



Abschlussprüfung Sommer 2020

Fachinformatiker für Systemintegration  
Dokumentation zur betrieblichen Projektarbeit

# Evaluierung von „Icinga“

## Open Source Monitoring

Abgabetermin: Augsburg, den 21.05.2020

**Prüfungsbewerber:**

Andreas Germer  
Hagelbach 9  
86316 Bachern



**Ausbildungsbetrieb:**

KUKA AG  
Zugspitzstraße 144  
86163 Augsburg

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 . . . . .	1
1.4 Projektabgrenzung . . . . .	1
<b>2 Projektplanung</b>	<b>2</b>
2.1 Projektphasen . . . . .	2
2.2 Abweichungen vom Projektantrag . . . . .	2
2.3 Ressourcenplanung . . . . .	2
<b>3 Planungsphase</b>	<b>3</b>
3.1 Ist-Analyse . . . . .	3
3.2 Soll-Analyse . . . . .	3
3.2.1 Befragung der Fachabteilungen . . . . .	3
3.2.2 Kriterienkatalog . . . . .	4
3.3 Wirtschaftlichkeitsanalyse . . . . .	4
3.3.1 Projektkosten . . . . .	4
3.3.2 Betriebskosten des Monitorings . . . . .	5
3.3.3 Amortisationsdauer . . . . .	5
<b>4 Entwurfsphase</b>	<b>6</b>
4.1 Zielplattform . . . . .	6
4.2 Architekturdesign . . . . .	6
4.3 Entwurf der Benutzeroberfläche . . . . .	6
4.4 Datenmodell . . . . .	7
4.5 Geschäftslogik . . . . .	7
4.6 Maßnahmen zur Qualitätssicherung . . . . .	8
4.7 Pflichtenheft/Datenverarbeitungskonzept . . . . .	8
<b>5 Implementierungsphase</b>	<b>8</b>
5.1 Implementierung der Datenstrukturen . . . . .	8
5.2 Implementierung der Benutzeroberfläche . . . . .	9

5.3	Implementierung der Geschäftslogik . . . . .	9
<b>6</b>	<b>Abnahmephase</b>	<b>9</b>
<b>7</b>	<b>Einführungsphase</b>	<b>10</b>
<b>8</b>	<b>Dokumentation</b>	<b>10</b>
<b>9</b>	<b>Fazit</b>	<b>10</b>
9.1	Soll-/Ist-Vergleich . . . . .	10
9.2	Lessons Learned . . . . .	11
9.3	Ausblick . . . . .	11
	<b>Quellenverzeichnis</b>	<b>12</b>
	<b>Eidesstattliche Erklärung</b>	<b>13</b>
<b>A</b>	<b>Anhang</b>	<b>i</b>
A.1	Berechnung des Stundensatzes von Mitarbeitenden . . . . .	i
A.2	Detaillierte Zeitplanung . . . . .	ii
A.3	Lastenheft (Auszug) . . . . .	iii
A.4	Use Case-Diagramm . . . . .	iv
A.5	Pflichtenheft (Auszug) . . . . .	iv
A.6	Datenbankmodell . . . . .	vi
A.7	Oberflächenentwürfe . . . . .	vii
A.8	Screenshots der Anwendung . . . . .	ix
A.9	Entwicklerdokumentation . . . . .	xi
A.10	Testfall und sein Aufruf auf der Konsole . . . . .	xiii
A.11	Klasse: ComparedNaturalModuleInformation . . . . .	xiv
A.12	Klassendiagramm . . . . .	xvii
A.13	Benutzerdokumentation . . . . .	xviii

## **Abbildungsverzeichnis**

1	Vereinfachtes ER-Modell . . . . .	7
2	Prozess des Einlesens eines Moduls . . . . .	8
3	Use Case-Diagramm . . . . .	iv
4	Datenbankmodell . . . . .	vi
5	Liste der Module mit Filtermöglichkeiten . . . . .	vii
6	Anzeige der Übersichtsseite einzelner Module . . . . .	viii
7	Anzeige und Filterung der Module nach Tags . . . . .	viii
8	Anzeige und Filterung der Module nach Tags . . . . .	ix
9	Liste der Module mit Filtermöglichkeiten . . . . .	x
10	Aufruf des Testfalls auf der Konsole . . . . .	xiv
11	Klassendiagramm . . . . .	xvii

## **Tabellenverzeichnis**

1	Zeitplanung . . . . .	2
2	Kostenaufstellung . . . . .	5
3	Kosten Monitoring . . . . .	5
4	Entscheidungsmatrix . . . . .	7
5	Soll-/Ist-Vergleich . . . . .	11

Listings

1	Testfall in PHP . . . . .	xiii
2	Klasse: ComparedNaturalModuleInformation . . . . .	xiv

## **Abkürzungsverzeichnis**

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

## 1 Einleitung

### 1.1 Projektumfeld

Die KUKA AG ist ein international tätiges Unternehmen in der Maschinenbau- und Automatisierungsbranche mit rund 14.200 Mitarbeitern. Zum Produktportfolio zählen neben Industrierobotern auch die Planung und der Bau von Produktionsstraßen.

