

## 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 Winter 2016

Fachinformatiker für Anwendungsentwicklung  
Dokumentation zur betrieblichen Projektarbeit

# Globales Aktualisieren von Dokumenten

Computergestützte Betriebsprüfung - Abschluss & Dokumente

Abgabetermin: Berlin, den 20.12.2016

**Prüfungsbewerber:**

Guido Eckelt  
Boddinstraße 30  
12053 Berlin



**Deutsche  
Rentenversicherung  
Bund**

**Ausbildungsbetrieb:**

DEUTSCHE RENTENVERSICHERUNG Bund  
Ruhrstraße 2  
10704 Berlin

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



## Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>III</b>
<b>Tabellenverzeichnis</b>	<b>IV</b>
<b>Listings</b>	<b>V</b>
<b>Abkürzungsverzeichnis</b>	<b>VI</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Projektumfeld . . . . .	1
1.2 Projektziel . . . . .	1
1.3 Projektbegründung . . . . .	2
1.4 Projektschnittstellen . . . . .	2
1.5 Projektabgrenzung . . . . .	2
<b>2 Projektplanung</b>	<b>3</b>
2.1 Projektphasen . . . . .	3
2.2 Abweichungen vom Projektantrag . . . . .	3
2.3 Ressourcenplanung . . . . .	3
2.4 Entwicklungsprozess . . . . .	4
<b>3 Analysephase</b>	<b>6</b>
3.1 Ist-Analyse . . . . .	6
3.2 Wirtschaftlichkeitsanalyse . . . . .	6
3.2.1 „Make or Buy“-Entscheidung . . . . .	6
3.2.2 Projektkosten . . . . .	6
3.2.3 Amortisationsdauer . . . . .	7
3.3 Nutzwertanalyse . . . . .	7
3.4 Anwendungsfälle . . . . .	7
3.5 Qualitätsanforderungen . . . . .	8
3.6 Lastenheft/Fachkonzept . . . . .	8
3.7 Zwischenstand . . . . .	8
<b>4 Entwurfsphase</b>	<b>8</b>
4.1 Zielplattform . . . . .	8
4.2 Architekturdesign . . . . .	9
4.3 Entwurf der Benutzeroberfläche . . . . .	9
4.4 Datenmodell . . . . .	9
4.5 Geschäftslogik . . . . .	10
4.6 Maßnahmen zur Qualitätssicherung . . . . .	10
4.7 Pflichtenheft/Datenverarbeitungskonzept . . . . .	11



*Inhaltsverzeichnis*

4.8	Zwischenstand . . . . .	11
<b>5</b>	<b>Implementierungsphase</b>	<b>11</b>
5.1	Implementierung der Datenstrukturen . . . . .	11
5.2	Implementierung der Benutzeroberfläche . . . . .	11
5.3	Implementierung der Geschäftslogik . . . . .	12
5.4	Zwischenstand . . . . .	12
<b>6</b>	<b>Abnahmephase</b>	<b>12</b>
6.1	Zwischenstand . . . . .	12
<b>7</b>	<b>Einführungsphase</b>	<b>13</b>
7.1	Zwischenstand . . . . .	13
<b>8</b>	<b>Dokumentation</b>	<b>13</b>
8.1	Zwischenstand . . . . .	14
<b>9</b>	<b>Fazit</b>	<b>14</b>
9.1	Soll-/Ist-Vergleich . . . . .	14
9.2	Lessons Learned . . . . .	15
9.3	Ausblick . . . . .	15
	<b>Literaturverzeichnis</b>	<b>16</b>
	<b>Eidesstattliche Erklärung</b>	<b>17</b>
<b>A</b>	<b>Anhang</b>	<b>i</b>
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	Vereinfachtes ER-Modell . . . . .	10
2	Prozess des Einlesens eines Moduls . . . . .	10
3	Use Case-Diagramm . . . . .	iii
4	Datenbankmodell . . . . .	v
5	Liste der Module mit Filtermöglichkeiten . . . . .	vi
6	Anzeige der Übersichtsseite einzelner Module . . . . .	vii
7	Anzeige und Filterung der Module nach Tags . . . . .	vii
8	Anzeige und Filterung der Module nach Tags . . . . .	viii
9	Liste der Module mit Filtermöglichkeiten . . . . .	ix
10	Aufruf des Testfalls auf der Konsole . . . . .	xiii
11	Klassendiagramm . . . . .	xvi



## **Tabellenverzeichnis**

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



## Listings

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



## **Abkürzungsverzeichnis**

<b>API</b>	Application Programming Interface
<b>HTTP</b>	Hypertext Transport Protocol
<b>SDK</b>	Software Development Kit
<b>ERM</b>	Entity-Relationship-Modell
<b>UML</b>	Unified Modeling Language
<b>GUI</b>	Graphical User Interface - Grafische Benutzeroberfläche
<b>SVN</b>	Subversion
<b>MVC</b>	Model View Controller
<b>PHP</b>	Hypertext Preprocessor
<b>SQL</b>	Structured Query Language
<b>EPK</b>	Ereignisgesteuerte Prozesskette
<b>XML</b>	Extensible Markup Language
<b>HTML</b>	Hypertext Markup Language
<b>CSV</b>	Comma Separated Value
<b>NatInfo</b>	Natural Information System
<b>Natural</b>	Programmiersprache der Software AG





## 1 Einleitung

### 1.1 Projektumfeld

Die DEUTSCHE RENTENVERSICHERUNG Bund ist ein bundesweit tätiger Träger der gesetzlichen Rentenversicherung in der Bundesrepublik Deutschland mit ca. 17000 Mitarbeitern.

