Architekturdokumentation project name

Version: V1

Status: {ENTWURF/FREIGEGEBEN/...}

Datum: So, 28 Feb 2016

Erstellt durch

Organisation A	
Max Mustermann (MM)	mm@a.com
Dieter Mustermann (DM)	dm@a.com
Organisation B	
Peter Mustermann (PM)	pm@b.com

Erstellt für

Organisation C	
Jemand Anderes (JA)	ja@c.com

Hinweis

(DE) Die vorliegende Architekturdokumentation beruht auf dem erprobten arc 42 Template in der Version 6.0, März 2012. Es steht zur freien Verfügung unter http://www.arc42.de bereit.

(EN) 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

Kurzfassung

(DE) Das vorliegende Dokument repräsentiert die Architekturdokumentation für das Projekt project name.

(EN)

Inhaltsverzeichnis

dal: Wenngleich dieses Dokument erprobt ist, hat sich herausgestellt, dass ein gleichmässiges Bearbeiten recht aufwendig sein kann. Ein iteratives Vorgehen kann bei diesem Problem helfen. Dabei sollte klar dargelegt werden, welche Punkte als erstes bearbeitet wurden, welche nachgeliefert werden und warum.

dal: Unter Umständen können umfangreiche Kapitel auf mehrere Markdowndateien verteilt werden.

dal: Noch ist unklar wie in diesem Markdowndokument mit der Traceability der aufgelisteten Items umgegangen wird. Es hat sich als gut erwiesen jede dargelegte Information (z.B. Anforderung) referenzierbar zu gestalten.

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:

- Was geschieht im Umfeld des Systems?
- 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:

- Geschäftsprozessen,
- funktionalen Anforderungen,
- nichtfunktionalen Anforderungen und andere Randbedingungen (die

wesentlichen müssen bereits als Architekturziele formuliert sein

oder tauchen als Randbedingungen auf) oder

- Mengengerüste.
- 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:

- Verfügbarkeit (availability)
- Änderbarkeit (modifiability)
- Performanz (performance)
- Sicherheit (security)
- Testbarkeit (testability)
- 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 [HruschkaRupp] "Agile Software-Entwicklung für Embedded Real-Time Systems mit der UML" (Hruschka, Rupp, Carl- Hanser-Verlag, 2002 auf Seite 9 zu finden.

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
Externer 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	uständig für Planung und Beschaffung der Infrastruktur (Server, Netzwerk, Router, Switches, Arbeitsplatzrechner, OS, …)
Sicherheitsbeauftragter	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

Stakeholder	Beschreibung
Standard-Software-Lieferant	Vertritt die Interessen der Arbeitnehmer
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	Bemerkung
...	...	...	...	...

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] (Software-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, …).

Hardwarevorgaben	**...**
...	Randbedingung
Softwarevorgaben	**...**
...	Randbedingung
Programmiervorgaben	**...**
...	Randbedingung

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

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.

- Programmierrichtlinien
- Dokumentationsrichtlinien
- Richtlinien für Versions- und Konfigurationsmanagement
- 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.

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

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

Fachlicher Kontext

Inhalt.

Festlegung aller [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.

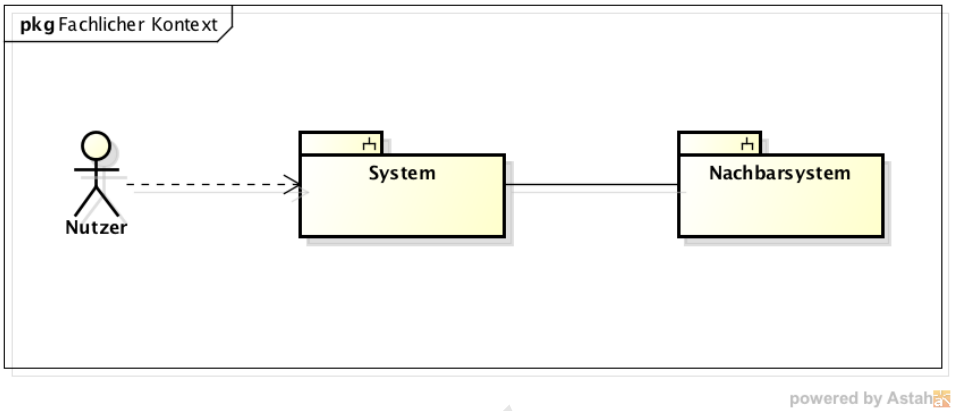
Motivation.

