



Abschlussprüfung Winter 2024

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Entwicklung von Dev Kickstarter

**Automatisierte Onboarding- und Ressourcenmanagement-Prozesse
bei TUI Group**

Abgabetermin: Hannover, den 18.11.2024

Prüfungsbewerber:

Paul Glesmann
Schopenhauerstraße 15
30625 Hannover



Ausbildungsbetrieb:

TUI InfoTec GmbH
Karl-Wichert-Allee 23
30627 Hannover

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Glossar	V
Listings	VII
Abkürzungsverzeichnis	VIII
1 Einleitung	1
1.1 Projektumfeld	1
1.2 Projektziel	1
1.3 Projektbegründung	2
1.4 Projektschnittstellen	2
1.5 Projektabgrenzung	3
2 Projektplanung	3
2.1 Projektphasen	3
2.2 Ressourcenplanung	4
2.3 Entwicklungsprozess	4
3 Analysephase	4
3.1 Ist-Analyse	4
3.2 Wirtschaftlichkeitsanalyse	5
3.2.1 „Make or Buy“-Entscheidung	5
3.2.2 Projektkosten	5
3.2.3 Amortisationsdauer	6
3.3 Nutzwertanalyse	6
4 Entwurfsphase	7
4.1 Zielplattform	7
4.2 Architekturdesign	9
4.3 Entwurf der E-Mail	9
4.4 Maßnahmen zur Qualitätssicherung	9
5 Implementierungsphase	10
5.1 Implementierung der Gitlab Systemhook	10
5.2 Implementierung der GitLab Systemhook	10
5.3 Implementierung der Nachrichtenzustellung an SNS	11
5.4 Bereitstellung und Test	11
5.5 Implementierung des Dev-Kickstarter	11

5.6	Implementierung der E-Mail	11
6	Abnahmephase	12
7	Einführungsphase	12
8	Dokumentation	12
9	Fazit	13
9.1	Soll-/Ist-Vergleich	13
9.2	Lessons Learned	13
9.3	Ausblick	13
	Literaturverzeichnis	14
	Eidesstattliche Erklärung	15
A	Anhang	i
A.1	Detaillierte Zeitplanung	i
A.2	Ressourcen Planung	ii
A.3	Iterationsplan	iii
A.4	Lastenheft (Auszug)	iii
A.5	Use Case-Diagramm	v
A.6	Pflichtenheft (Auszug)	v
A.7	Datenbankmodell	vii
A.8	Oberflächenentwürfe	viii
A.9	Screenshots der Anwendung	x
A.10	Entwicklerdokumentation	xii
A.11	Testfall und sein Aufruf auf der Konsole	xiv
A.12	Klasse: ComparedNaturalModuleInformation	xv
A.13	Klassendiagramm	xviii
A.14	Benutzerdokumentation	xix

Abbildungsverzeichnis

1	Finaler Entwurf des Infrastruktur-Designs und Soll-Zustand	8
2	Implementierung der Gitlab Systemhook	10
3	Use Case-Diagramm	v
4	Datenbankmodell	vii
5	Erster Entwurf der E-Mail	viii
6	Umsetzung in HTML mit CSS	ix
7	Finales Design der E-Mail	ix
8	Anzeige und Filterung der Module nach Tags	x
9	Liste der Module mit Filtermöglichkeiten	xi
10	Aufruf des Testfalls auf der Konsole	xv
11	Klassendiagramm	xviii

Tabellenverzeichnis

1	Grobe Zeitplanung	3
2	Kostenaufstellung	6
3	Soll-/Ist-Vergleich	13

Glossar

- Microsoft Teams: Cooles Kommunikations-Tool, das für Teamarbeit und Kommunikation verwendet wird.
- CI: Continuous Integration - Ein Entwicklungsansatz, bei dem Änderungen an einer Software regelmäßig in ein gemeinsames Repository integriert werden.
- CD: Continuous Deployment - Eine Softwarebereitstellungsmethode, die das automatisierte Deployment von Codeänderungen ermöglicht.
- CSS: Cascading Style Sheets - Eine Stylesheet-Sprache, die verwendet wird, um das Layout von Webdokumenten zu gestalten.
- ERM: Entity-Relationship-Modell - Ein konzeptionelles Datenmodell, das die Beziehungen zwischen Entitäten beschreibt.
- Go: Programmiersprache Go - Eine von Google entwickelte Programmiersprache.
- HTML: Hypertext Markup Language - Die Standardauszeichnungssprache für Dokumente, die im Web angezeigt werden.
- GitLab: Eine Plattform für Versionskontrolle und DevOps, die es Entwicklern ermöglicht, Code zu speichern, zu verwalten und zu teilen.
- GitLab-Systemhooks: Mechanismen in GitLab, die es ermöglichen, auf bestimmte Ereignisse (wie Pushes, Merge-Requests usw.) zu reagieren und automatisierte Aktionen auszulösen, z.B. das Auslösen von CI/CD-Pipelines oder Benachrichtigungen in externen Systemen.
- Onboarding: Der Prozess, durch den neue Mitarbeiter in ein Unternehmen integriert werden und die notwendige Einarbeitung erhalten.
- User Experience (UX): Die Gesamterfahrung eines Benutzers bei der Interaktion mit einem Produkt, insbesondere in Bezug auf Benutzerfreundlichkeit und Zufriedenheit.
- Schnittstelle: Ein definierter Punkt, an dem zwei Systeme oder Komponenten miteinander kommunizieren und Daten austauschen.
- Automatisierung: Der Einsatz von Technologien, um Prozesse ohne menschliches Eingreifen auszuführen, um Effizienz und Genauigkeit zu erhöhen.
- Jira: Eine Projektmanagement-Software, die für die Planung und Nachverfolgung von Aufgaben in Softwareentwicklungsprojekten genutzt wird.
- agil: Ein Entwicklungsansatz, der auf Flexibilität und Anpassungsfähigkeit abzielt und kurze Iterationen sowie regelmäßiges Feedback fördert.

- iterativ: Ein Ansatz, bei dem ein Prozess in wiederholten Zyklen durchgeführt wird, um kontinuierliche Verbesserungen basierend auf Feedback zu ermöglichen.



Listings

1	Testfall in PHP	xiv
2	Klasse: ComparedNaturalModuleInformation	xv

Abkürzungsverzeichnis

API	Application Programming Interface
HTML	Hypertext Markup Language
TUI	Touristik Union International

1 Einleitung

Diese Projektdokumentation beschreibt den Ablauf des Projektes „Implementierung automatisierter **Onboarding**- und Ressourcenmanagement-Prozesse zur Optimierung der **User Experience** für neue Entwickler bei Touristik Union International (TUI) Group“.

Die vorliegende Dokumentation wurde projektbegleitend erstellt und dient der Abschlussprüfung im Ausbildungsberuf Fachinformatiker Anwendungsentwicklung. Sie erläutert die Ziele, den Ablauf und die Ergebnisse des Projektes zur Automatisierung des **Onboarding**-Prozesses für neue Entwickler. Des Weiteren werden die eingesetzten Technologien und die Interaktionen mit verschiedenen Systemen beschrieben.

Die **blau** markierten Begriffe werden nicht direkt im Text erklärt, sondern im angehängten Glossar.

1.1 Projektumfeld

Die **TUI InfoTec GmbH** ist eine Tochtergesellschaft der **TUI AG**, einem weltweit führenden Unternehmen im Bereich Tourismus und Reisen. Als interner IT-Dienstleister übernimmt die **TUI InfoTec GmbH** die IT-Betreuung und -Optimierung für die gesamte **TUI AG**. Sie ist verantwortlich für die Bereitstellung und Wartung der IT-Infrastruktur sowie für die Entwicklung und Implementierung von Softwarelösungen, die die Geschäftsprozesse innerhalb des Konzerns unterstützen.

Die **TUI InfoTec GmbH** beschäftigt zurzeit etwas mehr als 600 Mitarbeiter, die in unterschiedlichen Bereichen der IT tätig sind. Die **Shared Services Abteilung**, die als Auftraggeber für dieses Projekt fungiert, bietet zentrale IT-Services für verschiedene Abteilungen des Unternehmens an. Dabei konzentriert sich diese Abteilung besonders darauf, den Entwicklern eine optimale Arbeitsumgebung bereitzustellen, um sie von zeitaufwendigen, wiederkehrenden Aufgaben zu entlasten und ihre Produktivität zu steigern.

