Masterarbeit

Zur Erlangung des akademischen Grades eines Master of Science über das Thema

---

Integration und Weiterentwicklung bestehender Energiemanagement-Applikationen mit dem OpenResKit-Framework

---

Eingereicht am: 11. Mai 2015
An der Hochschule für Technik und Wirtschaft Berlin, Fachbereich II, Studiengang Betriebliche Umweltinformatik

Von: Johannes Boß, Matrikelnummer: 539519

1. Gutachter: Prof. Dr. Volker Wohlgemuth
2. Gutachter: Prof. Dr. Jochen Wittmann

Kurzfassung

Für kleine und mittlere Unternehmen bietet eine ressourceneffiziente Produktion eine Vielzahl von Vorteilen. Ein strukturierter Ansatz ist die Einführung eines Energiemanagementsystems, zum Beispiel nach der Norm DIN ISO 50001. Die Umsetzung wird durch den Einsatz von Informationssystemen erleichtert. Im Partnerunternehmen existiert eine solche Anwendung und zusätzliche energiebezogene Daten stehen in weiteren Informationssystemen zur Verfügung. Diese Masterarbeit hat das Ziel, alle Aufgaben des Energiemanagements in einer Anwendung durchzuführen. Dazu werden zunächst Methoden und Konzepte zur Softwareevolution und zur Datenintegration theoretisch beleuchtet und die existierenden Anwendung analysiert. Der Schwerpunkt liegt auf der praktischen Umsetzung, indem die bestehende Energiemanagementanwendung weiterentwickelt und weitere Daten darin integriert werden. Die Implementierung erfolgt mit dem an der HTW Berlin entwickelten OpenResKit-Framework. Besonders vorteilhaft erweist sich die modulare Architektur mit einer leicht anpassbaren Datenbank und die nicht proprietären Austauschformate, um die Daten clientunabhängig verfügbar zu machen. Schwerpunkt bei der serverseitigen Entwicklung sind die Migration der bestehenden Datenbank und die Integration weiterer Daten. Bei der clientseitigen Entwicklung stehen die verwendeten Architekturmuster im Fokus. Die Herausforderungen, die ein Softwareevolutions- und Integrationsprojekt mit sich bringt, konnten mithilfe des OpenResKit-Frameworks gemeistert werden. Mit der entstandenen Anwendung ist die Erledigung der Energiemanagementaufgaben in einem System möglich.

Abstract

A resource-efficient production provides a range of advantages for small and medium-sized enterprises. A structured approach is the introduction of an energy management system pursuant to ISO 50001. The implementation is faciliated by the use of information systems. This thesis aims to carry out all the activities concering the energy management in one application. At first, the methods and concepts for software evolution and data integration are taken into account and the existing application is analyzed. The focus is on the practical implementation by integrating additional data into the existing application and its further development. The implementation is based on the OpenResKit-Framework, which was developed at HTW Berlin. The modular architecture, with an easily customizable database und and the non-proprietary exchange format are providing particular advantages. The migration of an exisiting database and the data integration are the focal aspects of the server-side development. The main emphasis on the client side is the use of design patterns. The challenges of an evolutionary project could be met with the OpenResKit-Framework. The resulting application allows to complete every energy management task in a single application.

Inhaltsverzeichnis

Abkürzungsverzeichnis

DI	Dependency Injection
DWH	Data Warehouse
EAI	Enterprise Application Integration
EF	Entity Framework
EMAS	Eco Management and Audit Scheme
EnMS	Energiemanagementsystem
IEEE	Institute of Electrical and Electronics Engineers
IoC	Inversion of Control
KMU	kleine und mittlere Unternehmen
MEF	Managed Extensibility Framework
MVC	Model View Controller
MVVM	Model View View-Model
ODBC	Open Database Connectivity
ORK-Framework	OpenResKit-Framework
ORK-Server	OpenResKit-Hub
ORM	Objektrelationaler Mapper
PDCA-Zyklus	Plan-Do-Check-Act Zyklus
SOA	Service-oriented Architecture
SSIS	SQL Server Integrations Services
SSDT	SQL Server Data Tools
WCF	Windows Communication Foundation
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XP	Extreme Programming

Quellcodeverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Der Energieverbrauch in Deutschland soll bis 2050 um 50% gegenüber 2008 gesenkt werden.[1] Dies ist nur mit einer Steigerung der energiebezogenen Leistung, also der Energieeffizienz, der deutschen Produktionsunternehmen zu erreichen. Die Unternehmen können durch die eingesparten Kosten ihre Wettbewerbsfähigkeit erhöhen. Ein Energiemanagementsystem ist dafür ein geeignetes Mittel. Das Partnerunternehmen, für diese Masterarbeit, ist die NOVAPAX Kunststofftechnik GmbH & Co. KG in Berlin. Das mittelständische Unternehmen fertigt Spritzgussteile und stellt die dazu erforderlichen Formen und Werkzeuge her. Aufgrund des langjährigen Engagements im Umweltbereich, hat das Unternehmen ein Energiemanagementsystem nach der ISO Norm 50001 eingeführt. Um den Dokumentationspflichten gerecht zu werden, wird bereits auf verschiedene IT-Anwendungen zurückgegriffen. Dennoch sind noch nicht alle Anforderungen, die das Partnerunternehmen an eine Energiemanagementsoftware stellt, abgedeckt. Eine grundständige Neuentwicklung ist im betrieblichen Alltag nicht praktikabel. Es existiert eine Vielzahl von Anwendungen und dazugehörige Datenbanken, in die bereits hoher personeller und finanzieller Aufwand investiert wurde. Gefordert sind also die Weiterentwicklung der Anwendungen und die Integration verschiedener Funktionalitäten. Aus diesen Gründen leitet sich direkt die Aufgabenstellung ab.

1.2 Aufgabenstellung und Zielsetzung

Im Mittelpunkt steht die Aufgabe, wie die IT-Abteilung eines kleinen und mittleren Unternehmens mit einfachen Mitteln Integrationsfragestellungen bearbeiten kann. Anhand eines konkreten Beispiels der Umweltinformatik wird erarbeitet, wie die Weiter-

[1] vgl. [Umw12, Seite 3]

entwicklung bestehender Anwendungen und deren Integration möglich ist. Ein betriebliches Umweltinformationssystem ist nach Rautenstrauch „ein organisatorisch-technisches System zur systematischen Erfassung, Verarbeitung und Bereitstellung umweltrelevanter Daten in einem Betrieb.“[2] Nach Marx Goméz sind sie eine Klasse von Anwendungssystemen der Wirtschaftsinformatik und haben den gleichen applikationstechnischen Rahmen eines Informationssystems der Wirtschaftsinformatik.[3] Deshalb können können auch die gleichen Modelle der Wirtschaftsinformatik zum Softwarelebenszyklus angewendet werden.

Die Zielsetzung der Arbeit ist die Weiterentwicklung und Integration der bestehenden Anwendungen für das Energiemanagement. Dazu soll der gesamte Softwareentwicklungsprozess mit Analyse, Konzeption und Implementierung durchgeführt werden. Der Prozess ist nicht prototypisch durchzuführen, sondern soll direkt umgesetzt werden, da nur so auf praxisrelevante Probleme bei der Weiterentwicklung eingegangen werden kann. Aufgrund der Gegebenheiten im Partnerunternehmen und der bestehenden Kooperation mit der HTW Berlin, erfolgt die technische Umsetzung mit dem OpenResKit-Framework. Die zweite Fragestellung ist deshalb, ob dieses Framework dafür geeignet ist und welche Verbesserungspotentiale erkennbar sind. Idealerweise tragen die Forschungsergebnisse direkt zu einer Weiterentwicklung des Frameworks bei.

1.3 Aufbau der Arbeit

Die Schwerpunkte sind die Informationsintegration verschiedener heterogener Datenquellen in eine bestehende Softwareanwendung, die Weiterentwicklung und Wartung eines existierenden Softwareprodukts und die Verwendung des OpenResKit-Frameworks. Diese Themen ziehen sich deshalb wie ein roter Faden durch die gesamte Arbeit. Der Aufbau der Arbeit beginnt nach der Einleitung mit der Erläuterung der theoretischen Grundlagen sowohl aus fachlicher als auch aus informationstheoretischer Sicht. Hier wird ein kurzer Überblick über die Anforderungen an ein Energiemanagementsystem nach der Norm ISO 50001 gegeben. Weiterhin werden Konzepte zur Weiterentwicklung einer Anwendung dargestellt. Dies ist an erster Stelle das Thema Softwareevolution, in dem auf verschiedene Softwarelebenszyklen und Tech-

[2] vgl. [Rau99, Seite 11]
[3] vgl. [LNMG+11, Seite 3]

niken eingegangen wird. Zum Zweiten wird ein theoretischer Abriss über das Thema Informations- und Datenintegration gegeben. Einige Aspekte des angewandten Vorgehensmodells runden das Kapitel ab.

Im dritten Kapitel wird zunächst die grundlegende Methodik der angewandten Anforderungsanalyse beschrieben und die ermittelten Ergebnisse in Form von User-Stories dargestellt. Weiterhin wird eine umfassende Softwareanalyse vorgenommen, die schon ein Teil der Weiterentwicklung ist. Dabei wird das verwendete OpenResKit-Framework beschrieben und die bereits damit entwickelte Software geprüft. Aufgrund des Schwerpunktes der Informationsintegration gibt dieses Kapitel auch eine Übersicht der Softwarelandschaft im Partnerunternehmen.

Das vierte Kapitel handelt von der Konzeption. Hier wird dargestellt, welche Module der vorhandenen Anwendung erweitert und welche neu entwickelt werden müssen. Darüber hinaus wird die Vorgehensweise der Datenintegration näher erläutert. Mit dem Entwurf der Benutzeroberflächen und der Überarbeitung der Gestaltung schließt das Kapitel ab.

Den Kern des Projekts bildet die Implementierung. Die entwickelte Software ist beim Partnerunternehmen im produktiven Einsatz, weshalb die Implementierung nicht nur prototypisch vorgenommen wurde. Dies macht die Beschreibung einiger zusätzlicher interessanter Aspekte erforderlich. Das Kapitel beinhaltet zunächst serverseitige Aspekte der Implementierung. Die wichtigsten Themen sind die Weiterentwicklung der Datendomäne und der Migration der Produktionsdatenbank, sowie der Zugriff auf externe Datenquellen. Bei der clientseitigen Entwicklung liegen die Schwerpunkte bei den verwendeten Entwurfsmustern und der Oberflächengestaltung. Die neu entwickelte Druckfunktion lässt sich durch eine enge Verknüpfung mit der Oberfläche schnell realisieren. Im letzten Abschnitt wird auf die durchgeführten Tests eingegangen, die die Softwarequalität erhöhen sollen.

Die erzielten Ergebnisse werden im sechsten Kapitel beschrieben und mögliche Verbesserungen des OpenResKits und Erweiterungen der erstellten Anwendung vorgeschlagen.

2 Grundlagen

2.1 Energiemanagement ISO 50001

Aus verschiedenen Gründen ist eine ressourceneffiziente Produktion für viele Unternehmen sinnvoll, oder gar notwendig.[4] Ein vordergründiges Argument ist der Umweltschutz. Die Verringerung der Emission von Treibhausgasen ist eine Folge der ressourceneffizienten Produktion, verlangsamt den Klimawandel und damit auch die negativen Auswirkungen, wie Starkwetterereignisse oder den Anstieg der Meeresspiegel. Wirtschaftliche Vorteile sind aus unternehmerischer Sicht jedoch häufig gewichtigere Gründe. Steigende Energiekosten können durch einen sparsameren Energieverbrauch abgefedert werden. Durch die planmäßige Suche nach Verbesserungspotentialen im eigenen Energieverbrauch und die Investition in energieeffiziente Technologien, sinken die Energieverbräuche im Unternehmen. Auch der Staat unterstützt Unternehmen, die helfen, die energiepolitischen Ziele Deutschlands zu erreichen, mit verschiedenen Fördermaßnahmen. Hier ist die Befreiung von der Umlage nach dem Erneuerbare-Energien-Gesetz (EEG-Umlage) für energieintensive Unternehmen zu nennen[5], oder die Steuervergünstigung zum Spitzenausgleich im Rahmen der Energie- und Stromsteuer, zu deren Voraussetzungen ein Energiemanagement im Unternehmen gehört[6]. Die Anstrengungen zur Energieeffizienz können nach außen wirksam durch eine Zertifizierung dargestellt werden. Dies kann für die öffentliche Wirkung generell sinnvoll sein, oder durch einen Auftraggeber explizit gefordert werden. Die strukturierte Analyse und Verbesserung der eigenen Unternehmensprozesse, die oftmals von externen Fachleuten begleitet wird, ist ein Nebeneffekt dieser Anstrengungen.

Ein wesentlicher Faktor zur Verwirklichung der Ressourceneffizienz im Unternehmen ist die Einführung eines Energiemanagementsystem (EnMS). „Ein EnMS dient der

[4] vgl. [VDI11, Seiten 16ff.]
[5] vgl. § 64 Abs. 1 EEG 2014 [Bun14]
[6] vgl. § 10 Abs. 3 StromStG [Bun12]

systematischen Erfassung der Energieströme und als Basis zur Entscheidung für Investitionen zur Verbesserung der Energieeffizienz."[7] Weit verbreitet ist die internationale Norm ISO 50001. Daneben gibt es die Norm ISO 14001 und das Eco Management and Audit Scheme (EMAS), die sich aber auf das generelle Umweltmanagement konzentrieren. Alle Systeme sind Managementnormen und eignen sich auch für kleine und mittlere Unternehmen (KMU).[8] Sie basieren auf einem kontinuierlichen Verbesserungsprozess und dem Plan-Do-Check-Act Zyklus (PDCA-Zyklus). Nachfolgend wird die ISO 50001 betrachtet, die 2011 eingeführt wurde, und verschiedene nationale und europäische Vorgängernormen, zum Beispiel DIN EN 16001, vereinheitlicht. Diese Norm wird, besonders in Deutschland, immer wichtiger. Im Jahr 2013 gab es insgesamt eine Steigerung der Zertifizierungen von 116 Prozent, wovon die meisten Unternehmen aus Deutschland kamen.[9] Die wichtigsten Aspekte eines EnMS sind die Bereiche Organisation, Dokumentation, Information und Überwachung. Die einzelnen Phasen im PDCA-Zyklus sind im Kontext des Energiemanagements folgende:[10]

Plan (Planung) In der Planungsphase werden zunächst die Energiedaten erfasst und daraus Energieleistungskennzahlen (EnPI - Energy Performance Indicator) gebildet. Dazu kann man Bestandsdaten analysieren, oder Messungen durchführen. Diese Daten sind der Ausgangspunkt für die Entwicklung der strategischen und operativen Energiesparziele. Um diese Energiesparziele zu erreichen, werden Maßnahmen entwickelt und in Form von Aktionsplänen dokumentiert.

Do (Einführung/Umsetzung) Die Umsetzung der Aktionspläne erfolgt in der „Do-Phase". Eine übergreifende Tätigkeit ist die Etablierung des Energiemanagements im gesamten Unternehmen. Dazu werden Managementstrukturen geschaffen und die Bewusstseinsbildung aller Mitarbeiter, zum Beispiel durch Schulungen, gefördert.

Check (Überprüfung) Die durchgeführten Maßnahmen werden überwacht, indem man Energiedaten misst und die daraus abgeleiteten Energieleistungskennzahlen überprüft. Die Ergebnisse werden ausreichend dokumentiert.

Act (Verbesserung) Aus den Daten der vorangegangenen Phase wird die Effektivität beziehungsweise die Wirksamkeit der durchgeführten Maßnahmen hinsichtlich der Energieziele durch das Management bewertet. Daraus können sich neue

[7] siehe [Umw12, Seite 16]
[8] vgl. [Umw12, Seite 10]
[9] vgl. [ISO14, Seite 4]
[10] vgl. [ISO11, Seite 5][Umw12, Seiten 20ff.]

Energieziele und weitere Aktionspläne ergeben, die im weiteren Durchlauf des Zyklus bearbeitet werden.

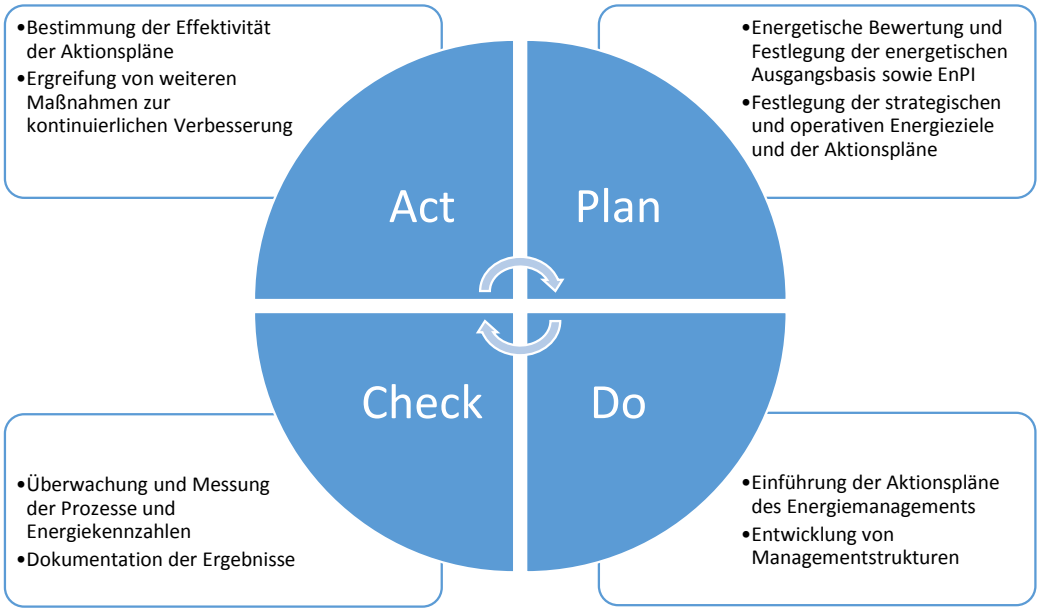

Abbildung 1: PDCA-Zyklus auf das Energiemanagement bezogen[11]

In diesem dynamischen Modell sind die Ergebnisse eines Zyklus Ausgangspunkt für den nächsten Zyklus. Die einzelnen Phasen können für verschiedene Bezugsobjekte parallel durchgeführt werden. Alle Ebenen und Funktionen eines Unternehmens sind verantwortlich für den Erfolg eines EnMS. Besondere Verantwortung trägt die oberste Leitungsebene, also das Top-Management. Dieses legt die Energiepolitik fest und stellt die benötigten Ressourcen im Bereich Personal, sowie technische und finanzielle Mittel bereit. Die Energiepolitik legt die langfristigen Gesamtziele im Unternehmen fest und verpflichtet das Top-Management zur kontinuierlichen Steigerung der Energieeffizienz und zum sparsamen Umgang mit Energie. Das Top-Management benennt einen oder mehrere Managementbeauftragten oder Energiemanager, die entsprechende Befugnisse zur Durchsetzung der Energieziele innehaben.

