

Prüfungsteil A

Prüfling (private Anschrift):	Ausbildungsbetrieb:
-------------------------------	---------------------

Bestätigung über durchgeführte Projektarbeit

diese Bestätigung ist mit der Projektdokumentation einzureichen

Ausbildungsberuf (bitte unbedingt angeben):

Projektbezeichnung:

Projektbeginn: _____	Projektfertigstellung: _____	Zeitaufwand in Std.: _____
----------------------	------------------------------	----------------------------

Bestätigung der Ausbildungsfirma:

Wir bestätigen, dass der/die Auszubildende das oben bezeichnete Projekt einschließlich der Dokumentation im Zeitraum

vom: _____ bis: _____ selbständig ausgeführt hat.

Projektverantwortliche(r) in der Firma:

Vorname	Name	Telefon	Unterschrift
---------	------	---------	--------------

Ausbildungsverantwortliche(r) in der Firma:

Vorname	Name	Telefon	Unterschrift
---------	------	---------	--------------

Eidesstattliche Erklärung:

Ich versichere, dass ich das Projekt und die dazugehörige Dokumentation selbständig erstellt habe.

Ort und Datum: _____ Unterschrift des Prüflings: _____



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

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

Inhaltsverzeichnis

Abbildungsverzeichnis	3
Tabellenverzeichnis	4
Listings	5
Abkürzungsverzeichnis	6
1 Einleitung	7
1.1 Auftraggeber	7
1.2 Projektumfeld	7
1.3 Projektziel	7
1.4 Projektbegründung	7
1.5 Projektschnittstellen	8
2 Projektplanung	8
2.1 Projektphasen	8
2.2 Ressourcenplanung	8
2.3 Entwicklungsprozess	9
3 Analysephase	9
3.1 Ist-Analyse	9
3.2 Wirtschaftlichkeitsanalyse	9
3.2.1 „Make or Buy“-Entscheidung	9
3.2.2 Projektkosten	9
3.2.3 Amortisationsdauer	10
3.3 Nutzwertanalyse	10
3.4 Anwendungsfälle	11
3.5 Qualitätsanforderungen	11
3.6 Lastenheft/Fachkonzept	11
3.7 Zwischenstand	11
4 Entwurfsphase	12
4.1 Zielplattform	12
4.2 Aufbau der Schemadateien	12
4.3 Architekturdesign	12
4.4 Entwurf der Benutzeroberfläche	12
4.4.1 Erste Iteration: Datenstrom zergliedert anzeigen	13
4.4.2 Zweite Iteration: Schema speichern	13
4.5 XML-Datenmodell	13
4.6 Geschäftslogik	13

Inhaltsverzeichnis

4.7	Maßnahmen zur Qualitätssicherung	14
4.8	Pflichtenheft/Datenverarbeitungskonzept	14
4.9	Zwischenstand	15
5	Implementierungsphase	15
5.1	Erstellen einer neuen Solution	15
5.2	Implementierung der Datenstrukturen	15
5.3	Implementierung der Benutzeroberfläche	16
5.4	Implementierung der Geschäftslogik	16
5.4.1	Grundschema der rekursive Methoden von AbstractNode/GroupNode	16
5.4.2	Parzen des Schemas in eine Baumstruktur	17
5.4.3	Zuweisen der Werte aus dem Datenstrom	18
5.5	Zwischenstand	18
6	Abnahmephase	18
6.1	Zwischenstand	19
7	Einführungsphase	19
7.1	Zwischenstand	19
8	Dokumentation	20
8.1	Zwischenstand	20
9	Fazit	20
9.1	Soll-/Ist-Vergleich	20
9.2	Lessons Learned	21
9.3	Ausblick	21
	Eidesstattliche Erklärung	23
A	Anhang	24
A.1	Detaillierte Zeitplanung	24
A.2	Lastenheft (Auszug)	25
A.3	Use Case-Diagramm	26
A.4	Pflichtenheft (Auszug)	26
A.5	Datenbankmodell	28
A.6	Oberflächenentwürfe	29
A.7	Screenshots der Anwendung	31
A.8	Entwicklerdokumentation	33
A.9	Testfall und sein Aufruf auf der Konsole	35
A.10	Klasse: ComparedNaturalModuleInformation	36
A.11	Klassendiagramm	39
A.12	Benutzerdokumentation	40

Abbildungsverzeichnis

1	Prozess des Einlesens eines Moduls	14
2	Use Case-Diagramm	26
3	Datenbankmodell	28
4	Liste der Module mit Filtermöglichkeiten	29
5	Anzeige der Übersichtsseite einzelner Module	30
6	Anzeige und Filterung der Module nach Tags	30
7	Anzeige und Filterung der Module nach Tags	31
8	Liste der Module mit Filtermöglichkeiten	32
9	Aufruf des Testfalls auf der Konsole	36
10	Klassendiagramm	39

