Masterarbeit

Domain Engineering einer Service-Organisation

Günther Schröttner

Graz, 2011

Institute für Software Technologie
Technische Universitaet Graz

und

PIDAS Ges.m.b.H.

Betreuer TU Graz: Univ.-Prof. Dipl.-Ing. Dr. techn. Franz Wotawa

Betreuer PIDAS: Dipl.-Ing. Dr. Robert Molidor

Zusammenfassung

Die Entwicklung neuer Software beginnt oft mit der Analyse eines Problems und dem anschließenden Erfassen von Requirements. Dies passiert jedoch ohne genaues Wissen über die Domäne, in der die Software letztendlich zum Einsatz kommen soll. Um Softwareentwicklung als Ingenieurdisziplin zu sehen, ist ein professionelleres Vorgehen notwendig.

Durch den Einsatz des **Domain Engineerings** soll vor dem Beginn der eigentlichen Softwareentwicklung ein sowohl breites als auch tiefes Wissen über die betroffene Domäne erarbeitet werden. Die Informationsquellen, um dieses Wissen zu erhalten, können vielfältig sein. Zum einen stellt eine solide Literaturrecherche einen Grundstock an Wissen zur Verfügung. Weiters stellt bestehende Dokumentation, die in der Domäne vorhanden ist, Wissen zur Verfügung. Einen großen Teil des Wissens kann man auch über Personen, die innerhalb der Domäne ihrer Arbeit nachgehen, beziehen. Durch Fragebögen und Interviews werden Vorgänge und Abläufe festgehalten und analysiert. Ziel dieser Analysen ist es Entitäten, Funktionen, Ereignisse und Verhaltensweisen innerhalb der Domäne zu erfassen.

Durch dieses erworbene Wissen über die Domäne, welches in einer Domänenbeschreibung festgehalten ist, sollen im weiteren Verlauf stabile und formal verifizierbare Requirements und ein vollständiges Design der Software erstellt werden. Während des Requirements Engineerings werden relevante Teile der Domänenbeschreibung herangezogen und in Anforderungen an die neue Software umgewandelt.

Weiters kann die Domänenbeschreibung auch für andere Bereiche im Unternehmen interessant sein. Sie dient zum Kennenlernen der Domäne, beantwortet Fragen über die Domäne und stellt den Ausgangspunkt für Dokumentation dar.

Dieses Vorgehen wurde anhand der Domäne *Service-Organisation* erprobt und auch praktisch bei der Erstellung eines neuen Softwaremoduls — der sogenannten *Template-Engine* — angewandt. Der hohe Aufwand, den Domain Engineering zu Beginn eines Projektes verursacht, muss bei der Entwicklung von Individualsoftware dem weiteren Nutzen gegenübergestellt werden. Hingegen ist das Vorgehen bei Erstellung einer Software in einer neuen Domäne als sehr nützlich anzusehen.

Diese Arbeit entstand in Zusammenarbeit mit der Firma PIDAS Ges.m.b.H. die im Bereich Kundenservice auf den Aufbau und die Optimierung sowie den Betrieb von Service-Organisationen spezialisiert ist.

Abstract (English)

There are several different approaches to manage software development projects. All of them have their advantages and disadvantages. Software development is often classified as „simple“ due to the lack of a comprehensive view. That's why **Domain Engineering** should help to fill these gaps and help software development become a engineer discipline.

The primary goal of Domain Engineering is to gather knowledge about the domain the software is developed for. There are many ways how to accomplish this. The study of literature is one way. Existing documentation within the domain is another source of knowledge. But a more important factor are the people who work within the domain. Their knowledge about the domain is collected with questionnaires and interviews. It is critical to Domain Engineering to explain how this approach works and why it is used. Otherwise people don't understand what the goal of Domain Engineering is and the quality of the analysis suffers.

The domain description contains entities, functions, events and behaviours of the domain. This description is used to derive requirements for the new software. Only a subset of the domain description is used for the new software. These requirements should be formally verified.

The concept of Domain Engineering was put to test on the domain *Service-Organization*. The result was used to implement a new software module called *Template Engine*. When developing individual software one has to weigh the costs of using Domain Engineering against its further benefit in the project. But when developing a new software in an unknown domain, Domain Engineering is a good choice to get started.

This thesis is a result of cooperation with PIDAS Ges.m.b.H. who specialized in creating and optimizing as well as operating service organizations in the customer care field.

Danksagung

Die Erstellung dieser Arbeit erfolgte parallel zu meiner Vollzeitbeschäftigung bei der Firma PIDAS und beanspruchte deswegen einen Großteil meiner Freizeit. Ich bedanke mich bei allen, die dafür Verständnis aufgebracht haben — allen voran bei meinen Eltern die mich trotz meines späten Entschlusses ein Studium zu absolvieren immer unterstützt haben.

Weiters bedanke ich mich bei Univ.-Prof. Dipl.-Ing. Dr. techn. Franz Wotawa der mir bei der Erstellung und Durchführung dieser Arbeit unterstützt hat und immer ein offenes Ohr für meine Anliegen hatte.

Natürlich gilt mein Dank auch der Firma PIDAS, speziell Dr. Robert Molidor, der es mir ermöglicht hat, diese Arbeit durchzuführen.

Günther Schröttner
Graz, 2011

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Ort, Datum Unterschrift

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Place, Date Signature

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Sehr viele Projekte im Bereich Softwareentwicklung scheitern oder werden aufgrund nicht erreichter Erwartungen, Ergebnisse oder Ziele abgebrochen. Die Ursachen dafür sind vielfältig. Diese reichen von nicht ausreichend vorhandenen finanziellen Mittel, über schlechtes Projektmanagement bis hin zu falschen Herangehensweisen bei der Entwicklung der Software. Ein nicht zu unterschätzender Faktor ist auch die Akzeptanz der Software bei den Endanwendern. Eine erfolgreiche Software muss viele Aspekte bieten um akzeptiert zu werden. Die Anforderungen unterschiedliche Stakeholder sollen erfüllt werden — und dies obwohl die Anforderungen oft in entgegengesetzte Richtungen zeigen. Dies alles zu bewältigen ist ein schwer durchzuführender Balanceakt.

1.1. Motivation

Aufgrund meiner Tätigkeit in der Softwareentwicklung durfte ich an einigen Softwareprojekten mitwirken. Die Herangehensweise in den unterschiedlichen Projekten war immer sehr ähnlich. Es wurden vom Kunden Requirements eingefordert die unter Rücksprache mit den Entwicklern dann verfeinert wurden. Der Softwareentwicklungsprozess selbst wurde nach dem klassischen Wasserfallmodell durchgeführt.

Das Ergebnis des ersten Releases war in den meisten Fällen zufriedenstellend. Es folgte der Einsatz der Software und dem Kunden wurden neue Anforderungen und Änderungen bestehender Prozesse klar. Genau hier war der Punkt erreicht, wo die Software nur mehr kundengetrieben entwickelt wurde. Was zu Beginn des Projektes sauber durchdacht und dokumentiert wurde, wurde mit einem einzigen Änderungswunsch des Kunden zunichte gemacht. Die nicht vorhandene Flexibilität der Softwarearchitektur wurde vielfach zum Stolperstein der Projekte. Die Anpassungen wurden immer als kleine Patches realisiert was im Laufe der Zeit zu einer mosaikartige Software geführt hat. Die Architektur und das Gesamtkonzept wurden nicht überarbeitet und waren eigentlich nicht mehr das Rückgrat der Applikation.

Der Fokus verschob sich darauf, die Software so gut wie möglich stabil zu halten. Die meisten Aufwände flossen in Tätigkeiten um diverse Bugs zu beheben. Dies hatte wieder zur Folge, dass die Seiteneffekte durch das Ausbessern von Fehlern sehr schwer absehbar waren. Ein zeitaufwendiges

und intensives Testen musste vor jedem Release der Applikation durchgeführt werden. Das Testen erfolgte weitgehend manuell den automatisierte Tests waren nur rudimentär vorhanden.

Auch war meist die Zeit für ein gründliches Refactoring nicht gegeben. Der Zeitdruck, um die Kundenwünsche für das nächste Release umzusetzen, war zu groß. Das aufwendige Testen der Software nahm ebenfalls einen großen Anteil der Zeit für ein Release in Anspruch.

Diese nicht zufriedenstellenden Vorgehensweisen haben mich dazu veranlasst, im Zuge meiner Masterarbeit den Ansatz des Domain Engineerings anzuwenden. Durch umfangreiches Wissen über die Domäne sollte sich die Entwicklung der Software — vor allem in Bezug auf spätere Änderungswünsche — um einiges vereinfachen.

1.2. PIDAS

Diese Arbeit entstand in Zusammenarbeit mit der Firma *PIDAS Österreich Ges.m.b.H.*[*] PIDAS ist ein Dienstleistungsunternehmen, das sich im Bereich Customer Care auf den Aufbau, Betrieb und Optimierung von Service-Organisationen spezialisiert hat.

Die Abteilung *Solutions* entwickelt Software, um den Betrieb in Helpdesks oder in Customer Care Centern zu unterstützen. Die Lösungen umfassen ein Ticketingsystem zur Erfassung von Kundenanliegen, ein E-Mail-Responsemanagement System um automatisiert auf Anfragen zu antworten und diese bereits richtig zu kategorisieren und zuzuordnen, Tools zur natursprachlichen Analyse von Kundenanfragen um automatisiert Antworten auf Standardprobleme vorzuschlagen sowie Reporting- und Auswertungsmöglichkeiten zur Kontrolle und Steuerung der Helpdesks.

1.3. Aufgabenstellung

Es soll eine sogenannte Template-Engine entwickelt werden. Diese Engine soll im Customer Care Umfeld in Helpdesks unterstützend zur bereits bestehender Software zum Einsatz kommen. Sehr viele Abläufe im Helpdesk sind repetitiv. Dadurch eröffnet sich die Möglichkeit, unterstützend durch Vorlagen oder Templates die täglichen Abläufe zu erleichtern und zu beschleunigen. Auszufüllende Formulare oder Anträge, Lösungsvorschläge für Standardprobleme, Standardantworten auf E-Mail Anfragen sowie Import bzw. Export von Terminen in standardisierten Formaten sind nur einige Beispiele, die mit Hilfe der Template-Engine verwaltet werden sollen.

Durch den Einsatz des Domain Engineerings soll eine robuste und erweiterbare Software entstehen. Die zuvor beschriebenen Fallstricke in der Entwicklung sollen dadurch vermieden oder zumindest reduziert werden.

1.4. Ergebnisse

Durch Domain Engineering wurde die Domäne, in der die Template-Engine zum Einsatz kommen soll, analysiert. Das Ergebnis ist in Kapitel 5 zu finden. Aufbauend auf diesem Ergebnis wurden danach die Requirements abgeleitet welche in Kapitel 6 näher beschrieben werden.

[*] http://www.pidas.com/

1.5. Gliederung

Der erste Teil dieser Arbeit befasst sich mit der Theorie des Domain Engineerings sowie des Requirement Engineerings (Kapitel 3 und 4). Die praktischen Ergebnisse werden im Anschluss daran erläutert (Kaptiel 5 und 6). Eine Zusammenfassung und Darstellung der erworbenen Erkenntnisse ist in Kapitel 7 zu finden.

Kapitel 2

Grundlagen

Dieses Kapitel zeigt, dass die Idee des Domain Engineerings nicht neu ist. Weiters werden unterschiedliche Auffassungen des Domain Engineerings aufgezeigt.

2.1. Idee des Domain Engineerings

Die Idee des Domain Engineerings ist nicht neu. Bereits 1976 schlug David L. Parnas vor, Software im Sinne von Domain Engineering zu entwickeln. Er benutzt den Begriff *Programm-Familie* um eine Menge von Programmen zu beschreiben, die sich aufgrund ihrer Funktionalitäten unterscheiden, aber eine gemeinsame Basis oder einen gemeinsamen Kern besitzen. Um sich mit diesen Programmen vertraut zu machen, untersucht man zuerst die Gemeinsamkeiten und erst dann geht man auf die Besonderheiten der einzelnen Programme ein.

„We consider a set of programms to constitute a *family*, whenever it is worthwhile to study programs from the set by *first* studying the common properties of the set and *then* determining the special properties of the individual family members.“ Parnas (1976)

Wie in Abbildung 2.1 dargestellt, wurden neue Programme einer Programm-Familie durch die Modifikation eines alten Programms erstellt. Ein X stellt eine fertige Version eines Programms dar. Ein Kreis mit der Nummer im Inneren stellt einen Zwischenschritt (Modul) dar. Ein Kreis mit der Nummer außerhalb ist eine modifiziert Version eines funktionierenden Programms um ein neue Version zu erhalten. Pfeile stellen Designentscheidungen dar, die eine neue Version des Programms zur Folge haben. Der Nachteil dieser Vorgehensweise liegt darin, dass eine neue Version des Programms immer aus einer alten Version resultiert und dadurch Teile mitgeschleppt werden, die eigentlich gar nicht mehr benötigt werden. Weiters ist es schwierig, Bugs in den unterschiedlichen Versionen zu beheben, wenn der Bug in einem sehr frühen Stadium des Entwicklungsfortschritts auftaucht. Es müssen sämtliche abgeleiteten Versionen des Programms geändert werden.

Anstelle der Entwicklung neuer Programme durch Modifikation eines bestehenden, fertigen Programms sollen — wie in Abbildung 2.2 dargestellt — neue Programme immer von einem Zwischenschritt aus abgeleitet werden. Der Vorteil bei dieser Methode stellt die parallele Entwicklung von Programmen innerhalb einer Programm-Familie dar. Die Entscheidung, wann ein neuer Zweig erstellt wird, muss jedoch gut überlegt sein.

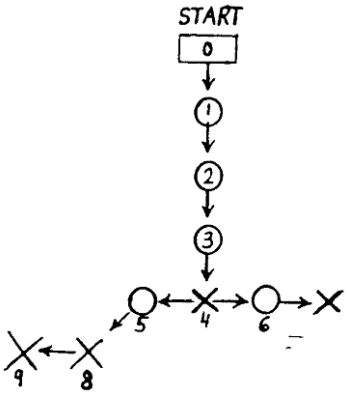

Abbildung 2.1.: Sequentielles Erstellen von Programmen einer Programm-Familie Parnas (1976)

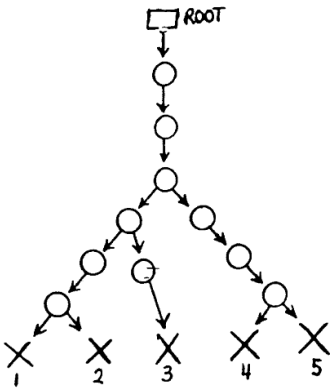

Abbildung 2.2.: Programmerstellung mit „Abstract Decisions“ Parnas (1976)

Beim Domain Engineering sollen also durch schrittweises Vorgehen und aufbauend auf einer gemeinsamen Basis an Funktionalitäten neue Programme entstehen. Das heißt, durch finden von Softwareeinheiten sollen Programme zusammengebaut werden.

2.2. Methoden des Domain Engineerings

Hier werden unterschiedliche Methoden des Domain Engineerings vorgestellt.

FODA

Feature-Oriented Domain Analysis (FODA) nutzt Merkmale (Features) von bestehenden Programmen und sucht Features die in zukünftigen Anforderungen an Programme notwendig sind. Diese gemeinsamen Merkmale stellen die Grundlage der Softwarearchitektur dar. Ergebnisse sind ein Information Model, Operation Model sowie ein Domain Dictionary. Jost (2007); Kang et al. (1990)

DARE

Domain Analysis and Reuse Environment (DARE) ist eine Analyse-Methode für Domänen. Es werden drei Quellen für die Analyse herangezogen:

1. Quellcode
2. Dokumentation
3. Expertenwissen

Das Ergebnis stellt ein sogenanntes *Domain Book* dar. Der Inhalt dieses Buchs ist eine umfassende Beschreibung der analysierten Domäne. Daraus werden dann wiederverwendbare Komponenten und eine Systemarchitekutr abgeleitet. Frakes et al. (1998)

FAST

Family-Oriented Abstraction, Specification and Translation (FAST) wird vor allem beim Product Line Engineering eingesetzt. FAST besteht aus drei Teilprozessen: Domain Qualification (DQ) versucht herauszufinden, ob eine Domäne überhaupt für das Domain Engineering im Sinne von Product Line Engineering geeignet ist. Danach folgt das eigentliche Domaine Engineering (DE). Nach Abschluss des Domain Engineerings werden die Anwendungen beim Application Engineering (AE) erstellt. Frakes and Kang (2005)

ODM

Organization Domain Modeling (ODM) nutzt ebenfalls wie FODA Merkmale bei der Definition und Analyse der Domäne. Ein Merkmal bei FODA ist aus Kunden- oder Benutzersicht definiert. ODM geht einen Schritt weiter und bezeichnet als Merkmal alles, was für beliebige Interessensgruppen (Stakeholder) interessant sein kann. Dadurch wird nicht nur die Sicht des Endanwenders oder -benutzers herangezogen, sondern ein viel umfassender Blick auf die Domäne geworfen. Simos et al. (1996); Jost (2007)

2.3. Domain Engineering nach Bjørner — Der Triptych-Ansatz

Der in dieser Arbeit verwendete Ansatz des Domain Engineerings entspricht dem von Dines Bjørner. Eine der wesentlichen Ideen von Dines Bjørner ist der Triptych-Ansatz. Seine Vorstellung von Software Engineering hält fest, dass bevor das Design einer Software entwickelt werden kann, die Requirements festgehalten werden müssen und bevor Requirements definiert werden, die Domäne bekannt sein muss:

„Before software can be designed one must understand its requirements. Before requirements can be expressed one must understand the application domain.“ Bjørner (2008b), Seite 4

Daraus ergibt sich ein dreistufiges Modell des Software Engineerings. Die einzelnen Schritte sind voneinander abhängig und müssen in der vorgegebenen Reihenfolge durchgeführt werden. Sehr viele der Softwareprojekte die heutzutage umgesetzt werden, beginnen mit dem Erfassen der Requirements. Dies wird von Bjørner als mangelhaftes Entwickeln von Software angesehen Bjørner (2009b). Das Erfassen von Requirements ohne die Domäne in der man sich bewegt zu kennen, führt zu unvollständigen und vielleicht sogar falschen Anforderungen. Wird die Software von Endanwendern benutzt die zuvor nicht befragt wurden, wird sich die Akzeptanz der neuen Software in Grenzen halten. Dazu kommt noch die natürliche Abneigungen gegenüber neuer Software und eventuell neuen Prozessen. Bilden diese Prozesse nicht die Wirklichkeit ab, wird es noch schwieriger die Akzeptanz der neuen Software zu steigern. Oft wird dann versucht, einen Kompromiss zwischen vorgegebenen Prozess und gewohntem Prozess zu finden. Diese Erfahrungen sollen durch einen umfassenden Ansatz wie dem des Domain Engineerings nicht wiederholt gemacht werden.

