



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
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 Einleitung	1
1.1 Auftraggeber	1
1.2 Projektumfeld	1
1.3 Projektziel	1
1.4 Projektbegründung	1
1.5 Projektschnittstellen	2
2 Projektplanung	2
2.1 Projektphasen	2
2.2 Ressourcenplanung	3
2.3 Entwicklungsprozess	3
3 Analysephase	3
3.1 Ist-Analyse	3
3.2 Wirtschaftlichkeitsanalyse	3
3.2.1 „Make or Buy“-Entscheidung	4
3.2.2 Projektkosten	4
3.2.3 Amortisationsdauer	5
3.3 Nutzwertanalyse	5
3.4 Anwendungsfälle	5
3.5 Qualitätsanforderungen	5
3.6 Lastenheft/Fachkonzept	6
3.7 Zwischenstand	6
4 Entwurfsphase	6
4.1 Zielplattform	6
4.2 Aufbau der Schemadateien	6
4.3 Architekturdesign	8
4.4 Entwurf der Benutzeroberfläche	8
4.4.1 Erste Iteration: Datenstrom zergliedert anzeigen	8
4.4.2 Zweite Iteration: Schema speichern	8

Inhaltsverzeichnis

4.5	XML-Datenmodell	9
4.6	Geschäftslogik	9
4.7	Maßnahmen zur Qualitätssicherung	9
4.8	Pflichtenheft/Datenverarbeitungskonzept	10
4.9	Zwischenstand	10
5	Implementierungsphase	10
5.1	Implementierung der Datenstrukturen	10
5.2	Implementierung der Benutzeroberfläche	11
5.3	Implementierung der Geschäftslogik	11
5.3.1	Grundschemata der rekursiven Methoden von AbstractNode/GroupNode	11
5.3.2	Parzen des Schemas in eine Baumstruktur	12
5.3.3	Zuweisen der Werte aus dem Datenstrom	12
5.4	Zwischenstand	13
6	Abnahme phase	13
6.1	Zwischenstand	13
7	Einführungsphase	14
7.1	Zwischenstand	14
8	Dokumentation	14
8.1	Zwischenstand	14
9	Fazit	15
9.1	Soll-/Ist-Vergleich	15
9.2	Lessons Learned	16
9.3	Ausblick	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	Datenbankmodell	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	Prozess des Einlesens eines Moduls	9
2	Use Case-Diagramm	iii
3	Datenbankmodell	v
4	Liste der Module mit Filtermöglichkeiten	vi
5	Anzeige der Übersichtsseite einzelner Module	vii
6	Anzeige und Filterung der Module nach Tags	vii
7	Anzeige und Filterung der Module nach Tags	viii
8	Liste der Module mit Filtermöglichkeiten	ix
9	Aufruf des Testfalls auf der Konsole	xiii
10	Klassendiagramm	xvi

Tabellenverzeichnis

1	Zeitplanung	2
2	Kostenaufstellung	4
3	Zwischenstand nach der Analysephase	6
4	Zwischenstand nach der Entwurfsphase	10
5	Zwischenstand nach der Implementierungsphase	13
6	Zwischenstand nach der Abnahmephase	13
7	Zwischenstand nach der Einführungsphase	14
8	Zwischenstand nach der Dokumentation	15
9	Soll-/Ist-Vergleich	15

Listings

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

Abkürzungsverzeichnis

GUI	Graphical User Interface
SVN	Subversion
UML	Unified Modeling Language

1 Einleitung

1.1 Auftraggeber

Die Phoenix Group IT GmbH ist der IT-Dienstleister des Pharmagroßhändlers Phoenix Pharmahandel GmbH & Co. KG. Phoenix Pharmahandel ist mit seinen Tochtergesellschaften unter dem Namen “Phoenix Group” europaweit tätig mit etwa 30000 Mitarbeitern. Haupttätigkeiten der Phoenix Group ist die Belieferung von Apotheken mit Medikamenten.

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

1.2 Projektumfeld

Phoenix verwendet für die Datenverarbeitung im Bereich Lager ein firmeneigenes Dateiformat, im Folgenden 1920Schema genannt. Anhand von 1920Schemas werden Datenströme zergliedert. 1920Schemas dienen als Vorlage für Copybooks¹, als Schnittstelle zu SSORT² und um Daten vom Mainframe zum Lagerrechner zu schicken und auf diesem in Logdateien zu speichern.

1920Schemas beschreiben einen Satz hierarchisch gegliederter Variablen, und für jede Variable deren Typ und Größe in Bytes. Datenströme von typischerweise 1920 Bytes³ werden anhand der Schemas zergliedert und erhalten so eine Bedeutung. Phoenix nutzt Dutzende verschiedene 1920Schemas für die Datenverarbeitung, regelmäßig arbeiten die Entwickler aber nur mit etwa 10.

1.3 Projektziel

Ziel des Projektes ist es, ein Programm (1920Parser) zu schreiben, in dem ein Datenstrom und ein 1920Schema angegeben werden, und das den Datenstrom anhand des Schemas zergliedert anzeigt.

1.4 Projektbegründung

Bei Kundenreklamationen, Änderungen an Programmen und Neuentwicklungen stehen die Programmierer vor zwei Aufgaben:

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

¹COBOL-Datei, in der eine Variablenstruktur definiert wird

²IBM-Programm, zeigt Copybooks an

³das Terminal Window ist 24 Zeilen * 80 Spalten groß und fasst 1920 Zeichen

Gegenwärtig zählen die Entwickler die passende Anzahl von Bytes in Schema und Datenstrom ab, einige erfahrene Entwickler erkennen bestimmte Werte in einem Datenstrom und müssen nicht beim ersten Byte anfangen zu zählenzählen.