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, Kommunikationsdiagramme - 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.



Quelle

Technischer- oder Verteilungskontext

Inhalt.

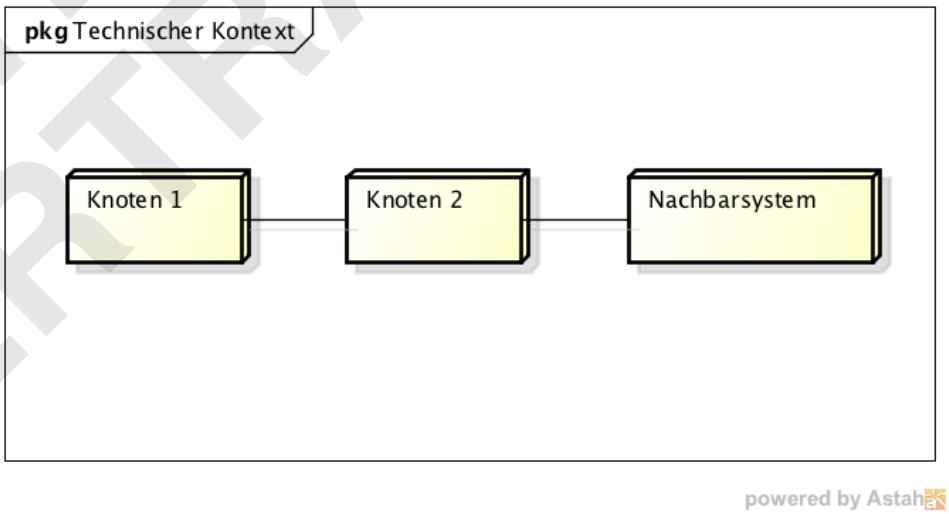
Festlegung der Kanäle zwischen Ihrem System, den Nachbarsystemen und der Umwelt; Zusätzlich eine Mapping-Tabelle, welcher logische Input (aus 3.1) über welchen Kanal ein- oder ausgegeben wird.

Motivation.

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

Form.

z.B.: UML Deployment-Diagramm mit den Kanälen zu Nachbarsystemen, begleitet von einer Mapping-Tabelle Kanal x Input/Output.



Quelle

Externe Schnittstellen

Inhalt.

Spezifikation der Kommunikationskanäle, die ihr System mit den Nachbar-Systemen und der Umwelt verbinden.

Externe Schnittstelle 1

Name / Bezeichnung der Schnittstelle	Name der Schnittstelle
Version	
Änderungen gegenüber Vorversion	
Wer hat geändert und warum?	
Verantwortlicher Ansprechpartner / Rolle	

Fachliche Abläufe.

Diagramm oder Beschreibung der fachlichen Abläufe

- ***Beschreibung der fachlichen Bedeutung***
- Technischer Kontext
- Form der Interaktion
- Laufzeit
- Durchsatz / Datenvolumen
- Verfügbarkeit
- Protokollierung
- Archivierung
- Datenformate
- Gültigkeits- und Plausibilitätsregeln
- Codierung, Zeichensätze
- Konfigurationsdaten
- Prüfdaten
- fachliche oder technischer Ablauf
- Nebenwirkungen, Konsequenzen
- Technische Protokolle
- Welche Fehler werden erkannt?
- Wie werden sie intern behandelt?
- Welche Fehler werden nach aussen gegeben?
- Berechtigungen
- Zeitliche Einschränkungen
- Parallele Benutzung
- Voraussetzungen zur Nutzung
- Verantwortliche
- Kosten der Nutzung
- Organisatorisches
- Versionierung
- 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.

Einstiegshilfe

In diesem Kapitel beschreiben wir Stellen im Code anhand von Klassen oder Methode, an denen wichtige fachliche Funktionalität implementiert ist und/oder die im weiteren Lebenszyklus der Anwendung potenziell geändert werden.