Tabellenverzeichnis

1	Zeitplanung	8
2	Kostenaufstellung	10
3	Zwischenstand nach der Analysephase	11
4	Zwischenstand nach der Entwurfsphase	15
5	Zwischenstand nach der Implementierungsphase	18
6	Zwischenstand nach der Abnahmephase	19
7	Zwischenstand nach der Einführungsphase	19
8	Zwischenstand nach der Dokumentation	20
9	Soll-/Ist-Vergleich	21

Listings

Listings/tests.php	35
Listings/cnmi.php	36

Abkürzungsverzeichnis

GUI	Graphical User Interface
SVN	Subversion
UML	Unified Modeling Language

1 Einleitung

Ausbildungsbetrieb ist die Phoenix Group IT GmbH.

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

1.2 Projektumfeld

Phoenix hat für die Datenverarbeitung im Bereich Lager ein eigenes Dateiformat (im Folgenden 1920Schema genannt) entwickelt, das als Schnittstelle zu verschiedenen Programmen dient. 1920Schemas dienen als Vorlage für Copybooks¹, als Schnittstelle zu SSORT² und um Daten vom Mainframe zum Lagerrechner zu schicken und dort 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.

¹COBOL-Datei, in der eine Variablenstruktur definiert wird

²IBM-Programm, zeigt Copybooks an

³das Terminal Window ist 24 Zeilen * 80 Spalten groß

- Datenstrom-Bytes einer Schemadatei-Variablen zuordnen.

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

1.5 Projektschnittstellen

Benutzer des Projektes sind die Programmierer der Phoenix Group IT GmbH. 1920Parser interagiert nicht unmittelbar mit anderen Systemen. Vorgesehen ist, dass der Benutzer die notwendigen Angaben direkt in eine Eingabemaske hineinkopiert.

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.

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 24.

2.2 Ressourcenplanung

Windows 7, Visual Studio Ultimate 2010, Microsoft Visio 2010, TexMaker, OpenText HostExplorer, Büro mit PC mit Verbindung zum Mainframe PC, Büro, 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

3.1 Ist-Analyse

3.2 Wirtschaftlichkeitsanalyse

Das Projekt verspricht nicht nur eine deutliche Zeitersparnis für die COBOL-Programmierer, sondern auch eine verringerte Fehlerquote.

3.2.1 „Make or Buy“-Entscheidung

Die Anforderungen sind sehr speziell, es ist daher unwahrscheinlich, dass es ein fertiges Programm gibt, das die Anforderungen erfüllt. Das Produkt wird daher selbst erstellt.

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 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.

⁴Räumlichkeiten, Arbeitsplatzrechner etc.

3 Analysephase

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

3.2.3 Amortisationsdauer

- Welche monetären Vorteile bietet das Projekt (z. B. Einsparung von Lizenzkosten, Arbeitszeiterparnis, bessere Usability, Korrektheit)?
- Wann hat sich das Projekt amortisiert?

Beispielrechnung (verkürzt) Bei einer Zeiteinsparung von 5 Minuten am Tag für jeden der 20 Anwender und 220 Arbeitstagen im Jahr ergibt sich eine gesamte Zeiteinsparung von

$$20 \cdot 220 \text{ Tage/Jahr} \cdot 5 \text{ min/Tag} = 22000 \text{ min/Jahr} \approx 366,67 \text{ h/Jahr} \quad (4)$$

Dadurch ergibt sich eine jährliche Einsparung von

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

Die Amortisationszeit 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

Beispiel Ein Beispiel für ein Use Case-Diagramm findet sich im Anhang A.3: Use Case-Diagramm auf Seite 26.

3.5 Qualitätsanforderungen

Schemas müssen frei angegebbar sein und Datenströme richtig zergliedert werden.

Performance ist nicht kritisch. Wünschenswert wäre eine Zergliederung des Datenstroms in unter 5 Sekunden. Die Benutzer sind Profis, es ist deshalb OK, auch mal den Benutzer eine XML-Datei editieren zu lassen.

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 25.

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. Diese laufen mit Intel i5 Prozessoren mit 4 GB Arbeitsspeicher und 32bit Versionen von Windows 7.

Die Wahl der Programmiersprache wurde zunächst auf die bei Phoenix bisher eingesetzten Sprachen COBOL, C++ und C# eingegrenzt. COBOL schied als Programmiersprache für ein Windows-Tool aus. Die Wahl fiel auf C# (Garbage Collection, moderner GUI-Designer, expressiver als C++).