Zum Aufgabenfeld zählen:

- Bearbeitung von Rentenanträgen
- Überprüfung von Sozialabgaben auf Richtigkeit
- Beratung zu gesetzlichen Pflichten und Rechten

Für die Überprüfung von Sozialabgaben entwickelt die IT-Abteilung der DEUTSCHE RENTENVERSICHERUNG Bund verschiedene Anwendung, um diesen Prozess elektronisch zu vereinfachen.

Computergestützte Betriebsprüfung - Nachberechnung ist eine Desktopanwendung, mit der Differenzen bei den Sozialabgaben von Betrieben berechnet werden können. Sie ist in VB.NET mit einer WindowsForms-UserInterface implementiert. Computergestützte Betriebsprüfung - Abschluss & Dokumente ist eine Desktopanwendung, mit der jeweilige benötigte Dokumente, Anlagen und Schreiben für Betriebsprüfung erzeugt werden können. Sie ist in VB.NET mit einer WindowsForms-UserInterface implementiert.

### 1.2 Projektziel

Die Betriebsprüfer erstellen und bearbeiten Dokumente und Anlagen, die mit Daten von Berechnungen aus der Anwendung Computergestützte Betriebsprüfung - Nachberechnung befüllt werden. Wenn in dieser Daten verändert werden, die schon in erzeugten Dokumenten vorkommen, sind diese Dokumente in einem „asynchronen“ Zustand und müssen vor Weiterverwendung aktualisiert werden.

Im Hauptmenü der Computergestützte Betriebsprüfung - Abschluss & Dokumente soll ein neuer Menüeintrag bereitgestellt werden, dessen Kommando einen Aktualisierungsprozess anstößt, der alle Dokumente auf Asynchronität prüft und anschließend veraltete aktualisiert.

Für die verschiedenen Dokumenttypen gibt es zurzeit auch noch unterschiedliche Strukturen, wie die jeweiligen Dokumente neu erzeugt werden. Für die Funktionalität „Globales Aktualisieren“ sollen nun alle Dokumenttypen auf eine Struktur umgebaut werden.

Erzeugungsstrukturen für einige Dokumenttypen berechnen ihren Fortschritt eigenständig und geben diesen in eigenen Fenstern aus. Für diese soll eine Möglichkeit der Unterdrückung dieser Fortschrittsausgabe implementiert werden, damit der Aktualisierungsprozess „Globales Aktualisieren“ dies einheitlich für alle Dokumenttypen ausgeben kann.



### 1.3 Projektbegründung

Durch diese Erweiterung wird eine Vereinheitlichung der Dokumentenaktualisierung erreicht, die zugleich eine erhebliche Vereinfachung für den Anwender mit sich bringt.

### 1.4 Projektschnittstellen

Daten aus den Berechnungen der Sozialabgaben von Betrieben und ihren Mitarbeitern werden über eine [HTTP-API](#) der Computergestützte Betriebsprüfung - Nachberechnung angefordert.

Die Benutzer der Anwendung sind die Betriebsprüfer der DEUTSCHE RENTENVERSICHERUNG Bund.

### 1.5 Projektabgrenzung

Dieses Projekt zur Erweiterung der Computergestützte Betriebsprüfung - Abschluss & Dokumente ist unabhängig von der Entwicklung der Computergestützte Betriebsprüfung - Nachberechnung, da nur bereits festgelegte Schnittstellen zum Datenaustausch benutzt werden und keine Änderung dieser notwendig sind.



## 2 Projektplanung

### 2.1 Projektphasen

Die (Vorbereitungs-)Projektphase begann am 17.10.2016 und endete am 16.12.2016. Die tägliche Arbeitszeit betrug 7 Stunden 48 Minuten mit 30 Minuten Mittagspause. Die Projektarbeit fand nicht durchgängig statt, da Betriebsinterne Ereignisse berücksichtigt werden müssen.

Projektphase	Teilzeit	Gesamtzeit
1. Analyse		5 h
1.1 Ist-Zustand	1 h	
1.2 Pflichtenheft	2 h	
1.3 Wirtschaftlichkeitsanalyse	2 h	
2. Entwurf		5 h
2.1 Klassendiagramm zur Architektur	3 h	
2.2 Sequenzdiagramm zur Abfolge	2 h	
3. Implementierungsphase		45 h
3.1 Aktualisierungsprozess „Globales Aktualisieren“	25 h	
3.2 Umbau der Dokumentenerzeugung	10 h	
3.3 Fortschrittsausgabe vereinheitlichen	10 h	
4. Qualitätssicherung		5 h
4.1 Unit-Tests	3 h	
4.2 Abnahme	2 h	
5. Dokumentation		10 h
5.1 Projektdokumentation	6 h	
5.2 Programmdokumentation	4 h	
<b>Gesamt</b>		<b>70 h</b>

Tabelle 1: Zeitplanung

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

### 2.2 Abweichungen vom Projektantrag

Entwurfsphase mit 5 Stunden ist dazu gekommen, da eine Planung her muss.