1.5 Projektschnittstellen

Benutzer des Projektes sind die Programmierer der Phoenix Group IT GmbH, hauptsächlich die der Abteilung Warehouse.

1920Parser interagiert nicht unmittelbar mit anderen Systemen⁴. Vorgesehen ist, dass die Benutzer die notwendigen Angaben direkt in eine Eingabemaske hineinkopieren.

Die 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 und wird 1920Parser auch selbst verwenden.

2 Projektplanung

2.1 Projektphasen

Das Projekt findet im Zeitraum vom 11.04.2016 - 15.05.2016 statt. Genaue Zeitplanung

Beispiel Tabelle 1 zeigt ein Beispiel für eine grobe Zeitplanung.

Projektphase	Geplante Zeit
Analysephase	9 h
Entwurfsphase	19 h
Implementierungsphase	29 h
Abnahmetest der Fachabteilung	1 h
Einführungsphase	1 h
Erstellen der Dokumentation	9 h
Pufferzeit	2 h
Gesamt	70 h

Tabelle 1: Zeitplanung

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

⁴der Anwendungsfall Schema vom Mainframe herunterladen wurde im Rahmen des Projektes nicht umgesetzt

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 wird ein agiler Entwicklungsprozess angewendet, der an Extreme Programming angelehnt ist. Der im Projekt angewandte Prozess inkludiert die Extreme Programming-Praktiken permanente Integration, testgetriebene Entwicklung, häufige Kundeneinbeziehung, häufiges Refaktorisieren, kurze Iterationen und einfaches Design.

3 Analysephase

Zur Analyse der gegenwärtigen Situation verabredete der Autor einen Termin mit Herrn Kemmer. Der Autor bereitete eine Liste mit Fragen vor, die er Herrn Kemmer stellte. Hier erfragte der Autor z. B. unter welchen Umständen genau das Programm benötigt wird, woher die Schemas stammen, 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.

3.1 Ist-Analyse

Die Entwickler der Phoenix verwenden Schemadateien, in denen Variablen definiert werden, als Schnittstelle zu anderen Programmen. Datenströme werden diesen Schemas entsprechend zergliedert. Die Entwickler müssen bei Kundenreklamationen und Programmänderungen die richtige Anzahl Bytes im Datenstrom abzählen, um ihre Bedeutung herauszufinden.

3.2 Wirtschaftlichkeitsanalyse

Durch 1920Parser nimmt den Entwicklern das bisher notwendige Abzählen von Zeichen in Schema und Datenstrom ab. Das Projekt verspricht dadurch nicht nur Zeitersparnis, sondern verhindert auch wirkungsvoll Zählfehler.

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 Ressourcen⁵ 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 2 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 2: Kostenaufstellung

⁵Räumlichkeiten, Arbeitsplatzrechner etc.

3.2.3 Amortisationsdauer

Es wird davon ausgegangen, dass ausschließlich die Entwickler der Abteilung Lager das Programm nutzen werden. Nach Einschätzung von Herrn Kemmer arbeitet die Hälfte der 20 Entwickler regelmäßig mit Schemadateien. Er schätzte weiter, 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 $\frac{1015,00 \text{ €}}{12833,45 \text{ €/Jahr}} \approx 0,08 \text{ Jahre} \approx 1 \text{ Monat}$.

3.3 Nutzwertanalyse

- Darstellung des nicht-monetären Nutzens (z. B. Vorher-/Nachher-Vergleich anhand eines Wirtschaftlichkeitskoeffizienten).

Beispiel Ein Beispiel für eine Entscheidungsmatrix findet sich in Kapitel 4.3: [Architekturdesign](#).

3.4 Anwendungsfälle

Zur Analyse der Hauptanwendungsfälle ist die Anzeige des Anzeigens eines anhand eines angegebenen Schemas zergliederten Datenstroms. Die Aufnahme der Anforderungen mit Herrn Kemmer ergab einige Wunschkriterien: Schemas stammen vom Mainframe oder aus Logdateien.

3.5 Qualitätsanforderungen

Schemas müssen frei angegebbar sein und Datenströme richtig zergliedert werden. Beim Abnahmetest soll dies anhand der 10 wichtigsten Schemadateien überprüft werden. Performance ist (fast) egal, das Programm soll aber flüssig benutzbar sein (Zergliederung von Datenstrom und Anzeige in unter 5 Sekunden). Die Benutzer sind Profis. Es ist deshalb nicht unbedingt notwendig, für alles eine Eingabemaske bereitzustellen. Es ist genügt auch, wenn der Benutzer eine Einstellung durch Editieren einer Konfigurations-Datei ändern kann.

3.6 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.

3.7 Zwischenstand

Tabelle 3 zeigt den Zwischenstand nach der Analysephase. c vb

Vorgang	Geplant	Tatsächlich	Differenz
1. Analyse des Ist-Zustands	3 h	4 h	+1 h
2. „Make or buy“-Entscheidung und Wirtschaftlichkeitsanalyse	1 h	1 h	
3. Erstellen eines „Use-Case“-Diagramms	2 h	2 h	
4. Erstellen des Lastenhefts	3 h	3 h	

Tabelle 3: Zwischenstand nach der Analysephase

4 Entwurfsphase

4.1 Zielplattform

Das Programm soll auf den Entwicklerrechnern der Phoenix laufen, auf denen Windows 7 installiert ist. 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.

Das Programm hätte auch in C++ mit Qt geschrieben werden können. Die hohe Performance, die C++ verspricht, wird für 1920Parser aber nicht wirklich benötigt und die Wahl fiel daher auf C# aufgrund von dessen Garbage Collection, Linq und sehr 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.