1.2 Projektziel

Das Ziel dieses Projekts ist es, die Effizienz und **User Experience** beim **Onboarding** neuer Entwickler durch automatisierte Prozesse zu verbessern. Der aktuelle Onboarding-Prozess ist überwiegend manuell und erfordert zeitaufwändige Schritte wie das Hinzufügen neuer Entwickler zu **Microsoft Teams** Gruppen, das Versenden von E-Mails mit wichtigen Informationen zu internen Prozessen und das Bereitstellen von Zugriffen auf **GitLab**-Repositories sowie weitere Entwicklungsressourcen. Diese manuellen Tätigkeiten sind nicht nur zeitintensiv, sondern bergen auch das Risiko von Fehlern.

Im Rahmen dieses Projekts wird eine Anwendung entwickelt, die auf **GitLab**-Events reagiert und automatisch neue Entwickler in die relevanten **Microsoft Teams** Gruppen integriert. Zusätzlich werden E-Mails mit zentralen Informationen zu internen Abläufen und Dokumentationsstandorten

verschickt, um den Einstieg für neue Entwickler zu erleichtern. Dadurch wird die Produktivität neuer Mitarbeiter gesteigert und gleichzeitig der manuelle Aufwand für das IT-Team deutlich reduziert.

Am Ende des Projekts soll eine Lösung stehen, die sowohl den aktuellen Anforderungen gerecht wird als auch einfach verwaltbar und skalierbar ist, um auf zukünftige Bedürfnisse flexibel reagieren zu können.

1.3 Projektbegründung

Der derzeit manuelle [Onboarding](#)-Prozess für neue Entwickler ist sowohl zeitaufwendig als auch fehleranfällig. Neue Entwickler werden oft nicht sofort in die richtigen Teams und Kommunikationskanäle integriert und erhalten möglicherweise nicht alle relevanten Informationen zum Einstieg in die Entwicklungsprozesse bei [TUI](#). Dies kann zu Verzögerungen führen, die den Produktivitätseintritt der Entwickler behindern. Ein automatisierter Prozess würde sicherstellen, dass jeder neue Entwickler sofort alle nötigen Ressourcen und Teammitgliedschaften erhält und gleichzeitig unnötige manuelle Arbeit für das IT-Team entfällt.

Die Automatisierung des [Onboarding](#)-Prozesses bietet somit klare Vorteile: - **Zeitersparnis**: Die Automatisierung reduziert die Zeit, die das IT-Team für manuelle Aufgaben aufwenden muss. Neue Entwickler können ohne Verzögerung in die relevanten Gruppen aufgenommen und erhalten automatisch alle notwendigen Informationen. - **Kosteneffizienz**: Durch die Reduktion des manuellen Aufwands werden nicht nur Fehler vermieden, sondern auch Ressourcen effizienter eingesetzt. Das IT-Team kann seine Kapazitäten für wichtigere Aufgaben nutzen.

Die Motivation hinter dem Projekt ist, die Einarbeitungszeit neuer Entwickler zu verkürzen und gleichzeitig eine höhere Konsistenz und Qualität im [Onboarding](#)-Prozess zu gewährleisten. So wird der Einstieg für neue Entwickler erleichtert, und sie können schneller produktiv arbeiten.

1.4 Projektschnittstellen

Die entwickelte Anwendung interagiert mit verschiedenen Systemen, um den [Onboarding](#)-Prozess für neue Entwickler zu automatisieren. Ein zentraler Bestandteil ist die Integration mit [GitLab](#), das bei [TUI](#) sowohl für die Versionskontrolle als auch für [Continuous Integration](#) und [Continuous Deployment](#) (CI/CD) genutzt wird. Um die Automatisierung zu ermöglichen, werden [Schnittstellen](#) in der Programmiersprache [Go](#) entwickelt, die auf [GitLab](#)-Ereignisse reagieren und die entsprechenden Prozesse auslösen.

Das Projekt wurde durch den **Head of Technology** der **Shared Services Abteilung** genehmigt. Innerhalb der Abteilung standen zudem Mitarbeiter zur Verfügung, um bei Rückfragen zu unterstützen und regelmäßiges Feedback während der Entwicklung zu geben.

Die **Benutzer** der Anwendung sind neue Entwickler, die automatisch in den [Onboarding](#)-Prozess integriert werden, um eine reibungslose Einarbeitung und den Zugang zu den erforderlichen Ressourcen zu gewährleisten.

1.5 Projektabgrenzung

Die aktuelle Refaktorisierung des bestehenden Systems ist nicht Teil dieses Projekts. Zukünftige Anpassungen des aktuellen Systems sind erforderlich, um die [GitLab-Systemhooks](#), die wir derzeit verwenden, entsprechend zu überarbeiten und anzupassen. Diese Änderungen werden separat behandelt und sind nicht im Rahmen der gegenwärtigen Automatisierungsprojekte enthalten.

2 Projektplanung

2.1 Projektphasen

Für die Durchführung des Projekts standen insgesamt 80 Stunden zur Verfügung. Diese Stunden wurden vor Projektbeginn auf verschiedene Phasen der Softwareentwicklung verteilt. Eine Übersicht der groben Zeitplanung und der Hauptphasen ist in [Tabelle 1](#) zu finden. Darüber hinaus können die einzelnen Hauptphasen in kleinere Unterphasen unterteilt werden. Eine detaillierte Darstellung dieser Phasen ist im [Anhang A.1](#) auf Seite i zu finden.

1

Projektphase	Geplante Zeit
Analysephase	6 h
Entwurfsphase	6 h
Implementierungsphase	34 h
Test- und Kontrollphase	18 h
Dokumentation + Nachbearbeitung	6 h
Erstellen der Projektdokumentation und Präsentation	8 h
Pufferzeit	2 h
Gesamt	80 h

Tabelle 1: Grobe Zeitplanung

Eine detailliertere Zeitplanung findet sich im [Anhang A.1](#).

2.2 Ressourcenplanung

Die Planung der benötigten Ressourcen ist ein wesentlicher Bestandteil der Projektorganisation. Dazu gehören sowohl Hard- und Software als auch die erforderlichen Räumlichkeiten. Eine detaillierte Übersicht über die verwendeten Ressourcen finden Sie im [Anhang A.2](#).

2.3 Entwicklungsprozess

Für die Durchführung des Projekts wurde ein [agiler](#) Entwicklungsprozess gewählt. Dieser Ansatz ermöglichte es, flexibel auf sich ändernde Anforderungen und Herausforderungen während der Entwicklung zu reagieren. Der Arbeitsprozess war stark [iterativ](#) geprägt, mit regelmäßigen Rücksprachen und enger Zusammenarbeit mit den Kollegen.

Der [iterative](#) Zyklus bestand darin, Arbeitsschritte zu planen, umzusetzen, zu evaluieren und anhand des erhaltenen Feedbacks kontinuierlich zu verbessern. Diese kurzen Iterationszyklen erlaubten es, Ergebnisse frühzeitig zu präsentieren und Feedback direkt in die nächste Phase einfließen zu lassen. Dies förderte nicht nur eine hohe Anpassungsfähigkeit, sondern auch eine kontinuierliche Verbesserung der Qualität des Projekts.

Die [agile](#) Vorgehensweise half dabei, schnelle Anpassungen an neuen Anforderungen vorzunehmen, die in den regelmäßigen Besprechungen und durch Feedback der Kollegen eingebracht wurden. Durch den Einsatz von Tools wie [Microsoft Teams](#) und [Jira](#) wurde die Zusammenarbeit und das Projektmanagement unterstützt, sodass der Fortschritt transparent blieb und die Arbeitsschritte effektiv geplant und dokumentiert werden konnten.

3 Analysephase

3.1 Ist-Analyse

Der bisherige Onboarding-Prozess stellt neue Mitarbeiter vor die Herausforderung, dass zentrale Informationen über interne Prozesse, wie die Standorte von Dokumentationen, eingesetzte Technologien und wichtige Einstiegshilfen, nicht rechtzeitig zur Verfügung stehen. Obwohl wir eine Entwicklerplattform anbieten, auf der alle notwendigen Dokumentationen gesammelt sind, entdecken neue Entwickler diese häufig erst spät. Zudem sind neue Mitarbeiter oft nicht in die relevanten Microsoft Teams-Kanäle integriert, was die Kommunikation und den Zugriff auf wichtige Informationen zusätzlich verzögert.