Der Fokus dieser Arbeit liegt im Bereich des *Domain Engineerings*. Es wird aber auch auf die weitere Vorgehensweise im Requirements Engineering eingegangen.

Kapitel 3

Domain Engineering

In diesem Kapitel werden die Grundlagen und Grundbegriffe des Domain Engineerings beschrieben. Es werden zuerst Begriffe des Domain Engineerings definiert und erläutert. Danach folgt eine Beschreibung der Vorgehensweise beim Domain Engineering laut Dines Bjørner et al. (2009).

3.1. Begriffsdefinitionen

Der Begriff der **Domäne** soll hier definiert werden:

„By a domain we shall understand a universe of discourse, small or large, a structure of entities, that is, of 'things', individuals, particulars some of which are designated as state components; of functions, say over entities, which when applied become possibly statechanging actions of the domain; of events, possibly involving entities, occurring in time and expressible as predicates over single or pairs of (before/after) states; and of behaviours, sets of possibly interrelated sequences of actions and events." Bjørner (2008b), Seite 3

Eine Domäne ist also ein abgeschlossener Bereich der sich von angrenzenden Bereichen unterscheiden lässt und möglichst keine Überlappungen mit anderen Bereichen aufweist. Diese Abgeschlossenheit einer Domäne ist eine Grundvoraussetzung für das Entwickeln einer Software nach dem Ansatz des Domain Engineerings. Auf die Komponenten einer Domäne wird in 3.2.1 näher eingegangen.

Der Begriff des **Domain Engineerings** ist wie folgt definiert:

„By domain engineering we understand the modeling of a domain: a careful description of the domain as it is, void of any reference to possibly desired new software, including requiremetns to such sofware." Dines Bjørner et al. (2009), Seite 5

Evans beschreibt einen Domain Engineer wie folgt:

„Effective domain modelers are knowledge crunchers. They take a torrent of information and probe for the relevant trickle. They try one organizing idea after another, searching for the simple view that makes sense of the mass." Evans (2004), Seite 13

3.1.1. Phenomena und Concepts

Es geht beim Domain Engineering also darum, die betroffene Domäne zu beschreiben ohne schon auf Anforderungen der zukünftigen Software einzugehen. Die Beschreibung der Domäne kann unterschiedlich erfolgen. Ein Möglichkeit besteht darin, die Domäne in einer natürlichen Sprache wie Deutsch oder Englisch zu beschreiben. Andere Möglichkeiten sind die mathematische Beschreibung der Domäne oder die Wahl einer abstrakten, formalen Beschreibungsform. Das Ergebnis soll eine umfassende Darstellung der Domäne sein.

Generell sollen *Erscheinungen* (Bjørner benutzt in seiner Literatur den englischen Begriff **Phenomena**) der Domäne beschrieben werden. Eine Erscheinung ist eine konkrete Instanz während unter einem *Konzept* (im englischen als **Concept** bezeichnet) die abstrakte Definition einer Erscheinung verstanden wird:

„By a phenomenon we shall loosely understand some physical thing that humans can sense or which natural science based technology can measure, and where a phenomenon is typically a natural thing or a human-made artifact.“ Bjørner (2006), Seite 122

„By a concept we shall loosely understand a mental construction conceived by people which, in an abstract manner captures an essence of usually a class of phenomena or a class of concepts.“ Bjørner (2006), Seite 122

Konzepte werden noch weiter in konkrete Konzepte und abstrakte Konzepte unterteilt. Eine nähere Beschreibung ist in Bjørner (2006) zu finden.

Die Erscheinungen oder Beobachtungen in einer Domäne werden in vier Typen unterteilt: Entitäten (**Entity**), Funktionen (**Function**), Ereignisse (**Events**) sowie Verhaltensweisen (**Behaviour**) die in der Domäne auftreten. Anhand dieser Fakten soll es möglich sein, relevante und wichtige Fragen bezüglich der Domäne zu beantworten. Sämtliche Erscheinungen in der Domäne sollen mit Hilfe dieser vier Elemente beschrieben werden können.

3.2. Ergebnisse des Domain Engineerings

Das Ergebnis vom Domain Engineering soll eine Domänenbeschreibung sein deren Bestandteile und Inhalt hier dargestellt wird.

3.2.1. Domänenbeschreibung

Laut Bjørner hat die Beschreibung der Domänen vier grundlegende Elemente: Entitäten, Funktionen, Ereignisse und Verhaltensweisen der Domäne. Auf den Zweck dieser vier Elemente wird hier eingegangen.

Entitäten

Entitäten stellen den Grundstock einer Domäne dar. Mit den Entitäten wird innerhalb der Domäne gearbeitet. Entitäten können einen Status haben, der sich durch Einflüsse von außen ändern kann. Vergleichbar sind Entitäten etwa mit einer Klasse in der objektorientierten Programmierung oder einer Tabelle in einer Datenbank.

Es wird zwischen atomaren Entitäten und zusammengesetzten Entitäten unterschieden. Während atomare Entitäten nur als gesamte Entität Sinn machen, kann bei zusammengesetzten Entitäten die Entität in mehrere Sub-Entitäten zerlegt werden. Bjørner (2006)

Eine Entität hat beschreibende Attribute. Diese Attribute haben Werte die die Eigenschaften der Entität widerspiegeln.

Funktionen

Funktionen werden auf Entitäten angewandt. Sie können Eingabeparameter und Ausgabeparameter haben. Funktionen können den Zustand einer Entität ändern.

Ereignisse

Ereignisse lösen Aktionen aus die wiederum Funktionen aufrufen oder andere Aktionen anstoßen. Eine Aktion ist ein Vorgang, der den Status einer Entität ändert.

Verhaltensweisen

Verhaltensweisen einer Domäne ist die Kombination von Ereignissen und Funktionen. Durch ein gewisses Verhalten können Ergebnisse innerhalb einer Domäne festgelegt werden.

3.2.2. Softwareeinheiten

Durch die umfassenden Beschreibung einer Domäne sollen dann kleine Softwareeinheiten entwickelt werden. Durch die Kombination dieser Einheiten werden neue Programme entwickelt. Das Domain Engineering bietet die Möglichkeit auch auf zukünftige Anforderungen flexibel und schnell reagieren zu können. Weiters bieten klar abgeschlossene Module eine bessere Testbarkeit.

3.3. Vorgehensweise beim Domain Engineering

Dieser Abschnitt beschreibt die Vorgehensweise und einzelnen Phasen beim Domain Engineering. Es werden wichtige Konzepte und beteiligte Personen am Domain Engineering vorgestellt. Abbildung 3.1 gibt einen Überblick über die einzelnen Bereiche welche in diesem Kapitel näher beschrieben werden.

3.3.1. Project Information

Wenn ein Projekt mit Hilfe des Domain Engineerings gestartet wird, gibt es dafür mehrere Gründe. In der ersten Phase sollen diese Gründe zusammengefasst und niedergeschrieben werden. Es soll dargelegt werden, warum das Projekt ins Leben gerufen wurde und notwendig ist. Weiters sind diverse Rahmenbedingungen zu definieren. Zu diesen Bedingungen zählen: Bjørner (2008b)

- Projektname

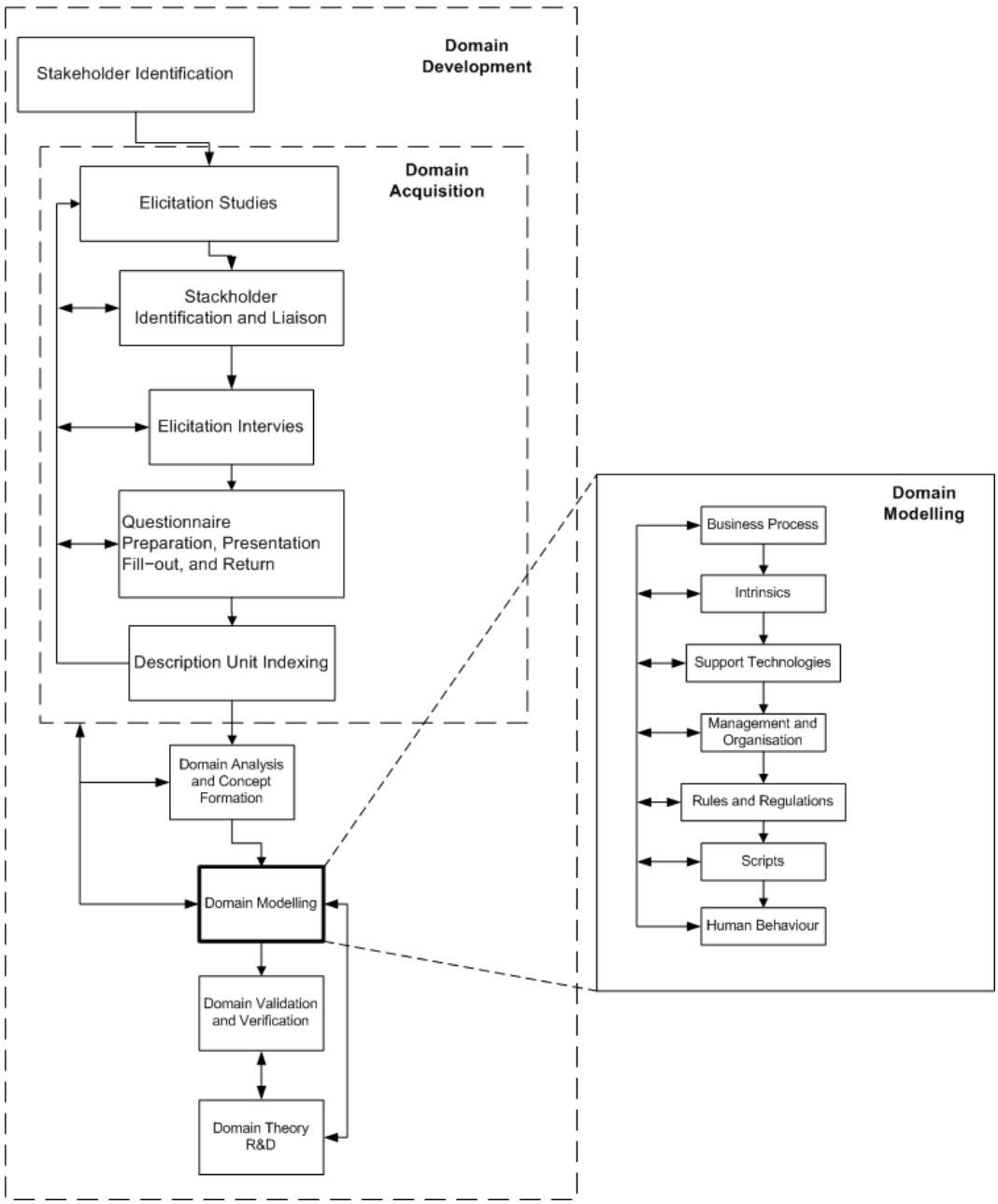

Abbildung 3.1.: Process Graph des Domain Engineerings nach Bjørner (2008b)

- Zeitlicher Rahmen des Projekts
- Projektpartner
- Ort der Projektdurchführung
- Warum wird das Projekt durchgeführt?
- Worum geht es in dem Projekt?
- Wie wird der Projektfortschritt verfolgt?

3.3.2. Domain Stakeholders

Ein Stakeholder ist jemand, der direkten oder indirekten Einfluss auf die Domäne oder Anforderungen der Software hat. Dies können natürliche und juristische Personen sein sowie auch Gruppen von Personen (zum Beispiel „Manager“). Ein weiteres Kennzeichen ist das gemeinsame Interesse an der Domäne oder eine Abhängigkeit von der Domäne. Rupp (2007); Bjørner (2008b)

Bjørner unterscheidet zwischen *Turnkey-Software* und *Commercial Off-the-Shelf-Software* (COTS). Turnkey-Software wird speziell für eine Domäne entwickelt. Es besteht ein Vertrag zwischen Auftraggeber und Entwicklung. Im Gegensatz wird bei COTS Software die Funktionalität von den Entwicklern festgelegt und dann versucht diese Software abzusetzen.

Diese Unterscheidung der Softwareentwicklung spiegelt sich auch bei den Stakeholdern wieder. Ein Turnkey-Software Stakeholder ist ein Stakeholder aus der Domäne. Die Anzahl an Stakeholdern lässt sich ermitteln. Im Gegensatz dazu ist die Anzahl der COTS Stakeholder nicht einfach zu ermitteln. Sie können in Gruppen eingeteilt werden und es sollen die Interessen dieser Gruppen abgebildet werden. Bjørner (2006)

Turnkey-Software Stakeholder sind Wissensträger der Domäne. Das Wissen über eine Domäne, welches sich der Domain Engineer aneigenen muss, bezieht er zum Teil von diversen Stakeholdern oder Stakeholder-Gruppen. Weiters werden die Stakeholder benötigt, um eine spätere Beschreibung der Domäne zu validieren und dem Domain-Engineer Feedback zu geben.

Die Stakeholder existieren auf unterschiedlichen Ebenen der Domäne. Jeder Stakeholder trägt abhängig von seiner Position unterschiedliche Fakten und Wissen zur Beschreibung der Domäne bei. Es ist Aufgabe des Domain Engineers diese Informationen zu sammeln, zu bewerten und zu klassifizieren (siehe 3.3.4).

Die Auflistung der Stakeholder einer Domäne soll im weiteren Verlauf der Analyse einer Domäne herangezogen werden, um die entsprechenden Ansichten auch korrekt abzubilden. Auch für die Validierung und Verifizierung der Domäne (3.3.8 und 3.3.9) werden die Stakeholder wiederum benötigt.

3.3.3. Domain Acquisition

Der Erwerb von Wissen über die Domäne stellt einen wesentlichen Aspekt des Domain Engineerings dar. Jegliche Form der Wissensaneignung (Dokumente, Bücher, Internet, Interviews, etc.) ist dafür geeignet. Ziel ist es, ein möglichst tiefes Wissen und Verständnis über die betrachtete Domäne zu erhalten (siehe 3.2.1). Der Prozess der Wissensaneignung sollte iterativ ablaufen. Neue Erkenntnisse

und Fakten werden klassifiziert und müssen immer wieder mit diversen Stakeholdern überprüft, bewertet und validiert werden. Dadurch soll ein sehr realitätsnahes Bild der Domäne in den Köpfen der Domain Engineers entstehen. Bjørner (2008b)

Die erarbeiteten Wissensteile der Domäne werden in sogenannten *Domain Description Units* zusammengefasst:

„By a domain description unit we shall mean an as far as possible well-formed sentence, something which names and describes some entity, function, event or behaviour of the domain, that is, something expressible which 'makes sense', that is, which can contribute to the modelling of an entity, a function, an event or a behaviour." Bjørner (2008b), Seite 59

Attribute, die jeder Domain Description Unit zugeordnet sein sollen sind:

- Name
- Art: Entität, Funktion, Ereigniss oder Verhalten
- Quelle: Name des Stakeholder, Domain Engineer
- Datum: Erstes Auftreten der Beschreibungs-Unit sowie Aktualisierungen oder neue Revisionen

3.3.4. Domain Analysis and Concept Formation

Die Analyse der Domäne erfolgt durch Sichtung aller Domain Description Units. Ziel ist es, die Domain Description Units frei von Inkonsistenzen und Widersprüchen zu bekommen und eine zufriedenstellende Vollständigkeit zu erreichen. Dabei kann es natürlich zu Konflikten zwischen einzelnen Domain Description Units kommen. Die Konflikte müssen durch Zusammenarbeit von Domain Engineer und Stakeholder gelöst werden. Ist das nicht möglich, so wird eine übergeordnete Instanz benötigt (zum Beispiel das Management) welche den Konflikt klärt. Bjørner (2008b)

Neben der Analyse der Domain Description Units wird in dieser Phase auch die Erstellung eines Domain Konzepts angestrebt:

„By a domain concept we mean a concept, an abstraction, a mental construction, which captures all essential properties and 'suppresses' expression of properties deemed not essential." Bjørner (2008b), Seite 61

3.3.5. Domain and Business Processes

In dieser Phase sollen alle wichtigen Geschäftsprozesse dargestellt werden. Der Zweck ist es zu überprüfen, ob die bisher gefundenen Entitäten, Funktionen und Ereignisse vollständig sind und damit die Geschäftsprozesse abgebildet werden können. Werden Lücken in den Domain Description Units gefunden, müssen diese durch Abklärung mit dem entsprchenden Stakeholder geschlossen werden. Diese Phase läuft wieder iterativ ab bis die definierten Geschäftsprozesse abgebildet werden können. Bjørner (2008b)

3.3.6. Domain Terminology

Eine gemeinsame Sprache zwischen Stakeholdern, Domain Engineers und Entwicklern zu finden, ist für das erfolgreiche Abschließen eines Projektes äußerst wichtig. Die zu definierenden Begriffe

können atomar sein oder aus mehreren atomaren Begriffen zusammengesetzt werden. Der Begriff selbst sowie die Erklärung des Begriffs müssen von allen beteiligten anerkannt und abgenommen werden. Somit ist sichergestellt, dass bei weiterer Verwendung der Begriffe ein einheitliches Bild vorhanden ist. Bjørner (2008b)

3.3.7. Domain Modeling

Beim Domain Modeling wird das gesammelte Wissen über die Domäne in ein Modell umgewandelt. Ein Modell stellt ein vereinfachtes Abbild der Wirklichkeit dar. Es sollen klare Abgrenzungen innerhalb der Domäne sowie Strukturen gefunden werden. Die Beschreibung des Modells soll sowohl in natürlicher Sprache als auch in einer formalen Sprache angefertigt werden. Die Domäne kann durch unterschiedliche Betrachtungsweisen modelliert werden. Laut Dines Bjørner et al. (2009) soll eine Domäne nach folgenden sechs Betrachtungswinkeln beschrieben werden:

1. Intrinsics
2. Support Technologies
3. Management & Organisation
4. Rules & Regulations
5. Scripts
6. Human Behaviour

Diese unterschiedlichen Betrachtungswinkel werden auch aus Domain-Facet bezeichnet:

„By a domain facet, we shall understand one amongst a finite set of generic ways of analysing a domain: a view of the domain, such that the different facets cover conceptually different views, and such that these views together cover the domain.“ Boca et al. (2009), Seite 11

Intrinsics

Intrinsics stellen die wesentlichen Elemente einer Domäne dar. Diese Element machen die Domäne aus und geben ihr Sinn. Intrinsische Elemente sind das Rückgrat oder die Essenz der Domäne. Es werden grundlegende Entitäten, Funktionen, Ereignisse und Verahltensweisen beschrieben, ohne die die Domäne nicht vorstellbar oder begreifbar wäre. Die Beschreibung selbst soll sowohl in einer natürlichen Sprache sowie auch formal erfolgen.

Support Technologies

Unter Support Technologies sollen Technologien beschrieben werden, die das Zusammenspielen von Entitäten und Funktionen genauer beschreiben. Jeglicher Einsatz von Technologie, die dem Verständnis der Domäne dienlich ist, soll in natürliche Sprache aber auch in formaler Sprache festgehalten werden.

