**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 |
|  |  |  |  |
|  |  |  |  |

**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

(engl.: Introduction and Goals)

Als Einführung in das Architekturdokument gehören hierher die treibenden Kräfte, die Software-Architekten bei Ihren Entscheidungen berücksichtigen müssen:

Einerseits die Erfüllung bestimmter fachlicher Aufgabenstellungen der Stakeholder, darüber hinaus aber die Erfüllung oder Einhaltung der vorgegebenen Randbedingungen (required constraints) unter Berücksichtigung der Architekturziele.

## Aufgabenstellung

(engl.: Requirements Overview)

Kurzbeschreibung der fachlichen Aufgabenstellung, Extrakt (oder Abstract) der Anforderungsdokumente.

Verweis auf ausführliche Anforderungsdokumente (mit Versionsbezeichnungen und Ablageorten).

Inhalt

Eine kompakte Zusammenfassung des fachlichen Umfelds des Systems. Beantwortet (etwa) folgende Fragen:

1. Was geschieht im Umfeld des Systems?
2. Warum soll es das System geben? Was macht das System wertvoll oder wichtig? Welche Probleme löst das System?

Motivation

Aus Sicht der späteren Nutzer ist die Unterstützung einer fachlichen Aufgaben der eigentliche Beweggrund, ein neues (oder modifiziertes) System zu schaffen.

Obwohl die Qualität der Architektur oft eher an der Erfüllung von nicht-funktionalen Anforderungen hängt, darf diese wesentliche Architekturtreiber nicht vernachlässigt werden.

Form

Kurze textuelle Beschreibung, eventuell in tabellarischer Use-Case Form.

In jedem Fall sollte der fachliche Kontext Verweise auf die entsprechenden Anforderungsdokumente enthalten.

Beispiele

Kurzbeschreibung der wichtigsten:

1. Geschäftsprozessen,
2. funktionalen Anforderungen,
3. nichtfunktionalen Anforderungen und andere Randbedingungen (die wesentlichen müssen bereits als [Architekturziele](#xpointer(/descendant-or-self::ap:Topic[@OId='UqOtqMbG6UanMkVohgRaNQ=='])) formuliert sein oder tauchen als [Randbedingungen](#xpointer(/descendant-or-self::ap:Topic[@OId='XHXfmljU/U+rsiRTMnQgRQ=='])) auf) oder
4. Mengengerüste.
5. Hintergründe

Hier können Sie aus den Anforderungsdokumenten wiederverwenden - halten Sie diese Auszüge so knapp wie möglich und wägen Sie Lesbarkeit und Redundanzfreiheit gegeneinander ab.

## Qualitätsziele

(engl.: Quality Goals)

Inhalt:

Die Hitparade (Top-3 bis Top-5) der Qualitätsziele für die Architektur und/oder Randbedingungen, deren Erfüllung oder Einhaltung den maßgeblichen Stakeholdern besonders wichtig sind.

Gemeint sind hier wirklich Qualitätsziele, die nicht unbedingt mit den Zielen des Projekts übereinstimmen. Beachten Sie den Unterschied.

Als Qualitätsziele findet man in der Praxis oft:

1. Verfügbarkeit (availability)
2. Änderbarkeit (modifiability)
3. Performanz (performance)
4. Sicherheit (security)
5. Testbarkeit (testability)
6. Bedienbarkeit (usability)

Motivation:

Wenn Sie als Architekt nicht wissen, woran Ihre Arbeit gemessen wird, ....

Form:

Einfache tabellarische Darstellung, geordnet nach Prioritäten

Hintergrund:

Beginnen Sie NIEMALS mit einer Architekturentwicklung, wenn diese Ziele nicht schriftlich festgelegt und von den maßgeblichen Stakeholdern akzeptiert sind.

|  |
| --- |
| Wir haben oft genug Projekte ohne definierte Qualitätsziele durchlitten. Wir leiden nicht gerne, daher sind wir inzwischen ziemlich überzeugt, dass sich diese paar Stunden lohnen. Sollte es in Ihrem Projekt Wochen oder Monate dauern, dann denken Sie besser rechtzeitig über berufliche Veränderungen nach :-)  PH & GS. |

Quellen:

Im DIN/ISO 9126 Standard finden Sie eine umfangreiche Sammlung möglicher Qualitätsziele.

Für alle, die es nicht so genau wissen wollen: ein lesbarer Auszug davon ist im Buch "Agile Software- Entwicklung für Embedded Real-Time Systems mit der UML" (Hruschka, Rupp, Carl- Hanser-Verlag, 2002 auf Seite 9 zu finden.  
PH

## Stakeholder

Inhalt

Eine Liste oder Tabelle der wichtigsten Personen oder Organisationen, die von der Architektur betroffen sind oder zur Gestaltung beitragen können.

Motivation

Sie sollten die Projektbeteiligten und -betroffenen kennen, sonst erleben Sie später im Entwicklungsprozess Überraschungen.

Form

EInfache Tabelle mit Rollennamen, Personennamen, deren Kenntnisse, die für die Architektur relevant sind, deren Verfügbarkeit, etc.

Beispiele

Die folgende Tabelle führt Stakeholder auf, die in Projekten relevant sein könn(t)en. Große Teile davon hat Uwe Friedrichsen zusammengetragen

| Stakeholder | Beschreibung |
| --- | --- |
| Management | Linien-Manager, die an dem Projekt beteiligt sind oder es beeinflussen |
| Projekt-Steuerungskreis | Oberstes Lenkungsgremium des Projektes, ultimative Instanz für Projektentscheidungen |
| Projektmanager | Verantwortet das Projekt-Budget, Scope und Zeitplan |
| Auftraggeber | Oft auch „Sponsor“ genannt |
| Produktmanager | Verantwortlich für das gesamte Produkt, das aus Hardware & Software sowie sonstigen Leistungen bestehen kann. |
| Fachbereich | In der Regel die Personengruppe, die die fachlichen Anforderungen formuliert |
| Unternehmens- oder Enterprisearchitekt | u.a. zuständig für strategische Ausrichtung des Anwendungsportfolios und projekt-übergreifende Richtlinien und Standards |
| Architektur-Abteilung | Gruppe, die Unternehmens-Frameworks und Entwicklungsstandards pflegt |
| Methoden und Verfahren | Verantworten Entwicklungsprozesse und häufig auch die eingesetzte Tool  Hinweis: I.d.R. hat man nicht gleichzeitig Unternehmensarchitekten, eine Architektur-Abteilung und Methoden und Verfahren, sondern max. 2 davon |
| IT-Strategie | Verantwortlich für die strategische Ausrichtung der IT. Siehe Enterprise-Architekt. |
| QA | Zentrale Test-Abteilung. Verantwortlich für die Qualitätssicherung |
| Software-Architekt | Oft auch Projekt-Architekt genannt. Verantwortlich für die (technische) Architektur innerhalb eines Projekts |
| Designer | Zuständig für das Anwendungs-Design. Häufig keine eigene Rolle mehr |
| Entwickler | Software-Entwickler im Projekt. Übernimmt häufig auch Design- und Testaufgaben |
| Tester | Tester im Projekt. Kann aus QA sein, häufig aber unabhängig davon. |
| Konfigurations-& Build-Manager | Zuständig für die Pflege von Repository, Konfigurations-Management und Build. Wird in kleineren Projekten häufig vom Entwickler übernommen. |
| Release-Manager | Verantwortlich für die Erstellung und Auslieferung von Release-Ständen. Koordiniert Releases häufig Projekt- und System-übergreifend |
| Wartungs-Team | Zuständig für die Pflege und Wartung des Systems nach Auflösung des Projekt-Teams |
| Externe Dienstleister | Zusätzliche externe Firmen, die Teile der Anwendung entwickeln. |
| Hardware-Designer | Zuständig für das Hardware-Design (im Embedded-Bereich) |
| Rollout-Manager | Zuständig für die Inbetriebnahme eines Systems oder eines Releases. Rolle wird manchmal vom Release-Manager übernommen |
| Infrastruktur-Planung | Zuständig für Planung und Beschaffung der Infrastruktur (Server, Netzwerk, Router, Switches, Arbeitsplatzrechner, OS, …) |
| Sicherheits-beauftragter | Verantwortlich für die IT-Sicherheit im Unternehmen |
| Anwender | Nutzer der Anwendung |
| Fach-Administrator | Zuständig für die fachliche Administration der Anwendung. Hat häufig keinen Zugang zu technischen Administrations-Zugängen |
| System-Administrator | Administriert die Anwendung auf technischer Ebene. Hat Zugang zu technischen Administrations-Zugängen |
| Operator | Überwacht den Anwendungsbetrieb, führt Routine-Pflegejobs durch (z.B. Datensicherung, Aufräumen von temporären Verzeichnissen), behebt einfache Fehler im Anwendungsbetrieb |
| Hotline | Häufig auch unter 1st oder 2nd Level Support bekannt. Nehmen Fehlermeldungen auf, helfen in Standardsituationen |
| Betriebsrat | Vertritt die Interessen der Arbeitnehmer |
| Standard-Software-Lieferant | Lieferant von im System eingesetzter Standard-Software. Unterstützen häufig auch bei Integration und Customizing |
| Verbundene Projekte | z.B. Nachbarprojekte mit gemeinsamen Schnittstellen, übergreifende Schnittstellenprojekte (z.B. EAI/ESB-Projekte) |
| Aufsichtsbehörden, Gesetzgeber, Normierungsgremien | Sind meistens nicht direkt mit dem Projekt verbunden, beeinflussen jedoch durch Ihre Vorgaben die Arbeit bzw. die Lösungsansätze. |
| Weitere externe Stakeholder | z.B. Verbände, Vereine, Mitbewerber, konkurrierende Geschäftsbereiche, Presse. Sind häufig nicht direkt vom Projekt betroffen, beeinflussen Entscheidungen aber dennoch |

