SPAsS-Tool

Architekturdokumentation nach ARC-42

2022-06-10

Einführung und Ziele

Aufgabenstellung

Entwickelt wird das SPAsS-Tool (Semester-Planungs-Anwendung für studierende Studierende). Hierbei handelt es sich um einen individuellen interaktivem Studienplaner.

Das System ist dazu gedacht Studenten / Studierenden (wie auch immer) den Aufbau und die Planung ihres Studiums (Modulwahl etc.) zu erleichtern.

Es wird möglich sein, dass jeder Studierende sich einen individuellen Stundenplan bauen kann – schließlich gibt es individuelle Fälle der Studierenden, die eine Planung abseits der Regelstudienzeit erfordern.

Es ist uns wichtig, die Anwendung Benutzerfreundlich zu gestalten, so dass der User mit gezielten Schritten durch unser System geleitet wird. Ein kleiner Teil der Anwendung wird nur für den Administrator sichtbar sein, damit er die Grundlagen festlegen kann und gegebenenfalls Module einrichten oder anpassen kann.

Qualitätsziele

Usability / Benutzbarkeit

- JavaFX GUI muss so eindeutig gestaltet sein, dass der Benutzer es ohne Erklärung individuell nutzen kann
- Leistungen / Module (Kästchen) können per Drag & Drop zwischen Semestern verschoben werden (Orientierung an bekannten Angebots-Restriktionen)

Verlässlichkeit, Robustheit

- Mindestens alle Hauptfunktionen müssen anhand von JUnit5-Tests testbar sein
- Der Benutzer muss darauf hingewiesen werden, wenn er falsch mit dem System interagiert – zum Beispiel die Fortschrittsregelung beim Verschieben der Module nicht einhält

Performance, Skalierbarkeit

- Erweiterbarkeit / Anpassbarkeit, ein Administrator soll ohne weitere Schulung neue Module hinzufügen können
- Jeder Student kann seinen individuellen Studienplan im System einpflegen

Stakeholder

Inhalt

Expliziter Überblick über die Stakeholder des Systems – über alle Personen, Rollen oder Organisationen –, die

- die Architektur kennen sollten oder
- von der Architektur überzeugt werden müssen,
- mit der Architektur oder dem Code arbeiten (z.B. Schnittstellen nutzen),
- die Dokumentation der Architektur für ihre eigene Arbeit benötigen,
- Entscheidungen über das System und dessen Entwicklung treffen.

Motivation

Sie sollten die Projektbeteiligten und -betroffenen kennen, sonst erleben Sie später im Entwicklungsprozess Überraschungen. Diese Stakeholder bestimmen unter anderem Umfang und Detaillierungsgrad der von Ihnen zu leistenden Arbeit und Ergebnisse.

Form

Tabelle mit Rollen- oder Personennamen, sowie deren Erwartungshaltung bezüglich der Architektur und deren Dokumentation.

Rolle	Kontakt	Erwartungshaltung
Entwickler	*Arthur Fieguth*	*<Erwartung-1>*
<Rolle-2>	*<Kontakt-2>*	*<Erwartung-2>*

Randbedingungen

Unser Team besteht aus 3 Leuten (Arthur Fieguth, Beate Arnold und Marie Bohnert).
Der zu erfüllende Zeitraum streckt sich vom 06. Mai 2022 bis 07. Juli 2022.
Es sind zwei Zwischenbesprechungen vereinbart. Am 27. Mai wird der Standpunkt „Anforderungsspezifikation“ (Was soll gemacht werden) überprüft. Am 17. Juni wird das „Design-Dokument“ (Konzept der softwaretechnischen Umsetzung) vorgestellt. Es wird nach Abgabe des Projekts eine Vorstellung der fertigen Anwendung geben.

Die technischen Rahmenbedingungen setzen sich aus einer Linux funktionsfähigen Anwendung mit Gradle build-, test- und ausführbaren JavaFX-Anwendung (Java 17) zusammen.

Kontextabgrenzung

Inhalt

Die Kontextabgrenzung grenzt das System von allen Kommunikationsbeziehungen (Nachbarsystemen und Benutzerrollen) ab. Sie legt damit die externen Schnittstellen fest.

Differenzieren Sie fachliche (fachliche Ein- und Ausgaben) und technische Kontexte (Kanäle, Protokolle, Hardware), falls nötig.

Motivation

Die fachlichen und technischen Schnittstellen zur Kommunikation gehören zu den kritischsten Aspekten eines Systems. Stellen Sie sicher, dass Sie diese komplett verstanden haben.