Implementierungsphase ist daher um gleiche Zeitspanne verringert.

### 2.3 Ressourcenplanung

Benötigte Ressourcen:

- Büro mit Tisch, Stuhl, Stromanschlüsse
- Arbeitsmaschine 1 mit Windows7



- dotNET-Framework + Visual Studio für Arbeitsmaschine 1
- Projektspezifische SDKs für Visual Studio
- Arbeitsmaschine 2 mit Entwicklungsnetz-Zugang + Kartenleser
- microTool inStep - (Projektverwaltungstool) für Arbeitsmaschine 2
- Projektbetreuer zur Unterstützung

## 2.4 Entwicklungsprozess

Die ausgewählte Vorgehensweise ist das Wasserfall-Modell<sup>1</sup>.

### 1. Systemanforderungen:

Alle Anforderungen, die selbst nicht direkt das Software-Produkt betreffen, werden zunächst festgelegt. Dazu zählen:

- Preis
- Verfügbarkeit
- Sicherheitsaspekte
- Dokumentation

### 2. Softwareanforderungen:

Alle Anforderung an die Software selber werden definiert. Jegliche Funktionen, Interaktionen und Besonderheiten werden konkretisiert, so dass sich aus den Systemanforderungen und Softwareanforderungen das Lastenheft ergibt.

### 3. Analyse:

Anforderungen aus Lastenheft und Ist-Zustand der Situation werden analysiert, so dass diese in ein Pflichtenheft umformuliert werden können. Die Wirtschaftlichkeit eines Projektes wird ebenfalls hier geprüft.

### 4. Entwurf:

Es wird das Datenmodell, die Architektur und die Schnittstellen zu anderen Anwendungen herausgearbeitet. Zwischenergebnisse:

- Entity-Relationship-Modell ([ERM](#))
- [UML](#)-Diagramme (Klassendiagramm, Sequenzdiagramm, Anwendungsfalldiagramm usw.)
- Mockups zur [GUI](#)
- Schnittstellen-Verzeichnis

---

<sup>1</sup>Wasserfall-Modell nach W. Royce



**5. Implementierung:**

Umsetzung der Funktionalitäten nach Pflichtenheft und Entwurf in eine lauffähige Anwendung.

**6. Test/Qualitätssicherung:**

Es wird nach der Implementierungsphase die Software auf Fehler, Schwachstellen und Unstimmigkeiten überprüft. weitverbreitete Testvorgänge:

- Komponententests (Unit-Test)
- Modultests
- Systemtests
- Integrationstests

**7. Inbetriebnahme:**

letzter Schritt in der Softwareentwicklung. Bei fehlerloser Überprüfung kann die Anwendung abgenommen werden und in Produktion gehen.

Alle Schritte im Wasserfall-Modell sind sequentiell zu bearbeiten, d. h. Kein Schritt darf vorgezogen werden oder übersprungen werden. Es dürfen nur Phasen zurückgesprungen werden, Wenn eine Phase nicht abgeschlossen werden kann.



## 3 Analysephase

### 3.1 Ist-Analyse

Die Betriebsprüfer müssen die Dokumente einzeln über den Dokumenttypenbaum aktualisieren. Dies kann sehr aufwendig sein, da manche Berechnungen in mehreren Dokumenten aufgelistet werden und sind diese erst ausfindig zu machen, um dann aktualisiert werden zu können.

Diese Funktionalität ist von den Betriebsprüfern dringend erwünscht, da es eine enorme Zeitersparnis für sie ergeben würde.

### 3.2 Wirtschaftlichkeitsanalyse

#### 3.2.1 „Make or Buy“-Entscheidung

Da die Entwicklung der Computergestützte Betriebsprüfung - Abschluss & Dokumente ein internes Projekt der DEUTSCHE RENTENVERSICHERUNG Bund ist und dieses Projekt nur eine Funktionserweiterung ist, lässt sich kein fertiges Produkt finden, dass alle Anforderungen, vor allem fachliche, umsetzt. Daher wird dieses Projekt nun umgesetzt

#### 3.2.2 Projektkosten

**Beispielrechnung (verkürzt)** Die Kosten für die Durchführung des Projekts setzen sich sowohl aus Personal-, als auch aus Ressourcenkosten zusammen. Laut Tarifvertrag verdient ein Auszubildender im dritten Lehrjahr pro Monat 1000 € Brutto.

$$8 \text{ h/Tag} \cdot 220 \text{ Tage/Jahr} = 1760 \text{ h/Jahr} \quad (1)$$

$$1000 \text{ €/Monat} \cdot 13,3 \text{ Monate/Jahr} = 13300 \text{ €/Jahr} \quad (2)$$

$$\frac{13300 \text{ €/Jahr}}{1760 \text{ h/Jahr}} \approx 7,56 \text{ €/h} \quad (3)$$

Es ergibt sich also ein Stundenlohn von 7,56 €. Die Durchführungszeit des Projekts beträgt 70 Stunden. Für die Nutzung von Ressourcen<sup>2</sup> wird ein pauschaler Stundensatz von 15 € angenommen. Für die anderen Mitarbeiter wird pauschal ein Stundenlohn von 25 € angenommen. Eine Aufstellung der Kosten befindet sich in Tabelle 2 und sie betragen insgesamt 2739,20 €.