Die folgende Tabelle zeigt Ihre konkreten Stakeholder für das System sowie deren Interessen oder Beteiligung.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rolle | Beschreibung | Ziel / Intention | Kontakt | Bemerkungen |
|  |  |  |  |  |

# Randbedingungen

(engl.: Architecture Constraints)

Inhalt

Fesseln, die Software-Architekten in ihren Freiheiten bezüglich des Entwurfs oder des Entwicklungsprozesses einschränken.

Motivation

Architekten sollten klar wissen, wo Ihre Freiheitsgrade bezüglich Entwurfsentscheidungen liegen und wo sie Randbedingungen beachten müssen.

Randbedingungen können vielleicht noch verhandelt werden, zunächst sind sie aber da.

Form

Informelle Listen, gegliedert nach den Unterpunkten dieses Kapitels.

Beispiele

siehe Unterkapitel

Hintergründe

Im Idealfall sind Randbedingungen durch die Anforderungen vorgegeben, spätestens die Architekten müssen sich dieser Randbedingungen bewusst sein.

Den Einfluss von Randbedingungen auf Software- und Systemarchitekturen beschreibt [Hofmeister+1999] (Softwware-Architecture, A Practical Guide, Addison-Wesley 1999) unter dem Stichwort „Global Analysis“.

## Technische Randbedingungen

Inhalt

Tragen Sie hier alle technischen Randbedingungen ein. Zu dieser Kategorie gehören Hard- und Software-Infrastruktur, eingesetzte Technologien (Betriebssysteme, Middleware, Datenbanken, Programmiersprachen, ...).

|  |  |
| --- | --- |
| Hardware-Vorgaben | |
|  | <hier Randbedingungen einfügen> |
|  | <hier Randbedingungen einfügen> |
| Software-Vorgaben | |
|  | <hier Radbedingungen einfügen> |
| Vorgaben des Systembetriebs | |
|  | <hier Randbedingungen einfügen> |
| Programmiervorgaben | |
|  | <hier Randbedingungen einfügen> |

Beispiele

|  |  |
| --- | --- |
| Randbedingung | Erläuterung |
| Hardware-Infrastruktur | Prozessoren, Speicher, Netzwerke, Firewalls und andere relevante Elemente der Hardware- Infrastruktur |
| Software-Infrastruktur | Betriebssysteme, Datenbanksysteme, Middleware, Kommunikationssysteme, Transaktionsmonitor, Webserver, Verzeichnisdienste |
| Systembetrieb | Batch- oder Onlinebetrieb des Systems oder notwendiger externer Systeme? |
| Verfügbarkeit der Laufzeitumgebung | Rechenzentrum mit 7x24h Betriebszeit?  Gibt es Wartungs- oder Backupzeiten mit eingeschränkter Verfügbarkeit des Systems oder wichtiger Systemteile? |
| Grafische Oberfläche | Existieren Vorgaben hinsichtlich grafischer Oberfläche (Style Guide)? |
| Bibliotheken, Frameworks und Komponenten | Sollen bestimmte „Software-Fertigteile“ eingesetzt werden? |
| Programmiersprachen | Objektorientierte, strukturierte, deklarative oder Regelsprachen, kompilierte oder interpretierte Sprachen? |
| Referenzarchitekturen | Gibt es in der Organisation vergleichbare oder übertragbare Referenzprojekte? |
| Analyse- und Entwurfsmethoden | Objektorientierte oder strukturierte Methoden? |
| Datenstrukturen | Vorgaben für bestimmte Datenstrukturen, Schnittstellen zu bestehenden Datenbanken oder Dateien |
| Programmierschnittstellen | Schnittstellen zu bestehenden Programmen |
| Programmiervorgaben | Programmierkonventionen, fester Programmaufbau |
| Technische Kommunikation | Synchron oder asynchron, Protokolle |
| Betriebssystem und Middleware | Vorgegebene Betriebssysteme oder Middleware |

## Organisatorische Randbedingungen

Inhalt

Tragen Sie hier alle organisatorischen, strukturellen und ressourcenbezogenen Randbedingungen ein. Zu dieser Kategorie gehören auch Standards, die Sie einhalten müssen und juristische Randbedingungen.

|  |  |
| --- | --- |
| Organisation und Struktur | |
|  | <hier Randbedingungen einfügen> |
| Ressourcen (Budget, Zeit, Personal) | |
|  | <hier Randbedingungen einfügen> |
| Organisatorische Standards | |
|  | <hier Randbedingungen einfügen> |
| Juristische Faktoren | |
|  | <hier Randbedingungen einfügen> |