Form

Verschiedene Optionen:

- Diverse Kontextdiagramme
- Listen von Kommunikationsbeziehungen mit deren Schnittstellen

Siehe Kontextabgrenzung in der online-Dokumentation (auf Englisch!).

Fachlicher Kontext

Inhalt

Festlegung **aller** Kommunikationsbeziehungen (Nutzer, IT-Systeme, …) mit Erklärung der fachlichen Ein- und Ausgabedaten oder Schnittstellen. Zusätzlich (bei Bedarf) fachliche Datenformate oder Protokolle der Kommunikation mit den Nachbarsystemen.

Motivation

Alle Beteiligten müssen verstehen, welche fachlichen Informationen mit der Umwelt ausgetauscht werden.

Form

Alle Diagrammarten, die das System als Blackbox darstellen und die fachlichen Schnittstellen zu den Nachbarsystemen beschreiben.

Alternativ oder ergänzend können Sie eine Tabelle verwenden. Der Titel gibt den Namen Ihres Systems wieder; die drei Spalten sind: Kommunikationsbeziehung, Eingabe, Ausgabe.

<Diagramm und/oder Tabelle>

<optional: Erläuterung der externen fachlichen Schnittstellen>

Technischer Kontext

Inhalt

Technische Schnittstellen (Kanäle, Übertragungsmedien) zwischen dem System und seiner Umwelt. Zusätzlich eine Erklärung (*mapping*), welche fachlichen Ein- und Ausgaben über welche technischen Kanäle fließen.

Motivation

Viele Stakeholder treffen Architekturentscheidungen auf Basis der technischen Schnittstellen des Systems zu seinem Kontext.

Insbesondere bei der Entwicklung von Infrastruktur oder Hardware sind diese technischen Schnittstellen durchaus entscheidend.

Form

Beispielsweise UML Deployment-Diagramme mit den Kanälen zu Nachbarsystemen, begleitet von einer Tabelle, die Kanäle auf Ein-/Ausgaben abbildet.

<Diagramm oder Tabelle>

<optional: Erläuterung der externen technischen Schnittstellen>

<Mapping fachliche auf technische Schnittstellen>

Lösungsstrategie

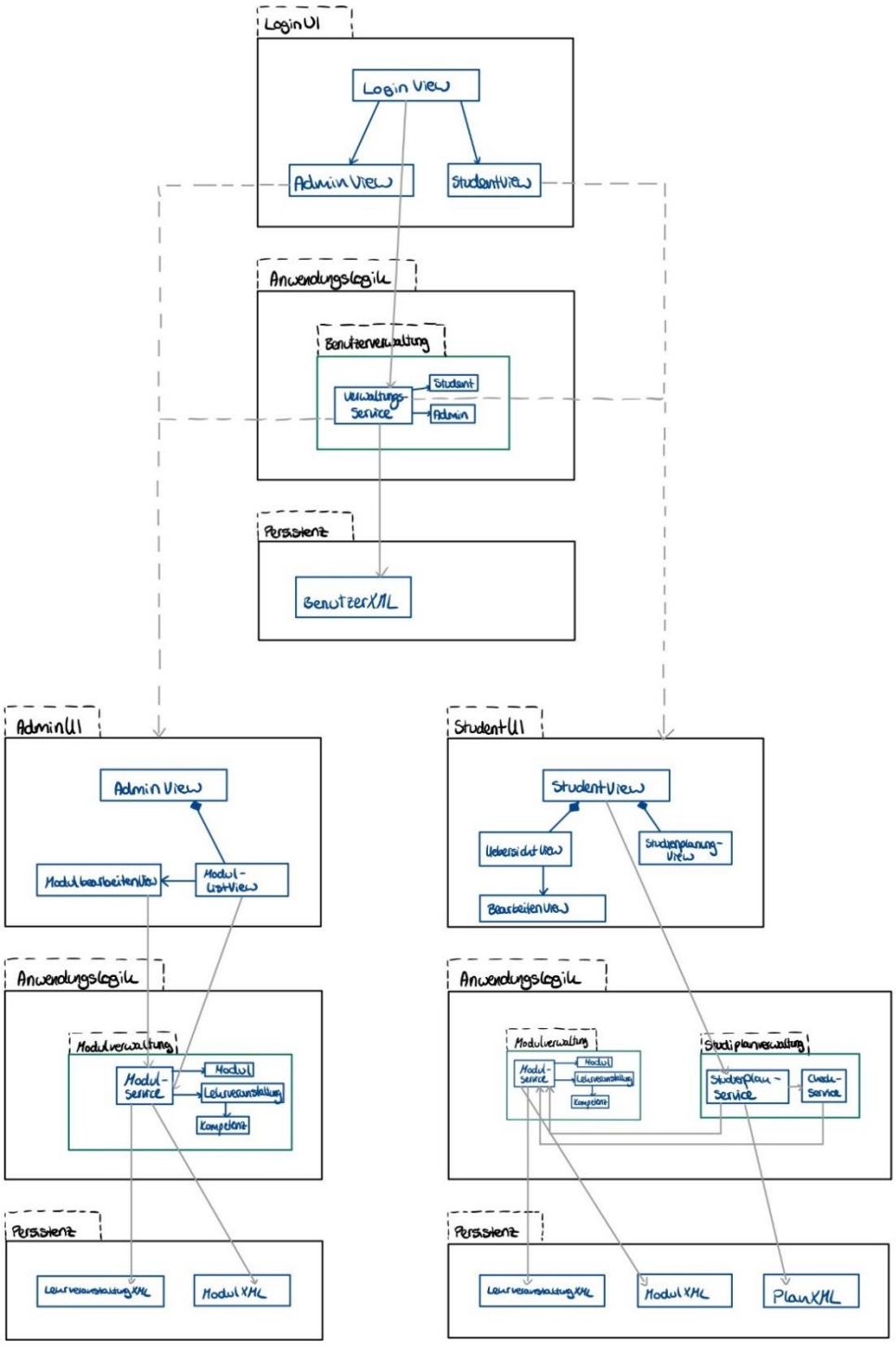
Wir haben uns dazu entschieden unser System mit Hilfe einer Schichtenarchitektur aufzubauen. Dabei werden die drei Schichten UI, Anwendungslogik und die Persistenz-Schicht unterschieden. Es erfolgt eine Trennung zwischen GUI und eigentlicher Programmlogik, sodass in unserem System sowohl UI als auch die Datenhaltung leicht austauschbar bleiben.