Management and Organisation

Oft werden mit dem Begriff Management sehr ausschweifenden und „weiche“ Beschreibungen in Verbindung gebracht. Für das Domain Engineering sind jedoch „harte“ Fakten notwendig, wenn eine formale Beschreibung verlangt wird. Es gilt jetzt herauszufinden, wie die Beschreibung des Managements auf eine für das Domain Engineering notwendige Basis gebracht werden kann.

Grundsätzlich soll der Punkt Management die strategische Ausrichtung der Domäne beschrieben. Weiters fallen im Zusammenhang mit Management Begriffe wie Mission, Vision, Ziele, Planung, Kontrolle, Führung, Motivation und viele mehr. Für das Domain Engineering müssen die wichtigsten Eigenschaften, die für die Domäne unerlässlich sind, und vor allem oft zum besseren Verständnis beitragen, herausgefiltert und festgehalten werden.

Die Organisation hingegen kann wiederum leichter dargestellt werden. Ein einfaches Organigramm gibt eine gute Übersicht über die Aufstellung einzelner Personen oder Gruppen innerhalb einer Domäne. Durch die Verknüpfung von Management und Organisation kann ein gewisser Nachrichten- oder Anweisungsfluss dargestellt werden.

Rules and Regulations

In einer Domäne herrschen gewisse Regeln und Vorschriften. Diese können explizit festgehalten sein (durch das Management) oder auch implizit vorherrschen. Die Regeln geben vor, wie die Arbeitsabläufe innerhalb einer Domäne erfolgen:

„By a domain rule we mean some text which prescribes how people or equipment are expected to behave when dispatching their duty, respectively when performing their functions.“ Bjørner (2008b), Seite 81

Vorschrift hingegen kommen zum Tragen, wenn Regeln gebrochen werden. Sie stellen eine Art Plan dar, wie im Ausnahmefall zu reagieren ist:

„By a domain regulation we mean some text which prescribes what remedial actions are to be taken when it is decided that a rule has not been followed according to its intention.“ Bjørner (2008b), Seite 81

Regeln und Vorschriften sollen nicht komplett in die Domänenbeschreibung übernommen werden. Der Zweck ist vielmehr, dass Funktionen und Entitäten, die bisher noch nicht festgehalten wurde, entdeckt und in die Domänenbeschreibung mit aufgenommen werden.

Scripts

Folgende Definition beschreibt den Begriff *Scripts* für das Konzept des Domain Engineerings:

„By a domain script we mean the structured wording of a rule or a regulation that has legally binding power, that is, which may be contested in a court of law.“ Bjørner (2008b), Seite 84

Es sind hier Verträge, Lizenzen oder sonstige Schriftstücke gemeint, deren Nichteinhaltung eingeklagt werden kann. Es gilt auch hier, dass bisher nicht bekannte Funktionen oder Entitäten gefunden werden und in die Domänenbeschreibung mit einfließen.

Human Behaviour

Der Punkt Human Behaviour soll laut Bjørner (2008b) die Ausführung der Regeln und Vorschriften von verschiedenen Menschen beschreiben. Die Art, ihre Arbeit zu verrichten, ist von Mensch zu Mensch verschieden. Dies hängt einerseits von der Interpretation der Regeln ab, andererseits von der Einstellung des Menschen selbst. Einige führen ihre Arbeit sehr gewissenhaft und genau durch. Andere wiederum erfüllen nur das Minimum, um die vorgegebene Regel einzuhalten. Wieder andere arbeiten ungenau, schlampig und beachten die Regeln so gut wie gar nicht. Regeln lassen Spielraum für Interpretationen. Darum ist die Durchführung und Beachtung einer Regel sehr unterschiedlich. Dieses dann auch noch zu beschreiben stellt eine gewisse Herausforderung an den Domain Engineer dar.

Es sollen auch hier wieder neue Entitäten und Funktionen gefunden werden, die bisher in der Beschreibung der Domäne nicht vorhanden waren. Dies soll dabei helfen, Randbedingungen und Ereignisse, die so nicht vorhersehbar sind, zu finden.

3.3.8. Domain Validation

„Domain validation is about getting the right domain. Not a domain description which describes entities, functions, events and behaviours that were not intended, but intrinsics, support technologies, meanagement & organisation, rules & regulations, scripts and human behaviours which indeed characterise the domain in question.“ Dines Bjørner et al. (2009), Seite 16

Das Ergebnis aus dem Domain Modeling (siehe 3.3.7) soll nun validiert werden. Das bedeutet sicherzustellen, dass die richtige Domäne beschrieben wurde. Die Validierung wird mit ausgewählten Stakeholdern durchgeführt. Punkt für Punkt muss jede Entität, Funktion, jedes Ereignis und jedes Verhalten der Domäne durchgegangen werden. Es muss eine Einigung zwischen dem Domain Engineer und den Stakeholdern für jeden Aspekt der Domäne gefunden werden. Bjørner (2008b).

3.3.9. Domain Verification

„Domain verification is about getting the domain right. Not a domain description with mistakes, errors and inconsistencies, but a domain description that is consistent and relative complete. “ Dines Bjørner et al. (2009), Seite 16

Eine Verifikation der Domäne kann dann durchgeführt werden, wenn eine formale Beschreibung der Domäne vorhanden ist. Zum Beispiel kann der Einsatz eines Proof Checkers die Richtigkeit und logische Konsistenz der Domäne sicherstellen. Dines Bjørner et al. (2009)

3.3.10. Domain Theory

„A domain theory is a theory, one of whose models is a particularly identified, abstracted or instantiated doamin. Typically a domain theory is based on a domain model — expressed in some one or more formal specification languages — with the axioms and inference rules being of those languages, an the (zero, one or more) theorems being (further) statements about properties of the domain model.“ Bjørner (2006), Seite 352

Eine Theorie über die untersuchte Domäne zu erstellen, soll folgende Zwecke erfüllen: Bjørner (2006)

- Verstehen der Domäne
- Zum Nachdenken über die Domäne anregen und Inspiration sein
- Zum Lernen der Domäne
- Annahmen über die Domäne bestätigen
- Implementieren der Domäne in der gewünschten Form

Kapitel 4

Requirements Engineering

Laut dem Triptych Ansatz (siehe 2.3) ist es notwendig, vor dem Erfassen der Requirements die Domäne in der man sich bewegt zu kennen. Nur so kann sichergestellt sein, dass ein fundamentales Wissen über die zu erstellende Software vorhanden ist. Dieser Wissensaufbau über die Domäne soll nun beim Ableiten der Requirements von den Domain Description Units hilfreich sein. Es wird angestrebt, nicht starr von einem Requirement zum nächsten vorzugehen. Das Domain Engineering zielt darauf ab, ein wenig über den Tellerrand hinauszublicken und die zu erstellenden Software etwas aus der Ferne zu betrachten um ein ganzheitliches Bild zu bekommen. Für die Requirements bedeutet das, dass Zusammenhänge und Auswirkungen sichtbar werden, die sonst erst viel später — vielleicht erst nach dem ersten Release der Software — erkannt werden. Unnötige Änderungen und Anpassungen der Software werden damit vermieden. Dabei soll die *Goldene Regel* nicht vergessen werden:

„Prescribe only those requirements that can be objectively shown to hold for the designed software.“ Bjørner (2006), Seite 367

Es sollen nur solche Requirements erstellt werden, die auch auf irgendeine Art und Weise auf ihre Erfüllung hin überprüft werden können (zum Beispiel durch automatisiert Tests oder mit Hilfe eines Model-Checkers).

Folgende Bereiche werden beim Requirements Engineering unterschieden: Bjørner (2009a)

- Business Process Reengineering
- Domain Requirements
- Interface Requirements
- Machine Requirements

Es wird im folgenden genauer auf die Erstellung der *Domain Requirements* eingegangen. Die Punkte *Interface Requirements* und *Machine Requirements* werden nur grob umrissen, da sie nicht Umfang dieser Arbeit sind. Entsprechende Literaturhinweise werden in den einzelnen Abschnitten aufgelistet.

4.1. Business Process Reengineering

Wenn neue Software zum Einsatz kommen soll, hat dies sehr oft Auswirkungen auf die Geschäftsprozesse. Der Wunsch oder die Notwendigkeit nach neuer Software kommt meist von der Organisation selbst, die mit den Prozessen, so wie sie im Moment ablaufen, nicht zufrieden ist. Business Process Reengineering ist immer mit Veränderungen verbunden und ist integraler Bestandteil wenn neue Computersystem zum Einsatz kommen sollen:

„One way or the other, business process reengineering is an integral component in deploying new computing systems." Bjørner (2006), Seite 406

Um die Prozesse systematisch anzupassen, werden die sechs Betrachtungswinkel der Domäne (siehe Kapitel 3.3.7) untersucht. Die Untersuchungen sollen ergeben, ob eine Änderung an den Inhalten der Betrachtungswinkeln notwendig ist (und somit eine Prozessänderung durchgeführt werden muss) oder nicht.

4.2. Domain Requirements

Domain Requirements stellen Anforderungen an die Domäne selbst dar. Diese Anforderungen können nur mit Hilfe von Konzepten aus der Domäne erstellt werden. Die Domain Requirements werden von den Domain Description Units abgeleitet. Dies kann nicht automatisiert erfolgen sondern wird unter Zuhilfenahme der entsprechenden Stakeholder durchgeführt. Um eine Domain Description Unit in ein Domain Requirement überzuführen, werden die folgenden Operationen angewandt: Bjørner (2009a), Bjørner (2008a)

- Projection
- Determination
- Instantiation
- Extension
- Fitting

4.2.1. Projection

Unter *Projection* versteht man die Bereinigung der Domain Description Units. Im Domain Engineering wird die gesamte Domäne beschrieben, in der die zu entwickelnde Software zum Einsatz kommen soll. Diese Beschreibung ist meist sehr umfangreich und umfasst auch nicht relevante Inhalte für die neue Software. Es werden nur solche Domain Description Units als ein Requirement aufgenommen, wofür die zu erstellenden Software Unterstützung anbieten soll. Alle anderen Domain Description Units werden für die Erstellung der neuen Software nicht benötigt oder spielen dabei keine Rolle. Das Ergebnis dieses Schritts ist eine Untermenge der ursprünglichen Domain Description Units. Bjørner (2006)

Nach diesem ersten Schritt im Requirements Engineering bilden die verbleibenden Domain Description Units eine Art Außengrenze zu den übrigen Beschreibungen der Domäne. Die zu erstellende Software soll Funktionalitäten in diesem Bereich abbilden und Abläufe darin unterstützen.

4.2.2. Determination

Die Operation *Determination* folgt nach dem Schritt Projection. Es werden die angepassten Domain Description Units verändert. Die Veränderung zielt dahingehend ab, als dass die Units weniger nicht-bestimmtes oder nicht-vorhersehbares Verhalten aufweisen sollen. Vor allem Funktionen können so beschrieben sein, dass der Output nicht immer vorhersehbar ist oder nicht alle möglichen Ergebnisse festgehalten wurden. Dies soll in diesem Schritt eingegrenzt werden. Bjørner (2006)

4.2.3. Instantiation

Die Beschreibung der Domäne ist in den meisten Fällen allgemein gehalten. Zum Beispiel wurde die Domäne *Bank* beschrieben. Die Beschreibung trifft auf den Großteil der Banken zu. Bei der Erstellung der Requirements muss jedoch auf einen bestimmten Bereich eingegangen werden, in dem die Software letztendlich zum Einsatz kommt (im Beispiel zuvor ist das eine ganz bestimmte Bank, die die Software in Auftrag gegeben hat). Dieser Schritt instanziert die Domain Description Units für das Einsatzgebiet der neuen Software. Es werden die Units dahingehend verändert, dass sie auf das Einsatzgebiet zutreffen. Bjørner (2006)

4.2.4. Extension

In diesem Schritt der Erstellung der Domain Requirements werden neue Entitäten, Funktionen, Ereignisse und Verhaltensweisen erstellt, die in der ursprünglichen Domäne nicht möglich waren. Anregung dafür geben die bereits bestehenden Domain Description Units und die Bedürfnisse der Stakeholder an die neue Software. Weiters wird zwischen *echten Erweiterungen* und *vergessenen Erweiterungen* unterschieden. Das heißt, es werden auch Requirements aufgenommen, die bei der ursprünglichen Analyse der Domäne übersehen oder vergessen wurden. Bjørner (2006)

4.2.5. Fitting

Dieser Schritt ist notwendig, falls mehrere Projekte an einer Softwarelösung in der selben Domäne, einer angrenzenden Domäne oder sich teilweise überschneidenden Domänen arbeiten. Es sollen die Requirements von allen Projekten zusammengeführt werden. Es entstehen dadurch gemeinsame Requirements und Requirements die nur in den einzelnen Projekten Sinn machen. Gemeinsame Requirements werden im Design so berücksichtigt, sodass entsprechenden Softwaremodule entstehen, die in mehreren Projekten wiederverwendbar sind. Bjørner (2006)

4.3. Interface Requirements

Interface Requirements bezeichnen Anforderungen, die sowohl mit Konzepten aus der Domäne sowie auch Konzepten der *Maschine* erstellt werden. Als Maschine wird dabei das Zielsystem bezeichnet, auf dem die zu entwickelnde Software installiert wird. Zur Maschine gehört auch die gesamte Hardware aus der diese besteht:

„By machine we shall understand a, or the, combination of hardware and software that is the target for, or result of the required computing systems development.“ Bjørner (2006), Seite 369

Es soll hier nur auf die einzelnen Facetten des Interface Requirements Engineering hingewiesen werden, da eine ausführliche Behandlung dieser nicht im Umfang dieser Arbeit liegt. Es werden grundsätzlich folgende sechs Facetten unterschieden:

- Shared Data Initialisation Requirements
- Shared Data Refreshment Requirements
- Computational Data and Control Interface Requirements
- Man-machine Dialogue Requirements
- Man-machine Physiological Interface Requirements
- Machine-machine Dialogue Requirements

In Bjørner (2006) wird auf diese Facetten und deren Inhalt sowie deren Zweck genauer eingegangen.

4.4. Machine Requirements

Machine Requirements werden ausschließlich durch Konzepte der Maschine erstellt:

„By machine requirements we understand those requirements that can be expressed solely in terms of (or with prime reference to) machine concepts.“ Bjørner (2006), Seite 445

Um diese Requirements zu erfassen, werden fünf Facetten vorgeschlagen:

- Performance Requirements
- Dependability Requirements
- Maintenance Requirements
- Platform Requirements
- Documentation Requirements

Die genaue Beschreibung dieser Facetten liegt außerhalb des Umfangs dieser Arbeit. In Bjørner (2006) wird auf diese Facetten und deren Inhalt sowie deren Zweck genauer eingegangen.

Kapitel 5

Umsetzung des Domain Engineerings

Dieses Kapitel beschreibt das Vorgehen beim Domain Engineering anhand der Entwicklung der Template-Engine. Es werden die Vorteile des Domain Engineerings dargestellt aber auch die Schwierigkeiten die dieser Ansatz mit sich bringt.

5.1. Domain Information

Wie in Kapitel 3.3.1 beschrieben werden zu Beginn des Projektes einige Rahmenbedingungen festgelegt. Der Projektname, die Projektpartner, der zeitliche Rahmen des Projekts, die Gründe für die Durchführung des Projekts sowie ein grober Überblick über die nötigen Funktionalitäten des Endproduktes wurden schriftlich festgehalten. Erstellt wurde dieses Dokument von der Projektleitung seitens PIDAS. Dadurch wurden alle am Projekt beteiligten Personen auf einen einheitlichen Standpunkt gebracht.

5.2. Stakeholder Identifikation

Die Identifikation der Stakeholder war der nächste Schritt. Es haben sich folgende Personenkreise herauskristallisiert:

Auf Seite des Helpdesk sind folgende Stakeholder relevant:

- Supervisor
- Teamleiter
- Dispatcher
- Supporter
- Callagent

Auf der Kundenseite sind diese Stakeholder relevant:

- Servicemanager

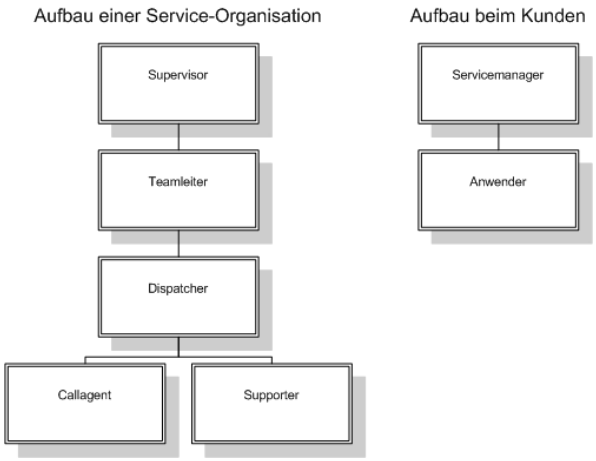

Abbildung 5.1.: Hierarchische Aufstellung der Stakeholder der Domäne Service-Organisation

- Anwender

Diese identifizierten Personengruppen (Abbildung 5.1) werden im weiteren Verlauf des Domain Engineerings ein wichtige Rolle spielen. Sie sollen mit ihrer Sicht auf die Domäne umfassende Informationen liefern um alle Bereich beim Modellieren der Domäne (siehe 3.3.7) zu erfassen.

5.3. Domain Acquisition

Das Verständnis der Domäne ist der wesentliche Punkt beim Domain Engineering. Das Vorgehen zur Wissensaneignung wird in diesem Abschnitt beschrieben. Zuvor wurde aber festgelegt, welche Domäne überhaupt untersucht werden sollte. Das Einsatzgebiet der zu entwickelnden Template-Engine wird in Service-Organisationen sein. Darum wurde die zu untersuchende Domäne auf *Service-Organisation* festgelegt.

5.3.1. Domänenübersicht

Zuerst wurde eine grobe Übersicht über die Abläufe innerhalb der Domäne erstellt. Dies soll dem Domain Engineer dabei helfen, das „Big Picture“ zu erfassen und die grundsätzlichen Zusammenhänge innerhalb der Domäne zu erkennen. Anhand dieser Übersicht können im weiteren Verlauf schon weitaus konkretere Fragen über die Domäne und über gewisse Vorgehensweisen innerhalb der Domäne gestellt werden.

Überblick über die Domäne Service-Organisation

Die Shared Service Factory (SSF), welche Teil von Managed Services ist, ist ein länderübergreifender Helpdesk. Es werden Störungsfälle von diversen Kunden mit Hilfe eines Ticketingsystems erfasst und abgearbeitet. Aufgenommen werden Störfälle über mehrere Kanäle. Der betroffene Benutzer kann

sein Anliegen per Telefon an die SSF übermitteln. Weiters besteht die Möglichkeit Probleme per E-Mail an eine definierte E-Mail-Adresse zu senden. Die dritte Variante bietet Benutzern die Möglichkeit, das Ticketingtool direkt zu benutzen und über ein Webformular eine Anfrage an die SSF zu stellen.