Die IT-Infrastruktur am Standort Augsburg besteht aus ca. 1.200 größtenteils virtualisierten Servern; an anderen Standorten befindet sich vereinzelt eine kleine Anzahl an Servern. Es kommen alle gängigen Betriebssysteme zum Einsatz. Auftraggeber dieses Projekts ist die Abteilung „Datacenter & Network“, die den Betrieb der Rechenzentren sowie des internen IT-Netzwerks der KUKA AG überwacht und koordiniert.

### 1.2 Projektziel

Momentan wird kein einheitliches Monitoring der IT-Systeme durchgeführt. Die verschiedenen Standorte und Abteilungen setzen auf unterschiedliche und teils veraltete Lösungen; zur besseren Administration sollen diese durch ein einheitliches und zentrales System ersetzt werden.

Es soll evaluiert werden, ob die freie Monitoringsoftware „Icinga“ für den firmeninternen Einsatz geeignet ist. Dazu werden in einem ersten Schritt Anforderungen der Process Owner und Systemadministratoren gesammelt. Anschließend wird eine Testumgebung eingerichtet, um zu prüfen, inwieweit die Ansprüche des Unternehmens durch „Icinga“ erfüllt werden können. Abschließend werden die Ergebnisse analysiert und eine Empfehlung ausgesprochen.

### 1.3 Projektbegründung

Durch die zunehmende Digitalisierung aller Geschäftsprozesse sind Unternehmen äußerst abhängig von Computersystemen. Wird das Schutzziel der Verfügbarkeit nicht ausreichend gut verfolgt, kommt es zu Systemausfällen und der Geschäftsbetrieb ist nicht mehr möglich - mit unabsehbaren wirtschaftlichen Folgen. Um Ausfallsicherheit zu gewährleisten, ist ein umfangreiches und zuverlässiges Monitoring aller Ressourcen unerlässlich.

### 1.4 Projektabgrenzung

Als Monitoringsystem interagiert „Icinga“ potenziell mit allen Geräten und Diensten im Netzwerk. Für dieses Projekt wird das Zusammenspiel auf virtuelle Maschinen mit ausgewählten Betriebssystemen und Diensten eingegrenzt.



## 2 Projektplanung

### 2.1 Projektphasen

Das Projekt wird innerhalb einer 35-Stunden-Arbeitswoche durchgeführt. Die einzelnen Phasen werden sich teils überschneiden; beispielsweise können virtuelle Maschinen bereits eingerichtet werden, während noch auf Rückmeldungen bezüglich der Anforderungen an das System gewartet wird.

Planungsphase		Durchführungsphase		Projektabschluss	
Ist-Analyse	2h	Einrichtung Wirt und VMs	6h	Ergebnisbewertung	1h
Soll-Analyse	2h	Installation Icinga	2h	Soll-Ist-Vergleich	1h
Definierung Kriterienkatalog	3h	Installation Testsysteme	3h	Empfehlungsformulierung	1h
		Testdurchführung	7h	Dokumentation	7h

Tabelle 1: Zeitplanung

### 2.2 Abweichungen vom Projektantrag

Es gab keine Abweichungen vom Projektantrag.

### 2.3 Ressourcenplanung

Folgende Hardware wird zur Durchführung des Projekts benötigt:

- Notebook mit RJ-45 Port und USB Port
- Workstation mit mindestens folgender Ausstattung
  - 2x RJ-45 Port
  - 2x USB Port
  - 16 GB Arbeitsspeicher
  - 500 GB HDD
- Software
  - Windows 10 (min. Professional), Windows Server (min. 2012)
  - Debian, Ubuntu (aktuelle Versionen)
  - ESXi (aktuelle Version)
  - Aktueller Webbrowser

– SSH-Client

- 2x Netzkabel, min. 1,5m
- USB-Datenträger, 8GB

## 3 Planungsphase

### 3.1 Ist-Analyse

Zu Projektbeginn existiert im Unternehmen keine einheitliche Lösung zur Serverüberwachung. Für den Großteil der Server am Standort Augsburg wird der „Advanced Host Monitor“ eingesetzt. Diese Lösung erhält trotz ihres Alters noch regelmäßige Updates, aufgrund des veralteten User Interfaces und einem beschränkten Funktionsumfang, sowie der nicht ausreichenden Leistungsfähigkeit, wird seit mehreren Jahren der Wunsch nach einem neuen Monitoring-System geäußert.

Die Entwicklungsabteilungen überwachen die von ihnen betreuten Server mit einer auf „Elasticsearch“ und „Kibana“ basierenden Lösung. Um Know-how zu bündeln und Personalkosten zu sparen, signalisierten die Verantwortlichen eine Bereitschaft zur Zusammenlegung des Servermonitoring.

Am Standort Bremen wird seit mehreren Jahren auf die Open-Source-Anwendung „Icinga“ gesetzt. Die Systembetreuer haben mit dieser Lösung positive Erfahrungen gemacht, und loben insbesondere die einfache Erweiterbarkeit durch Plugins, sowie die Möglichkeit ohne Aufwand optisch ansprechende Dashboards zu erstellen. Aufgrund der Komplexität dieser Software und einem anstehenden Update auf „Icinga 2“ wurde auch von dieser Seite der Wunsch nach einem einheitlichen und zentral verwaltetem Servermonitoring geäußert.

