Duale Hochschule Baden-Württemberg

Mannheim

Erste Projektarbeit

Erarbeitung eines Lösungsentwurfes für eine IT-Lösung zur Kapazitätsplanung der Kabinencrew

Studiengang Wirtschaftsinformatik - Sales & Consulting

Bearbeitungszeitraum: 15.05.2017 - 29.08.2017

Verfasser:	Julian Garske
Matrikelnummer:	6728241
Kurs:	WWI SCA16
Studiengangsleiter:	Dr. Frank Koslowski
Wissenschaftlicher Betreuer:	Günter Stumpf
Telefon:	01511 8237778
Mailadresse:	guenter.stumpf@esosec.de
Ausbildungsbetrieb:	Lufthansa Systems GmbH & Co. KG Am Prime Parc 1 D 65479 Raunheim
Unternehmensbetreuer:	Berger, Iwan
Telefon(Firma):	+49 (0)69 696 74135
Mailadresse(Firma):	iwan.berger@lhsystems.com

Inhaltsverzeichnis

Kurzfassung (Abstract)

Abkürzungsverzeichnis

BT Beschäftigungstage

CAB Compas Cabin

CDB Crew Database

CMS Crew Management System

COC Compas Cockpit

KG Kleingruppe

NL/C NetLine/Crew

PU Planungseinheit

Abbildungsverzeichnis

Anlagenverzeichnis

1 Einleitung

1.1 Motivation

Bereits seitdem es kommerzielle Passagierflüge gibt, ist eine konkrete Zuteilung der Besatzung für die Flüge notwendig. Dabei müssen nicht nur Fehlzeiten wie Urlaub oder Krankheit, sondern auch andere Faktoren z.B. Teilzeit berücksichtigt werden. Außerdem werden für unterschiedliche Flugzeuge auch unterschiedliche Qualifikationen benötigt. Bei immer größer werdenden Fluggesellschaften, wie der Lufthansa mit ca. 20.000 Besatzungsmitgliedern, stellt die Zuteilung daher oft eine Herausforderung dar. Die sogenannte Kapazitätsplanung ist deshalb für viele Airlines ein wichtiger Bestandteil des Flugbetriebs.
Eine Erweiterung der Kapazitätsplanung ist die Schulungsplanung. Schulungen dauern oft einige Wochen und müssen daher langfristig vorher geplant werden, sodass Fehlzeiten ausgeglichen werden können und die Qualifikationen nach der Schulung aktualisiert werden. Das Ziel der Kapazitäts- und Schulungsplanung ist, „durch rechtzeitige Neueinstellungen und Umschulungen die richtige Menge an Piloten mit der richtigen Qualifikation zum richtigen Zeitpunkt auf einer Flotte bereitzustellen“ (vgl. Berger, 2016, S.19).

Für die Kapazitäts- und Schulungsplanung der Cockpit-Besatzung im Lufthansa Konzern gibt es seit 2000 das von Lufthansa Systems entwickelte System Compas Cockpit (COC). Dadurch ist eine Planung der Einteilung und Schulungen für die etwa 5.000 Piloten bis zu 15 Monate in Zukunft möglich. Bis heute wird die Funktionalität dieses Programms regelmäßig erweitert.
Mit der Entwicklung von COC entstand eine Vorstellung, ein ähnliches Programm auch für die Kabinenbesatzung zu entwickeln. Durch diese systematische Lösung sollen Zeit gespart und Fehler minimiert werden (vgl. Winkel, 2017, S.4). Ein Auftrag für das Projekt wurde von dem Kunden, der Lufthansa Passage Airline, noch nicht veröffentlicht, weshalb es bis heute nicht zu einem Projekt mit einer konkreten Analyse oder Entwicklung kam.

Letztes Jahr hatte sich die Firma M2P Consulting bereits mit der bestehenden Kapazitätsplanung auseinandergesetzt und geprüft, wie man diese verbessern könne und ob COC dafür in Frage käme. Sie kamen zu dem Ergebnis, dass kurzfristig zwar die bisherige Vor-

gehensweise der Kapazitätsplanung (siehe 1.6) optimiert werden könne, um die Qualität zu verbessern, langfristig solle es aber durch eine systemseitige Lösung abgelöst werden (vgl. M2P, 2016, S.8-10).

1.2 Problemstellung und -abgrenzung

Zurzeit werden die Einsätze der Kabinenbesatzung mithilfe von verschiedenen Excel-Tabellen geplant. Diese ca. 200 Tabellen enthalten Daten aus unterschiedlichen Quellen, die für die Zuteilung der Besatzung erforderlich sind. Aufgrund der großen Datenmenge und den Verflechtungen der Tabellen untereinander kommt es oft zu Problemen und Fehlern bei der Kapazitätsplanung (vgl. Berg u. a., 2017).

Nach dem Vorbild von COC soll eine automatisierte Kapazitäts- und Schulungsplanung jetzt auch für die Kabine, also die komplette Crew, unter dem Titel Compas Cabin (CAB) ermöglicht werden. Im Vordergrund steht dabei erst einmal die Kapazitätsplanung, die in dieser Arbeit behandelt wird. Im ersten Schritt wird dafür die Bestandsrechnung analysiert und angepasst.
Als Ausgangspunkt für das Projekt wird COC genutzt, welches die gewünschten Funktionen schon für die Cockpit-Besatzung enthält. Um CAB in COC zu integrieren müssen nur einige Unterschiede, die es zwischen der Kapazitätsplanung der Cockpit- und der der Kabinenbesatzung gibt, behoben und geklärt werden. Diese Unterschiede, die zu der Problematik führen, dass COC nicht so wie es ist auf die Kabinenbesatzung angewendet werden kann, werden im Kapitel 2 genauer erläutert . Der Berechnungsalgorithmus soll nach der Behebung der Probleme ähnlich wie in COC angewendet werden können.

Besonderer Fokus liegt auf der Integration in die bestehende Systemlandschaft von COC und die Entwicklung einer mandantenfähigen Lösung, um mehrere Airlines der Lufthansa Gruppe zu integrieren. Dabei sollen modernere Architekturansätze verwendet werden, um sich von der bereits veralteten Architektur von COC loszulösen.
Im Anwenderkonzept wird gefordert , dass die Ergebnisse zur Weiterverarbeitung in Excel exportiert werden können. Darüberhinaus sollten die Berechnungsläufe getrennt von COC erfolgen, um sich gegenseitig nicht zu beeinträchtigen (vgl. dazu Winkel, 2015).