Unabhängig davon, welchen Kanal der betroffene Anwender gewählt hat, wird ein Ticket für sein Anliegen erstellt. Durch die Kategorisierung des Problems wird mit Hilfe eines automatisierten Routings das Ticket dem dafür zuständigen Supporter bzw. der dafür zuständigen Supportergruppe (Pool) zugewiesen. Die Abarbeitung der Tickets wird mit Hilfe eines Service-Level-Agreements (SLA), welches zwischen SSF und dem Kunden zuvor definiert wurde, überwacht.

Benachrichtigungen (Notifications) geben dem Benutzer Feedback über den Abarbeitungsstatus des Tickets. Der betroffene Benutzer bekommt diese Benachrichtigungen per E-Mail zugestellt. Der Inhalt dieser E-Mails setzt sich aus Freitext, Textbausteinen und Variablen zusammen. Die Variablen werden durch Attribute aus dem Ticket ersetzt, bevor die Benachrichtigung verschickt wird. Dieses Vorgehen bietet die Möglichkeit, dem betroffenen Anwender genaue Informationen über die Abarbeitung seines Tickets zu liefern. Die Zusammenführung von Modell (Vorlage) und Daten (Ticketinhalt) soll von der Template-Engine durchgeführt werden. Auch die Sprache des betroffenen Anwenders soll bei der Benachrichtigung berücksichtigt werden. Vorlagen in den benötigten Sprachen werden vorausgesetzt.

Für die Aufnahme von Störfällen per Telefon sind Callagents in der SSF beschäftigt. Ihre Hauptaufgabe besteht darin, das Anliegen des Benutzers möglichst genau zu erfassen. Als erster Schritt ist eine Identifizierung des Benutzers notwendig. Ist der Benutzer nicht im System erfasst, wird das Ticket für einen Dummy-Benutzer aufgenommen. Danach wird das Anliegen des Benutzers freitextlich erfasst. Da es immer wieder gewisse Standardprobleme gibt, ist hier der Einsatz der Template-Engine vorgesehen. Vorlagen, die bereits Standardtexte (eventuell mit Lücken, die vom Callagent auszufüllen sind) enthalten, erhöhen die Produktivität der Callagents.

Zusätzlich wird das Ticket noch vom Callagent kategorisiert. Es stehen drei unterschiedliche Kategoriebäume zur Verfügung. Abhängig vom Kunden, müssen ein, zwei oder alle drei Kategorien ausgewählt werden (Auswirkungen auf das Reporting). Auch hier kann die Template-Engine Hilfestellung leisten. Die definierten Vorlagen können bereits gültige Kategorien hinterlegt haben. Durch die Auswahl der richtigen Vorlage ist dann das Ticket bereits ordnungsgemäß kategorisiert.

Ist es dem Callagent möglich das Problem direkt am Telefon zu lösen, wird er das Ticket in den „Gelöst“-Status versetzen und das freitextliche Lösungsfeld ausfüllen. Wurde ein Template für ein Standardproblem ausgewählt, können bereits Lösungsvorschläge im Lösungsfeld des Tickets stehen. Zusätzlich bietet die Template-Engine aufgrund der Kategorisierung weitere Lösungsvorschläge an. Kann der Callagent das Ticket hingegen nicht lösen, wird er das Ticket abspeichern und somit zur weiteren Bearbeitung freigeben. Der Routingprozess im Hintergrund leitet das Ticket aufgrund der gewählten Kategorien an den richtigen Supporter oder die richtige Supportergruppe weiter.

Die Hauptaufgabe der Supporter besteht darin, Tickets die nicht sofort am Telefon gelöst werden konnten oder über einen anderen Kanal (E-Mail oder Webformular) eingelastet wurden, abzuarbeiten. Jeder Supporter hat Einblick in seinen Ticketpool und nimmt sich noch nicht gelöste Tickets aus dem Pool um diese zu erledigen.

Das Erledigen eines Tickets kann für einen Supporter unterschiedlichste Tätigkeiten aufweisen. Diese reichen von Bestellung neuer Hardware bis hin zur Installation von Programmen für den Benutzer. Vorlagen können wiederum genutzt werden, um Standardanliegen der Anwender zu konkretisieren. Oft sind die Beschreibungen der Benutzerwünsche nicht klar oder unvollständig formuliert.

Durch die Wahl einer entsprechenden Vorlage kann hier Klarheit geschaffen werden. Auch Bestellvorgänge und der Kontakt mit Lieferanten kann durch die Template-Engine vereinheitlicht werden. Bestellformulare oder Anträge können in der Template-Engine abgelegt werden. Die in der Vorlage enthaltenen Variablen werden durch Daten aus dem Ticket ersetzt (zum Beispiel PC-Name des Anwenders, E-Mail-Adresse des Anwenders, Name und Adresse des Supporters für Hardwarebestellungen, und viele mehr). Somit ist es zum Beispiel möglich, das Bestellformular in der Template-Engine zu erfassen und als PDF-Ausdruck dann an den Lieferanten zu übermitteln.

Da es oft vorkommt, dass Dritte die Abarbeitung von Tickets verzögern, besteht für den Supporter die Möglichkeit, die Überwachung durch das SLA auszusetzen. Dadurch wird eine Eskalation des Anliegens verhindert.

Auch das Kontaktieren des betroffenen Anwenders um nähere Einzelheiten abzuklären, gehört zu den Standardaufgaben des Supporters. Die Kontaktaufnahme mit dem Endanwender kann über Telefon oder per E-Mail erfolgen. Standardfragen müssen durch die Verwendung der Template-Engine nicht immer neu formuliert werden. Eine durch die Kategorien des Tickets eingeschränkte Sicht auf die Vorlagen erlaubt eine rasche Auswahl. Auch hier werden wieder Variablen in der Vorlage (zum Beispiel in der Anrede oder in der Grußformel) durch Daten aus dem Ticket ersetzt. Die Sprache des Anwenders wird ebenfalls berücksichtigt. Antworten auf Rückfrage-E-Mails werden automatisch an das bestehende Ticket angehängt. Nähere Auskünfte durch ein Telefonat müssen händisch vom Supporter im Ticket ergänzt werden. Hier können Templates dabei helfen, die richtigen Fragen für den Supporter vor dem Telefonat aufzubereiten. Lücken im Text werden dann noch während oder nach dem Gespräch vom Supporter ausgefüllt.

Die kontrollierende Instanz des operativen Helpdesks stellt der Dispatcher dar. Zu seinen Aufgaben zählen unter anderem das Verteilen von noch nicht erledigten Störfällen an die Supporter und die Kontrolle von bereits gelösten Anliegen. Der Dispatcher entscheidet dann, ob ein Anliegen ordnungsgemäß gelöst und kategorisiert wurde und kann dieses endgültig abschließen. Dieses Vorgehen soll eine Qualitätssicherung gewährleisten.

Tritt ein Störfall immer wieder auf, kann der Dispatcher eine Vorlage in der Template-Engine dafür erstellen. Dadurch erleichtert er es den Supportern und Callagents die richtigen Informationen zu erfassen. Auch Anwender selbst sehen diese Vorlage, wenn sie über das Webformular ein Problem melden möchten. Durch das zusätzliche Hinterlegen von Kategorien bei der Vorlage ist auch sichergestellt, dass zukünftige Tickets richtig kategorisiert sind.

Durch das Abschließen eines Tickets bekommt der betroffene Benutzer eine Benachrichtigung per E-Mail zugestellt. Dieser kann dann immer noch entscheiden, dass für ihn sein Anliegen nicht ordnungsgemäß erledigt wurde und sein Ticket reaktivieren. Dadurch beginnt die Abarbeitung wieder von vorne.

Auf der Seite des Kunden hat der Servicemanager den Überblick über alle Anfragen seiner Mitarbeiter. Er sieht vom Helpdesk abgeschlossene Tickets und kann über die Qualität des Servicedesk urteilen. Die Funktionalität der Template-Engine wird der Servicemanager sehr wenig benötigen.

Die Gegenseite im Helpdesk stellt der Supervisor dar. Seine Aufgabe besteht darin, die Arbeit am Helpdesk mit zu verfolgen und gegebenenfalls steuernd einzugreifen um die Kundenzufriedenheit sicherzustellen. Die Funktionalität der Template-Engine wird der Supervisor sehr wenig benötigen.

Ausdruck	Beschreibung
Template	Ein Text mit eventuell vorhandenen Lücken für diverse Anwendungszwecke in einem Helpdesk
Variable	Ein Platzhalter im Text, der später durch Inhalt ersetzt wird.
Ticket	Ein Kundenanliegen jeglicher Form.
Betroffener Benutzer	Der Kunden den das Problem oder die Anfrage betrifft.
Meldender Benutzer	Der Benutzer, der das Ticket am Helpdesk einlastet
Verantwortlicher Benutzer	Der Benutzer, das Ticket zur Abarbeitung zugewiesen bekommt.
Lösender Benutzer	Der Benutzer, der das Ticket gelöst hat.
Abschließender Benutzer	Der Benutzer, der das Ticket geschlossen hat.
Anwender	Siehe betroffener Benutzer
Reported By User	Siehe meldender Benutzer
Responsible User	Siehe verantwortlicher Benutzer
Issue	Siehe Ticket
Unit	Organisatorische Einheit. Abbildung der organisatorischen Hierarchie des Kunden und des Helpdesk.
Pool	Unit in der sich Tickets befinden, die noch nicht in der Abarbeitung sind.
SO	Service-Organisation. Eine organisatorische Einheit, die Serviceleistungen für einen Kunden erbringt.
SLA	Service-Level-Agreement. Eine vertragliche Vereinbarung mit dem Kunden über die Qualität der zu liefernden Services.
SL	Service-Level. Eine mit dem Kunden festgelegte Zeit, in der auf bestimmte Ereignisse innerhalb der Serviceerbringung reagiert werden muss.

Tabelle 5.1.: Gemeinsame Terminologie

5.3.2. Terminologie

In der Tabelle 5.1 werden Begriffe und ihre Beschreibung erklärt um eine gemeinsame Sprache für alle Stakeholder zu schaffen.

5.3.3. Domänenfragebogen

Nach der Identifikation der Stakeholder wurde ein Fragebogen ausgearbeitet. Dieser Fragebogen wurde an ausgewählte Personen in den Stakeholder-Gruppen verteilt. Der Aufbau des Fragebogens war an Bjørner (2006) angelehnt. Da sich die Beantwortung der Fragen als nicht ganz trivial herausgestellt hat, blieb das Feedback unter den Erwartungen zurück. Einer der genannten Gründen war die mangelnde Zeit, während der täglichen Arbeit auch noch einen umfassenden Fragebogen im gewünschten Detailierungsgrad auszufüllen. Es hat sich also als sehr wichtig herausgestellt, dass wenn der Ansatz des Domain Engineerings gewählt wird, auch das Management dahinter stehen muss und den Mitarbeitern Zeit zur Verfügung stellt, die nötigen Informationen an den Domain Engineer weiterzugeben.

Der konkrete Domänenfragebogen

Frage 1 Als erstes sollen **Entitäten** aufgelistet werden. Entitäten sind Dinge (meistens Hauptwörter innerhalb der Domäne) auf die man zeigen kann bzw. die man sonst irgendwie erfassen kann.
Bsp:Telefon, E-Mail, Ticket, Kategorie, Pool, Anwender, etc.

1. Liste so viele Entitäten wie möglich auf, die in der täglichen Arbeit oft, manchmal oder selten benötigt werden.
2. Für die aufgelisteten Entitäten beschreibe die Eigenschaften, Charakteristik und die Verwendung.

Frage 2 Als nächstes sollen **Funktionen** aufgelistet werden - Dinge die mit den Entitäten gemacht werden.
Bsp: Ticket aufnehmen, Ticket zuweisen, Beschreibung eintippen, Kategorie auswählen, Attachment hinzufügen, etc.

1. Liste so viele Funktionen wie möglich, die in der täglichen Arbeit oft, manchmal oder selten durchgeführt werden.
2. Beschreibe für jede Funktion ihre Charakteristik: Was ist die Eingabe/Voraussetzung für die Funktion und wie werden dadurch Entitäten verändert.

Frage 3 Als nächstes sollen **Verhaltensmuster** aufgelistet werden. Verhaltensmuster stellen eine Abfolge von Funktionen dar sowie auch Ereignisse, die das Verhalten auslösen.
Bsp: Ticket aufnehmen: Telefon läutet (=Ereignis), Benutzer identifizieren, Problem erfassen, Ticket speichern, etc.

1. Liste so viele Verhaltensmuster wie möglich, die in der täglichen Arbeit oft, manchmal oder selten auftreten.
2. Beschreibe die Charakteristik der Verhaltensmuster.

Frage 4 Als nächstes sollen die unter Frage 1 aufgelisteten Entitäten kategorisiert werden. Folgende Kategorien stehen zur Verfügung:

- **Intrinsisch (I)**: Absolut notwendig für die Domäne; für die Grundzüge der Domäne notwendig (zB: Ticket).
- **Support Technologie (ST)**: Unterstützende Technologien für die Arbeit mit den Entitäten (zB: Routing)
- **Management & Organisation (MO)**: Welche der Entitäten werden eher vom Teamleiter verwendet (zB: Rechte)
- **Regeln & Vorschriften (RV)**: Welche Entitäten beschreiben Regeln bzw. Vorschriften und was passierte, wenn diese Vorschriften nicht eingehalten werden (zB: SLA)
- **Menschliches Verhalten (MV)**: Entitäten, deren Qualität durch menschliches Verhalten beeinflusst werden kann. (zB: Kurzbeschreibung, Lösung)

1. Versuche jede Entität aus Frage 1 durch ein oder mehrere Kategorien zu beschreiben.
2. Sind während der Beantwortung dieser Frage neue Entitäten aufgetaucht, soll Frage 1 entsprechend ergänzt werden.

Frage 5 Als nächstes sollen die Verhaltensmuster aus Frage 3 mit näheren Details versehen werden. Speziell soll auf folgendes geachtet werden:

- Kann die Reihenfolge der Funktionen innerhalb der Verhaltensmuster verändert werden (zB: Ticketaufnahme)
- Welche Ereignisse treten auf und wie soll darauf reagiert werden (zB: Termin abgelaufen)
- Welche Personen und Informationen sind in das Verhaltensmuster involviert.

1. Für jedes Verhaltensmuster in Frage 3 sollen die zuvor genannten Details ergänzt werden.

5.3.4. Interviews

Da mit Hilfe des Domänenfragebogens nicht alle Bereiche abgedeckt werden konnten und das Feedback bei weitem unter den Erwartungen blieb, wurden zusätzlich Interviews zur Informationsbeschaffung durchgeführt. Auch hier musste wieder auf die zeitliche Verfügbarkeit der Mitarbeiter Rücksicht genommen werden. Aus diesem Grund wurde versucht, die Interviews in den normalen Entwicklungsverlauf der bestehenden Software einzubinden. In der Releaseplanung für die Ticketingsoftware wurden Anforderungen, die die zukünfitige Template-Engine betreffen, genauer untersucht. Dazu wurde mit dem Mitarbeiter, der die Anforderung eingelastet hat, ein Gespräch über seine Vorstellung der Umsetzung geführt. Dadurch konnte einerseits der normale Entwicklungszyklus der Ticketingsoftware fortgeführt werden und andererseits wertvolle Information über die Domäne gewonnen werden.

5.3.5. Geschäftsprozesse

Als wichtigster Geschäftsprozess wurde das Abarbeiten eines Kundenanliegens identifiziert. Die Eckpunkte diese Prozesses sind die Kontaktaufnahme des Kunden mit der Serviceorganisation (Eingangskanal), das Benachrichtigen des Kunden (automatisch oder manuell) sowie die Kontaktaufnahme der Service-Organisation mit dem Kunden (Ausgangskanal). Die Kontaktaufnahme der Service-Organisation mit dem Kunden erfolgt entweder per E-Mail oder per Telefon. Die bevorzugte Variante ist der Kontakt per E-Mail. Diese hat den Vorteil, dass sämtliche Interaktionen mit dem Kunden automatisch am Ticket mitprotokolliert werden. Hingegen muss bei einem Telefonat eine kurze Zusammenfassung vom Supporter/Callagent geschrieben und beim Ticket hinterlegt werden.

5.4. Erstellen von Domain Description Units

Aus den bisher gesammelten Informationen (Interviews, Fragebögen) wurden dann Domain Description Units erstellt. Die Summe der Domain Description Units legt den Betrachtungsumfang der Domäne fest. Durch diese Beschreibungseinheiten wird eine Grenze zwischen der Domäne und ihrer Umwelt gezogen. Dadurch ist für den weiteren Ablauf der Entwicklung der Template-Engine klar festgelegt, in welchem Umfeld sie zum Einsatz kommt.

Die Domain Description Units wurden vom Domain Engineer verfasst. Bei Konflikten oder Unklarheiten wurde die entsprechende Quelle konsolidiert und Unklarheiten beseitigt.