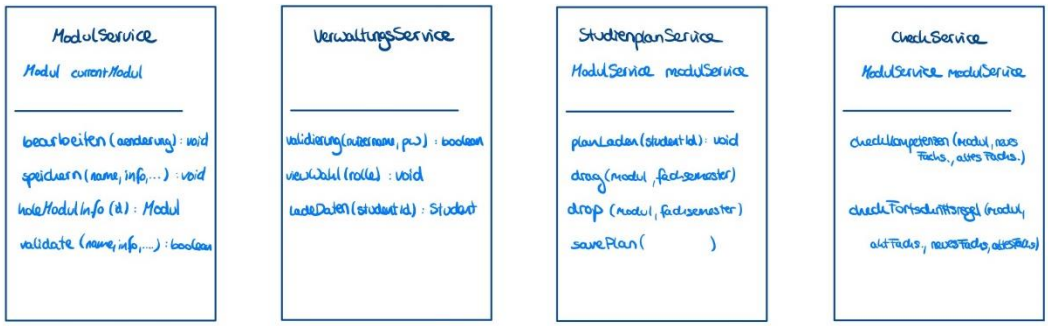
Bausteinsicht

Unser Gesamtsystem ist aufgeteilt in drei Subsysteme (Loginsicht, Adminsicht, Studentensicht). Jede dieser Ansichten ist in der Schichtenarchitektur gegliedert, bestehend aus UI, Anwendungslogik und Persistenz. Die LoginUI besteht aus der Komponente LoginView, welche zu der AdminView oder der Studentview nach erfolgreichem Login wechselt. Die LoginView hält außerdem eine Instanz des Verwaltungsservice der Anwendungslogik. Diese ist für die Benutzerverwaltung der Studenten und Administratoren zuständig. Der Verwaltungsservice hält eine Instanz der BenutzerXML, welche sich in der Persistenzschicht befindet. In der BenutzerXML werden die einzelnen Studenten und Administratoren mit id, name und passwort gespeichert.

Nach erfolgreichem Login des Administrators landet dieser in der AdminView, die sich in der AdminUI befindet. Teil der AdminView ist die ListView, diese besitzt eine Instanz der ModulbearbeitenView. Die ModulbearbeitenView sowie die ModulListView haben beide eine Instanz des Modulservices, welcher sich in der Anwendungslogik befindet. Der ModulService hält eine Instanz des Moduls sowie der Lehrveranstaltung, diese besitzt zudem eine Instanz Kompetenz. Außerdem hält der ModulService eine Instanz der LehrveranstaltungXML-Datei und die ModulXML-Datei aus der Persistenzschicht, um neue Module zu speichern oder vorhandene bearbeiten zu können.