Fall	Konsequenz	Stelle	Hinweise
(Beispiel:) Neue Entität einführen	Ggf. neue UseCases implementieren	de.arc42.sample.ManagerCRUDImpl	ggf Factory Methoden erweitern

Bausteinsicht

Inhalt.

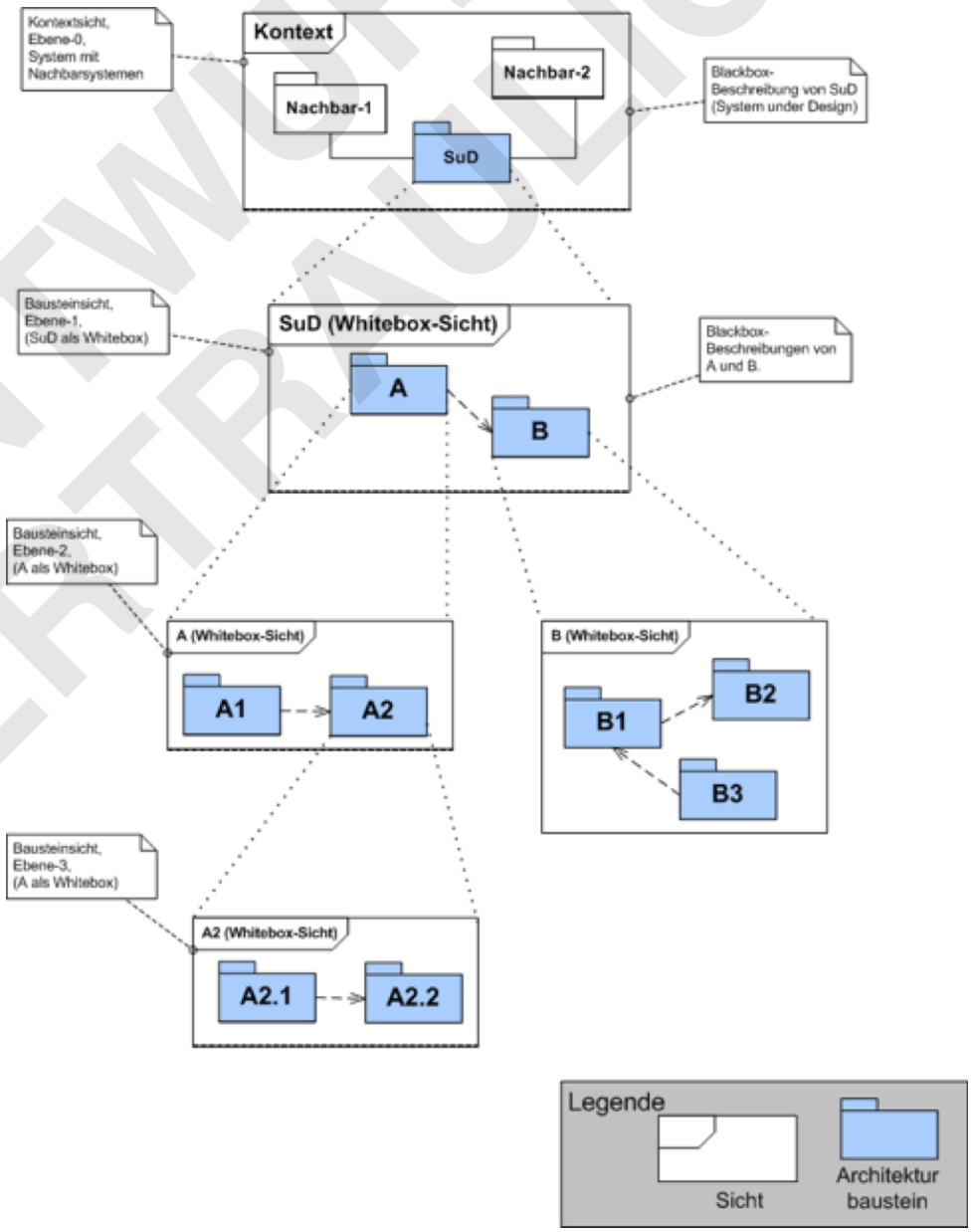
Statische Zerlegung des Systems in Bausteine (Module, Komponenten, Subsysteme, Teilsysteme, Klassen, Interfaces, Pakete, Bibliotheken, Frameworks, Schichten, Partitionen, Tiers, Funktionen, Makros, Operationen, Datenstrukturen...) sowie deren Beziehungen.