Da jeder neue Entwickler früher oder später in unserem GitLab-System arbeitet, besteht hier die Chance, den Onboarding-Prozess zu automatisieren. Momentan fehlen jedoch automatisierte Prozesse, die neue Entwickler nahtlos in die vorhandenen Systeme einbinden.

Um dieses Problem zu lösen, haben wir beschlossen, eine GitLab Systemhook zu implementieren, die auf `user_create`-Ereignisse lauscht und entsprechend reagiert. Zum Zeitpunkt des Projektbeginns

waren bereits drei separate Systemhooks im Einsatz. Daher entschieden wir uns für die Entwicklung einer zentralen Systemhook, die alle GitLab-Ereignisse an ein zentrales **AWS SNS** Topic (Simple Notification Service) sendet. Von dort aus können verschiedene Anwendungen durch **SQS** Queues (Simple Queue Service) und spezifische Filterregeln gezielt auf die GitLab-Ereignisse reagieren. Diese Lösung bietet einen zentralen Punkt für GitLab-Ereignisse und ermöglicht es uns, den Onboarding-Prozess effizient zu erweitern und zu automatisieren.

3.2 Wirtschaftlichkeitsanalyse

Das Projekt zur Automatisierung des Onboardings ist für die TUI InfoTec GmbH von großem wirtschaftlichen Nutzen. Die Automatisierung sorgt dafür, dass neue Entwickler sofort Zugriff auf alle relevanten Ressourcen erhalten, ohne dass manuelle Eingriffe notwendig sind. Dadurch werden sowohl Verzögerungen als auch die Gefahr, Mitarbeiter zu übersehen oder Zugänge zu vergessen, deutlich reduziert. Dies steigert nicht nur die Effizienz der neuen Entwickler, sondern entlastet auch das IT-Team, das sich auf wichtigere Aufgaben konzentrieren kann. Durch die Einsparung von Zeit und die Reduktion der Arbeitsunterbrechungen werden langfristig Kosten gesenkt und die Produktivität gesteigert.

3.2.1 „Make or Buy“-Entscheidung

Es existieren zwar bereits Softwarelösungen, die Teile des Onboardings automatisieren könnten, jedoch gibt es kein Produkt, das die spezifischen Anforderungen der TUI InfoTec GmbH vollständig abdeckt. Besonders die Integration mit den bereits genutzten [GitLab-SystemHooks](#) und die spezifischen Anpassungen, die für die internen Abläufe nötig sind, erfordern eine maßgeschneiderte Lösung. Aus diesem Grund wurde beschlossen, das Projekt intern umzusetzen, um eine optimale Integration in die bestehende Infrastruktur und eine hohe Flexibilität für zukünftige Anpassungen sicherzustellen.

3.2.2 Projektkosten

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 1450 € Brutto.

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

$$1450 \text{ €/Monat} \cdot 13.3 \text{ Monate/Jahr} = 19285 \text{ €/Jahr} \quad (2)$$

$$\frac{19285 \text{ €/Jahr}}{1760 \text{ h/Jahr}} \approx 10.96 \text{ €/h} \quad (3)$$

Die Kosten, die während der Entwicklung des Projekts anfallen, werden im Folgenden kalkuliert. Die für die Realisierung des Projekts benötigten Personal- und Ressourcenkosten sind von der Personalabteilung festgelegte Pauschalsätze, die nicht weiter angepasst werden dürfen. Der Stundensatz eines Auszubildenden beträgt demzufolge 10.96€, der eines Mitarbeiters 25€. Die Ressourcennutzung umfasst einen Büroarbeitsplatz, die Hardware- und Softwarenutzung sowie Stromkosten. Hierfür wird von der Personalabteilung ein pauschaler Stundensatz von 15€ vorgegeben. Die folgende Tabelle zeigt die detaillierten Projektkosten auf, die für die einzelnen Vorgänge anfallen.

Vorgang	Mitarbeiter	Zeit (h)	Personal (€) ⁶	Ressourcen (€) ⁷	Gesamt (€)
Entwicklungskosten	1 x Auszubildender	80	800,00	1.050,00	1.750,00
Fachgespräch	2 x Mitarbeiter, 1 x Auszubildender	3	180,00	135,00	315,00
Code-Review	1 x Mitarbeiter	4	100,00	60,00	160,00
Abnahme	2 x Mitarbeiter	1	50,00	30,00	80,00
Projektkosten gesamt					2.305,00

Tabelle 2: Kostenaufstellung

3.2.3 Amortisationsdauer

Das Projekt bringt erhebliche Zeiteinsparungen mit sich, insbesondere für das IT-Team und die neuen Entwickler. Bei einer durchschnittlichen Zeiteinsparung von 10 Minuten pro Tag und 220 Arbeitstagen im Jahr für jeden der 25 Anwender ergibt sich eine jährliche Zeiteinsparung von etwa 917 Stunden. Bei einem kombinierten Stundensatz von 40 € (inklusive Mitarbeiter- und Ressourcenkosten) ergibt sich eine jährliche Einsparung von 36.680 €. Die Amortisationsdauer des Projekts beträgt somit etwa 4 Wochen, was das Projekt sehr kosteneffizient macht.

3.3 Nutzwertanalyse

Neben den monetären Vorteilen bietet das Projekt auch erhebliche nicht-monetäre Vorteile. Die Automatisierung des Onboardings sorgt dafür, dass neue Entwickler sofort Zugriff auf alle relevanten Ressourcen haben, ohne dass Mitarbeiter ständig an das Einrichten von Zugängen erinnert werden müssen. Dies reduziert Ablenkungen und Unterbrechungen im täglichen Arbeitsablauf, was zu einer höheren Produktivität und einer angenehmeren Arbeitsatmosphäre führt. Dadurch wird nicht nur der Einstieg für neue Entwickler erleichtert, sondern auch die Belastung der bestehenden Teams deutlich verringert.

⁶Personalkosten pro Vorgang = Anzahl Mitarbeiter · Zeit · Stundensatz.

⁷Ressourcenbeitrag pro Vorgang = Anzahl Mitarbeiter · Zeit · 15 € (Ressourcenbeitrag pro Stunde).

4 Entwurfsphase

4.1 Zielplattform

Aus der Ist-Analyse ging hervor, dass wir zwei wesentliche Schnittstellen benötigen, um die Anforderungen des Projekts zu erfüllen. Erstens eine **GitLab System Hook**, die GitLab-Ereignisse an ein **Amazon Simple Notification Service (SNS)**-Thema sendet. Zweitens eine Anwendung, die spezifische Ereignisse aus einer **Amazon Simple Queue Service (SQS)**-Warteschlange konsumiert und darauf reagiert, wie z.B. die Verarbeitung von `user_create`-Ereignissen. Beide Schnittstellen haben wir in **Go** entwickelt, um unsere bestehende Expertise zu nutzen und den Entwicklungsprozess zu beschleunigen.

Die Wahl der Programmiersprache **Go** basiert auf einer Reihe von technischen und praktischen Überlegungen. Da Go die primär in unserer Abteilung verwendete Programmiersprache ist und das Team über umfangreiche Erfahrung damit verfügt, konnten wir die Einarbeitungszeit minimieren und effizient arbeiten.

Für die Bereitstellung der Anwendung haben wir uns für **Amazon Web Services (AWS)** entschieden. AWS bietet eine breite Palette an Diensten, die unseren Projektanforderungen ideal entsprechen, und ist bereits tief in den Prozessen unseres Unternehmens verankert. Diese Vertrautheit mit AWS ermöglicht eine reibungslose Implementierung und die Nutzung bestehender Best Practices und interner Ressourcen.

Für die zentrale Erfassung und Verarbeitung der GitLab-Systemereignisse nutzen wir **AWS SNS** in Kombination mit **AWS SQS**. Der SNS-Dienst sammelt die relevanten GitLab-Ereignisse und verteilt sie über SQS an verschiedene Anwendungen. Diese Architektur garantiert eine skalierbare und effiziente Verarbeitung der Ereignisse.