Domain Description Units

1. Es wird ein Gespräch aus dem Call-Pool übernommen.

   Namen: Gespräch, Call-Pool

   Art: Entity (Gespräch, Call-Pool), Event (Übernahme)

   Quelle: TS

   Datum: 26.04.2010

2. Für Benachrichtigungen wird einmalig ein Design (CSS) hinterlegt.

   Namen: Benachrichtigung, Design

   Art: Entity

   Quelle: TS

   Datum: 26.04.2010

3. Ein Ticket wird gedruckt (PDF) oder verschickt (E-Mail).

   Namen: Ticket, PDF, E-Mail

   Art: Entity (Ticket, PDF, E-Mail), Funktion (drucken, verschicken)

   Quelle: TS

   Datum: 26.04.2010

4. Der Inhalt einer Benachrichtigung wird in mehreren Sprachen angegeben.

   Namen: Benachrichtigung, Sprache

   Art: Entity

   Quelle: TS

   Datum: 26.04.2010

5. In einer Benachrichtigung gibt es sowohl fixen als auch variablen Text.

   Namen: Benachrichtigung, fixer Text, variabler Text

   Art: Entity

   Quelle: TS

   Datum: 26.04.2010

6. Benachrichtigungen werden in einer hierarchischen Ordnerstruktur abgelegt.

   Namen: Benachrichtigung, Hierarchie, Ordnerstruktur

   Art: Entity

   Quelle: TS

   Datum: 26.04.2010

7. Eine Vorlage wird vom Teamleiter genehmigt.

   Namen: Vorlage, Teamleiter, genehmigen

Art: Entity (Vorlage, Teamleiter), Function(genehmigen)

Quelle: TS

Datum: 26.04.2010

8. Die Variablen in einer Vorlage haben deutsche Bezeichnungen.

   Namen: Variable, Vorlage

   Art: Entity (Variable, Vorlage)

   Quelle: GS

   Datum: 26.04.2010

9. Eine Vorlage kann nur vom ASC-Support erstellt, bearbeitet und gelöscht werden.

   Namen: Vorlage, ASC-Support, erstellen, bearbeiten, löschen

   Art: Entity (Vorlage, ASC-Support), Function(erstellen, bearbeiten, löschen)

   Quelle: GS

   Datum: 26.04.2010

10. Eine Vorlage wird von einem Supporter/Callagent vorgeschlagen.

    Namen: Vorlage, Supporter/Callagent, vorschlagen

    Art: Entity (Vorlage, Supporter/Callagent), Function(vorschlagen)

    Quelle: GS

    Datum: 26.04.2010

11. Vor dem Abschicken einer E-Mail wird eine Vorschau aufgerufen.

    Namen: E-Mail, Vorschau,

    Art: Entity (E-Mail, Vorschau), Function(vorschlagen)

    Quelle: TS

    Datum: 26.04.2010

12. Verschickte E-Mails werden in der „E-Mail-History“ betrachtet.

    Namen: E-Mail, E-Mail-History, betrachten

    Art: Entity (E-Mail, E-Mail-History), Function(betrachten)

    Quelle: TS

    Datum: 26.04.2010

13. Nicht mehr benötigte Vorlagen werden aus der Ordnerstruktur gelöscht.

    Namen: Vorlagen, Ordnerstruktur, löschen

    Art: Entity (Vorlage, Ordnerstruktur), Function(löschen)

    Quelle: TS

    Datum: 26.04.2010

14. Der Pfad zur Ordnerstruktur kann pro Unit hinterlegt werden.

    Namen: Pfad, Ordnerstrukutr, Unit

    Art: Entity

    Quelle: TS

    Datum: 26.04.2010

15. Lösungen für Standardtickets werden jedesmal vom Callagent/Supporter neu eingegeben.

    Namen: Lösung, Standardticket, Callagent/Supporter

    Art: Entity

    Quelle: RM

    Datum: 27.04.2010

16. An eine Benachrichtigung werden Anhänge beigefügt.

    Namen: Benachrichtigung, Anhänge, beifügen

    Art: Entity (Benachrichtigung, Anhänge), Function (beifügen)

    Quelle: RM

    Datum: 27.04.2010

17. Anhänge werden extra hochgeladen oder kommen direkt aus dem Ticket

    Namen: Anhänge, Ticket, hochladen

    Art: Entity (Ticket, Anhänge), Function (hochladen)

    Quelle: RM

    Datum: 27.04.2010

18. Benachrichtigungen werden an einen Benutzer verschickt.

    Namen: Benachrichtigung, Benutzer, verschicken

    Art: Entity (Benachrichtigung, Benutzer), Function (verschicken)

    Quelle: RM

    Datum: 27.04.2010

19. Der Benutzer kann entscheiden, ob er Benachrichtigungen bekommen will oder nicht.

    Namen: Benachrichtigung, Benutzer, entscheiden

    Art: Entity (Benachrichtigung, Benutzer), Function (entscheiden)

    Quelle: RM

    Datum: 27.04.2010

20. Benutzer werden aufgrund von Statusänderungen des Tickets benachrichtigt.

    Namen: Benachrichtigung, Benutzer, Statusänderung, benachrichtigen

    Art: Entity (Benachrichtigung, Benutzer, Statusänderung), Function (benachrichtigen)

Quelle: RM

Datum: 27.04.2010

21. Ein Ticket muss richtig klassifiziert werden.

    Namen: Ticket, klassifizieren

    Art: Entity (Ticket), Function (klassifizieren)

    Quelle: HCW

    Datum: 03.05.2010

22. Ein Ticket kann einen Termin zur Abarbeitung hinterlegt haben.

    Namen: Ticket, Termin

    Art: Entity

    Quelle: HCW

    Datum: 03.05.2010

23. Benutzer werden manuell über den Status des Tickets benachrichtigt.

    Namen: Benutzer, Ticket, manuell benachrichtigen

    Art: Entity (Benutzer, Ticket), Function (manuell benachrichtigen)

    Quelle: HCW

    Datum: 03.05.2010

24. Benutzer wird über Aufwandschätzung informiert.

    Namen: Benutzer, Aufwandschätzung, informieren

    Art: Entity (Benutzer, Aufwandschätzung), Function (informieren)

    Quelle: HCW

    Datum: 03.05.2010

25. Benutzer antwortet auf eine Benachrichtigung

    Namen: Benutzer, Benachrichtigung, antworten

    Art: Entity (Benutzer, Benachrichtigung), Function (antworten)

    Quelle: HD

    Datum: 05.05.2010

26. Eine E-Mail wird aus der E-Mail-History heraus weitergeleitet.

    Namen: E-Mail, E-Mail-History, weiterleiten

    Art: Entity (E-Mail, E-Mail-History), Function (weiterleiten)

    Quelle: HD

    Datum: 05.05.2010

27. Eine E-Mail an den Benutzer beinhaltet Ticketdaten

Namen: E-Mail, Kunde, Ticketdaten

Art: Entity

Quelle: HD

Datum: 05.05.2010

28. Eine E-Mail an den Benutzer beinhaltet Standardtexte

    Namen: E-Mail, Kunde, Standardtext

    Art: Entity

    Quelle: HD

    Datum: 05.05.2010

29. Vorlagen werden aufgrund der Kategorisierung des Tickets ausgewählt.

    Namen: Vorlagen, Kategorisierung, Ticket, auswählen

    Art: Entity (Vorlagen, Kategorisierung, Ticket), Function (auswählen)

    Quelle: GS

    Datum: 07.05.2010

30. Der Betreff einer E-Mail an den Kunden wird aus einer vordefinierten Liste gewählt.

    Namen: Betreff, E-Mail, Kunde

    Art: Entity

    Quelle: GS

    Datum: 07.05.2010

31. Benachrichtigungen an Benutzer werden als HTML-E-Mail verschickt.

    Namen: Benachrichtigung, Benutzer, HTML-E-Mail

    Art: Entity

    Quelle: GS

    Datum: 07.05.2010

32. Die Vorschau bietet die Möglichkeit, den variablen Teil einer E-Mail aufzulösen.

    Namen: Vorschau, variabler Teil, auflösen

    Art: Entity (Vorschau, variabler Teil), Function (auflösen)

    Quelle: GS

    Datum: 07.05.2010

33. Die zur Verfügung stehenden Variablen sind über ein Hilfe-Symbol abrufbar.

    Namen: Variablen, Hilfe-Symbol

    Art: Entity

    Quelle: GS

Datum: 07.05.2010

34. Für jede Variable ist eine kurze Erklärung hinterlegt.

    Namen: Variable, Erklärung

    Art: Entity

    Quelle: GS

    Datum: 07.05.2010

35. Variablen sind nur in einer Sprache (Deutsch) verfügbar.

    Namen: Varialben, Sprache

    Art: Entity

    Quelle: GS

    Datum: 07.05.2010

36. Benachrichtigungen beinhalten das Corporate Design der Kunden.

    Namen: Benachrichtigung, Corporate Design

    Art: Entity

    Quelle: GS

    Datum: 14.05.2010

37. Ein Ticket hat einen Gesamstatus und einen Detailstatus

    Namen: Gesamtstatus, Detailstatus, Ticket

    Art: Entity

    Quelle: GS

    Datum: 14.05.2010

38. Einem Ticket könnten Attachments hinzugefügt werden.

    Namen: Ticket, Attachments

    Art: Entity

    Quelle: GS

    Datum: 14.05.2010

39. Dem aktuellen Bearbeitungsschritt des Ticket kann eine Bearbeitungszeit zugewiesen werden.

    Namen: Ticket, Bearbeitungsschritt, Bearbeitungszeit

    Art: Entity

    Quelle: GS

    Datum: 14.05.2010

5.5. Abbilden der Geschäftsprozesse

Der nächste Schritt war die Abbildung der zuvor identifizierten Geschäftsprozesse. Es wurde versucht mit den bisher ausgearbeiteten Domain Description Units diese Prozesse darzustellen. Konnte dies nicht bewerkstelligt werden, wurden die Domain Description Units erweitert.

5.6. Definition einer einheitlichen Teminologie

Bei den bisherigen Schritten zur Analyse der Domäne wurde festgestellt, dass für einen Begriff unterschiedliche Ausdrücke verwendet werden. Um auf einen gemeinsamen Nenner zu kommen, wird im Zuge des Domain Engineerings versucht eine einheitliche Terminologie zu finden. Dies erleichtert das weitere Vorgehen dahingehend, dass allen Beteiligten klar ist wie ein Begriff definiert ist und welche Bedeutung er im Zusammenhang mit der zu entwickelnden Software hat.

5.7. Modellierung der Domäne

Die Beschreibung der Domäne erfolgt unter den in Kapitel 3.3.7 beschriebenen Betrachtungswinkeln. Die Modellierung erfolgt in zwei Teilen. Zuerst wird das Modell freitextlich beschrieben. Danach werden besonders wichtige Teile der Domäne mit Hilfe der Prädikatenlogik formal festgehalten.

5.7.1. Betrachtungswinkel: Intrinsics

Das Bearbeiten von Kundenanliegen stellt die Hauptkomponente der Domäne dar. Ein Kundenanliegen wird mit einem sogenannten Ticket abgebildet. Der Inhalt des Tickets stellt alle notwendigen Informationen zur Verfügung, um das Anliegen des Kunden abzuarbeiten. Einhergehend mit dem Bearbeitungsprozess eines Tickets wird der Kunde laufend über den Fortschritt informiert. Der Kunde kann einen gewünschten Benachrichtigungskanal wählen. Der Inhalt einer Kundeninformation wird über eine Vorlage festgelegt. Diese Vorlage wird mit ticketspezifischen Daten angereichert.

Der Kunde hat verschiedenen Möglichkeiten sein Problem oder sein Anliegen an den Helpdesk heranzutragen. Die Kanäle E-Mail und Telefon werden dabei am häufigsten genutzt. Wurde das Ticket am Helpdesk erstellt, beginnt die Bearbeitung.

Ein Ticket hat drei Freitextfelder: Kurzbeschreibung, Beschreibung und Lösung. Die Kurzbeschreibung fasst in kurzen Worten den Inhalt des Tickets zusammen. Die Beschreibung stellt eine ausführliche Sicht auf das Anliegen oder Problem des Kunden dar. Ist die Abarbeitung des Tickets abgeschlossen, wird im Feld Lösung festgehalten wie das Kundenanliegen erledigt beziehungsweise gelöst wurde.

Ein Ticket hat zwei Status: Einen Gesamtstatus und einen Detailstatus. Je nach Bearbeitungsschritt werden Gesamtstatus und Detailstatus automatisch vom Ticketsystem gesetzt. Diese Statusübergänge sind die Grundlage für die Benachrichtigung des Kunden. Wird ein Ticket von einem Status in den nächsten gehoben, so besteht die Möglichkeit, den Kunden darüber zu informieren.

Um ein Ticket genauer zu klassifizieren wird dieses kategorisiert. Es stehen drei Kategorien zur Verfügung. Die Auswahlmöglichkeiten dieser Kategorien hängen vom Kunden und vom jeweiligen

Bearbeitungsschritt ab. Wird ein Ticket nur zwischengespeichert ist die Auswahl der Kategorien nicht von großer Bedeutung. Wird ein Ticket jedoch gelöst oder abgeschlossen, muss eine entsprechend genaue Kategorisierung vorgenommen werden.

Es besteht die Möglichkeit einem Ticket Dateianhänge beizufügen. Diese Attachments dienen dem besseren Verständnis des Kundenanliegen. Aber auch der umgekehrte Weg ist möglich: Bei Abschluss eines Tickets können dem Kunden diverse Dokumente zur Verfügung gestellt werden, die sein Anliegen betreffen. Zum Beispiel können, um zukünftige Probleme zu vermeiden, Schulungsunterlagen zur Verfügung gestellt werden.

Für Auswertungszwecke werden die Bearbeitungszeiten eines Tickets mitprotokolliert. Dies passiert einerseits transparent für den Bearbeiter des Tickets. Andererseits kann er korrigierend eingreifen und diese Zeiten manuell pflegen, um Korrekturen durchführen zu können. Jeder Bearbeitungsschritt des Tickets wird mit einer Zeitdauer versehen. Die Gesamtbearbeitungszeit ergibt sich aus der Summe der Zeiten der Bearbeitungsschritte.

5.7.2. Betrachtungswinkel: Support Technology

Verschiedenen automatisierte Mechanismen helfen dem Helpdesk bei der Abarbeitung von Tickets. Diese unterstützenden Technologien werden hier beschrieben. Es werden Technologien aufgelistet die nicht bereits im Kapitel 5.7.1 erwähnt wurden. Dieser Blickwinkel ergänzt mit seinen Informationen die Analyse der Domäne.

Automatisches Ticketrouting

Im Helpdesk gibt es unterschiedliche Zuständigkeiten. Diese Zuständigkeiten richten sich nach verschiedenen Kriterien. Es gibt eine Gruppe von Supportern die für einen bestimmten Kunden zuständig ist. Weiters gibt es Expertengruppen die für ein spezielles Fachgebiet verantwortlich sind. Um die Zuteilung der Tickets zu vereinfachen, werden diese bei der Erstellung automatisch an die richtige Position im Helpdesk geroutet. Der Routingprozess erkennt anhand der Ticketdaten wohin das Ticket geschickt werden muss, um eine entsprechende Abarbeitung zu veranlassen. Die Entscheidung wird anhand des Kunden oder anhand der Kategorisierung des Tickets getroffen.

Benachrichtigung des Kunden

Der Kunde soll über den Bearbeitungsfortschritt seines Anliegens informiert werden. Dazu wird ein automatisierter Benachrichtigungsprozess benutzt. Wichtige Ereignisse im Lebenszyklus eines Tickets sind mit Statusübergängen behaftet. Diese Statusübergänge werden herangezogen um den Kunden zu informieren.

Es besteht natürlich auch die Möglichkeit dem Kunden manuell eine Nachricht zukommen zu lassen. Eine eventuelle Antwort des Kunden wird automatisch an das Ticket angehängt.

Überwachung der Service-Level-Agreements

Ein Kundenanliegen muss innerhalb einer definierten Zeit abgearbeitet werden. Diese Zeiten werden durch SLAs definiert. Ein SLA wird individuell mit jedem Kunden vereinbart. Eine visuelle Darstellung dieser Zeiten hilft den Supportern bei der Bearbeitung der Tickets. Tickets werden „grün“

dargestellt wenn das SLA eingehalten wird. Kurz bevor das SLA verletzt wird, wird das Ticket „gelb“ dargestellt (siehe Abbildung 5.2). Wurde das SLA nicht eingehalten, wird das Ticket „rot“ dargestellt.

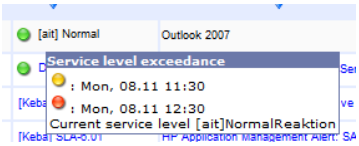

Abbildung 5.2.: Visualisierung der SLA-Überwachung

E-Mail

Ein immer beliebter werdender Kommunikationsweg mit dem Kunden ist der E-Mail-Kanal. Einerseits kippen Kunden ihre Problem oder Anliegen per E-Mail beim Helpdesk ein. Andererseits verwenden Supporter den Kanal E-Mail um beim Kunden Rückfrage zu halten. Die Antworten des Kunden werden dann automatisch an das entsprechende Ticket angehängt. Das Ticketingsystem besitzt die Funktionalität, per E-Mail mit dem Kunden in Kontakt zu treten. Im Ticket gibt es einen eigenen Bereich der die Kommunikation per E-Mail mit dem Kunden protokolliert.

5.7.3. Betrachtungswinkel: Management and Organisation

Die strategische Ausrichtung der Shared Service Factory (SSF) geht ganz klar in Richtung Neukundengewinnung und Expansion der SSF. Es gibt bereits jetzt internationale Kunden die Services der SSF beziehen. Die Mehrsprachigkeit der Callagents sowie auch der eingesetzten Software ist ein wichtiger Punkt der bei der Analyse der Domäne herausgefunden wurde. Diese Information wurde in die Domain Description Units aufgenommen.

Weiters müssen durch die unterschiedlichen Kunden auch unterschiedliche Designs unterstützt werden. Diese Designs beziehen sich auf die Oberfläche der Software sowie auch auf die Kundenbenachrichtigungen. Kunden möchten ihr Corporate-Design auch bei den Benachrichtigungen, die sie vom Helpdesk empfangen, sehen. Diese Anforderung wurden ebenfalls in Domain Description Units umgesetzt.

Die Organisation des Helpdesk findet sich in Abbildung 5.3 wieder. Vorlagen werden einsereits von Teamleitern erstellt. Diese werden dann von den Supportern und Callagents benutzt, um die Kunden über Bearbeitungsfortschritte zu informieren. Andererseits werden Vorlagen von Supportern und Callagents vorgeschlagen. Da sie den direkten Kontakt zum Kunden haben, kommen Anforderungen über Vorlagen aus dem täglichen, immer wiederkehrenden Abläufen. Um diese Abläufe zu vereinfachen, wird beim entsprechenden Teamleiter der Vorschlag für eine Vorlage deponiert. Dieser entscheidet dann, ob die Vorlage realisiert wird und gibt diese dann in Auftrag. Wurde die Vorlage erstellt, wird sie vom Teamleiter freigegeben. Dieser Ablauf wurde in den Domain Description Units hinzugefügt.

5.7.4. Betrachtungswinkel: Rules and Regulations

Bevor ein Callagent in den Produktivbetrieb übergeht, wird eine Schulung abgehalten. Diese erstreckt sich über zwei bis drei Wochen und ist in mehrere Phasen unterteilt. Die aus Sicht des Domain Engineering-Prozesses wichtigsten Punkte werden hier dargestellt.

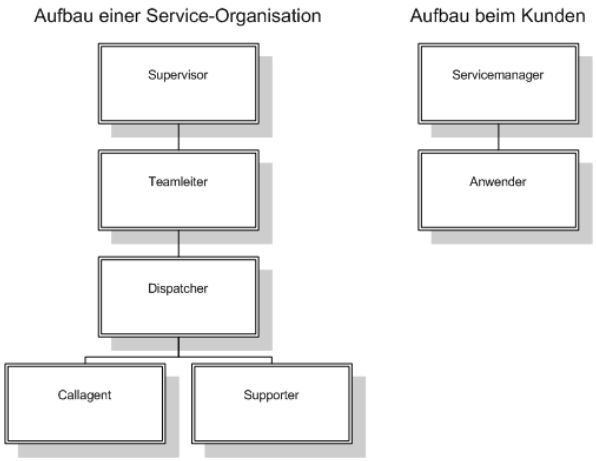

Abbildung 5.3.: Organigramm des Helpdesks in der Shared Service Factory

Telefonie

Ein reibungsloser Umgang mit dem Telefon ist einer der wichtigsten Punkte die ein Callagent beherrschen muss. Funktionen wie Rückfragen, Halten, Aktivierung des Pausemodus sowie die richtige Begrüßung werden dem Callagent vermittelt.