1.3 Ziel der Arbeit

Das Ziel dieser Projektarbeit ist, einen Lösungsentwurf für die Entwicklung eines explorativen Prototyps zur Kapazitätsplanung des Kabinenpersonals zu entwerfen. Daraus sollen vor allem die Anforderungsspezifikationen erkennbar sein. Der explorative Prototyp soll nach seiner Entwicklung evolutionär genutzt werden und damit Ausgangspunkt für Erweiterungen sein.
Dieser zu entwickelnde Prototyp soll Grundlage für die sogenannte Deltarechnung sein. Dabei wird der aktuelle Personenbestand mit dem Bedarf ins Verhältnis gesetzt. Für zukünftige Prognosen werden Prämissen und Erfahrungswerte genutzt. (zu ergänzen: nur Bestandsrechnung oder hauptsächlig, je nachdem wie weit wir kommen). Zunächst wird dafür die Bestandsrechnung angepasst. Darüber hinaus wird beschrieben, wie weit COC dazu als Vorlage dienen und ob NetLine/Crew (NL/C) für die Entwicklung hilfreich sein kann. Für die in Kapitel 2 genannten Probleme und Merkmale soll eine Lösung gefunden werden, um den Entwurf zu realisieren.

1.4 Vorgehen

Zunächst geht es darum, die Anforderungen zu ermitteln und eine Anwenderdokumentation zu erstellen. Dafür muss die aktuelle Vorgehensweise der Kapazitätsplanung verstanden und die Anforderungen, die das zu entwickelnde Programm erfüllen soll, erfasst werden. Die Anforderungen werden in diesem Projekt hauptsächlich von den momentanen Planerinnen, die das spätere Programm anwenden, gestellt.
Danach wird entschieden, wie COC als Vorlage genutzt werden kann, indem Gemeinsamkeiten und Unterschiede zwischen CAB und COC ermittelt werden und ein Weg gefunden wird, ein COC-ähnliches Programm auf die Kabinenbesatzung anzuwenden. Im Vorderung steht dabei als erstes die Bestandsrechnung. Bestimmte Bereiche des Programms müssen im Vergleich zu COC modernisiert werden, vieles kann aber auch so übernommen werden (vgl. Wagner u. a., 2017b).
Die erstellte Anwenderdokumentation dient als Grundlage für die Entwicklung des Prototyps. Man soll daraus präzise erfassen können, welche Anforderungen das zu entwickelnde Programm wie erfüllen soll.

1.5 Überblick über Compas Cockpit

COC (Crew Operation Manpower Planning Advanced System) dient als Ausgangspunkt für das Projekt und nur einige Besonderheiten der Kabine müssen angepasst werden, um COC so dafür zu übernehmen (vgl. Wagner u. a., 2017b). Deshalb sind ein Überblick über COC und Kenntnisse der wichtigsten Funktionen davon wichtig. Dieser Überblick stammt aus einem betriebsinternen Dokument (vgl. Berger, 2016).

COC ist das zentrale Tool für die Kapazitätsplanung des Cockpitpersonals. Ziel dieser Planung ist es, die richtige Anzahl an Piloten mit passenden Qualifikationen zu einem bestimmten Zeitpunkt auf eine Flotte bereitstellen zu können. Das kann kurzfristig gesteuert oder langfristig prognostiziert werden. Diese Planung ist sehr komplex, da sehr viele Einflussfaktoren dort berücksichtigt werden müssen.

Um während der Planung den Überblick zu behalten, teilt COC jeden Piloten eindeutig in eine Planungseinheit (PU) ein, die durch Flugzeugtyp, Flotte, Homebase , Funktion und Subfunction definiert ist. Dadurch kann für jede PU für einen bestimmten Zeitraum tagesgenau z.B. der Bestand und Bedarf ermittelt und verglichen werden.

Zusätzlich wurde das Programm mehrmals erweitert, sodass es mittlerweile aus vier internen Hauptmodulen, z.B. einem Schulungsplaner und der Deltarechnung, die den Flugbestand ins Verhältnis zu ihrem Bedarf setzt, und zwei angrenzenden Zusatzmodulen besteht. Die Zusatzmodule sind Schnittstellen zu der Compas Datenbank und zu dem Berwerbersystem für das Cockpit (vgl. Berger, 2016, S.19).

1.6 Bisherige Vorgehensweise der Kapazitätsplanung für die Kabinenbesatzung

Bisher gibt es für die Kapazitätsplanung der Kabine noch kein wirkliches Tool. Zwei Angestellte der Lufthansa Passage planen die Kapazitäten mithilfe von Excel-Tabellen. Dafür verknüpfen sie ca. 220 Tabellen miteinander (vgl. Berg u. a., 2017). Da es kaum möglich ist, jede einzelne Person konkret einzuteilen, wird jede einer Kleingruppe (KG) eingeteilt. Diese werden ähnlich wie die PUs durch Funktion, Homebase, Unterfunktion, Fluggesellschaft und ein oder mehreren Flugzeugtypen definiert. Auf Grundlage dieser KGs wird die Kapazitätsplanung durchgeführt und nur bei Bedarf können einzelne Personen genauer betrachtet werden.

Nach M2P Consulting sei „die Handlungsfähigkeit der Kapazitätsplanung in Bezug auf

die zukünftige Herausforderungen stark eingeschränkt“ (M2P, 2016, S.5) und „deckt keine der definierten Soll-Funktionalitäten der Kapazitätsplanung ausreichend ab“ (M2P, 2016, S.6). Bei den Soll-Funktionalitäten handelt es sich nach M2P um langfristige Bereederung und Budgetplanung und um mittelfristige Bereederung (vgl. M2P, 2016, S.6).

2 Problemanalyse

2.1 Erhebung der Pilotendaten von der Crew Database

Wie schon im ersten Gespräch festgestellt, soll CAB im Vergleich zu COC, das die Daten von etwa 6.000 Piloten verarbeitet, mit Daten von ca. 20.000 Personen umgehen können. Das wöchentliche Formatieren dieser Daten dauert bereits in COC einige Stunden, für CAB wäre es also ungefähr das Vierfache. Es besteht dabei jedoch das Risiko, dass die Laufzeit eher exponentiell ansteigt. Hinzu kommt, dass die Datenbank nicht lange von CAB ausgelastet werden darf, weil auch andere Schnittstellen darauf zugreifen. Eine zu starke Auslastung kann dabei zu großen Problemen in sehr unterschiedlichen Bereichen und Systemen führen (vgl. Wagner u. a., 2017b).
Bei der Erhebung der Daten in COC werden von der Crew Database (CDB), der zentralen Datenbank, die Daten aus unterschiedlichen Systemen herausgesucht und dann in eine neue View geladen, sodass sie alle tagesgenau für die nächsten 450 Tage vorliegen. Daraus entstehen zwei Tabellen, einmal für jede Person (personenbasiert) und die andere für jede PU (kumuliert). Die hohe Anzahl an Datensätzen führt dazu, dass die Tabellen knapp 45.000 und 3.000.000 Einträge enthalten, die alle von der CDB erhoben und berechnet werden (vgl. Hentschel u. a., 2017).

Aus diesem Problem resultiert, dass zur Datenerhebung aus der CDB eine neue Architektur benötigt wird, um die fast 20 Jahre alte von COC zu ersetzen. Hinzu kommt, dass historisch bedingt in COC viel zu viele nicht benötigte Daten angefordert werden, die früher genutzt wurden aber mittlerweile keinen Zweck mehr erfüllen (vgl. Hentschel u. a., 2017). Eine Veränderung des Architekturansatzes zur Datenerhebung setzt dabei natürlich Verständnis der bisherigen Architektur von COC voraus.