Für den Versand von E-Mails setzen wir **AWS Simple Email Service (SES)** ein. Die E-Mails werden mithilfe von HTML und CSS gestaltet, um ansprechende und benutzerfreundliche Inhalte zu erzeugen.

Die Schnittstellen selbst werden in einem bestehenden **Amazon Elastic Kubernetes Service (EKS)** Cluster bereitgestellt, der bereits für andere Anwendungen in unserer Abteilung genutzt wird. Durch die Wiederverwendung dieser Infrastruktur können wir zusätzliche Ressourcen sparen und den Verwaltungsaufwand minimieren.

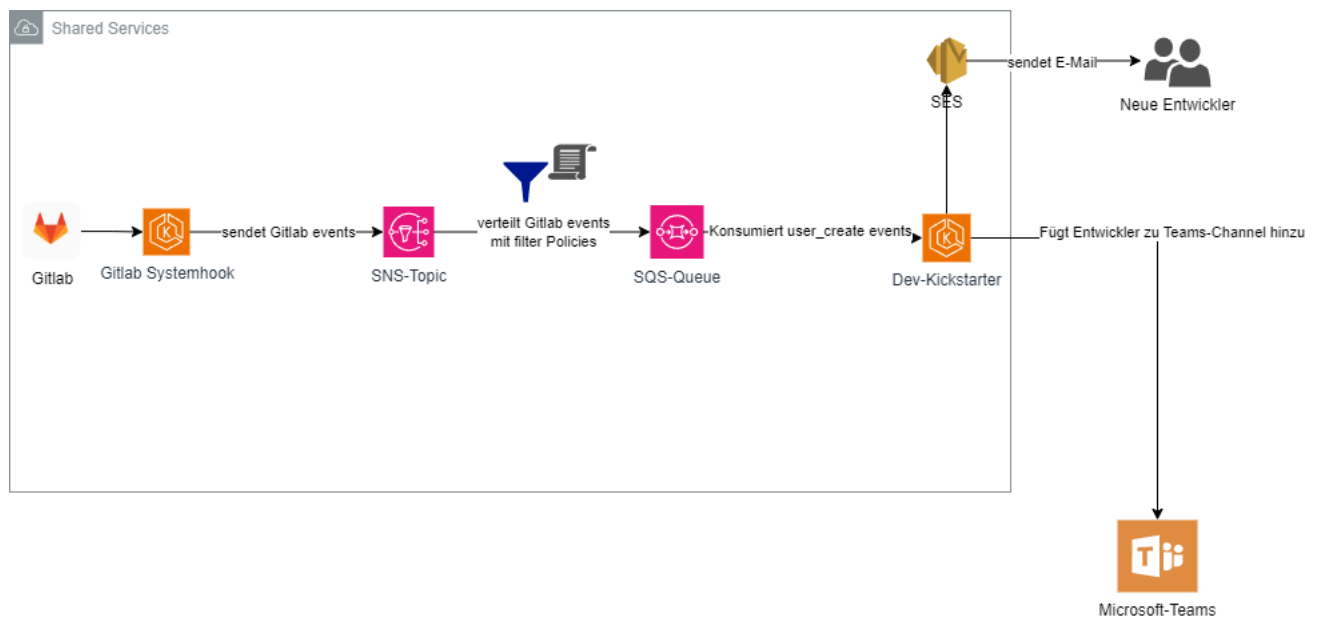


Abbildung 1: Finaler Entwurf des Infrastruktur-Designs und Soll-Zustand

4.2 Architekturdesign

Für das Projekt wurde eine **ereignisgesteuerte Architektur (Event-Driven Architecture)** gewählt. Diese Architektur nutzt lose gekoppelte Microservices, die Informationen durch das Erzeugen und Konsumieren von Ereignissen austauschen. GitLab sendet Systemereignisse über einen System Hook an ein **AWS SNS**-Topic, von dem aus die Ereignisse an verschiedene **SQS**-Warteschlangen weitergeleitet werden. Diese asynchrone Kommunikation ermöglicht eine flexible und skalierbare Verarbeitung.

Ein in **Go** geschriebener Service konsumiert die relevanten Ereignisse von der SQS-Warteschlange. Bei bestimmten Ereignissen, wie z.B. `user_create`, versendet dieser Service Willkommens-E-Mails über **AWS SES** und fügt neue Entwickler zu Microsoft Teams hinzu. Filter Policies auf den SQS-Warteschlangen sorgen dafür, dass nur relevante Ereignisse verarbeitet werden. Diese Architektur gewährleistet Skalierbarkeit, Entkopplung und effiziente Ereignisverarbeitung.

4.3 Entwurf der E-Mail

Das Design der E-Mail wurde zu Beginn der Entwurfsphase skizzenhaft entworfen, um sicherzustellen, dass sie benutzerfreundlich und einladend wirkt. Besonders wichtig war es, die Informationen so aufzubereiten, dass sie nicht überladen sind und die Empfänger, insbesondere neue Entwickler, schnell und einfach erfassen können.

Die Gestaltung fokussierte sich auf eine klare und ansprechende Struktur, die dazu einlädt, die E-Mail aufmerksam zu lesen. Durch regelmäßige Rücksprache mit dem Auftraggeber und iteratives Feedback konnte das Design kontinuierlich optimiert werden. Letztlich wurde die E-Mail in HTML und CSS umgesetzt, um eine moderne und einladende Benutzererfahrung zu gewährleisten.

Beispielentwurf findet sich im Anhang A.8: Oberflächenentwürfe auf Seite viii.

4.4 Maßnahmen zur Qualitätssicherung

Um die Qualität des Projektergebnisses zu sichern, wurden verschiedene Maßnahmen ergriffen. Dazu gehören die Implementierung von Integrationstests und Unit-Tests, die automatisch bei jedem Git-Commit ausgeführt werden. Diese Tests gewährleisten, dass Änderungen am Code keine bestehenden Funktionen beeinträchtigen.

Darüber hinaus fand eine iterative Überprüfung der Codequalität durch erfahrene Mitarbeiter statt. Durch regelmäßige Code-Reviews konnte sichergestellt werden, dass der Code den Qualitätsstandards entspricht und Best Practices eingehalten werden. Diese Kombination aus automatisierten Tests und manuellem Feedback trägt maßgeblich zur hohen Qualität des Projekts bei.

5 Implementierungsphase

Bevor mit der Implementierung begonnen wurde, wurde ein Iterationsplan erstellt. In diesem wurden die einzelnen Schritte und deren Reihenfolge festgelegt. In jeder Iteration wurde eine spezifische Funktionalität umgesetzt und am Ende der jeweiligen Iteration dem Team präsentiert. Dieses Vorgehen folgt den in Abschnitt 2.3 beschriebenen Prinzipien der agilen Softwareentwicklung. Der vollständige Iterationsplan befindet sich im Anhang A.3: Iterationsplan auf Seite iii.

5.1 Implementierung der Gitlab Systemhook

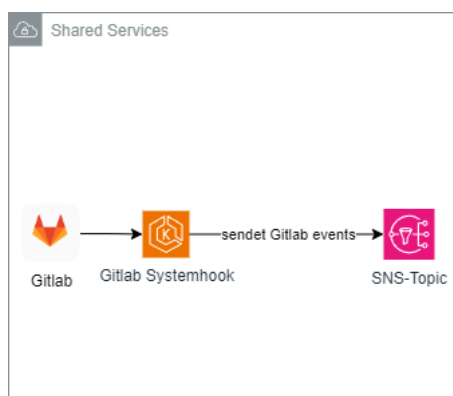


Abbildung 2: Implementierung der Gitlab Systemhook

5.2 Implementierung der GitLab Systemhook

Die Implementierung der GitLab Systemhook ermöglicht es, Ereignisse wie das Erstellen neuer Benutzer in GitLab zu erfassen und diese an ein SNS (Simple Notification Service) Topic weiterzuleiten. Der Service empfängt HTTP-POST-Anfragen von GitLab, validiert das Token und verarbeitet die Payload asynchron, um die Performance zu optimieren.

Hierfür wurde das Echo-Framework in Kombination mit dem AWS SDK für Go eingesetzt. Zu Beginn wurde der Hook auf der Testumgebung des EKS-Clusters (Elastic Kubernetes Service) implementiert. Nach der erfolgreichen Validierung der Funktionalitäten wurde der Code schließlich in die Produktionsumgebung überführt.

