**Architekturdokumentation**

MedDevMM

erstellt von

*Template Revision: 6.0 DE (Release Candidate)  
19. März 2012*

|  |  |
| --- | --- |
| We acknowledge that this document uses material from the arc 42 architecture  template, <http://www.arc42.de>. Created by Dr. Peter Hruschka & Dr. Gernot Starke. For additional contributors see arc42.de/about/contributors.html |  |

**Änderungsübersicht**

| **Version** | **Datum** | **Bearbeiter** | **Beschreibung** |
| --- | --- | --- | --- |
| 0.1 | 27.10.2014 | Mario Murrent | Basisversion |
| 0.2 | 02.11.2014 | Roland Lehner | Bearbeitung Punkt 8.2. Broker und Pipes und Filters Muster hinzugefügt |
| 0.3 | 12.11.2014 | Daniel Kienböck | Patternentscheidungen und Beschreibung |
| 0.4 | 16.11.2014 | Bernhard Stöckl | Bearbeitung F/R , Command-Processor |
| 0.5 | 17.11.2014 | Mario Murrent | Qualitätsbaum hinzugefügt |
| 0.6 | 18.11.2014 | Mario Murrent | Risiken, Stakeholder, Qualitätsziele hinzugefügt |
| 0.7 | 20.11.2014 | Mario Murrent | Bewertungsszenarien hinzugefügt |
| 0.8 | 21.11.2014 | Daniel Kienböck | Konzepte + Tippfehler ausbessern |
| 0.9 | 05.12.2014 | Daniel Kienböck | Feedback einarbeiten |
| 0.10 | 22.12.2014 | Mario Murrent | Randbedingungen ausgebaut |
| 0.11 | 31.12.2014 | Daniel Kienböck | Domänenmodel |
| 0.12 | 09.01.2014 | Roland Lehner |  |

**Basisdokumente**

|  |  |
| --- | --- |
| **Dokument** | **Beschreibung** |
|  |  |
|  |  |

Inhaltsverzeichnis

1. Einführung und Ziele 5

1.1 Aufgabenstellung 5

1.2 Qualitätsziele 5

1.3 Stakeholder 5

2. Randbedingungen 7

2.1 Technische Randbedingungen 7

2.2 Organisatorische Randbedingungen 8

2.3 Konventionen 9

3. Kontextabgrenzung 9

3.1 Fachlicher Kontext 9

3.2 Technischer- oder Verteilungskontext 9

4. Lösungsstrategie 9

5. Bausteinsicht 10

5.1 Ebene 1 10

5.2 Ebene 2 10

5.3 Ebene 3 14

6. Laufzeitsicht 14

6.1 Laufzeitszenario 1 14

6.2 Laufzeitszenario 2 14

6.3 ... 14

6.4 Laufzeitszenario n 14

7. Verteilungssicht 14

7.1 Infrastruktur Ebene 1 14

7.2 Infrastruktur Ebene 2 15

8. Konzepte 15

8.1 Fachliche Strukturen und Modelle 15

8.2 Typische Muster und Strukturen 15

8.3 Persistenz 15

8.4 Benutzungsoberfläche 16

8.5 Ergonomie 16

8.6 Ablaufsteuerung 16

8.7 Transaktionsbehandlung 16

8.8 Sessionbehandlung 16

8.9 Sicherheit 16

8.10 Kommunikation und Integration mit anderen IT-Systemen 16

8.11 Verteilung 16

8.12 Plausibilisierung und Validierung 16

8.13 Ausnahme-/Fehlerbehandlung 16

8.14 Management des Systems & Administrierbarkeit 16

8.15 Logging, Protokollierung, Tracing 16

8.16 Geschäftsregeln 16

8.17 Konfigurierbarkeit 17

8.18 Parallelisierung und Threading 17

8.19 Internationalisierung 17

8.20 Migration 17

8.21 Testbarkeit 17

8.22 Skalierung, Clustering 17

8.23 Hochverfügbarkeit 17

9. Entwurfsentscheidungen 17

9.1 Entwurfsentscheidung 1 17

9.2 Entwurfsentscheidung n 17

10. Qualitätsszenarien 17

10.1 Qualitätsbaum 18

10.2 Bewertungsszenarien 18

11. Risiken 18

12. Glossar 18

Anmerkung: In der Microsoft-Word-Variante enthält dieses Template Anleitungen und Ausfüllhinweise als „ausgeblendeten Text“. Durch den Befehl „Formate ein-/ausblenden“ können Sie die Anzeige dieser Texte bestimmen.