2.2 Integration in das Compas-Umfeld

Sobald CAB weit genug entwickelt worden ist, soll es, den Anforderungen nach, in das bereits bestehende COC eingegliedert werden. Dabei soll eine Auswahl des Programms möglich sein und CAB soll als Modul oder Erweiterung von COC genutzt werden können (vgl. Winkel, 2015, S.6). Daher muss das neue Programm in das bisherige System integriert werden und darf sich nicht zu stark davon unterscheiden. Der Grund davon ist,

dass es für alle Planer, die COC nutzen können, auch möglich sein soll, CAB zu nutzen. Deshalb ist eine ähnliche Anordnung der Buttons und Felder sowie ähnliches Design wesentlich.

2.3 Erfassen der Anforderungen

Bei der Anforderungsermittlung geht es darum, die passenden Anforderungen von den Stakeholdern zu ermitteln. In diesem Projekt werden die Anforderungen größtenteils von den Kabinenplanerinnen gestellt, dem sogenannten Fachbereich.
Herausforderungen bei der Ermittlung sind unterschiedliche und häufig wechselnde Anforderungen. Die Stakeholder können sie oft selber nicht genau benennen und ausdrücken, sodass die Wünsche sich zu widersprechen scheinen oder durch neue Ideen und Vorschläge ändern.Ein anderer problematischer Teil ist das Sprachverständnis während der Anforderungsermittlung. Der “Fachbereich [kennt] in den seltensten Fällen die Fachbegriffe des Entwicklers [.] und umgekehrt [...]. Somit ist eine Art ’Übersetzungsprozess’ zwischen der Sprache des Fachbereichs und der des Entwicklers notwendig“ (Alpar u. a., 2016, S.319) . Für den Auftragnehmer bedeutet das eine ständige Hinterfragung aller Details und Fachbegriffe sowie die Absprache jeder Kleinigkeit mit den Stakeholdern, sodass sie vollständig ihren Anforderungen entsprechen.

2.4 Mehrfachqualifikationen der Kabinenbesatzung

Die Zuteilung jeder Person zu einer PU ist in COC eindeutig. Im Gegensatz dazu kann die Kabinenbesatzung mehreren Flotten zugeordnet sein. Jedes Kabinenmitglied wird genau einer KG zugeteilt. Dabei kann es bis zu drei, aber auch weniger Flugzeugtypen haben, für die es qualifiziert ist (vgl. Berg u. a., 2017). Diese Mehrfachqualifikationen müssen sinnvoll in das Programm mit aufgenommen und strukturiert werden, sodass der Algorithmus von COC darauf angewendet werden kann.

3 Grundlagen/Methodischer Ansatz

3.1 Der Software-Lebenszyklus

Software Projekte lassen sich in einzelne Phasen unterteilen. Diese „Phasen, die ein Softwareprodukt bei seiner Herstellung und dem späteren Einsatz durchläuft“ (Brich u. Hasenbalg, 2013, S.173) nennt man den Software-Lebenszyklus eines Produktes.

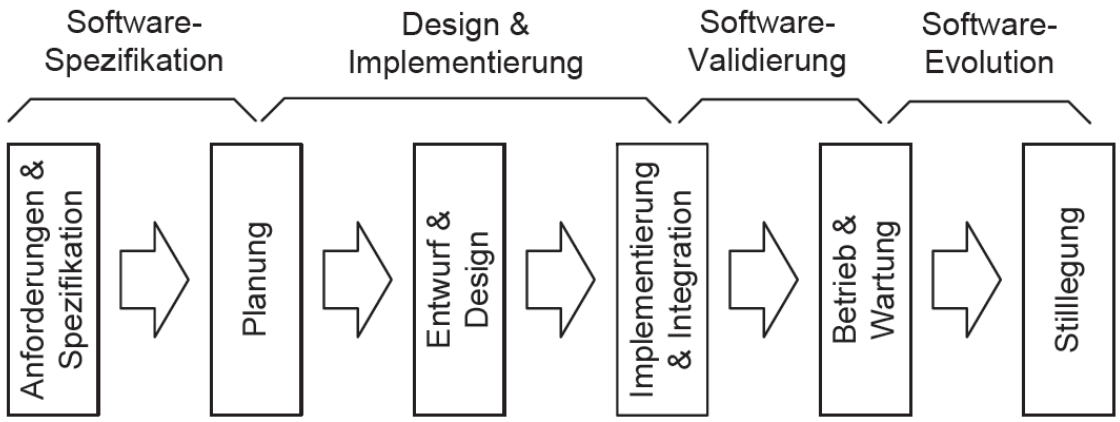

Abbildung 1: Lebenszyklus eines Softwareprojekts
(Schatten u. a., 2010, S.13)

In Abbildung 1 lassen sich die wesentlichen Schritte des Software-Lebenszyklus erkennen. Diese sind hier entweder grob in vier Stufen oder differenzierter in sechs Stufen unterteilt. Es gibt keine Vorschriften über die Anzahl oder die Bezeichnungen der Abstufungen, sodass es sehr viele unterschiedliche Versionen des Software-Lebenszyklus und damit Vorgehensmodelle für die Softwareentwicklung gibt. Grundlegende Dinge, die hier durch *Software-Spezifikation*, *Design und Implementierung*, *Software-Validierung* und S*oftware-Evolution* dargestellt werden, „finden sich [jedoch] nahezu in allen Projekten wieder“ (Schatten u. a., 2010, S.13).
Detaillierter lässt sich der Software-Lebenszyklus in die sechs Phasen *Anforderung und Spezifikation*, *Planung*, *Entwurf und Design*, *Implementierung und Integration*, *Betrieb und Wartung* und *Stilllegung* aufteilen.
Die Einteilung eines Softwareprojekts in Phasen „kann den Entwicklerteams als Leitlinie dienen, um ein Software-Projekt im Team erfolgreich zu strukturieren und abzuwickeln“ (Schatten u. a., 2010, S.11). Dabei können die Phasen je nach Vorgehensmodell variieren oder wahrgenommen werden.

3.2 Vorgehen bei der Beschreibung und Analyse der Anforderungen

Die Beschreibung und Analyse von Anforderungen ist der erste Schritt bei fast jedem IT-Projekt. Zunächst wird „aus der in einer systematischen Ermittlung gewonnenen Information [.] bei der Dokumentation eine präzise Anforderungsspezifikation erstellt“ (Partsch, 2010, S.44). Das Ziel dabei ist, „möglichst vollständige Kundenanforderungen in guter Qualität zu dokumentieren und dabei Fehler möglichst frühzeitig zu erkennen und zu beheben“ (Pohl u. Rupp, 2015, S.11). Für so eine Anforderungsanalyse gibt es unterschiedliche Methoden, die jedoch alle „vor allem gesunden Menschenverstand voraus[setzen]“ (Partsch, 2010, S.58).