4 Entwurfsphase

Da 1920Schemas zentral für das Projekt sind, wird ihr Aufbau hier kurz dargestellt. Eine Zeile eines 1920Schemas enthält entweder eine Variablendeklaration oder Metainformationen zum Schema. Die Metainformationen sind für 1920Parser nicht relevant und werden ignoriert. Variablenzeilen haben Angaben zu diesen Feldern:

- Redefiniert diese Variable die gleichen Bytes wie eine frühere
- Levelnummer
- Variablenname
- Typ
- Anzahl der definierten Bytes
- Wiederholzahl
- Kommentar
-

Alle Felder sind durch Leerzeichen getrennt. Variablenzeilen haben nicht-leere Angaben zu Levelnummer und Variablenname, alle anderen Felder können leer sein⁶ Die Variablen von 1920Schemas sind anhand ihrer Levelnummer hierarchisch gegliedert. Eine Schemavariabel mit größerer Stufennummer als die vorhergehende ist dessen Kind. Eine Variable kann entweder eine Gruppen- oder eine Wertvariable sein. Wertvariablen haben einen Typ (C, X, N oder P) und eine Längenangabe, wie viele Bytes ihnen zugewiesen werden. Gruppenvariablen haben nie einen Typ und nie eine Länge. Ihnen sind weitere Gruppen- oder Wertvariablen als Kinder untergeordnet. Sowohl Gruppen- als auch Wertvariablen können eine Angabe haben, wie oft sie sich wiederholen (default ist 1). Wenn sich Gruppenvariablen wiederholen, wiederholt sich auch die gesamte Variablen-Hierarchie unterhalb von ihnen.

Viele Schemas definieren als erste Variable eine Variable für ihren Schemanamen, so dass viele Datenströme mit dem Namen des zu verwendenden Schemas beginnen.

Beim Treffen von Annahmen über den Aufbau der Schemazeilen ergaben sich unerwartet Schwierigkeiten mit dem Feld Kommentar. In der Schemadatei IOVK92 wurde eine Variable gefunden, die keine Wiederholzahl hatte, aber ein Kommentar-Feld, das mit "0" begann. Es musste eine Regel festgelegt werden, damit diese 0 nicht als Wiederholzahl interpretiert wird, sondern als Anfang des Kommentars.

Nach Rücksprache mit Herrn Kemmer wurde auf dessen Vorschlag und nach erneuter Prüfung der Schemadateien festgelegt, dass der Kommentar mit mindestens 2 Leerzeichen vom vorhergehenden Feld getrennt sein muss.

⁶das bedeutet, alle Zeilen, die nicht mit einer Zahl, gefolgt von Leerzeichen und Buchstaben beginnen, werden ignoriert

4.3 Architekturdesign

Die Anwendungslogik wurde von der GUI getrennt, indem sie in eigene Klassen kam (Schema, Abstract-Node).

4.4 Entwurf der Benutzeroberfläche

Nachdem als Programmiersprache C# feststand, kamen für die Benutzeroberfläche nur Plattformen aus dem .NET-Framework in Frage. Die Wahl zwischen ASP.NET, WPF und Winforms fiel schließlich auf Winforms. Da Benutzer Schemas speichern können sollen, wäre ein Webinterface (ASP.NET) mit zusätzlichem Aufwand verbunden gewesen, vor allem für die Benutzerverwaltung. WPF und Winforms waren beide gut geeignet, der Autor ist aber erfahrener mit Winforms. Er entschied sich daher für Winforms.

4.4.1 Erste Iteration: Datenstrom zergliedert anzeigen

Zunächst wurden drei TextBoxen zur Eingabe von Datenstrom, Schema und zur Anzeige des Ergebnisses erstellt. Jede dieser TextBoxen bekam als Überschrift ein Label (Datenstrom, Schema und Ergebnis). Da die Schemadateien fast immer formatiert sind (gleiche Eigenschaften stehen untereinander) wurde ein Monospace-Font (Courier New) gewählt, um diese Formatierung beizubehalten. Das Ergebnis-Textfeld bekam aufgrund seiner höheren Bedeutung die doppelte Größe.

4.4.2 Zweite Iteration: Schema speichern

Nachdem der Anwendungsfall "Datenstrom zergliedert anzeigen" erledigt war, wurde ein Bedienkonzept für die Anwendungsfälle

- Schema speichern
- Datenstrom importieren
- Variablen ausblenden
- Schema automatisch auswählen

entwickelt. Im Gegensatz zum Hauptanwendungsfall "Datenstrom zergliedern", wird die Funktionalität dieser Anwendungsfälle seltener gebraucht. Um die Benutzeroberfläche übersichtlich zu halten, entschied sich der Autor, ein Menü anzulegen, mit Einträgen für jeden dieser Anwendungsfälle.

Das Phoenix-Icon wurde als Programmsymbol gewählt und das Firmenlogo von Phoenix rechts in der Menüleiste eingefügt.

4.5 XML-Datenmodell

Um die Informationen zur automatischen Auswahl eines 1920Schemas⁷ zu speichern, wurde ein XML-Schema definiert. Mit dem Microsoft-Tool xsd.exe wurde daraus eine C# Klasse generiert.

4.6 Geschäftslogik

Die hierarchische Aufbau von 1920Schemas lässt sich gut mit einer rekursiven Baumstruktur im Programm abbilden. Der Programmablauf beim Zergliedern eines Schemas muss dann zunächst das Schema in diese Baumstruktur überführen, danach können die Werte aus dem Datenstrom an die Knoten des Baumes zugewiesen werden.

Beispiel Für die Baumstruktur wurde die abstrakte Klasse `AbstractNode` erstellt. Von dieser Klasse erben die Klassen `GroupNode` und `ValueNode`, die Gruppen- und WerteVariablen darstellen. `GroupNode` hat ein Attribut `children` vom Typ `List<AbstractNode>`, die auf seine KindKnoten verweist. `GroupNode` ist rekursiv definiert, weil sie wiederum `GroupNodes` als Kindknoten haben kann.