4.2 Aufbau der Schemadateien

Eine Zeile eines 1920Schemas enthält entweder eine Variablendeklaration oder Metainformationen zum Schema. Die Metainformationen sind für 1920Parser nicht relevant und werden ignoriert.

Variablendeklarations-Zeilen sind gekennzeichnet durch Angaben zu Stufennummer und Variablennamen. 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. Kinder haben immer eine größere Stufennummer als ihre Eltern. Sowol Gruppen- als auch Wertvariablen können eine Angabe haben, wie oft sie sich wiederholen (default ist 1); außerdem haben sie eine Redefine-Angabe, ob sie die gleichen Bytes verwenden wie ihr vorhergehender Geschwister⁵.

4.3 Architekturdesign

GUI und Anwendungslogik wurden getrennt.

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 (Benutzerverwaltung usw.). WPF bietet mehr Möglichkeiten für Databindings und große Kontrolle über das Design als Winforms, dagegen ist Winforms einfacher. Die Wahl fiel auf Winforms.

⁵Geschwister haben die gleichen Eltern

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 Bedeutung die doppelte Größe.

4.4.2 Zweite Iteration: Schema speichern

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

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

entwickelt. Um die Benutzeroberfläche übersichtlich zu halten,

Das Phoenix-Icon wurde als Programm-Icon gewählt und das Firmen-Logo rechts in der Menüleiste eingefügt.

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

Beispiel Beispielentwürfe finden sich im Anhang [A.6: Oberflächenentwürfe](#) auf Seite 29.

4.5 XML-Datenmodell

Zur automatischen Auswahl eines Schemas wurde ein XML-Schema verwendet. Siehe Anhang.

4.6 Geschäftslogik

Es wird eine Baumstruktur verwendet, um den hierarchischen Aufbau der Schemadateien im Programm zu repräsentieren. Zunächst wird das Schema in eine Baumstruktur überführt. Danach werden die Werte aus dem Datenstrom zugewiesen.

4 Entwurfsphase

Beispiel Der hierarchische Aufbau der Schemavariablen wird im Programm durch eine rekursive Baumstruktur repräsentiert. Dazu wurden die abstrakte Klasse `AbstractNode` erstellt. Von dieser Klasse erben die Klassen `GroupNode` und `ValuNode`, die Gruppen- und WerteVariablen darstellen. `GroupNode` hat ein Attribut `List<AbstractNode>`, die auf seine KindKnoten verweist.

Ein Klassendiagramm, welches die Klassen der Anwendung und deren Beziehungen untereinander darstellt kann im Anhang [A.11: Klassendiagramm](#) auf Seite 39 eingesehen werden.

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

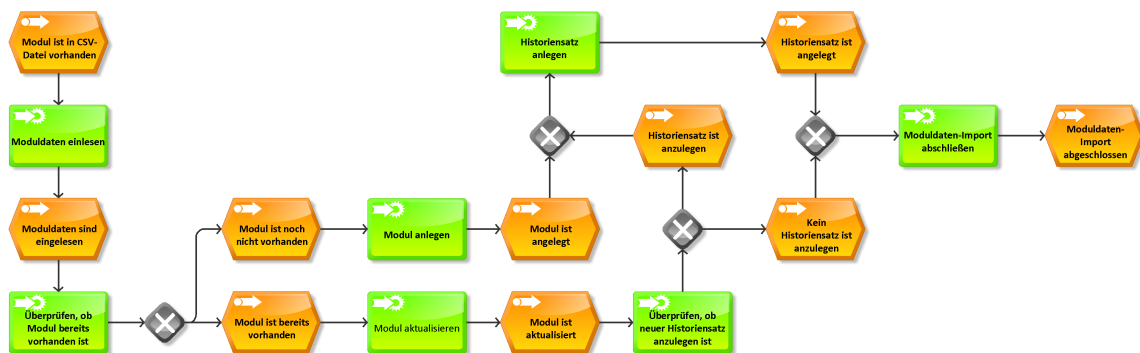


Abbildung 1: Prozess des Einlesens eines Moduls

4.7 Maßnahmen zur Qualitätssicherung

Es wurde testgetrieben entwickelt.