# Einführung und Ziele

## Aufgabenstellung

Es soll ein System für den medizinischen Bereich, mit dem Fokus auf Krankenhäuser, entwickelt werden, mit dem medizinische Daten aus unterschiedlichen Quellen bzw. Quellsystemen gesammelt und abgerufen werden können. Für die Speicherung der Daten soll ein zentrales Datenmanagement zuständig sein. Das Abrufen und Anzeigen der Daten soll auf unterschiedlichen Devices möglich sein mit dem speziellen Fokus auf mobile Devices wie z.B.: Tablet oder Handy. Das System soll sowohl für Ärzte als auch für Patienten nutzbar sein.

## Qualitätsziele

|  |  |  |
| --- | --- | --- |
| Prio | Qualitätsmerkmal | Ziel |
| 1 | Verfügbarkeit | Die Software soll einen hohe Verfügbarkeit erzielen |
| 2 | Bedienbarkeit | Die Software soll intuitiv bedienbar sein. Bedienelemente werden entsprechend der Benutzerrechte ein- bzw. ausgeblendet |
| 3 | Unabhängigkeit | Das System soll unabhängig von der darunterliegenden Plattform funktionieren |
| 4 | Erweiterbarkeit | Die Erweiterung des bestehenden System soll so einfach wie nur möglich gehalten werden |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rolle | Beschreibung | Ziel/Intention | Kontakt | Bemerkung |
| Projektmanager |  |  |  |  |
| Auftraggeber |  |  |  |  |
| Software Architekt |  |  |  |  |
| Anwender |  |  |  |  |
| Patienten |  |  |  |  |
| Krankenhausangestellte |  |  |  |  |
| IT Service |  |  |  |  |

## Stakeholder

# Randbedingungen

Beim Lösungsentwurf sind verschiedene Randbedingungen zu beachten. Dieser Abschnitt stellt sie dar und erklärt auch, wenn nötig, deren Motivation.

## Technische Randbedingungen

|  |  |
| --- | --- |
| Randbedingung | Erläuterung |
| Moderate Hardwareausstattung | Die Lösung muss auf marktüblichen Geräten, wie Tablets, Notebooks, PCs betrieben werden können. |
| Implementierung in HTML5 / Javascript | Die Implementierung soll auf HTML5 / Javascript basieren um eine plattformunabhängigkeit zu gewährleisten |

## Organisatorische Randbedingungen

|  |  |
| --- | --- |
| Randbedingung | Erläuterung |
| Team |  |
| Zeitplan |  |
| Entwicklungswerkzeuge | Der Entwurf erfolgt mit Stift und Papier, ergänzend Visio / Pencil oder Enterprise Architect. |
| Konfigurations- und Versionsverwaltung | Subversion |
| Testwerkzeuge |  |
| Vorgehensmodell |  |
| Projektbudget | Für die Entwicklung wurde ein Festpreis vereinbart |

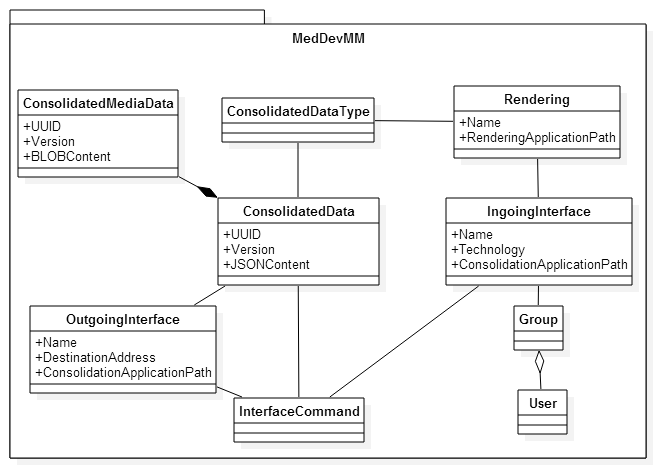
## Konventionen

|  |  |
| --- | --- |
| Konvention | Erläuterung |
| Architekturdokumentation | Gliederung nach dem deutschen arc42-Template |
| Coderichtlinien für JavaScript | siehe CodingGuidelines JS 1.5.docx |
| Modelle | UML wird zur Erstellung von Diagrammen und zur einfachen Beschreibung verwendet |
| Sprache | Innerhalb dieser (deutschen) arc42-Architekturdokumentation werden Dinge (Komponenten, Schnittstellen) in Diagrammen und Texten deutsch benannt |