---

<sup>2</sup>Räumlichkeiten, Arbeitsplatzrechner etc.



Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	70 h	7,56 € + 15 € = 22,56 €	1579,20 €
Fachgespräch	3 h	25 € + 15 € = 40 €	120 €
Abnahmetest	1 h	25 € + 15 € = 40 €	40 €
Anwenderschulung	25 h	25 € + 15 € = 40 €	1000 €
			<b>2739,20 €</b>

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 10 Minuten am Tag für jeden der 25 Anwender und 220 Arbeitstagen im Jahr ergibt sich eine gesamte Zeiteinsparung von

$$25 \cdot 220 \text{ Tage/Jahr} \cdot 10 \text{ min/Tag} = 55000 \text{ min/Jahr} \approx 917 \text{ h/Jahr} \quad (4)$$

Dadurch ergibt sich eine jährliche Einsparung von

$$917 \text{ h} \cdot (25 + 15) \text{ €/h} = 36680 \text{ €} \quad (5)$$

Die Amortisationszeit beträgt also  $\frac{2739,20 \text{ €}}{36680 \text{ €/Jahr}} \approx 0,07 \text{ Jahre} \approx 4 \text{ Wochen}$ .

### 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 4.2: [Architekturdesign](#).

### 3.4 Anwendungsfälle

- Welche Anwendungsfälle soll das Projekt abdecken?
- Einer oder mehrere interessante (!) Anwendungsfälle könnten exemplarisch durch ein Aktivitätsdiagramm oder eine Ereignisgesteuerte Prozesskette (EPK) detailliert beschrieben werden.



**Beispiel** Ein Beispiel für ein Use Case-Diagramm findet sich im Anhang [A.3: Use Case-Diagramm](#) auf Seite [iii](#).

### 3.5 Qualitätsanforderungen

- Welche Qualitätsanforderungen werden an die Anwendung gestellt (z. B. hinsichtlich Performance, Usability, Effizienz etc. (siehe [ISO/IEC 9126-1 \[2001\]](#)))?

### 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.

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

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





## 4.2 Architekturdesign

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

**Beispiel** Anhand der Entscheidungsmatrix in Tabelle 4 wurde für die Implementierung der Anwendung das [PHP](#)-Framework [Symfony](#)<sup>3</sup> ausgewählt.

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

Tabelle 4: Entscheidungsmatrix

## 4.3 Entwurf der Benutzeroberfläche

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

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

## 4.4 Datenmodell

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

**Beispiel** In [Abbildung 1](#) wird ein [ERM](#) dargestellt, welches lediglich Entitäten, Relationen und die dazugehörigen Kardinalitäten enthält.

---

<sup>3</sup>Vgl. [SENSIO LABS](#) [2010].



#### 4 Entwurfsphase

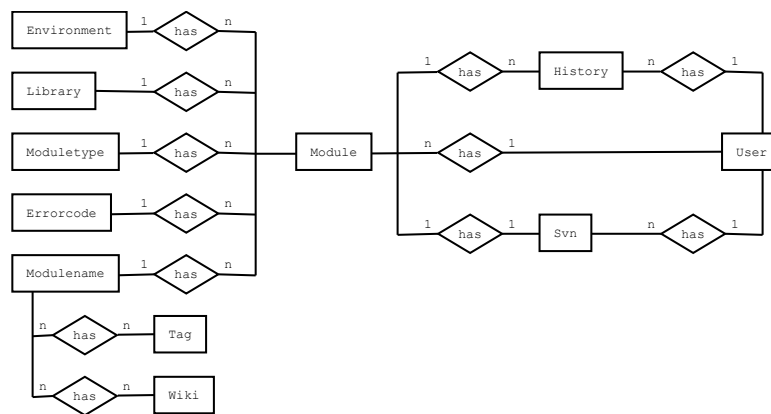


Abbildung 1: Vereinfachtes ER-Modell

### 4.5 Geschäftslogik

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

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

[Abbildung 2](#) zeigt den grundsätzlichen Programmablauf beim Einlesen eines Moduls als [EPK](#).