Quelle zur Ermittlung der Anforderungen sind Dokumente, bereits existierende Systeme und hauptsächlich die sog. Stakeholder. Diese sind „Person[en] oder Organisation[en], die (direkt oder indirekt) Einfluss auf die Anforderungen ha[ben]“(Pohl u. Rupp, 2015, S.21). Gemeint sind damit also alle Menschen, die in irgendeiner Weise mit der Software zu tun haben oder haben werden, z.B. der Kunde, der Nutzer, die Entwickler etc.

Da der Auftraggeber oft nicht in der Lage ist genau auszudrücken, was er will und braucht, ist es die Aufgabe der Entwickler „in Software-Projekten frühzeitig mit dem Kunden und den späteren Nutzern [zu] reden, um zu erfahren, was sie sich vorstellen“ (Kleuker, 2016, S.15).

Es muss also eine ständige Kommunikation und Zusammenarbeit sichergestellt werden. So eine Zusammenarbeit setzt „ein gewisses Verständnis der Arbeitsabläufe des Kunden, genauer dessen Geschäftsprozesse (vgl. Gadatsch, 2010), [voraus], die mit der zu entwickelnden Software im Zusammenhang stehen“ (Kleuker, 2016, S.15).

3.3 Qualitätssicherung der Anforderungen

„Grundsätzlich gilt, dass die Produkte aller Tätigkeiten bei der Softwareentwicklung [...] qualitätsgesichert [...] werden müssen“ (Winter, 1999, S.55). Deshalb „ist es notwendig, die Qualität der entwickelnden Anforderungen zu überprüfen“(Pohl u. Rupp, 2015, S.95), um Probleme der Anforderungsermittlung (siehe Kapitel 2.3) zu lösen.

Diese ständige Qualitätssicherung dient dazu, Fehler möglichst früh in dem Softwarelebenszyklus zu erkennen und zu beheben. Der Grund dafür ist, dass die Fehlerbehebung in der Softwareentwicklung mit steigendem Projektfortschritt mehr Aufwand verursacht (vgl. Hußmann, 2001, S.2). Aufwand kann dabei Geld, Zeit oder auch andere zu erbrin-

gende Leistung oder Einsatz sein. Das Ziel ist daher, Fehler schon möglichst früh in dem Projektverlauf zu beseitigen.

Dieser Aufwand der Fehlerbehebung ist der Grund für die Qualitätssicherung der Anforderungsermittlung und dadurch ist sie so wichtig für den Erfolg eines Projektes. Diese wichtige Rolle wird in dem folgenden Balkendiagramm dargestellt:

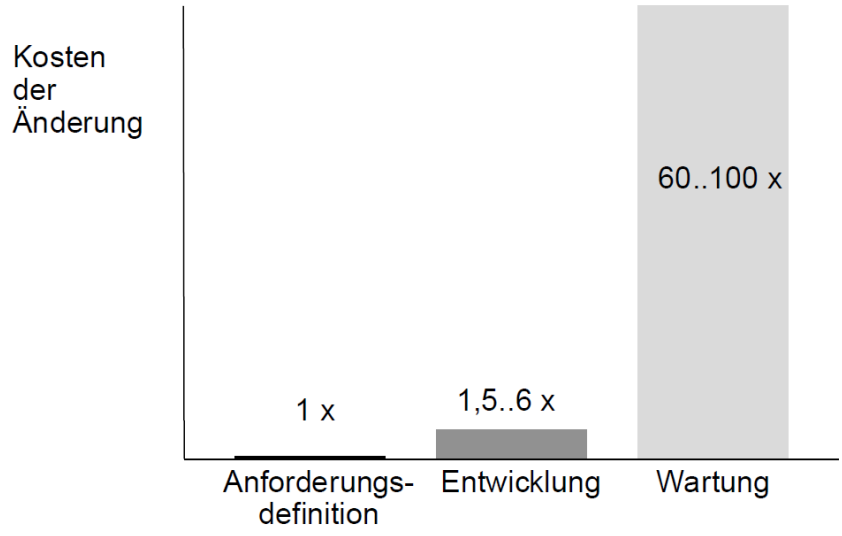

Abbildung 2: Aufwand der Fehlerbehebung in Softwareprojekten
(Hußmann, 2001, S.2)

Das Diagramm beschreibt den Aufwand der Fehlerbehebung in einem Softwareprojekt in Abhängigkeit von dem Projektfortschritt. Der Fortschritt wird dabei als Phasen des Softwarelebenszyklus dargestellt, wobei dieser vereinfacht nur in Anforderungsdefinition, Entwicklung und Wartung aufgeteilt ist.

Wird ein Fehler in der Phase der Anforderungsermittlung oder am Anfang des Projektes behoben, wird der Aufwand dafür mit dem Faktor Eins multipliziert. Sobald ein Fehler gefunden wurde, gilt es die Anforderung zu überarbeiten und ihn einfach zu korrigieren. Während der Entwicklung des Projektes ist der Aufwand meist um das 1,5 bis 6-fache höher. Grund dafür ist, dass Fehler, die sich dann noch in der Software befinden, oft mehrere Bereiche betreffen und viel verändert und berücksichtigt werden muss bevor man sie entgültig beheben kann.
Am meisten Aufwand verursacht die Fehlerbehebung, falls der Fehler erst am Ende des Software-Lebenszyklus entdeckt wird. Dabei ist die Software schon in Betrieb und befin-

det sich in der Wartung. Um dort einen Fehler zu beheben, müssen oft ganze Programmabschnitte verändert werden, damit die Software wieder fehlerfrei läuft. Es hat also große Folgen, wenn ein Teil des Programmes während der Wartung verändert werden muss, weshalb der Aufwand etwa 60 bis 100 mal höher ist als zu Anfang des Projekts.
Deshalb ist es so wichtig, die einzelnen Phasen eines Software-Lebenszyklus sorgfältig zu bearbeiten und zu überprüfen, sodass keine Fehler auftreten oder diese schon so früh wie möglich erkannt werden. Es sollte daher nicht zu früh und unvorbereitet mit der Entwicklung begonnen werden.
Obwohl es Ausnahmen gibt, lohnt sich der Aufwand in der Analyse meistens, da „viele schwerwiegende Fehler in den frühen Phasen IS-Entwicklung [Informationssystem-Entwicklung; Anmerk. d. Verf.] gemacht werden“ (Alpar u. a., 2016, S.316). Trotzdem wird in vielen Fällen zu voreilig mit der Entwicklung angefangen.
"Die Analyse der in einer Anforderungsdefinition festgelegten Anforderungen zielt letztlich darauf ab, Aufschluss über die Qualität der Anforderungsbeschreibung zu erhalten“ (Partsch, 2010, S.51). Kriterien für die Qualität lassen sich hauptsächlich in Inhalt, Dokumentation und Abgestimmtheit gliedern. Es wird sich also damit befasst, ob die Anforderungen vollständig, detailliert, passend dokumentiert und mit allen Stakeholdern abgestimmt sind. Für jeden der drei Qualitätsaspekte gibt es damit unterschiedliche Prüfkriterien (vgl. Pohl u. Rupp, 2015, S.97).