Motivation.

Dies ist die wichtigste Sicht, die in jeder Architekturdokumentation vorhanden sein muss. In der Analogie zum Hausbau bildet die Bausteinsicht den *Grundrissplan*.

Form.

Die Bausteinsicht ist eine hierarchische Sammlung von Blackbox- und Whitebox- Beschreibungen (siehe Abbildung unten):



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

Ebene 2 zoomt dann in einige Bausteine der Ebene 1 hinein. Sie enthält somit alle vorhandenen Whitebox-Beschreibungen von Bausteinen der Ebene 1, zusammen mit den Blackbox-Beschreibungen der Bausteine von Ebene 2.

Ebene 3 zoomt in einige Bausteine der Ebene 2 hinein, usw.

Whitebox-Template.

Enthält mehrere Bausteine (== Blackboxes), zu denen Sie jeweils eine Blackbox Beschreibung erstellen können.

Blackbox-Template (Kurzfassung).

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

(eine ausführlichere Fassung des Blackbox-Templates finden Sie weiter unten.)

Whitebox Gesamtsystem

An dieser Stelle beschreiben Sie die Whitebox-Sicht der Ebene 1 gemäß dem Whitebox-Template.

Das Überblicksbild zeigt das Innenleben des Gesamtsystems mit den darin enthaltenen Bausteinen 1 .. n, sowie deren Zusammenhänge und Abhängigkeiten.

Begründen Sie die Struktur: Warum gibt es diese Bausteine mit diesen Abhängigkeiten/Schnittstellen.

Erklären Sie ggfs. auch verworfene Alternativen, mitsamt Begründung, warum sie verworfen wurden.

Baustein 1 (Blackbox)

Beschreiben Sie diesen Baustein anhand des Blackbox-Templates. Nachfolgend finden Sie eine kompakte und eine ausführliche Fassung davon.

(In den folgenden Abschnitten finden Sie nur noch die Kurzfassung des Blackbox-Template)

Baustein 2 (Blackbox)

Baustein n (Blackbox)

Ebene 2

An dieser Stelle können Sie den inneren Aufbau (einiger) Bausteine aus Ebene 1 als Whitebox beschreiben.

Baustein 1 (Whitebox)

- ...zeigt das Innenleben von *Baustein 1* in Diagrammform mit den

  lokalen Bausteinen 1.1 - 1.n, sowie deren Zusammenhänge und

  Abhängigkeiten.

- begründet diese Struktur

  Baustein 1.1 **(Blackbox)**

  Baustein 1.2 **(Blackbox)**

  Baustein 1.n **(Blackbox)**

  Baustein 2 **(Whitebox)**

- ...zeigt das Innenleben von *Baustein 2* in Diagrammform mit den lokalen Bausteinen 2.1 - 2.n, sowie deren Zusammenhänge und Abhängigkeiten.
- begründet diese Struktur

  Baustein 2.1 **(Blackbox)**

  Baustein 2.2 **(Blackbox)**

  Baustein 2.n **(Blackbox)**

  Baustein 3 **(Whitebox)**

- ...zeigt das Innenleben von *Baustein 3* in Diagrammform mit den lokalen Bausteinen 3.1 - 3.n, sowie deren Zusammenhänge und Abhängigkeiten.
- begründet diese Struktur

  Hier Whitebox-Erläuterung für Baustein 3 einfügen

  Baustein 3.1 **(Blackbox)**

  Baustein 3.2 **(Blackbox)**

  Baustein 3.n **(Blackbox)**

Ebene 3

An dieser Stelle können Sie den inneren Aufbau (einiger) Bausteine aus Ebene 2 als Whitebox beschreiben.

Welche Bausteine Ihres Systems Sie hier beschreiben, müssen Sie selbst entscheiden. Bitte stellen Sie dabei Relevanz vor Vollständigkeit. Skizzieren Sie wichtige, überraschende, riskante, komplexe oder besonders volatile Bausteine. Normale, einfache oder standardisierte Teile sollten Sie weglassen.