Nach erfolgreichem Login des Studierenden landet dieser in der StudentView, die sich in der StudentUI befindet. Teil der StudentView sind die StudienplanungView sowie die UebersichtView, welche eine Instanz der BearbeitenView besitzt. Die StudentView hält eine Instanz des StudienplanServices aus der Anwendungslogik. Der StudienplanService eine Instanz des CheckService hält, befindet sich im Package „Studiplanverwaltung“. Der StudiplanService und der CheckService haben zudem eine Instanz des Modulservices des Modulverwaltungspackage. Der ModulService besitzt ebenfalls eine Instanz des Moduls und der Lehrveranstaltung, diese hält eine Instanz der Kompetenz und kann ebenfalls neue Module und Lehrveranstaltungen in den XML-Dateien speichern/ändern. Der StudienplanService besitzt eine Instanz der PlanXML Datei, um einzelne Studienpläne der Studierenden zu speichern oder zu bearbeiten.


LoginUI
Login View
Admin View
Student View
Anwendungslogik
Benutzerverwaltung
Verwaltungs-Service
Student
Admin
Persistenz
BenutzerXML
AdminUI
Admin View
Modul bearbeiten View
Modul-List View
Anwendungslogik
Modulverwaltung
Modul-Service
Modul
Lehrveranstaltung
Kompetenz
Persistenz
Lehrveranstaltung XML
Modul XML
StudentUI
Student View
Uebersicht View
Studienplanung-View
Bearbeiten View
Anwendungslogik
Modulverwaltung
Modul-Service
Modul
Lehrveranstaltung
Kompetenz
Studiplanverwaltung
Studienplan-Service
Checks-Service
Persistenz
Lehrveranstaltung XML
Modul XML
Plan XML



Um einen genaueren Einblick in die Service Klassen zu erhalten sind diese im Folgenden in ihren Funktionalitäten noch einmal detaillierter beschrieben:

ModulService:

Der ModulService hat ein aktuelles Modul und vier essenzielle Methoden. Die Bearbeitungsmethode ist vom Typ void und bekommt als Parameter die Änderungen von name, info, etc. mit. Ähnlich verhält sich die Speichermethode vom Typ void. OFFEN

Die Methode holeModulInfo ist vom Typ Modul und holt mit Hilfe der übergebenen Id das passende Modul.

Die validate Method vom Typ boolean bekommt die Parameter name, info, etc. übergeben und überprüft die jeweiligen Moduländerungen/Modulerzeugungen, ob die Eingaben zulässig sind oder das Modul schon vorhanden ist.

VerwaltungsService:

Der VerwaltungsService hat drei essenzielle Methoden. Die Validierungsmethode vom Typ boolean bekommt die Parameter nutzername und passwort. Hiermit prüft die Methode, ob der Nutzer Zugriff auf das Spass Tool hat.

Die ViewWahl Methode vom Typ void bekommt den Parameter Rolle und weist dem Benutzer die jeweilige View (AdminView oder StudentView) zu.

Die Methode LadeDaten vom Typ Student bekommt die StudentId übergeben und läd die jeweiligen Daten des Studenten.

StudentenplanService:

Der StudentenplanService hat den ModulService und vier essenzielle Methoden.

Die Methode PlanLaden vom Typ void mit dem Parameter Studentid läd den Studienplan des jeweils eingeloggten Studierenden.