Ein Beispiel für die Handhabung der eingehenden Hooks ist in Listing ?? im Anhang zu finden, während die Logik zur Nachrichtenübermittlung an SNS in Listing ?? dargestellt wird. Die Implementierung der Umgebungsvariablen und der Authentifizierung erfolgt im Codeabschnitt in Listing ??.

Die gesamte Lösung ermöglicht eine effiziente und skalierbare Verarbeitung von GitLab-Ereignissen und deren gezielte Weiterleitung an AWS SNS, wodurch ein automatisierter Workflow für das Benutzer-Management geschaffen wird.

5.3 Implementierung der Nachrichtenzustellung an SNS

Nach Empfang eines GitLab-Ereignisses wird die Nachricht, zusammen mit Attributen wie dem Eventnamen, an ein vordefiniertes SNS Topic gesendet. Ein Attributfilter stellt sicher, dass nur relevante Ereignisse wie `user_create` verarbeitet werden. Dadurch wird eine gezielte Filterung und Weiterverarbeitung auf Empfängerseite ermöglicht.

5.4 Bereitstellung und Test

Der Service wurde zunächst auf einer Testumgebung des EKS-Clusters bereitgestellt. Nach erfolgreicher Validierung der Funktionalitäten erfolgte die Bereitstellung in der Produktionsumgebung. Die Lösung ermöglicht eine skalierbare Verarbeitung von GitLab-Ereignissen und deren Weiterleitung an AWS SNS.

2100 active user

150 150

5.5 Implementierung des Dev-Kickstarter

- 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.9: Screenshots der Anwendung auf Seite x.

5.6 Implementierung der E-Mail

- 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.

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.11: Testfall und sein Aufruf auf der Konsole auf Seite xiv. 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.14: Benutzerdokumentation auf Seite xix. Die Entwicklerdokumentation wurde mittels PHPDoc¹ automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse findet sich im Anhang A.10: Entwicklerdokumentation auf Seite xii.

¹Vgl. PHPDOC.ORG [2010]

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

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

Tabelle 3: 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

phpdoc.org 2010

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

Eidesstattliche Erklärung

Ich, Paul Glesmann, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

Entwicklung von Dev Kickstarter – Automatisierte Onboarding- und Ressourcenmanagement-Prozesse bei TUI Group

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.

Hannover, den 18.11.2024

PAUL GLESMANN

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase	6 h
1. Besprechung der bestehenden GitLab-System-Hooks und Identifikation von Verbesserungsmöglichkeiten.	
2. Feedback und Anforderungen der Entwickler und Fachbereiche einholen.	
Entwurfsphase	6 h
1. Design der Systemarchitektur	
2. Entwurf der Infrastruktur auf AWS	
3. Detaillierte Analyse der GitLab-Events	
Implementierungsphase	34 h
1. Entwicklung des Webservers (Go)	14 h
2. Implementierung der GitLab-System-Hook	10 h
3. Infrastruktur-Bereitstellung mit AWS & EKS	10 h
Test- und Kontrollphase	18 h
1. Erstellung von Unit-Tests	8 h
2. Integrationstests	10 h
Dokumentation + Nachbearbeitung	6 h
1. Dokumentation der Codebasis, API-Integrationen und Infrastruktur	3 h
2. Vorstellung der Software im Team und Übergabe	3 h
Erstellen der Projektdokumentation und Präsentation	8 h
Puffer	2 h
1. Puffer für unvorhergesehene Ereignisse: Zeitreserve für unerwartete Herausforderungen oder Verzögerungen im Projektverlauf.	
Gesamt	80 h

A.2 Ressourcen Planung

Hardware

- Büroarbeitsplatz mit Thin-Client

Software

- Windows 7 Enterprise mit Service Pack 1 - Betriebssystem
- Visual Studio Code - Hauptentwicklungsumgebung
- Docker - Containerisierung von Anwendungen
- Go - Programmiersprache zur Entwicklung des Webservers
- AWS - Cloud-Infrastruktur
- EKS (Elastic Kubernetes Service) - Orchestrierung der Container
- Terraform - Infrastruktur als Code
- GitLab - Versionskontrolle und CI/CD
- MS Teams - Zusammenarbeit im Team
- Jira - Projektmanagement

Personal

- Entwickler - Umsetzung des Projektes
- Anwendungsentwickler - Review des Codes

A.3 Iterationsplan

Der folgende Iterationsplan beschreibt die Schritte der Implementierungsphase und deren Reihenfolge. Jede Funktionalität wurde zunächst auf der Testumgebung implementiert und nach erfolgreicher Validierung in die Produktionsumgebung auf dem bestehenden **Amazon EKS Cluster** ausgerollt:

1. **Implementierung der GitLab System Hook:** Entwicklung der GitLab System Hook, die relevante GitLab-Ereignisse an ein SNS Topic sendet.
2. **Erstellung des SNS Topics mit Terraform:** Definition und Bereitstellung des SNS Topics mittels Terraform, um die GitLab-Ereignisse zu empfangen.
3. **Erstellung der SQS Queue mit Filter Policy:** Definition und Bereitstellung einer SQS Queue mit Terraform, inklusive einer Filter Policy, um nur `user_create`-Ereignisse vom SNS Topic zuzulassen.
4. **Implementierung des Dev-Kickstarter Services:** Entwicklung des in Go geschriebenen Dev-Kickstarter Services, der die `user_create`-Ereignisse von der SQS Queue konsumiert.
5. **Extraktion und Verarbeitung der relevanten Daten:** Extraktion der relevanten Informationen, wie die E-Mail-Adresse des erstellten Benutzers, aus den empfangenen GitLab-Ereignissen.
6. **Versand von E-Mails und Hinzufügen zu MS Teams:** Implementierung der Logik, um mithilfe von AWS SES eine Willkommens-E-Mail an den neuen Benutzer zu senden und ihn zu einem MS Teams-Channel hinzuzufügen.

Jeder dieser Schritte wurde iterativ auf der Testumgebung getestet und anschließend auf dem Amazon EKS Cluster in die Produktionsumgebung ausgerollt. Der vollständige Iterationsplan befindet sich im Anhang A.15: Iterationsplan auf S. xiii.

A.4 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.5 Use Case-Diagramm

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

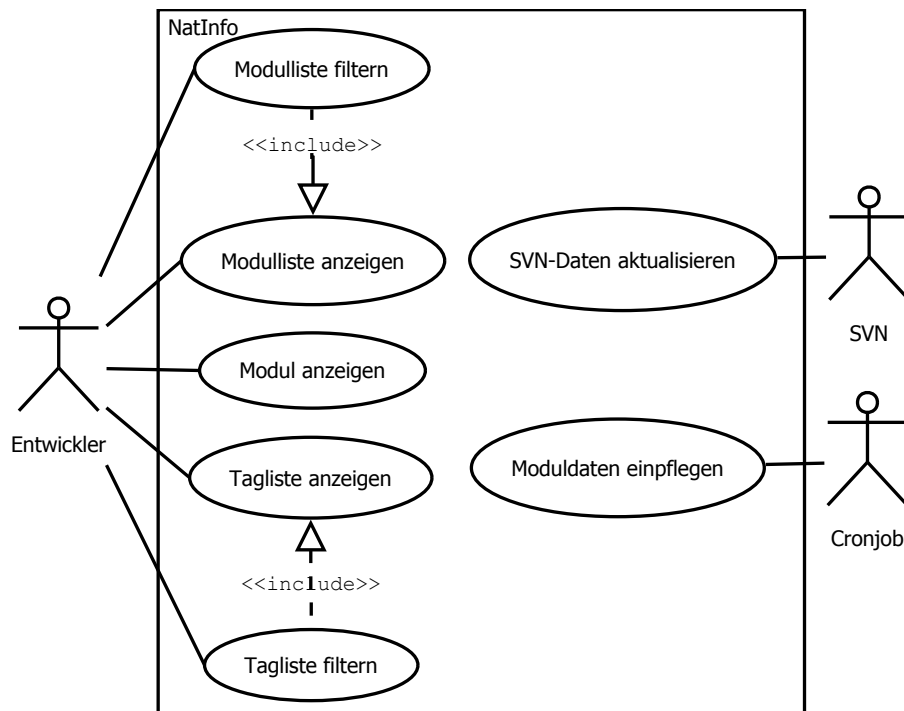


Abbildung 3: Use Case-Diagramm

A.6 Pflichtenheft (Auszug)

Zielbestimmung

1. Musskriterien

1.1. Modul-Liste: Zeigt eine filterbare Liste der Module mit den dazugehörigen Kerninformationen sowie Symbolen zur Einhaltung des Entwicklungsprozesses an

- In der Liste wird der Name, die Bibliothek und Daten zum Source und Kompilat eines Moduls angezeigt.
- Ebenfalls wird der Status des Moduls hinsichtlich Source und Kompilat angezeigt. Dazu gibt es unterschiedliche Status-Zeichen, welche symbolisieren in wie weit der Entwicklungsprozess eingehalten wurde bzw. welche Schritte als nächstes getan werden müssen. So gibt es z. B. Zeichen für das Einhalten oder Verletzen des Prozesses oder den Hinweis auf den nächsten zu tätigenden Schritt.
- Weiterhin werden die Benutzer und Zeitpunkte der aktuellen Version der Sourcen und Kompilate angezeigt. Dazu kann vorher ausgewählt werden, von welcher Umgebung diese Daten gelesen werden sollen.