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

Abbildung 1 zeigt den grundsätzlichen Programmablauf beim Einlesen eines Moduls als **EPK! (EPK!)**.

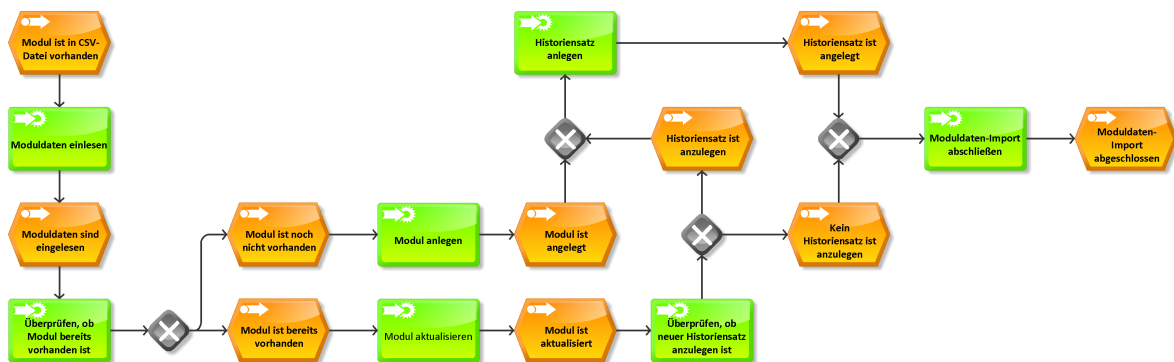


Abbildung 1: Prozess des Einlesens eines Moduls

4.7 Maßnahmen zur Qualitätssicherung

Das korrekte Parsen des 1920Schemas und die richtige Zergliederung des Datenstroms sind zentral für 1920Parser. Um sicherzugehen, dass dieser Programmteil funktioniert, und weil sich der Autor Zeitersparnis durch schnelle Rückmeldung erhoffte, wurden daher die Methoden der Klassen `Schema`, `Abstract`,

⁷und eventuell einmal zum Ausblenden von Schemavariablen

Group- und ValuNode testgetrieben entwickelt. Die Programmfunktionen zum Speichern von Schemas und zur automatischen Auswahl eines Schemas sind nicht so zentral, diese wurden von Hand getestet.

4.8 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.

4.9 Zwischenstand

Tabelle 4 zeigt den Zwischenstand nach der Entwurfsphase.

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 4: Zwischenstand nach der Entwurfsphase

5 Implementierungsphase

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 Datenstrukturen

Für Gruppen- und Wertvariablen wurden die beiden Klassen GroupNode und ValueNode erstellt. GroupNode soll eine Liste mit Referenzen auf seine Kindknoten (vom Typ GroupNode oder vom Typ ValueNode) bekommen. Damit beide Klassenarten in diese Liste aufgenommen werden können, müssen sie das

gleiche Interface implementieren oder von der gleichen Klasse erben. Beide Klassen haben einige Attribute gemeinsam (Level, Variablenname, Kommentar), deshalb erben sie von der abstrakten Oberklasse `AbstractNode`.

Diese Klassenstruktur kann einen beliebig tief verschachtelbaren `AbstractNode`-Baum darstellen, weil die Kindknoten-Liste von `GroupNode` wiederum `GroupNodes` enthalten kann (und diese wieder, und deren wieder, und so weiter).

Die Methode zum Parsen eines Schemas passte in keine der Klassen, sie kam in eine eigene Klasse namens `Schema`. Die Methode, die das Schema parst, soll den Wurzelknoten als `GroupNode` zurückgeben. Daher musste die Implementierung mit den Klassen der Baumstruktur (`AbstractNode`, `GroupNode` und `ValueNode`) begonnen werden. Zum Schreiben von Unit-Tests wurde eine Methode zum Vergleichen der Ergebnisse benötigt. Dazu wurden die Methoden `equals()` und `hashCode()` der Klasse `object` überschrieben. Danach wurde die Baumstruktur gemockt und die Methoden `AssignValue()` und `ToString()` implementiert.

Danach wurde die Methode `Parse()` der Klasse `Schema` geschrieben. Hier war ein Problem, dass gleichzeitig

5.2 Implementierung der Benutzeroberfläche

Die Graphical User Interface (GUI) muss dem Benutzer Funktionalität bereitstellen, damit er Schema und Datenstrom angeben kann und sie muss den zergliederten Datenstrom anzeigen können. Um diese Funktionalitäten zu bieten werden 3 Textboxen angezeigt, jeweils mit einem Label als Überschrift. Auf ein Menü wurde zunächst verzichtet.

Die Angaben in den Schemadateien sind formatiert, gleiche Angaben wie Level oder Variablenname stehen untereinander. Damit die Angaben in der GUI auch untereinander stehen, wurde für das Schema-Textfeld ein Monospace-Font verwendet (Courier New). Um das Erscheinungsbild einheitlich zu halten, zeigen auch die Textfelder für Datenstrom und Ergebnis Ihren Inhalt mit diesem Zeichensatz an.

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

5.3.1 Grundschema der rekursive Methoden von `AbstractNode/GroupNode`

`Group-` und `ValueNode` erben von `AbstractNode` Methoden. Alle diese Methoden ähneln sich und laufen für `Group-` und `ValueNodes` nach folgendem Schema ab:

`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.

Es ist kein Zufall, dass alle Methoden rekursiv sind, denn die Klassenstruktur ist rekursiv definiert.

