung - Schema speichern

Die Implementierung dieses Anwendungsfalls war eher Routine.

7.1 Erstellung des XSD-Datei

Es wurde eine xml-Datei (schemaConfig.xml) mit den benötigten Angaben und der gewünschten Struktur erstellt. Mit xsd.exe wurde daraus eine XSD-Datei erzeugt, und mit erneutem Aufruf von xsd.exe aus dieser die C#-Klasse SchemaConfig. Diese wird verwendet um die der Xml-Datei zu deserialisieren.

⁵z. B. der Datenstrom enthält an Stelle 3 den Text abc

8 Abnahmephase

8.1 Abnahmetest

Der Abnahmetest erfolgte durch den Projektbetreuer Herrn Kemmer. Herr Kemmer ließ sich die erstellten Unit-Tests zeigen überprüfte, dass alle Tests erfolgreich waren. Danach ludt er die 5 am Häufigsten verwendeten Schemas zusammen mit passenden Datenströmen vom Mainframe. 1920Parser zeigte alle getesteten Schemas richtig an und zergliederte die Datenströme in gewünschter Weise. Im Anschluss testete Herr Kemmer die Funktionalität zum Speichern und zur automatischen Auswahl von Schemas. Nach Löschung der XML-Konfigurationsdatei trat bei der Auswahl des Menüpunktes "Config editieren" eine FileNotFoundException auf. Diese wurde behoben, danach testete Herr Kemmer noch einmal.

8.2 Zwischenstand

Tabelle 8 zeigt den Zwischenstand nach der Abnahmephase.

Vorgang	Geplant	Tatsächlich	Differenz
1. Abnahmetest	2 h	1,5 h	-0,5 h
2. Nachkorrektur und erneute Überprüfung	0 h	0,5 h	+0,5 h

Tabelle 8: Zwischenstand nach der Abnahmephase

9 Einführungsphase

Die Einführung von 1920Parser wurde von Herrn Kemmer übernommen. Das Programm soll im Intranet bereitgestellt werden, da die Benutzer nicht alle Subversion (SVN) installiert haben.

10 Dokumentation

Für 1920Parser wurde eine Benutzerdokumentation und eine Entwicklerdokumentation geschrieben.

Beispiel Ein Ausschnitt aus der erstellten Benutzerdokumentation befindet sich im Anhang A.12: Benutzerdokumentation auf Seite xviii. Die Entwicklerdokumentation wurde mittels PHPDoc⁶ automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse findet sich im Anhang A.8: Entwicklerdokumentation auf Seite xi.

⁶Vgl. ?

10.1 Zwischenstand

Tabelle 9 zeigt den Zwischenstand nach der Dokumentation.

Vorgang	Geplant	Tatsächlich	Differenz
1. Erstellen der Benutzerdokumentation	1 h	1 h	
2. Erstellen der Projektdokumentation	7 h	7 h	
3. Erstellen der Entwicklerdokumentation	2 h	2 h	

Tabelle 9: Zwischenstand nach der Dokumentation

11 Fazit

11.1 Soll-/Ist-Vergleich

Wie gewünscht können im Programm Datenstrom und Schema frei angegeben werden. Das Ergebnis wird richtig angezeigt. Insbesondere das Zuweisen der Werte aus dem Datenstrom und die Ausgabe des Baumes als String nahmen deutlich weniger Zeit in Anspruch, als ursprünglich geplant. So konnte noch der Anwendungsfall “Schema speichern” im Rahmen des Projektes umgesetzt werden. Herr Kemmer ist mit dem Programm sehr zufrieden.

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 10: Soll-/Ist-Vergleich

11.2 Lessons Learned

RichTextBox ist besser als TextBox. Interessant war, wie sich der Projektumfang erweitert hatte (ursprünglich war der Vorschlag, dass nur ein bestimmtes Schema zergliedert werden soll) Rekursion ist nicht gut darstellbar mit UML. Für Polymorphie gilt das selbe. Stacks sind super. Rekursion vereinfacht manche Aufgaben enorm. Bemerkenswert, wie sehr sich die Anforderungen ausgeweitet haben. Bemerkenswert war, dass sich der Projektumfang stark erhöhte (ursprünglich wurde mir der Projektvorschlag gemacht, nur die Schemadatei VK60 zu zergliedern).