Die Norm beinhaltet lediglich Leitlinien. Die konkrete Ausgestaltung und praktische Umsetzung bleibt dem Unternehmen überlassen. Der Softwareeinsatz ist nicht vorgeschrieben, bietet sich aber in vielen Bereichen an. In einem softwarebasierten EnMS können typischerweise die folgenden Aufgaben erfüllt werden:[12]

[11]Eigene Abbildung basierend auf [ISO11, Seite 5]
[12]vgl. [BAF15, Seite 2]

- Datenauswertung
- Visualisierung
- Berichtswesen
- Integration in bestehende Systeme
- Alarme

2.2 Konzepte zur Weiterentwicklung der Anwendungslandschaft

Die IT-Abteilung hat die Aufgabe, die Geschäftsprozesse im Unternehmen zu unterstützen. Durch sich immer schneller verändernde Gegebenheiten, Technologien und Märkte wird sie vor große Herausforderungen gestellt, da sie die neuen Anforderungen der Fachabteilungen mit den passenden Anwendungen unterstützen muss. Es gibt unterschiedliche Herangehensweisen, diesem Problem zu begegnen. Häufig wird ein Standardprodukt eingeführt, oder eine neue Anwendung in Auftrag gegeben, die ausschließlich die neu entstandenen Anforderungen abdeckt. Diese Variante ist zwar meist günstig und funktional, vergrößert aber sukzessive die Komplexität der IT-Landschaft im Unternehmen.[13] Eine weitere Option ist eine radikale Neuentwicklung der Anwendungslandschaft einer Abteilung. Für diesen „Greenfield-Approach“ steht aber häufig nicht genug Geld zur Verfügung. Laut Huber ist die erfolgversprechendste Option „bestehende Anwendungen durch Erweiterungen oder Änderungen an neue Anforderungen anzupassen.“[14] Insbesondere dann, wenn es sich um eine Eigenentwicklung handelt und noch ausreichend Wissen im Unternehmen verfügbar ist. In der Literatur wird der dieser Ansatz häufig als Softwareevolution bezeichnet[15]. Die Konzepte der Informations- und Datenintegration helfen, die Komplexität der Anwendungslandschaft zu reduzieren.

[13] vgl. [Tie13, Seite 13]
[14] siehe [Hub14, Seite 33]
[15] vgl. [Sne12, Seite]

2.2.1 Softwareevolution

Nach der erstmaligen Auslieferung eines Softwareproduktes, ist die Entwicklung noch nicht abgeschlossen. Die Tätigkeiten, die danach erfolgen, werden in der Softwaretechnik mit dem Begriff Softwareevolution beschrieben. Weitere Begriffe sind Wartung, Pflege oder Veränderungsmanagement. Die Benennung der einzelnen Prozesse ist auch abhängig vom verwendeten Vorgehensmodell und der Betrachtungsweise. Bei agilen Vorgehensweisen wird jeder Evolutionsprozess als Entwicklung einer neuen Softwareversion gesehen[16], weswegen die Begriffe Wartung oder Pflege sich hier nicht eignen. Auf die Besonderheiten im Zusammenhang mit agilen Vorgehensweise wird im Abschnitt 2.3 detaillierter eingegangen.

Einordnung der Begriffe

Jedes Softwaresystem durchläuft, nach gängigen Phasenmodellen, einen Softwarelebenszyklus. Bei Balzert[17] hat der Zyklus fünf Phasen und wird in Spezifikation, Entwurf, Implementierung, Installation und Betrieb unterteilt und ist weit verbreitet. Die Phase Betrieb wird nochmal weiter unterteilt in Wartung und Pflege.[18] Während der Wartung werden Fehler gesucht und behoben, sowie Leistungsverbesserungen vorgenommen. Die Softwarequalität wird damit erhöht. Unter den Begriff Pflege fallen Änderungen und Erweiterungen im laufenden Betrieb. Im Gabler Wirtschaftslexikon gibt es im Softwarelebenszyklus acht Phasen und die Anpassungen werden bei der Softwarewartung vorgenommen.[19] In der angloamerikanischen Literatur wird üblicherweise der Begriff Maintenance verwendet. So auch beim Institute of Electrical and Electronics Engineers (IEEE)[20] Viele dieser Prozessmodelle fokussieren sich aber auf die erstmalige Entwicklung und behandeln die Weiterentwicklung dementsprechend nur am Rande. Angesichts der Tatsache, dass der Betrieb und Wartung den Großteil der Zeit und finanziellen Ressourcen beansprucht, ist ein umfassenderer Blick notwendig. So kann der Wartungsanteil bis zu 80% des Aufwands ausmachen und 40% davon nur, um die Software zu verstehen.[22] Auf diesen Aspekt gehen beispielsweise Bennett et al.[23] näher

[16] vgl. [SS14, Seiten 15 u. 16]
[17] vgl. [Bal11, Seiten 1 ff.]
[18] vgl. [Bal11, Seiten 533 ff.]
[19] vgl. [Lac15]
[20] vgl. [BF14, Seite 5-1]
[21] entnommen aus: [Bal11]
[22] vgl. [Bal11, Seite 2]
[23] vgl. [BRW02]

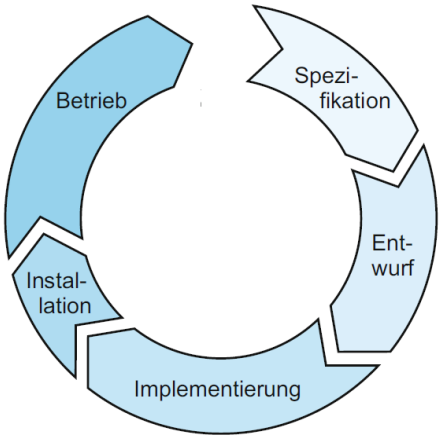

Abbildung 2: Lebenszyklus eines Softwaresystems nach Balzert[21]

ein, die das Staged Model of the Software Lifecycle entwickelt haben. Es beschreiben fünf Phasen:

Entwicklungsphase: Die erste lauffähige Version des Softwareproduktes wird entwickelt.

Evolutionsphase: Die Software wird um einen größeren Funktionsumfang erweitert, oder Anforderungen korrigiert.

Servicephase: In dieser Phase werden kleinere Fehler behoben, oder minimale Funktionalitäten angepasst.

Abwicklungsphase: Hier finden keinen Serviceleistungen mehr statt und das Produkt wird für die Stilllegung vorbereitet.

Stilllegungsphase: Das Produkt wird nicht mehr verwendet und etwaige verbliebene Nutzer auf nachfolgende Software verwiesen.

In der untenstehenden Grafik sind wie bei Sommerville[24] die letzten beiden Phasen zusammengefasst, da diese sich beide auf das Ende des Lebenszyklus beziehen.

Bennett et al. bezeichnen die Evolutionsphase als Schlüsselphase. Eine initiale Entwicklung auf der grünen Wiese findet in der Praxis so gut wie nie statt. Nach einer erfolgreichen Auslieferung der ersten Version erreichen die Entwickler Rückmeldungen der Benutzer und neue Anforderungen. In dieser Phase gibt es üblicherweise mehrere

[24]vgl. [Som11]
[25]Eigene Abbildung angelehnt an [Sne12] und [Som11, Seite 236]

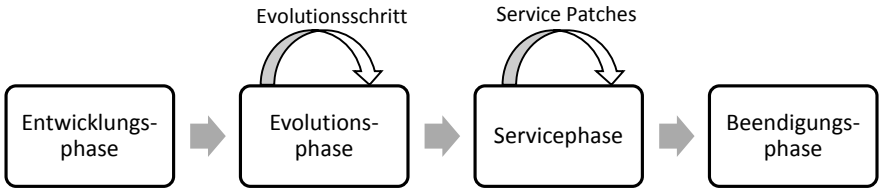

Abbildung 3: Staged Model of the Software Lifecycle[25]

neue Versionen. Deren Organisation ist von den Rahmenbedingungen abhängig und kann in Form von Upgrades oder neuen Haupt- oder Nebenversionen stattfinden.

Gesetze der Softwareevolution

Lehman und Belady[26] führten schon seit den 80-er Jahren empirische Studien durch, um mehr über die Charakteristiken der Softwareevolution zu erfahren. Daraus entstanden acht Gesetze der Softwareevolution.[27]

- Kontinuierlicher Wandel: Eine Software muss sich regelmäßig den wechselnden Anforderungen anpassen, oder die Nützlichkeit sinkt.
- Zunehmende Komplexität: Mit steigender Größe der Anwendung, wird auch die Programmstruktur komplexer. Es muss zunehmend auf den Erhalt und Vereinfachung der Struktur geachtet werden.
- Selbstregulation: Der Prozess der Softwareevolution ist Selbstregulierend. Kennzahlen wie Releasegröße, Zeit zwischen den Releases und Fehleranzahl bleiben nahezu gleich.
- Erhaltung der organisatorischen Stabilität: Über den gesamten Lebenszyklus bleibt die Entwicklungsgeschwindigkeit, unabhängig von den eingesetzten Ressourcen ungefähr gleich.
- Erhaltung der Vertrautheit: Langfristig betrachtet bleibt der inkrementelle Zuwachs in jeder Version über den gesamten Lebenszyklus ungefähr konstant.
- Anhaltendes Wachstum: Um die Nutzerzufriedenheit zu gewährleisten, muss die Funktionalität kontinuierlich steigen.

[26] vgl. [Leh80] und [LRW+97]
[27] vgl. [Som11, Seite 241]

- Sinkende Qualität: Solange ein System nicht systematisch gepflegt und an Änderungen der Umgebung angepasst wird, sinkt die Qualität
- Feedback-System: Evolutionsprozesse sind Feedback-Systeme und müssen als solche behandelt werden, um signifikante Verbesserungen des Produktes zu erhalten.

Techniken der Softwareevolution

Es gibt einige Techniken die bei Evolutionsprojekten angewendet werden und nachfolgend umrissen werden.[28]

Softwareanalyse: Zunächst muss der Versuch unternommen werden, den Quellcode zu lesen und zu verstehen. Eine Dokumentation der Architektur kann hilfreich sein, ist aber oft nicht vorhanden. Es gibt einige Tools, die die Codeanalyse unterstützen und viele Entwicklungsumgebungen verfügen mittlerweile über solche Werkzeuge. Beispielsweise bietet die Entwicklungsumgebung Visual Studio das Werkzeug Code Map an um Abhängigkeiten zu visualisieren.

Reengineering: Unter Reengineering versteht man die Überprüfung und Verbesserung kompletter Komponenten. Die Funktionalität bleibt vollständig erhalten. Ziel ist, mittels einer verbesserten Organisation des Codes die Wartbarkeit zu verbessern. Teil des Reengineerings ist das Refactoring, das kleinere Teileinheiten des Quellcodes (Methoden, Variablen) umfasst und mit modernen Entwicklungswerkzeugen (teil-)automatisiert durchgeführt werden kann.

Reverse Engineering: Als Reverse Engineering wird die Analyse eines Programms mit dem Ziel einer vollständigen Neuentwickelung ganzer Komponenten bezeichnet. Gründe können die Verbesserung der Softwarequalität, die Anpassung an neue Plattformen, oder das Ersetzen von Altsystemen (Legacy Code) sein. Auch hier bleibt die Funktionalität gleich. Von steigender Bedeutung ist das Reverse Engineering von Datenbanken. Hierbei wird auf Basis einer physischen Datenbank das Datenbankschema ermittelt. Eng mit dem Reverse Engineering verbunden ist auch die Nachdokumentation. Auch bei dieser Technik können moderne Entwicklungswerkzeuge sehr hilfreich sein.

Migration: Unter Migration wird die Portierung einer Software, oder Teile davon, auf

[28] vgl. [BF14]

andere Plattformen verstanden. Hierbei kann zum Beispiel die Änderung einer unterliegenden Datenbank oder der Ersatz eines nicht mehr unterstützen Frameworks sein.

Retirement: Am Ende des Softwarelebenszyklus steht das Abschalten der Software. Hier müssen die Auswirkungen überprüft werden, wie zum Beispiel die Zugänglichkeit von Archivdaten, und der Ersatz sichergestellt sein. Das Auslaufen einer Software ist eng mit der Migration verknüpft.

2.2.2 Informations- und Datenintegration

Wie in der Einführung des Abschnitts bereits beschrieben, findet man im Unternehmensalltag oftmals eine fragmentierte, heterogene Anwendungslandschaft vor.[29] Eine Vielzahl von Eigen- und Insellösungen, häufig realisiert mit Standardprodukten, wie Microsoft Office, beherrschen die IT-Landschaft. Die Daten liegen in unterschiedlichen Formaten, Medien und Applikationen vor, und stehen für einen unternehmensweiten Gebrauch nicht zur Verfügung.[30] Konsolidierungsprojekte in den Bereichen Applikationen und Daten können helfen, die Heterogenität der Anwendungen und Systeme aufzulösen. Das IT-Management wird vereinfacht, die Qualität der Anwendungslandschaft erhöht und Kosten reduziert.[31] Ein Mittel ist die Überführung von einzelnen Datensilos zu einer möglichst unternehmensweiten Informationssicht. Somit lassen sich Mehrfacherfassungen und Datenredundanzen vermeiden.[32] Bei der Anwendungsentwicklung muss außerdem auf eine verstärkte Geschäftsprozessorientierung geachtet werden. Zusätzlich sollte der Abbau von Komplexität innerhalb der Applikationen Teil der Konsolidierungsstrategie sein. Beispielsweise werden nur Funktionalitäten angeboten, die im Unternehmensalltag tatsächlich benötigt werden. Weiterhin wird die Verwendung von standardisierten Bausteinen empfohlen.

Im Zusammenhang mit solchen Konsolidierungsprojekten wird häufig von Informations- und Datenintegration gesprochen. Jung gibt eine Übersicht über verschiedenen Definitionen.[33] Die meisten Autoren nehmen eine Differenzierung vor, ob sich die Integration durch Verbinden, oder durch Vereinigen der Datenelemente

[29] vgl. [Tie13, Seite 71]
[30] vgl. [Tie13, Seite 136]
[31] vgl. [Tie13, Seiten 129ff.]
[32] vgl. [Tie13, Seite 135]
[33] vgl. [Jun06, Seiten 36ff.]

erreichen lässt. Beim Verbinden bleiben die möglicherweise redundanten Ausgangsbestände erhalten, aber es werden logische Beziehungen zwischen den Beständen auf Anwendungsebene hergestellt. Manchmal wird diese Art auch als Interoperabilität bezeichnet.[34] Bei der Integration durch Vereinigen werden diese Datenbestände verschmolzen und Redundanzen eliminiert.

Weitere Begriffe tauchen in diesem Zusammenhang auf. Ein Data Warehouse (DWH) ist kein einzelnes System, sondern eher ein die gesamte IT-Landschaft umfassendes Konzept. Es wird von verschiedenen unternehmensinternen und -externen Datenquellen gespeist und bereitet die Daten auf. Ziel ist es nicht, über einzelne Datensätze mittels einer einheitlichen Zugriffsschicht zu verfügen, sondern durch eine weitere Abstraktionsschicht zusätzliche Informationen zu generieren.[35] Geschieht die Koppelung der einzelnen Informationsquellen nicht auf Datenebene, sondern auf Anwendungsebene, spricht man von der Enterprise Application Integration (EAI). Hierbei wird jedes eingebundene System mit einem Adapter an einen zentralen Bus angeschlossen. Die Ursprungssysteme bleiben in ihrer Funktionalität ebenfalls unangetastet. Mit dem Begriff Service-oriented Architecture (SOA) wird ein ähnliches Konzept beschrieben. Dieses Architekturkonzept orientiert sich nicht an einzelnen Anwendungen, sondern an Geschäftsprozessen.[36]

Ebene	Konzept	Techniken
Daten	Data Warehouse	Datenintegration
Anwendungen	Enterprise Application Integration	Anwendungsintegration
Prozesse	Service-oriented Architecture	Geschäftsprozessintegration

Die Schwierigkeiten einer Daten- und Informationsintegration sind in der Verteilung der Informationssysteme und deren Heterogenität begründet.[37] Die Verteilung von Informationssystemen wird durch den Gebrauch von Internettechnologien begünstigt. Sie ist erwünscht, da die Ausfallsicherheit gefördert und die Performance durch Parallelverarbeitung erhöht wird. Gerade moderne Dateisysteme wie Hadoop[38] machen sich diese verteile Architektur zu Nutze. Sie wird von großen Webanwendungen, wie Facebook oder Ebay, genutzt. Weitaus relevanter für Integrationsfragestellungen ist aber

[34] vgl. [PB06, Seite 388]
[35] vgl. [MBK+12, Seiten 50]
[36] vgl. [MBK+12, Seiten 71]
[37] vgl. [Ros13, Seiten 24ff.] und [LN07, Seiten 49ff]
[38] http://hadoop.apache.org/

die Heterogenität der Systeme. Man kann zwischen vier verschiedenen Arten unterscheiden, die in aufsteigender Reihenfolge zunehmend schwerer zu lösen sind:

Technisch: Technische Heterogenität bezeichnet Unterschiede im verwendeten Kommunikationsprotokoll (z.B. HTTP, JDBC, SOAP), in der Anfragemöglichkeit (Anfragesprache, Benutzerformulare) oder der verwendeten Anfragesprache (SQL, XQuery).

Syntaktisch: Gleiche Sachverhalte können unterschiedlich dargestellt werden. Beispiele sind die Verwendung von Kommata und Punkte als Dezimaltrennzeichen, verschiedene Codierungen von Textdateien (UTF, ANSI, ASCII), Dateitypen in verschiedenen Datenbanken oder unterschiedliche Dateiformate.

Strukturell: Strukturelle Heterogenität liegt vor, wenn gleiche Dinge verschieden modelliert werden. Dies kann durch unterschiedliche Datenbankschemata ausgeprägt sein oder durch unterschiedliche Modellierung innerhalb der Datenbank, wie zum Beispiel die Verwendung von Bitwerten als Markierung beziehungsweise Statusindikator (engl.: Flag). Häufige gibt es auch bei der Darstellung von Datumswerten oder Startwerten (0 oder 1) Unterschiede.