3.3.1 Qualitätsaspekt Inhalt

„Der Qualitätsaspekt »Inhalt« bezieht sich auf die Überprüfung von Anforderungen auf inhaltliche Fehler“ (Pohl u. Rupp, 2015, S.98). Dafür gibt es acht Prüfkriterien(vgl. Pohl u. Rupp, 2015, S.98):

Vollständigkeit: Wurden alle relevanten Anforderungen erfasst?

Korrektheit: Beschreibt jede Anforderung die dafür notwendigen Informationen?

Verfolgbarkeit: Können die Anforderungen verfolgt, also z.B. auf die Quelle zurückgeführt werden?

Adäquatheit: Enthalten die Anforderungen die Bedürfnisse und Wünsche der Stakeholder angemessen?

Konsistenz: Gibt es Widersprüche zwischen den Anforderungen?

Vorzeitige Entwurfsentscheidungen: Wurden Entwurfentscheidungen vorweggenommen, die nicht durch Randbedingungen bestimmt sind?

Überprüfbarkeit: Können Abnahme- und Prüfkriterien anhand der Anforderungen definiert werden?

Notwendigkeit: Trägt jede Anforderung zu dem definierten Ziel bei?

3.3.2 Qualitätsaspekt Dokumentation

Bei der Dokumentation geht es darum, die „Anforderungen auf Mängel in der Dokumentation bzw. auf Verstöße gegen geltende Dokumentationsvorschriften“(Pohl u. Rupp, 2015, S.99) zu überprüfen. Hierbei gibt es folgende vier Prüfkriterien(vgl. Pohl u. Rupp, 2015, S.99f.):

Konformität: Wurden die Anforderungen in dem vorgeschriebenen Dokumentationsformat strukturiert und in der richtigen Modellierungssprache dokumentiert?

Verständlichkeit: Können die Anforderungen in dem gegebenen Kontext ggf. mithilfe eines Glossars verstanden werden?

Eindeutigkeit: Ist eine eindeutige Interpretation möglich?

Konfirmität mit Dokumentationsregeln: Sind vorgegebene Dokumentationsregeln und -richtlinien eingehalten worden?

3.3.3 Qualitätsaspekt Abgestimmtheit

Die Abgestimmtheit stellt sicher, dass keine “Mängel in der Abstimmung der Anforderungen unter relevanten Stakeholdern“(Pohl u. Rupp, 2015, S.100) vorliegen. Auch hierbei gibt es folgende drei Prüfkriterien(vgl. Pohl u. Rupp, 2015, S.100) :

Abstimmung: Wurde jede Anforderung mit relevanten Stakeholdern abgestimmt?

Abstimmung nach Änderungen: Wurde jede Änderung der Anforderungen auch abgestimmt?

Konflikte: Wurden alle bekannten Konflikte gelöst?

3.4 Anforderungskategorisierung

Anforderungen werden kategorisiert und nach Wichtigkeit eingeteilt. Das ist hilfreich, da die Anforderungen unterschiedlich zur Zufriedenheit der Stakeholder beitragen (Pohl

u. Rupp, 2015, vgl. S.24). Durch Kategorisierung lassen sie sich leichter einordnen, um Aufwand und Priorität besser abschätzen zu können.
In diesem Fall werden die Anforderungen nach dem Kano-Modell kategorisiert. Demnach gibt es drei Kategorien (Pohl u. Rupp, 2015, vgl. S.24):

Basisfaktoren sind selbstverständliche unterbewusste Systemmerkmale, die vorausgesetzt werden.

Leistungsfaktoren sind bewusste, explizit geforderte Systemmerkmale.

Begeisterungsfaktoren sind für den Stakeholder unbekannte Systemmerkmale, die er während der Benutzung als angenehme Überraschung entdeckt.

Mit der Zeit werden aus Begeisterungsfaktoren Leistungsfaktoren und aus Leistungsfaktoren Basisfaktoren, da der Nutzer sich an die Merkmale gewöhnt und sie irgendwann voraussetzt. Das Modell lässt sich grafisch darstellen:

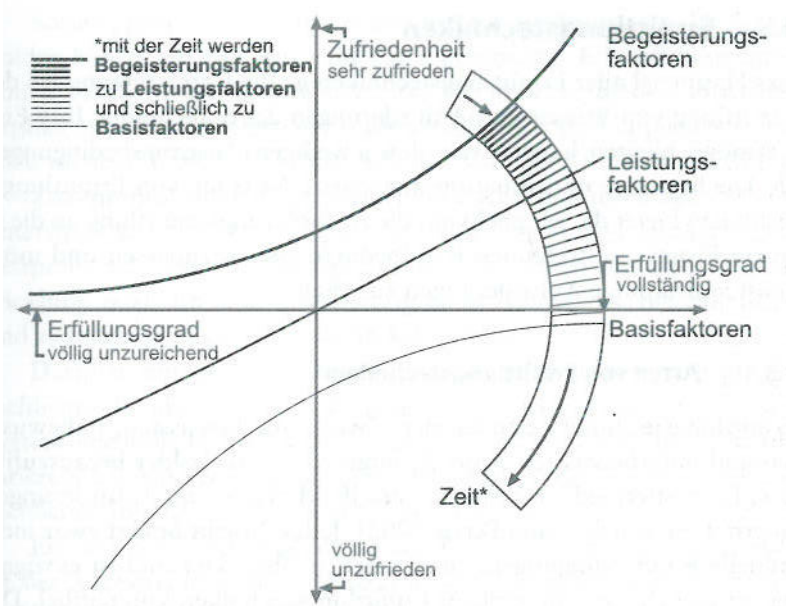

Abbildung 3: Grafische Darstellung des Kano-Modells
(Pohl u. Rupp, 2015, S.25)

Die Grafik zeigt die Zufriedenheit der Stakeholder in Abhängigkeit von dem Erfüllungsgrad der jeweiligen Faktoren. Auf einer dritten Achse wird die Zeit dargestellt, wodurch die Anforderungen die Kategorien wechseln können.
Die Basisfaktoren müssen erfüllt sein, um eine massive Unzufriedenheit der Stakeholder zu vermeiden (Zhu u. a., 2010, vgl. S.106). Sie werden vorrausgesetzt und haben daher

nur den Nutzen, Unzufriedenheit zu vermeiden. Daher verläuft die Funtkion negativ exponentiell.
Die Funktion der Leistungsfaktoren verläuft linear und schneidet den Nullpunkt. Diese Faktoren erzeugen proportional Unzufriedenheit, wenn sie nicht erfüllt sind, und Zufriedenheit, wenn sie erfüllt sind (Zhu u. a., 2010, vgl. S.106).
Da die Begeisterungsfaktoren vorher den Stakeholdern nicht bekannt sind, kann ihr unzureichender Erfüllungsgrad keine Unzufriedenheit verursachen, da sie nicht erwartet werden oder bekannt sind (Zhu u. a., 2010, vgl. S.106). Das Vorhandensein dieser Faktoren freut jedoch die Stakeholder, wodurch es zu einem überproportionalen Nutzen bei steigender Erfüllung kommt. Deshalb ist die Funktion der Begeisterungsfaktoren linear und erreicht niemals den Wert null oder weniger, aber steigt sehr schnell an.