11.3 Ausblick

Im Rahmen des Projektes wurden nicht alle Wunschkriterien erfüllt. Der Autor hofft aber, dass diese Aufgabe vielleicht an nachfolgende Auszubildende übergeben wird und sich 1920Parser noch weiterentwickelt. Vielleicht wird einmal die Größe der Datenströme geändert, dann könnte der Algorithmus von 1920Parser verwendet werden.

Eidesstattliche Erklärung

Ich, René Ederer, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

Parsen eines Schemas in eine Baumstruktur – und zergliedern eines Datenstroms anhand dieses Schemas

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Nürnberg, den 15.05.2016

RENÉ EDERER

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 geparsten 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>.

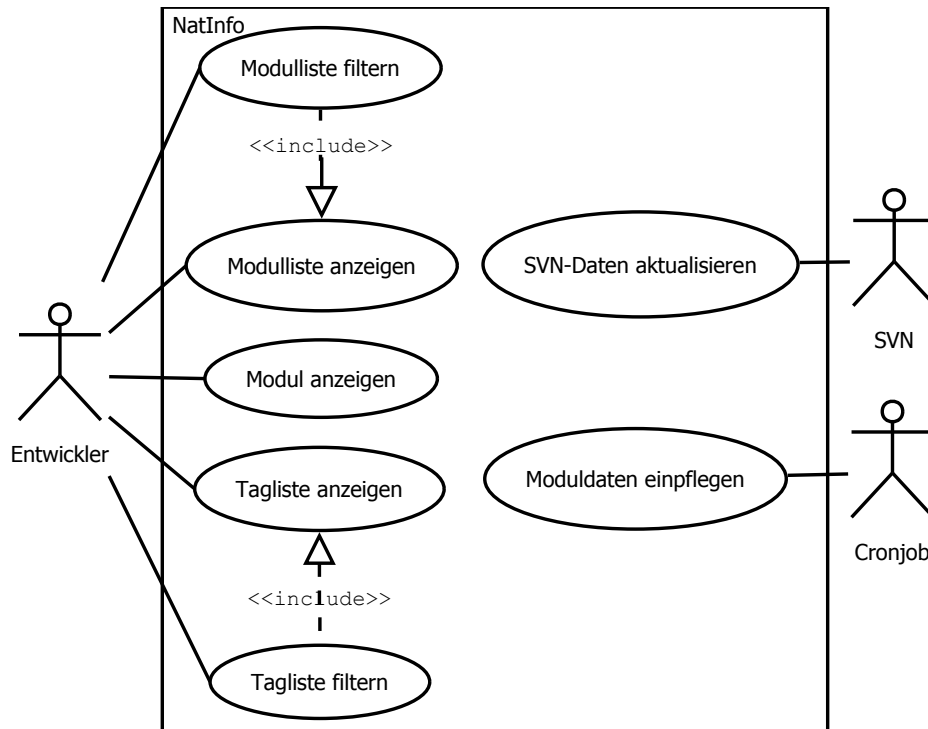


Abbildung 1: Use Case-Diagramm

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!** (**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 Datenbankmodell

ER-Modelle kann man auch direkt mit \LaTeX zeichnen, siehe z.B. <http://www.texample.net/tikz/examples/entity-relationship-diagram/>.