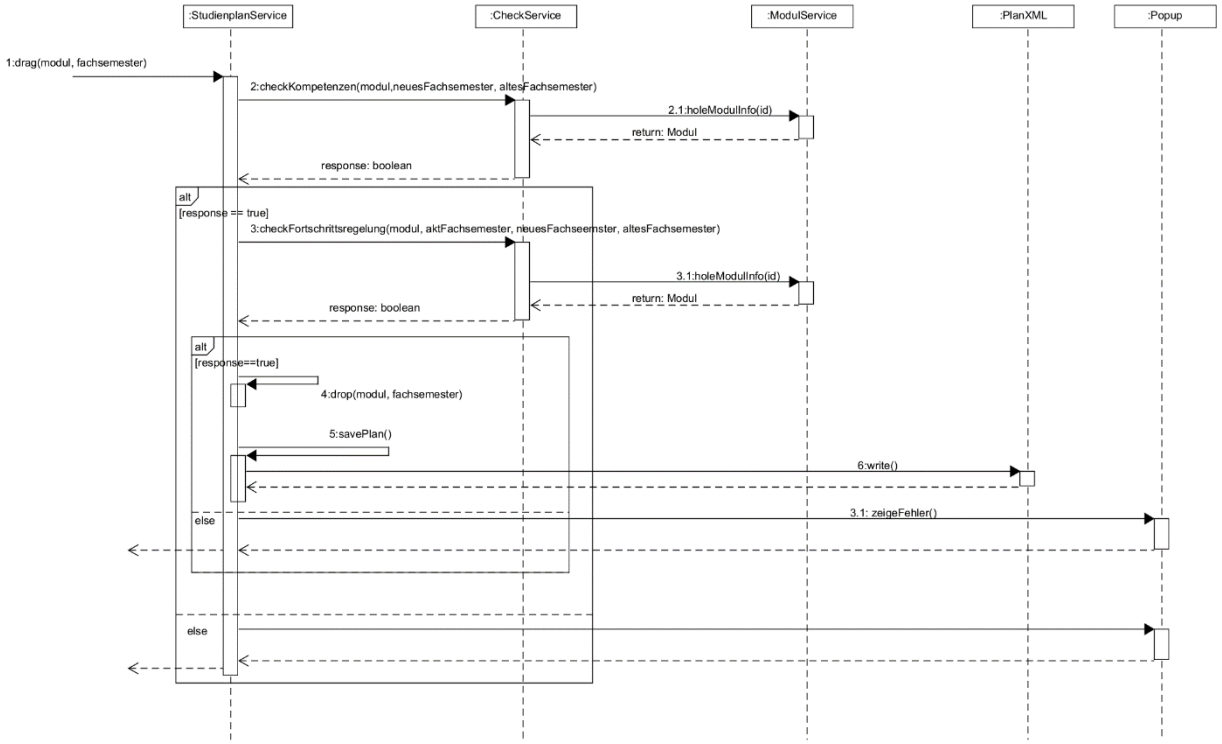
Die Methoden drag und drop vom Typ void bekommen die Parameter modul und fachsemester übergeben, welche das zu verschiebene Modul definieren.

CheckService:

Der CheckService hat den ModulService und zwei essenzielle Methoden. Die CheckKompetenzen Methode vom Typ boolean bekommt als Parameter modul, neuesFachsemester und altesFachsemester übergeben, um prüfen zu können, ob nötige Kompetenzen bereits vorhanden sind.

Die Methode CheckFortschrittsregel vom Typ boolean bekommt die Parameter modul, aktuellesFachsemester, neuesFachsemester und altesFachsemester, um prüfen zu können, ob das Modul in das Zielsemester verschoben werden kann.

Laufzeitsicht



Das erste Sequenzdiagramm zeigt den Vorgang des Drag and Drop Prozess mit den zusammenhängenden Klassen und Methoden.

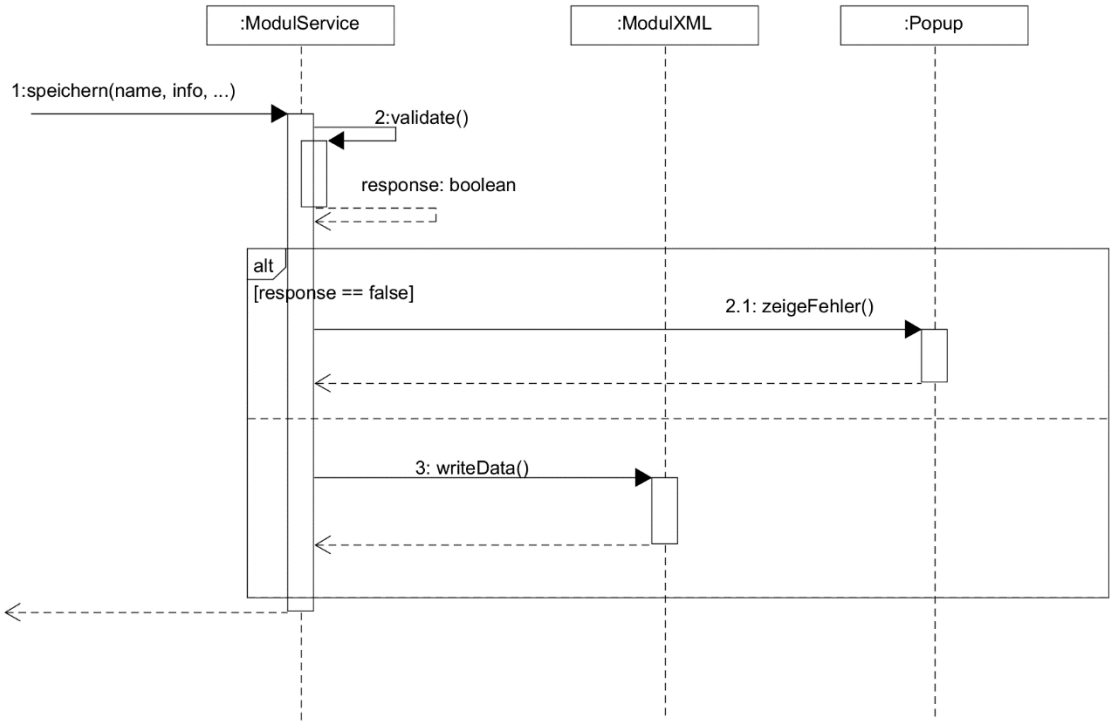
Zunächst wird die drag Methode im StudienplanService aufgerufen, indem ein bestimmtes Modul aus einem Fachsemester mit der Maus genommen wird.