5.3.2 Parsen des Schemas in eine Baumstruktur

Da der Baum einen einzigen Wurzelknoten⁸ haben soll, Schemas aber nicht zwingend einen haben, erhält der Baum einen künstlichen Wurzelknoten mit der Stufennummer 0. Würde man keinen solchen Knoten einführen, müsste statt mit GroupNode mit List<AbstractNode> als Rückgabewert gearbeitet werden, da es auf höchster Ebene mehrere Geschwister geben kann oder Wertvariablen ohne Eltern.

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.

5.3.3 Zuweisen der Werte aus dem Datenstrom

Die Methode zum Zuweisen eines Wertes an einen Knoten hat den Prototypen

```
int AssignValue(string data).
```

Die Methode erwartet den Datenstrom als string und gibt die Anzahl der verbrauchten Bytes zurück. Das aufrufende Objekt (normalerweise eine GroupNode) bekommt dadurch Informationen, wie viele Bytes des Datenstroms benutzt wurden. So kann der Anfang des Datenstroms für jedes Kind passend verschoben werden (und bei Redefines auch wieder zurückverschoben werden).

ValueNodes und GroupNodes machen beim Aufruf Folgendes:

ValueNodes weisen ihrem Value-Attribut die benötigte Anzahl Buchstaben aus data zu und geben diese Anzahl zurück.

GroupNodes rufen für alle ihre Kinder nacheinander AssignValue(data) auf. Jedes Kind gibt die Anzahl der verwendeten Bytes zurück. GroupNodes merken sich diese Anzahl und können dadurch für jedes Kind

⁸Knoten, der Vorfahr von allen anderen Knoten ist

den Anfang des Datenstroms um diese Anzahl verschieben. GroupNode gibt die Summe der verwendeten Bytes aller seiner Kinder zurück.

Beispiel Die Klasse `ComparedNaturalModuleInformation` findet sich im Anhang A.10: Klasse: `ComparedNaturalModuleInformation` auf Seite xiii.

5.4 Zwischenstand

Tabelle 5 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 5: Zwischenstand nach der Implementierungsphase

6 Abnahmephase

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.

6.1 Zwischenstand

Tabelle 6 zeigt den Zwischenstand nach der Abnahmephase.

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

Tabelle 6: 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 7 zeigt den Zwischenstand nach der Einführungsphase.

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

Tabelle 7: 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 PHPDoc⁹ automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse findet sich im Anhang [A.8: Entwicklerdokumentation](#) auf Seite x.

8.1 Zwischenstand

Tabelle 8 zeigt den Zwischenstand nach der Dokumentation.

⁹Vgl. ?

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 8: Zwischenstand nach der Dokumentation

9 Fazit

9.1 Soll-/Ist-Vergleich

Die Musskriterien wurden vollständig umgesetzt. Die Projektvorgaben wurden vollständig umgesetzt. Zusätzlich wurde Funktionalität implementiert zum Speichern von Schemas und zur automatisierten Auswahl des zum Datenstrom passenden Schemas. Für den Anwendungsfall "Datenstrom importieren" wurde ein Bedienkonzept entwickelt, aber mit der Umsetzung nicht begonnen. Ausblenden von Schemaknoten ist nicht implementiert.

- 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).

Beispiel (verkürzt) Wie in Tabelle 9 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 9: Soll-/Ist-Vergleich

9.2 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).

9.3 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.

Einige Ideen für Weiterentwicklungen von 1920Parser sind:

- Ausblenden von angebbaren Knotenpunkten
- Implementierung des Anwendungsfalls "Datenstrom importieren"
- Bei Klick auf eine Zeile im Schema wird zur entsprechenden Zeile im Ergebnis-Textfeld gesprungen.
- Bei Klick ins Datenstrom-Textfeld wird zur entsprechenden Zeile im Ergebnis-Datenfeld gesprungen
- Erweiterung der Eingabemaske für Schema speichern, so dass mehrere Angaben zur automatischen Auswahl getroffen werden können.
- Automatisches Herunterladen von Schemas vom Mainframe
-

1920Parser könnte eine nette Spielwiese für einen zukünftigen Auszubildenden 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>.

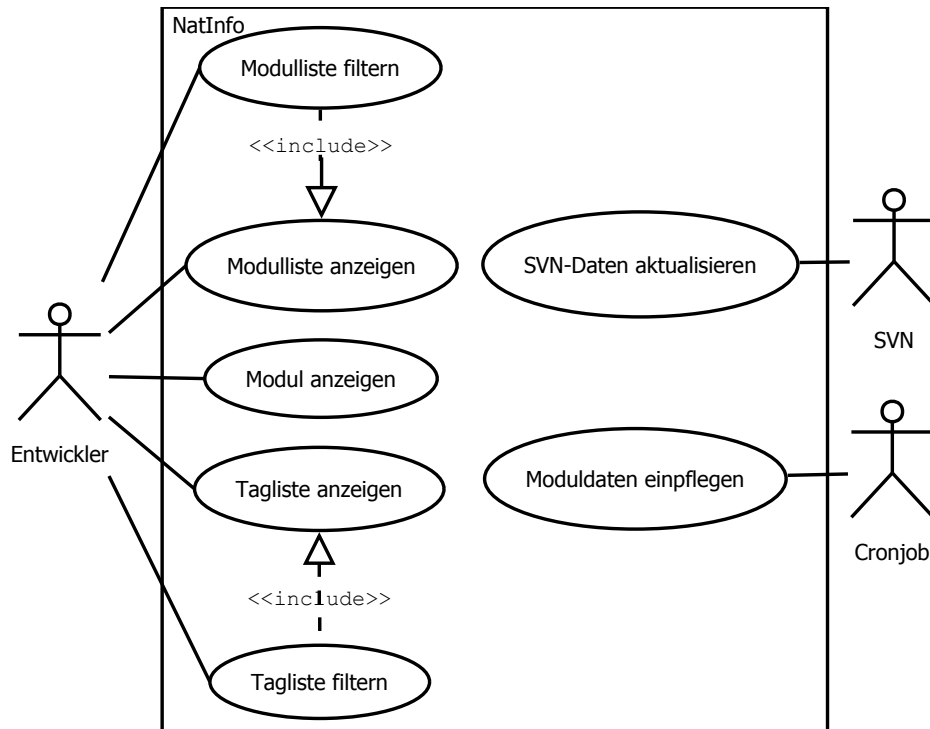


Abbildung 2: Use Case-Diagramm

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 tätigen 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 Subversion (**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/>.