Baustein 1.1 **(Whitebox)**

- ...zeigt das Innenleben von *Baustein 1.1* in Diagrammform mit den lokalen Bausteinen 1.1.1 - 1.1.n, sowie deren Zusammenhänge und

Abhängigkeiten.

- begründet diese Struktur

Baustein 1.1.1 **(Blackbox)**

Baustein 1.1.2 **(Blackbox)**

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:

- Dynamische Sichten
- Prozesssichten
- Ablaufsichten

Suchen Sie sich interessante Laufzeitszenarien heraus, z.B.: - Wie werden die wichtigsten Use-Cases durch die Architekturbausteine

bearbeitet?

- Welche Instanzen von Architekturbausteinen gibt es zur Laufzeit und

wie werden diese gestartet, überwacht und beendet?

- Wie arbeiten Systemkomponenten mit externen und vorhandenen

Komponenten zusammen?

- 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:

- Die wichtigsten 3-5 Anwendungsfälle
- Systemstart
- Das Verhalten an den wichtigsten externen Schnittstellen
- 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

- Laufzeitdiagramm
- Erläuterung der Besonderheiten bei dem Zusammenspiel der

  Bausteininstanzen in diesem Diagramm

Laufzeitszenario 2

- Laufzeitdiagramm
- Erläuterung der Besonderheiten bei dem Zusammenspiel der

  Bausteininstanzen in diesem Diagramm

  ...

Laufzeitszenario *n*

Verteilungssicht

Deployment: Auf welcher Hardware werden die Bausteine betrieben?

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).

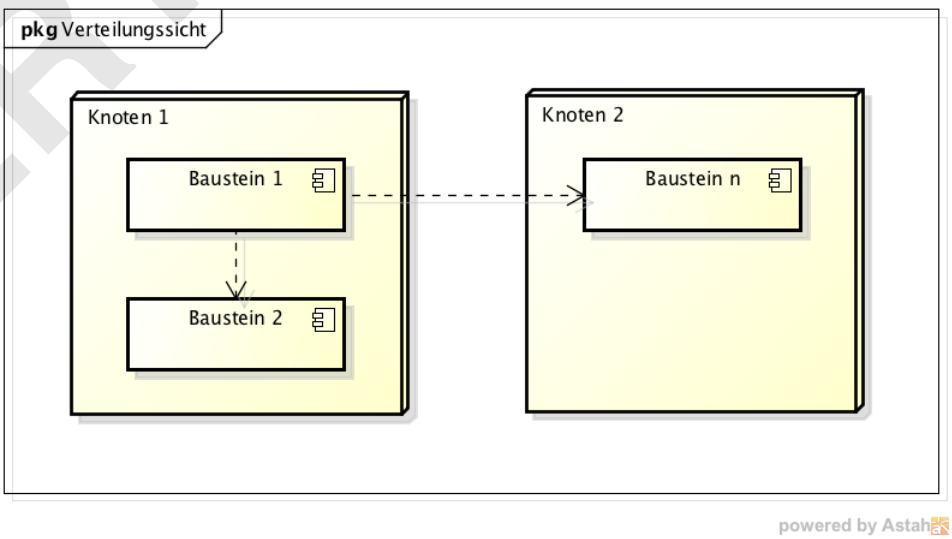
Motivation.

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



Quelle

Verteilungsdiagramm Ebene 1

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

Prozessor 1

Struktur gemäß Knoten-Template (node-template):

- Beschreibung
- Leistungsmerkmale
- Zugeordnete Software- Bausteine
- Sonstige Verwaltungsinformationen
- Offene Punkte

Prozessor 2

Struktur gemäß Knoten-Template:

- Beschreibung
- Leistungsmerkmale
- Zugeordnete Software- Bausteine
- Sonstige Verwaltungsinformationen
- Offene Punkte

...

Prozessor *n*

Struktur gemäß Knoten-Template:

- Beschreibung
- Leistungsmerkmale
- Zugeordnete Software- Bausteine
- Sonstige Verwaltungsinformationen
- Offene Punkte

Kanal 1

Inhalt.