Beispiele

|  |  |
| --- | --- |
| Randbedingung | Erläuterung |
| Organisation und Struktur | |
| Organisationsstruktur beim Auftraggeber | Droht Änderung von Verantwortlichkeiten?  Änderung von Ansprechpartnern |
| Organisationsstruktur des Projektteams | mit/ohne Unterauftragnehmer  Entscheidungsbefugnis der Projektleiterin |
| Entscheidungsträger | Erfahrung mit ähnlichen Projekten  Risiko-/Innovationsfreude |
| Bestehende Partnerschaften oder Kooperationen | Hat die Organisation bestehende Kooperationen mit bestimmten Softwareherstellern?  Solche Partnerschaften geben oftmals Produktentscheidungen (unabhängig von Systemanforderungen) vor. |
| Eigenentwicklung oder externe Vergabe | Selbst entwickeln oder an externe Dienstleister vergeben? |
| Entwicklung als Produkt oder zur eigenen Nutzung? | Bedingt andere Prozesse bei Anforderungsanalyse und Entscheidungen. Im Fall der Produktentwicklung:  Neues Produkt für neuen Markt?  Verbessertes Produkt für bestehenden Markt?  Vermarktung eines bestehenden (eigenen) Systems     Entwicklung ausschließlich zur eigenen Nutzung? |
| Ressourcen (Budget, Zeit, Personal) | |
| Festpreis oder Zeit/Aufwand? | Festpreisprojekt oder Abrechnung nach Zeit und Aufwand? |
| Zeitplan | Wie flexibel ist der Zeitplan? Gibt es einen festen Endtermin? Welche Stakeholder bestimmen den Endtermin? |
| Zeitplan und Funktionsumfang | Was ist höher priorisiert, der Termin oder der Funktionsumfang? |
| Release-Plan | Zu welchen Zeitpunkten soll welcher Funktionsumfang in Releases/Versionen zur Verfügung stehen? |
| Projektbudget | Fest oder variabel? In welcher Höhe verfügbar? |
| Budget für technische Ressourcen | Kauf oder Miete von Entwicklungswerkzeugen (Hardware und Software)? |
| Team | Anzahl der Mitarbeiter und deren Qualifikation, Motivation und kontinuierliche Verfügbarkeit. |
| Organisatorische Standards | |
| Vorgehensmodell | Vorgaben bezüglich Vorgehensmodell? Hierzu gehören auch interne Standards zu Modellierung, Dokumentation und Implementierung. |
| Qualitätsstandards | Fällt die Organisation oder das System in den Geltungsbereich von Qualitätsnormen (wie ISO-9000)? |
| Entwicklungs­werkzeuge | Vorgaben bezüglich der Entwicklungswerkzeuge (etwa: CASE-Tool, Datenbank, Integrierte Entwicklungsumgebung, Kommunikationssoftware, Middleware, Transaktionsmonitor). |
| Konfigurations- und Versionsverwaltung | Vorgaben bezüglich Prozessen und Werkzeugen |
| Testwerkzeuge und prozesse | Vorgaben bezüglich Prozessen und Werkzeugen |
| Abnahme- und Freigabeprozesse | Datenmodellierung und Datenbankdesign  Benutzeroberflächen  Geschäftsprozesse (Workflow)  Nutzung externer Systeme (etwa: schreibender Zugriff bei externen Datenbanken) |
| Service Level Agreements | Gibt es Vorgaben oder Standards hinsichtlich Verfügbarkeiten oder einzuhaltender Service-Levels? |
| Juristische Faktoren | |
| Haftungsfragen | Hat die Nutzung oder der Betrieb des Systems mögliche rechtliche Konsequenzen?  Kann das System Auswirkung auf Menschenleben oder Gesundheit besitzen?  Kann das System Auswirkungen auf Funktionsfähigkeit externer Systeme oder Unternehmen besitzen? |
| Datenschutz | Speichert oder bearbeitet das System „schutzwürdige“ Daten? |
| Nachweispflichten | Bestehen für bestimmte Systemaspekte juristische Nachweispflichten? |
| Internationale Rechtsfragen | Wird das System international eingesetzt?  Gelten in anderen Ländern eventuell andere juristische Rahmenbedingungen für den Einsatz (Beispiel: Nutzung von Verschlüsselungsverfahren)? |

## Konventionen

Inhalt

Fassen Sie unter dieser Überschrift alle Konventionen zusammen, die für die Entwicklung der Software-Architektur relevant sind.

Form

Entweder die Konventionen als Kapitel hier direkt einhängen oder aber auf entsprechende Dokumente verweisen.

Beispiele

1. Programmierrichtlinien
2. Dokumentationsrichtlinien
3. Richtlinien für Versions- und Konfigurationsmanagement
4. Namenskonventionen

# Kontextabgrenzung

Inhalt

Die Kontextsicht grenzt das System, für das Sie die Architektur entwickeln, von allen Nachbarsystemen ab. Sie legt damit die wesentlichen externen Schnittstellen fest.

Stellen Sie sicher, dass die Schnittstellen mit allen relevanten Aspekten (was wird übertragen, in welchem Format wird übertragen, welches Medium wird verwendet, ...) spezifiziert wird, auch wenn einige populäre Diagramme (wie z.B. das UML Use-Case Diagramm) nur ausgewählte Aspekte der Schnittstelle darstellen.

Motivation

Die Schnittstellen zu Nachbarsystemen gehören zu den kritischsten Aspekten eines Projektes. Stellen Sie rechtzeitig sicher, dass Sie diese komplett verstanden haben.

Form

1. Diverse Kontextdiagramme (siehe folgende Abschnitte)
2. Listen von Nachbarsystemen mit deren Schnittstellen.

Die folgenden Unterkapitel zeigen die Einbettung unseres Systems in seine Umgebung.

## Fachlicher Kontext

Inhalt

Festlegung aller[[1]](#footnote-1) Nachbarsysteme des betrachteten Systems mit Spezifikation aller fachlichen Daten, die mit diesen ausgetauscht werden. Zusätzlich evtl. Datenformate und Protokolle der Kommunikation mit Nachbarsystemen und der Umwelt (falls diese nicht erst bei den spezifischen Bausteinen präzisiert wird.

Motivation

Verstehen, welche (logischen) Informationen mit Nachbarsystemen (in welcher Form) ausgetauscht werden.

Form

Logisches Kontextdiagramm,

in der UML z.B. simuliert durch Klassendiagramme, Use Case Diagramme, Kommunikations–diagramme - kurz durch alle Diagramme, die das System als Black Box darstellen und die Schnittstellen zu den Nachbarsystemen (mehr oder weniger ausführlich) beschreiben.

Alternativ oder ergänzend können Sie einfach eine Tabelle verwenden. Der Titel gibt den Namen Ihres Systems wieder; die drei Spalten sind: Nachbarsystem, Input, Output. Auch so kommen Sie zu einer kompletten Schnittstellenbeschreibung.

## Technischer- oder Verteilungskontext

Inhalt

Festlegung der Kanäle zwischen Ihrem System, den Nachbarsystemen und der Umwelt;

Zusätzlich eine Mappingtabelle, welcher logische Input (aus 3.1) über welchen Kanal ein- oder ausgegeben wird.

Motivation

Verstehen, über welche Medien Informationen mit Nachbarsystemen bzw. der Umwelt ausgetauscht werden.

Form

z.B.: UML Deploymentdiagramm mit den Kanälen zu Nachbarsystemen, begleitet von einer Mappingtabelle Kanal x Input/Output.

## Externe Schnittstellen

Contents

Specification of the communication channels linking your system to neighboring systems and the environment.

### Externe Schnittstelle 1

#### Identifikation der Schnittstelle

|  |  |
| --- | --- |
| Name / Bezeichnung der Schnittstelle | <Name der Schnittstelle> |
| Version |  |
| Änderungen gegenüber Vorversion |  |
| Wer hat geändert und warum? |  |
| Veranwortlicher Ansprechpartner / Rolle |  |

#### Fachlicher Kontext der Schnittstelle

#### Fachliche Abläufe

<Diagramm oder Beschreibung der fachlichen Abläufe>

#### Fachliche Bedeutung der Daten

<Beschreibung der fachlichen Bedeutung>

  Technischer Kontext

  Form der Interaktion

#### Anforderungen an die Schnittstelle

#### Sicherheitsanforderungen

#### Mengengerüste

   Laufzeit

   Durchsatz / Datenvolumen

  Verfügbarkeit

  Protokollierung

  Archivierung

#### Beteiligte Resourcen

#### Syntax: Daten und Formate

   Datenformate

   Gültigkeits- und Plausibilitätsregeln

   Codierung, Zeichensätze

   Konfigurationsdaten

#### Syntax: Methoden/Funktionen

  Prüfdaten

#### **Ablauf der Schnittstelle**

fachliche oder technischer Ablauf

#### Semantik

 Nebenwirkungen, Konsequenzen

#### Technische Infrastruktur

 Technische Protokolle

#### Fehler- und Ausnahmebehandlung

#### Einschränkungen und Voraussetzungen

 Berechtigungen

  Zeitliche Einschränkungen

  Parallele Benutzung

  Voraussetzungen zur Nutzung

#### Betrieb der Schnittstelle

#### Metainformationen der Schnittstelle

Verantwortliche

Kosten der Nutzung

Organisatorisches

Versionierung

#### Beispiele für Nutzung und Daten

Beispieldaten

Beispielabläufe / -interaktionen

Programmierbeispiele

# Lösungsstrategie

Inhalt

Kurzer Überblick über Ihre grundlegenden Entscheidungen und Lösungsansätze, die jeder, der mit der Architektur zu tun hat, verstanden haben sollte.