Ticketingtool

Ein weiteres wichtiges Werkzeug für die tägliche Arbeit ist das Ticketingtool. Hier wird besonders auf die Qualität der Tickets geachtet. Das richtige Aufnehmen eins Tickets wird durch theoretische und praktische Beispiele erläutert. Die Aufnahme von Testtickets unterstützt diesen Prozess. Je nach Kunden sind unterschiedliche Daten zu erfassen. Um diese Arbeit zu erleichtern, gibt es für die einzelnen Kunden Ticketvorlagen. Diese Ticketvorlagen sind als eine Art Leitfaden für den Callagent gedacht. Die nötigen Punkte, die beim Aufnehmen eines Tickets angegeben werden müssen, sind in der Ticketvorlage hinterlegt. Damit wird eine gewisse Grundqualität der Tickets sichergestellt.

Weiters muss das Ticket in der Sprache des Kunden aufgenommen werden. Der Benachrichtigungsprozess des Ticketingtools sieht auch Rückmeldungen an den Kunden vor. Darin sind eventuell die Problembeschreibung sowie die Lösung des Problems vorhanden. Der Kunde möchte natürlich diese Daten in seiner Sprache vorfinden.

Wissensdatenbank

Die Wissensdatenbank dient als Nachschlagewerk für Standardprobleme. Immer wiederkehrende Abläufe und Anfragen können mit Standardantworten abgedeckt werden. Die Gliederung der Wissensdatenbank wird dem Callagent näher gebracht, damit ein rasches Auffinden von Antworten und Lösungen gewährleistet werden kann.

Auch interne Fragen, die immer wieder auftauchen, werden in die Wissensdatenbank aufgenommen. Zu denen gehören zum Beispiel „Welcher Kunde bekommt welche Services?“ und „Wie buchstabiere ich richtig?“.

Regelverletzungen

Die Einhaltung der Regeln wird von der jeweils übergeordneten Instanz kontrolliert. Dazu stehen mehrere Hilfswerkzeuge zur Verfügung. Einerseits wird ein internes Monitoring-Tool eingesetzt, um die Abläufe im Helpdesk zu beaufsichtigen. Jeder Anruf im Helpdesk wird aufgezeichnet und steht für einen gewissen Zeitraum zum Nachhören zur Verfügung. Dispatcher und Teamleiter kontrollieren stichprobenartig einzelne Anrufe auf Korrektheit. Werden Fehler entdeckt, wird der entsprechende Callagent darauf aufmerksam gemacht. Wiederholen sich die Fehler, wird eine Nachschulung durchgeführt.

Bei Abschluss eines Tickets durch den Dispatcher wird ebenfalls auf die Qualität der Tickets geachtet. Vorgeschriebene Daten müssen aufgenommen worden sein, die Sprache des Tickets muss stimmen, und vieles mehr. Auch hier wird bei Fehlern der entsprechende Bearbeiter des Tickets darauf hingewiesen. Zur Durchführung von Nachschulungen kommt es bei vermehrtem Auftreten von Fehlern.

5.7.5. Betrachtungswinkel: Scripts

Dieser Bereich stellt vertragliche Vereinbarungen dar, die für die Domäne relevant sind. Es werden wiederum nur ergänzende Informationen, die nicht schon in Kapitel 5.7.1 aufgelistet sind, aufgezeigt. In Abbildung 5.4 wird eine solche Vereinbarung mit dem Kunden dargestellt.

Global Call Desk

Service Levels / KPI's

Validity date: 01.01.2011

SLO name	SLO values
Telephone Response	• >1000 Calls per month: 95.00% answered within a 30 second threshold. Lost calls within 30 seconds excluded. • <1000 Calls per month: 90% answered within a 30 second threshold Lost calls within 30 seconds excluded.
Abandon Call Rate	• >1000 Calls per month: 7.00% abandon (excluded lost calls in 5 seconds) • <1000 calls per month: 10.00% abandon (excluded lost calls in 5 seconds)
Response time	• Office Hours (07:00-18:00) 90% in 10 Minuten • 7x24h: 85% in 15 Minutes
Correct log and route	< 5% routing errors

Abbildung 5.4.: Service-Level KPIs

Service-Level-Agreement

Ein Service-Level-Agreement wird mit jedem Kunden individuell vereinbart. Ein Kunde kann mehrere Service-Level-Agreements zugewiesen bekommen. Die Unterscheidung basiert auf der Kategorisierung der Tickets. Die Priorität eines Tickets hat keinen Einfluss auf das SLA. Die Überwachung der

SLAs erfolgt automatisiert (siehe Kapitel 5.7.2). Der Supporter oder Callagent hat die Möglichkeit die Überwachung vorübergehend auszusetzen. Dies muss mit der Eingabe eines Kommentars begründet werden. Es ist im Moment nicht vorgesehen, dem Kunden ein Aussetzen der SLA-Überwachung mitzuteilen.

Kurz vor der Verletzung eines SLAs und bei der Überschreitung der im SLA festgelegten Zeiten kann eine Benachrichtigung erfolgen. Die entsprechenden Vorlagen dafür beinhalten den Zeitpunkt an dem die Bearbeitung des Tickets abgeschlossen sein sollte.

Ein Service-Level-Agreement beinhaltet mehrere zu überwachende Service-Level. Ein Service-Level definiert einen in einer bestimmten Zeit zu erreichenden Status eines Tickets. Diese Zeit wird in Minuten angegeben. Ein weiteres Attribut eines Service-Levels ist ein Prozentsatz der festlegt, ab wann ein Ticket als „kurz vor Verletzung des Service-Level“ gekennzeichnet wird. Die Grundlage dieses Prozentsatzes ist die Anzahl der Minuten bis zur Verletzung des Service-Levels.

Abbildung 5.4 zeigt folgendes Beispiel: Die Reaktionszeit innerhalb der Normalarbeitszeit (07:00 - 18:00 Uhr) beträgt zehn Minuten und muss für 90% der Tickets eingehalten werden. Ist ein 7x24h-Betrieb gewünscht, beträgt die Reaktionszeit 15 Minuten und muss für 85% der Tickets erreicht werden.

Antwortzeiten Telefonanruf

Eine weitere Vereinbarung die mit dem Kunden getroffen wird, ist die Antwortzeit eines Callagents am Telefon. Bei einem Callvolumen von über 1000 Anrufen pro Monat müssen 95% der Anrufe innerhalb von 30 Sekunden angenommen werden. Bei unter 1000 Anrufen pro Monat müssen 90% der Anrufe innerhalb von 30 Sekunden angenommen werden.

Nicht angenommene Anrufe

Auch hier gibt es wieder eine Unterscheidung nach Callvolumen. Bei mehr als 1000 Anrufen im Monat dürfen nicht mehr als sieben Prozent der Anrufe nicht angenommen worden sein. Bei weniger als 1000 Anrufen pro Monat steigt der Wert von sieben auf zehn Prozent. Anrufe die innerhalb der ersten fünf Sekunden abgebrochen werden finden keine Berücksichtigung.

Korrektes Log & Route

Weniger als fünf Prozent der Tickets, die über das Service „Log & Route“ aufgenommen werden, dürfen falsch weitergeleitet worden sein.

5.7.6. Betrachtungswinkel: Human Behaviour

Die unterschiedlichen Ausprägungen bei der Befolgung der Regeln sollen hier festgehalten werden.

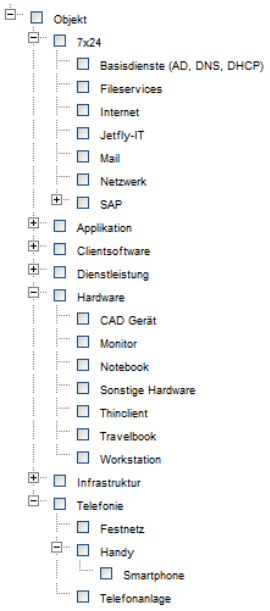

Abbildung 5.5.: Kategoriebaum

Verwenden von Ticketvorlagen bei Standardprolemen

Bei Standardproblemen der Benutzer (Passwort vergessen, Freischalten einer Patchdose, Mitarbeiter Ein- bzw. Austritt, etc.) sind Ticketvorlagen zu verwenden. Diese Vorlagen geben Felder vor, die vom Supporter oder Callagent auszufüllen sind. Ob dies Vorlagen dann auch verwendet werden, hängt vom Bearbeiter des Tickets ab. Weiters sind diese Felder als Freitext definiert und nicht in einer Eingabemaske verankert. Somit besteht die Möglichkeit diese Felder zu ignorieren und eigenen Text für die Beschreibung eines Tickets zu erfassen.

Kategorisierung der Tickets

Supporter und Callagents sind angewiesen, ein Ticket so gut als möglich zu kategorisieren. Die Kategorisierung eines Tickets hat weitreichende Auswirkungen. Zum einen wird das Service-Level-Agreement für dieses Ticket anhand der Kategorien festgelegt (siehe 5.7.2). Weiters wird auch das automatische Ticketrouting aufgrund der Kategorisierung des Tickets durchgeführt (siehe Support-Technologie 5.7.2).

Kategorien sind in einer Baumstruktur angeordnet (siehe Abbildung 5.5). Auf den oberen Ebenen werden die Kategorien grob unterteilt. Umso tiefer die Auswahl im Baum getroffen wird, umso konkreter ist die Kategorisierung.

5.8. Validierung der Domäne

Die Validierung der Domäne wurde mit einer niedrigen Priorität eingestuft. Die langjährige Erfahrung des Domain-Engineers sowie eine ständige Weiterentwicklung der Ticketingsoftware stellten sicher,

dass die richtige Domäne untersucht wurde. Weiters war der enge Kontakt mit den wichtigsten Stakeholdern der Domäne auch ein Faktor, der die Validierung der Domäne bis zu einem gewissen Grad nicht notwendig machte.

5.9. Verifikation der Domäne

Zur Verifikation der Domäne wurde die Prädikatenlogik gewählt. Mit Hilfe von Prover9 (McCune (2010)) wurden ausgewählte Domain Description Units in Aussagen der Prädikatenlogik umgewandelt. Prover9 ist ein automatischer Theorem-Beweiser.

5.9.1. Verifikation mit Prover9

Der folgende Code repräsentiert eine Auswahl an Domain Description Units, die als Annahmen (Assumptions) in Prover9 formuliert wurden. Weiters wurden dann die Requirements als Ziel (Goals) festgehalten. Durch das Auflösen der Ziele mit Prover 9 kann gezeigt werden, dass die Requirements hinsichtlich der untersuchten Domäne umsetzbar sind. Die einzelnen Eigenschaften werden hier ausführlicher Beschrieben.

Zu Beginn wurde festgelegt, dass jeder Benutzer mindestens eine Sprache hat und mindestens in einer Rolle im System vorhanden sein muss:

```
% Eigenschaften:
% E1: Jeder Benutzer hat min. eine Sprache
all Benutzer exists Sprache benutzerHatSprache(Benutzer, Sprache).

% E2: Jeder Benutzer hat min. eine Rolle
all Benutzer exists Rolle benutzerHatRolle(Benutzer, Rolle).
```

Jede Vorlage, oder auch als Template bezeichnet, muss in mindestens einer Sprache verfügbar sein. Änderungen an Vorlagen dürfen nur von Benutzern durchgeführt werden, die im Besitz der Rolle „templateEditor“ sind:

```
% E3: Jede Vorlage ist in min. einer Sprache verfügbar
all Vorlage exists Sprache vorlageHatSprache(Vorlage, Sprache).

% E4: Änderungen an Vorlagen dürfen nur von der Rolle
% "templateEditor" durchgeführt werden
vorlage(Vorlage) &
benutzerHatRolle(Benutzer, templateEditor)
  -> benutzerAendertVorlage(Benutzer, Vorlage).
```

Eine Vorlage besteht aus Textbausteinen. Diese bestehen wiederum aus Variablen und Freitext:

```
% E5: Ein Textbaustein besteht aus Freitext und Variablen
textbaustein(Textbaustein) &
```

```
variable(Variable) &
freitext(Freitext)
  -> textbausteinBestehtAus(Textbaustein, Variable, Freitext).

% E6: Jede Vorlage besteht aus Textbausteinen
vorlage(Vorlage) &
textbaustein(Textbaustein) &
exists Variable exists Freitext
  textbausteinBestehtAus(Textbaustein, Variable, Freitext) &
vorlageHatTextbaustein(Vorlage, Textbaustein)
  -> vorlageBestehtAus(Vorlage, Textbaustein).
```

Vorlagen müssen freigegeben werden bevor sie benutzt werden dürfen. Diese Freigaben dürfen nur von Benutzern mit der Rolle „dispatcher“ durchgeführt werden:

```
% E7: Eine freigegebene Vorlage muss vorher von einem
% Benutzer der Dispatcher-Rolle freigegeben werden.
vorlage(Vorlage) &
benutzerGibtVorlageFrei(Benutzer, Vorlage) &
benutzerHatRolle(Benutzer, dispatcher)
  -> freigegebeneVorlage(Vorlage).
```

Vorlagen werden in einem Content-Repository abgelegt. Die Ablage erfolgt in einer Ordnerstruktur. Eine gültige Vorlage muss in einem Ordner liegen:

```
% E8: Jede Vorlage liegt in einem Ordner
exists Ordner ordner(Ordner) &
vorlage(Vorlage) &
vorlageLiegtImOrdner(Vorlage, Ordner)
  -> vorlageImOrdner(Vorlage).
```

Eine Vorlage kann einen Gültigkeitsbereich haben. Dieser Bereich wird mit einem von-Datum und einem bis-Datum dargestellt. Ist eine Vorlage immer gültig, kann die Angabe eines bis-Datums und eines von-Datums entfallen. Weiters muss das von-Datum vor dem bis-Datum liegen.

```
% E9: Eine Vorlage hat einen Gültigkeitsbereich
vorlage(Vorlage) &
(
  (
    exists Von exists Bis
    datum(Von) &
    datum(Bis) &
    istVorher(Von, AktuellesDatum) &
    istVorher(AktuellesDatum, Bis) &
    vorlageMitGueltigkeitsbereich(Vorlage, Von ,Bis)
  ) |
  vorlageImmerGueltig(Vorlage)
```

```
)
  -> vorlageHatGueltigenBereich(Vorlage, AktuellesDatum).
```

Wird eine Vorlage geändert, wird immer ein Kopie erzeugt bevor die Änderung übernommen wird. So kann ist zu jeder Vorlage eine Historie vorhanden.

```
% E10: Eine Vorlage hat eine History
vorlage(VorlageAlt) & vorlage(VorlageAktuell)
  - > vorgaenger(VorlageAlt, VorlageAktuell).
```

Eine Vorlage muss einem Vorlagen-Typ zugeordnet werden können.

```
% E11: Eine Vorlage hat eine Typ.
vorlage(Vorlage) &
vorlagenTyp(Typ) &
vorlageIstVonTyp(Vorlage, Typ)
  -> vorlageHatTyp(Vorlage, Typ).
```

Um eine gültige Vorlage zu erhalten, muss diese folgende Punkte erfüllen:

- Die Vorlage muss aktiv sein.
- Die Vorlage muss freigegeben sein.
- Die Vorlage muss in einem Ordner im Content-Repository liegen
- Die Vorlage muss aus Textbausteinen bestehen
- Die Vorlage muss einen Vorlagen-Typ haben
- Die Vorlage ist nur in der angegebenen Zeitspanne gültig, oder sie ist immer gültig.
- Die Vorlage muss ein Stylesheet hinterlegt haben.

```
% E12: Eine Vorlage ist gültig, wenn
% * sie aktiv ist
% * sie von einem Benutzer freigegeben wurde
% * sie in einem Ordner liegt
% * sie aus Textbausteinen besteht
% * sie einen Typ hat
% * der Gültigkeitsbereich (von-bis-Datum) eingehalten wird
% * sie ein Stylesheet (CSS) hinterlegt hat
vorlage(Vorlage) &
aktiv(Vorlage) &
freigegebeneVorlage(Vorlage) &
exists Ordner vorlageLiegtImOrdner(Vorlage, Ordner) &
exists Textbaustein vorlageBestehtAus(Vorlage, Textbaustein) &
exists Typ vorlageHatTyp(Vorlage, Typ) &
exists Css vorlageHatCssHinterlegt(Vorlage)
  -> gueltig(Vorlage).
```

Die nachfolgenden Eigenschaften beziehen sich auf Benachrichtigungen, die aus Vorlagen erstellt werden.