Semantisch: Die semantische Heterogenität bezeichnet die unterschiedliche Bedeutung eines gleich dargestellten Sachverhaltes oder vice versa die gleiche Bedeutung eines unterschiedlich dargestellten Wertes. Das betrifft insbesondere aber nicht ausschließlich die Darstellung von *NULL*-Werten, die für die Sachverhalte *unbekannt*, *nicht erfasst* oder *nicht relevant* stehen können, die Verwendung von unterschiedlichen Sprachen oder Abkürzungen und die Verwendung unterschiedlicher Maße und Skalen. Die Integration semantischer Unterschiede gehört zu den schwierigsten Aufgabenstellungen, da die Bedeutung nicht aus dem Datenbankentwurf hervorgeht und die Konsolidierung nur manuell und mit Expertenwissen möglich ist.

Eine typische Vorgehensweise zur Datenkonsolidierung ist der ETL-Prozess. ETL steht für Extract-Transform-Load also Extrahieren-Transformieren-Laden. Zunächst werden die relevanten Daten aus den Quellsystemen entnommen. Der Prozess muss nicht einmalig ablaufen, sondern kann ereignisgesteuert, periodisch oder permanent sein. In der Transformationsphase werden die Datensätze so geändert, dass sie einheitlich in das Zielsystem passen. Dieser Teilschritt ist die aufwändigste Aufgabe, da die genannten Heterogenitäten der Quellsysteme überwunden werden müssen. Die Transforma-

tionen umfassen Umrechnungen, Aggregation, Anreicherung von Daten, Standardisierungen sowie die Eliminierung von Dubletten. Anschließend werden die Daten in das Zielsystem geladen.[39]

2.3 Vorgehensmodell

Um den Gegebenheiten eines Einzelprojektes im Rahmen einer Masterarbeit gerecht zu werden, basiert die Vorgehensweise auf dem *Manifestio for agile Software Development*. Die Autoren gehören zu den bekanntesten Vordenkern der Softwareentwicklung und postulieren folgende Werte:

> „Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:
>
> - Individuen und Interaktionen mehr als Prozesse und Werkzeuge
> - Funktionierende Software mehr als umfassende Dokumentation
> - Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
> - Reagieren auf Veränderung mehr als das Befolgen eines Plans
>
> Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.“ [40]

Diese Werte schlagen sich in Vorgehensmodellen wie Scrum[41], dass von den Manifesto Autoren Sutherland und Schwaber entwickelt wurde, Kanban[42], Extreme Programming (XP)[43] oder der Crystal Familie von Alistair Cockburn[44] nieder. Alle Modelle betonen auch Aspekte der Zusammenarbeit und Interaktion innerhalb des Projektteams oder geben eine Rollenverteilung vor. In einem „Ein-Mann-Projekt“ lassen sich diese nicht anwenden.

[39] vgl. [Ros13, Seiten 37 - 40]
[40] siehe: [BCF+14]. Am Rande sei erwähnt, dass die Autoren das Kopieren ihres Manifestes begrüßen, allerdings nur in ihrer vollständigen Form erlauben. Da letzter Satz in den meisten Zitationen oft ausgelassen wird, entsteht der Eindruck, dass die Werte auf der rechten Seite unwichtig seien, was aber nicht der Meinung der Autoren entspricht.
[41] siehe: [SS14]
[42] siehe: [LK13]
[43] siehe: [Bec00]
[44] siehe: [Coc04]

Ein missverstandenes Thema ist immer wieder die Dokumentation.[45] Ein Prinzip des agilen Manifestes schätzt eine funktionierende Software höher ein als eine umfassende Dokumentation. Richtig ist, dass mit einer funktionierenden und intuitiven Software ein Benutzerhandbuch überflüssig wird. Ebenso ist die Kommentierung des Quellcodes überwiegend unnötig und deutet eher auf einen schlechten Quellcode hin. In einigen Fällen ist die Quellcodekommentierung nützlich, beispielsweise um ein Argument oder Rückgabewert klarzustellen, wenn eine Standard-Bibliothek verwendet wird.[46] Meistens jedoch werden Kommentare durch eine wohlüberlegte Restrukturierung (Refactoring) überflüssig. Allerdings darf eine Dokumentation nicht außer acht gelassen werden. Auch nach agilen Vorgehensweisen, zum Beispiel die Crystal Family[47], wird empfohlen, gewisse Entscheidungen festzuhalten. Das können Oberflächenentwürfe, die Systemarchitektur, oder das Domänenmodell sein. Die Entscheidung, was dokumentiert wird, bleibt letztendlich den Entwicklern und Auftraggebern überlassen. Dies dient insbesondere neuen Entwicklern sich einzuarbeiten, aber auch den Teammitgliedern, die schon länger am Projekt beteiligt sind, alte Entscheidungen zu reflektieren.

In Zusammenarbeit mit den zukünftigen Anwendern der Software werden laufend die Anforderungen erfasst und in Form von User-Stories festgehalten. User-Stories beschreiben die Anforderungen der Benutzer. Auch technische Anforderungen können aus Sicht der Entwickler erfasst werden und heißen dann Technical Stories. Diese nichtfunktionalen Anforderungen können auch in die Akzeptanzkriterien einfließen. Mittels Themes oder Epics werden die einzelnen User-Stories gruppiert. In einer weiteren Abstraktionsschicht kann man eine Product Vision formulieren. Eine User-Story ist meist nach folgendem Schema aufgebaut: „Als *Rolle* möchte ich *Funktion* oder *Eigenschaft*, um *Nutzen*. In manchen Fällen werden die verschiedenen Benutzertypen in Personas gruppiert. Das sind typische Benutzerrollen, die häufig vorkommen. Das können einerseits Anwender verschiedener Fachabteilungen sein, oder Anwender mit unterschiedlichen IT-Erfahrungen. Um die Fertigstellung einer Anforderung zu verifizieren, gibt es die Definition of Done. Hierbei werden, neben der eigentlichen Anforderung, nichtfunktionale Anforderungen oder Qualitätskriterien die erfüllt sein müssen, definiert. Das kann zum Beispiel eine Testabdeckung sein, oder die Abnahme durch den Kunden.

Da bei agilen Projekten nach jedem Inkrement ein lauffähiges Teilprodukt existiert, gibt es Parallelen zur Softwareevolution. Die Phase der initialen Entwicklung fällt also

[45] vgl. [RS14, Seite 61]
[46] vgl. [Mar09]
[47] vgl. [Coc04, Seite 193]

dementsprechend kurz aus. Ein agil durchgeführtes Projekt kann man als drei Projekte in einem verstehen:[48]

- Entwicklungsprojekt: Erstellung neuer Komponenten
- Evolutionsprojekt: Ausbau und Integration neuer Komponenten
- Wartungsprojekt: Anpassung und Korrektur alter Komponenten

Der größte Unterschied zur Softwareevolution ist, dass agile Vorgehensmodelle von einem durchgehenden Entwicklerteam ausgehen und den Wissensverlust, der bei der Weiterentwicklung durch ein anderes Team auftritt, nicht berücksichtigt.

[48] vgl. [SS13, Seite 11]

3 Analyse

Im folgenden Kapitel werden die Methodik der vorgenommenen Anforderungsanalyse und deren Ergebnisse beschrieben. Diese Masterarbeit baut auf der bereits eingesetzten Software zum Maßnahmenmanagement auf und erweitert diese. Daher werden diese Anwendung und das eingesetzte OpenResKit-Framework im Besonderen analysiert. Auch die Softwareumgebung im Bereich Energiemanagement des Unternehmens ist von Bedeutung, da hieraus Daten entnommen werden und die Funktionalität teilweise in die bestehende Anwendung übertragen werden sollen.

3.1 Anforderungsanalyse

Zentraler Bestandteil eines jeden Projekts ist die Anforderungsanalyse oder auch Requirements Engineering. „Mit einer hochwertigen Anforderungsanalyse [schafft man] eine stabile Basis für eine erfolgreiche Systementwicklung.“[49] Wie in Abschnitt 2.3 beschrieben, gibt es bei der agilen Entwicklung drei Artefakte. Im Artefakt *Product Backlog* spiegeln sich die Ergebnisse aus der Anforderungsanalyse, in Form von User-Stories, wieder. Das IEEE definiert Anforderungen folgendermaßen.

> Eine Anforderung ist:
>
> 1. Eine Eigenschaft oder Fähigkeit, die von einem Benutzer (Person oder System) zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt wird.
> 2. Eine Eigenschaft oder Fähigkeit, die ein System oder Teilsystem erfüllen oder besitzen muss, um einen Vertrag, eine Norm, eine Spezifikation oder andere formell vorgegebene Dokumente zu erfüllen.

[49] [RS14, Seite 9]

3. Eine dokumentierte Repräsentation einer Eigenschaft oder Fähigkeit gemäß (1) oder (2).[50]

Klassischerweise lässt sich zwischen funktionalen und nichtfunktionalen Anforderungen unterscheiden[51]. Funktionale Anforderungen beschreiben die gewünschten Systemfunktionen oder die Tätigkeiten der Software. Zu den nichtfunktionalen Anforderungen, werden oft nur die Qualitätsanforderungen gezählt, können aber noch Weitere umfassen, zum Beispiel Anforderungen an zusätzliche Lieferbestandteile (Dokumentation), technologische Anforderungen oder Anforderungen an die Benutzungsoberfläche.[52]

3.1.1 Methodik

Die Methodik der Anforderungsanalyse muss zu den Projektcharakteristika passen. Da im Rahmen dieser Arbeit eine agile Vorgehensweise gewählt wurde, muss auch das Requirements Engineering sich darin widerspiegeln. Das vorliegende Softwareprojekt ist, wie die meisten Projekte, keine Neuentwicklung, sonder basiert auf vorhanden Systemen. Bei der Weiterentwicklung müssen nicht alle Anforderungen neu erfasst werden, sondern nur solche, die noch nicht implementiert wurden. Hierzu eignet sich der Delta-Ansatz von Tayeh und Häußer und wird im folgenden Abschnitt vorgestellt.[53] Nach deren Definition ist eine Delta-Anforderung „eine funktionale oder nicht-funktionale Anforderung, welche die Lücke von Forderungen zwischen Alt- und Neusystem/erweitertem System schließt, ohne dass dabei die Funktionalitäten des Altsystems in seiner Gesamtheit verstanden und erfasst werden müssen.“[54] Die Autoren sehen die Vorteile dieses Systems in Fällen, bei denen das vorhandene System nicht ausreichend dokumentiert ist, unabhängig von dem verwendeten Vorgehensmodell. Weitere Fälle sind die Spezifikation von verschiedenen Produktvarianten einer gemeinsamen Basis und die systemübergreifende Weiterentwicklung in einer nicht ausreichend dokumentierten Systemlandschaft. Erfasst werden alle neuen Anforderungen und zusätzlich bereits umgesetzte Anforderungen, die für die Weiterentwicklung relevant sind, weil zum Beispiel Schnittstellen bestehen. Für den Einsatz im agilen Kontext wird dieses System explizit empfohlen, da hier eine komplette Nachdokumentation des Gesamtsystems

[50] zitiert nach: [RS14]
[51] vgl. [Hru14, Seite 12]
[52] vgl. [RS14, Seite 17]
[53] vgl. [RS14]
[54] siehe: [RS14]

den Prinzipien des agilen Manifestes widerspricht. Dieser Ansatz ist, nach Meinung der Autoren, sinnvoll, wenn im Projekt mehr als 10 bis 15 Prozent, aber weniger als 50 Prozent des vorhandenen Systems geändert werden müssen.[55] Diese Beschränkung hängt mit der Eigenschaft des Delta-Ansatzes zusammen, die bestehende Lösung nur zu erweitern, aber nicht kritisch zu hinterfragen. Die Erweiterung des Maßnahmentools auf Basis des ORK-Frameworks fällt in diesen Bereich, da auch das Framework zum vorhandenen System gezählt werden muss.

3.1.2 User-Stories

Nachfolgend sind die User-Stories dargestellt, wie Sie in Zusammenarbeit mit den Benutzern erfasst wurden. Aufgrund der geringen Nutzeranzahl wurde auf eine Einteilung in Personas verzichtet. Die User-Stories sind hier auf einer hohen Abstraktionsebene dargestellt. In der Praxis wurden diese noch weiter zergliedert (geschnitten).

1.0	Als Benutzer möchte ich eine integrierte Verbraucherverwaltung, um diese mit den Maßnahmen verknüpfen zu können.

- Die Verbraucherverwaltung erfasst die Verbraucher, Verteiler und Verbrauchergruppen.
- In der Verbraucherverwaltung sind die Beziehungen zwischen Verbrauchergruppen, Verteilern und Verbrauchern schnell erkennbar.
- Zu den Verbrauchern und Verteilern lassen sich Messergebnisse hinzufügen.

1.1	Als Benutzer möchte ich Messungen an Verbraucher und Verteilern verwalten können, um diese auswerten zu können.

- Es lassen sich alle CRUD-Operationen durchführen.
- Eine Messung enthält mindestens ein Datum und einen Messwert.

1.2	Als Benutzer möchte ich Verbraucher verwalten können.

- Es lassen sich alle CRUD-Operationen durchführen.
- Man kann aus Listen die zugehörige Verbrauchergruppe mit Typ und den Verteiler auswählen.
- Es wird zwischen eigenständigen Verbrauchern mit Typenbezeichnung und Verbraucherarten, die mehrere Einzelverbraucher zusammenfassen unterschieden.

[55] vgl. [RSP09]

1.3	Als Benutzer möchte ich Verteiler verwalten können.

• Es lassen sich alle CRUD-Operationen durchführen.

1.4	Als Benutzer möchte ich Verbrauchergruppen verwalten können.

• Zu jeder Verbrauchergruppe lässt sich eine Typenliste anlegen.
• Die Gruppierung der Maßnahmen bezieht sich auf die Verbrauchergruppe und ersetzt die bisher bestehen Kataloge.
• Es lassen sich alle CRUD-Operationen durchführen.

1.5	Als Benutzer möchte ich im Bereich Maßnahmenmanagement auf die Daten der Verbraucherverwaltung zurückgreifen können.

• Die vormals manuell eingetragenen Messwerte werden ersetzt durch eine Auswahl der Messungen in der Verbraucherverwaltung.
• Zusatzinformationen aus der Verbraucherverwaltung, wie Raum- und Gebäudename und Herstellerangaben, werden automatisch übernommen.
• Die Oberfläche für die Maßnahmenverwaltung wird überarbeitet und übersichtlich gestaltet.

2.0	Als Benutzer möchte ich ein einheitliches und helles Farbschema, um die Benutzerfreundlichkeit zu verbessern.

• Es sollen so wenig unterschiedliche Benutzersteuerelemente wie möglich verwendet werden.
• Die Akzentfarbe soll orange bleiben.
• Die Benutzeroberflächen sind ähnlich aufgebaut.
• Die Buttons für die CRUD-Operationen sehen immer gleich aus.

3.0	Als Benutzer möchte ich einen Aktionsplan für jede Maßnahme ausdrucken können.

• Die verwendeten Farben sind gut lesbar.
• Es gibt keine überflüssigen Benutzersteuerelemente, wie zum Beispiel Auswahlboxen oder Buttons
• Falls es sehr lange Einträge, oder Listen im Aktionsplan gibt, bleibt der Ausdruck trotzdem lesbar.

4.0	Als Benutzer möchte ich keine Fehler mehr haben, die in der alten Version bereits vorhanden sind.

• Die Grafiken in der Maßnahmenübersicht werden auch bei neu angelegten Maßnahmen angezeigt.
• Die Rechtschreibfehler werden korrigiert.
• Die Benennung der Textboxen wird angepasst.
• Neu angelegte Maßnahmen werden fehlerfrei immer gespeichert.
• Die Amortisationszeit einer Maßnahme wird auf Jahresbasis berechnet.
• Der eingesparte CO_2-Verbrauch berechnet sich aufgrund der tatsächlich eingesparten Werte.

5.0	Als Benutzer möchte ich ein regelmäßig Rückmeldungen des Programms bekommen, um zu wissen, wieso eine Aktion nicht durchführbar ist.

• Das Deaktivieren von Benutzersteuerelementen wird vermieden und stattdessen durch Hinweismeldungen oder Dialogfeldern ersetzt.
• Löschvorgänge schlagen fehl, falls es verknüpfte Elemente gibt und auf die Verknüpfung wird in einem Dialog hingewiesen.

6.0	Als Benutzer möchte ich gesuchte Maßnahmen oder Verbraucher generell schneller finden, um mit der Software auch bei größeren Datenmengen gut arbeiten zu können.

• Alle Listen sind alphabetisch sortiert.
• Es gibt zu jeder Liste ein Suchfeld.

7.0	Als Benutzer möchte ich die Raumverwaltung der MS-Access Datenbank „Raumbuch“ nutzen.

• Das Raumbuch wird in die zentralen Datenbank überführt.
• Es lassen sich alle CRUD-Operationen zu den, für das Maßnahmenmanagement relevanten Räumen durchführen.
• Das Raumvolumen wird automatisch berechnet.
• Verbraucher und Verteiler können einem Raum zugeordnet werden.

8.0	Als Benutzer möchte ich eine Auswertung der Energieverbräuche vornehmen können, um den rechtlichen Vorgaben zu entsprechen.

• Die tatsächlichen Messwerte der Verbraucher und Verteiler werden den prognostizierten Werten gegenüber gestellt.
• Die zeitliche Auswertung erfolgt über den gesamten Messzeitraum.
• Eine Filterung nach den verschiedenen Verteilern ist möglich.

9.0	Als Entwickler möchte ich einen gut strukturierten Quellcode, um die Weiterentwicklung schneller voranzutreiben.

• Es wird ein Refactoring durchgeführt
• Die angepassten Klassen im Domänenmodell werden von dem Basismodell abgeleitet
• Zu große Klassen und Methoden werden zerlegt

3.2 OpenResKit

OpenResKit ist ein Forschungsprojekt, dass an der HTW Berlin im Studiengang Betriebliche Umweltinformatik durchgeführt worden ist. Hierbei wurden Open-Source-basierende Softwarewerkzeuge zu innerbetrieblichen Ressourcen- und Effizienzfragestellungen erstellt.[56] Der Nutzerfokus liegt auf kleinen und mittleren Unternehmen, die mit einfachen und problemspezifischen Applikationen die Ressourceneffizienz in ihrem Betrieb erhöhen können. Ziel ist es, einen zentralen Datenpool mit einer standardisierten Schnittstelle zu schaffen. Dieser wurde in Form des OpenResKit-Hubs, hier nachfolgend auch ORK-Server genannt, realisiert. Die Plattform ist leicht erweiterbar und modular verwendbar. Der OpenResKit-Server kann über die OpenSource-Paketverwaltung Nuget eingebunden werden. Die Server-Module und die Client-Anwendungen sind auf der Projektwebsite[57] erhältlich. Es sind bereits einige Softwarebausteine entstanden, so auch eine Anwendung zum Maßnahmenmanagement. Diese Anwendung wurde im Rahmen einer vorangegangenen Abschlussarbeit erweitert und an die Bedürfnisse des Partnerunternehmens angepasst. Zunächst ist deshalb eine sorgfältige Analyse und Beschreibung der allgemeinen OpenResKit-Architektur, aber auch das bereits eingesetzte Maßnahmenmanagement, erforderlich. Abbildung 4

[56] vgl. [WKZS14]
[57] http://openreskit.htw-berlin.de/

zeigt die Architektur des OpenResKit-Frameworks. Es handelt sich um eine typische zweischichtige Client-Server Architektur.