Motivation

Dieses Kapitel motiviert übergreifend die zentralen Gestaltungskriterien für Ihre Architektur. Beschränken Sie sich hier auf das Wesentliche. Detailentscheidungen können immer noch bei den einzelnen Bausteinen oder im Kapitel 10 festgehalten werden. Das Kapitel soll Ihren Lesern die gewählte Strategie verdeutlichen.

Form

Fassen Sie auf wenigen Seiten die Beweggründe für zentrale Entwurfsentscheidungen zusammen. Motivieren Sie ausgehend von Aufgabenstellung, Qualitätszielen und Randbedingungen, was Sie entschieden haben und warum Sie so entschieden haben. Verweisen Sie – wo nötig - auf weitere Ausführungen in Folgekapiteln.

# Bausteinsicht

Inhalt

Statische Zerlegung des Systems in Bausteine (Module, Komponenten, Subsysteme, Teilsysteme, Klassen, Interfaces, Pakete, Bibliotheken, Frameworks, Schichten, Partitionen, Tiers, Funktionen, Makros, Operationen, Datenstrukturen...) sowie deren Beziehungen.

Motivation

Dies ist die wichtigste Sicht, die in jeder Architekturdokumentation vorhanden sein muss. Wenn Sie es mit dem Hausbau vergleichen ist das der Grundrissplan.

Form

Die Bausteinsicht ist eine hierarchische Sammlung von BlackBox- und White-Box- Beschreibungen (siehe Abbildung unten):



Ebene 1 ist die White-Box-Beschreibung des Gesamtsystems (System under Development / SUD) mit den Black- Box- Beschreibungen der Bausteine des Gesamtsystems

Ebene 2 zoomt dann in die Bausteine der Ebene 1 hinein und ist somit die Sammlung aller White-Box- Beschreibungen der Bausteine der Ebene 1 zusammen mit den Black-Box-Beschreibungen der Bausteine der Ebene 2.

Ebene 3 zoomt in die alle Bausteine der Ebene 2 hinein, u.s.w.

============================

Whitebox-Template:

Enthält mehrere Bausteine, zu denen Sie jeweils eine Black-Box Beschreibung erstellen.

Blackbox-Template

Für jeden Baustein aus dem White-Box-Template sollten Sie folgende Angaben machen: (Kopieren Sie diese Punkte in die folgenden Unterkapitel)

1. Zweck / Verantwortlichkeit:
2. Schnittstelle(n):
3. Erfüllte Anforderungen:
4. Ablageort / Datei:
5. Sonstige Verwaltungsinformation: Autor, Version, Datum, Änderungshistorie

## Ebene 1

An dieser Stelle beschreiben Sie die White-Box-Sicht der Ebene 1 gemäß dem Whitebox-Template. Die Struktur ist im folgenden bereits vorgegeben.

Das Überblicksbild zeigt das Innenleben des Gesamtsystems in Form der Bausteine 1 - n, sowie deren Zusammenhänge und Abhängigkeiten.

Sinnvoll sind hier auch Beschreibungen der wichtige Begründungen, die zu dieser Struktur führen, insbesondere die Beschreibung der Abhängigkeiten (Beziehungen) zwischen den Bausteinen dieser Ebene.

Evtl. verweisen Sie auch auf verworfene Alternativen (mit der Begründung, warum es verworfen wurde

Die folgende Abbildung zeigt die Hauptbausteine unseres Systems und deren Abhängigkeiten.

<hier Überblicksdiagramm einfügen>

Erläuterung zu Struktur und Abhängigkeiten der Ebene 1:

### Bausteinname 1 (BlackBox-Beschreibung)

1. Struktur gemäß Black-Box- Template:
2. Zweck / Verantwortlichkeit:
3. Schnittstelle(n):
4. Erfüllte Anforderungen:
5. Variabilität:
6. Leistungsmerkmale:
7. Ablageort / Datei:
8. Sonstige Verwaltungsinformation:
9. Offene Punkte:

<Black Box Template des Bausteins hier einfügen>

### Bausteinname 2 (Black Box-Beschreibung)

<Black Box Template des Bausteins hier einfügen>

### ...

<Black Box Template des Bausteins hier einfügen>

### Bausteinname n (Black Box-Beschreibung)

<Black Box Template des Bausteins hier einfügen>

### Offene Punkte

## Ebene 2

An dieser Stelle beschreiben Sie die White-Box-Sichten aller Bausteine der Ebene 1 als Folge von White-Box-Templates. Die Struktur ist im folgenden bereits vorgegeben. Die Struktur ist im folgenden für 3 Bausteine vorgegeben. Bitte kopieren Sie diese Struktur so oft, wie benötigt.

### Bausteinname 1 (Whitebox-Beschreibung)

zeigt das Innenleben des Bausteines in Diagrammform mit den lokalen Bausteinen 1 - n, sowie deren Zusammenhänge und Abhängigkeiten.

beschreibt wichtige Begründungen, die zu dieser Struktur führen

verweist evtl. auf verworfene Alternativen (mit der Begründung, warum es verworfen wurde

< Hier Überblicksdiagramm für Innenleben von Baustein 1 einfügen>

#### Bausteinname 1.1 (BlackBox Beschreibung)

Struktur gemäß Black-Box- Template:

1. Zweck / Verantwortlichkeit:
2. Schnittstelle(n):
3. Erfüllte Anforderungen:
4. Variabilität:
5. Leistungsmerkmale:
6. Ablageort / Datei:
7. Sonstige Verwaltungsinformation:
8. Offene Punkte:

#### Bausteinname 1.2 (Black Box Beschreibung)

Struktur gemäß Black-Box- Template:

1. Zweck / Verantwortlichkeit:
2. Schnittstelle(n):
3. Erfüllte Anforderungen:
4. Variabilität:
5. Leistungsmerkmale:
6. Ablageort / Datei:
7. Sonstige Verwaltungsinformation:
8. Offene Punkte:

#### ...

#### Bausteinname 1.n (Black Box Beschreibung)

Struktur gemäß Black-Box- Template:

1. Zweck / Verantwortlichkeit:
2. Schnittstelle(n):
3. Erfüllte Anforderungen:
4. Variabilität:
5. Leistungsmerkmale:
6. Ablageort / Datei:
7. Sonstige Verwaltungsinformation:
8. Offene Punkte:

#### Beschreibung der Beziehungen

#### Offene Punke

### Bausteinname 2 (Whitebox-Beschreibung)

#### Überblicksdiagramm 2 Ebene 2

1. zeigt das Innenleben des Bausteines in Diagrammform mit den lokalen Bausteinen 1 - n, sowie deren Zusammenhänge und Abhängigkeiten.
2. beschreibt wichtige Begründungen, die zu dieser Struktur führen
3. verweist evtl. auf verworfene Alternativen (mit der Begründung, warum es verworfen wurde

#### Bausteinname 2.1 (BlackBox Beschreibung)

Struktur gemäß Black-Box- Template:

1. Zweck / Verantwortlichkeit:
2. Schnittstelle(n):
3. Erfüllte Anforderungen:
4. Variabilität:
5. Leistungsmerkmale:
6. Ablageort / Datei:
7. Sonstige Verwaltungsinformation:
8. Offene Punkte:

#### Bausteinname 2.2 (Black Box Beschreibung)

Struktur gemäß Black-Box- Template:

1. Zweck / Verantwortlichkeit:
2. Schnittstelle(n):
3. Erfüllte Anforderungen:
4. Variabilität:
5. Leistungsmerkmale:
6. Ablageort / Datei:
7. Sonstige Verwaltungsinformation:
8. Offene Punkte:

#### ...

#### Bausteinname 2.n (Black Box Beschreibung)

Struktur gemäß Black-Box- Template:

1. Zweck / Verantwortlichkeit:
2. Schnittstelle(n):
3. Erfüllte Anforderungen:
4. Variabilität:
5. Leistungsmerkmale:
6. Ablageort / Datei:
7. Sonstige Verwaltungsinformation:
8. Offene Punkte:

#### Beschreibung der Beziehungen

#### Offene Punkte

### Bausteinename 3 (Whitebox-Beschreibung)

#### Überblicksdiagramm 3 Ebene 2

1. zeigt das Innenleben des Bausteines in Diagrammform mit den lokalen Bausteinen 1 - n, sowie deren Zusammenhänge und Abhängigkeiten.
2. beschreibt wichtige Begründungen, die zu dieser Struktur führen
3. verweist evtl. auf verworfene Alternativen (mit der Begründung, warum es verworfen wurde

#### Bausteinname 3.1 (BlackBox Beschreibung)

Struktur gemäß Black-Box- Template:

1. Zweck / Verantwortlichkeit:
2. Schnittstelle(n):
3. Erfüllte Anforderungen:
4. Variabilität:
5. Leistungsmerkmale:
6. Ablageort / Datei:
7. Sonstige Verwaltungsinformation:
8. Offene Punkte:

#### Bausteinname 3.2 (Black Box Beschreibung)

Struktur gemäß Black-Box- Template:

1. Zweck / Verantwortlichkeit:
2. Schnittstelle(n):
3. Erfüllte Anforderungen:
4. Variabilität:
5. Leistungsmerkmale:
6. Ablageort / Datei:
7. Sonstige Verwaltungsinformation:
8. Offene Punkte:

#### ...

#### Bausteinname 3.n (Black Box Beschreibung)

Struktur gemäß Black-Box- Template:

1. Zweck / Verantwortlichkeit:
2. Schnittstelle(n):
3. Erfüllte Anforderungen:
4. Variabilität:
5. Leistungsmerkmale:
6. Ablageort / Datei:
7. Sonstige Verwaltungsinformation:
8. Offene Punkte:

#### Beschreibung der Beziehungen

#### Offene Punkte

## Ebene 3

An dieser Stelle beschreiben Sie die White-Box- Sichten aller Bausteine der Ebene 2 als Folge von White-Box-Templates. Die Struktur ist identisch mit der Struktur auf Ebene 2. Kopieren Sie die entsprechenden Gliederungspunkte hierhier.

Bei tieferen Gliederungen der Architektur kopieren Sie bitte das ganze Kapitel für die nächsten Ebenen.

# Laufzeitsicht

Inhalt

Diese Sicht beschreibt, wie sich die Bausteine des Systems als Laufzeitelemente (Prozesse, Tasks, Activities, Threads, ...)  verhalten und wie sie zusammenarbeiten.

Als alternative Bezeichnungen finden Sie dafür auch:

1. Dynamische Sichten
2. Prozesssichten
3. Ablaufsichten

Suchen Sie sich interessante Laufzeitszenarien heraus, z.B.:

1. Wie werden die wichtigsten Use-Cases durch die Architekturbausteine bearbeitet?
2. Welche Instanzen von Architekturbausteinen gibt es zur Laufzeit und wie werden diese gestartet, überwacht und beendet?
3. Wie arbeiten Systemkomponenten mit externen und vorhandenen Komponenten zusammen?
4. Wie startet das System (etwa: notwendige Startskripte, Abhängigkeiten von externen Subsystemen, Datenbanken, Kommunikationssystemen etc.)?

Anmerkung: Kriterium für die Auswahl der möglichen Szenarien (d.h. Abläufe) des Systems ist deren *Architekturrelevanz.* Es geht nicht darum, möglichst viele Abläufe darzustellen, sondern eine angemessene Auswahl zu dokumentieren.

Kandidaten sind:

1. Die wichtigsten 3-5 Anwendungsfälle
2. Systemstart
3. Das Verhalten an den wichtigsten externen Schnittstellen
4. Das Verhalten in den wichtigsten Fehlerfällen

Motivation

Sie müssen (insbesondere bei objektorientierten Architekturen) nicht nur die Bausteine mit ihren Schnittstellen spezifizieren, sondern auch, wie Instanzen von Bausteinen zur Laufzeit miteinander kommunizieren.

Form

Dokumentieren Sie die ausgesuchten Laufzeitszenarien mit UML-Sequenz-, Aktivitäts-, oder Kommunikationsdiagrammen.

Mit Objektdiagrammen können Sie Schnappschüsse der existierenden Laufzeitobjekte darstellen und auch instanziierte Beziehungen. Die UML bietet dabei die Möglichkeit zwischen aktiven und passiven Objekten zu unterscheiden.

## Laufzeitszenario 1

1. Laufzeitdiagramm
2. Erläuterung der Besonderheiten bei dem Zusammenspiel der Bausteininstanzen in diesem Diagramm

## Laufzeitszenario 2

1. Laufzeitdiagramm
2. Erläuterung der Besonderheiten bei dem Zusammenspiel der Bausteininstanzen in diesem Diagramm

## ...

## Laufzeitszenario n

# Verteilungssicht

Inhalt

Diese Sicht beschreibt, in welcher Umgebung das System abläuft. Sie beschreiben die geographische Verteilung Ihres Systems oder die Struktur der Hardwarekomponenten, auf denen die Software abläuft. Sie dokumentiert Rechner, Prozessoren, Netztopologien und Kanäle, sowie sonstige Bestandteile der physischen Systemumgebung. Die Verteilungssicht zeigt das System aus Betreibersicht.

Zeigen Sie in dieser Sicht auch, wie die Bausteine des Systems zu *Verteilungsartefakten* zusammengefasst oder –gebaut werden (engl. deployment artifacts oder deployment units).

Motivation

Software ohne Hardware tut wenig. Das Minimum, was Sie als Software-Architekt daher brauchen, sind so viele Angaben zu der zugrunde liegenden (Hardware- )Verteilung, dass Sie jeden Software-Baustein, der für den Betrieb interessant ist, irgendwelchen Hardware-Einheiten zuordnen können. (Das gilt auch für Standardsoftware, die Voraussetzung für das Funktionieren des Gesamtsystems ist). Sie sollen mit diesen Modellen die Betreiber in die Lage versetzen, die Software auch komplett und richtig zu installieren.

Form

Die UML stellt mit Verteilungsdiagrammen (Deployment diagrams) eine Diagrammart zur Verfügung, um diese Sicht auszudrücken. Nutzen Sie diese, evtl. auch geschachtelt, wenn Ihre Verteilungsstruktur es verlangt. (Das oberste Deployment- Diagramm sollte bereits in Ihrer Kontextsicht enthalten sein mit Ihrer Infrastruktur als EINE Black-Box. Jetzt zoomen Sie in diese Infrastruktur mit weiteren Deployment- Diagrammen hinein.

Andere Diagramme Ihrer Hardware-Kollegen, die Prozessoren und Kanäle darstellen sind hier ebenfalls einsetzbar. Abstrahieren Sie aber auf die Aspekte, die für die Software-Verteilung relevant sind.

## Infrastruktur Ebene 1

### Verteilungsdiagramm Ebene 1

1. zeigt das Verteilung des Gesamtsystems auf 1 - n Prozessoren (oder Standorte) sowie die physischen Verbindungskanäle zwischen diesen.
2. beschreibt wichtige Begründungen, die zu dieser Verteilungsstruktur, d.h. zur Auswahl der Knoten und zhur Auswahl der Kanäle führten
3. verweist evtl. auf verworfene Alternativen (mit der Begründung, warum es verworfen wurden

### Prozessor 1

Struktur gemäß Knoten-Template (node-template):

1. Beschreibung
2. Leistungsmerkmale
3. Zugeordnete Software- Bausteine
4. Sonstige Verwaltungsinformationen
5. Offene Punkte

### Prozessor 2

Struktur gemäß Knoten-Template:

1. Beschreibung
2. Leistungsmerkmale
3. Zugeordnete Software- Bausteine
4. Sonstige Verwaltungsinformationen
5. Offene Punkte

### ...

### Prozessor n

Struktur gemäß Knoten-Template:

1. Beschreibung
2. Leistungsmerkmale
3. Zugeordnete Software- Bausteine
4. Sonstige Verwaltungsinformationen
5. Offene Punkte

### Kanal 1

Inhalt

Spezifikation der Eigenschaften des Kanals, soweit für die Software- Architektur interessant ist.