Die Drag Methode muss während des Drags prüfen, ob ein Drop an der jeweiligen Stelle möglich ist. Dazu müssen die checkKompetenzen und die checkFortschrittsregelung aus dem CheckService aufgerufen werden. Beide benötigen die Informationen aus dem Modul und müssen dementsprechend die Methode holeModulInfo aus dem ModulService aufrufen.

In der checkKompetenz Methode wird anhand der benötigten Kompetenzen überprüft, ob das zu verschiebende Modul aus dem alten Fachsemester ins neue Fachsemester geschoben werden kann.

Gibt die checkKompetenz Methode true zurück, so wird im nächsten Schritt die Fortschrittsregelung überprüft. Gibt die checkFortschrittsregelung ebenfalls true zurück kann das Modul gedropped werden. Diese Methode wird aus dem StudienplanService aufgerufen. Dann wird die savePlan Methode ebenfalss aus dem StudienplanService aufgerufen, um den Plan zu speichern. Dazu muss der neue Plan in die PlanXML-Datei geschrieben werden.

Gibt die checkKompetenz Methode oder die checkFortschrittsregelung Methode false zurück, so wird ein Popup mit dem Fehler angezeigt.

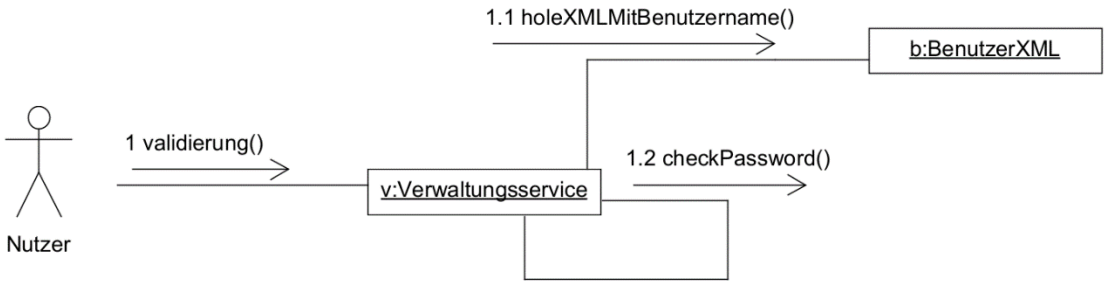


Das zweite Sequenzdiagramm beschreibt den Prozess des Speicherns eines Moduls. Die speichern Methode des ModulService wird mit name, info, etc. aufgerufen. Die Methode ruft

wiederum die validate Methode aus dem ModulService auf, um die eingetragenen Daten zu prüfen und festzustellen, ob es ein solches Modul bereits gibt.

Gibt die validate Methode false zurück, so wird ein Popup geöffnet, in dem eine Fehlermeldung angezeigt wird.

Gibt sie jedoch true zurück, so wird das neue Modul in die ModulXML-Datei geschrieben.



TODO:beschreiben

Verteilungssicht

Inhalt

Die Verteilungssicht beschreibt:

1. die technische Infrastruktur, auf der Ihr System ausgeführt wird, mit Infrastrukturelementen wie Standorten, Umgebungen, Rechnern, Prozessoren, Kanälen und Netztopologien sowie sonstigen Bestandteilen, und

2. die Abbildung von (Software-)Bausteinen auf diese Infrastruktur.