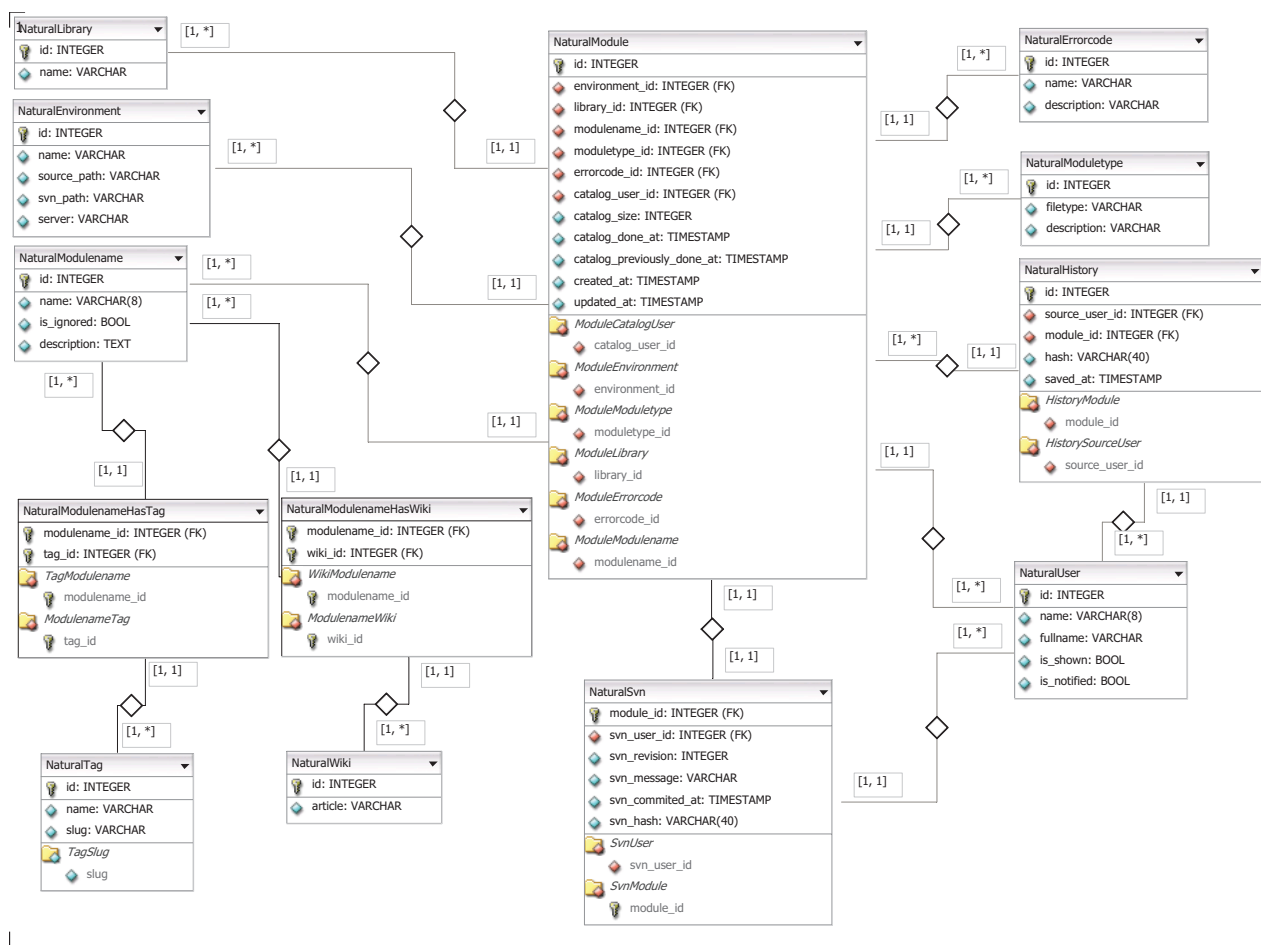


Abbildung 2: Datenbankmodell

A.6 Oberflächenentwürfe

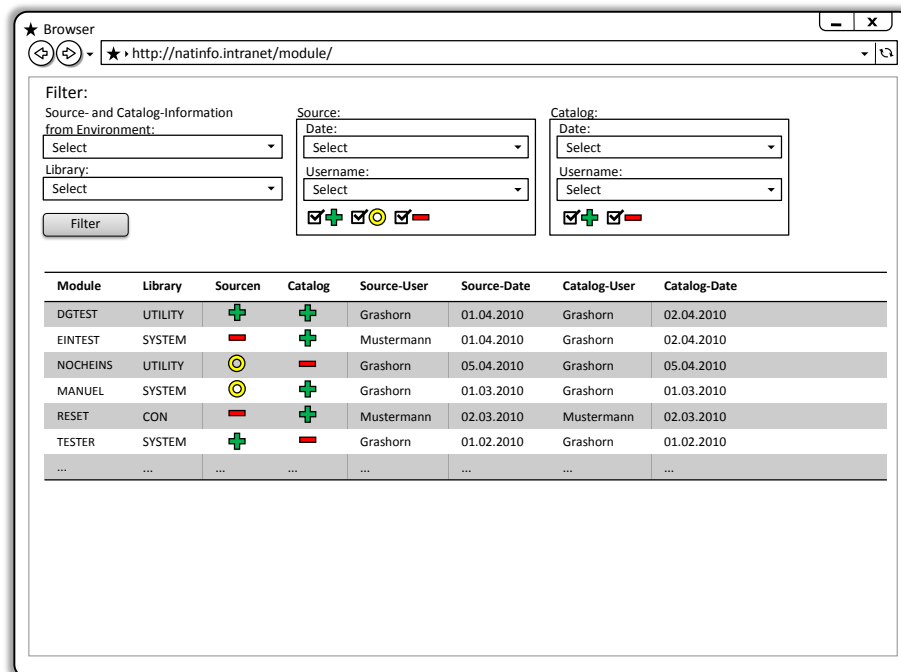


Abbildung 3: Liste der Module mit Filtermöglichkeiten

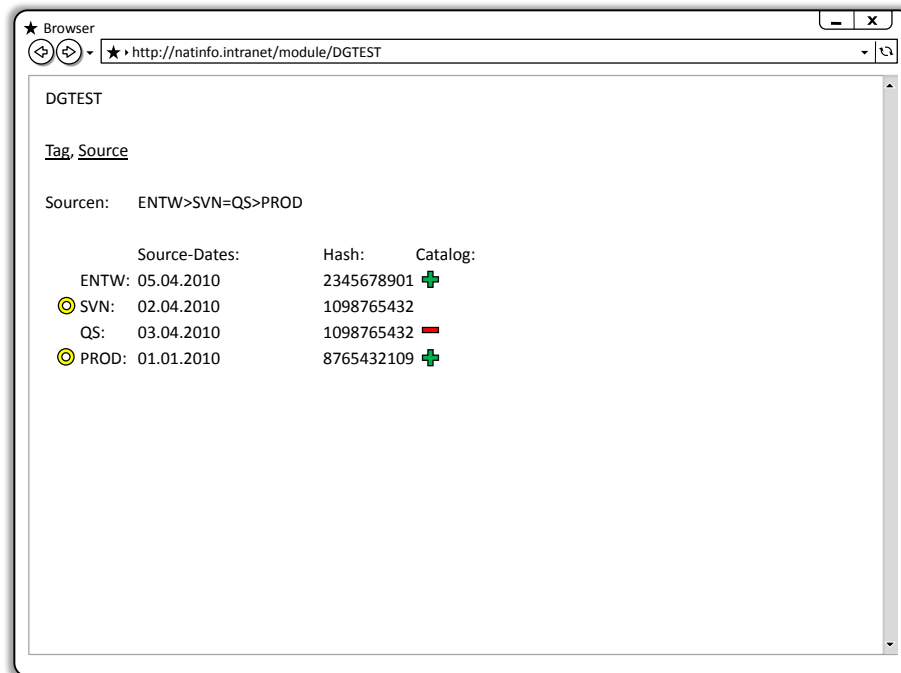


Abbildung 4: Anzeige der Übersichtsseite einzelner Module

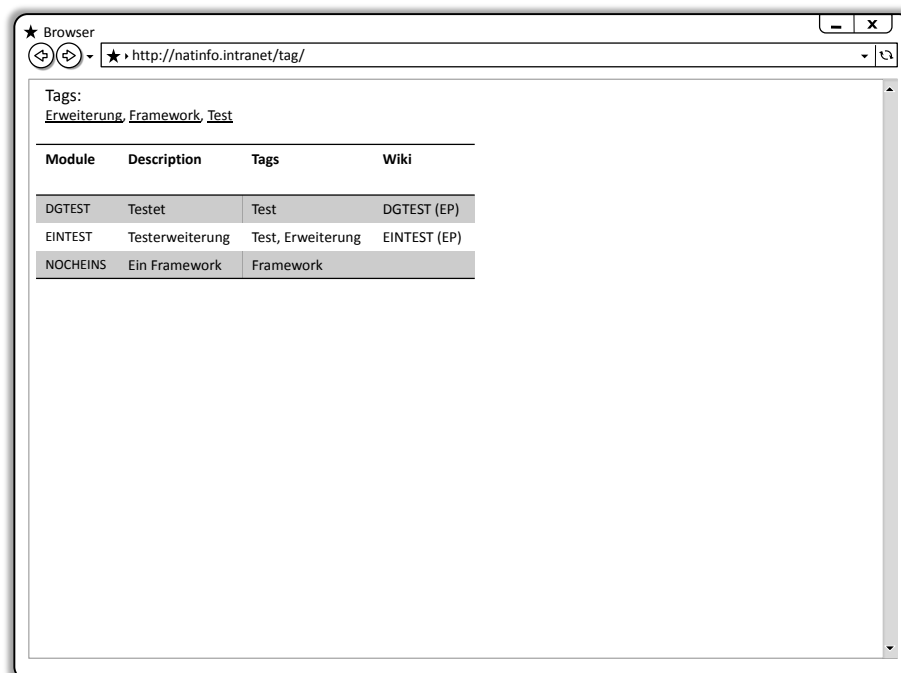


Abbildung 5: Anzeige und Filterung der Module nach Tags