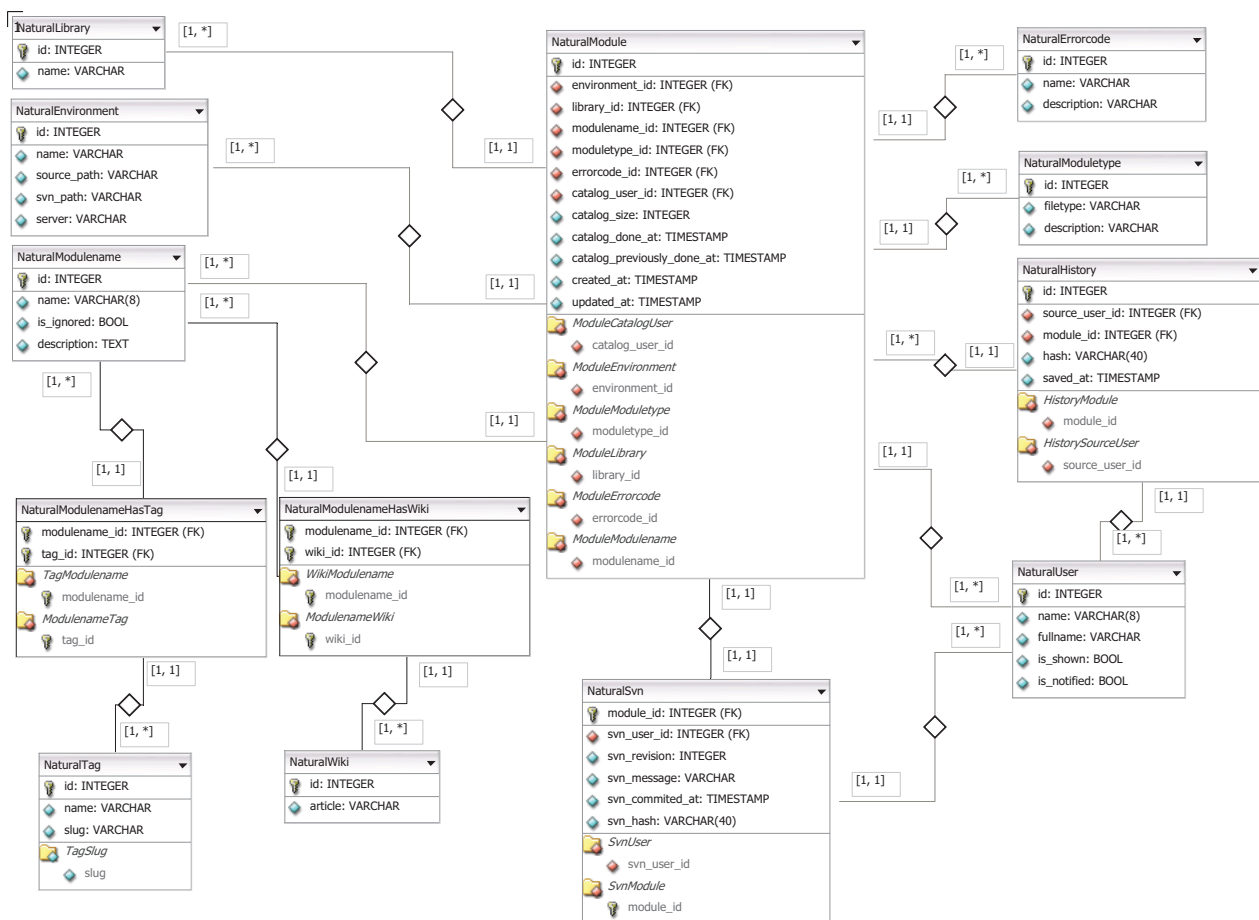


Abbildung 3: Datenbankmodell

A.6 Oberflächenentwürfe

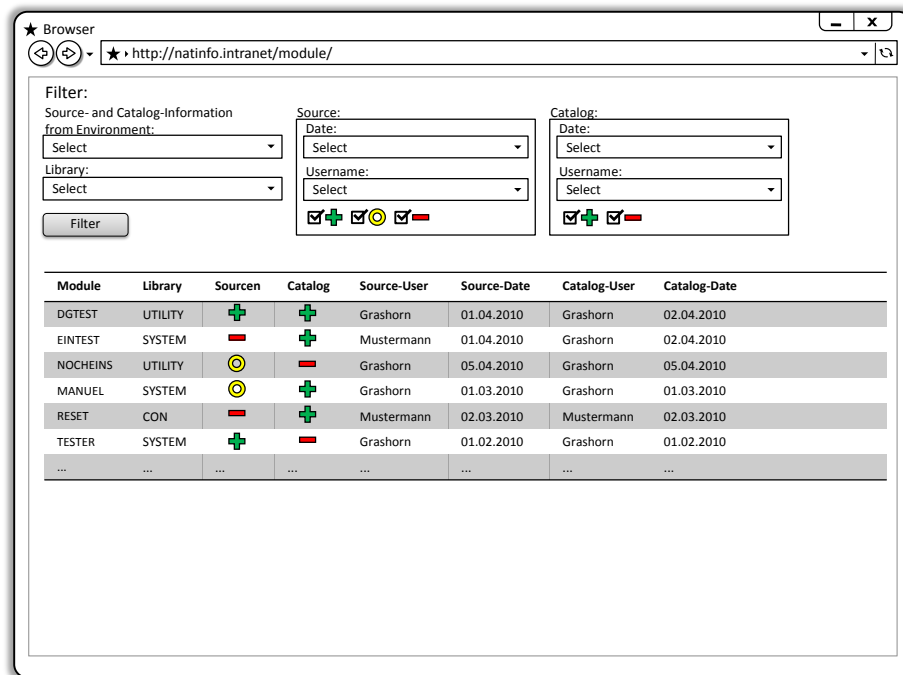


Abbildung 4: Liste der Module mit Filtermöglichkeiten

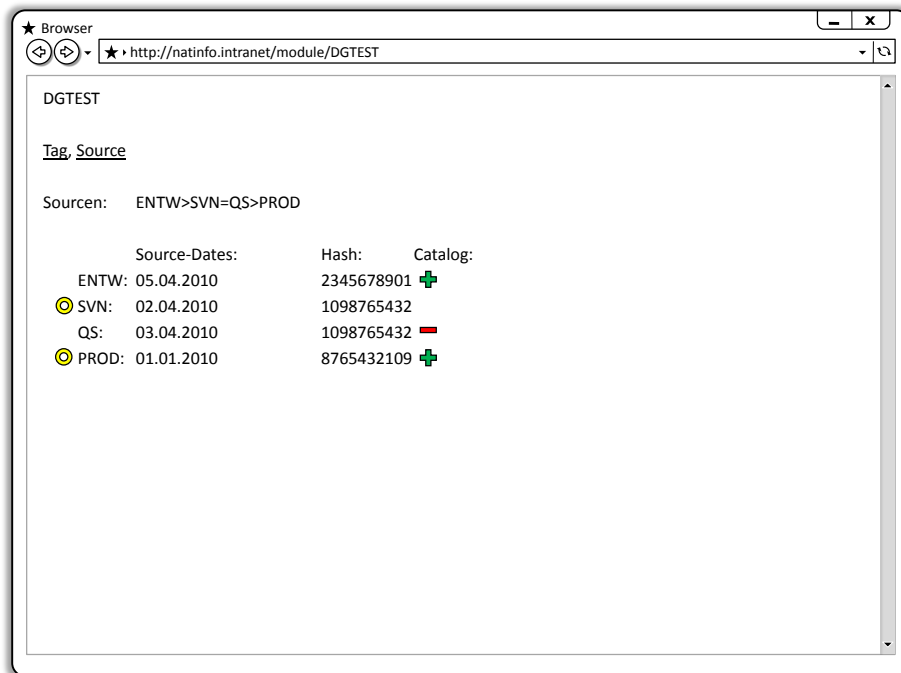


Abbildung 5: Anzeige der Übersichtsseite einzelner Module