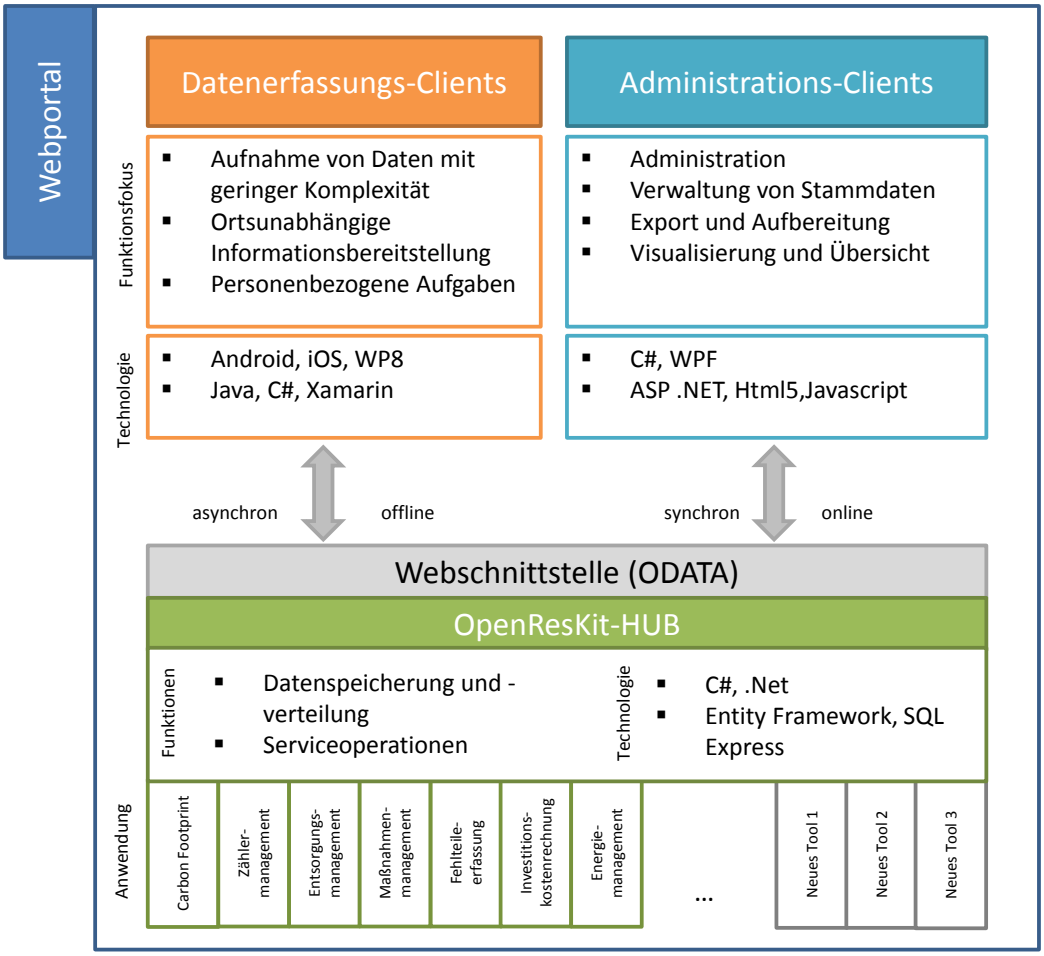

Abbildung 4: OpenResKit-Architektur[58]

3.2.1 OpenResKit Server

Der Server ist für zwei zentrale Aufgaben verantwortlich. Er stellt die Domänenmodelle bereit und persistiert sie in der eigenen Datenbank, üblicherweise eine Microsoft SQL Server Datenbank (Express Edition). Dabei wird das Microsoft Entity Framework (EF) in Version 5 als Objektrelationaler Mapper (ORM) verwendet. Ein ORM verbindet die Klassenstrukturen mit einer relationalen Datenbank. Diese Abstraktionsschicht wandelt Abfragen und Änderungen an das Modell in einen SQL-Befehl um, der

[58] entnommen aus: [WKZS14]

auf der Datenbank ausgeführt wird. Somit muss der Entwickler sich nicht mehr mit der Technik der zugrundeliegenden Datenbank beschäftigen. Ohne die Verwendung eines ORM müsste jedes SQL-Statement für Datenbankoperationen selbst geschrieben werden. Zum Datenaustausch mit den Clients wird eine Schnittstelle angeboten. Diese wurde mit einer Webschnittstelle nach dem OData-Protokoll implementiert.[59] OData ermöglicht die Erstellung und Verarbeitung von REST-basierten Datendiensten.[60] Ressourcen werden dabei über eine HTTP-Nachricht angefragt und bearbeitet. Antworten werden im JSON-Format[61] oder ATOM-Format[62] zurückgegeben. Es stehen für gängige Programmiersprachen verschiedene Clients zur Verfügung.[63] Darüber hinaus können, mittels eines WCF-Services, auf dem Server Dienste beziehungsweise Methoden ausgeführt werden. Diese Umsetzung ermöglicht die Nutzung verschiedener Technologien für die Entwicklung der Clients, da lediglich die Unterstützung des OData-Formats gewährleistet sein muss.

Entwicklung des Domänenmodells

Bei der Erstellung eines Domänenmodells können mit dem EF zwei Workflows verfolgt werden. Beide Vorgehensweisen können unabhängig davon, ob bereits eine Datenbank vorhanden ist oder nicht, angewandt werden, bieten aber unterschiedliche Vor- und Nachteile. Bei der Code-First-Vorgehensweise wird das Datenmodell in einer C#-Klasse[64] beschrieben. Diese Klassen werden auch POCO (Plain Old CLR Object) genannt, da sie keine Abhängigkeiten zu anderen Frameworks haben. Mit einer Konfigurationsdatei können Abhängigkeiten und Schlüsselbeziehungen modelliert werden. Hierbei kommt die *Fluent-API* zum Einsatz. Beispiele für Konfigurationen sind Eigenschaftszuordnungen, Typzuordnungen und Beziehungen.[65] Manuelle Änderungen am Datenbankschema sind nicht möglich, da diese durch den Programmcode definiert wird. Aufgrund der großen Flexibilität und der Verwendung von Quellcode wird dieser Ansatz von vielen Entwicklern bevorzugt und ist auch bei OpenResKit im Einsatz.

Beim Model-First-Ansatz wird zunächst mit einem Designer ein grafisches Daten-

[59] vgl. [WKZS14]
[60] REST steht für Representational State Transfer und bezeichnet ein Programmierparadigma für verteile Systeme
[61] JSON steht für JavaScript Object Notation und ist ein einfach lesbares Textformat
[62] ATOM ist ein XML-Format
[63] vgl. [Wik15]
[64] oder in einem anderen CLR-konformen Objekt des .NET-Frameworks wie VB.Net
[65] vgl. [Mic15a]

modell erstellt, welches als XML-basierte Datei[66] gespeichert wird. Dies kann für die Kommunikation mit den Auftraggebern hilfreich sein, um das Datenmodell zu visualisieren. Dieser Ansatz wird oftmals von Software-Architekten genutzt, hat aber als Entwicklungsansatz einige Nachteile. Die Verwendung eines Versionsverwaltungssystems und das Lösen von Konflikten ist bei Code-First-Ansatz leichter. Weiterhin ist es einfacher Programmcode zu debuggen, als eine XML-Datei. Fehlermeldungen sind im codebasierten Ansatz aussagekräftiger. Zudem unterstützt EF keine automatische Migration bei dem Model-First-Ansatz. Eine bereits existierende Datenbank kann bei dieser Methode ebenfalls genutzt werden. Dabei wird das Modell aufgrund der bestehenden Datenbank erzeugt. Manchmal wird dieser Ansatz auch Database-First genannt.

Die Änderungen des letzten Releases von EF auf Version 6 zeigen, dass Microsoft den Code-First Ansatz bevorzugt, da hier zusätzliche Möglichkeiten bereitstehen.[67] Hinzugefügte Funktionen sind beispielsweise ein Reverse Engineering Prozess für Code-First-Entwicklung aus bestehenden Datenbanken. Der Code-First Quellcode wird automatisch erzeugt. Gerade für Integrationsfragestellungen, bei denen vorhandene Datenbanken eine große Rolle spielen, ist dies ein enormer Vorteil. In Entity Framework 7 wird es nur noch den codebasierten Ansatz geben.[68] Für die nachträgliche Visualisierung der Modelle stehen jedoch Werkzeuge bereit.

Umsetzung von Vererbungsbeziehungen

Vererbung ist ein grundlegendes Konzept der objektorientierten Programmierung. Es gibt verschiedene Möglichkeiten dieses Konzept in relationalen Datenbanken umzusetzen. Der Vorteil eines ORM ist zwar, dass der Entwickler sich über die Erstellung der Datenbank keine Gedanken machen muss. Falls der direkte Datenbankzugriff für mögliche Export-Fragestellungen allerdings relevant ist, lohnt sich die eingehendere Beschäftigung mit dem Thema. Objektvererbung kann auf drei Arten in der Datenbank widergespiegelt werden, welche auch mit EF umsetzbar sind. Standardmäßig wird die Form Table-per-Hierarchy (TPH) gewählt. Das bedeutet, dass alle Objekte einer Hierarchiestufe in einer Tabelle gespeichert werden. Es gibt also nur eine Tabelle für alle Typen einer Basisklasse. Sie werden durch eine zusätzliche *Discriminator*-Spalte voneinander unterschieden. Die Datenbank wird also bewusst denormalisiert. In der zweiten

[66] Das Dateiformat heißt bei EF *EDMX*
[67] vgl. [Mic15b]
[68] vgl. [Mil14]

Variante Table-per-Type (TPT) gibt es für jeden Typ, also Basisklasse und abgeleitete Klassen, eine eigene Tabelle. Die Vererbungsbeziehung wird über eine Fremdschlüsselbeziehung hergestellt. Zuletzt gibt es noch die Variante Table-per-Concrete-Type (TPC). Hier gibt es für jede abgeleitete Klasse eine einzelne Tabelle. Die Attribute der Basisklasse kommen als redundante Spalten in den Tabellen vor. Im Anhang zeigt Abbildung 21 beispielhaft das Klassendiagramm und die Tabellen einer möglichen Umsetzung. In einem Blogbeitrag von Alex James werden die Vor- und Nachteile diskutiert.[69] Generell ist die Variante Table-per-Hierarchy am performantesten, allerdings ist die normalisierte Form bei Table-per-Type übersichtlicher, da diese Struktur mit der Domäne übereinstimmt.

Dynamische Anpassbarkeit

Besonders erwähnenswert ist das dynamisch anpassbare Domänenmodell, dass für die große Flexibilität und Modularität verantwortlich ist. So können zur Laufzeit neue Module hinzugefügt werden und die Datenbank reagiert mit einer entsprechenden Schemaänderung und einer automatischen Datenbankmigration. Bei der Umsetzung wurde dabei auf das Managed Extensibility Framework (MEF)[70] von Microsoft zurückgegriffen. Diese Bibliothek ist Teil des Microsoft .NET-Frameworks und ermöglicht die Entwicklung erweiterbarer Anwendungen, ohne eine Konfiguration vornehmen zu müssen, oder fest codierte Abhängigkeiten zu berücksichtigen. Somit können unbekannte, kompilierte Erweiterungen eingebunden werden.

MEF basiert auf zwei interessanten Entwurfsmustern, die für das bessere Verständnis der Funktionsweise des ORK-Servers näher erläutert werden. Fowler erklärt in einem Blogbeitrag die Unterschiede der Muster Dependency Injection (DI) und Inversion of Control (IoC).[71] Beide Muster werden genutzt, um entwickelte Komponenten wiederverwendbar und austauschbar zu machen. Ein einfaches Beispiel für IoC sind Benutzeroberflächen. Die Kontrolle liegt nicht mehr, wie beispielsweise in Konsolenanwendungen, bei der Anwendungslogik, sondern bei den Elementen der Benutzeroberfläche selbst. Die Hauptanwendung reagiert mittels Eventhandler auf die Eingaben. Eine spezifische Form der IoC ist die DI. Um eine feste Koppelung zweier Objekte zu verhindern, kennt eine aufrufende Komponente nur die öffentliche Schnittstelle (Interface)

[69] vgl. [Jam09]
[70] siehe: https://mef.codeplex.com/
[71] vgl. Fowler.23.01.2004

eines benötigten Objektes. Ein zentraler Container, in diesem Fall MEF, übernimmt die Lokalisierung und Instanziierung des benötigten Objektes mit seiner konkreten Implementierung. An folgendem Code-Beispiel wird die Funktionsweise veranschaulicht.

Listing 1: Einsatz von MEF bei der Erstellung des Kontextes[72]

```
1   internal class DomainModelContextFactory
2   {
3     private readonly IEnumerable<IDomainEntityConfiguration>
          m_DomainEntityConfigurations;
4     private readonly IDatabaseInitializer<DomainModelContext>
          m_Initializer;
5
6     [ImportingConstructor]
7     public DomainModelContextFactory([Import]
          IDatabaseInitializer<DomainModelContext> initializer,
          [ImportMany] IEnumerable<IDomainEntityConfiguration>
          domainEntityConfigurations)
8     {
9       m_Initializer = initializer;
10      m_DomainEntityConfigurations = domainEntityConfigurations;
11    }
12
13    [Export]
14    public DomainModelContext CreateContext()
15    {
16      return new DomainModelContext(m_Initializer,
            m_DomainEntityConfigurations);
17    }
18  }
```

Die Klassifizierung als injizierbare Komponente erfolgt bei MEF über Annotationen. In Zeile 13 wird beispielsweise kenntlich gemacht, dass die nachfolgende Methode als MEF-Komponente exportiert wird. Diese Methode wird im zentralen Container registriert und steht auch anderen Komponenten zur Verfügung. Die Methode CreateContext() erstellt das Domänenmodell. Er benötigt einen Initialisierer, der die Erstellungsstrategie der Datenbank festlegt und erhält eine Liste mit Konfigurationsdateien. Diese werden wiederum im Konstruktor der Factory-Klasse über MEF importiert.

[72] siehe: [HTW13]

3.2.2 Erweitertes Maßnahmenmanagement

Im Partnerunternehmen wird eine erweiterte Form der Basis-Applikation Maßnahmenmanagement[73] eingesetzt. Mit dieser Anwendung lassen sich innerbetriebliche Maßnahmen anlegen und nachverfolgen. Die Basisversion enthält eine Mitarbeiterverwaltung und einen Arbeitsbereich für Einstellungen. Der Kern der Anwendung ist eine tabellarische Übersicht aller Maßnahmen und deren termingerechte Auswertung. Einer Maßnahme lassen sich Dokumente und Bilder hinzufügen und der verantwortliche Mitarbeiter zuordnen. Der geplante Erledigungstermin wird ebenfalls eingetragen. Nach Durchführung der Maßnahme, wird das tatsächliche Abschlussdatum eingetragen und es kann eine Bewertung vorgenommen werden. Der Vergleich zwischen den geplanten und tatsächlichen Abschlussdaten ermöglicht eine Auswertung über die Termintreue der Durchführung und wird in einem Balkendiagramm dargestellt. Die Maßnahmen lassen sich in verschiedene Kataloge gruppieren. Die mobile Applikation wird nicht verwendet.[74]

In der, durch eine Bachelorarbeit erweiterten Version, wurden einige Ergänzungen vorgenommen. Eine Spezifikation der durchgeführten Erweiterungen oder eine Dokumentation liegt nicht vor. Die Analyse wird also aufgrund des laufenden Systems durchgeführt. Diese Version umfasst mehr Eingabefelder für die Maßnahmen. Jeder Maßnahme ist ein Raum, ein Verbraucher und ein Messgerät zuzuordnen. Weiterhin ist eine statische Amortisationsrechnung durchführbar, weshalb entsprechende finanzwirtschaftliche Parameter erfasst werden. Die Angabe von Verbrauchsdaten ermöglicht eine ökologische Auswertung. Außerdem können Untermaßnahmen angelegt werden. Die Unterschiede im Domänenmodell sind in Abbildung 5 dargestellt. Eine zweite Auswertemöglichkeit wurde hinzugefügt. Die tabellarische Übersicht der Maßnahmen lässt sich auf eine Ansicht, mit Fokus auf den Energieverbrauch und die wirtschaftliche Bewertung der Maßnahmen umschalten. Zusätzlich wird ein entsprechendes Balkendiagramm mit gleichem Schwerpunkt angezeigt. Zudem wurden noch einige kleinere Anpassungen, wie Namensänderungen oder die zusätzliche Spalten in der tabellarischen Übersicht vorgenommen. Die erweitere Version wurde in enger Zusammenarbeit mit dem Partnerunternehmen entwickelt und entspricht weitgehend den bisherigen Anforderungen, jedoch sind nicht alle Erfordernisse für eine Energiemanagement-

[73] http://openreskit.htw-berlin.de/node/37
[74] vgl. [HTW15]
[75] eigene Darstellung