Eine Benachrichtigung muss aus einer gültigen Vorlage erstellt worden sein:

```
% E13: Eine Benachrichtigung wird aus einer gültigen Vorlage erstellt
benachrichtigung(Benachrichtigung) &
gueltig(Vorlage) &
erstellteBenachrichtigung(Benachrichtigung, Vorlage)
  -> benachrichtigungAusVorlage(Benachrichtigung, Vorlage).
```

Der Benutzer bekommt nur dann eine Benachrichtigung, wenn er oder sie das auch wünscht:

```
% E14: Eine Benachrichtigung geht an einen Benutzer,
% wenn der Benutzer dies wünscht.
benachrichtigung(Benachrichtigung) &
benutzer(Benutzer) &
benutzerWuenschtBenachrichtigung(Benutzer)
  -> benachrichtigungAnBenutzer(Benachrichtigung, Benutzer).
```

Eine Benachrichtigung muss in mindestens einer Sprache vorhanden sein:

```
% E15: Eine Benachrichtigung muss eine Sprache haben.
benachrichtigung(Benachrichtigung) &
sprache(Sprache) &
benachrichtigungHatSprache(Benachrichtigung, Sprache)
  -> benachrichtigungExistiertInSprache(Benachrichtigung, Sprache).
```

Es muss sichergestellt sein, dass ein Benutzer nur Benachrichtigungen in seiner Sprache erhält.

```
% E16: Der Benutzer erhält nur Benachrichtigungen in seiner Sprache
benachrichtigungExistiertInSprache(Benachrichtigung, Sprache) &
benutzerHatSprache(Benutzer, Sprache) &
benachrichtigungAnBenutzer(Benachrichtigung, Benutzer) &
exists Vorlage benachrichtigungAusVorlage(Benachrichtigung, Vorlage)
  -> benutzerErhaeltBenachrichtigungInSprache(Benutzer,
       Benachrichtigung, Sprache, Datum).
```

Alle Ordner, außer der root-Ordner, haben einen übergeordneten Ordner:

```
% E17: Alle Ordner, außer der root-Ordner, haben einen Parent-Ordner
(
  ordner(Ordner) &
  ordner(Parent) &
  ordnerHatParent(Ordner, Parent)
) |
istRoot(Ordner)
  -> gueltigerOrdner(Ordner).
```

Eine Vorlage kann eine Cascading Style Sheet hinterlegt haben oder ist ohne CSS definiert worden:

```
% E18: Eine Vorlage hat entweder kein CSS hinterlegt oder
% hat ein gültiges CSS hinterlegt.
(
  vorlage(Vorlage) &
  css(Css) &
  vorlageHatCss(Vorlage, Css)
) |
vorlageOhneCss(Vorlage)
  -> vorlageHatCssHinterlegt(Vorlage).
```

Für die Modellierung des Gültigkeitbereichs einer Vorlage wird der Kettenschluss für ein Datum festgelegt:

```
% ------------ Datum-Prädikate
istVorher(D1, D2) & istVorher(D2, D3) -> istVorher(D1, D3).
```

Somit ist die Definition der Eigenschaften abgeschlossen. Als nächstes werden die nötigen Instanzen erstellt, um dann das Modell zu testen:

```
% Beginn Definitionen der Instanzen
% Benutzer definieren
benutzer(s1callagent).
benutzerWuenschtBenachrichtigung(s1callagent).
benutzer(s1supporter).
benutzerWuenschtBenachrichtigung(s1supporter).
benutzer(s1dispatcher).

% Sprachen definieren
sprache(de).
sprache(en).

% Rollen definieren
rolle(supporter).
rolle(callagent).
rolle(anwender).
rolle(dispatcher).
rolle(templateEditor).

% Beziehung: Benutzer <--> Sprache
benutzerHatSprache(s1supporter, en).
benutzerHatSprache(s1callagent, de).
benutzerHatSprache(s1dispatcher, de).

% Beziehung: Benutzer <--> Rollen
```

```
benutzerHatRolle(s1callagent, callagent).
benutzerHatRolle(s1supporter, supporter).
benutzerHatRolle(s1supporter, templateEditor).
benutzerHatRolle(s1dispatcher, dispatcher).

% Testbausteine für Vorlagen definieren
textbaustein(anrede).
textbaustein(inhalt).
textbaustein(grussFormel).

% Variablen für Textbausteine definieren
variable(anwenderName).
variable(bearbeiterName).
variable(ticketNummer).
variable(erstellungsZeitpunkt).

% Freitext für Textbausteine definieren
freitext(beliebigerText).

% Vorlagen-Typen definieren
vorlagenTyp(notification).
vorlagenTyp(email).
vorlagenTyp(ackEmail).
vorlagenTyp(subject).
vorlagenTyp(tAMTemplate).

% CSS definieren
css(druckCss).
css(bildschrimCss).

% Datum definieren
datum(20110101).
datum(20110201).
datum(20110301).
datum(20110406).
datum(20110701).
datum(20111231).
istVorher(20110101, 20110201).
istVorher(20110201, 20110301).
istVorher(20110301, 20110406).
istVorher(20110406, 20110701).
istVorher(20110701, 20111231).

% Vorlagen definieren
% Vorlage: neuesTicket
vorlage(neuesTicket).
aktiv(neuesTicket).
```

```
vorlageIstVonTyp(neuesTicket, notification).
vorlageHatCss(neuesTicket, druckCss).
vorlageHatCss(neuesTicket, bildschrimCss).
vorlageHatSprache(neuesTicket, de).
vorlageHatSprache(neuesTicket, en).
vorlageLiegtImOrdner(neuesTicket,kunden).
vorlageImmerGueltig(neuesTicket).
vorlageHatTextbaustein(neuesTicket, inhalt).

% Vorlage: ticketAbgeschlossen
vorlage(ticketAbgeschlossen).
aktiv(ticketAbgeschlossen).
vorlageIstVonTyp(ticketAbgeschlossen, notification).
vorlageOhneCss(ticketAbgeschlossen).
vorlageLiegtImOrdner(ticketAbgeschlossen,kunden).
vorlageImmerGueltig(ticketAbgeschlossen).
vorlageHatTextbaustein(ticketAbgeschlossen, inhalt).

% Vorlage: passwortZuruecksetzen
vorlage(passwortZuruecksetzen).
aktiv(passwortZuruecksetzen).
vorlageIstVonTyp(passwortZuruecksetzen, email).
vorlageOhneCss(passwortZuruecksetzen).
vorlageLiegtImOrdner(passwortZuruecksetzen,kunden).
vorlageImmerGueltig(passwortZuruecksetzen).
vorlageHatTextbaustein(passwortZuruecksetzen, inhalt).

% Vorlage: userEintritt
vorlage(userEintritt).
aktiv(userEintritt).
vorlageIstVonTyp(userEintritt, email).
vorlageMitGueltigkeitsbereich(userEintritt, 20110101, 20111231).
vorlageHatTextbaustein(userEintritt, inhalt).

% Vorlage: userAustritt
vorlage(userAustritt).
aktiv(userAustritt).
vorlageIstVonTyp(userAustritt, email).
vorlageMitGueltigkeitsbereich(userAustritt, 20110101, 20110301).
vorlageHatTextbaustein(userAustritt, inhalt).

% Benachrichtigungen definieren
benachrichtigung(b1).
benachrichtigungHatSprache(b1, en).
benachrichtigungHatSprache(b1, de).

benachrichtigung(b2).
```

```
benachrichtigungHatSprache(b2, en).
benachrichtigungHatSprache(b2, de).

% Verknüpfung Benachrichtigungen <--> Vorlagen
erstellteBenachrichtigung(b1, neuesTicket).
erstellteBenachrichtigung(b2, ticketAbgeschlossen).

% Freigeben von Vorlagen
benutzerGibtVorlageFrei(s1dispatcher, neuesTicket).
benutzerGibtVorlageFrei(s1dispatcher, ticketAbgeschlossen).
benutzerGibtVorlageFrei(s1dispatcher, passwortZuruecksetzen).
benutzerGibtVorlageFrei(s1dispatcher, userEintritt).
benutzerGibtVorlageFrei(s1dispatcher, userAustritt).

% Ordnerstruktur für Vorlagen definieren.
% root              (root-Ordner)
%  |--email         (Ordner für E-Mail-Vorlagen)
%  |    |--kunden   (Ordner für Kunden E-Mail-Vorlagen)
%  |    |--so       (Ordner für Service-Organisation-E-Mail-Vorlagen)
%  |--notifications (Ordner für Benachrichtigungen)
ordner(root).
ordner(email).
ordner(notifications).
ordner(kunden).
ordner(so).
istRoot(root).
ordnerHatParent(email, root).
ordnerHatParent(kunden, email).
ordnerHatParent(so, email).
ordnerHatParent(notifications, root).
```

Somit sind jetzt das Modell und die geforderten Eigenschaften modelliert. Nun kann durch Angabe eines Ziels (Goals in prover9) die Konsistenz der Eigenschaften überprüft werden:

```
% Zu Prüfen: Der Benutzer "s1supporter" erhält die Benachrichtigung
% "b2" in der Sprache "en" zum Datum "20110406".
benutzerErhaeltBenachrichtigungInSprache(s1supporter, b2, en, 20110406).
```

Ist prover9 in der Lagen die Anfrage aufzulösen, ist das Modell und die erzeugten Instanzen konsistent. Würde prover9 kein Ergebnis liefern, muss untersucht werden, warum die Anfrage nicht beantwortet werden konnte. Mögliche Ursachen können in den definierten Instanzen liegen. Eine weitere Möglichkeit ist eine Inkonsistenz in den Eigenschaften des Modells.

Kapitel 6

Requirements der Template-Engine

Dieses Kapitel beschreibt die Requirements die an die neue Template-Engine gestellt werden. Dabei wird die beschriebene Vorgehensweise von Kapitel 4 eingehalten.

6.1. Business Process Reengineering

Die bestehenden Prozesse werden durch die Implementation der Template-Engine nicht verändert. Die Template-Engine bietet kein Benutzerinterface für Anwender. Es werden API * Funktionen angeboten, die von anderen Tools im Helpdesk verwendet werden können. Eine indirekte Auswirkung auf Prozesse gibt es dahingehen, als dass bestehende Produkte die Funktionalität der Template-Engine nutzen.

6.1.1. Kommunikation mit Anwender über E-Mail-Kanal

Die Auswahl der Vorlagen basiert jetzt auf der Sprache des betroffenen Benutzers, der das Ticket aufgegeben hat. Dadurch ergibt sich nur mehr eine Untermenge an E-Mail-Vorlagen, die zur Kommunikation mit dem betroffenen Benutzer ausgewählt werden können. Die Menge wird durch die Ticketsprache eingeschränkt. Dadurch liegt die Entscheidung, in welcher Sprache mit dem betroffenen Anwender kommuniziert werden soll, nicht mehr beim Callagent. Diese Entscheidung wird automatisch von der beim Benutzer hinterlegten Sprache abgeleitet.

6.1.2. Kategorisierung des Tickets

Durch die Integration von trueAct NLA † ist es möglich, über den Beschreibungstext des Tickets einen Standardlösungstext zu finden. Dieser Standardlösungstext wird in einer Vorlage hinterlegt. Dabei ist es möglich, dass sich die Kategorisierung des Tickets automatisch verfeinert oder gar korrigiert wird.

*Application Programmer Interface
†Natural Language Analysis

6.1.3. Erfassen von neuen Benutzern

Bei der Erfassung von neuen Benutzern ist darauf zu achten, dass auch die Sprache richtig eingepflegt wird. Die Benutzersprache hat Auswirkungen auf die Benachrichtigungen (sowohl automatische als auch manuelle Benachrichtigungen) und auch auf die Ticketsprache.

6.2. Requirements der Domäne

Die Sichtung der Domain Description Units ergab eine Untermenge an Units die für die Template-Engine relevant waren. Zusätzlich wurden neue Requirements gefunden und ebenfalls in die Liste aufgenommen. Eine genauere Beschreibung warum diese Domain Description Unit zu einem Requirement wurde, wurde ebenfalls ergänzt.

6.2.1. Übernommene Domain Description Units

Folgende Domain Description Units wurden als Requirement aufgenommen:

1. Für Benachrichtigungen wird einmalig ein Design (CSS) hinterlegt.

   Namen: Benachrichtigung, Design

   Art: Entity

   Quelle: TS

   Datum: 26.04.2010

   Requirements-Ergänzung: CSS muss ebenfalls im Repository abgelegt werden.

2. Ein Ticket wird gedruckt (PDF) oder verschickt (E-Mail).

   Namen: Ticket, PDF, E-Mail

   Art: Entity (Ticket, PDF, E-Mail), Funktion (drucken, verschicken)

   Quelle: TS

   Datum: 26.04.2010

   Requirements-Ergänzung: Variabler Teil der Vorlagen wird durch Ticketdaten aufgelöst.

3. Der Inhalt einer Benachrichtigung wird in mehreren Sprachen angegeben.

   Namen: Benachrichtigung, Sprache

   Art: Entity

   Quelle: TS

   Datum: 26.04.2010

   Requirements-Ergänzung: Vorlagen werden in mehreren Sprachen im Repository abgelegt.

4. In einer Benachrichtigung gibt es sowohl fixen als auch variablen Text.

   Namen: Benachrichtigung, fixer Text, variabler Text

Art: Entity

Quelle: TS

Datum: 26.04.2010

Requirements-Ergänzung: Variabler Text wird durch *${Variable}* gekennzeichnet.

5. Benachrichtigungen werden in einer hierarchischen Ordnerstruktur abgelegt.

   Namen: Benachrichtigung, Hierarchie, Ordnerstruktur

   Art: Entity

   Quelle: TS

   Datum: 26.04.2010

   Requirements-Ergänzung: Die Ordnerstruktur wird im Repository abgebildet.

6. Die hierarchischen Ordnerstruktur soll in mehreren Sprachen verfügbar sein.

   Namen: Benachrichtigung, Hierarchie, Ordnerstruktur

   Art: Entity

   Quelle: GS

   Datum: 23.01.2011

   Requirements-Ergänzung: Zur Ordnerstruktur wird zusätzlich noch die Sprache abgelegt.

7. Eine Vorlage wird vom Teamleiter genehmigt.

   Namen: Vorlage, Teamleiter, genehmigen

   Art: Entity (Vorlage, Teamleiter), Function(genehmigen)

   Quelle: TS

   Datum: 26.04.2010

   Requirements-Ergänzung: Zugang zum Repository muss durch Rollen eingeschränkt werden.

8. Eine Vorlage wird von einem Supporter/Callagent vorgeschlagen.

   Namen: Vorlage, Supporter/Callagent, vorschlagen

   Art: Entity (Vorlage, Supporter/Callagent), Ereigniss(vorschlagen)

   Quelle: GS

   Datum: 26.04.2010

   Requirements-Ergänzung: Der Vorschlag wird an den Teamleiter übermitteltn.

9. Die Variablen in einer Vorlage haben deutsche Bezeichnungen

   Namen: Variable, Vorlage

   Art: Entity (Variable, Vorlage)

   Quelle: GS

Datum: 26.04.2010

Requirements-Ergänzung: Variablen sind in mehreren Sprachen verfügbar.

10. Vor dem Abschicken einer E-Mail wird eine Vorschau aufgerufen.

    Namen: E-Mail, Vorschau,

    Art: Entity (E-Mail, Vorschau), Function(vorschlagen)

    Quelle: TS

    Datum: 26.04.2010

    Requirements-Ergänzung: Die Vorschau wird in der Sprache des dazugehörigen Tickets angezeigt.

11. Nicht mehr benötigte Vorlagen werden aus der Ordnerstruktur gelöscht.

    Namen: Vorlagen, Ordnerstruktur, löschen

    Art: Entity (Vorlage, Ordnerstruktur), Function(löschen)

    Quelle: TS

    Datum: 26.04.2010

    Requirements-Ergänzung: Für das Löschen von Vorlagen wird ein eigenes Recht benötigt.

12. An eine Benachrichtigung werden Anhänge beigefügt.

    Namen: Benachrichtigung, Anhänge, beifügen

    Art: Entity (Benachrichtigung, Anhänge), Function (beifügen)

    Quelle: RM

    Datum: 27.04.2010

    Requirements-Ergänzung: In der Vorlage muss gekennzeichnet werden, welche Anhänge zur Vorlage gehören.

13. Anhänge werden extra hochgeladen oder kommen direkt aus dem Ticket

    Namen: Anhänge, Ticket, hochladen

    Art: Entity (Ticket, Anhänge), Function (hochladen)

    Quelle: RM

    Datum: 27.04.2010

    Requirements-Ergänzung: Übernahme der Anhänge aus dem Ticket oder direkt aus der vordefinierten Vorlage.

14. Benachrichtigungen werden an einen Benutzer verschickt.

    Namen: Benachrichtigung, Benutzer, verschicken

    Art: Entity (Benachrichtigung, Benutzer), Function (verschicken)

    Quelle: RM

    Datum: 27.04.2010

Requirements-Ergänzung: Der Benutzer kann die Sprache selbst bestimmen, in der er die Benachrichtigungen erhalten möchte.

15. Benutzer werden manuell über den Status des Tickets benachrichtigt.

    Namen: Benutzer, Ticket, manuell benachrichtigen

    Art: Entity (Benutzer, Ticket), Function (manuell benachrichtigen)

    Quelle: HCW

    Datum: 03.05.2010

    Requirements-Ergänzung: Die manuelle Benachrichtigung kann über eine Vorlage erfolgen.

16. Eine E-Mail an den Benutzer beinhaltet Ticketdaten

    Namen: E-Mail, Kunde, Ticketdaten

    Art: Entity

    Quelle: HD

    Datum: 05.05.2010

    Requirements-Ergänzung: Ticketdaten werden über Variablen in die Vorlage gerendert.

17. Eine E-Mail an den Benutzer beinhaltet Standardtexte

    Namen: E-Mail, Kunde, Standardtext

    Art: Entity

    Quelle: HD

    Datum: 05.05.2010

    Requirements-Ergänzung: Der Standardtext steht in mehreren Sprachen zur Verfügung.

18. Vorlagen werden aufgrund der Kategorisierung des Tickets ausgewählt.

    Namen: Vorlagen, Kategorisierung, Ticket, auswählen

    Art: Entity (Vorlagen, Kategorisierung, Ticket), Function (auswählen)

    Quelle: GS

    Datum: 07.05.2010

    Requirements-Ergänzung: Vorlagen haben Eigenschaften hinterlegt, die auf Kategorien umgelegt werden können.

19. Der Betreff einer E-Mail an den Kunden wird aus einer vordefinierten Liste gewählt.

    Namen: Betreff, E-Mail, Kunde

    Art: Entity

    Quelle: GS

    Datum: 07.05.2010

    Requirements-Ergänzung: Betreffe werden auch im Repository — getrennt von den normalen Vorlagen — abgelegt.

20. Benachrichtigungen an Benutzer werden als HTML-E-Mail verschickt.

    Namen: Benachrichtigung, Benutzer, HTML-E-Mail

    Art: Entity

    Quelle: GS

    Datum: 07.05.2010

    Requirements-Ergänzung: Die Vorlagen können sowohl als reiner Text als auch im HTML-Format abgelegt werden.

21. Die Vorschau bietet die Möglichkeit, den variablen Teil einer E-Mail aufzulösen.

    Namen: Vorschau, variabler Teil, auflösen

    Art: Entity (Vorschau, variabler Teil), Function (auflösen)

    Quelle: GS

    Datum: 07.05.2010

    Requirements-Ergänzung: Die Variablen werden durch Ticketdaten aufgelöst.

22. Die zur Verfügung stehenden Variablen sind über ein „Hilfe“-Symbol abrufbar.

    Namen: Variablen, Hilfe-Symbol

    Art: Entity

    Quelle: GS

    Datum: 07.05.2010

    Requirements-Ergänzung: Die verfügbaren Variablen werden in der Datenbank hinterlegt.

23. Für jede Variable ist eine kurze Erklärung hinterlegt.

    Namen: Variable, Erklärung

    Art: Entity

    Quelle: GS

    Datum: 07.05.2010

    Requirements-Ergänzung: Die verfügbaren Variablen haben einen Beschreibungstext hinterlegt.

24. Variablen sind nur in einer Sprache (Deutsch) verfügbar.

    Namen: Varialben, Sprache

    Art: Entity

    Quelle: GS

    Datum: 07.05.2010

    Requirements-Ergänzung: Variablen sind in mehreren Sprachen verfügbar.

25. Benachrichtigungen beinhalten das Corporate Design der Kunden.

    Namen: Benachrichtigung, Corporate Design

Art: Entity

Quelle: GS

Datum: 14.05.2010

Requirements-Ergänzung: Das Corporate Design kann als CSS hinterlegt werden.

6.3. Requirements des Interface

Die Template-Engine hat kein direktes Interface für den Endanwender. Sie bietet eine API an, von der sich andere Applikationen diverse Vorlagen beziehen können. Das Bereitstellen von Vorlagen soll über ein Standardprotokoll erfolgen (zum Beispiel HTTP).

Die Ablage der einzelnen Vorlagen soll in einem Repository erfolgen. Die Anforderungen an das Repository gehen aus den Anforderungen an die Domäne hervor (siehe 6.2).