Motivation

Spezifizieren Sie mindest die Eigenschaften der Übertragungskanäle, die Sie brauchen, um nicht-funktionale Anforderungen nachzuweisen, wie maximaler Durchsatz, Störungswahrscheinlichkeiten oder ähnliche.

Form

Verwenden Sie ein ähnliches Muster wie für die Prozessorspezifikationen.

Oftmals verweisen Sie auf einen Standard (z.B: CAN-Bus, 10Mbit Ethernet, Druckerkabel, ...).

### Kanal 2

### ...

### Kanal m

### Offene Punkte

## Infrastruktur Ebene 2

Inhalt

Weitere Deploymentdiagramme mit gleicher Beschreibungsstruktur wie oben.

Motivation

Zur Verfeinerung der Infrastruktur soweit, wie Sie es für die Verteilung der Software benötigen.

# Konzepte

Inhalt:

Die folgenden Kapitel sind Beispiele für übergreifende Aspekte.

Falls einige der Aspekte für Ihr Projekt nicht wichtig sind oder nicht zutreffen, so halten Sie *diese* Information ebenfalls fest, anstatt das Kapitel zu löschen.

Motivation:

Manche der Aspekte lassen sich nur schwer "zentral" als Baustein in der Architektur unterbringen (z.B. das Thema "Sicherheit". Hier ist der Platz im Template, wo Sie Konzepte zu derartigen Themen geschlossen behandeln können.

Alle Aspekte, die in der Architektur an vielen Stellen Konsequenzen zeigen, beispielsweise ein Domänen-/Fachklassen- oder Business-Modell, haben ebenfalls hier einen guten Platz.

Schließlich kommen manche Strukturen in der Architektur wiederholt vor, beispielsweise ein an mehreren Stellen eingesetztes Pattern. Auch solche Aspekte können Sie hier zentral erläutern.

Form:

Kann vielfältig sein. Teilweise Konzeptpapiere mit beliebiger Gliederung, teilweise auch übergreifende Modelle/Szenarien mit Notationen, die Sie auch in den Architektursichten nutzen.

## Fachliche Strukturen und Modelle

Fachliche Modelle, Domänenmodelle, Business-Modelle – sie alle beschreiben Strukturen der reinen Fachlichkeit, also ohne Bezug zur Implementierungs- oder Lösungstechnologie.

Oftmals tauchen Teile solcher fachlichen Modelle an vielen Stellen in der Architektur, insbesondere der Bausteinsicht, wieder auf.

## Typische Muster und Strukturen

### Nicht funktionale Muster

### Bridge

### Broker

### Master-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

### Bridge

Oftmals tauchen einige typische Lösungsstrukturen oder Grundmuster an mehren Stellen der Architektur auf. Beispiele dafür sind die Abhängigkeiten zwischen Persistenzschicht, Applikation sowie die Anbindung grafischer Oberflächen an die Fach- oder Domänenobjekte. Solche wiederkehrenden Strukturen beschreiben Sie möglichst nur ein einziges Mal, um Redundanzen zu vermeiden. Dieser Abschnitt erfüllt genau diesen Zweck.

## Persistenz

Persistenz (Dauerhaftigkeit, Beständigkeit) bedeutet, Daten aus dem (flüchtigen) Hauptspeicher auf ein beständiges Medium (und wieder zurück) zu bringen.

Einige der Daten, die ein Software-System bearbeitet, müssen dauerhaft auf einem Speichermedium gespeichert oder von solchen Medien gelesen werden:

1. Flüchtige Speichermedien (Hauptspeicher oder Cache) sind technisch nicht für dauerhafte Speicherung ausgelegt. Daten gehen verloren, wenn die entsprechende Hardware ausgeschaltet oder heruntergefahren wird.
2. Die Menge der von kommerziellen Software-Systemen bearbeiteten Daten übersteigt üblicherweise die Kapazität des Hauptspeichers.
3. Auf Festplatten, optischen Speichermedien oder Bändern sind oftmals große Mengen von Unternehmensdaten vorhanden, die eine beträchtliche Investition darstellen.

Persistenz ist ein technisch bedingtes Thema und trägt nichts zur eigentlichen Fachlichkeit eines Systems bei. Dennoch müssen Sie sich als Architekt mit dem Thema auseinander setzen, denn ein erheblicher Teil aller Software-Systeme benötigt einen effizienten Zugriff auf persistent gespeicherte Daten. Hierzu gehören praktisch sämtliche kommerziellen und viele technischen Systeme. Eingebettete Systeme (embedded systems ) gehorchen jedoch oft anderen Regeln hinsichtlich ihrer Datenverwaltung.

## Benutzungsoberfläche

IT-Systeme, die von (menschlichen) Benutzern interaktiv genutzt werden, benötigen eine Benutzungsoberfläche. Das können sowohl grafische als auch textuelle Oberflächen sein.

## Ergonomie

Ergonomie von IT-Systemen bedeutet die Verbesserung (Optimierung) deren Benutzbarkeit aufgrund objektiver und subjektiver Faktoren. Im wesentlichen zählen zu ergonomischen Faktoren die Benutzungsoberfläche, die Reaktivität (gefühlte Performance) sowie die Verfügbarkeit und Robustheit eines Systems.

## Ablaufsteuerung

Ablaufsteuerung von IT-Systemen bezieht sich sowohl auf die an der (grafischen) Oberfläche sichtbaren Abläufe als auch auf die Steuerung der Hintergrundaktivitäten. Zur Ablaufsteuerung gehört daher unter anderem die Steuerung der Benutzungsoberfläche als auch die Workflow-Steuerung.

## Transaktionsbehandlung

Transaktionen sind Arbeitsschritte oder Abläufe, die entweder alle gemeinsam oder garnicht durchgeführt werden. Der Begriff stammt aus den Datenbanken - wichtiges Stichwort hier sind ACID-Transaktionen (atomar, consistent, isolated, durable).

## Sessionbehandlung

Eine Session, auch genannt Sitzung, bezeichnet eine stehende Verbindung eines Clients mit einem Server. Den Zustand dieser Sitzung gilt es zu erhalten, was insbesondere bei der Nutzung zustandsloser Protokolle (etwa HTTP) wichtige Bedeutung hat. Sessionbehandlung stellt für Intra-  und Internetsysteme eine kritische Herausforderung dar und beeinflusst häufig die Performance eines Systems.

## Sicherheit

Die Sicherheit von IT-Systemen befasst sich mit Mechanismen zur Gewährleistung von Datensicherheit und Datenschutz sowie Verhinderung von Datenmissbrauch.

Typische Fragestellungen sind:

1. Wie können Daten auf dem Transport (beispielsweise über offene Netze wie das Internet) vor Missbrauch geschützt werden?
2. Wie können Kommunikationspartner sich gegenseitig vertrauen?
3. Wie können sich Kommunikationspartner eindeutig erkennen und vor falschen Kommunikationspartner schützen?
4. Wie können Kommunikationspartner die Herkunft von Daten für sich beanspruchen (oder die Echtheit von Daten bestätigen)?

Das Thema IT-Sicherheit hat häufig Berührung zu juristischen Aspekten, teilweise sogar zu internationalem Recht.

## Kommunikation und Integration mit anderen IT-Systemen

Kommunikation: Übertragung von Daten zwischen System-Komponenten. Bezieht sich auf Kommunikation innerhalb eines Prozesses oder Adressraumes, zwischen unterschiedlichen Prozessen oder auch zwischen unterschiedlichen Rechnersystemen.

Integration: Einbindung bestehender Systeme (in einen neuen Kontext). Auch bekannt als: (Legacy) Wrapper, Gateway, Enterprise Application Integration (EAI).