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

Motivation.

Spezifizieren Sie mindestens 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

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: - Flüchtige Speichermedien (Hauptspeicher oder Cache) sind technisch

nicht für dauerhafte Speicherung ausgelegt. Daten gehen verloren,

wenn die entsprechende Hardware ausgeschaltet oder heruntergefahren

wird.

- Die Menge der von kommerziellen Software-Systemen bearbeiteten Daten

übersteigt üblicherweise die Kapazität des Hauptspeichers.

- 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 gar nicht 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: - Wie können Daten auf dem Transport (beispielsweise über offene Netze

wie das Internet) vor Missbrauch geschützt werden?

- Wie können Kommunikationspartner sich gegenseitig vertrauen?
- Wie können sich Kommunikationspartner eindeutig erkennen und vor

  falschen Kommunikationspartner schützen?
- 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: - Logging kann fachliche oder technische Protokollierung sein, oder

eine beliebige Kombination von beidem.

- 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.
- Technische Protokolle sind Informationen für Betreiber oder

  Entwickler. Sie dienen der Fehlersuche sowie der Systemoptimierung.
- 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-Systemen wird unter anderem durch ihre Konfigurierbarkeit beeinflusst, die Möglichkeit, manche Entscheidungen hinsichtlich der Systemnutzung erst spät zu treffen. Konfigurierbarkeit kann zu folgenden Zeitpunkten erfolgen: - Während der Programmierung: Dabei werden beispielsweise Server-,

Datei- oder Verzeichnisnamen direkt ("hart") in den Programmcode

aufgenommen.

- Während des Deployments oder der Installation: Hier werden

  Konfigurationsinformationen für eine bestimmte Installation

  angegeben, etwa der Installationspfad.
- Beim Systemstart: Hier werden Informationen vor oder beim

  Programmstart dynamisch gelesen.
- 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. - Konzept, Vorgehensweise oder Werkzeuge zur Datenübernahme und

initialen Befüllung mit Daten

- Konzept zur Systemeinführung oder zeitweiliger Parallelbetrieb von

Alt- und Neusystem

Müssen Sie bestehende Daten migrieren? Wie führen Sie die benötigten syntaktischen oder semantischen 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)?

Stapel-/Batchverarbeitung

Welche Geschäftsprozess-Schritte lassen sich in Stapelverarbeitung erledigen? Wie werden dazu Datenflüsse und Verarbeitungsschritte organisiert? Welche Mechanismen zur Fehlerverarbeitung werden eingesetzt? Sollen fehlgeschlagene Schritte wieder aufgesetzt werden können? Welche Bereinigungsschritte sind dazu notwendig? Welche Ablaufrahmen (Batch-Framework) wird dazu eingesetzt?

Drucken

Welche spezifischen Anforderungen zum Ausdrucken von Tabellen, Listen, Reports hat das System: z.B. Formate, Layouts, Druckmengen, Lieferzeiten, techn. Integration und Schnittstellen? Welche Eigenschaften haben die Druckgeräte? Können Spool-Verfahren eingesetzt werden?

Reporting

Welche Anforderungen gibt es zum Erstellen von Berichten / Reports inkl. Kennzahlen? Welche Repoorting-Werkzeuge werden eingesetzt? Welche Berechtigungen sind mit bestimmten Kennzahlen verbunden? Wie schützt man die Echtheit der Reports vor Manipulation? Müssen Reports sicher abgelegt werden können?

Archivierung

Ist für das System zu erwarten, dass bestimmte Daten aus technischer oder fachlicher Sicht archiviert werden müssen, ggf. periodisch? Welche Konzept existiert dazu? Wie lauten die Rahmenbedingen für die Archivierung (Dauer der Aufbewahrung, Geschwindigkeit der Wiederherstellung, usw.)?

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?

Entscheidungskriterien

Nach welchen Kriterien treffen Sie (oder die jeweiligen Entscheider) diese Entscheidung? Denken Sie an technische, organisatorische, juristische oder kommerzielle Kriterien, befragen Sie dazu die relevanten ?

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: - 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.

- 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

- 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.