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

4.8 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 26 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

test

5.1 Erstellen einer neuen Solution

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

5.2 Implementierung der Datenstrukturen

Die hierarchische Gliederung der Schemavariablen wird im Programm durch eine rekursive Baumstruktur abgebildet. Schemavariablen können Gruppen- oder Wertvariablen sein, Entsprechend gibt es im Programm die Klassen GroupNode und ValueNode. Die den beiden Klassen gemeinsamen Attribute und Methoden, unter anderem

- Level
- VarName
- RepeatCount
- RepeatIndex
- Comment
- AssignValue()

- ToString()

wurden in der Oberklasse AbstractNode definiert und von GroupNode und ValueNode geerbt. GroupNodes erhielten ein Attribut children vom Typ List<AbstractNode> und können so wiederum auf weitere GroupNodes verweisen (und diese auch wieder). Dadurch kann eine beliebig tief verschachtelbare Baumstruktur repräsentiert werden.

Die Methode, die das Schema parst, soll eine 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

5.3 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, das die Bedeutung beschreibt. Auf ein Menü wird verzichtet.

Die Angaben in den Schemadateien sind formatiert, zum Beispiel stehen Stufennummern oder Kommentare immer genau untereinander. Damit die Angaben in der Benutzerfläche auch untereinander stehen, musste für das Schema-Textfeld ein Monospace-Font verwendet (Courier New). Auch die Textfelder für Datenstrom und Ergebnis erhielten diesen Font, um das Erscheinungsbild einheitlich zu halten.

Beispiel Screenshots der Anwendung in der Entwicklungsphase mit Dummy-Daten befinden sich im Anhang [A.7: Screenshots der Anwendung](#) auf Seite 31.

5.4 Implementierung der Geschäftslogik

5.4.1 Grundschema der rekursive Methoden von AbstractNode/GroupNode

Group- und ValueNode erben von AbstractNode Methoden. Alle diese Methoden funktionieren nach dem gleichen Prinzip:

Ist die `AbstractNode` eine `GroupNode`, dann ruft sie für jedes seiner Kinder die gleiche Methode erneut auf (Rekursion). Aus den Rückgabewerten der Kinder wird ein Wert akkumuliert und dieser zurückgegeben. Eine `ValueNode` hat keine Kinder und kann direkt ein Wert zurückgeben. Bei `ValueNodes` und `GroupNodes` ohne Kindern enden die Rekursionen.

5.4.2 Parsen des Schemas in eine Baumstruktur

Da der Baum zwingend einen Wurzelknoten⁶ braucht, Schemas aber nicht zwingend einen haben, erhält der Baum einen künstlichen Wurzelknoten mit der (in Schemadateien nicht verwendeten) Stufennummer 0. Ohne Wurzelknoten würden auf höchster Ebene Geschwister ignoriert, oder es müsste statt mit `GroupNode` mit `List<GroupNode>` gearbeitet werden. Beides soll vermieden werden.

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.

Der Algorithmus macht folgendes: Der künstliche Wurzelknoten wird erstellt und auf den Stack gepusht. Aus den Schemazeilen wird eine `List<AbstractNode>` erstellt (unverbundene Nodes, das heißt `children` ist eine leere Liste für alle `GroupNodes`). Im nächsten Schritt werden die unverbundenen Nodes anhand ihrer Stufennummern an der richtige Position im Baum hinzugefügt. Dazu wird nacheinander jedes Element `x` der Node-Liste mit `stackTop` verglichen.

Es gibt drei Möglichkeiten, wie `x` zu `stack.Peek()` in Beziehung steht:

- er hat eine größere Stufennummer, dann ist `x` ein Kindknoten von `stack.Peek()`.
- er hat die gleiche Stufennummer, dann ist `x` ein Geschwisterknoten von `stack.Peek()`.
- er hat eine kleinere Stufennummer, dann ist `x` ein Kind eines Vorfahren von `stack.Peek()`. Der richtige Elternknoten für `x` muss erst noch gefunden werden.

Im Fall 1 gibt es vielleicht einen Baum unterhalb von `x`. `x` wird auf den Stack gepusht.

Fall 2 und 3 können gleich behandelt werden. Dass `x` kein Kindknoten von `stack.Peek()` ist, bedeutet, dass `stack.Peek()` keine weiteren Kinder hat und der Baum unterhalb von `stack.Peek()` vollständig ist. `StackTop` wird vom Stack gepoppt und (dem neuen) `StackTop` `repeatCount` mal als Kind hinzugefügt. Der Ablauf wird solange wiederholt, bis der richtige Elternknoten für `x` gefunden wurde. Danach wird `x` (genau wie in Fall 1) auf den Stack gepusht.

Der Ablauf geht oben weiter mit der Verarbeitung der nächsten Node.

Nachdem alle Nodes der Liste durchgegangen wurden, können noch Nodes auf dem Stack liegen. Genau wie in Fall 2/3 wird `StackTop` vom Stack gepoppt und dem neuen `StackTop` als Kind hinzugefügt bis der Stack leer ist oder nur noch den Wurzelknoten enthält. Der Wurzelknoten wird zurückgegeben.