## Verteilung

Verteilung: Entwurf von Software-Systemen, deren Bestandteile auf unterschiedlichen und eventuell physikalisch getrennten Rechnersystemen ablaufen.

Zur Verteilung gehören Dinge wie der Aufruf entfernter Methoden (remote procedure call, RPC), die Übertragung von Daten oder Dokumenten an verteilte Kommunikationspartner, die Wahl passender Interaktionsstile oder Nachrichtenaustauschmuster (etwa: synchron / asynchron, publish- subsribe, peer-to- peer).

## Plausibilisierung und Validierung

Wo und wie plausibilisieren und validieren Sie (Eingabe-)daten, etwa Benutzereingaben?

## Ausnahme-/Fehlerbehandlung

Wie werden Programmfehler und Ausnahmen systematisch und konsistent behandelt?

Wie kann das System nach einem Fehler wieder in einen konsistenten Zustand gelangen? Geschieht dies automatisch oder ist manueller Eingriff erforderlich?

Dieser Aspekt hat mit Logging, Protokollierung und Tracing zu tun.

Welche Art Ausnahmen und Fehler behandelt ihr System? Welche Art Ausnahmen werden an welche Außenschnittstelle weitergeleitet und welche Ausnahmen behandelt das System komplett intern?

Wie nutzen Sie die Exception-Handling Mechanismen ihrer Programmiersprache? Verwenden Sie checked- oder unchecked-Exceptions?

## Management des Systems & Administrierbarkeit

Größere IT-Systeme laufen häufig in kontrollierten Ablaufumgebungen (Rechenzentren) unter der Kontrolle von Operatoren oder Administratoren ab. Diese Stakeholder benötigen einerseits spezifische Informationen über den Zustand der Programme zur Laufzeit, andererseits auch spezielle Eingriffs- oder Konfigurationsmöglichkeiten.

## Logging, Protokollierung, Tracing

Es gibt zwei Ausprägungen der Protokollierung, das *Logging* und das *Tracing* . Bei beiden werden Funktions- oder Methodenaufrufe in das Programm aufgenommen, die zur Laufzeit über den Status des Programms Auskunft geben.

In der Praxis gibt es zwischen Logging und Tracing allerdings sehr wohl Unterschiede:

1. Logging kann fachliche oder technische Protokollierung sein, oder eine beliebige Kombination von beidem.
2. Fachliche Protokolle werden gewöhnlich anwenderspezifisch aufbereitet und übersetzt. Sie dienen Endbenutzern, Administratoren oder Betreibern von Softwaresystemen und liefern Informationen über die vom Programm abgewickelten Geschäftsprozesse.
3. Technische Protokolle sind Informationen für Betreiber oder Entwickler. Sie dienen der Fehlersuche sowie der Systemoptimierung.
4. Tracing soll Debugging -Information für Entwickler oder Supportmitarbeiter liefern. Es dient primär zur Fehlersuche und -analyse.

## Geschäftsregeln

Wie behandeln Sie Geschäftslogik oder Geschäftsregeln? Implementieren die beteiligten Fachklassen ihre Logik selbst, oder liegt die Logik in der Verantwortung einer zentralen Komponente? Setzen Sie eine Regelmaschine (rule-engine) zur Interpretation von Geschäftsregeln ein (Produktionsregelsysteme, forward- oder backward-chaining)?

## Konfigurierbarkeit

Die Flexibilität von IT-Systemem wird unter anderem durch ihre Konfigurierbarkeit beeinflusst, die Möglichkeit, manche Entscheidungen hinsichtlich der Systemnutzung erst spät zu treffen. Konfigurierbarkeit kann zu folgenden Zeitpunkten erfolgen:

1. Während der Programmierung: Dabei werden beispielsweise Server-, Datei- oder Verzeichnisnamen direkt ("hart") in den Programmcode aufgenommen.
2. Während des Deployments oder der Installation: Hier werden Konfigurationsinformationen für eine bestimmte Installation angegeben, etwa der Installationspfad.
3. Beim Systemstart: Hier werden Informationen vor oder beim Programmstart dynamisch gelesen.
4. Während des Programmablaufs: Konfigurationsinformation wird zur Programmlaufzeit erfragt oder gelesen.

## Parallelisierung und Threading

Programme können in parallelen Prozessen oder Threads ablaufen - was die Notwendigkeit von Synchronisationspunkten mit sich bringt. Die Grundlagen dieses Aspekten legt die Parallelverarbeitung. Für die Architektur und Implementierung nebenläufiger Systeme sind viele technische Detailaspekte zu berücksichtigen (Adressräume, Arten von Synchronisationsmechanismen (Guards, Wächter, Semaphore), Prozesse und Threads, Parallelität im Betriebssystem, Parallelität in virtuellen Maschinen und andere).

## Internationalisierung

Unterstützung für den Einsatz von Systemen in unterschiedlichen Ländern, Anpassung der Systeme an länderspezifische Merkmale. Bei der Internationalisierung (aufgrund der 18 Buchstaben zwischen I und n des englischen Internationalisation auch i18n genannt) geht es neben der Übersetzung von Aus- oder EIngabetexten auch um verwendete Zeichensätze, Orientierung von Schriften am Bildschirm und andere (äußerliche) Aspekte.

## Migration

Für die meisten Systeme gibt es existierende Altsysteme, die durch die neuen Systeme abgelöst werden sollen. Denken Sie als Architekt nicht nur an Ihre neue, schöne Architektur, sondern rechtzeitig auch an alle organisatorischen und technischen Aspekte, die zur Einführung oder Migration der Architektur beachtet werden müssen.

Beispiele

1. Konzept, Vorgehensweise oder Werkzeuge zur Datenübernahme und initialen Befüllung mit Daten
2. Konzept zur Systemeinführung oder zeitweiliger Parallelbetrieb von Alt- und Neusystem

Müssen Sie bestehende Daten migrieren? Wie führen Sie die benötigten syntaktischen oder semantischern Transformationen durch?

## Testbarkeit

Unterstützung für einfache (und möglichst automatische) Tests. Diese Eigenschaft bildet die Grundlage für das wichtige Erfolgsmuster "Continous Integration". In Projekten sollte mindestens täglich der gesamte Stand der Entwicklung gebaut und (automatisch) getestet werden - daher spielt Testbarkeit eine wichtige Rolle. Wichtige Stichworte hierzu sind Unit- Tests und Mock-Objekte.

## Skalierung, Clustering

Wie gestalten Sie Ihr System „wachstumsfähig“, so daß auch bei steigender Last oder steigenden Benutzerzahlen die Antwortzeiten und/oder Durchsatz erhalten bleiben?

## Hochverfügbarkeit

Wie erreichen Sie hohe Verfügbarkeit des Systems? Legen Sie Teile redundant aus? Verteilen Sie das System auf unterschiedliche Rechner oder Rechenzentren? Betreiben Sie Standby-Systeme?

## Codegenerierung

Wie und wo verwenden Sie Codegeneratoren, um Teile Ihres Systems aus Modellen oder domänenspezifischen Sprachen (DSL’s) zu generieren?

## Buildmanagement

Wie wird das gesamte System aus Sourcecode Bausteinen gebaut? Welche Repositories (Versionsverwaltungssysteme) enthalten welchen Sourcecode, wo liegen Konfigurationsdateien, Testdaten und/oder Build-Skripte (make, ant, maven, gradle oder Ähnliche)?

# Entwurfsentscheidungen

Inhalt

Dokumentieren Sie hier alle wesentlichen Entwurfsentscheidungen und deren Gründe!

Motivation

Es ist wünschenswert, alle wichtigen Entwurfsentscheidungen geschlossen nachlesen zu können. Wägen Sie ab, inwiefern Entwurfsentscheidungen hier zentral dokumentiert werden sollen oder wo eine lokale Beschreibung (z.B in der Whitebox-Sicht von Bausteinen) sinnvoller ist. Vermeiden Sie aber redundante Texte. Verweisen Sie evtl. auf Kap. 4 zurück, wo schon zentrale Architekturstrategien motiviert wurden.