### 3.2 Soll-Analyse

#### 3.2.1 Befragung der Fachabteilungen

Es wurden die entsprechenden Verantwortlichen via E-Mail, Telefonaten und Meetings zu den betreuten Servern und deren Monitoring befragt. Hierbei zeigten sich große Überschneidungen bei den Serverumgebungen sowie den Anforderungen an deren Überwachung, was eine Zusammenlegung logisch erscheinen lässt.

Über alle Abteilungen hinweg werden verschiedenste Windows-Versionen sowie Linux-Distributionen eingesetzt, weswegen ausschließlich ein plattformunabhängiges Monitoring-System wie „Icinga“ eingesetzt werden kann. Zwar sind die meisten Server virtualisiert, allerdings werden gerade bei Datenbanken auch dedizierte Rechner eingesetzt, welche selbstverständlich ebenfalls überwacht werden müssen.

Die zu überwachenden Parameter beziehungsweise Dienste ähneln sich zwischen den Abteilungen ebenfalls sehr stark. Neben Leistungsmetriken wie Prozessorauslastung, Arbeitsspeicherbelegung oder freiem Massenspeicher wird viel Wert auf die Überprüfung der Erreichbarkeit im Netzwerk gelegt. Aufgrund der zunehmenden Verbreitung von webbasierten Anwendungen soll auch der Betrieb von Webservern überwacht werden. Weiterhin wurde der Wunsch nach einer grafischen Aufbereitung geäußert.

### 3.2.2 Kriterienkatalog

Auf Basis der Befragungen wurde folgender Kriterienkatalog aufgestellt:

- Überwachbare Betriebssysteme
  - Alle aktuellen Windows Server Versionen
  - Alle gängigen Linux-Distributionen
- Überwachbare Systemparameter
  - CPU-Auslastung
  - RAM-Belegung
  - Massenspeicherbelegung
- Überwachung von Webservern
- Ansprechende und anpassbare grafische Aufbereitung

## 3.3 Wirtschaftlichkeitsanalyse

### 3.3.1 Projektkosten

Die Kosten, die durch das Projekt verursacht werden, setzen sich sowohl aus Personal- als auch aus Ressourcenkosten zusammen. Die Berechnung der Stundensätze findet sich in [A.1 Berechnung des Stundensatzes von Mitarbeitenden](#).

Die Kosten für die 35-Stunden-Woche des Auszubildenden belaufen sich auf 350 €. Die Arbeitszeit, die bei mitarbeitenden Personen zur Durchführung des Projekts angefallen ist (z.B. für Befragungen, Konfiguration der Netzwerkkomponenten, Anpassungen der Firewall für Updates, Abnahme) wurde auf vier Stunden geschätzt. Dafür fielen Personalkosten in Höhe von 160 € an. Die für das Projekt eingesetzte Hardware ist bereits beschrieben, und wird mit einer Pauschale von 360 € pro Jahr verrechnet. Auf die Projektdauer von fünf Tagen ergibt das Betriebskosten von 4,93 €.

Eine Aufstellung der Kosten befindet sich in Tabelle 2. Sie betragen insgesamt 514,93 €.

Vorgang	Zeit	Kosten	Gesamtkosten
Projektdurchführung	35 h	10 € / h	350,00 €
Kollegiale Unterstützung	4 h	40 € / h	160,00 €
Betriebskosten Server	5 d	360 € / a	4,93 €
			<b>514,93 €</b>

Tabelle 2: Kostenaufstellung

### 3.3.2 Betriebskosten des Monitorings

Die Betriebskosten für ein Monitoringsystem umfassen Personalkosten für Wartungsarbeiten und Anpassungen, sowie die Kosten die für zwei redundant ausgelegte Hardware-Server anfallen. Diese werden benötigt, da das Monitoring nicht innerhalb der Virtualisierungsumgebung betrieben werden sollte, um eine funktionierende Serverüberwachung auch bei Ausfall der VM-Infrastruktur zu gewährleisten.

Für das Einpflegen neuer Server oder Funktionen wird eine Arbeitsstunde pro Woche veranschlagt. Weiterhin wird der Arbeitsaufwand für Updates und Fehlerbehebungen nach Erfahrungsberichten auf acht Stunden pro Quartal geschätzt. Auf ein Jahr summiert ergibt beides einen Arbeitsaufwand von insgesamt 84 Stunden.

Die geschätzten Kosten für einen Hardware-Server belaufen sich auf 4.000 € jährlich. Die Gesamtkosten für den Betrieb einer Monitoring-Instanz belaufen sich, wie in Tabelle 3 dargelegt, auf 11.360,00 € pro Jahr.