- Es kann eine Filterung nach allen angezeigten Daten vorgenommen werden. Die Daten zu den Sourcen sind historisiert. Durch die Filterung ist es möglich, auch Module zu finden, die in der Zwischenzeit schon von einem anderen Benutzer editiert wurden.

1.2. Tag-Liste: Bietet die Möglichkeit die Module anhand von Tags zu filtern.

- Es sollen die Tags angezeigt werden, nach denen bereits gefiltert wird und die, die noch der Filterung hinzugefügt werden könnten, ohne dass die Ergebnisliste leer wird.
- Zusätzlich sollen die Module angezeigt werden, die den Filterkriterien entsprechen. Sollten die Filterkriterien leer sein, werden nur die Module angezeigt, welche mit einem Tag versehen sind.

1.3. Import der Moduldaten aus einer bereitgestellten **CSV!**-Datei

- Es wird täglich eine Datei mit den Daten der aktuellen Module erstellt. Diese Datei wird (durch einen Cronjob) automatisch nachts importiert.
- Dabei wird für jedes importierte Modul ein Zeitstempel aktualisiert, damit festgestellt werden kann, wenn ein Modul gelöscht wurde.
- Die Datei enthält die Namen der Umgebung, der Bibliothek und des Moduls, den Programmtyp, den Benutzer und Zeitpunkt des Sourcecodes sowie des Kompilats und den Hash des Sourcecodes.
- Sollte sich ein Modul verändert haben, werden die entsprechenden Daten in der Datenbank aktualisiert. Die Veränderungen am Source werden dabei aber nicht ersetzt, sondern historisiert.

1.4. Import der Informationen aus **SVN!** (**SVN!**). Durch einen „post-commit-hook“ wird nach jedem Einchecken eines Moduls ein **PHP!**-Script auf der Konsole aufgerufen, welches die Informationen, die vom **SVN!**-Kommandozeilentool geliefert werden, an **NatInfo!** übergibt.

1.5. Parsen der Sourcen

- Die Sourcen der Entwicklungsumgebung werden nach Tags, Links zu Artikeln im Wiki und Programmbeschreibungen durchsucht.
- Diese Daten werden dann entsprechend angelegt, aktualisiert oder nicht mehr gesetzte Tags/Wikiartikel entfernt.

1.6. Sonstiges

- Das Programm läuft als Webanwendung im Intranet.
- Die Anwendung soll möglichst leicht erweiterbar sein und auch von anderen Entwicklungsprozessen ausgehen können.
- Eine Konfiguration soll möglichst in zentralen Konfigurationsdateien erfolgen.

Produkteinsatz

1. Anwendungsbereiche

Die Webanwendung dient als Anlaufstelle für die Entwicklung. Dort sind alle Informationen

für die Module an einer Stelle gesammelt. Vorher getrennte Anwendungen werden ersetzt bzw. verlinkt.

2. Zielgruppen

NatInfo wird lediglich von den **Natural!** (**Natural!**)-Entwicklern in der EDV-Abteilung genutzt.

3. Betriebsbedingungen

Die nötigen Betriebsbedingungen, also der Webserver, die Datenbank, die Versionsverwaltung, das Wiki und der nächtliche Export sind bereits vorhanden und konfiguriert. Durch einen täglichen Cronjob werden entsprechende Daten aktualisiert, die Webanwendung ist jederzeit aus dem Intranet heraus erreichbar.