⁶Knoten, der Vorfahr von allen anderen Knoten ist

5.4.3 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 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 36.

5.5 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 ludt die zehn 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 wurde die Funktion der Benutzeroberfläche getestet (korrekte Auswahl eines Schemas

7 Einführungsphase

anhand einer Vorgabe, speichern neuer Schemas, Löschen der XML-Konfigurationsdatei, Kopieren einer Datei in den Schema-Ordner). Nach Löschung der XML-Konfigurationsdatei trat bei der Auswahl des Menüpunktes "Config editieren" eine FileNotFoundException auf.

Beispiel Ein Auszug eines Unit Tests befindet sich im Anhang [A.9: Testfall und sein Aufruf auf der Konsole](#) auf Seite 35.

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 40. Die Entwicklerdokumentation wurde mittels PHPDoc⁷ automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse findet sich im Anhang [A.8: Entwicklerdokumentation](#) auf Seite 33.

8.1 Zwischenstand

Tabelle 8 zeigt den Zwischenstand nach der Dokumentation.

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

Tabelle 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?

⁷Vgl. ?

9 Fazit

- 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

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.

9 Fazit

- 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 geparsen Metadaten	3 h
3.8. Erstellen einer Übersichtsseite für ein einzelnes Modul	1 h
4. Nächtlichen Batchjob einrichten	1 h
Abnahmetest der Fachabteilung	1 h
1. Abnahmetest der Fachabteilung	1 h
Einführungsphase	1 h
1. Einführung/Benutzerschulung	1 h
Erstellen der Dokumentation	9 h
1. Erstellen der Benutzerdokumentation	2 h
2. Erstellen der Projektdokumentation	6 h
3. Programmdokumentation	1 h
3.1. Generierung durch PHPdoc	1 h
Pufferzeit	2 h
1. Puffer	2 h
Gesamt	70 h

A.2 Lastenheft (Auszug)

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

Die Anwendung muss folgende Anforderungen erfüllen:

1. Verarbeitung der Moduldaten

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

2. Darstellung der Daten

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

3. Sonstige Anforderungen

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

A.3 Use Case-Diagramm

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

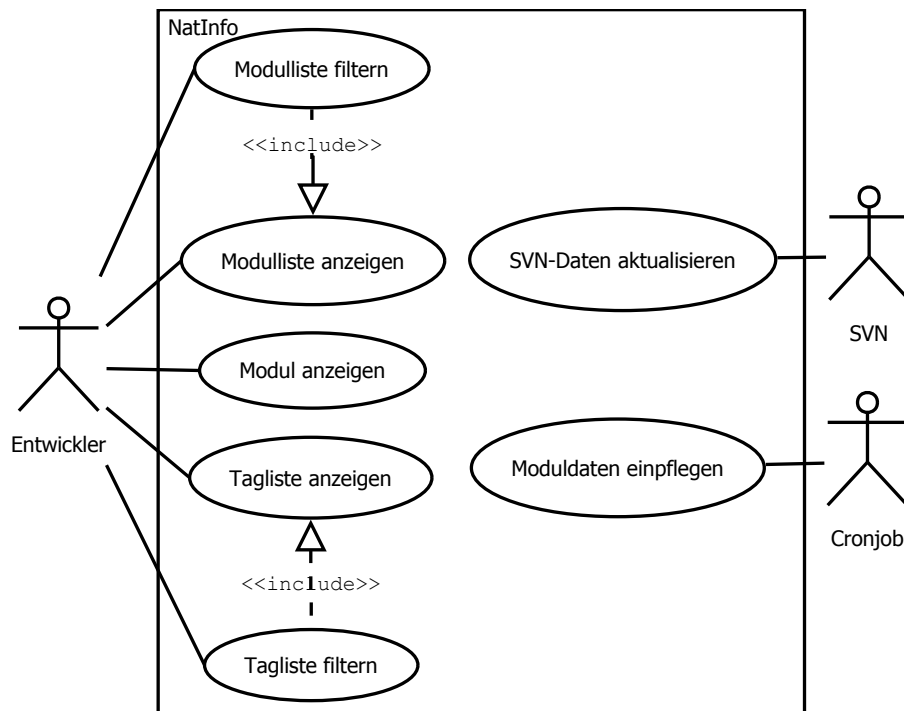


Abbildung 2: Use Case-Diagramm

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 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 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

A Anhang

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/>.