Beschreibung	Einzelkosten	Einheiten	Jahreskosten
Wartungs- und Pflegearbeiten	40,00 € / h	84 h / a	3.360,00 €
Hardwarekosten	4.000,00 € / a	2	8.000,00 €
			<b>11.360,00 €</b>

Tabelle 3: Kosten Monitoring

### 3.3.3 Amortisationsdauer

Sollte die Evaluation durch diese Projektarbeit ergeben, dass „Icinga“ für den firmenweiten Einsatz geeignet ist, werden drei momentan im Betrieb befindliche Monitoringinstanzen (siehe 3.1 Ist-Analyse) durch ein zentrales „Icinga“-System ersetzt (siehe 1.2 Projektziel). Die Betriebskosten für die derzeitigen Überwachungssysteme entsprechen etwa den in 3.3.2 Betriebskosten des Monitorings berechneten 11.360,00 € pro Jahr; die jährliche Einsparung, wenn statt drei Systemen nur noch eines betrieben wird, beläuft sich also auf:

$$11.360,00 \text{ €} \cdot 2 \text{ Tage/Jahr} = 22.720,00 \text{ €} \quad (1)$$

Für die Ersteinrichtung des neuen Systems werden etwa zwei Wochen benötigt, also insgesamt 70 Arbeitsstunden. Das verursacht einmalige Kosten in Höhe von:

$$70 \text{ h} \cdot 40,00 \text{ €/h} = 2.800 \text{ €} \quad (2)$$

Die Amortisationszeit beträgt also  $\frac{2.800 \text{ €}}{22.720,00 \text{ €/a}} \approx 0,123 \text{ Jahre} \approx 45 \text{ Tage}$ .

## 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>1</sup> ausgewählt.

### 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.

---

<sup>1</sup>Vgl. SENSIO LABS [2010].

Eigenschaft	Gewichtung	Akelos	CakePHP	Symfony	Eigenentwicklung
Dokumentation	5	4	3	5	0
Reengineering	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

**Beispiel** Beispielentwürfe finden sich im Anhang A.7: Oberflächenentwürfe auf Seite vii.

## 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 Entity-Relationship-Modell (ERM) dargestellt, welches lediglich Entitäten, Relationen und die dazugehörigen Kardinalitäten enthält.

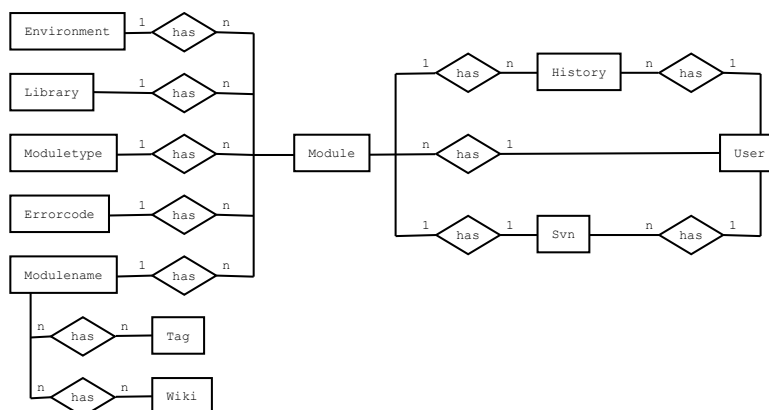


Abbildung 1: Vereinfachtes ER-Modell

## 4.5 Geschäftslogik

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

## 5 Implementierungsphase

**Beispiel** Ein Klassendiagramm, welches die Klassen der Anwendung und deren Beziehungen untereinander darstellt kann im Anhang A.12: Klassendiagramm auf Seite xvii eingesehen werden.

Abbildung 2 zeigt den grundsätzlichen Programmablauf beim Einlesen eines Moduls als EPK.

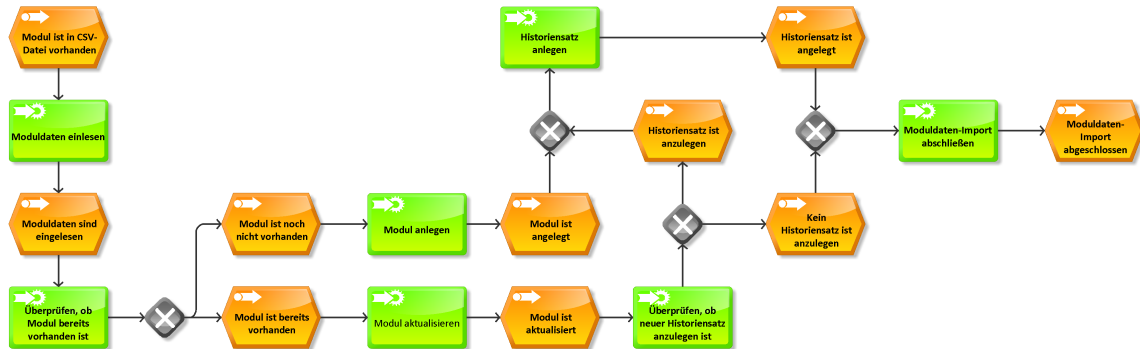


Abbildung 2: Prozess des Einlesens eines Moduls

### 4.6 Maßnahmen zur Qualitätssicherung

- Welche Maßnahmen werden ergriffen, um die Qualität des Projektergebnisses (siehe Kapitel ??: ??) 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 ??: ??) aufbauende Pflichtenheft ist im Anhang A.5: Pflichtenheft (Auszug) auf Seite iv zu finden.

## 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.8: Screenshots der Anwendung](#) auf Seite [ix](#).

## 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.11: Klasse: ComparedNaturalModuleInformation](#) auf Seite [xiv](#).

# 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.10: Testfall und sein Aufruf](#) auf der Konsole auf Seite [xiii](#). Dort ist auch der Aufruf des Tests auf der Konsole des Webservers zu sehen.