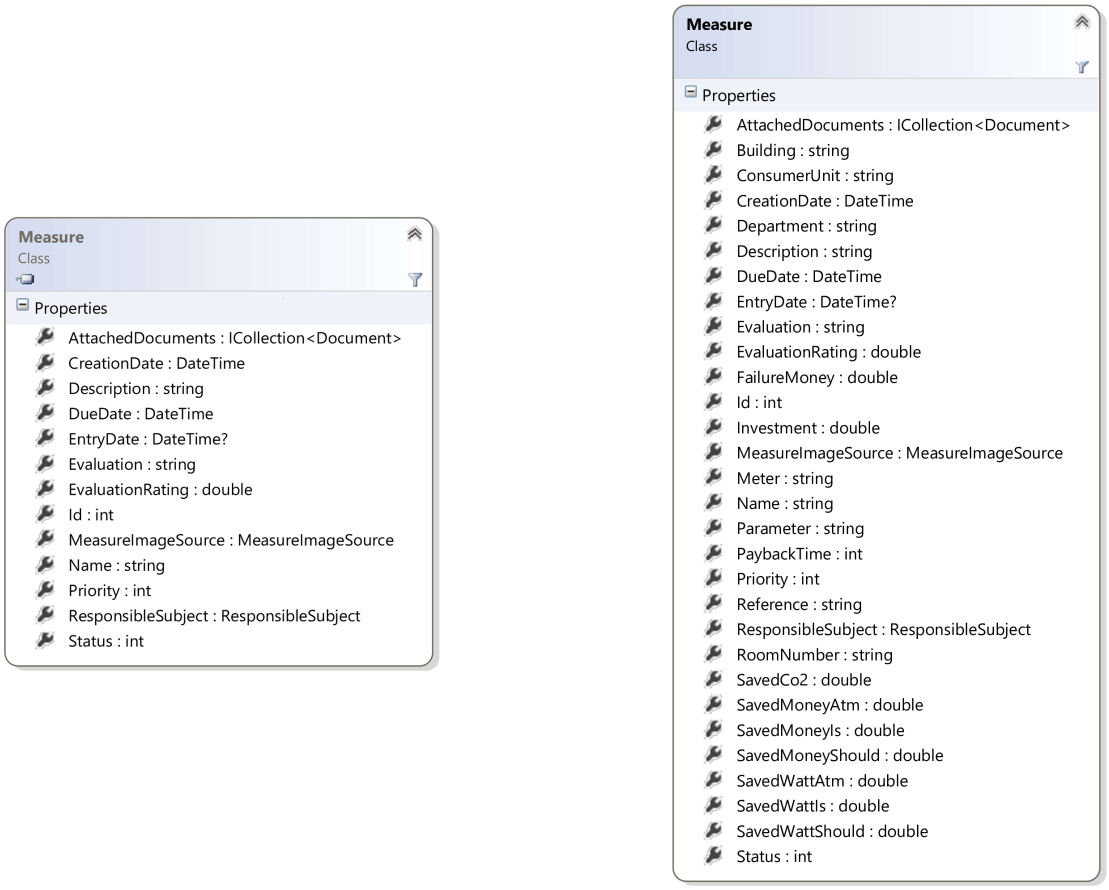

Abbildung 5: Klassendiagramm für Maßnahmen der Basisversion und der erweiterten Version[75]

software erfüllt. Es fehlen Auswertemöglichkeiten und die Integration in bestehende Systeme, wie sie typischerweise in solchen Anwendungen vorkommen.[76]

Eine technische Beurteilung wurde bisher nicht durchgeführt. In einem weiteren Schritt wird deshalb der Quellcode analysiert. Hierbei fallen viele sogenannte Code-Smells[77] auf. Mit diesem Begriff wird Code bezeichnet, der, auch wenn er keine Programmfehler beinhaltet, überarbeitet werden soll. Damit wird der Code leichter verständlich und besser strukturiert, womit Fehler, die bei Korrekturen und Erweiterungen entstehen, vermieden werden. Bei der ersten Übersicht des Clients fällt die sehr große Klasse MeasureManagementViewModel.cs mit knapp 1000 Zeilen Länge auf. Dieses Anti-Pattern wird auch als „Gottklasse“ bezeichnet und beschreibt Klassen, die zu viele

[76] vgl. [BAF15]

[77] Der Begriff kann im deutschen als *übelriechender Code* übersetzt werden und geht auf Kent Beck und Martin Fowler zurück. vgl. [FB99]

Methoden und Attribute hat. Die Zerlegung in kleinere Objekte entspricht den Anforderungen der objektorientierten Programmierung und erleichtert die Wartung. Es gibt mehrere ähnliche Beispiele.

Listing 2: Oberflächendesign

```
1          <StackPanel
2            Grid.Row="0"
3              Margin="10,-289,-0.2,208.8"
                   RenderTransformOrigin="0.5,0.5">
4                <StackPanel.RenderTransform>
5                  <TransformGroup>
6                    <ScaleTransform/>
7                    <SkewTransform AngleX="-0.179"/>
8                    <RotateTransform/>
9                    <TranslateTransform X="0.397"/>
10                 </TransformGroup>
11               </StackPanel.RenderTransform>
12               <TextBlock
13                 Text="{lex:Loc Ork.Measure:Translate:Picture}"
14                                  Margin="0,0,-0.8,0" />
15                 </StackPanel>
```

Die Programmierung der Benutzeroberflächen ist nicht mehr nachvollziehbar und wurden vermutlich mit einem Interface-Designer realisiert. Ohne eine Dokumentation ist eine Wartung nicht möglich. Auffällig sind negative Abstände in Zeile 3 und 14 oder die Drehung des Stackpanels um einen minimalen negativen Wert in Zeile 7 und sind vermutlich nicht beabsichtigt. Ähnliche Beispiele sind über die gesamte Implementierung der Oberflächen zu finden. Der Einsatz passender Benutzersteuerelemente zur Layoutunterstützung, sogenannte *Panel-Elemente* wurde vernachlässigt. An folgender Methode lassen sich anschaulich weitere Schwächen des Codes verdeutlichen:

Listing 3: Methode zum Wechsel zwischen zwei Ansichten

```
1  public void OpenEconomy()        // Button-Methode für Ansicht
       umschalten
2  {
3          if (m_DgvVisibleEco == true)
4          {
5                  VisibleEco = false;
6                  VisibleEcoPlot = false;
7                  VisibleEcoPlot2 = false;
8
9                  VisibleNormal = true;
10                 VisiblePlot = true;
11                 VisibleCat = true;
```

```
12          }
13          else
14          {
15                  VisibleEco = true;
16                  VisibleEcoPlot = true;
17                  VisibleEcoPlot2 = true;
18
19                  VisibleNormal = false;
20                  VisiblePlot = false;
21                  m_PlotIsVisible = false;
22                  VisibleCat = false;
23          }
24          NotifyOfPropertyChange(() => SelectedCatalog);
25          NotifyOfPropertyChange(() => CanAdd);
26          NotifyOfPropertyChange(() => Measures);
27  }
```

Die Methode dient zum Wechsel zwischen zwei verschiedenen Ansichten, die bei einem Klick auf einen Umschaltbutton aufgerufen wird. Zunächst lässt sich eine besserer, also beschreibender Methodenname finden, z.B. SwitchBetweenEconomicAndStandardView(), damit dieser Name auch ausdrückt, was die Aufgabe der Methode ist. Auch Martin empfiehlt: „Ein langer beschreibender Name ist besser als ein kurzer geheimnisvoller Name und besser als ein langer beschreibender Kommentar.“[78] Die Namen der verwendeten Flag-Variablen lassen ebenfalls kaum Rückschlüsse auf die Funktion zu, außer dass vermutlich die Sichtbarkeit damit gesteuert wird. In den Zeilen 24 bis 26 wird suggeriert, dass sich drei Eigenschaften verändert haben und diese deshalb neu geladen werden müssen. Tatsächlich bleiben aber alle drei Eigenschaften unabhängig vom Methodenaufruf gleich. Der unnötige Wiederaufruf geht zu Lasten der Performance und die Übersichtlichkeit leidet.

Der Datenbankentwurf wurde unnötig verkompliziert. Einige Felder tauchen redundant in der Datenbank auf, obwohl die entsprechenden Werte aus anderen Daten berechnet werden, so zum Beispiel die Amortisationszeit oder der eingesparte Wert in CO2-Äquivalenten. Die Benennung und Verwendung einzelner Felder ist teilweise widersprüchlich und unterscheidet sich zwischen den einzelnen Abstraktionsebenen, also zwischen Datenbank, View-Model und Benutzeroberfläche. Die Kommentierung stiftet zusätzliche Verwirrung, da sie in mindestens einem Fall zu den Widersprüchen beiträgt. Auch hier gilt es, aussagekräftige und zweckbeschreibende Namen zu verwen-

[78] siehe [Mar09, Seite 70]

den[79]. Somit werden erklärende Kommentare überflüssig. Ein gründliches Reengineering ist also schon im Vorfeld nötig, um die Weiterentwicklung zu ermöglichen.

3.3 Softwareumgebung im Unternehmen

Mit verschiedenen Anwendungen werden die Anforderungen des Energiemanagements im Partnerunternehmen abgedeckt. Teilweise werden Daten in Textdokumenten und Tabellen erfasst. Für die Integrationsfragestellung sind drei Anwendungen beziehungsweise Datenbanken relevant.

MySQL-Datenbank: In einer zentralen Datenbank werden Daten gespeichert, die nicht nur das Energiemanagement betreffen. Diese Datenbank ist über viele Jahre gewachsen und basiert auf einem MySQL-Verwaltungssystem. Es gibt unterschiedliche Clients, die diese Datenbank ansteuern. Die Datenbank basiert auf Version 4.1.18, die im Januar 2006 veröffentlicht wurde und deren Support am Ende des Jahres 2010 eingestellt wurde[80]. Für den Bereich Energiemanagement sind vor allem die gespeicherten Verbraucher, wie Maschinen und Anlagen, interessant. Bei eingehender Analyse kann festgestellt werden, dass die Verbraucher unvollständig erfasst sind und ein Teil des Datenbestandes veraltet ist.

Raumbuch: In der bisherigen Version des Maßnahmenmanagements wird zu jedem Verbraucher ein Standort angegeben. Im Unternehmen existiert eine Raumverwaltung in Form einer MS-Access Datei. Dieses Raumbuch wird für verschiedene Zwecke, insbesondere für die Betriebstechnik benutzt und enthält neben Rauminformationen auch Angaben zu Fußböden oder Hausalarmmeldern. Für das Energiemanagement sind lediglich die Raum- und Gebäudenamen, sowie die Daten über die Fläche und Raumhöhe relevant. Dadurch lässt sich das Raumvolumen errechnen, was beispielsweise für die Bewertung der aufgewendeten Heizenergie genutzt werden kann.

e!Sankey: Die Energieflüsse des Unternehmens werden mit einem Sankey-Diagramm dargestellt. Ein Sankey-Diagramm zeigt die Größe der betrachteten Flussmengen mit proportional dicken Pfeilen an. Diese Darstellungsweise ist seit vielen Jahren für die Visualisierung von Energieflüssen und Materialflüssen

[79] vgl. [Mar09, Seiten 45ff.]
[80] vgl [MyS]

etabliert.[81] Zum Einsatz kommt dabei die Software e!Sankey der ifu Gmbh.[82] Die Software bietet mit „Live-Link to Excel“ eine Schnittstelle an, um Flusswerte automatisch zu importieren.

[81] vgl. [Sch08, Seite 83]
[82] http://www.e-sankey.com

4 Konzeption

Die Analyse der bestehenden Anwendungslandschaft und die theoretische Recherche in Abschnitt 2.2 führen zu dem im Folgenden beschriebenen Konzept. Damit können die Anforderungen des Partnerunternehmens erfüllt werden. Die Konzeption verfolgt mit dem Bottom-Up-Prinzip einem pragmatischen Ansatz. Das heißt, dass bestehende Bestandteile wiederverwendet und zu einem vollständigen Gesamtsystem zusammengesetzt werden. Basis der Softwarelösung ist der bestehende OpenResKit-basierte Client zum Maßnahmenmanagement und das dazugehörige Servermodul. Die erste der beiden Säulen des Konzeptes ist die Weiterentwicklung der bestehenden Anwendung. Die zweite Säule ist die Integration bestehender Applikationen. Die Anforderungen des Partnerunternehmens sehen vor, die gesamten Aufgaben im Rahmen des Energiemanagements nach DIN ISO 50001 innerhalb einer Anwendung zu bearbeiten. Dazu sollen die Informationen der Energieverbraucher in das Maßnahmenmanagement migriert werden. Zusätzlich sollen zu den Energieverbrauchern Messwerte und Ablesungen gespeichert werden können. Diese Informationen sind auch für eine zeitliche Energieauswertung innerhalb der Anwendung zu nutzen, die die geschätzten Energieeinsparungen mit den tatsächlichen Betriebswerten vergleicht. Die lokale MySQL-Datenbank wird für diesen Anwendungsbereich nicht mehr weitergeführt. Das Raumbuch ist in die Anwendung zu integrieren. Optional gibt es die Möglichkeit, weiterhin mit einem Access-Frontend auf die Datensätze zuzugreifen und auch dort Veränderungen vorzunehmen. Die Daten sollen auch für weitere Anwendungen, zum Beispiel für die Sankey-Auswertungen zur Verfügung stehen. Für eine einheitliche Benutzeroberfläche werden die Stile der bereits verwendeten Steuerelemente durch die Verwendung eines Frameworks vereinheitlicht. Die erstellte Software bildet damit ein vollständiges Energiemanagementsystem, das alle Anforderungen der DIN ISO 50001, wie in Abschnitt 2.1 beschrieben, abdeckt.

4.1 Funktionale Erweiterungen OpenResKit

Die Defizite im Quellcode erfordern zunächst ein Reengineering der Client-Anwendung, wie in Abschnitt 2.2.1 beschrieben. Damit wird die Wartbarkeit und Qualität erhöht, um die Weiterentwicklung überhaupt erst zu ermöglichen. Da keine Tests im Programmcode vorliegen, muss hier besonders sorgfältig gearbeitet werden, um die Funktionalität nicht versehentlich zu verändern. Gegebenenfalls müssen zunächst entsprechende Tests geschrieben werden. Die Anforderungen werden in zwei neuen Modulen umgesetzt, die sich im Client und im Server widerspiegeln. Das Modul Verbraucher beinhaltet die Verbraucherverwaltung und das Modul Raumbuch die Informationen des Raumbuches. Das Modul Maßnahmen wird angepasst, um die neuen Informationen nutzen zu können. Mithilfe einer Druckfunktion können die wichtigsten Daten druckfertig aufbereitet werden. Die nicht-domänenbezogenen ORK-Server-Module werden unverändert übernommen. Im Client bleiben die Komponenten zur Benutzerverwaltung und Konfiguration ebenfalls gleich. In Abbildung 6 ist das Konzept grafisch dargestellt.

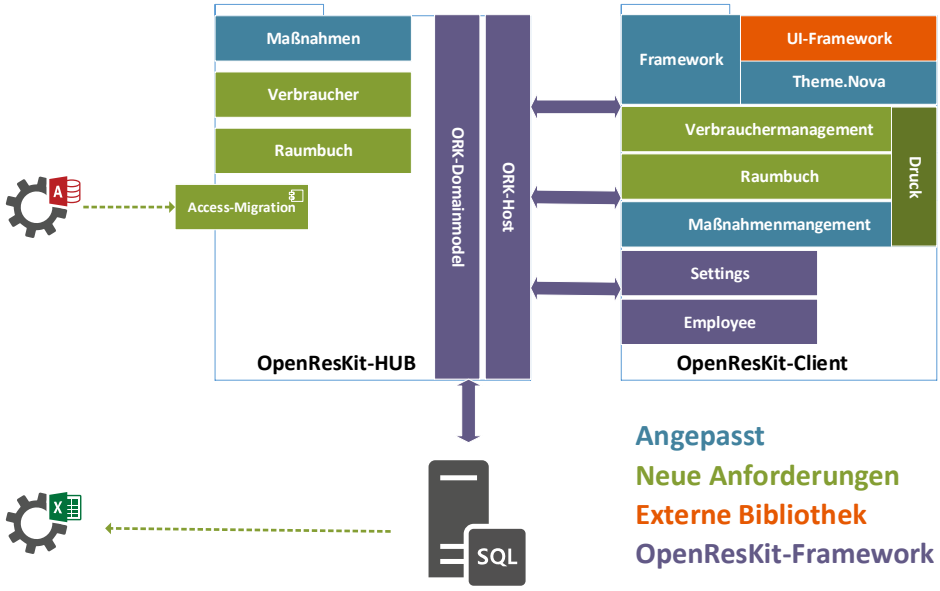

Abbildung 6: Konzeption der Weiterentwicklung[83]

[83] eigene Abbildung

4.2 Datenmodell und Integration

Die Integration in den OpenResKit-Server erfolgt aus verschiedenen Gründen durch das Vereinigen der Quelldaten. Zunächst erleichtert die Architektur die Erstellung neuer Datendomänen und weniger das Verbinden verschiedener externer Datenquellen zu einer Sicht. Die ursprünglichen Anwendungen sollen abgelöst werden, da sie entweder veraltet sind (MySQL) oder es sich um Eigenentwicklungen handelt (Access-Datei). Generell wird eine umfassendere Verwendung des OpenResKit-Hubs angestrebt. Die Datenintegration wird deshalb einmalig durchgeführt. Der korrespondierende Client ist zwar die wichtigste Benutzeranwendung, aber die Schnittstelle des OpenResKit-Servers ermöglicht anderen Clients eine leichte Anbindung. Während der Implementierungsphase wurde festgestellt, dass die Datenqualität der Verbraucherdaten in der MySQL Datenbank den Anforderungen nicht genügt. Daher war zusätzlich ein manueller Datenabgleich mit schriftlich vorhandenen Dokumenten erforderlich.

Die Architektur von OpenResKit begünstigt die Code-First Vorgehensweise bei der Entwicklung eines Domänenmodells.[84] Die Modellierung des Datenmodells erfolgt dabei direkt im Quellcode, was eine separate Konzeption des Datenmodells überflüssig macht und zwar ad-hoc anhand der Anforderungen.

4.3 Oberflächen

Die Gestaltung der Benutzeroberfläche orientiert sich am Microsoft Modern UI Design[85], auch unter dem internen Codenamen Metro oder einfach „Modernes Design“ bekannt. Der Gestaltungsstil lehnt sich am sogenannten Flat Design an, welches heutzutage bei allen gängigen Plattformen zum Einsatz kommt. Bei Android ist der Stil unter dem Namen Material Design bekannt und auch Apple lehnt sich daran an. Grundlegende Merkmale sind der Verzicht auf plastische Gestaltungselemente wie 3D-Effekte, Schlagschatten oder Texturen. Alle überflüssigen Designelemente werden entfernt. Schwerpunkte bilden eine klare Typographie, der bewusste Einsatz von Farben und hohen Farbkontrasten, sowie verschiedene Schriftgrößen und -stärken, um Hierarchiestufen kenntlich zu machen.