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 6: 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 7: 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:

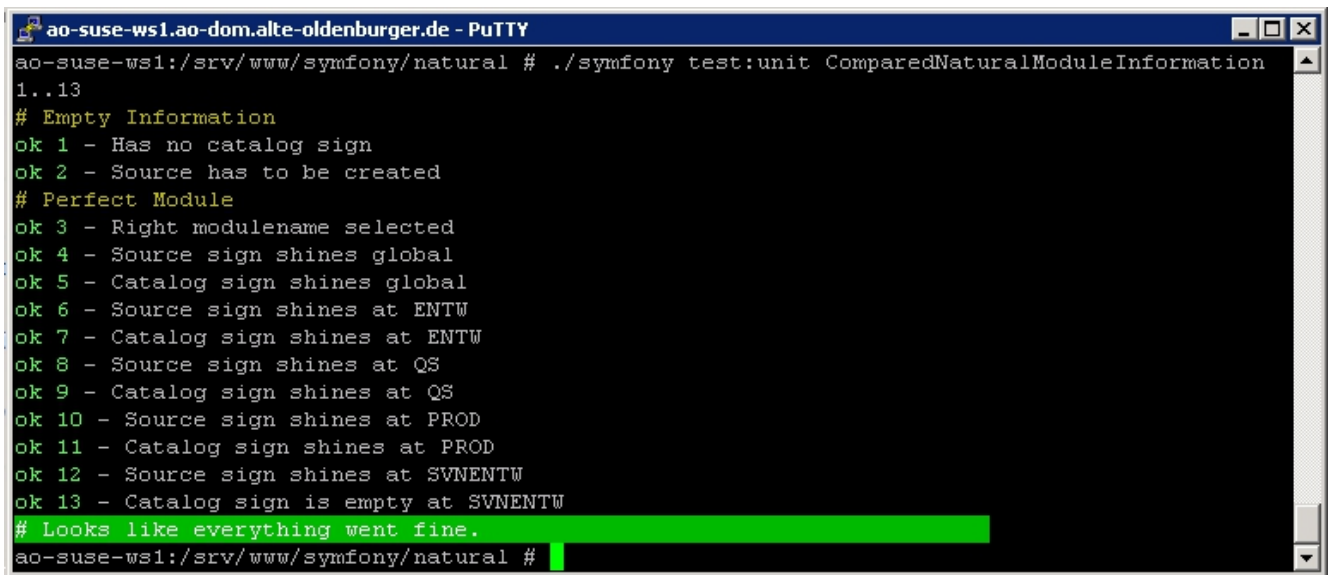
access: public

[\[Top \]](#)

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(NaturalmodulePeer::NAME, 'SMTAB');
14 $moduleName = NaturalmodulePeer::doSelectOne($criteria);
15 $t->is($moduleName->getName(), 'SMTAB', 'Right module name 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 8: 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     {
```

A Anhang

```
25     $this->allocateModulesToEnvironments($naturalInformations);
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         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>.