A.7 Datenbankmodell

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

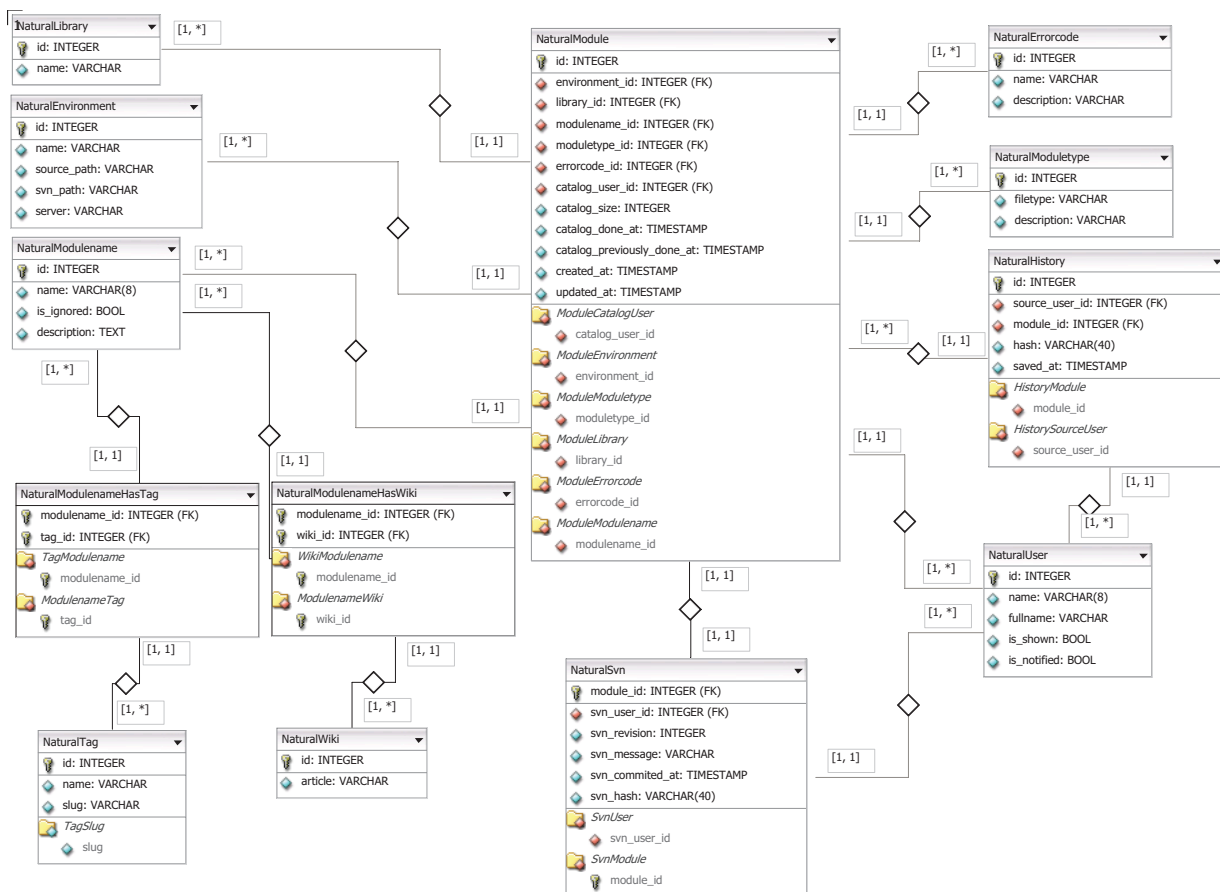


Abbildung 4: Datenbankmodell

A.8 Oberflächenentwürfe

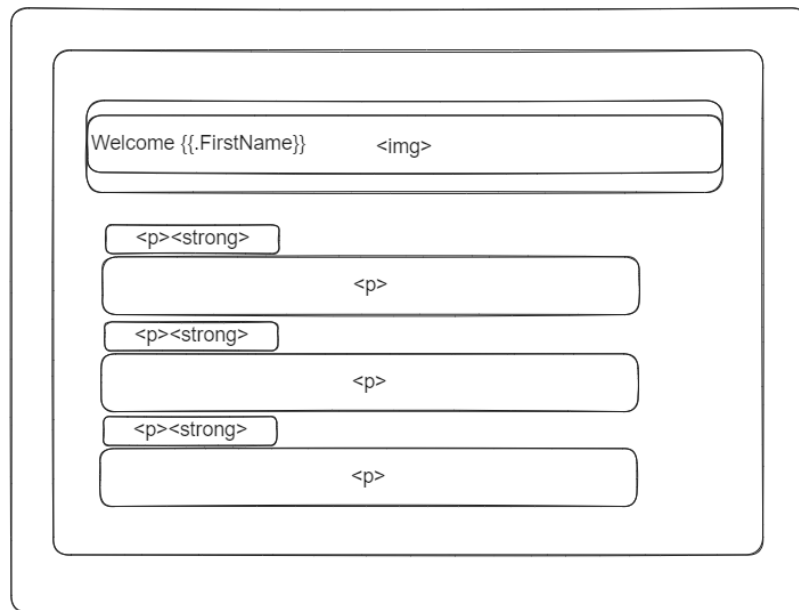


Abbildung 5: Erster Entwurf der E-Mail

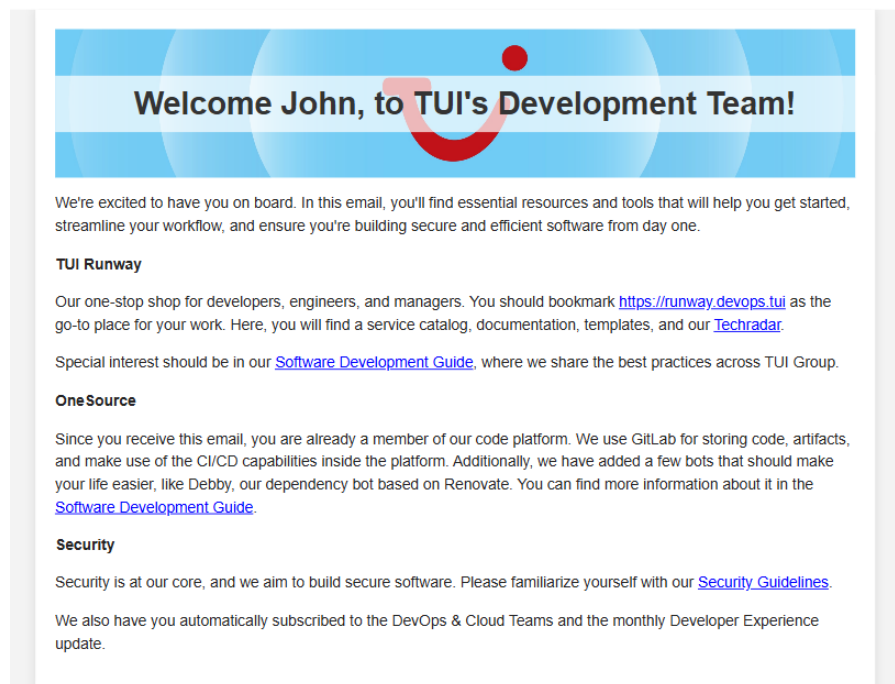


Abbildung 6: Umsetzung in HTML mit CSS

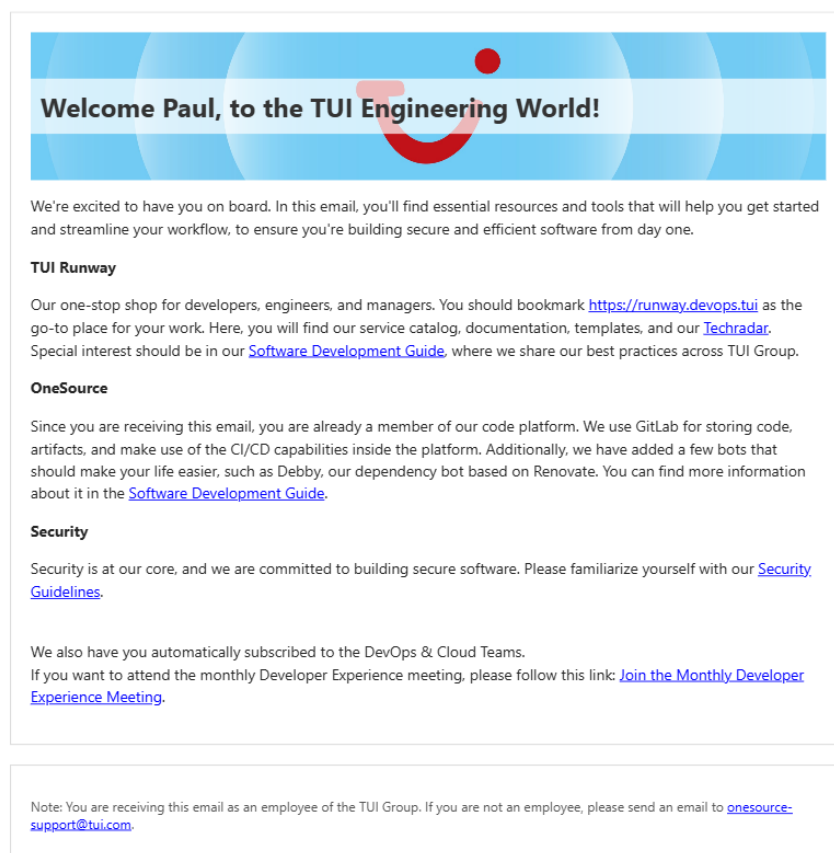
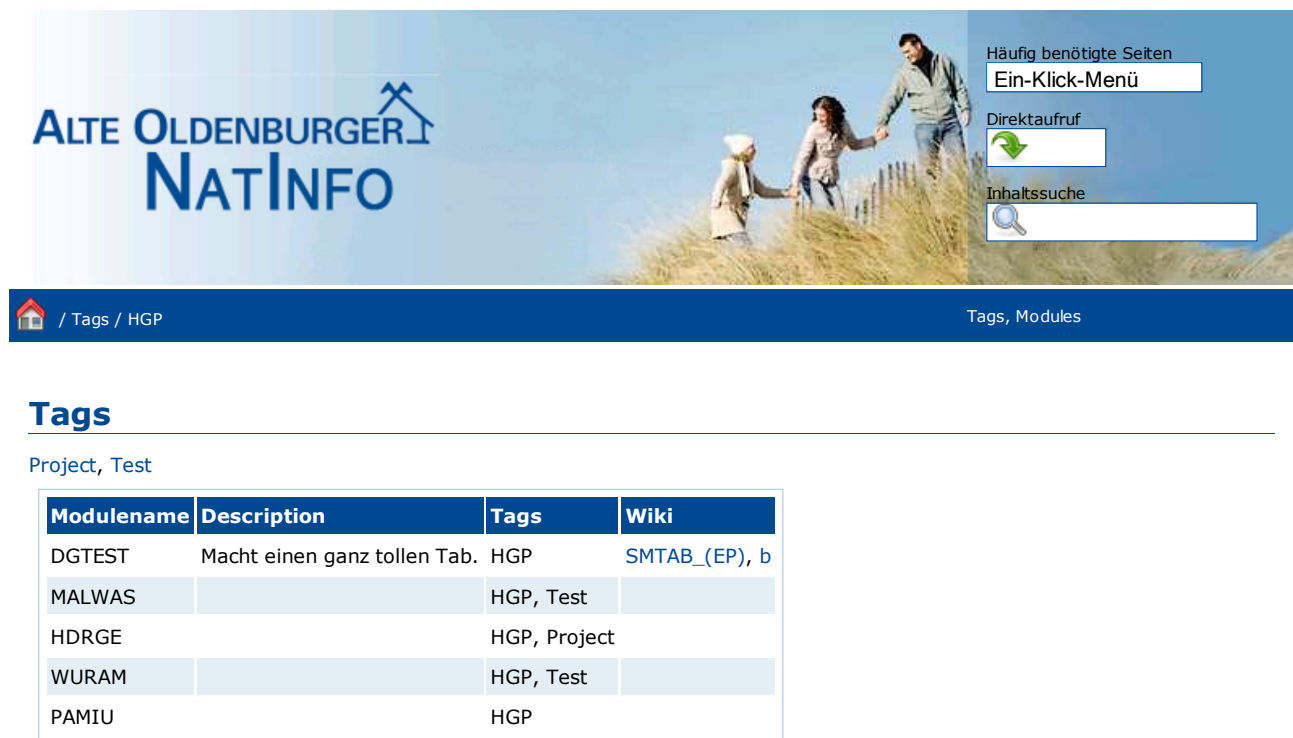