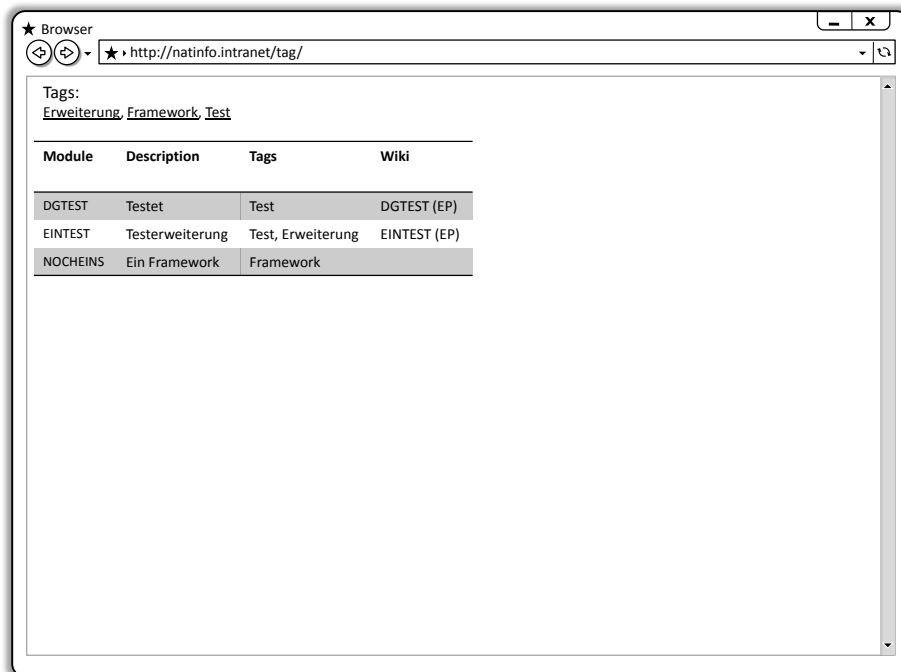


Abbildung 6: Anzeige und Filterung der Module nach Tags

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 7: 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 8: 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.