Es ist ungeklärt, ob man diese Faktoren und ihre Eigenschaften so auf die Anforderungen bei der Erstellung eines Prototyps anwenden kann. Laut Pombergers Definition soll ein explorativer Prototyp ermöglichen, die Vorstellungen der Anwender „anhand von Anwendungsbeispielen zu prüfen und die gewünschte Funktionalität zu ermitteln“ (Pomberger u. Pree, 2004, S.27), es soll also die Analyse und die Anforderungsspezifikation unterstützen. Das bedeutet, dass der Prototyp nicht alle Anforderungen enthalten kann, besonders die Leistungs- und Begeisterungsfaktoren nicht, und genau im Gegenteil dazu dient, Anforderungen zu ermitteln.
Kanos Begeisterungsfaktoren widersprechen zudem dem weit verbreiteten Vorgehen bei einer erfolgreichen Anforderungsanalyse. Wie von Pohl und Rupp beschrieben ist dafür „eine gute Kommunikation und [...] Qualität der Zusammenarbeit mit den Stakeholdern“ (Pohl u. Rupp, 2015, S.33) notwendig, um ihn „erfolgreich in den Ermittlungsprozess einzubinden“ (Pohl u. Rupp, 2015, S.34). Durch diese enge Zusammenarbeit werden alle Anforderungen abgesprochen, sodass es keine Begeisterungsfaktoren mehr geben kann.

3.5 Abgrenzung des Systems und Systemkontextes

Die Abgrenzung eines Systemes zu seiner umgebung und zu den Schnittstellen ist essentiell für den Entwurf eines korrekten Produktes. Dafür müssen die System- und Kontextgrenzen bestimmt werden.
Dadurch wird Aufwand vermieden, indem bereits vorhandene Funktionen evtl. integriert oder nur in etwas abgeänderter Weise übernommen werden. Außerdem bringt sie Klarheit

über die Arten der Informationsgewinnung und -weiterverarbeitung, um Missverständnisse zu vermeiden. Es ist wichtig, „die Grenzen des Systems zum Systemkontext und die Grenzen des Systemkontexts zur irrelevanten Umgebung zu bestimmen“ (Pohl u. Rupp, 2015, S.20), da der Systemkontext „auch die Anforderungen an das zu entwickelnde System bestimmt“ (Pohl u. Rupp, 2015, S.20).
Der praktische Teil zu diesem Kapitel fällt weg, da diese Arbeit COC als Grundlage nutzt und Systemgrenzen und -schnittstellen daher schon klar definiert sind.

3.6 Arten und Ziele des Prototypings

Prototyping beschreibt „eine Vorgehensweise bei der Softwareentwicklung, bei der nicht sofort ein endgültiges Softwaresystem, sondern zunächst ein oder mehrere Prototypen erstellt werden“ (Brich u. Hasenbalg, 2013, S.152). Dabei wird nach Kuhrmann zwischen horizontalem und vertikalem Prototyping unterschieden (vgl. Kuhrmann, 2012):
Horizontales Prototyping bezieht sich auf einen bestimmten Bereich der Software, z.B. die Benutzeroberfläche. Der Bezug zu der technischen Funktionalität und der tatsächlichen Implementierung ist dabei nicht gegeben. Nur die eine Ebene des Programms soll vorgestellt werden.
Beim **vertikalen Prototyping** wird ein Ausschnitt komplett in allen Ebenen vollständig implementiert. Diese Art dient zur Demonstration von komplexer Funktionalität.

Darüber hinaus unterscheidet die IEEE das Prototyping anhand von Anwendungszwecken (vgl. Lichter u. a., 1994, S.826):

Exploratives Prototyping wird benutzt, wenn man das Problem und die Anforderungen noch nicht genau kennt. Indem viele Ideen und Ansätze ausprobiert werden, lernt der Entwickler die Arbeit und die Anforderungen des Kunden besser kennen, um die Anforderungen zu identifizieren.

Experimentelles Prototyping zielt auf das Sammeln von Erfahrungen und Ideen. Der Anwender kann mit dem Prototypen experimentieren, um Ideen an das Programm weiterzuentwickeln und Anforderungen zu konkretisieren. Währenddessen enthält der Entwickler Eindrücke von der Realisierbarkeit des Systems und technischen Herausforderungen.

Evolutionäres Prototyping stellt die Softwareentwicklung nicht als ein temporäres Projekt, sondern als eine fortlaufende Entwicklung dar. Dabei wird das Programm nach und nach erweitert, wobei der Entwickler eng an der Seite des Anwenders das System immer weiter erweitert.

Die Definition des evolutionären Prototypings widerspricht jedoch der allgemeinen Definition nach Brich u. Hasenbalg (2013). Das Ergebnis des explorativen und des experimentellen Prototypings ist ein Prototyp im engeren Sinne, der der Demonstration dient. Dieser wird nach der Erfüllung seiner Aufgaben nicht mehr benötigt (vgl. Liggesmeyer, 2012, S.21). Beim evolutionären Prototyping hingegen entsteht ein Pilotsystem, das der Kern des Produktes ist. Der Prototyp wird also fließend zum laufenden Produkt (vgl. Liggesmeyer, 2012, S.24) und ist damit eigentlich kein Prototyp laut Definition mehr.

4 Istanalyse

4.1 Bestandsrechnung in Compas Cockpit

Der aktuelle Algorithmus für die Bestandsrechnung in COC ermittelt zuerst den Bruttobestand der Besatzungen (vgl. DLH, 1995, S.8). Dazu wird der aktuelle Pilotenbestand mit allen Formen von Zu- und Abgängen z.B. durch Umschulungen, Altersabgänge oder Neueinstellungen und Kündigungen sowie allen bezahlungswirksamen Fehlzeiten wie Teilzeit, Mutterschutz oder Fluguntauglichkeit errechnet (vgl. LSY, 2001, S.19). Insgesamt gibt es 16 Kategorien, die hier aber der Übersicht halber nicht alle genannt werden.
Von diesem Bruttobestand werden alle weiteren Fehlzeiten, die sich grob in Urlaub und Krankheit einteilen lassen, subtrahiert, was zu dem Nettobestand führt (vgl. DLH, 1995, S.8).
Das Ergebnis aller verfügbaren Piloten ergibt sich dann durch die Subtraktion von freien Tagen, die den Piloten zustehen. Auch die Arten und die Berechnung der freien Tage werden hier nicht weiter erläutert.
Dadurch ergibt sich der Bestand aller verfügbaren Piloten in Beschäftigungstage (BT). Diese BT lassen sich auf Tages-, Wochen- und Monatsbasis darstellen und zusammenfassen.