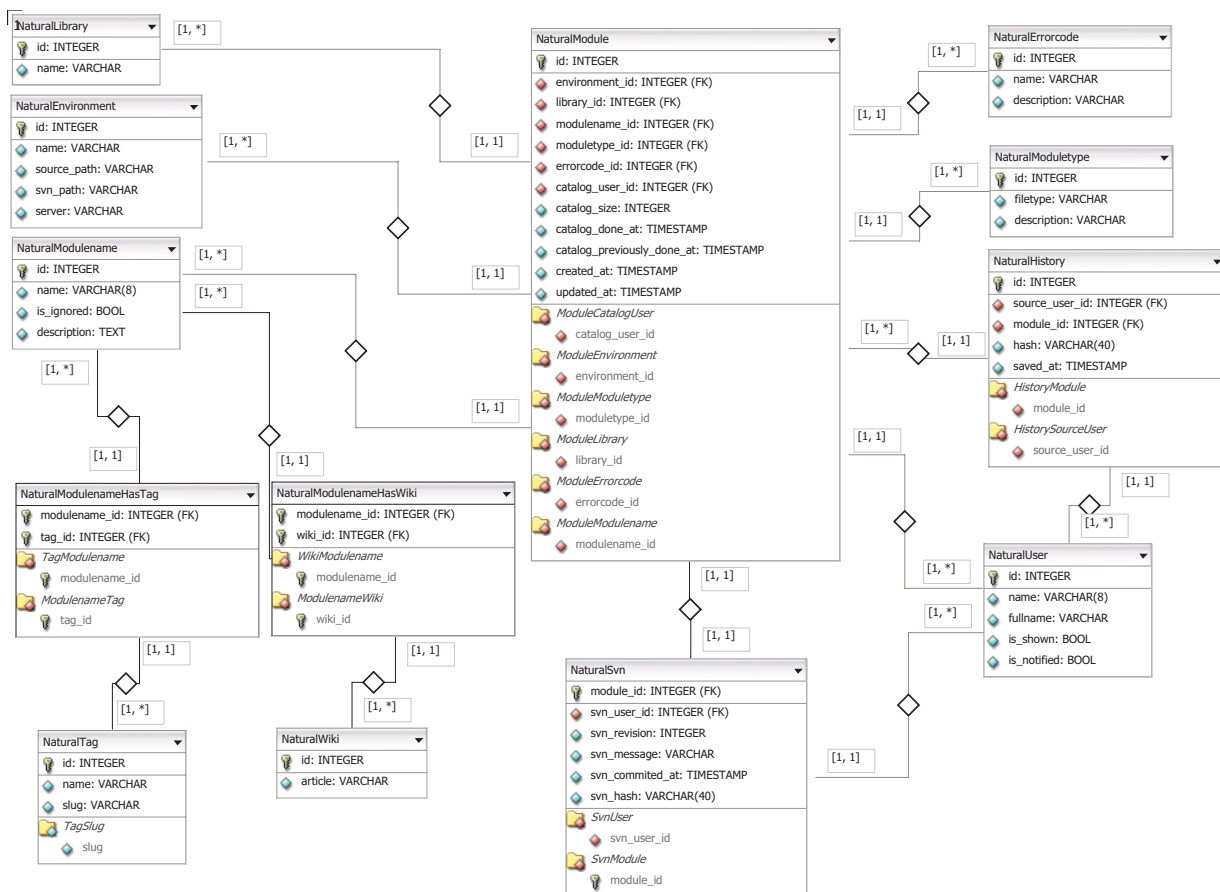


Abbildung 3: Datenbankmodell

[illegible]