Form

informelle Liste, möglichst nach Wichtigkeit und Tragweite der Entscheidungen für den Leser aufgebaut.

Alternativ auch ausführlicher in Form von einzelnen Unterkapiteln je Entscheidung. Die folgende Mindmap (Quelle: Kolumne „Architekturen dokumentieren“ von S. Zörner im Java Magazin 3/2009) soll Sie dabei unterstützen, wichtige Entscheidungen zu treffen und festzuhalten. Die Hauptäste stellen dabei die wesentlichen Schritte dar. Sie können auch als Überschriften innerhalb eines Unterkapitels dienen (siehe Beispiel unten).



Die Fragen sind nicht sklavisch der Reihe nach zu beantworten. Sie sollen Sie lediglich leiten. In der Vorlage löschen Sie diese heraus, und lassen nur die Inhalte/Antworten stehen.

## Entwurfsentscheidung 1

### Fragestellung

Was genau ist das Problem?

Warum ist es für die Architektur relevant?

Welche Auswirkung hat die Entscheidung?

### Rahmenbedingungen

Welche festen Randbedingungen haben Sie einzuhalten?

Welche EInflussfaktoren sind zu beachten?

### Annahmen

Welche Annahmen haben Sie getroffen?

Welche Annahmen können wie vorab überprüft werden?

Mit welchen Risiken müssen Sie rechnen?

### Betrachtete Alternativen

Welche Lösungsoptionen ziehen Sie in die nähere Auswahl?

Wie bewerten Sie jede einzelne?

Welche Optionen schließen Sie bewusst aus?

### Entscheidung

Wer (wenn nicht Sie selbst) hat die Entscheidung getroffen?

Wie ist sie begründet?

Wann wurde entschieden?

## Entwurfsentscheidung n

# Qualitätsszenarien

Dieses Kapitel fasst alles zusammen, was Sie zur systematischen Bewertung Ihrer Architektur gegen vorgegebene Qualitätsziele benötigen.

## Qualitätsbaum

Inhalt

Der Qualitätsbaum ( a la ATAM) mit Qualitätsszenarien an den Blättern.

Motivation

Insbesondere wenn Sie die Qualität Ihrer Architektur mit formalen Methoden wie ATAM überprüfen wollen, bedürfen die in Kapitel 1.2 genannten Qualitätsziele einer weiteren Präzisierung bis auf die Ebene von diskutierbaren und nachprüfbaren Szenarien. Dazu dient dieses Kapitel.

Form

Eine mögliche Darstellung ist eine baumartige Verfeinerung des Begriffes „Qualität“

## Bewertungsszenarien

Inhalt

Szenarien beschreiben, was beim Eintreffen eines Stimulus auf ein System in bestimmten Situationen geschieht. Sie charakterisieren damit das Zusammenspiel von Stakeholdern mit dem System. Szenarien operationalisieren Qualitätsmerkmale und machen sie messbar.

Wesentlich für die meisten Software-Architekten sind zwei Arten von Szenarien:

1. Nutzungsszenarien (auch genannt *Anwendungs- oder Anwendungsfallszenarien*) beschreiben, wie das System zur Laufzeit auf einen bestimmten Auslöser reagieren soll. Hierunter fallen auch Szenarien zur Beschreibung von Effizienz oder Performance. Beispiel: Das System beantwortet eine Benutzeranfrage innerhalb einer Sekunde.
2. Änderungsszenarien beschreiben eine Modifikation des Systems oder seiner unmittelbarer Umgebung. Beispiel: Eine zusätzliche Funktionalität wird implementiert oder die Anforderung an ein Qualitätsmerkmal ändert sich.

Falls Sie sicherheitskritische Systeme entwerfen, ist eine dritte Art von Szenarien für Sie wichtig, die

1. Grenz- oder Stress-Szenarien beschreiben, wie das System auf Extremsituationen reagiert. Beispiele: Wie reagiert das System auf einen vollständigen Stromausfall, einen gravierenden Hardwarefehler oder ähnliches.



**Abbildung: Schematische Darstellung von Szenarien (nach [Bass+03])**

Szenarien bestehen aus folgenden wesentlichen Teilen (hier zitiert aus [Starke05], die ursprüngliche Gliederung stammt aus [Bass+03]):

1. Auslöser (*stimulus*): beschreibt eine spezifische Zusammenarbeit des (auslösenden) Stakeholders mit dem System. Beispiele: Ein Benutzer ruft eine Funktion auf, ein Entwickler programmiert eine Erweiterung, ein Administrator installiert oder konfiguriert das System.
2. Quelle des Auslösers (*source*): beschreibt, woher der Auslöser kommt. Beispiele: intern oder extern, Benutzer, Betreiber, Angreifer, Manager.
3. Umgebung (*environment*): beschreibt den Zustand des Systems zum Zeitpunkt des Auslösers. Befindet sich das System unter Normal- oder Höchstlast? Ist die Datenbank verfügbar oder nicht? Sind Benutzer online oder nicht? Hier sollten Sie alle Bedingungen beschreiben, die für das Verständnis des Szenarios wichtig sind.
4. Systembestandteil (*artifact*): beschreibt, welcher Bestandteil des Systems vom Auslöser betroffen ist. Beispiele: Gesamtsystem, Datenbank, Webserver.
5. Antwort (*response*): beschreibt wie das System durch seine Architektur auf den Auslöser reagiert. Wird die vom Benutzer aufgerufene Funktion ausgeführt? Wie lange benötigt der Entwickler zur Programmierung? Welche Systemteile sind von Installation/Konfiguration betroffen?
6. Antwortmetrik (*response measure*): beschreibt, wie die Antwort gemessen oder bewertet werden kann. Beispiele: Ausfallzeit in Stunden, Korrektheit Ja/Nein, Änderungszeit in Personentagen, Reaktionszeit in Sekunden.

Motivation

Szenarien benötigen Sie zur Prüfung und Bewertung von Architekturen. Sie dienen als "Maßstab" und helfen helfen Ihnen, die "Zielerreichung" der Architektur hinsichtlich der nichtfunktionalen Anforderungen und Qualitätsmerkmale zu messen.

Form

Entweder tabellarisch oder als Freitext. Sie sollten die Bestandteile (Quelle, Umgebung, Systembestandteil, Antwort, Antwortmetrik) explizt kenntlich machen.

Hintergründe

Es gibt inhaltliche Zusammenhänge zwischen Szenarien und Laufzeitsicht. Häufig können Sie die Szenarien der Laufzeitsicht für die Bewertung wieder verwenden oder zugrunde legen. In die Bewertungsszenarien fließen (im Gegensatz zu den Laufzeitszenarien) noch Antwortmetriken ein, die bei der reinen Ablaufbetrachtung der Laufzeitsichten häufig entfallen.

# Risiken

Inhalt

Eine nach Prioritäten geordnete Liste der erkannten technischen Risiken

Motivation

"Risikomanagement ist Projektmanagement für Erwachsene" (Tim Lister, Atlantic Systems Guild.) Unter diesem Motto sollten Sie technische Risiken in der Architektur gezielt ermitteln, bewerten und dem Projektmanagement als Teil der gesamten Risikoanalyse zur Verfügung stellen.

Form

Risikolisten mit Eintrittswahrscheinlichkeit, Schadenshöhe, Maßnahmen zur Risikovermeidung oder Risikominimierung, ...

# Glossar

Inhalt

Die wichtigsten Begriffe der Software-Architektur in alphabetischer Reihenfolge

Motivation

Die Notwendigkeit für ein Glossar sollte nicht erläutert werden müssen. Oder haben Sie es in Ihren Projekten noch nie vermisst?

Form

einfache zweispaltige Tabelle mit <Begriff> und <Definition>

1. Zwar sind wir an vielen Stellen zu Pragmatismus bereit – hier jedoch bestehen wir auf der vollständigen Auflistung aller (a-l-l-e-r) Nachbarsysteme. Zu viele Projekte sind daran gescheitert, dass sie ihre Nachbarn nicht kannten ☹ [↑](#footnote-ref-1)