René Ederer

Seite x von i

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:

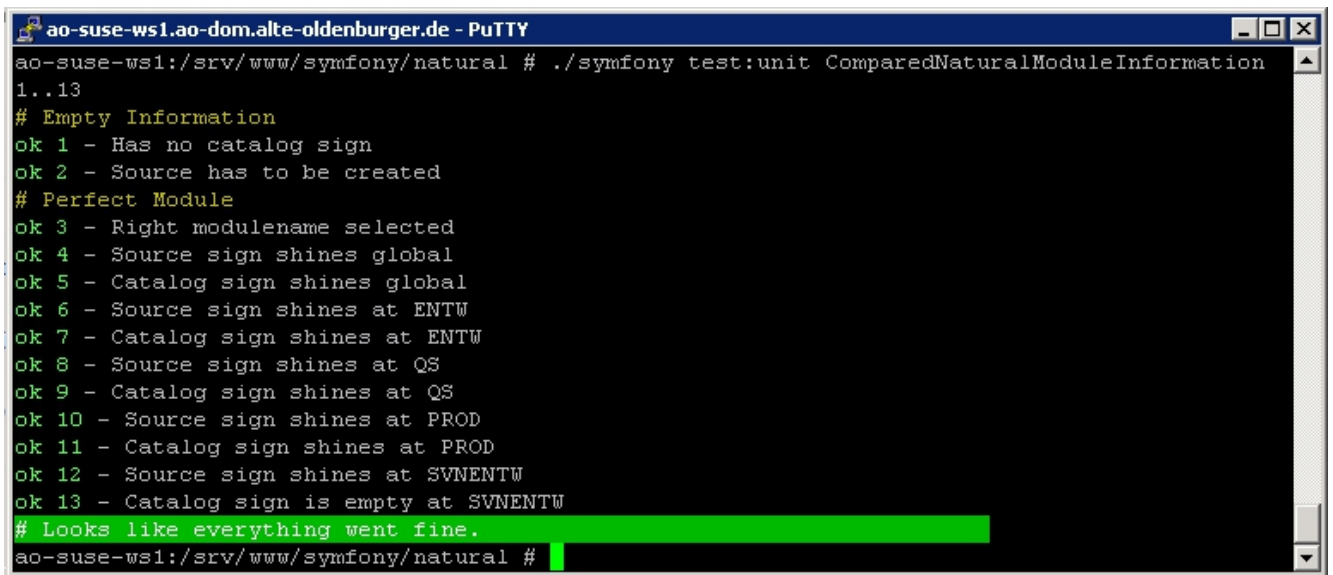
access: public

[\[Top \]](#)

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 9: 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 }
```

```
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>.

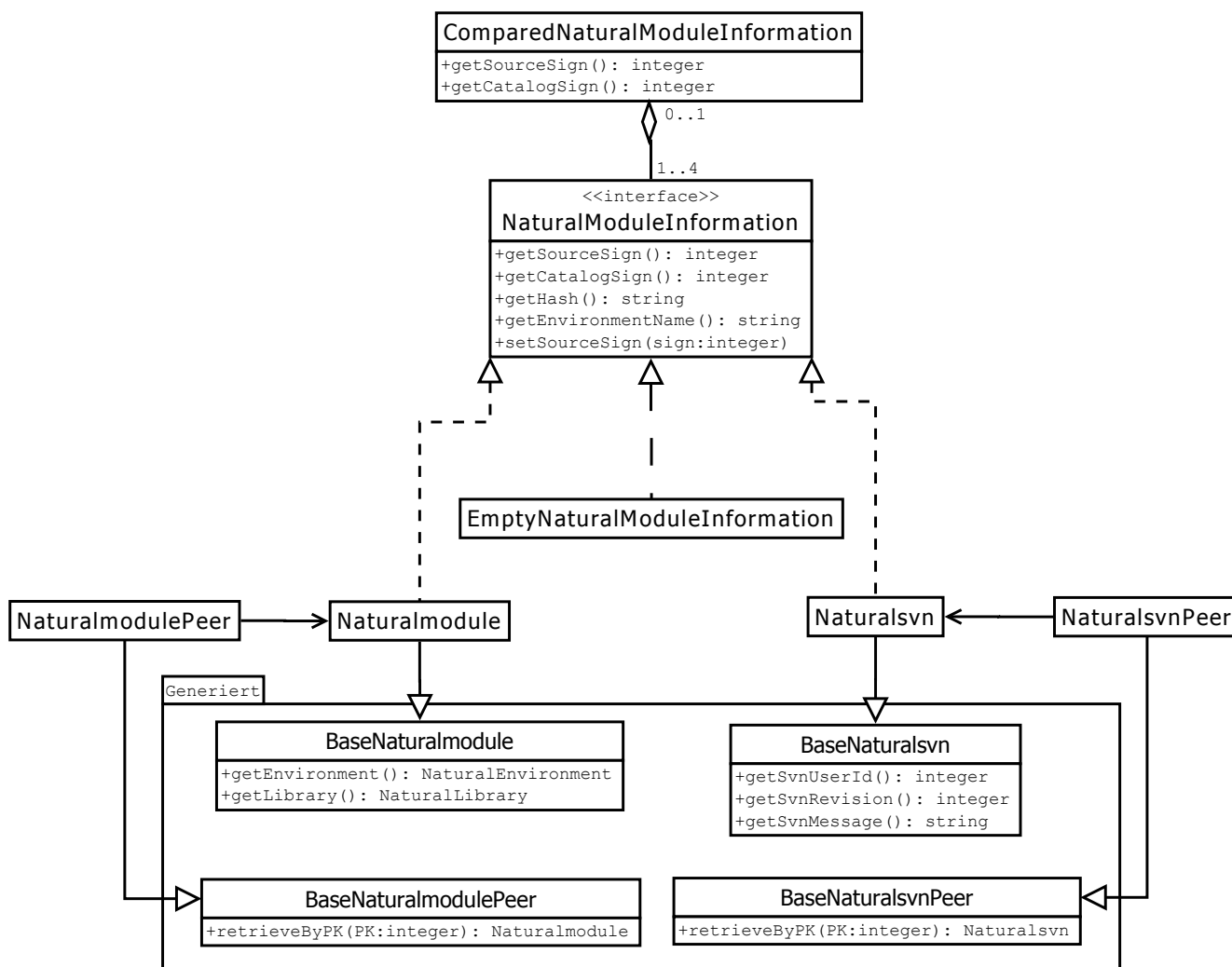







Abbildung 10: Klassendiagramm

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.