Abbildung 4: Liste der Module mit Filtermöglichkeiten

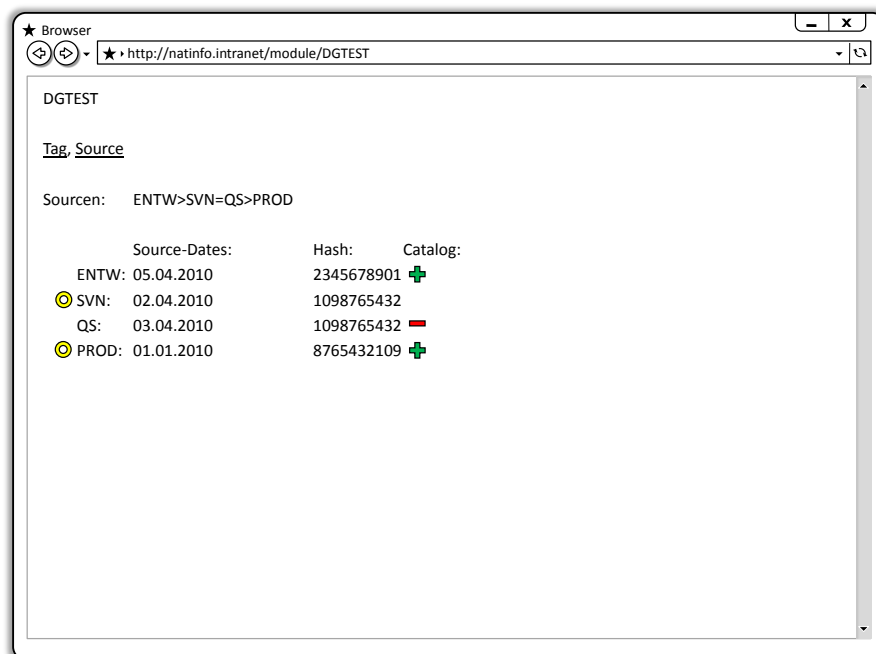


Abbildung 5: Anzeige der Übersichtsseite einzelner Module

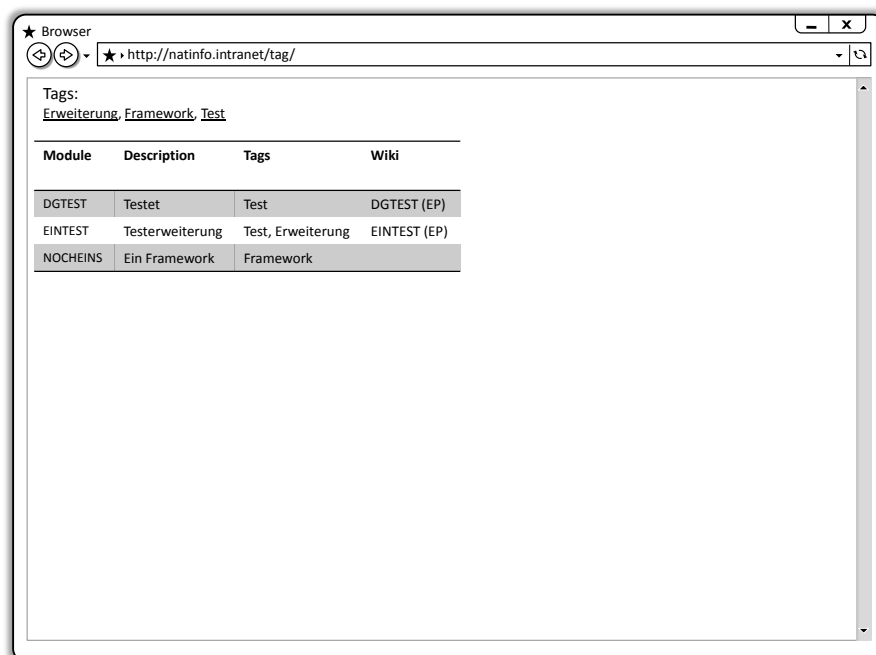


Abbildung 6: Anzeige und Filterung der Module nach Tags

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.

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:

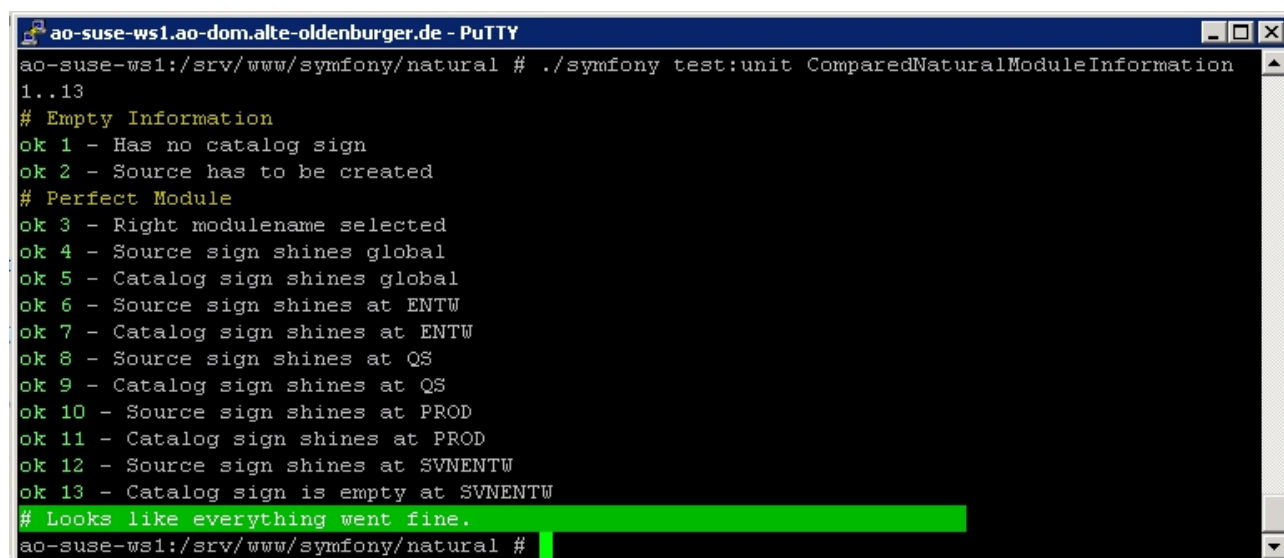
access: public

[\[Top \]](#)