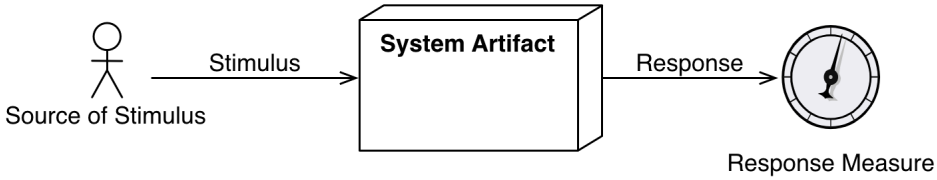


Abbildung: Schematische Darstellung von Szenarien (nach [Bass+03])

Szenarien bestehen aus folgenden wesentlichen Teilen (hier zitiert aus [Starke05], die ursprüngliche Gliederung stammt aus []): - 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.

- Quelle des Auslösers (source): beschreibt, woher der Auslöser kommt. Beispiele: intern oder extern, Benutzer, Betreiber, Angreifer, Manager.
- 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.
- Systembestandteil (artifact): beschreibt, welcher Bestandteil des Systems vom Auslöser betroffen ist. Beispiele: Gesamtsystem, Datenbank, Webserver.
- 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?
- 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 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) explizit 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.

Szenario	Auslöser	Metrik
...	...	...

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, ...

Risiko	Priorität	Konsequenz	Erläuterung
...	...	...	...

Glossar

Inhalt.

Die wichtigsten Begriffe der Software-Architektur in alphabetischer Reihenfolge

Motivation.

Sie sollten relevante Begriffe im Rahmen eines Systems oder eines Teams konsistent verstehen und verwenden.

Form.

einfache zweispaltige Tabelle mit ***Begriff*** und ***Definition*** bzw. Zitat und Verweis.

Begriff	Definition
...	...

dal: **Hinweis:** Wenn das Glossar zu lang wird, ist es ggf. ratsam eine Anlage anzufertigen und darauf zu verweisen.

Anlagen

Markdown Compatibility

Basically taken from http://learn.getgrav.org/content/markdown

kramdown : A Markdown-superset converter

markdown : Super awesome, tasty markup language

Footnotes[2] have a label[3] and the footnote's content.

h2 Heading

h3 Heading

h4 Heading

h5 Heading

h6 Heading

rendered as bold text

rendered as italicized text

~~Strike through this text.~~

> **Fusion Drive** combines a hard drive with a flash storage (solid-state drive) and presents it as a single logical volume with the space of both drives combined.

- valid bullet
- valid bullet
- valid bullet

1. Lorem ipsum dolor sit amet
2. Consectetur adipiscing elit
3. Integer molestie lorem at massa
4. Facilisis in pretium nisl aliquet
5. Nulla volutpat aliquam velit
6. Faucibus porta lacus fringilla vel
7. Aenean sit amet erat nunc
8. Eget porttitor lorem
9. Lorem ipsum dolor sit amet
10. Consectetur adipiscing elit
11. Integer molestie lorem at massa
12. Facilisis in pretium nisl aliquet
13. Nulla volutpat aliquam velit
14. Faucibus porta lacus fringilla vel
15. Aenean sit amet erat nunc
16. Eget porttitor lorem

In this example, `<section></section>` should be wrapped as **code**.

`markup Sample text here...`

```
grunt.initConfig({
  assemble: {
    options: {
      assets: 'docs/assets',
      data: 'src/data/*.{json,yml}',
      helpers: 'src/custom-helpers.js',
      partials: ['src/partials/**/*.{hbs,md}']
    },
    pages: {
      options: {
        layout: 'default.hbs'
      },
      files: {
        './': ['src/templates/pages/index.hbs']
      }
    }
  }
};
```

Option	Description
data	path to data files to supply the data that will be passed into templates.
engine	engine to be used for processing templates. Handlebars is the default.
ext	extension to be used for dest files.

Option	Description
data	path to data files to supply the data that will be passed into templates.
engine	engine to be used for processing templates. Handlebars is the default.
ext	extension to be used for dest files.

Assemble

Upstage

1. This is a footnote content. ↩
2. This is a footnote content. ↩
3. A footnote on the label: "@#$%". ↩