4.2 Anforderungsanalyse- und kategorisierung

Die Planerinnen formulierten folgende Anforderungen (Berg u. a., 2017, vgl.):

Basisfaktoren: Es soll ein Programm erstellt werden, dass die Kapazitäts- und später auch die Schulungsplanung automatisch regelt. Dafür müssen vor allem die Gruppen und Untergruppen eingeteilt und dargestellt werden können. Der Zeitraum soll 15 Monate betragen, um die Urlaubsplanung des jeweils kommenden Jahres mit einfließen lassen zu können. Eine Bestandsrechnung wird also benötigt, die später um die Bedarfsrechnung ergänzt werden soll. Das Programm wird dann nach und nach erweitert bis es in etwa den Funktionalitäten von COC entspricht.

Der Kunde legt außerdem viel Wert auf Nachvollziehbarkeit. Es soll bei jeder Zelle der Tabellen verstanden werden können, woher die Daten her kommen oder wie sie berechnet werden. Dazu soll es wie bei COC die Möglichkeit geben, Reports für einzelne bestimmte Daten abfragen zu können.

Leistungsfaktoren: Da es sich um den Entwurf eines Prototyps zur Kapazitätsplanung handelt, ist die Schulungsplanung ein Leistungsfaktor. Dazu gehören auch Bewerbungen für Schulungen. Ein weiterer Leistungsfaktor ist die Bedarfsrechnung. Diese Leistungsfaktoren werden aber später ergänzt, sodass sie nicht Bestandeil dieses Lösungsentwurfs sind.

Darüber hinaus wird eine gute Performance erwartet, um die Planung zu erleichtern und Zeit zu sparen.

Begeisterungsfaktoren: Modernes Design ist einer der Begeisterungsfaktoren. Dabei ist wünschenswert, das Programm so zu designen, dass sich Cockpit- und Kabinenplaner/innen ersetzen können und die Bedienung und Anordnung der Elemente gleich ist.

Da sich diese Arbeit aber nur mit der Entwicklung eines Lösungsentwurfs für den Prototypen beschäftigt, stehen die Basisfaktoren im Vordergrund. Die Leistungs- und Begeisterungsfaktoren sind für einen explorativen Prototypen erst einmal nur optional. Sie kommen im Verlauf des Projektes dazu. Trotzdem ist eine Bestimmung davon wichtig, um das System abzugrenzen und Schwerpunkte festzulegen.

4.3 Die Schnittstelle: Das Crew Management System

Die Quelle, aus der COC die benötigten Daten entnimmt, ist das Crew Management System (CMS) oder genauer gesagt die CDB. Unter dem CMS wird das gesamte Planungssystem der Lufthansa verstanden, das aus vielen verschiedenen Systemen besteht. Im Mittelpunkt davon steht die CDB, wo die Daten zusammen fließen und in der Datenbank gespeichert werden. Es wird also von sehr vielen Seiten aus auf die CDB zugegriffen, was ihr sehr viel Leistung und Performance abverlangt.

Damit COC die passenden Daten in der passenden Form erhalten kann, formatiert (spiegelt) die CDB diese Daten wöchentlich, indem sie sie aus allen anderen Tabellen anderer Systeme herausfiltert und dann eine View erstellt, die der benötigten Form entspricht. Das Erstellen dieser View verursacht dabei sehr viel Aufwand und Auslastung der CDB.

Wenn über COC die aktuellen Daten dann angefordert werden, werden sie in die eigene Compas Datenbank geladen, wo direkt drauf zugegriffen werden kann. Das Aktualisieren der Tabellen aus der View nimmt jedoch nicht viel Zeit in Anspruch, da die Daten nur geladen und nicht verändert oder formatiert werden müssen.

5 Konkrete Modernisierung und Anpassung von Compas Cockpit

5.1 Veränderungen bei der Datenspiegelung

(folgt) - Problem mit personenbezogenen Daten bei Idee das auf COMPAS Seite zu verlagern
- Idee: Verschlüsselung zu CRM-ID -> KG Bildung, Umformen CRM-ID -> Originaldaten nur bei Anfordern
- anderer Algorithmus? nur falls es muss

5.2 Veränderungen in der Bestandsrechnung

In der in Kapitel 4.1 beschriebenen Bestandsrechnung von COC müssen einige Zeilen angepasst werden, damit sie der bisherigen Bestandsrechnung der Kabine entspricht. Diese Anpassung wurde mit dem Kunden genau besprochen (vgl. Winkel u. a., 2017).
Einige Zeilen wie „Verlängerer Teilzeit“ oder „Alterssonderurlaub“ fallen weg, da so etwas bei der Kabinenbesatzung nicht vorkommen kann. Eine Änderung wird für das Feld „Umschulungen“ benötigt, die in COC einfach verrechnet wurden. In CAB ist es notwendig, zwischen Schulungen, die kürzer als 5 Tage, und welchen, die mindestens 5 Tage dauern zu unterscheiden, weshalb zwei Zeilen für die beiden Arten der Schulungen benötigt werden.
Auch das Feld „FO-Leistung gesamt“ muss angepasst werden. Die Vorgehensweise in COC ist, dass Umschüler bereits zu Beginn einer Schulung auf eine neue PU zu dieser PU zählen und die Flugstunden während der Schulung schon als FO-Leistung zu dem verfügbaren Bestand addiert werden. Bei der Kabinenbesatzung zählt eine Person erst zu der neuen KG, wenn sie die Schulung beendet hat, leistet aber während der Schulung auch schon Flugstunden für die neue KG.
Bei der restlichen Bestandsrechnung kann genau so vorgegangen werden, wie es bereits in COC für die Cockpit-Mitarbeiter implementiert ist.

5.3 Prämissen der Bestandsrechnung

5.4 Abbilden der Mehrfachqualifikationen und Einbindung der Kleingruppen

Nach Absprache des Problems der Mehrfachqualifikationen mit dem Fachbereich haben sich die Anforderungen geändert (vgl. Wagner u. a., 2017a):
Die Anwender haben eine Liste von PUs erstellt, die wie in COC aufgebaut sind, mit dem Unterschied, dass die PUs mehrere Muster besitzen können. Sie stellen es sich so vor, dass die fest eingeteilten KGs jeweils manuell einer PU vom Nutzer zugeordnet werden können. Diese Zuteilung muss natürlich veränderbar und flexibel sein.
Deshalb muss anstatt einer direkten Verbindung zwischen einer Person und einer PU ein Zwischenschritt mit den KGs eingebaut werden. Die personalisierte tagesgenaue View enthält also eine Referenz mit der KG, in der sich die Person befindet. Danach ist eine neue Tabelle notwendig, wo die KGs den PUs zugeordnet werden, die vom Anwender bearbeitet werden kann. Eine KG hat damit Bezug zu maximal einer PU.