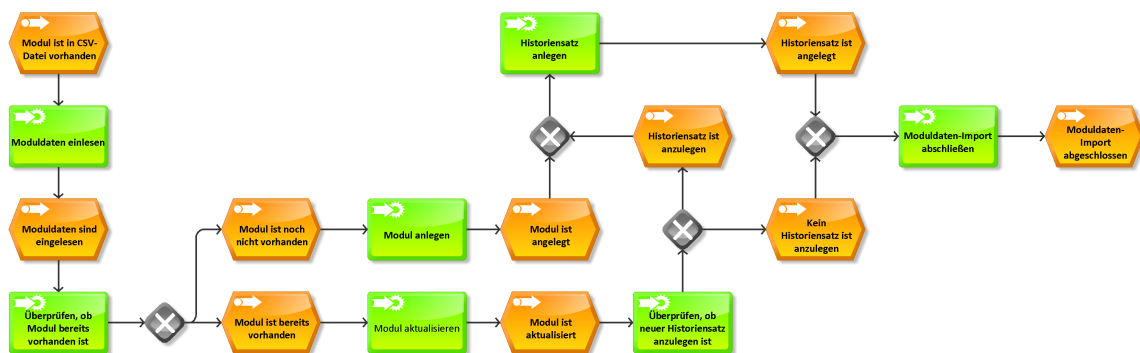


Abbildung 2: Prozess des Einlesens eines Moduls

### 4.6 Maßnahmen zur Qualitätssicherung

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



## 4.7 Pflichtenheft/Datenverarbeitungskonzept

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

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

## 4.8 Zwischenstand

Tabelle 5 zeigt den Zwischenstand nach der Entwurfsphase.

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 5: Zwischenstand nach der Entwurfsphase

# 5 Implementierungsphase

## 5.1 Implementierung der Datenstrukturen

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

## 5.2 Implementierung der Benutzeroberfläche

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

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



### 5.3 Implementierung der Geschäftslogik

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

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

### 5.4 Zwischenstand

Tabelle 6 zeigt den Zwischenstand nach der Implementierungsphase.

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

Tabelle 6: Zwischenstand nach der Implementierungsphase

## 6 Abnahmephase

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

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

### 6.1 Zwischenstand

Tabelle 7 zeigt den Zwischenstand nach der Abnahmephase.



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

Tabelle 7: Zwischenstand nach der Abnahmephase

## 7 Einführungsphase

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

### 7.1 Zwischenstand

Tabelle 8 zeigt den Zwischenstand nach der Einführungsphase.

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

Tabelle 8: 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 PHPDoc<sup>4</sup> automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse findet sich im Anhang [A.8: Entwicklerdokumentation](#) auf Seite [x](#).

---

<sup>4</sup>Vgl. [PHPDOC.ORG](http://PHPDOC.ORG) [2010]



## 8.1 Zwischenstand

Tabelle 9 zeigt den Zwischenstand nach der Dokumentation.

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

Tabelle 9: Zwischenstand nach der Dokumentation

## 9 Fazit

### 9.1 Soll-/Ist-Vergleich

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

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

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

Tabelle 10: Soll-/Ist-Vergleich



## 9.2 Lessons Learned

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

## 9.3 Ausblick

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



## **Literaturverzeichnis**

### **ISO/IEC 9126-1 2001**

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

### **phpdoc.org 2010**

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

### **Sensio Labs 2010**

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





## **Eidesstattliche Erklärung**

Ich, Guido Eckelt, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

*Globales Aktualisieren von Dokumenten – Computergestützte Betriebsprüfung - Abschluss  
& Dokumente*

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.

Berlin, den 20.12.2016

---

GUIDO ECKELT



## A Anhang

### A.1 Detaillierte Zeitplanung

<b>Analysephase</b>	<b>9 h</b>
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
<b>Entwurfsphase</b>	<b>19 h</b>
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
<b>Implementierungsphase</b>	<b>29 h</b>
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
<b>Abnahmetest der Fachabteilung</b>	<b>1 h</b>
1. Abnahmetest der Fachabteilung	1 h
<b>Einführungsphase</b>	<b>1 h</b>
1. Einführung/Benutzerschulung	1 h
<b>Erstellen der Dokumentation</b>	<b>9 h</b>
1. Erstellen der Benutzerdokumentation	2 h
2. Erstellen der Projektdokumentation	6 h
3. Programmdokumentation	1 h
3.1. Generierung durch PHPdoc	1 h
<b>Pufferzeit</b>	<b>2 h</b>
1. Puffer	2 h
<b>Gesamt</b>	<b>70 h</b>



## 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  $\text{\LaTeX}$  zeichnen, siehe z. B. <http://metauml.sourceforge.net/old/usecase-diagram.html>.