6.4. Requirements der Hardware

Die Anforderungen an die Hardware werden in den folgenden Punkten beschrieben:

Performance

Die Template-Engine stellt keine außergewöhnlichen Anforderungen an die Hardware. Das zur Verfügung stellen von Vorlagen (lesende Zugriffe) stellt keine besondere Herausforderung an die Hardware dar. Eine für lesende Zugriffe optimierte Festplattenkonfiguration stellt die entsprechende Performance sicher. Da sich Vorlagen nur selten ändern werden, sind schreibende Zugriffe in der Minderzahl. Auch die Anzahl an gleichzeitigen Benutzern für lesende Zugriffe ist um ein vielfaches höher als für schreibende Zugriffe.

Die Zugriffe auf das Repository sind mittels REST-Technologie [‡] zu realisieren. Durch Verwendung des URI-Standards [§] werden Vorlagen in das Repository kopiert, bearbeitet, gelöscht und gelesen. Dies hat zur Folge, dass der Webserver, die die Anfragen entgegen nimmt, entsprechend abgesichert sein muss.

Dependability

Da die Template-Engine über keine direkte Endbenutzerschnittstelle verfügt, ist die Verfügbarkeit an die Software geknüpft, die mit der Template-Engine kommuniziert. Es gelten also die gleichen Parameter wie für die übrigen Module trueAct TICKET, trueAct MAIL und trueAct ADMIN [¶]

[‡] Representational State Transfer

[§] http://tools.ietf.org/html/rfc1630

[¶] Produkte, die ebenfalls von der Firma Pidas entwickelt werden.

Maintenance

Für Wartungsarbeiten ist vorzusehen, dass die Template-Engine unabhängig von den anderen Produkten wartbar ist. Durch Einsatz eines entsprechenden Applikationsservers ist dies sichergestellt. Die unterschiedlichen Anwendungen lassen sich getrennt voneinander herunterfahren. Somit stören Wartungsarbeiten einzelner Module den Betrieb anderer Module nicht.

Platform

Die Template-Engine ist in der Sprache Java [‖] zu entwickeln. Als Applikationsserver wird JBoss [**] verwendet. Durch die Verwendung einer Virutal Machine bleibt die Frage des Betriebssystems dem Kunden überlassen. Bevorzugt wird die Windows-Plattform.

Documentation

Für die Implementatoren der Software ist eine Dokumentation der Template-Engine zu erstellen. Der Inhalt soll folgende Puntke umfassen:

- Voraussetzungen einer Installation für die Template-Engine
- Installation der Template-Engine
- Wartung der Template-Engine
- Update der Template-Engine
- Notwendige Benutzername und Passwörter für die Installation

‖http://java.sun.com/
**http://www.jboss.org/

Kapitel 7

Zusammenfassung

Ausgehend von einem nicht optimalen Softwareentwicklungsprozess wurde das Domain Engineering als neuer Ansatz zur Verbesserung der in Zukunft zu entwickelnden Software ausgewählt. Die bisherige Vorgehensweise der Softwareentwicklung war mit ständigen Änderungswünschen der Kunden konfrontiert. Dies führte im Laufe der Zeit zu einer mosaikartigen Software, welche zwar am Anfang gut durchdacht und geplant war, allerdings auf Änderungen und Wünsche der Kunden nicht flexibel genug reagieren konnte. Für ausführliches Refactoring stand oft keine Zeit mehr zur Verfügung, da durch sehr aufwendige Tests, die großteils manuell durchgeführt wurden, die bestehende Funktionalität sichergestellt werden musste.

Die eigentliche Ursache, die von sehr vielen Softwareentwicklern als normal hingenommen wird, sind Anpassungen und Korrekturen der Initialversion ausgehend vom Kunden, nachdem er die Software für eine gewisse Zeit im Einsatz hatte. Gemeint sind hier Änderungen die eigentlich schon von Beginn an klar sein hätten müssen, wenn man die Domäne des Kunden analysiert hätte.

Oft wird im laufenden Betrieb festgestellt, was an der schon fertigen Software noch geändert und angepasst werden muss. Einige Abläufe sind doch anders als in den Requirements festgelegt. Es wird doch noch eine neue Rolle benötigt, die zu Beginn nicht vorgesehen war. All diese Änderungen werden dem Kunden erst bewusst, nachdem er sich richtig in die Anwendung der Software hinein versetzt hat. Doch eigentlich sollten diese „neuen“ Requirements schon von Anfang an definiert gewesen sein.

Dies ist jedoch nicht mit der Erweiterbarkeit einer Software zu verwechseln. Das Anpassen und Erweitern der Software in einem geplanten Rahmen zeichnet eine gute Software aus. Es sollte jedoch von Beginn an klar sein, in welchen Stufen die Software erweitert und neue Funktionalität hinzugefügt werden soll. Weiters sollte das Grundgerüst oder der Grundstock des Sourcecodes solche Anpassungen ohne Probleme erlauben.

7.1. Der Triptych-Ansatz

Abhilfe soll der sogenannte Triptych-Ansatz schaffen: Bevor ein Softwaredesign erstellt werden kann, müssen die Requirements bekannt sein. Bevor die Requirements festgelegt werden, muss die Domäne bekannt sein. Der Begriff *Triptych* leitet sich aus den drei großen Schritten, in denen die Softwareentwicklung als Ingenieurdisziplin laut D. Bjørner untergliedert ist, ab:

1. Domain Engineering
2. Requirements Engineering
3. Software Design

Bjørner behauptet, dass auch die Softwareentwicklung als Ingenieurdisziplin zu betrachten ist. Er weißt darauf hin, dass andere Disziplinen wie zum Beispiel die Autoindustrie oder die Mobiltelefonindustrie sehr viel Domänenwissen besitzen, um überhaupt erfolgreich zu sein. Es reicht nicht aus zu wissen, wie man ein Auto zusammenbaut oder ein Mobiltelefon herstellt. Es ist auch notwendig, Wissen in den Bereichen Thermodynamik, Mathematik oder elektromagnetische Feldtheorie zu besitzen um erfolgreich zu sein. Genau dies ist der Kritikpunkt von Bjørner an den meisten Softwareentwicklungsprozessen. Software Ingenieure beschränken sich „nur“ darauf, die geforderte Software zu entwickeln ohne die gesamte Domäne, in der die Software zum Einsatz kommen soll, zu betrachten.

Deswegen soll die Entwicklung einer Software auch mit der Analyse der Domäne beginnen, in der die Software zum Einsatz kommt. Ein breites Wissen über die Vorgänge und Abläufe innerhalb der Domäne soll vor dem Festlegen der Requirements angesammelt werden. Für das Sammeln des Wissens gibt es viele Möglichkeiten. Dazu zählen das Lesen von domänenspezifischer Literatur, Recherche im Internet, Fragebögen, Interviews mit den Mitarbeitern in der Domäne sowie das Miteinbeziehen von Schlüssel-Mitarbeitern die über gutes Domänenwissen verfügen.

7.2. Domain Description Units

Ein erstes Ergebnis der Analyse der Domäne sind die sogenannten Domain Description Units (siehe 3.2.1). Dabei wird festgehalten, um welche Art der Beobachtung es sich handelt, die Quelle aus der die Domain Description Unit stammt, das Datum und ein eindeutiger Name der Unit. Die Art kann eine von vier möglichen sein:

- Entität
- Funktion
- Ereignis
- Verhalten

Entitäten bilden den Grundstock der Domäne und bezeichnen Objekte innerhalb dieser. Entitäten haben Attribute die die Entität genauer beschreiben. *Funktionen* können Attribute in den Entitäten, und somit deren Zustand, verändern. *Ereignisse* lösen wiederum Funktionen aus. *Verhalten* ist die Kombination aus Ereignissen und Funktionen innerhalb einer Domäne.

Um diese Domain Description Units zu erhalten, wurde im Verlauf dieser Arbeit ein Fragebogen erarbeitet und an zuvor identifizierte Stakeholder der Domäne verteilt. Der Fragebogen (siehe 5.3.3) bestand aus fünf Fragen deren Beantwortung einen guten Einblick in die Domäne verschaffen sollte. Es stellte sich jedoch heraus, dass das Konzept des Domain Engineerings sehr abstrakt ist und vielen Stakeholdern nicht begreiflich gemacht werden konnte. Von den verteilten 15 Fragebögen wurden lediglich fünf ausgefüllt. Daraus lassen sich folgende Punkte ableiten:

1. Es muss ein klares Commitment von allen beteiligten Stakeholdern, vor allem aber vom Management, zum Domain Engineering geben.

2. Es muss den Mitarbeitern Zeit zur Verfügung gestellt werden, um mit dem Domänen Engineer zusammenzuarbeiten.
3. Das Konzept *Domain Engineering* muss ausreichend greifbar und verständlich gemacht werden.

Weiters muss sehr viel Zeit in die Argumentation investiert werden, warum das Domain Engineering einen wesentlichen Vorteil gegenüber „herkömmlicher“ Softwareentwicklung hat. Der zu Beginn des Projektes höherer Aufwand muss dem Nutzen im weiteren Projektverlauf gegenübergestellt werden. Das umfassende Wissen über die Domäne und deren Abläufe stellt für die Entwicklung der Software einen wesentlichen Vorteil dar.

Um trotz dieser Schwierigkeiten einen guten und brauchbaren Überblick über die Domäne zu bekommen, wurden zusätzlich zu den Fragebögen Interviews durchgeführt. Der Vorteil dieser Interviews war, dass diese während des normalen Entwicklungszyklus der Ticketingsoftware durchgeführt werden konnten. Change-Requests die von Anwendern der Ticketingsoftware vorgeschlagen wurden, wurden zum Anlass genommen, um etwas mehr über die Hintergründe der gewünschten Änderung zu erfahren. Die Ursache, warum ein Change-Request gewünscht wurde, gab einen sehr guten Einblick in die Abläufe innerhalb der Domäne. So konnte im Laufe der Zeit auch ein sehr gutes Bild der Domäne erstellt werden.

7.3. Domänen Modellierung

Nun war es möglich, die Domäne aus den von Bjørner geforderten Blickwinkel zu betrachten (siehe 3.3.7). Diese Betrachtungsweisen sind:

- Intrinsics
- Support Technologies
- Management and Organisation
- Rules and Regulations
- Scripts
- Human Behaviour

Intrinsics stellen den Grundstock der Domäne dar ohne die die Domäne nicht verstanden werden kann. *Support Technologies* beschreibt die verwendeten Technologien, die von Entitäten und Funktionen genutzt wird. *Management and Organisation* stellen zum einen die strategische Ausrichtung der Domäne dar und zum anderen die organisatorische Abbildung der Domäne. *Rules and Regulations* beschreiben die festgelegten Regeln, an denen sich Mitarbeiter bei ihrer täglichen Arbeit richten müssen. Weiters wird auch das Vorgehen beschrieben, wenn solche Regeln nicht eingehalten und verletzt werden. *Scripts* beschreiben die rechtlichen Rahmenbedingungen und Verträge, die innerhalb einer Domäne vorliegen. *Human Behaviour* beschreibt die Ausführung der Arbeit innerhalb der Domäne und berücksichtigt dabei die individuellen Arbeitseinstellungen der Mitarbeiter.

7.4. Domänenvalidierung und -verifizierung

Die Validierung der Domäne soll sicherstellen, dass auch die richtige Domäne untersucht wird. Dies konnte durch die laufenden Interviews die parallel zum Entwicklungszyklus durchgeführt wurden, sichergestellt werden.

Die Verifizierung der Domäne wurde mit Hilfe der Prädikatenlogik durchgeführt. Dazu wurden die gefundenen Domain Description Units in Prädikatenlogik übergeführt und mit Hilfe von Prover9 verifiziert (siehe 5.9) McCune (2010).

7.5. Requirements

Die Requirements wurden dann aus den einzelnen Domain Description Units abgeleitet. Die Überführung von Domain Description Units in Requirements kann unter Zuhilfenahme folgender Operationen durchgeführt werden:

- Projection
- Determination
- Instantiation
- Extension
- Fitting

Projection sorgt dafür, dass nur mehr relevante Domain Description Units betrachtet werden. Der Umfang der betrachteten Domäne ist meist größer als das Einsatzgebiet der Software. Durch *Determination* sollen Domain Description Units ein vorhersehbares und vorherbestimmtes Verhalten bekommen. Der Schritt *Instantiation* legt für die Units ein konkretes Umfeld fest, in dem sie eingesetzt werden. Durch *Extension* werden neue Entitäten, Funktionen, Ereignisse und Verhaltensweisen festgelegt, die zuvor in der Domäne nicht möglich waren und durch die neue Software dann aber Teil der Domäne sein werden. *Fitting* ermöglicht das Teilen von Requirements über mehrere Projekte hinweg, die an der selben Domäne, einer angrenzenden oder einer sich überschneidenden Domäne arbeiten.

7.6. Erkenntnisse

Domain Engineering stellt einen sehr guten Ansatz dar, wenn Software in einer neuen, nicht bekannten Domäne entwickelt werden muss. Der sehr wissenschaftliche Zugang zur Erstellung der Software muss durch oft schwierige Diskussionen argumentiert werden. Der Nutzen, der durch die umfassenden Analyse der Domäne entsteht, ist nicht immer gleich ersichtlich. Der Vorteil, den das Domain Engineering mit sich bringt, wird nicht von allen beteiligten Stakeholdern als groß genug angesehen, um die zusätzlichen Aufwände zu rechtfertigen. Deswegen ist es wichtig Domain Engineering von Beginn an klar zu positionieren und die Vorteile hervorzustreichen. Die genau Darstellung der Domäne kann nicht nur für die Entwicklung von Software herangezogen werden. Fragen zu Domäne können mit Hilfe der Domänenbeschreibung beantwortet werden.

Wird eine neue Software (COTS-Software *) für eine bestimmte Branche entwickelt ist Domain Engineering eine ausgezeichnete Wahl. Für das Entwickeln von Individualsoftware (Keyturn-Software) muss der zusätzliche Aufwand der zu Beginn des Projektes betrieben werden muss, dem weiteren Nutzen gegenübergestellt werden. Dem Kunden sollte der Ansatz dieser Entwicklungsmethode näher gebracht werden. Auch die Größe des Projektes und der zu entwickelnden Software stellen Faktoren dar, die berücksichtigt werden müssen.

*Commercial-Of-The-Shelf-Software

Um die Beschreibung der Domäne ausreichend genau zu erhalten, reicht es nicht aus nur Literaturrecherche zu betreiben. Die Zusammenarbeit mit Menschen die innerhalb der Domäne ihrer Arbeit nachgehen, ist unerlässlich. Der Faktor Mensch stellt den wichtigsten aber auch zugleich den schwierigsten Punkt im Domain Engineering dar. Die Stakeholder, die am Domain Engineering teilnehmen, müssen diese Herangehensweise verstehen und davon überzeugt sein. Wenn die Auskunft der beteiligten Personen nur spärlich ist, wird auch das Gesamtergebnis nur einen gewissen Teil der Domäne abdecken — und diesen eventuell nicht korrekt. Das Ziel, welches mit dem Domain Engineering verfolgt wird, muss klar und verständlich an alle Stakeholder vermittelt werden. Die Unterstützung der Stakeholder durch den Domain Engineer muss über die gesamte Dauer des Domain Engineerings sichergestellt sein.

Des weiteren stellt die formale Verifikation der Domäne oder der Requirements eine gewisse Herausforderung an das Entwicklerteam dar. Es wird eine „Übersetzung“ der formalen Mechanismen für den Großteil der der Entwicklermannschaft benötigt um die Richtigkeit der Requirements zu belegen. Die Lesbarkeit und Verständlichkeit von formalen Methoden stellt für den Großteil der Entwickler sicher ein Herausforderung dar.

Abschließend kann gesagt werden, dass Domain Engineering auf jeden Fall eine Berechtigung hat. Der Einsatz muss aber gut überlegt und kalkuliert sein. Ist es nicht möglich Domain Engineering als ganzheitlichen Ansatz zu wählen, so können einzelne Teilbereiche wie zum Beispiel die Betrachtungswinkel trotzdem gute Ergebnisse erzielen. Eine Sicht aus unterschiedlichen Richtungen trägt viel zum besseren Verständnis der Domäne bei.

Literaturverzeichnis

BJØRNER, D. 2006. *Software Engineering 3*. Springer. (Cited on pages 10, 11, 13, 17, 18, 19, 20, 21, 22, and 27.)

BJØRNER, D. 2008a. Domain requirements modelling. TU Graz Lecture Notes. (Cited on page 20.)

BJØRNER, D. 2008b. *Software Engineering*. Vol. 1: The Triptych Approach. Springer. (Cited on pages xiii, 8, 9, 11, 12, 13, 14, 15, 16, and 17.)

BJØRNER, D. 2009a. From domains to requirements on a triptych of software development. Lecture Notes, Vienna Lectures April 2010. (Cited on pages 19 and 20.)

BJØRNER, D. 2009b. Role of domain engineering in software development. *Perspectives of system informatics*. (Cited on page 8.)

BOCA, P. P., BOWEN, J. P., AND SIDDIQI, J. I. 2009. *Formal Methods: State of the Art and New Directions*, 1st ed. Springer-Verlag Gmbh. (Cited on page 15.)

DINES BJØRNER, D., OF SCIENCE, J. A. I., TECHNOLOGY, AND JAIST. 2009. *Domain engineering: technology, management, research and engineering*. COE research monography series, vol. 4. JAIST Press, Nomi, Ishikawa 923-1292, Japan. (Cited on pages 9, 15, and 17.)

EVANS, E. 2004. *Domain-Driven Design*. Addison-Wesley. (Cited on page 9.)

FRAKES, W., PRIETO-DIAZ, R., AND FOX, C. 1998. Dare: Domain analysis and reuse environment. *Annals of Software Engineering 5*, 125–141. (Cited on page 7.)

FRAKES, W. B. AND KANG, K. 2005. Software reuse research: Status and future. *IEEE Transactions on Software Engineering 31,* 7, 529–536. (Cited on page 7.)

JOST, P. 2007. Evolutionäres domain-engineering zur entwicklung von automatisierungssystemen. Forschungsbericht 2, Institut für Automatisierungs- und Softwaretechnik, Universität Stuttgart. (Cited on page 7.)

KANG, K. C., COHEN, S. G., HESS, J. A., NOVAK, W. E., AND PETERSON, A. S. 1990. Feature-oriented domain analysis (foda) feasibility study. Tech. rep., Software Engineering Institute, Pittsburgh, Pennsylvania 15213. (Cited on page 7.)

MCCUNE, W. 2005–2010. Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`. (Cited on pages 43 and 62.)

PARNAS, D. L. 1976. On the design and development of program families. *IEEE Transactions on Software Engineering SE-2*, 1–9. (Cited on pages xiii, 5, and 6.)

RUPP, C. 2007. *Requirements-Engineering und Management*. Carl Hanser Verlag München Wien. (Cited on page 13.)

SIMOS, M., CREPS, D., KLINGLER, C., LEVINE, L., AND ALLEMANG, D. 1996. Organization domain modeling (odm) guidebook. Informal technical report, Softwrae Technology for Adaptable, Reliable Systems (STARS). STARS-VC-A025/001/00. (Cited on page 7.)