## 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?

## 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.13: Benutzerdokumentation auf Seite xviii. Die Entwicklerdokumentation wurde mittels PHPDoc<sup>2</sup> automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse findet sich im Anhang A.9: Entwicklerdokumentation auf Seite xi.

## 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 5 zu erkennen ist, konnte die Zeitplanung bis auf wenige Ausnahmen eingehalten werden.

---

<sup>2</sup>Vgl. PHPDOC.ORG [2010]

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 5: 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)?

## Quellenverzeichnis

### Dr. Peter Hoberg

DR. PETER HOBERG: *Vollständige Ermittlung von Personalkosten.* <https://www.controllingportal.de/Fachinfo/Kostenrechnung/Vollstaendige-Ermittlung-von-Personalkosten.html>, Abruf: 22.04.2020

### igmetall-bayern.de

IGMETALL-BAYERN.DE: *Tarifinfos Metall- und Elektroindustrie Bayern.* <https://www.igmetall-bayern.de/metall-elektro/>, Abruf: 22.04.2020

### igmetall.de

IGMETALL.DE: *Metall- und Elektroindustrie ERA – Ausbildungsvergütungen.* [https://www.igmetall.de/download/docs\\_MuE\\_ERA\\_Ausbildung\\_Juni2018\\_9bdc083c9c0ed885c63bd1b076830a817eaec814.pdf](https://www.igmetall.de/download/docs_MuE_ERA_Ausbildung_Juni2018_9bdc083c9c0ed885c63bd1b076830a817eaec814.pdf), Abruf: 22.04.2020

### phpdoc.org 2010

PHPDOC.ORG: *phpDocumentor-Website2.* 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, Andreas Germer, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

*Evaluierung von „Icinga“ – Open Source Monitoring*

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.

Augsburg, den 21.05.2020

---

ANDREAS GERMER

## A Anhang

### A.1 Berechnung des Stundensatzes von Mitarbeitenden

Nach der Entgelttabelle der IG Metall Bayern für Metall und Elektro<sup>3</sup> erhalten Angestellte der Entgeltgruppe EG 08 ein Monatsbruttogehalt von 3.800,00 €. Zusätzlich dazu müssen ungefähr ein Fünftel für die Arbeitgeberanteile der Sozialversicherung und Mehrleistungen wie Urlaubsgeld hinzugerechnet werden.<sup>4</sup> Aufsummiert ergibt das jährliche Personalkosten von 54.720,00 €.

$$3.800,00 \text{ €} \cdot 1,2 \cdot 12 \text{ m} = 54.720,00 \text{ € /a} \quad (1)$$

Eine Arbeitszeit von 35 Stunden pro Woche bedeutet bei 52 Wochen eine Jahresarbeitszeit von 1820 Stunden. Abzüglich der Urlaubstage (30), Feiertage (ca. 11) und Fehltag durch Krankheit und Fortbildungen (ca. 15) ergibt das eine Jahresarbeitszeit von 1498 Stunden pro Jahr.

$$35 \text{ h/w} \cdot 52 \text{ m} = 1820 \text{ h/a} \quad (2)$$

$$1820 \text{ h/a} - [(30 + 11 + 15) \text{ d} \cdot 35 \text{ h/d}] = 1498 \text{ h/a} \quad (3)$$

Werden die Jahrespersonalkosten von 54.720,00 € durch die Jahresarbeitszeit von 1498 Stunden dividiert, ergibt dies Stundenkosten für den Arbeitgeber in Höhe von 36,53 €. Da es sich hierbei um eine Schätzung handelt, wird vereinfacht von 40,00 € pro Stunde ausgegangen.

$$54.720,00 \text{ € /a} \div 1498 \text{ h/a} = 36,53 \text{ € /h} \approx 40,00 \text{ € /h} \quad (4)$$

Für Auszubildende mit einer Ausbildungsvergütung in Höhe von 1.207,00 €<sup>5</sup> ergeben sich nach selber Kalkulation Personalkosten in Höhe von 10,57 €, also vereinfacht 10,00 € pro Stunde.

$$25 \cdot 220 \text{ Tage/Jahr} \cdot 10 \text{ min/Tag} = 55000 \text{ min/Jahr} \approx 917 \text{ h/Jahr} \quad (5)$$

---

<sup>3</sup>Vgl. IGMETALL-BAYERN.DE

<sup>4</sup>Vgl. DR. PETER HOBERG

<sup>5</sup>Vgl. IGMETALL.DE

## A.2 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.3 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.4 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>.