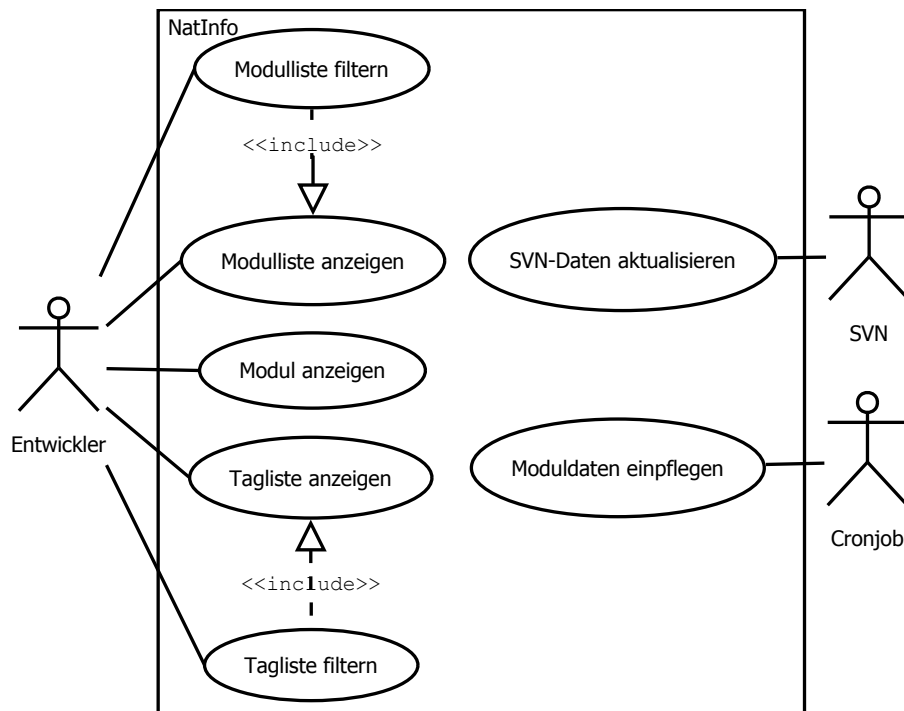


Abbildung 3: Use Case-Diagramm

### 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**-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  $\text{\LaTeX}$  zeichnen, siehe z. B. <http://www.texample.net/tikz/examples/entity-relationship-diagram/>.

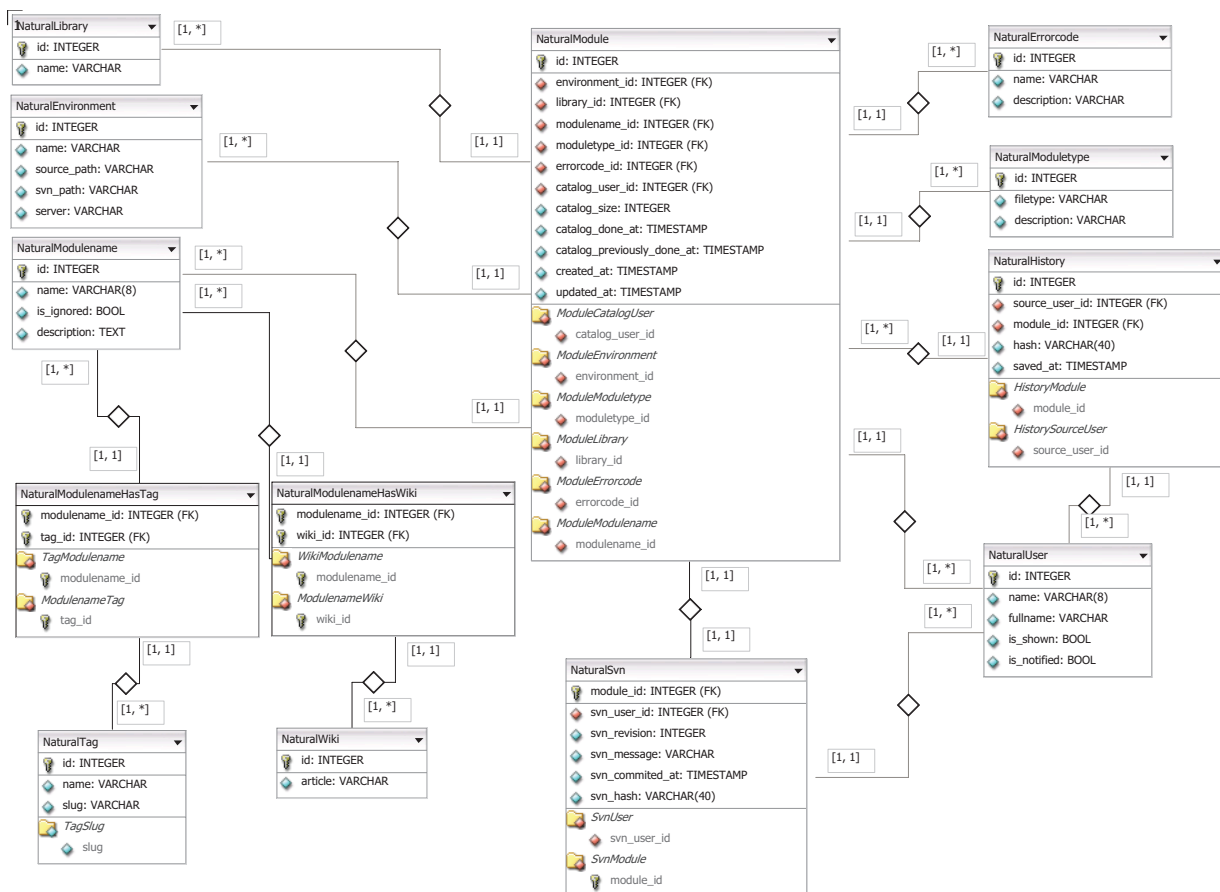


Abbildung 4: Datenbankmodell



## A.6 Oberflächenentwürfe

★ Browser

★ http://natinfo.intranet/module/

Filter:  
 Source- and Catalog-Information  
 from Environment:

Library: Select

Filter

Source:  
 Date: Select  
 Username: Select

Catalog:  
 Date: Select  
 Username: Select

Module	Library	Sourcen	Catalog	Source-User	Source-Date	Catalog-User	Catalog-Date
DGTEST	UTILITY	+	+	Grashorn	01.04.2010	Grashorn	02.04.2010
EINTEST	SYSTEM	-	+	Mustermann	01.04.2010	Grashorn	02.04.2010
NOCHEINS	UTILITY	○	-	Grashorn	05.04.2010	Grashorn	05.04.2010
MANUEL	SYSTEM	○	+	Grashorn	01.03.2010	Grashorn	01.03.2010
RESET	CON	-	+	Mustermann	02.03.2010	Mustermann	02.03.2010
TESTER	SYSTEM	+	-	Grashorn	01.02.2010	Grashorn	01.02.2010
...	...	...	...	...	...	...	...

Abbildung 5: Liste der Module mit Filtermöglichkeiten

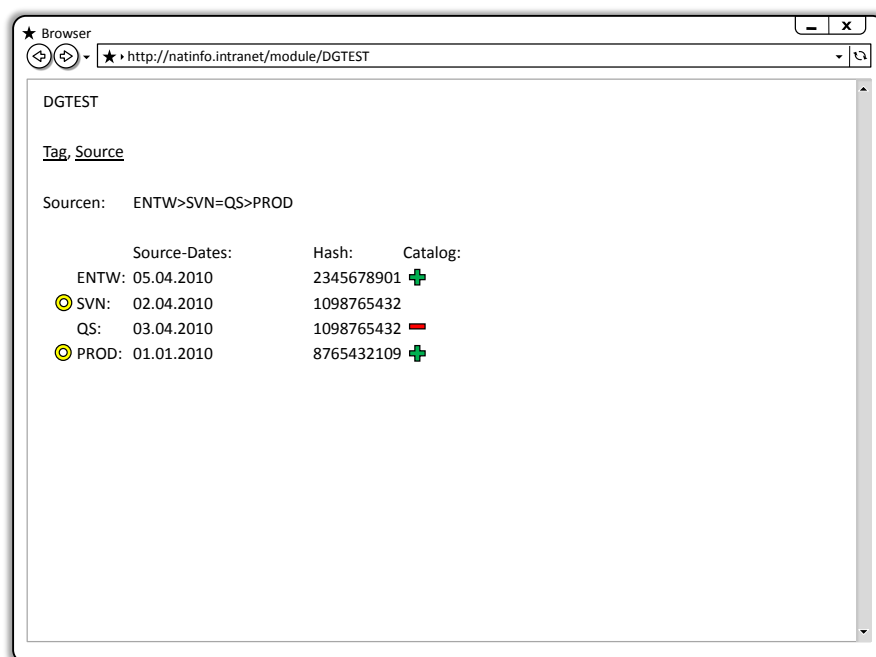


Abbildung 6: Anzeige der Übersichtsseite einzelner Module

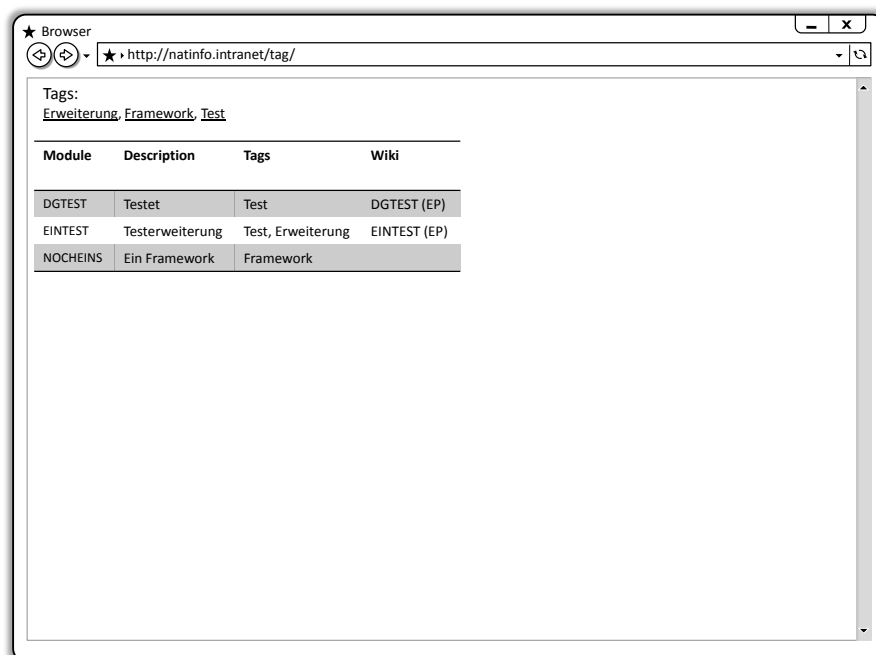


Abbildung 7: Anzeige und Filterung der Module nach Tags





## 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 8: Anzeige und Filterung der Module nach Tags





Häufig benötigte Seiten

[Ein-Klick-Menü](#)