[84] vgl. Unterabschnitt 3.2.1
[85] vgl. [Mic14]
[86] Eigene Abbildungen

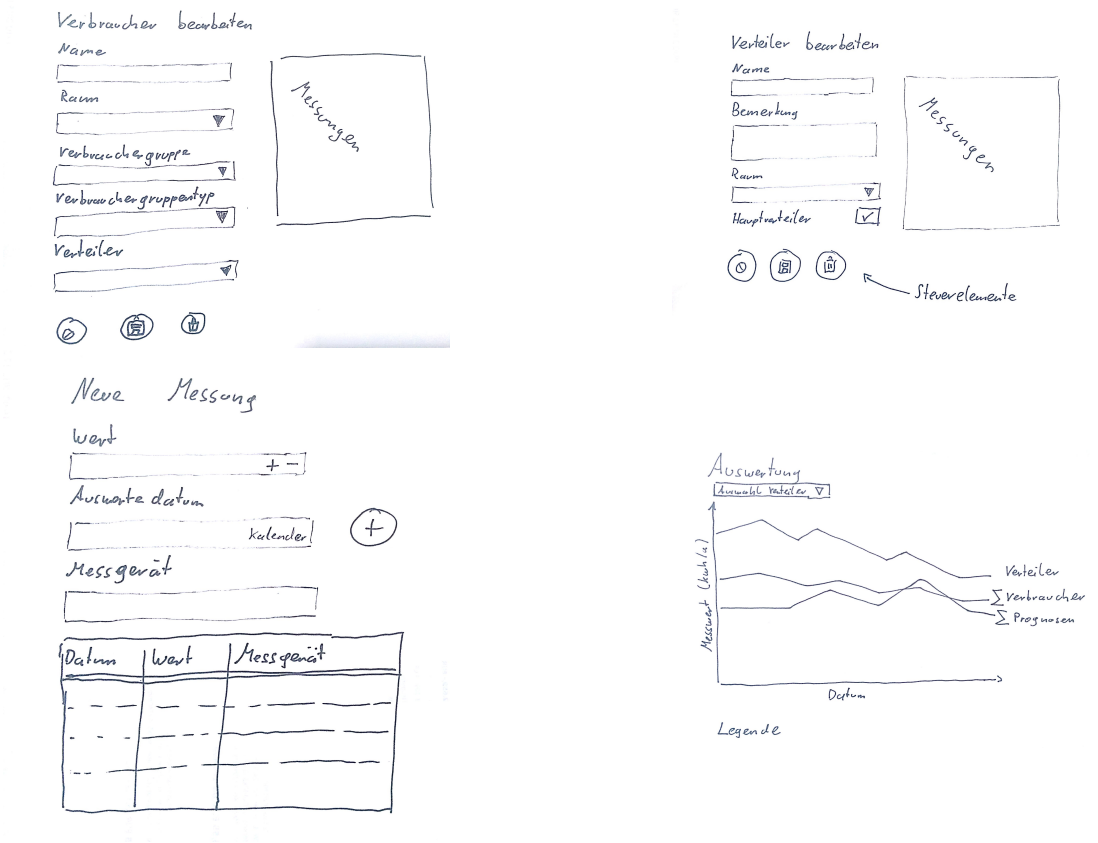

Abbildung 7: Whiteboard-Entwürfe der Detailmasken[86]

In der bisherigen Version des Clients wurde versucht, die Designrichtlinien in einem selbst entwickleten Grafikstil umzusetzen. Dies führte aber zu keinem zufriedenstellenden Ergebnis, da der Stil nicht auf alle Steuerelemente angewendet werden kann. Zudem wurde die Weiterentwicklung des Stils erschwert. Eine Vereinheitlichung und die konsequentere Ausrichtung an den Designrichtlinien ist deshalb notwendig.

Um eine Benutzeroberfläche zu konzeptionieren, gibt es verschiedene Möglichkeiten. Die Erstellung von Mockups ist eine schnelle und einfache Herangehensweise. Mockups sind Attrappen der Benutzeroberfläche ohne jede Funktion. Sie können sehr schnell auf Papier oder Whiteboards entworfen werden, oder man nutzt Softwarewerkzeuge, die das Aussehen der verwendeten Benutzersteuerelemente besser wiedergeben können. Die Konzeption auf Papier oder Whiteboards ist jedoch schneller zu realisieren und flexibler, da die Hemmungen, ein Konzept zu verwerfen, geringer ist. Aufwändiger ist es einen Prototyp der Benutzeroberfläche zu erstellen. Hierbei kann allerdings der grundlegende Programmablauf durch den Benutzer nachvollzogen werden. Ein Prototyp wurde nicht erstellt, da durch die agile Vorgehensweise der Anwender die Software

bereits im Einsatz beurteilen kann. In den Abbildungen 7 und 8 sind die Beispiele der erstellten Mockups zu sehen.

Abbildung 7 zeigt verschiedene auf einem Whiteboard erstellte Entwürfe. Schon bei der Konzeptionierung wird auf die Wiederverwendbarkeit der Steuerelemente geachtet. Beispielsweise können Messungen bei Verteilern und Verbrauchern erfasst werden. Für diese Aufgabe soll ein wiederverwendbares Steuerelement separat entworfen werden. Im Anhang sind die Entwürfe in einem größerem Format abgebildet. In der Konzeptionsphase kommen nicht nur händisch erstellte Mockups zum Einsatz, sondern es werden auch Softwarewerkzeuge, wie zum Beispiel bei Abbildung 8 genutzt. Die Gestaltung der Verbraucherübersicht orientiert sich nicht an Tabellen, sondern an Listen. Es gibt für die drei wichtigsten Objekte Verteiler, Verbrauchergruppe und Verbraucher je eine Liste mit Suchfunktionen. Die Objekte enthalten den Namen und wichtige Details, wie zum Beispiel das Datum der letzten durchgeführten Messung oder die Anzahl der zugehörigen Verbraucher.

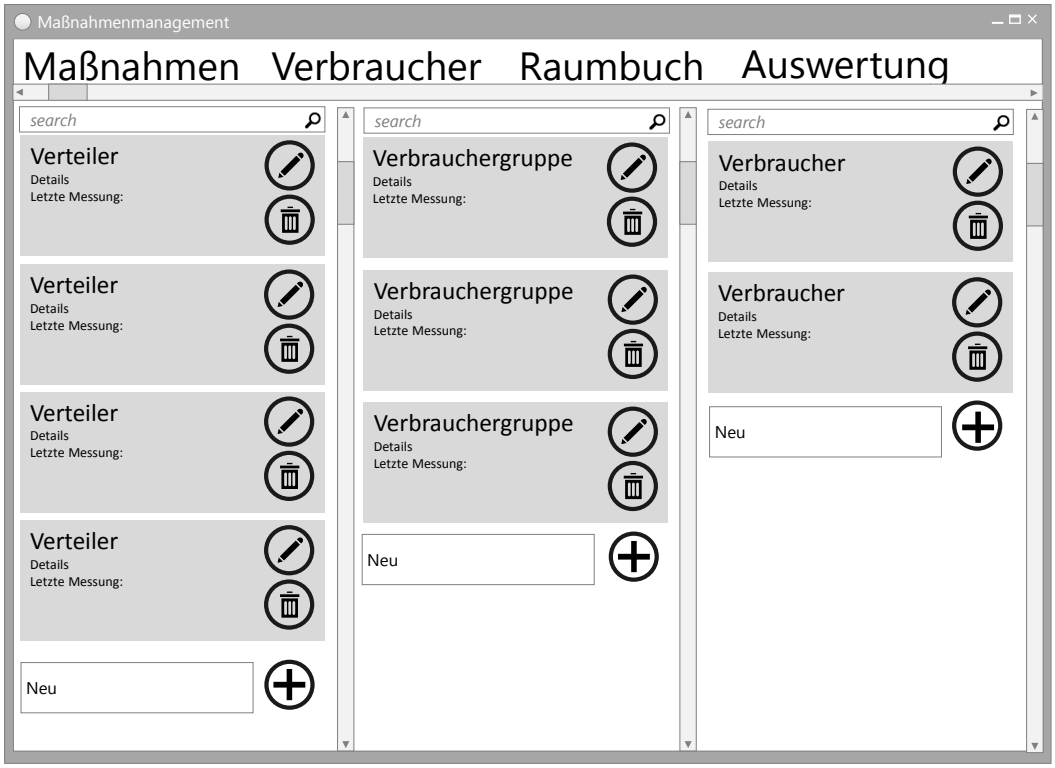

Abbildung 8: Oberflächenentwurf Verbraucheransicht[87]

[87] eigene Abbildung

5 Implementierung und Test

Das auf den Anforderungen des Partnerunternehmens basierende Konzept wird vollständig implementiert. Durch die agile Entwicklungsweise sind regelmäßig neue Softwareversionen auszuliefern, die sofort in den laufenden Betrieb gehen. Im Gegensatz zu einer prototypischen Entwicklung müssen also einige zusätzliche Faktoren berücksichtigt werden. Im folgenden Kapitel sind die wesentlichen Aspekte, die die Implementierung betreffen dargestellt. Hierbei kann man zwischen der server- und clientseitigen Entwicklung unterscheiden. Weiterhin wird noch auf die Datenkommunikation zwischen Server und möglichen Clients eingegangen und die verwendeten Testverfahren beschrieben.

5.1 OpenResKit-Server

Im OpenResKit-Server wurde die neue Datendomäne implementiert und die bestehenden Klassen angepasst. Ein vollständiges Klassenmodell ist im Anhang in Abbildung 20 abgebildet. In der letzten Erweiterung wurde das Klassenmodell der Basisversion verändert. In der objektorientieren Programmierung ist die Klassenvererbung üblich, bei der die abgeleitete Klasse alle Definitionen der Basisklasse beinhaltet. Die Basisklasse bleibt unverändert und kann wiederverwendet werden. Die Nutzung dieses Konzeptes bietet sich bei der Weiterentwicklung an. Die OpenResKit-Domänen verfügen zwar über keine abstrakten Klassen, allerdings sind üblicherweise alle Eigenschaften mit dem Schlüsselwort *virtual* gekennzeichnet, um das Überschreiben zu ermöglichen. In Abbildung 9 ist das neue Klassendiagramm zu sehen. Die Klasse EnergyMeasure leitet von der Basisklasse Measure ab und implementiert die veränderten Eigenschaften. Durch diese Vorgehensweise wird die im OpenResKit-Projekt angestrebte Modularisierung umgesetzt.

[88] eigene Abbildung

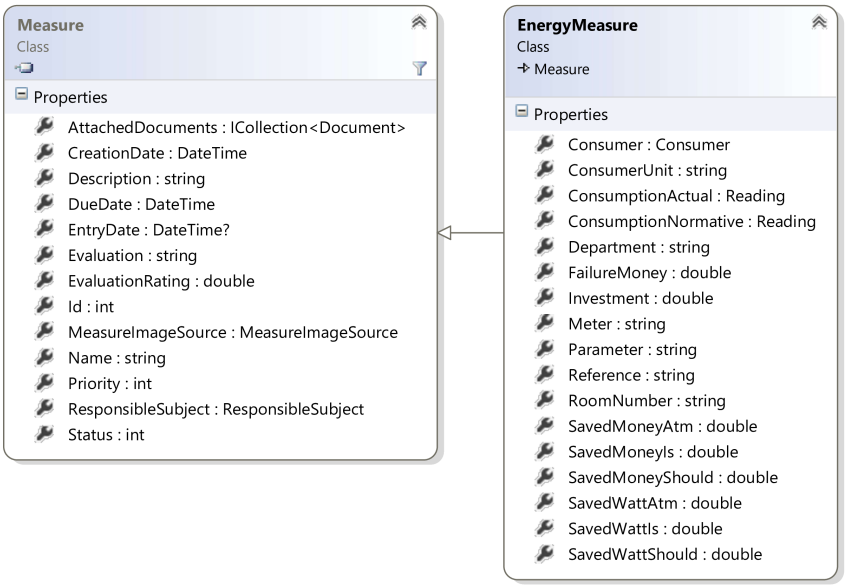

Abbildung 9: Klassenmodell Maßnahmen[88]

5.1.1 Code-First Migrations

Seit Version 4.3 bietet EF zwei Migrationsstrategien an, um in der Datenbank die Modelländerungen widerzuspiegeln. OpenResKit setzt auf die automatische Migration. Hierbei wird das bestehende Datenmodell mit der Datenbank abgeglichen und gegebenenfalls ein Update durchgeführt. Der automatisierte Updatemechanismus weist einige Schwächen auf, zum Beispiel erkennt er keine Umbenennungen von Tabellen oder Spalten, so dass alte Spalten gelöscht und neue automatisch erstellt werden. Zudem kann das Datenbankschema nicht mehr auf eine bestimmte Version wiederhergestellt werden. Alternativ gibt es die eine codebasierte Migration. Hierbei kann ein Skript generiert werden, mit dem ein Upgrade und das korrespondierende Downgrade der gewünschten Migration entspricht. Der Entwickler kann dieses Skript anpassen, um die gewünschten Änderungen, zum Beispiel die angesprochenen Umbenennungen, vorzunehmen. Das Skript beinhaltet einen Zeitstempel, so dass eine chronologische Abfolge der Upgrades gewährleistet ist. Um sicherzustellen, dass sich das angewendete Skript auch auf das aktuelle Datenbankschema bezieht, enthält es eine Art Snapshot des Datenbankschemas. Sollte der Snapshot des Migrationsskriptes nicht mit dem tatsächlichen Datenbankschema übereinstimmen, schlägt die Migration fehl. Aufgrund des modularen Aufbaus von OpenResKit ist das Datenbankschema erst zur Laufzeit

bekannt. Der Entwickler kann also nicht wissen, wie das Datenbankschema, auf das er eine codebasierte Migration anwenden will, aussieht. Somit ist diese Variante zunächst nicht anwendbar. Das gleiche Problem betrifft allerdings auch Entwickler in verteilten Teamumgebungen.[89] Die verschiedenen Entwickler können sich nicht auf die durchgeführten Migrationen der anderen Entwickler beziehen. Mittels eines *Merge-Migration* Befehls werden ausstehende Migrationen, die sich auf unterschiedliche Schemata beziehen, gemeinsam angewandt. Ob sich diese Vorgehensweise auf OpenResKit übertragen lässt, bedarf einer weiteren Prüfung. Von einer Vermischung der automatisierten Methode mit der codebasierten Methode rät Microsoft explizit ab.[90]

Weiter oben wurde die Umsetzung von Vererbungsbeziehungen im Zuge des Code-Refactorings beschrieben. Dadurch musste auch eine Datenbankanpassung vollzogen werden. Durch die automatische Migration und der Verwendung von Type-per-Hierachy als Strategie für Vererbungen ist in der entsprechenden Datenbanktabelle lediglich eine *Discriminator*-Spalte hinzugefügt worden. Bei der Verwendung einer anderen Vererbungsstrategie würden die Daten nicht erhalten bleiben. Der Objekttyp der Einträge in der Measure-Tabelle muss nun manuell in die Discriminator-Spalte eingetragen werden. Mit dem Microsoft SQL Server Management Studio können verschiedene Analyse- und Monitoringaufgaben durchgeführt, aber auch SQL-Skripte ausgeführt werden. Folgendes Skript kennzeichnet alle Objekte in der Tabelle als *EnergyMeasure*. In diesem Anwendungsfall war es nicht nötig, nur bestimmte Objekte so zu klassifizieren, dies könnte aber in einer *Where*-Bedingung im SQL-Skript sichergestellt werden.

Listing 4: SQL-Statement zum Klassifizieren des Objekttyps

```
1 UPDATE [OpenResKit.DomainModel.DomainModelContext].[dbo].[Measures]
2 SET Discriminator = 'EnergyMeasure';
```

5.1.2 Integration mit SSIS

Die Datenmigration der Access-Datei, wurde mithilfe der ETL-Software SQL Server Integrations Services (SSIS) von Microsoft durchgeführt. Da bereits eine Microsoft-Datenbank eingesetzt wird und auch eine Quelldatenbank ein Microsoft-Produkt ist, liegt die Entscheidung nahe, auch das entsprechende ETL-Paket zu verwenden. Zudem steht mit SQL Server Data Tools (SSDT) eine Entwicklungsumgebung für die Integrati-

[89] vgl. [MSDb]
[90] vgl. [MSDa]

onspakte zur Verfügung, die sich komplett innerhalb der Visual Studio IDE ausführen lässt. Es gibt eine Vielzahl von Alternativen, zum Beispiel Talend Open Studio[91] im Open-Source Umfeld oder CloverETL[92] und natürlich die Tools großer Datenbankhersteller wie Oracle.

In diesem Projekt wurden eine Access-Datei in die SQL Express Zieldatenbank einmalig migriert. Die Zielverbindung zur SQL-Datenbank kann mit dem Assistenten hergestellt werden. Das gewünschte Datenbankschema muss bereits vorhanden sein und wird nach dem Code-First-Ansatz erstellt. Die Quellverbindung kann mit einer OLE DB oder ODBC Schnittstelle realisiert werden. Beides sind von Microsoft entwickelte Schnittstellen, die eine einheitliche Datenbankzugriffsschicht bilden. Gegebenenfalls muss für das verwendete Quellsystem ein passender Treiber oder Connector dem Entwicklungssystem hinzugefügt werden. Nachdem beide Verbindungen hergestellt waren, konnten die einzelnen Datenfelder einander zugeordnet werden. Hierbei hätten gegebenenfalls auch Anpassungen vorgenommen werden können, was aber nicht nötig war, da die Zieldatenbank genau auf den Ausgangsdatenbestand angepasst wurde.

5.1.3 Verbindung zur MySQL-Datenbank

Eine große Herausforderung war die Anbindung der MySQL-Datenbank. Für die verwendete Datenbankversion 4.1.18 standen weder bei EF noch für SSIS Konnektoren zur Anbindung zur Verfügung. MySQL bietet selbst keine Konnektoren für die Verwendung in Visual Studio an, da diese 2006 noch nicht vorhanden waren, jedoch bietet MySQL noch Open Database Connectivity (ODBC)-Treiber für Legacy-Versionen der Datenbank an. Weiterhin existieren Third-Party-Bibliotheken, die die gewünschte Funktionalität zur Verfügung stellen können. Der Hersteller Devart bietet mit dotConnect für MySQL eine Bibliothek zur umfangreichen Einbindung in .Net-Projekte und die Nutzung der gesamten EF-Funktionalität.[93] Leider ist nur der reine Konnektor kostenlos verfügbar. Wie eine einfache Verbindung hergestellt werden kann, zeigen die folgenden zwei Codebeispiele.