# Kontextabgrenzung

## Fachlicher Kontext

Namensgebung aus Sicht des Endbenutzers



* IngoingInterface repräsentiert die Daten eines Endgerät-Interfaces
* Daten zu Authentication-System beim IngoingInterface (Group/User)
* IngoingInterface erzeugt ein nachvollziehbares InterfaceCommand zur Abarbeitung
* IngoingInterface repräsentiert eine Ausgabe (Rendering) einer Datenart
* InterfaceCommand wird von IngoingInterface generiert und am OutgoingInterface im zusammenspiel mit den zupassenden Daten in einer gewissen Version ausgeführt
* Das OutgoingInterface repräsentiert ein internes System
* Daten kommen vom IngoingInterface und werden dort in das konsolidierte Format übertragen
* Zusätzlich zu den konsolidierten Daten können noch weitere Infos gespeichert werden

## Technischer- oder Verteilungskontext

## Externe Schnittstellen

Zu Datenquellen, variabel integrierbar, wenn zupassende bridge gebaut und im Broker registriert

Aktuell:

- Datenformat 1: medizinische Befunde (Textfiles, Datenquelle Radiologie)

- Datenformat 2: Bilder (DICOM, DQ: PACS-App)

- streaming!

# Lösungsstrategie

Webpage

# Bausteinsicht

**Middleware:**

* **Schnittstellenmodul (gelb)**
* **Anfragenmodul (grün)**

Anfrage

Schnittstelle

Kern

* **Kern (blau)**

Middleware

Client 1

Client 2

Client 3

Client 4

Client 5

Client 6

…

Client X

Datenquelle1

Datenquelle2

Datenquelle3

Datenquelle4

Datenquelle5

Datenquelle6

…

DatenquelleX

Cmd-Pattern

Master/Slave  
(bridged)

Broker

Pipes / Filters

F/R

Anfrage immer gleich:

* Pipes and Filters ändert die Anfrage sodass die Anfrage für das Endgerät optimiert werden kann (Mobile / Tablet / Browser) + eigener Filter für Authentifizierung
* Broker gibt über „Forward Receiver“ (IPC) dem Datenquellen-Master bekannt „wo was wie“ … Master instanziiert einen gebridgeten Slave und speichert sich über Command-Pattern

# Laufzeitsicht

# Verteilungssicht

# Konzepte

Dieser Abschnitt beschreibt allgemeine Strukturen und Aspekte, die systemweit gelten. Darüber hinaus stellt er verschiedene technische Lösungskonzepte vor.

## Fachliche Strukturen und Modelle

## Typische Muster und Strukturen

### Nicht funktionale Muster

### Broker

Der Broker ist dafür verantwortlich die Anfrage an das bestehende Service weiterzuleiten (mapping von Request auf Data-Source). Der Broker nimmt die bereits angepasste Anfrage (siehe pipes and filters) entgegen und fragt die Datenquellen an. Sollten neue Datenquellen hinzugefügt werden, muss der Broker um die Datenquellen upgedatet werden.

### Master-Slave

Master

+ result

+launch()  
*+makeSlave() : Slave*

*Slave*

+ result

*+ run()*

ConcreteSlave

+ run()

ConcreteMaster

+ makeSlave() : Slave

Für die Anfrage der Daten aus verschiedenen Schnittstellen wir prinzipiell im Schnittstellen – Modul ein Objekt als Schnittstellen-Master (Singleton) instanziiert und in Folge vom Master pro Anfrage ein (paralleler) Slave erstellt (siehe Bridge-Pattern zu Implementierungsdetails des Slave).

### Command-Prozessor

Clients können die Änderungen(commands) an Daten in den Ressourcen wieder rückgängig machen. Sobald ein Client seine Änderungen speichert wird das Command in einer Liste gespeichert und andere Clients können über die undo-Funktion die Änderungen bzw. das Command rückgängig machen.