Direktaufruf

[↻](#)

Inhaltssuche

[🔍](#)

 / Modules / ENTW
Tags, Modules

## Modules

<b>Environment</b>	ENTW
<b>Library</b>	Select
<b>Catalog user</b>	Select
<b>Catalog date</b>	Select
<b>Source user</b>	Select
<b>Source date</b>	Select
<a href="#">Reset</a> <a href="#">Filter</a>	

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 9: 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:/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 10: Aufruf des Testfalls auf der Konsole

## A.10 Klasse: ComparedNaturalModuleInformation

Kommentare und simple Getter/Setter werden nicht angezeigt.

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



A Anhang

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



A Anhang

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





## A.11 Klassendiagramm

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

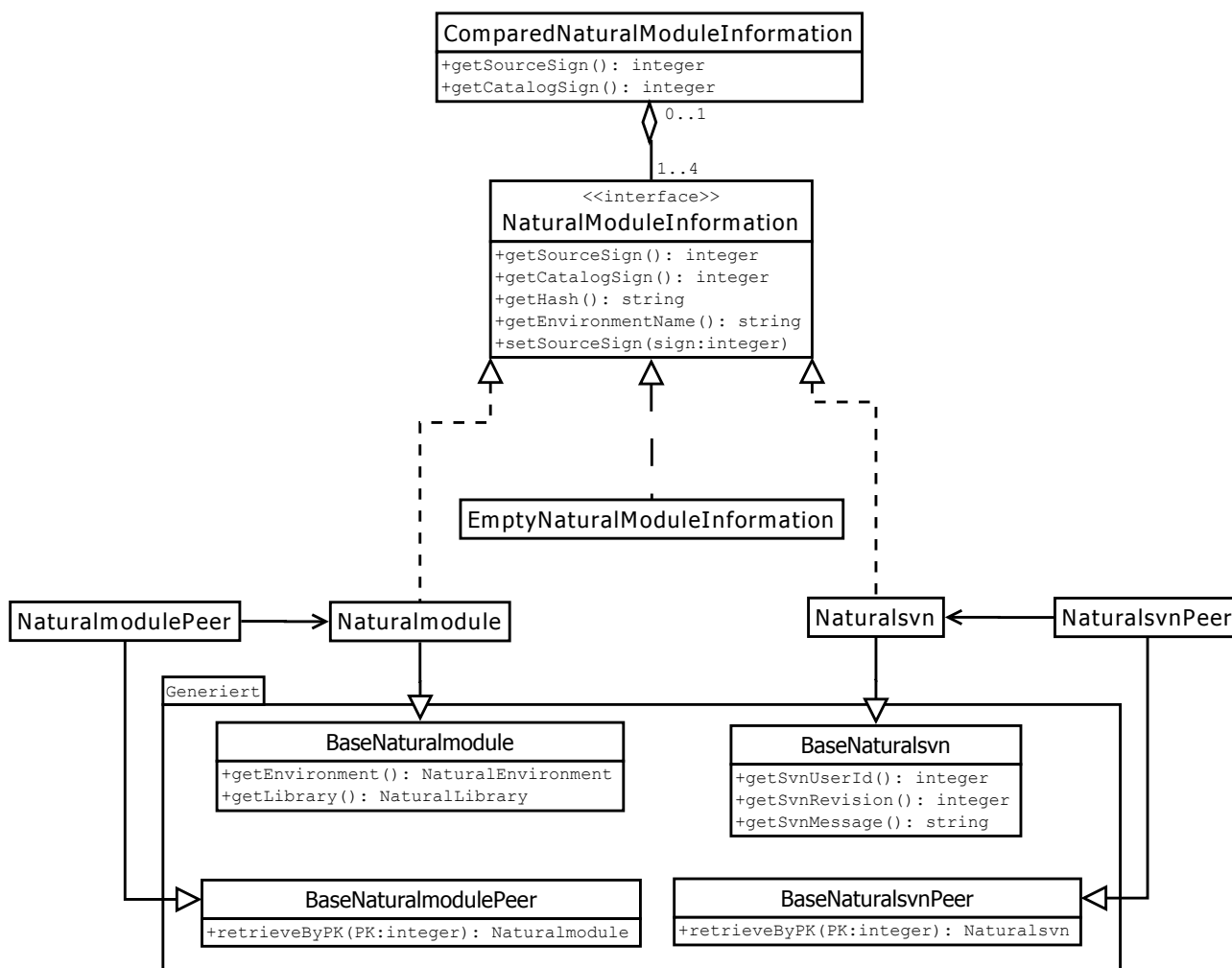







Abbildung 11: Klassendiagramm



## 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.