Listing 5: Erstellung einer Datenverbindung

```
1 using Devart.Data.MySql;
```

[91] https://de.talend.com/
[92] http://www.cloveretl.com/
[93] vgl. [Dev15]

```
2  public class DBConnector{
3    private MySqlConnection Connection;
4    public bool IsConnect() {
5      if (Connection == null) {
6        var connectionString = string.Format ("Server=servdata;
             database={0}; UID=NovaRead; password=*********", "nova2");
7        Connection = new MySqlConnection(connectionString);
8        Connection.Open();
9      }
10     return true;
11   }
12
13   public MySqlConnection GetConnection(){
14             return Connection;
15   }
16 }
```

Mit dieser Methode erhält man eine Datenbankverbindung, auf der beliebige SQL-Befehle ausgeführt werden können. In Zeile 6 wird der Connection-String erstellt, der den Servernamen, Benutzernamen mit Passwort und den Datenbanknamen enthält. Die verwendete Klasse MySQLConnection kommt aus der externen Bibliothek und stellt den Konnektor dar.

Listing 6: Abfrage der Daten mittels SQL-Statement

```
1  ...
2  using Devart.Data.MySql;
3  namespace OpenResKit.Energy
4  {
5    public class NovaGrabber
6    {
7      private readonly DBConnector connector = new DBConnector();
8      public Collection<Consumer> getData()
9      {
10       var collection = new Collection<Consumer>();
11       if (connector.IsConnect())
12       {
13         string query = "SELECT * FROM geraete WHERE
               GERAETE_TYP_GER_TYP_ID = 8;";
14         var command = new MySqlCommand(query,
               connector.GetConnection());
15         var adapter = new MySqlDataAdapter();
16         adapter.SelectCommand = command;
17         var machineDataSet = new DataSet();
18         adapter.Fill(machineDataSet, "geraete");
19         foreach (DataTable myTable in machineDataSet.Tables)
20         {
21           foreach (DataRow myRow in myTable.Rows)
22           {
23             collection.Add (ConsumerFactory.CreateMachine(myRow));
```

```
24              }
25            }
26            ...
27          }
28        return collection;
29        }
30    }
31  }
```

Die Klasse liest Daten aus der Datenbank aus und übersetzt sie in die Domänenobjekte. In Zeile 13 werden mit einem SQL-Befehl alle Geräte abgerufen, die einem bestimmten Typ entsprechen. In einer Schleife werden alle Zeilen des Ergebnisses durchlaufen und eine Factory-Klasse übernimmt die Erstellung der Datenobjekte. Die Methoden der Factory-Klasse müssen bei der Objekterstellung die Spaltennamen kennen, Nullwertüberprüfungen durchführen und gegebenenfalls Datentypen konvertieren. Das folgende Codebeispiel verdeutlicht die Problematik.

Listing 7: Mapping der SQL-Tabelle in Domänenobjekte

```
1  public class ConsumerFactory
2    {
3      internal static Consumer CreateMachine(DataRow myRow)
4      {
5        var machine = new Consumer();
6        machine.Localid = (int) myRow["GER_ID"];
7        machine.LocalGr = (int) myRow["GER_NR"];
8            ...
9        machine.LocalGnr = myRow["GER_GNR"] == DBNull.Value
10         ? ""
11         : (string) myRow["GER_GNR"];
12       machine.Year = (int) myRow["GER_BAUJ"];
13           ...
14       return machine;
15     }
16   }
```

Das Beispiel zeigt, dass eine Datenmigration auch bei sehr alten Datenquellen möglich, aber mit erheblichem Aufwand verbunden ist. Die Verwendung einer kostenpflichtigen Fremdsoftware könnte den Entwicklungsaufwand verringern. Alternativ ist natürlich ein Upgrade der MySQL-Datenbank selbst möglich. Der Entwicklungssprung auf die aktuelle Version ist jedoch so enorm, dass das Upgrade in mehreren Schritten durchgeführt werden muss. Dabei sind ebenso eine Vielzahl manueller Schritte durchzuführen, die sehr aufwändig sind.

5.2 OpenResKit-Client

Den größten Implementierungsaufwand erforderte der Benutzerclient. Der folgende Abschnitt beschreibt zunächst das eingesetzte Architekturmuster und die verwendeten Bibliotheken, um ein besseres Verständnis zu erlangen. Diese stammen zum Teil bereits aus der OpenResKit-Basisversion und wurden so oder so ähnlich weiterverwendet.

5.2.1 Architekturmuster

Der Client verwendet das Windows Presentation Foundation (WPF)-Grafikframework, das Teil der .NET-Bibliothek ist. WPF trennt die Geschäftslogik von der Präsentationsschicht, schreibt aber nicht die Verwendung eines konkreten Musters wie wie Model View Controller (MVC) oder Model View View-Model (MVVM) vor. Dennoch hat Microsoft speziell für das WPF-Grafikframework das Model-View-ViewModel Entwurfsmuster geschaffen, welches eine Variante des MVC-Musters darstellt.[94] Das Model ist hauptsächlich ein Datencontainer, der dem Datenaustausch zwischen den einzelnen Schichten dient. Die grafische Oberfläche wird View genannt und durch die Auszeichnungssprache Extensible Application Markup Language (XAML) definiert. Das View-Model verbindet dieses beiden Schichten, da es Informationen über den Zustand der View und die zu verarbeitenden Daten enthält.

Es gibt Bibliotheken zur Vereinfachung der MVVM-Entwicklung. In diesem Projekt ist *Caliburn.Micro*[95] im Einsatz. Es erleichtert die Entwicklung, da die Abhängigkeiten zwischen ViewModel und View verringert werden. Dabei greift das Framework auf Konventionen zurück. Dies erschwert die Weiterentwicklung für Entwickler, die das Framework nicht kennen, da der Programmablauf schwieriger nachzuvollziehen ist. Caliburn vereinfacht die Komposition verschiedener Sichten, also die Zusammenstellung mehrerer Ansichten, mit dem Ziel, die Wiederverwendbarkeit und Modularisierung zu erhöhen.

Der Datenzugriff auf den vom Server zur Verfügung gestellten WCF-Datendienst ist mit einem .NET-Client einfach zu realisieren. Es kann automatisch eine Proxy-Klasse erstellt werden, womit die Abfragen des Datendienstes durchführbar sind. Zusätzlich ist im Client das Repository-Entwurfsmuster implementiert. Es vermittelt zwischen der

[94] vgl. [Weg13, Seite 583]
[95] https://github.com/Caliburn-Micro

Datenzugriffsschicht, also der automatisch erstellten Proxy-Klasse, und der Domänenschicht.[96] Dies ist eigentlich nicht nötig, kann aber sinnvoll sein, wenn es eine Vielzahl an Domänenobjekten gibt oder die Abfragen aufwändig sind. In diesem Beispiel trifft das zu. Aufgrund der tiefen Verschachtelung der Objekte untereinander, müssen viele *expand*-Befehle ausgeführt werden. Nachfolgendes Codebeispiel verdeutlicht dies. Das Repository erleichtert zudem das Nachverfolgen und Zurücksetzten von Änderungen. Die im Codebeispiel gezeigten Listen vom TypEntityDescriptor und LinkDescriptor enthalten Felder mit denen man geplante, aber noch nicht ausgeführte Änderungen nachvollziehen kann.

Listing 8: Codeausschnitt aus dem Repository

```
1  private void LoadConsumers(){
2    Consumers = new DataServiceCollection<Consumer>(m_Context);
3    var query = m_Context.Consumers
4           .Expand("OpenResKit.DomainModel.Consumer/ConsumerGroup")
5      .Expand("OpenResKit.DomainModel.Consumer/Distributor")
6      .Expand("OpenResKit.DomainModel.Consumer/ConsumerType")
7      .Expand("Readings")
8      .Expand("Room");
9    Consumers.Load(query);
10 }
11 public IEnumerable<EntityDescriptor> Entities{
12   get { return m_Context.Entities; }
13 }
14 public IEnumerable<LinkDescriptor> Links{
15   get { return m_Context.Links; }
16 }
```

5.2.2 WPF-Oberflächengestaltung

Die Umsetzung eines einheitlichen Stils, der sich an den Microsoft Modern Design Richtlinien orientiert, macht die Verwendung eines Frameworks notwendig. Es gibt eine Vielzahl an Bibliotheken[97], die gängige Benutzersteuerelemente im Metro Design anbieten. Ein häufig eingesetztes Framework ist *mahapps.metro*[98], das sehr einfach zu implementieren ist und eine umfangreiche Sammlung an Benutzersteuerelementen zur Verfügung stellt. In Unterabschnitt 3.2.2 wurde auf die mangelhafte Implementierung

[96] vgl. [Fow15]
[97] zum Beispiel MUI (https://github.com/firstfloorsoftware/mui) oder Elysium (http://elysium.asvishnyakov.com/en/)
[98] http://mahapps.com/

der grafischen Oberfläche hingewiesen. Dies machte ein Reverse Engineering anhand der bestehenden Oberflächen erforderlich. Zusätzlich wurden einige Steuerelemente angepasst, um das gewünschte Benutzerverhalten zu erreichen.

In der ursprünglichen Implementierung wird das Aussehen vieler Steuerelemente allgemein festgelegt. Grundsätzlich ist diese Vorgehensweise geeignet, um ein konsistentes Aussehen der Benutzeroberfläche zu erreichen. Allerdings ist darauf zu achten, dass die Festlegungen nicht zu tief in die Anwendung eingreifen und änderbar bleiben. Beispielsweise werden im ursprünglichen Theming für eine Vielzahl an Steuerelementen die Parameter wie Schriftgröße, Schriftart und Farben festgelegt. Gelegentlich sind jedoch Abweichungen nötig. Üblicherweise werden in Microsoft-Anwendungen keine Vektorgrafiken verwendet, um Symbole in verschiedenen Größen darzustellen, sondern die Schriftart Marlett bietet die entsprechenden Symbole skalierbar an.[99] Es ist deshalb nicht sinnvoll, auf einer tiefen Ebene die Darstellungseigenschaften zu verändern. Ohnehin wurde im konkreten Beispiel die Schriftart auf den Standardwert für das Windows-Betriebssystems festgelegt, der auch ohne explizite Nennung herangezogen werden würde, aber dennoch überschreibbar bliebe.

Der ursprüngliche Theming-Mechanismus lädt einen ausgewählten Stil in den Einstellungen. Dort werden beim Start der Anwendung oder bei einer Änderung des Stils zunächst alle Anwendungsressourcen gelöscht, bevor das neue Thema geladen wird. Das betrifft aber nicht nur die Style-Dateien, sondern gegebenenfalls auch Wörterbücher oder Bilder. Um die Kompatibilität mit der ursprünglichen Version und anderen Modulen beizubehalten, wurde dennoch der gleiche Mechanismus weiterverwendet, obwohl kein eigener Stil entwickelt wurde. Die gewünschten Änderungen an der Benutzeroberfläche wurde mithilfe des UI-Frameworks *mahapps.metro* umgesetzt. Die erforderlichen Ressourcendateien sind in einer Datei hinterlegt, die durch das Einstellungsmodul geladen werden kann. Die selbst entwickelten Stile sind weiterhin verwendbar. Wenn keine Angaben zum Stil gemacht werden, wird das vom Framework vorgegebene Aussehen verwendet.

Die Definition von eigenen Ressourcen bietet sich dann an, wenn nicht das Aussehen, sondern das Verhalten beeinflusst werden soll. Bei der Implementierung der Listen ist dies der Fall. Es soll pro Liste immer nur ein Element selektierbar sein und die Auswahl mit einem weiteren Klick auch wieder aufgehoben werden können. Üblicherweise wird dies durch das Drücken der Strg-Taste erreicht. Dies ist aber nur wenigen Nutzern

[99] vgl. [Mic15d]

bekannt. Deshalb wurde der Liste eine sogenannte Attached-Property hinzugefügt, die dieses Verhalten umsetzt.[100]

Listing 9: Verschiedenen Eigenschaften eines Benutzersteuerelements

```
1 <ListBox
2 ItemsSource="{Binding Distributors}"
3 Grid.Row="1"
4 BorderThickness="0"
5 ScrollViewer.HorizontalScrollBarVisibility="Disabled"
6 ScrollViewer.CanContentScroll="false"
7 customControls:ListBoxSelectionBehavior.ClickSelection="True"
8 SelectedItem="{Binding SelectedDistributor}"/>
```

Normale Eigenschaftselemente (Properties) steuern das Aussehen oder Verhalten des Controls. Dies ist beispielsweise die Randdicke in Zeile 4. Darüber hinaus gibt es sogenannte Dependency Properties, die benötigt werden, um Datenbindungen zu ermöglichen oder Abhängigkeiten von Werten aufzulösen. Beispiele sind die Datenbindungen in Zeile 2 und 8. Attached Properties beziehen sich auf das Verhalten oder Aussehen eines anderen Elementes. Die ListBox definiert seine Position im Grid-Element durch die Attached-Property in Zeile 3, ebenso das Verhalten der umgebenden Scrollviewer in den Zeilen 5 und 6. Der Wert einer eigenen Attached-Property wurde in Zeile 7 bestimmt. Die Implementierung in einer Ressourcendatei zeigt nachfolgendes Codebeispiel.

Listing 10: Attached Behavior für eine Liste

```
1  public static class ListBoxSelectionBehavior{
2  ...//Getter und Setter
3    private static void OnClickSelectionChanged(DependencyObject
       dpo, DependencyPropertyChangedEventArgs e) {
4      ListBox listBox = dpo as ListBox;
5      ...
6          listBox.SelectionMode = SelectionMode.Multiple;
7          listBox.SelectionChanged += OnSelectionChanged;
8      ...
9    }
10   private static void OnSelectionChanged(object sender,
       SelectionChangedEventArgs e){
11     if (e.AddedItems.Count > 0){
12       ListBox listBox = sender as ListBox;
13       var newSelection = e.AddedItems[0];
14       foreach (var item in new ArrayList(listBox.SelectedItems)){
15         if (item != newSelection){
```

[100] vgl. [Sta12a]

```
16              listBox.SelectedItems.Remove(item);}}}}
17    public static readonly DependencyProperty ClickSelectionProperty
          = DependencyProperty.RegisterAttached("ClickSelection",
18        typeof (bool), typeof (ListBoxSelectionBehavior), new
              UIPropertyMetadata(false, OnClickSelectionChanged));
19  }
```

Der Auswahlmodus wird auf *multiple* gesetzt, da so die Selektion eines Objektes mit einem weiteren Klick aufgehoben wird. Um die nicht gewünschte Mehrfachselektion zu verhindern, wird die Auswahl für jedes Element, außer dem gerade selektierten, wieder aufgehoben.

5.2.3 Druckfunktion

Bestimmte Daten können auch direkt aus der Anwendung heraus gedruckt werden. Dies ist erforderlich, wenn zum Beispiel ein Maßnahmenplan direkt in einer Produktionshalle benötigt wird. Hierbei wurde die Möglichkeit genutzt, sogenannte Visual-Objekte direkt auszudrucken. Visuals sind Objekte, die durch WPF darstellbar sind. Deshalb kann jede durch eine XAML-Datei definierte Ansicht mit allen Steuerelementen direkt ausgedruckt werden. Um ein ansprechendes Druckbild zu erhalten, wurde nicht die Original-Ansicht verwendet, sondern eine druckbare Oberfläche entwickelt, die gleichzeitig als Druckvorschau dient. In dieser Oberfläche sind zum Beispiel Auswahlboxen durch Textfelder ersetzt, die den ausgewählten Wert enthalten oder Steuerelemente für den Programmablauf, also Speichern- oder Hinzufügen-Buttons, entfernt. Weiterhin wurde die Oberfläche so gestaltet, dass der Platz einer DIN-A4 Seite optimal ausgenutzt wird. Die Implementierung ist einfach zu realisieren.

Listing 11: Druckbefehl

```
1  private void Print_Command(object sender, RoutedEventArgs e)
2     {
3     var printDialog = new PrintDialog();
4     printDialog.PrintVisual(this, "Aktionsplan");
5     }
```

Für den eigentlichen Druckbefehl sind grundsätzlich zwei Zeile Code ausreichend. In Zeile 3 wird zunächst das standardmäßige Druckdialogfenster geöffnet und danach der Druckbefehl ausgeführt. In sehr seltenen Ausnahmefällen passen die anzuzeigenden Informationen nicht auf eine Seite, wenn bestimmte Felder durch eine intensive Befüllung ausgedehnt werden. In diesem Beispiel können sehr viele Untermaßnahmen

vorhanden sein, oder ein Beschreibungsfeld beinhaltet umfangreichen Text. Mit folgendem Quellcode wird die Seitengröße berechnet und die Ansicht entsprechend verkleinert, damit der gesamte Inhalt auf eine Seite passt.

Listing 12: Skalierung eines druckbaren Dokuments[101]

```
1 ...
2 FullControl.ScrollToTop();
3 var capabilities = printDialog.PrintQueue.GetPrintCapabilities
      (printDialog.PrintTicket);
4 var scale = Math.Min(capabilities.PageImageableArea.ExtentWidth /
      ActualWidth, capabilities.PageImageableArea.ExtentHeight /
      ActualHeight);
5 LayoutTransform = new ScaleTransform(scale, scale);
6 var size = new Size(capabilities.PageImageableArea.ExtentWidth,
      capabilities.PageImageableArea.ExtentHeight);
7 Measure(size);
8 Arrange(new Rect(new
      Point(capabilities.PageImageableArea.OriginWidth,
      capabilities.PageImageableArea.OriginHeight), size));
9 ...
```

Diese Vorgehensweise ist schnell umzusetzen, da sie auf den bereits vorhandenen Ansichten basiert. Die Ähnlichkeit des gedruckten Dokuments mit der Programmansicht erleichtert dem Benutzer die Orientierung erheblich. Dennoch ist der Ansatz flexibel genug, um auf die Unterschiede zwischen Desktop-Ansicht und gedrucktem Dokument einzugehen. Es ist mit einer Implementierung des IDocumentPaginator-Interfaces auch möglich, den Inhalt auf mehreren Seiten auszugeben. Jedoch ist die Durchführung aufwändiger und führt nicht immer zu den gewünschten Ergebnissen, da die Stelle, an dem die Seiten getrennt werden nicht ohne Weiteres bestimmt werden kann. Deshalb wurde die einfachere Vorgehensweise implementiert.

5.3 Möglichkeiten des Datenimports

In Unterabschnitt 3.2.1 wurde die Umsetzung von Vererbungsbeziehungen auf der Serverseite erläutert. Der Windows Communication Foundation (WCF)-Datendienst veröffentlicht die Informationen über die angebotenen Daten und stellt eine Schnittstelle bereit, auf die mit dem OData-Protokoll zugegriffen werden kann. Der erweiterte Client kann nur Objekte verarbeiten, deren Typ auch der Erweiterung entspricht. In diesem

[101] Implementierung wurde mithilfe von [Sta12b] erstellt.

konkreten Beispiel kann also die Benutzersoftware nur Maßnahmen vom Typ EnergyMeasure verarbeiten. Maßnahmen, des Basistyps Measure, oder gegebenenfalls anderer abgeleitete Objekte, können jedoch ebenfalls in der Datenbank vorhanden sein. Nachfolgendes Codebeispiel zeigt, wie mittels des `OfType<DerivedType>`-Filters nur die benötigten Typen abgerufen werden.