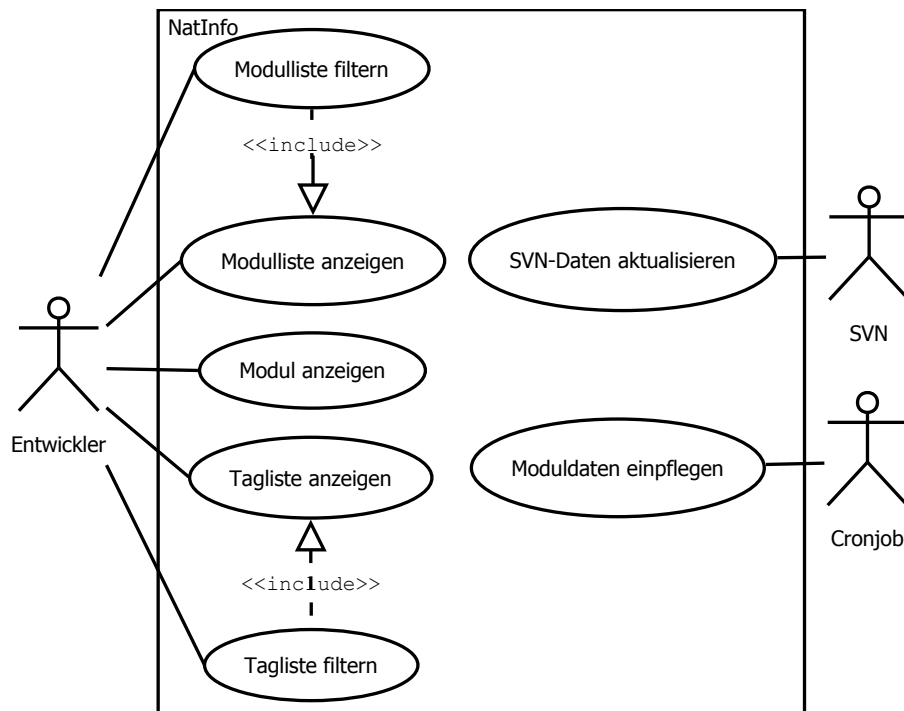


Abbildung 3: Use Case-Diagramm

## A.5 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.6 Datenbankmodell

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

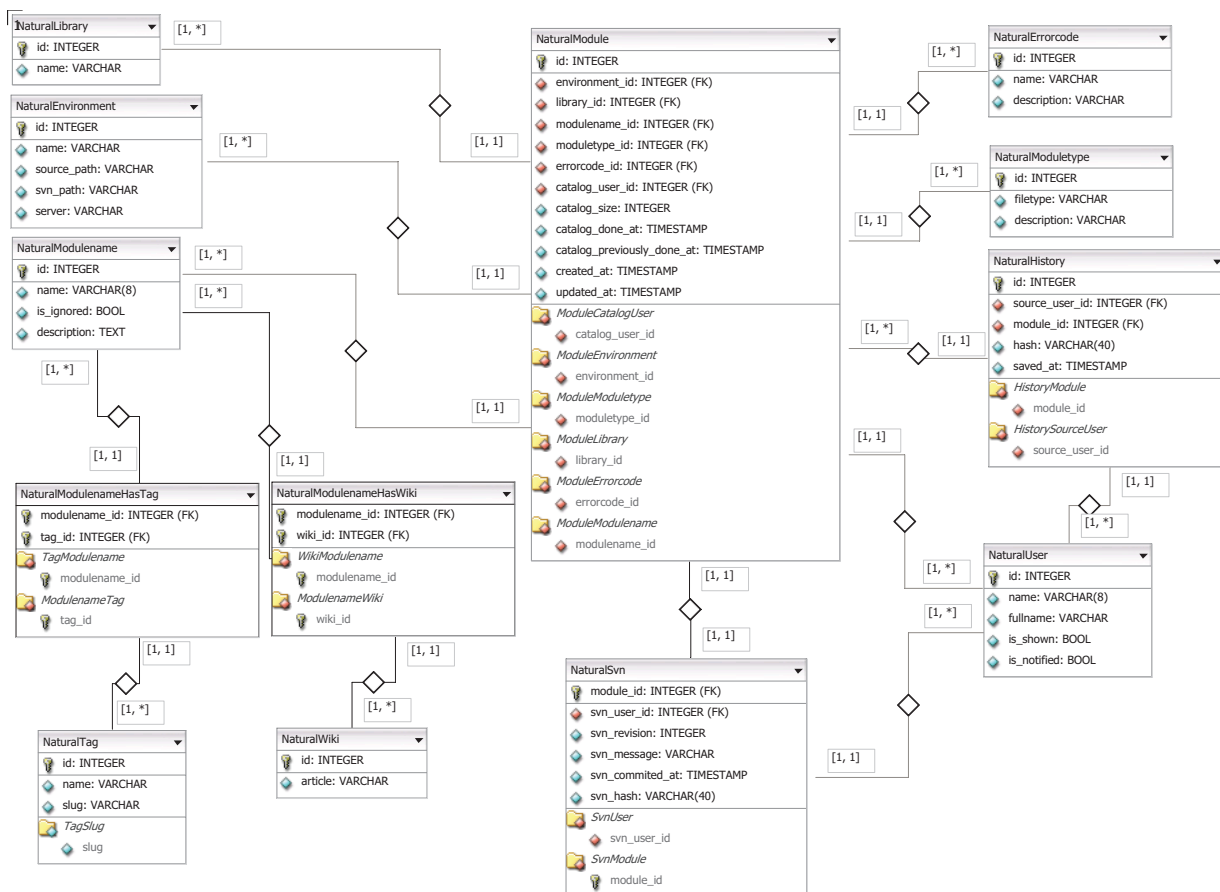


Abbildung 4: Datenbankmodell

## A.7 Oberflächenentwürfe

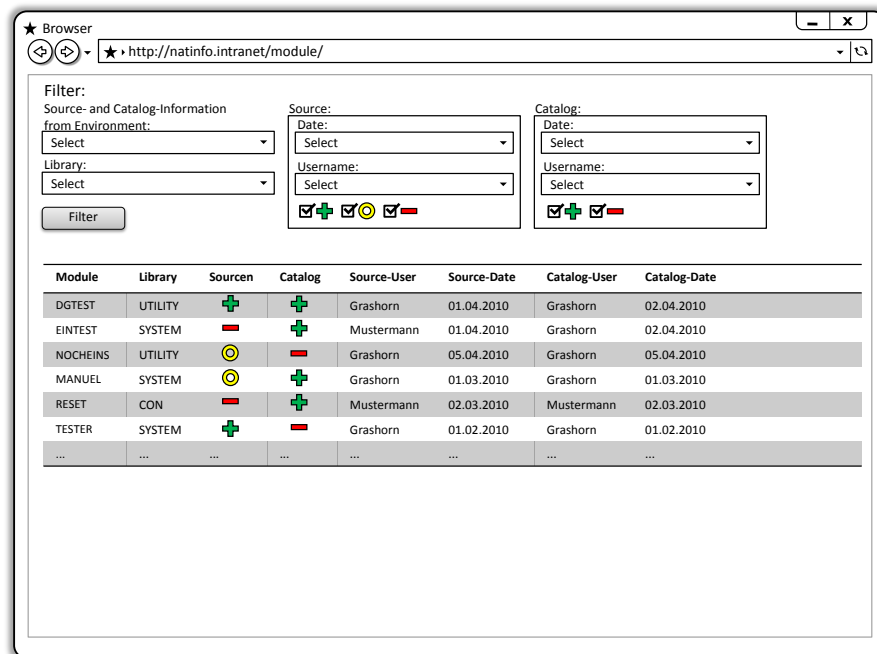


Abbildung 5: Liste der Module mit Filtermöglichkeiten

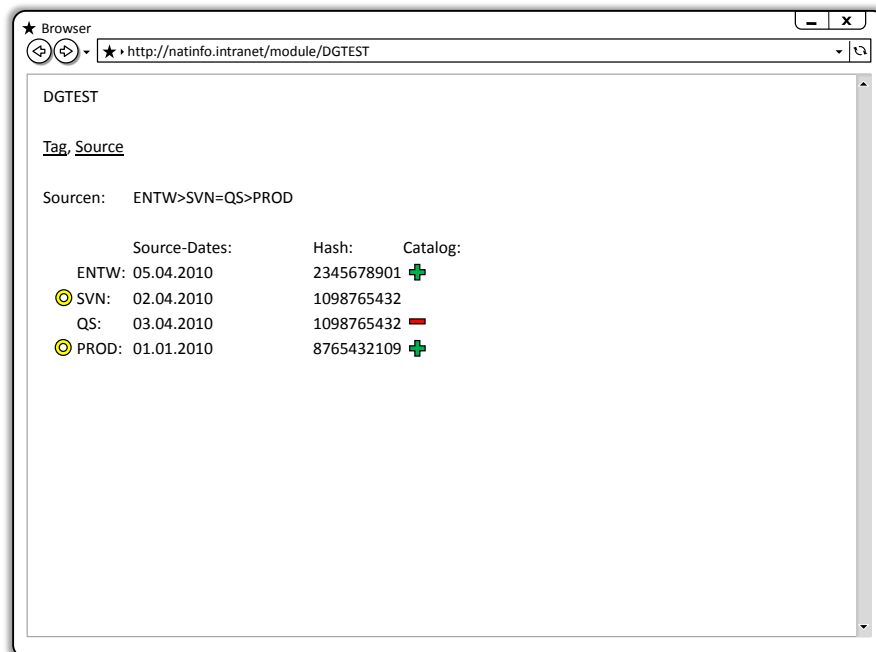


Abbildung 6: Anzeige der Übersichtsseite einzelner Module

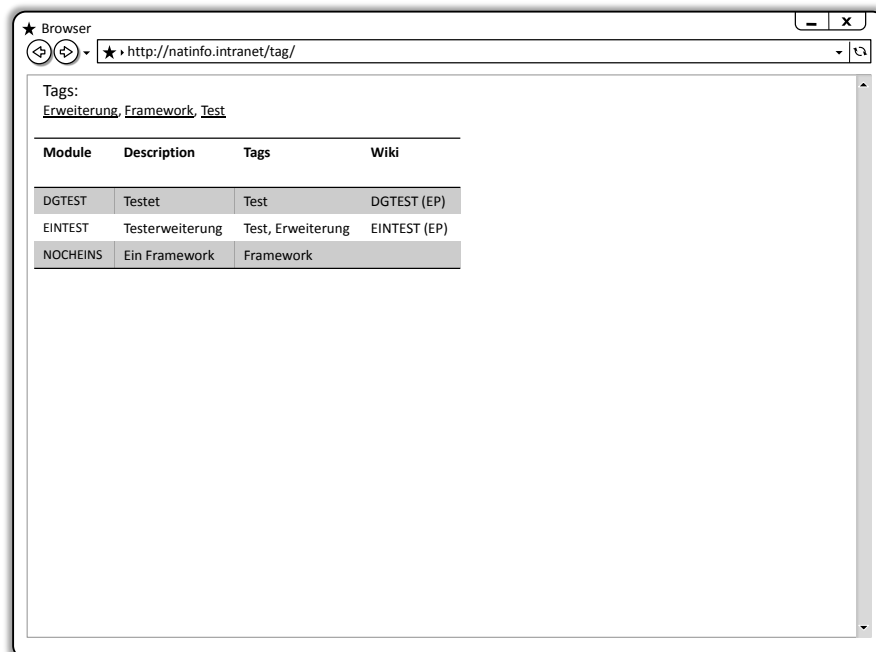


Abbildung 7: Anzeige und Filterung der Module nach Tags