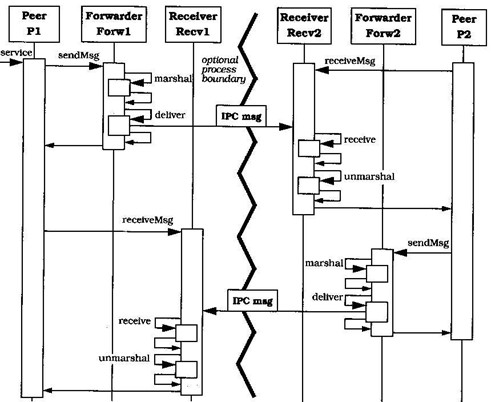
### Funktionale Muster

### Forward Receiver

Das Forward-Receiver Pattern (siehe Abb.) bietet die Möglichkeit das senden und empfangen von Nachrichten, welche von den verschiedenen Clients kommen zu entkoppeln. Eingesetzt wird der F/R im aktuellen Projekt in der Middleware zwischen Master/Slave und Broker um eine asynchrone IPC innerhalb der Middleware sicher zu stellen.



Über den Broker kommende Anfragen werden über dessen Forwarder an den Receiver des Master gestellt. Die folgende Abbildung zeigt den zeitlichen Ablauf einer IPC über das F/R Pattern.



### Pipes and Filters

Das Pipes and Filters Architekturmuster, kommt bei der Anfrage der Clients zum Einsatz. Dieser Prozess wird in der Middleware durchgeführt. Es werden somit dem Endgerät die optimierten Daten zur Verfügung gestellt, die es benötigt. Zum Beispiel soll ein mobiles Endgerät keine großen Bilderdaten erhalten, um den Datenkonsum klein zu halten. Darüber hinaus kann hier auch ein Authentifizierungsfilter implementiert werden.

Bsp.: = Pipe

Endgerättypenbestimmung

Authentifizierung

Bildfilter

### Bridge

Abstraction

- impl : Implementor

+ function()

*Implementor*

+ implementation()

ConcreteImplementor

+ implementation()

RefinedAbstraction

+ function()

Da viele verschiedene Schnittstellen (und dessen libraries) unter einen Hut gebracht werden müssen, wurde das Bridge-Pattern gewählt, um hier ein gemeinsames (triviales) Interface für jede Schnittstelle verwenden zu können. Jede neue Schnittstelle zu einem Fremdsystem kann somit über die „Abstraction“ (siehe Bild; interface zur Verwendung) aufgerufen werden. Die RefinedAbstraction (siehe Bild; Code, der den Zugriff der Abstraktion auf die konkrete Library mapped) muss beim Integrationsprozess ausdefiniert werden.

**Anwendungsfall Schnittstellen**

* Medizinische Befunde mit Bildern werden beim Holen der Daten konsolidiert und werden in einem standardisierten JSON-Format ausgegeben bzw. Bilder im Portable Network Graphics (png)-Format mit einem eindeutig vergebenen Namen hinzugefügt (UUID) mit Referenz im JSON-File.
* Mit demselben JSON/PNG Format werden DICOM Files gelesen und genauso konsolidiert gespeichert.

Über das Pattern ist es möglich dem implementierenden Team nicht ein Interface sondern nur ein Requirement (also abstrakte Anforderung bspw. von einem reinen, nicht technischen Projekt-Manager) vorzugeben (bzw. Software zuzukaufen, wobei man dem Hersteller meist nur schwer eine Schnittstelle vorgeben kann). Das Requirement wird dann implementiert, getestet und über die RefinedAbstraction in das Projekt von einem - unter Umständen anderen - Team integriert.

## Persistenz

In Quellsystem wird die Persistenz gewährleistet.

## Benutzungsoberfläche

Grundsätzlich wird die Benutzeroberfläche in einem Webinterface dargestellt. Wird das System von einem mobilen Endgerät angesteuert, kann eine mobile Website, oder eine native Implementierung der Benutzeroberfläche zu Grunde liegen.

## Ergonomie

## Ablaufsteuerung

## Transaktionsbehandlung

In Geschäftstransaktionen VCRP (Visibility; Consistency; Recovery; Permanence) statt ACID. Recovery über das Command Pattern.

## Sessionbehandlung

## Sicherheit

Eine Anfrage auf einen Datensatz wird mit Hilfe eines Zugriffstokens gewährleistet. Zu Grunde liegt das Pipes and Filters Pattern, wo Sicherheitsfilter implementiert werden, die diesen Token überprüfen.