6 Zusammenfassung und Ausblick

Literatur

[Alpar u. a. 2016] ALPAR, Paul ; ALT, Rainer ; BENSBERG, Frank ; GROB, Heinz L. ; WEIMANN, Peter ; WINTER, Robert: *Anwendungsorientierte Wirtschaftsinformatik : Strategische Planung, Entwicklung und Nutzung von Informationssystemen*. 8.Auflage. Wiesbaden : Springer Vieweg, 2016 (477152740). – ISBN 978–3–658–14146–2

[Berg u. a. 2017] BERG, Stefanie ; WAGNER, Vanessa J. ; GARSKE, Julian: *Erstes Gespräch mit den Planerinnen über die aktuelle Vorgehensweise*. Frankfurt am Main, 24. Mai 2017

[Berger 2016] BERGER, Iwan: *Anwendungsdokumentation gemäß CMS-BetrVbg; COMPAS - Kapazitätsplanung Cockpit*. Version 1.20. Raunheim: Lufthansa Systems GmbH & Co.KG, 4. Mai 2016

[Brich u. Hasenbalg 2013] BRICH, Stefanie ; HASENBALG, Claudia: *Kompakt-Lexikon Wirtschaftsinformatik : 1.500 Begriffe nachschlagen, verstehen, anwenden*. Wiesbaden : Springer Gabler, 2013. – ISBN 978–3–658–03028–5

[DLH 1995] DLH: *Funktionale Beschreibung Kapazitätsplanung Cockpit*. Version 0.1 - Entwurf. Frankfurt am Main: Deutsche Lufthansa AG, 20. März 1995

[Gadatsch 2010] GADATSCH, Andreas: *Grundkurs Geschäftsprozess-Management*. Wiesbaden : Vieweg + Teubner, 2010. – ISBN 978–3–8348–9346–8

[Hentschel u. a. 2017] HENTSCHEL, Norman ; RAPP, Dominik ; GARSKE, Julian: *Gespräch über die Datenbank-Schnittstelle*. Raunheim, 21. Juni 2017

[Hußmann 2001] HUSSMANN, Heinrich: *Anforderungsermittlung; Vorlesungsskript in „Softwaretechnologie 2“*. Dresden: Technische Universität Dresden, 2001. – URL: `http://st.inf.tu-dresden.de/Lehre/WS00-01/st2/vorlesung/st2k2a-extra.pdf`; abgerufen am 05.07.2017

[Kleuker 2016] KLEUKER, Stephan: *Grundkurs Datenbankentwicklung - Von der Anforderungsanalyse zur komplexen Datenbankanfrage*. 4. Auflage. Wiesbaden : Springer Vieweg, 2016. – ISBN 978–3–658–12338–3

[Kuhrmann 2012] KUHRMANN, Marco: Prototyping. In: *Enzyklopädie der Wirtschaftsinformatik, Online-Lexikon*. München : Hrsg.: Norbet Gronau, Jörg Becker,

Elmar J. Sinz, Leena Suhl und Jan Marco Leimeister, September 2012. – URL: http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Vorgehensmodell/Prototyping/index.html; abgerufen am 26.06.2017

[Lichter u. a. 1994] LICHTER, Horst ; SCHNEIDER-HUFSCHMIDT, Matthias ; ZÜLLIGHOVEN, Heinz: Prototyping in Industrial Software Projects-Bridging the Gap Between Theory and Practice. In: *IEEE Transactions on Software Engineering* Vol. 20 (1994), S. 825–832

[Liggesmeyer 2012] LIGGESMEYER, Peter: *Software Entwicklung 2 - Prozessmodelle; Vorlesungsskript*. Kaiserslautern: Technische Universität Kaiserslautern, 2012. – URL: http://agde.cs.uni-kl.de/teaching/se2/ss2012/material/vorlesung/se2_ss2012_03_Prozessmodelle.pdf; abgerufen am 27.06.2017

[LSY 2001] LSY: *Anhang zum Benutzerhandbuch - Berechnungen*. Version 1.2. Frankfurt am Main: Lufthansa Systems GmbH & Co.KG, 26. September 2001

[M2P 2016] M2P CONSULTING (Hrsg.): *Projektergebnisse der Prüfung von M2P*. Frankfurt am Main: M2P Consulting, Februar 2016

[Partsch 2010] PARTSCH, Helmuth A.: *Requirements-Engineering systematisch : Modellbildung für softwaregestützte Systeme*. 2. überarbeitete und erweiterte Auflage. Berlin, Heidelberg : Springer-Verlag, 2010. – ISBN 978–3–642–05358–0

[Pohl u. Rupp 2015] POHL, Klaus ; RUPP, Chris: *Basiswissen Requirements Engineering*. 4. Edition. Heidelberg : dpunkt.verlag, 2015. – ISBN 978–3–89864–550–8

[Pomberger u. Pree 2004] POMBERGER, Gustav ; PREE, Wolfgang: *Software Engineering: Architektur-Design und Prozessorientierung*. 3., völlig überarbeitete Auflage. Carl Hanser Verlag GmbH & Co.KG, 2004. – ISBN 978–3–446–22788–0

[Schatten u. a. 2010] SCHATTEN, Alexander ; DEMOLSKY, Markus ; WINKLER, Dietmar ; BIFFL, Stefan ; GOSTISCHA-FRANTA, Erik ; ÖSTREICHER, Thomas: *Best Practice Software-Engineering*. Heidelberg : Spektrum Akademischer Verlag, 2010. – ISBN 978–3–8274–2487–7

[Wagner u. a. 2017a] WAGNER, Vanessa J. ; WINKEL, Alfred ; GARSKE, Julian: *Gespräch über Mehrfachqualifikationen und die Datenerhebung*. Lufthansa-Basis Frankfurt am Main, 26. Juni 2017

[Wagner u. a. 2017b] WAGNER, Wolfgang ; BECKER, Torsten ; RAPP, Dominik ; GARSKE, Julian: *Erstes Gespräch über Compas Kabine*. Raunheim, 22. Mai 2017

[Winkel 2015] WINKEL, Alfred: *Anwenderkonzept Personalbestandsermittlung Kabine*. 1.2. Frankfurt am Main: Deutsche Lufthansa AG, 29. Oktober 2015

[Winkel 2017] WINKEL, Alfred: *High Level Items - Personalbestandsermittlung Kabine*. Version 1.3. Frankfurt am Main: Deutsche Lufthansa AG, 20. Februar 2017

[Winkel u. a. 2017] WINKEL, Alfred ; WAGNER, Vanessa J. ; GARSKE, Julian: *Gespräch über die detaillierte Bestandsrechnung und die Datenbank-Strukturen*. 4. Juli 2017. – Aufzeichnung: Meeting 05.07. - Bestandsrechnung detailliert.docx

[Winter 1999] WINTER, Mario: *Qualitätssicherung für objektorientierte Software: Anforderungsermittlung und Test gegen die Anforderungsspezifikation*, Fachbereich Informatik der FernUniversität - Gesamthochschule - in Hagen, Doktorarbeit, 1999

[Zhu u. a. 2010] ZHU, Dauw-Song ; LIN, Chih-Te ; TSAI, Chung-Hung ; WU, Ji-Fu: A Study on the Evaluation of Customers Satisfaction - the Perspective of Quality. In: *International Journal for Quality Research* (2010), August, S. 105–115

Glossar

Anlagenverzeichnis