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);
```

```
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>.

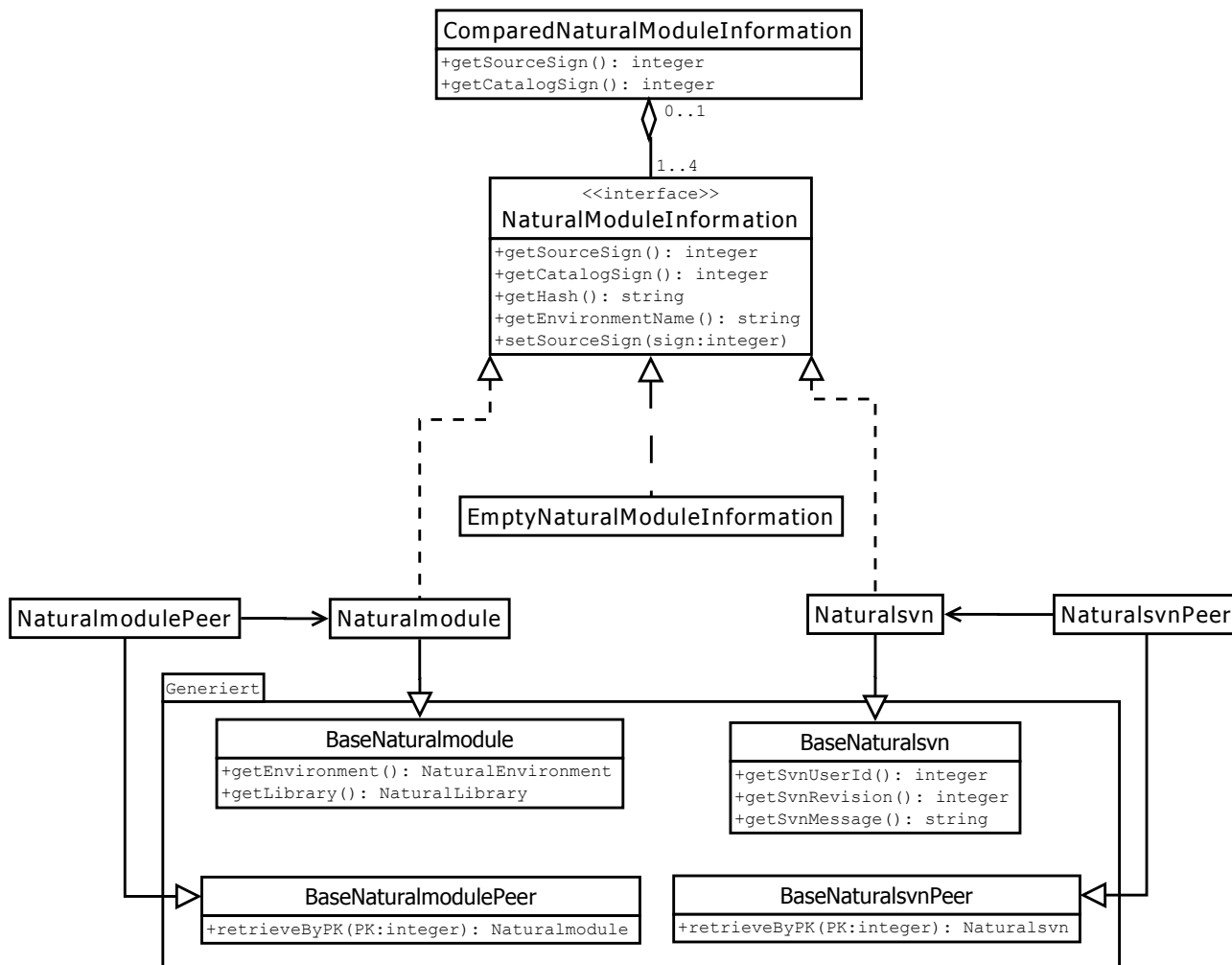







Abbildung 10: Klassendiagramm

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.