## Kommunikation und Integration mit anderen IT-Systemen

Wird ein neues System angeschlossen, muss dieses ein Interface implementieren, über welches die Systeme miteinander kommunizieren können. Hierfür wird ein Bridge Pattern verwendet, welches alle Verbindungen unter einen Hut bringen kann, um so eine einfache Erweiterbarkeit zu gewährleisten.

## Verteilung

## Plausibilisierung und Validierung

## Ausnahme-/Fehlerbehandlung

## Management des Systems & Administrierbarkeit

## Logging, Protokollierung, Tracing

Kommt es zu Problemen im System wird dies in einem Logfile protokolliert. In diesem Logfile kann der Fehler mit der jeweiligen Komponente in Verbindung gebracht werden. Google Analytics kommt zum Einsatz, um genau nachvollziehen zu können bei welchem User Interface der Fehler aufgetreten ist. Falls es Datenschutzbedenken geben sollte, kann eine eigene Lösung für Traceability auf einem eigenen Server gehostet werden. Hierfür müssten Produkte wie zum Beispiel Webtrekk zugekauft werden.

## Geschäftsregeln

## Konfigurierbarkeit

Der Broker ist dafür verantwortlich, dass die Endgeräte die richtigen Daten bekommen. Hier kann für jedes Endgerät eine Konfiguration durchgeführt werden, Auflösung, Ressourcen usw. Auch Services sind auf die einzelnen Endgeräte individuell konfigurierbar.

## Parallelisierung und Threading

Parallelisierung und Threading wird im Master/Slave Pattern behandelt. Werden zwei geleichzeitige Anfragen auf ein Service gesendet, so werden auch zwei Slaves erstellt, die diese Anfrage abarbeiten.

## Internationalisierung

Bei der Internationalisierung wird auf die Spracheinstellung des jeweiligen Endgerätes zurückgegriffen. Die Sprachfiles befinden sich bei den Webanwendungen (Mobil/Desktop) direkt auf dem Server. Bei nativen Anwendungen für mobile Endgeräte muss für jede Plattform ein eigenes Sprachfile erstellt werden.

## Migration

Eine Migration ist nicht notwendig, da die Middleware zu existierenden Services geschaffen wird. Die Datenquellen bleiben unverändert.

## Testbarkeit

* Unit-Test
* Integrationstest
* Äquivalenzklassentest pro Dokumententyp

## Skalierung, Clustering

## Hochverfügbarkeit

## Codegenerierung

## Buildmanagement

# Entwurfsentscheidungen

In diesem Kapitel wird vor allem auf die Entwurfsentscheidungen im Bereich der gewählten Muster (Patterns)

* Command -Pattern
* Master-Slave -Pattern
* Forward/Reveiver -Pattern
* Broker -Pattern
* Pipes / Filters -Pattern
* Bridge -Pattern

unter Berücksichtigung der Qualitätsziele:

* Verfügbarkeit
* Bedienbarkeit
* Unabhängigkeit
* Erweiterbarkeit

eingegangen.

## Entscheidung 1 – Bridged Master/Slave Pattern für Quellsysteme

Das Schnittstellenmodul zu den Quellsystemen ist zentraler Baustein für diese Middleware-Applikation. Hierbei sollte beachtet werden, dass so ein zentrales Modul so performant wie möglich gestaltet werden soll.

Das Pattern Master/Slave dient der Verfügbarkeit / Hoch-Verfügbarkeit. Anstatt sequenzielle Abfragen abzuarbeiten werden die Abfragen pro Quelle parallel pro Datenquelle (=1 Slave pro Datenquelle) ausgeführt (bridged).

Annahme dabei ist, dass die Anfragen gut verteilt auf die Datenquellen kommen, um wirklich einen Performance-Gewinn durch das Pattern zu erreichen.

Alternativ wäre es möglich einen Load-Balancer zu verwenden, jedoch kann bei dedizierten Verbindungen jede Verbindung zum Quellsystem effektiver genutzt werden (keine Zeitverluste durch connection-establishing).