Häufig laufen Systeme in unterschiedlichen Umgebungen, beispielsweise Entwicklung-/Test- oder Produktionsumgebungen. In solchen Fällen sollten Sie alle relevanten Umgebungen aufzeigen.

Nutzen Sie die Verteilungssicht insbesondere dann, wenn Ihre Software auf mehr als einem Rechner, Prozessor, Server oder Container abläuft oder Sie Ihre Hardware sogar selbst konstruieren.

Aus Softwaresicht genügt es, auf die Aspekte zu achten, die für die Softwareverteilung relevant sind. Insbesondere bei der Hardwareentwicklung kann es notwendig sein, die Infrastruktur mit beliebigen Details zu beschreiben.

Motivation

Software läuft nicht ohne Infrastruktur. Diese zugrundeliegende Infrastruktur beeinflusst Ihr System und/oder querschnittliche Lösungskonzepte, daher müssen Sie diese Infrastruktur kennen.

Form

Das oberste Verteilungsdiagramm könnte bereits in Ihrem technischen Kontext enthalten sein, mit Ihrer Infrastruktur als EINE Blackbox. Jetzt zoomen Sie in diese Infrastruktur mit weiteren Verteilungsdiagrammen hinein:

- 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.
- Falls Ihre Infrastruktur-Stakeholder andere Diagrammarten bevorzugen, die beispielsweise Prozessoren und Kanäle zeigen, sind diese hier ebenfalls einsetzbar.

Siehe Verteilungssicht in der online-Dokumentation (auf Englisch!).

Infrastruktur Ebene 1

An dieser Stelle beschreiben Sie (als Kombination von Diagrammen mit Tabellen oder Texten):

- die Verteilung des Gesamtsystems auf mehrere Standorte, Umgebungen, Rechner, Prozessoren o. Ä., sowie die physischen Verbindungskanäle zwischen diesen,
- wichtige Begründungen für diese Verteilungsstruktur,
- Qualitäts- und/oder Leistungsmerkmale dieser Infrastruktur,
- Zuordnung von Softwareartefakten zu Bestandteilen der Infrastruktur

Für mehrere Umgebungen oder alternative Deployments kopieren Sie diesen Teil von arc42 für alle wichtigen Umgebungen/Varianten.

<Übersichtsdiagramm>

Begründung
<Erläuternder Text>

Qualitäts- und/oder Leistungsmerkmale
<Erläuternder Text>

Zuordnung von Bausteinen zu Infrastruktur
<Beschreibung der Zuordnung>

Infrastruktur Ebene 2

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

Für jedes Infrastrukturelement kopieren Sie die Struktur aus Ebene 1.

<Infrastrukturelement 1>

<Diagramm + Erläuterungen>

<Infrastrukturelement 2>

<Diagramm + Erläuterungen>

…

<Infrastrukturelement n>

<Diagramm + Erläuterungen>

Querschnittliche Konzepte

Inhalt

Dieser Abschnitt beschreibt übergreifende, prinzipielle Regelungen und Lösungsansätze, die an mehreren Stellen (=*querschnittlich*) relevant sind.

Solche Konzepte betreffen oft mehrere Bausteine. Dazu können vielerlei Themen gehören, beispielsweise:

- Modelle, insbesondere fachliche Modelle
- Architektur- oder Entwurfsmuster
- Regeln für den konkreten Einsatz von Technologien
- prinzipielle — meist technische — Festlegungen übergreifender Art
- Implementierungsregeln

Motivation

Konzepte bilden die Grundlage für *konzeptionelle Integrität* (Konsistenz, Homogenität) der Architektur und damit eine wesentliche Grundlage für die innere Qualität Ihrer Systeme.

Manche dieser Themen lassen sich nur schwer als Baustein in der Architektur unterbringen (z.B. das Thema „Sicherheit“).

Form

Kann vielfältig sein:

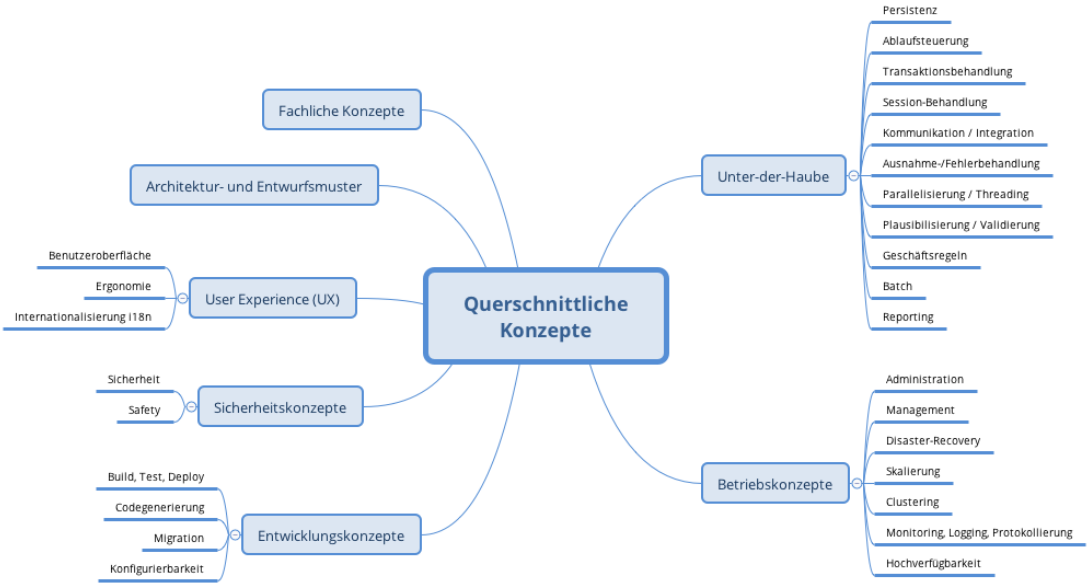
- Konzeptpapiere mit beliebiger Gliederung,
- übergreifende Modelle/Szenarien mit Notationen, die Sie auch in den Architektursichten nutzen,

- beispielhafte Implementierung speziell für technische Konzepte,
- Verweise auf „übliche“ Nutzung von Standard-Frameworks (beispielsweise die Nutzung von Hibernate als Object/Relational Mapper).

Struktur

Eine mögliche (nicht aber notwendige!) Untergliederung dieses Abschnittes könnte wie folgt aussehen (wobei die Zuordnung von Themen zu den Gruppen nicht immer eindeutig ist):

- Fachliche Konzepte
- User Experience (UX)
- Sicherheitskonzepte (Safety und Security)
- Architektur- und Entwurfsmuster
- Unter-der-Haube
- Entwicklungskonzepte
- Betriebskonzepte



Siehe Querschnittliche Konzepte in der online-Dokumentation (auf Englisch).

<Konzept 1>

<Erklärung>

<Konzept 2>

<Erklärung>

...

<Konzept n>

<Erklärung>

Architekturentscheidungen

Inhalt

Wichtige, teure, große oder riskante Architektur- oder Entwurfsentscheidungen inklusive der jeweiligen Begründungen. Mit "Entscheidungen" meinen wir hier die Auswahl einer von mehreren Alternativen unter vorgegebenen Kriterien.

Wägen Sie ab, inwiefern Sie Entscheidungen hier zentral beschreiben, oder wo eine lokale Beschreibung (z.B. in der Whitebox-Sicht von Bausteinen) sinnvoller ist. Vermeiden Sie Redundanz. Verweisen Sie evtl. auf Abschnitt 4, wo schon grundlegende strategische Entscheidungen beschrieben wurden.

Motivation

Stakeholder des Systems sollten wichtige Entscheidungen verstehen und nachvollziehen können.

Form

Verschiedene Möglichkeiten:

- ADR (Architecture Decision Record) für jede wichtige Entscheidung
- Liste oder Tabelle, nach Wichtigkeit und Tragweite der Entscheidungen geordnet
- ausführlicher in Form einzelner Unterkapitel je Entscheidung

Siehe Architekturentscheidungen in der arc42 Dokumentation (auf Englisch!). Dort finden Sie Links und Beispiele zum Thema ADR.

Qualitätsanforderungen

Qualitätsszenarien

Inhalt

Konkretisierung der (in der Praxis oftmals vagen oder impliziten) Qualitätsanforderungen durch (Qualitäts-)Szenarien.

Diese Szenarien beschreiben, was beim Eintreffen eines Stimulus auf ein System in bestimmten Situationen geschieht.

Wesentlich sind zwei Arten von Szenarien:

- Nutzungsszenarien (auch bekannt als 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 unmittelbaren Umgebung. Beispiel: Eine zusätzliche Funktionalität wird implementiert oder die Anforderung an ein Qualitätsmerkmal ändert sich.

Motivation

Szenarien operationalisieren Qualitätsanforderungen und machen deren Erfüllung mess- oder entscheidbar.

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

Form

Entweder tabellarisch oder als Freitext.

Glossar

Begriff	Definition
<Begriff-1>	*<Definition-1>*
<Begriff-2	*<Definition-2>*