Abbildung 7: Finales Design der E-Mail

A.9 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



Modules

Environment	ENTW
Library	Select
Catalog user	Select
Catalog date	Select
Source user	Select
Source date	Select
Reset Filter	











Name	Library	Source	Catalog	Source-User	Source-Date	Catalog-User	Catalog-Date
SMTAB	UTILITY			MACKE	01.04.2010 13:00	MACKE	01.04.2010 13:00
DGTAB	CON			GRASHORN	01.04.2010 13:00	GRASHORN	01.04.2010 13:00
DGTEST	SUP			GRASHORN	05.04.2010 13:00	GRASHORN	05.04.2010 13:00
OHNETAG	CON			GRASHORN	05.04.2010 13:00	GRASHORN	01.04.2010 15:12
OHNEWIKI	CON			GRASHORN	05.04.2010 13:00	MACKE	01.04.2010 15:12

Abbildung 9: Liste der Module mit Filtermöglichkeiten

A.10 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](#)

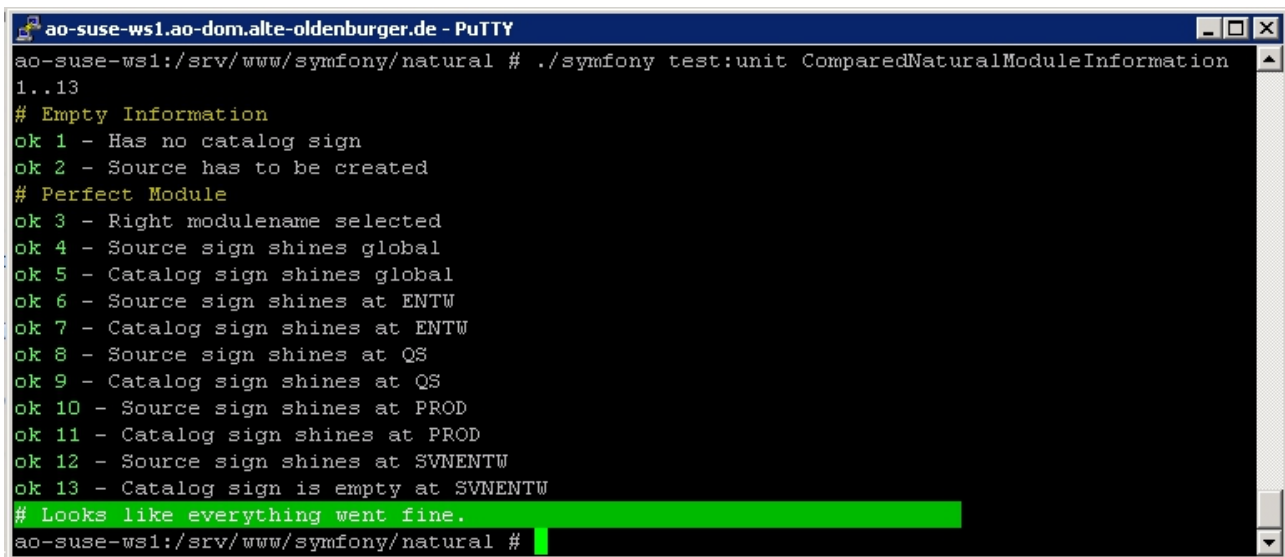
A.11 Testfall und sein Aufruf auf der Konsole

```

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(NaturalmodulenamePeer::NAME, 'SMTAB');
14 $moduleName = NaturalmodulenamePeer::doSelectOne($criteria);
15 $t->is($moduleName->getName(), 'SMTAB', 'Right modulename 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.12 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.13 Klassendiagramm

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

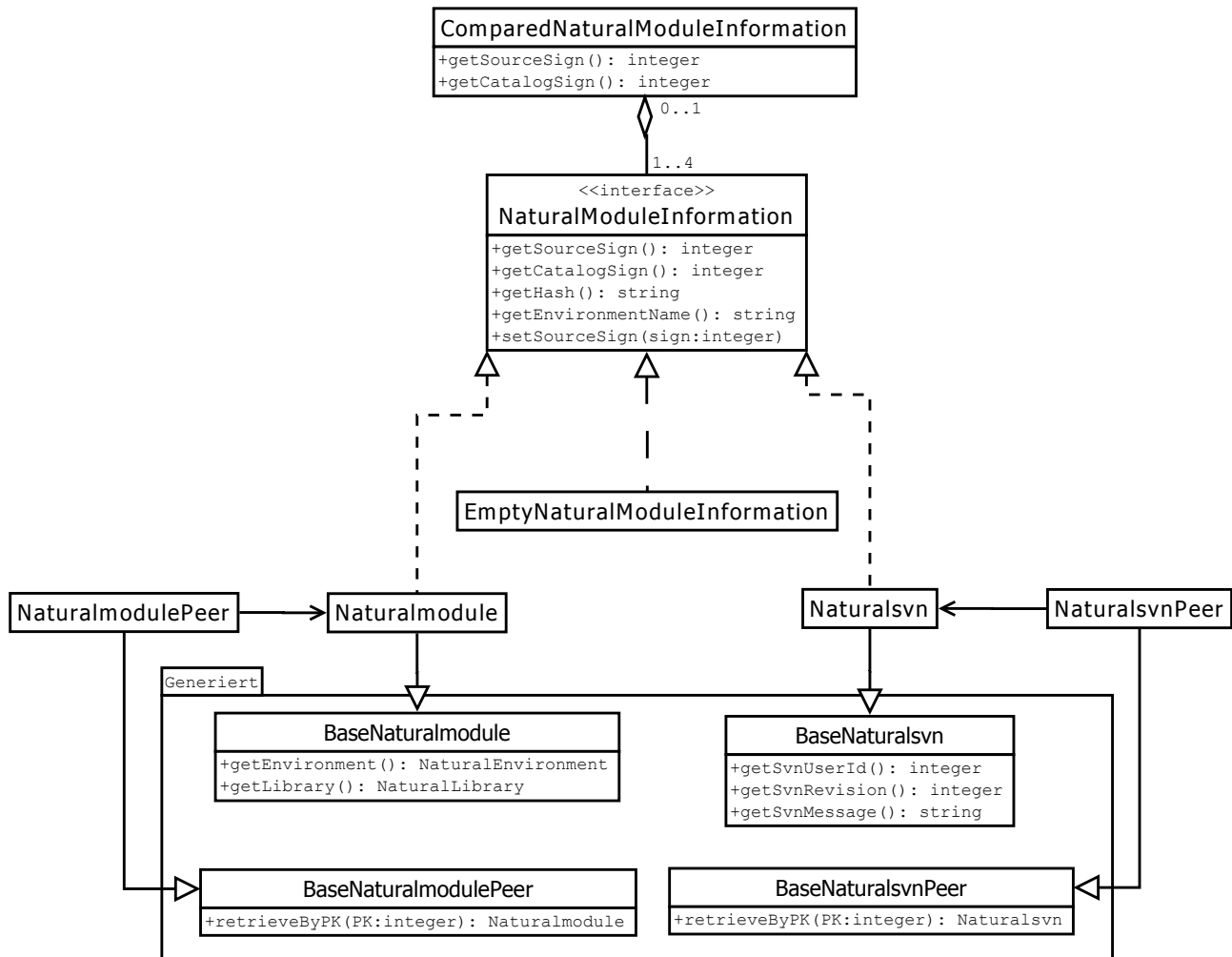







Abbildung 11: Klassendiagramm

A.14 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.