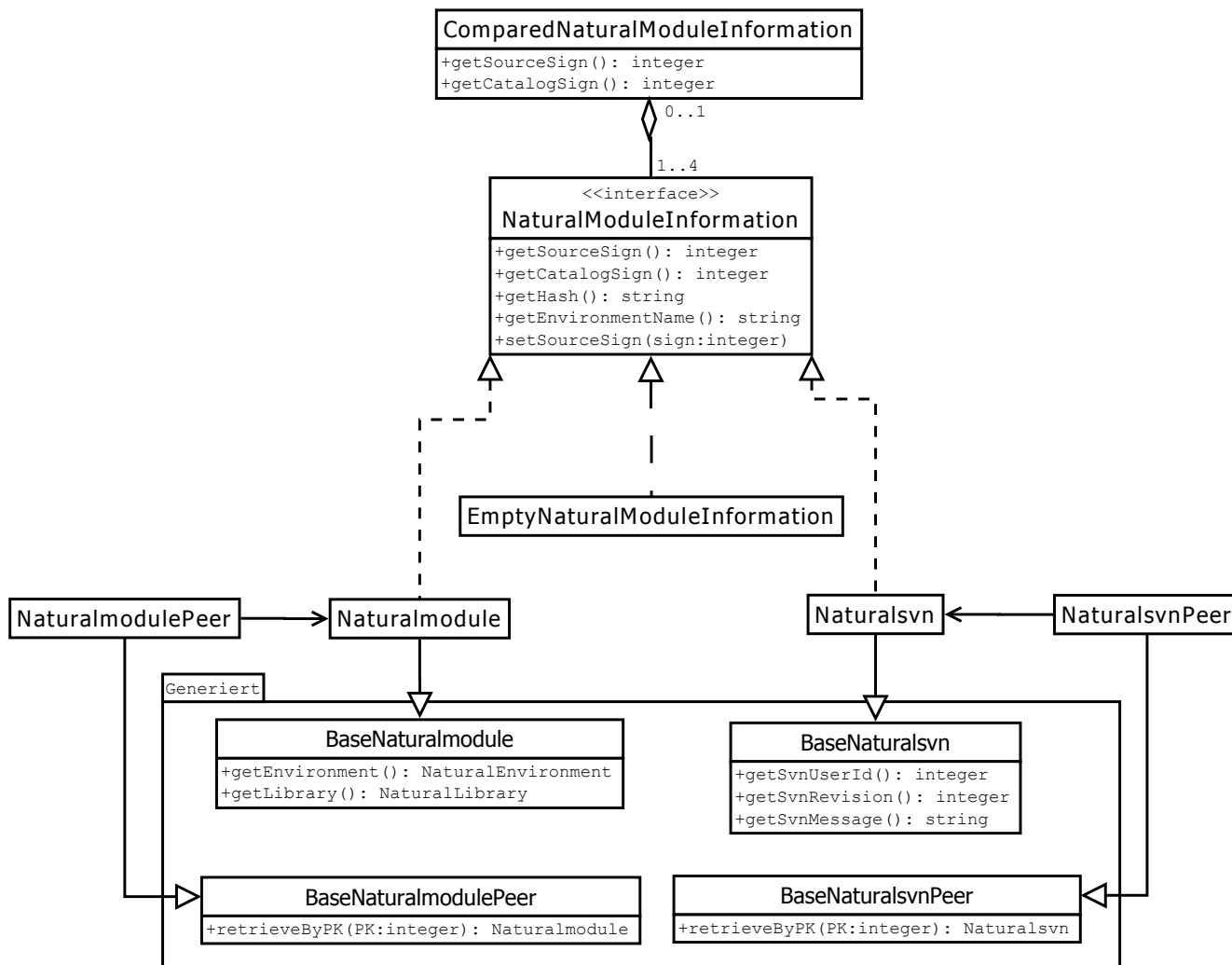







Abbildung 9: Klassendiagramm

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.