## Entscheidung 2 – Pipes and Filters

Bedienbarkeit, Unabhängigkeit und Erweiterbarkeit sind Argumente, die alle in eine Richtung schlagen: keiner weiß was morgen ist. Es geht um die Möglichkeit schnell und flexibel auf verschiedenste, sinnvolle Endgeräte eine akkurate und gut bedienbare Lösung zu schaffen. So gilt aktuell noch Android Wear (bspw. Uhren mit Android-Betriebssystem, die mit dem Handy gekoppelt Nachrichten anzeigen können und über Audio-Interface einfache Dienste wie Mail und SMS schreiben übernehmen können) als absolutes Entwickler-Hacker-Spielzeug. Es ist jedoch durchaus möglich, dass sich diese Technologie in den nächsten 3-5 Jahren im Endkundensegment etabliert und durchgesetzt haben wird. Es ist in der Lösung somit angestrebt, dass auch für solche „esoterischen“ Geräte als auch noch nicht berücksichtigbare Softwarelösungen berücksichtigt werden und die Lösung leicht erweiterbar ist, um diesen kommenden Herausforderungen stand zu halten.

Annahme hierbei ist, dass das Internet als solches mit dem Konzept von Webseiten und einem Browser sich zwar weiter-entwickeln wird, jedoch nicht komplett als Technologie wie wir sie jetzt kennen abgelöst wird. Anzubindende Endgeräte haben entweder eine Kommunikationsschnittstelle, die Daten in einem definierten Format zu erhalten haben (Bspw. REST-Service mit XML-Daten für eine i-Phone App) oder einen Browser installiert haben, auf den eine Webseite optimiert auf das Endgerät (Bspw. Webseite am Surface-Tablet sieht anders aus als auf dem Linux-Desktop-System) angezeigt wird.

Pipes and Filters bieten hierbei die gewünschten Möglichkeiten. Beispielsweise können verschiedene Informationen je nach Authentifizierung durch einen Filter entfernt werden oder aus verschiedenen Renderings (Ausgabeformaten) optimiert für das Endgerät gewählt werden. Bilder können für kleine Geräte heruntergerechnet und im Filterelement (als Mini-Baustein) ge-cache-t werden, um auch hier Performance-Gewinne zu erzielen. Komplett neue Anforderungen können durch das Hinzufügen eines neuen Filters bewältigt werden.

Alternativ könnte man verschiedene Endpunkte angeben, die verschieden abgearbeitet werden und die jede Funktionalität (bspw. Authentifizierungsabhängige Sichten) ausprogrammieren, jedoch ist hierbei eine - auf den ersten Blick - komplex wirkende Lösung zu entwickeln, wie man gemeinsame Teile wiederverwenden kann. Außerdem ist diese Art der Anbindung für den Anwender nicht transparent.

# Qualitätsszenarien

## Qualitätsbaum





## Bewertungsszenarien

|  |  |
| --- | --- |
| Nr | Szenario |
| 1 | Ein Interessierter mit Grundkenntnissen in UML möchte einen Einstieg in die Architektur von MedDevMM finden.  Lösungsstrategie und Entwurf erschließen sich innerhalt einer Stunde. |
| 2 | Ein Architekt, der arc42 anwenden möchte, sucht zu einem beliebigen Kapitel des Templates einen konkreten Beispielinhalt und findet ihn unverzüglich in der Dokumentation |
| 3 | Ein erfahrener Webentwickler sucht die Implementierung eines im Entwurf beschriebenen Bausteins.  Er findet sie ohne Umwege oder fremde Hilfe im Quelltext |
| 4 | Ein System Engineer integriert ein neues Gerät in das bestehende System. Er kann dies ohne Probleme und zusätzlichen Aufwand tun |
| 5 | Das System wird auf Grund geänderter Anforderungen umgebaut. Durch die sauber definierten Schnittstellen des Systems und die gewählte Architektur ist es möglich Änderungen ohne Herunterfahren des Systems zuzulassen. |
| 6 | Der Anwender gibt seine Logindaten ein und drückt auf Login. Die Benutzerdaten werden vom validiert und liefert entweder eine Fehlermeldung zurück oder leitet den Anwender zur vorgesehenen Seite weiter. |

# Risiken

Die folgenden Risiken wurden zu Beginn des Projekts identifiziert. Diese beeinflussten die Planung der Iterationen maßgeblich.

|  |  |  |
| --- | --- | --- |
| Prio | Risiko | Erläuterung |
| 1 |  | Die Schnittstellen ändern sich ungewollt gegen Ende des Projekts |
| 2 |  | Das System ist zum definierten Zeitpunkt nicht ausreichend getestet |
| 3 |  | Die verwendeten Frameworks bzw. Libraries weisen nicht die gewollte Stabilität auf |
| 4 |  | Nichterreichen der verlangten Leistungen |

# Glossar