## A.8 Screenshots der Anwendung

The screenshot shows the website 'ALTE OLDENBURGER NATINFO'. The header features the logo and a background image of a family walking on a grassy dune. A navigation menu on the right includes 'Häufig benötigte Seiten', 'Ein-Klick-Menü', 'Direktaufruf', and 'Inhaltssuche'. The main content area is titled 'Tags' and shows a filter for 'Project, Test'. Below this is a table of modules.

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



## Modules

Environment	ENTW
Library	Select
Catalog user	Select
Catalog date	Select
Source user	Select
Source date	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.9 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.10 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 ?>

```

Listing 1: Testfall in PHP

```

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 10: Aufruf des Testfalls auf der Konsole

## A.11 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 ?>
```

Listing 2: Klasse: ComparedNaturalModuleInformation

## A.12 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>.

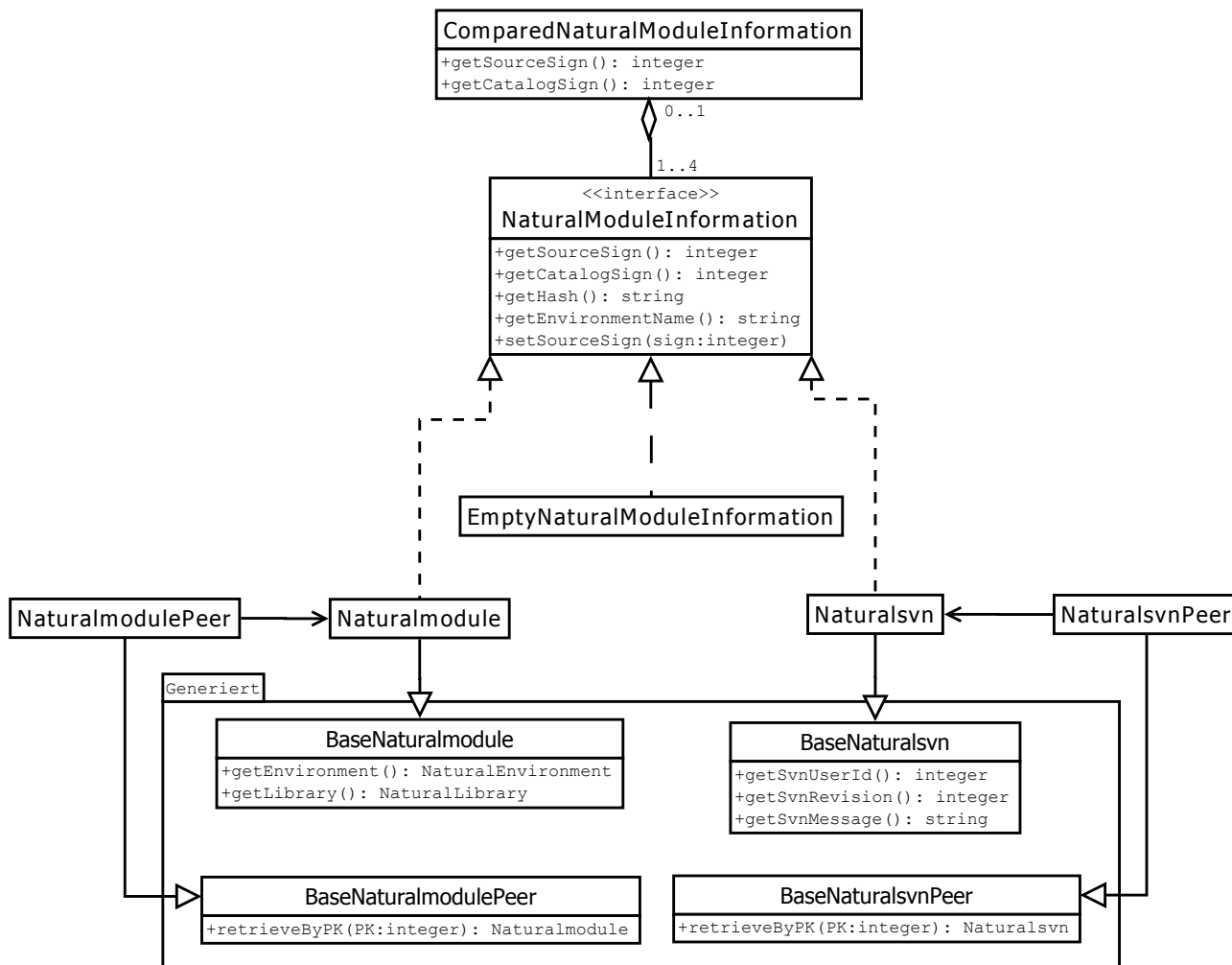







Abbildung 11: Klassendiagramm

## A.13 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.