Listing 13: Laden von abgeleiteten Objekten

```
1 private void LoadMeasures()
2   {
3     var Measures = new
          DataServiceCollection<EnergyMeasure>(m_Context);
4     var query = m_Context.Measures.OfType<EnergyMeasure>();
5     Measures.Load(query);
6   }
```

Bei einem .NET-Client können die Proxy-Klassen für den Datenzugriff automatisch generiert werden, weshalb hier auch diese einfache Abfrage möglich ist. Allgemein müssen, um einen abgeleiteten Typ, abzurufen, der qualifizierte Objektname und der Namespace angegeben werden.[102] Im Browser liefert folgende Abfrage die Metadaten für alle angebotenen Objekte: `http://host/OpenResKitHub/$metadata`

Listing 14: Metadaten der angebotenen Domäne

```
1 <EntityType Name="Measure">...</EntityType>
2 <EntityType Name="EnergyMeasure"
     BaseType="OpenResKit.DomainModel.Measure">
3 <Property Name="RoomNumber" Type="Edm.String" MaxLength="Max"
     FixedLength="false" Unicode="true"/>
4 ...
5 <NavigationProperty Name="Consumer"
     Relationship="OpenResKit.DomainModel.EnergyMeasure_Consumer"
     ToRole="EnergyMeasure_Consumer_Target"
     FromRole="EnergyMeasure_Consumer_Source"/>
```

Im diesem kleinen Ausschnitt des Ergebnisses sieht man den qualifizierten Objektnamen mit allen Attributen und den Namespace. Das allgemeine Schema, um alle Objekte eines abgeleiteten Typs abzufragen, lautet: `http://myodataservice.svc/baseentities/Namespace.DerivedType`. Im konkreten Anwendungsbeispiel lautet die Abfrage: `http://localhost:7000/OpenResKitHub/-Measures/OpenResKit.DomainModel.EnergyMeasure`. Die Abfrage von abgeleiteten Typen ist deshalb interessant, da OData Vererbung erst seit OData V3 un-

[102] vgl. [ODa15]

terstützt.[103] In anderen Microsoft-Produkten, wie zum Beispiel MS Excel, wird Vererbung innerhalb des OData-Protokolls immer noch zweitrangig behandelt. So kann man zwar über einen einfachen Assistenten sehr schnell einen OData-Dienst mithilfe einer Auswahlmaske anbinden. Um abgeleitete Typen zu verarbeiten, muss jedoch der oben beschriebene, genaue Objektname ermittelt und angegeben werden. Fremdschlüsseleigenschaften können nur mithilfe von MS-PowerQuery abgefragt und verarbeitet werden.

5.4 Softwarequalität und Test

Eine hohe Softwarequalität erleichtert die Weiterentwicklung und kann durch verschiedene Maßnahmen unterstützt werden.[104] Es gibt einige konstruktiven Verfahren, die auf den Softwareentwicklungsprozess abzielen. Das kann beispielsweise mit einer hochwertigen Anforderungserhebung beginnen aber auch das Architekturdesign, der Einsatz von Entwurfsmustern oder bestimmter Code-Richtlinien kann damit gehören zu den konstruktiven Methoden. Analytische Verfahren verbessern die Codequalität nicht per se, decken aber Fehler auf. Gerade bei agilen Vorgehensweisen wird großer Wert auf analytische Verfahren gelegt, da Prozess- oder Organisationsvorgaben eher vermieden werden.[105] Zu den wichtigsten Methoden des XP, die die Softwarequalität sicherstellen, gehören Pair Programming, Code Reviews und Refactoring.[106] Unerlässliche Bestandteile sind aber auch Testverfahren, auf die im folgenden Abschnitt näher eingegangen wird.

5.4.1 Unit-Test

Bei XP gibt es zwei Haupttestverfahren.[107] Das erste ist der Akzeptanztest durch den Benutzer. Bei diesem Verfahren, liegen bereits die Abnahmekriterien durch die User-Stories vor, weshalb der Test sowohl vom Entwickler als auch vom Benutzer, einfach durchzuführen ist. Der Test kann mit jedem neuen Release, der eine neue Funktion be-

[103] vgl. [ODa14]
[104] vgl. [Lig09, Seiten 2ff.]
[105] vgl. [Bau13, Seite 56]
[106] vgl. [Bec00, Seiten 47 und 48]
[107] vgl. [Bec00, Seiten 91 ff.]

inhaltet, durchgeführt werden. Fehler können im nächsten Release behoben werden, oder es entstehen neue Anforderungen.

Das zweite Testverfahren ist das Unit-Testing oder Modultest und ist ein automatisiertes Testverfahren. Hierbei wird ein einzelnes Objekt oder Funktion getestet. Abhängigkeiten werden durch Platzhalter ersetzt. Nach Beck muss nicht jede Funktion einem Test unterzogen werden. Typische Fälle sind unklare Schnittstellen, eine aufwändige Implementierung, abzusehende Spezialfälle, oder wenn bei Funktionstests Probleme auftreten. Ebenso soll vor dem Refactoring ein Test vorhanden sein, um die korrekte Funktionalität zu gewährleisten.[108] Ein Beispiel ist der Unit-Test der Verteilerklasse.

Listing 15: Unit-Test der Klasse DistributorModifyViewModel

```
1  using Moq;
2  using NUnit.Framework;
3  namespace Ork.Energy.Tests
4  {
5    [TestFixture]
6    internal class ReadingTest
7    {
8      private DistributorModifyViewModel
           m_DistributorModifyViewModel;
9          private Mock<Distributor> m_DistributorModel;
10     private Mock<IEnergyRepository> m_EnergyRepository;
11     private Mock<IEnergyViewModelFactory> m_EnergyViewModelFactory;
12     [SetUp]
13     public void SetUp()
14     {
15       m_DistributorModel = new Mock<Distributor>();
16       m_DistributorModel.SetupGet(x => x.Name)
17                         .Returns("Test");
18       m_EnergyRepository = new Mock<IEnergyRepository>();
19       m_EnergyViewModelFactory = new
             Mock<IEnergyViewModelFactory>();
20       m_DistributorModifyViewModel = new
             DistributorModifyViewModel(m_DistributorModel.Object,
             m_EnergyRepository.Object,
21           m_EnergyViewModelFactory.Object);
22     }
23     [Test]
24     public void GetName(){
25       Assert.AreEqual(m_DistributorModifyViewModel.Name, "Test");
26     }
27     [Test]
28     public void AddReading(){
29       m_DistributorModifyViewModel.AddNewReading(new object());
```

[108] vgl. [Bec00, Seiten 92 und 93]

```
30          Assert.IsTrue(m_DistributorModifyViewModel.Readings.Count()
                == 1);
31      }
32    }
33  }
```

Zum Test wird das NUnit-Framework[109] verwendet, mit dem sich Komponententests ausführen lassen. Normalerweise werden abhängige Module, wie eine Datenbank oder andere Komponenten, durch isolierte Hilfsobjekte ersetzt. Diese werden je nach Implementierungsgrad Dummy, Fake, Mock oder Stub genannt.[110] Die Hilfsdaten werden mit dem Moq-Framework erzeugt.[111] Bei diesem Komponententest werden ein Verteilerobjekt (Zeile 10), ein Repository (Zeile 11) und eine Factory (Zeile 12) als Hilfsobjekte erzeugt. Die letzte beiden sind Fake-Objekte, da sie inhaltlich zum Test nichts beitragen, sondern lediglich existieren müssen. Das Verteilerobjekt ist ein Stub, da eine Anfrage an dieses Objekt ein vorgegebenes Ergebnis zurückgibt. Damit kann das eigentlich zu testende ViewModel erstellt werden (Zeile 22). Im ersten Test wird die Anzeige des Namens aus dem Verteilerobjekt im Verteiler-ViewModel überprüft. Das Mockobjekt wird in Zeile 18 definiert. Ein weiterer Test ist das Hinzufügen einer Ablesung. Die entsprechende Methode wird im zu testenden ViewModel ausgeführt (Zeile 33) und die Existenz des Objektes daraufhin abgefragt. Die Tests erscheinen möglicherweise zunächst trivial, sind aber so beabsichtigt, da jeweils die kleinsten möglichen Einheiten getestet werden sollen.

5.4.2 Statische Komponenten-Analyse

Die bestehende Software verwendet, wie bereits in Abschnitt Unterabschnitt 3.2.1 beschrieben, im großen Umfang das MEF-Framework. Dies betrifft nicht nur die Server-Komponente, sondern auch die Client-Komponente. Hierbei kann es zu Kompositionsfehlern kommen, die aufgrund der vielen Unterprojekte und der losen Kopplung schwer zu diagnostizieren sind. Auch während der Implementierungsphase traten solche Fehler auf, weshalb die gesamte bestehende Code-Basis analysiert wurde. Hierbei war das Kompositionsanalysetool *Mefx* hilfreich, welches auch von MSDN empfohlen wird. Es „dient in erster Linie dazu, Entwicklern eine Möglichkeit zur Diagnose von Kompositionsfehlern in MEF-Anwendungen zu bieten, ohne dass der Anwendung

[109] http://www.nunit.org/
[110] vgl. [Bau13, Seiten 152 ff.]
[111] https://github.com/Moq/moq4

selbst unübersichtlicher Ablaufverfolgungscode hinzugefügt werden muss.“[112] Mefx kann unter Microsofts eigener Hosting-Plattform Codeplex heruntergeladen werden. In einer einfachen Befehlszeilenumgebung können die zu betrachtenden kompilierten Bibliotheken geladen werden und mögliche Kompositionsfehler gefunden werden. Die Analyse liefert folgendes Ergebnis:

Listing 16: Ergebnis der statischen Codeanalyse zur Aufdeckung von Kompositionsfehlern

```
<Verzeichnis> mefx.exe /dir:Wpf/bin /causes

Ork.Framework.ViewModels.DialogConductorViewModel
Ork.Framework.ViewModels.ShellViewModel
```

Diese Ausgabe weist darauf hin, dass in den Klassen DialogConductorViewModel und ShellViewModel Kompositionsfehler auftreten. Mit einer weiteren Analyse durch den *verbose*-Befehl, kann die genaue Verletzung der Kompositionsbedingung ausgemacht werden. Der Fehler im ShellViewModel verursacht keine Störung und der Fehler im DialogConductorViewModel konnte behoben werden, da dafür nur die Verwendung eines falschen Datentyps ursächlich war. Während der Entwicklungsphase war das Tool allerdings oft hilfreich. Beispielsweise wurde beim Hinzufügen eines neuen Repositorys in einem Teilprojekt das Erstellen des Kontextes zunächst vergessen. Dieser soll wie in anderen Teilprojekten auch, importiert werden. Fehlende Compilerwarnungen und eine lauffähige Software verhindern zunächst das Erkennen eines Fehlers. Auch die reguläre Ausgabe zeigt lediglich an, dass das Repository nicht erstellt werden kann. Erst durch die Kompositionsanalyse erkennt man schnell den fehlenden Import, also in diesem Fall der fehlende Kontext.

[112] vgl. [Mic15c]

6 Ergebnisse

Die Weiterentwicklung der Anwendung erfolgte, wie auch das OpenResKit-Projekt, ohne die Verwendung kostenpflichtiger Bibliotheken und ist öffentlich quelloffen verfügbar.[113] Nachfolgender Abschnitt zeigt die wesentliche Ausschnitte der erzielten Ergebnisse und die beschreibt die größten organisatorischen Herausforderungen bei der Umsetzung des Projektes. Daran schließen sich Verbesserungsvorschläge und Erweiterungsmöglichkeiten an.

6.1 Darstellung der Anwendung

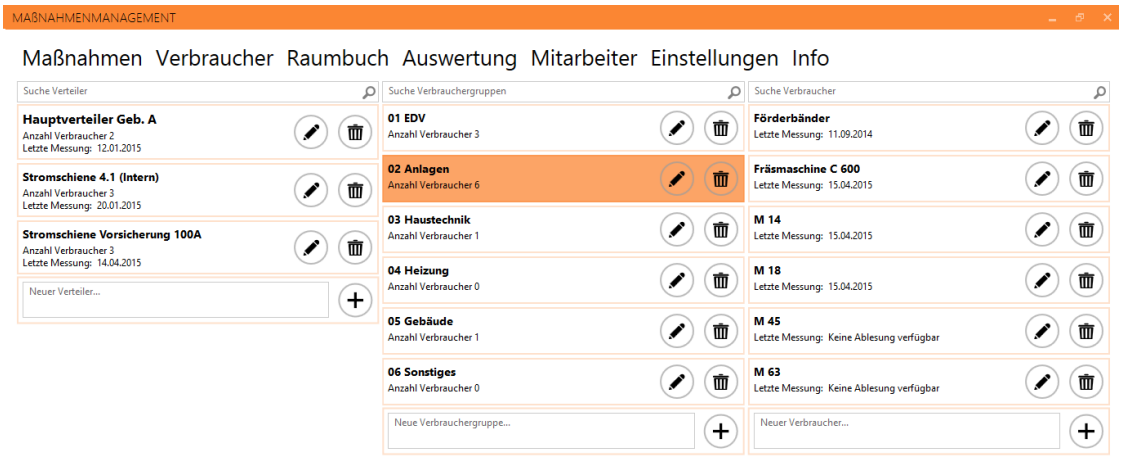

Abbildung 10: Screenshot der Verbraucherübersicht[114]

Die auffälligste optische Neuerung ist die Änderung des Farbschemas mit einem hellgrauen Hintergrund, einer schwarzen Textfarbe und einer orangen Akzentfarbe. Die uneinheitlichen Buttons mit Textfeldern wurden durch runde Buttons mit Piktogrammen ersetzt.

[113] Die Basismodule sind unter http://openreskit.htw-berlin.de/ aufzufinden. Die Erweiterungen sind unter https://github.com/J-Bossi/ verfügbar.

[114] Screenshot der Anwendung

Die inhaltlich wichtigste neue Komponente ist die Verbraucherverwaltung, die in Abbildung 10 gezeigt wird. Hier sind alle Verteiler, Verbrauchergruppen und Verbraucher in je einer Liste dargestellt. Jeder Verbraucher ist einem Verteiler und einer Verbrauchergruppe zugeordnet. Die Durchsuchbarkeit wird durch je ein Suchfeld gewährleistet, das jede Liste durchsuchen kann. Die Suchfunktion erstreckt sich nicht nur auf den Namen des gesuchten Objekts, sondern auch auf weitere für die Klasse relevante Eingabefelder. So kann zum Beispiel auch nach den Typen einer Verbrauchergruppe, oder nach dem Hersteller eines Verbrauchers gesucht werden. Zudem sind die Zugehörigkeiten der Objekte untereinander schnell erkennbar. Die Anzahl der verknüpften Verbraucher ist bei jedem Verteiler und jeder Verbrauchergruppe bereits in der Übersicht zu sehen. Mit einem Klick auf eine Verbrauchergruppe wird die Verbraucherliste gefiltert und nur die zugehörigen Verbraucher angezeigt. Wichtiger ist jedoch die gleichzeitige Filterung der Verteilerliste. Hier werden auch nur die Verteiler angezeigt, die eine entsprechende Verknüpfung mit den angezeigten Verbrauchern haben. Die Filterung berücksichtigt also auch die Verknüpfung aller Objekte untereinander. Abbildung 10 zeigt die 6 Verbraucher, die zur ausgewählten Verbrauchergruppe *02 Anlagen* gehören und die Verteiler, die zu den 6 angezeigten Verbrauchern gehören. Mit einem Klick auf das Mülltonnen-Symbol wird ein Objekt gelöscht, falls keine Abhängigkeiten bestehen. Andernfalls werden diese Abhängigkeiten in einem Hinweisfenster angezeigt.

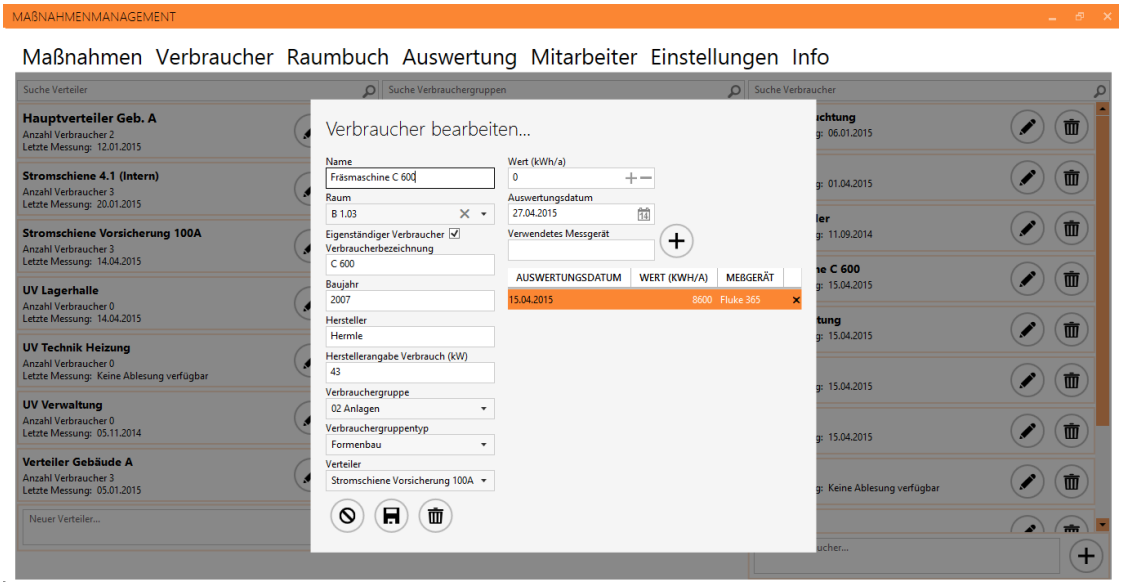

Abbildung 11: Screenshot der Verbraucherübersicht mit geöffneter Maske für Verbraucher[115]

[115]Screenshot der Anwendung

Die meisten Felder eines Objektes sind optional. Jedoch wird immer ein eindeutiger Bezeichner benötigt. Am Ende jeder Liste befindet sich ein Eingabefeld, um neue Verbraucher, Verteiler und Verbrauchergruppen anzulegen. Mit einem Doppelklick auf das Objekt oder mit einem Klick auf das Bearbeiten-Symbol öffnet sich die Detailmaske, in der weitere Daten hinterlegt werden können. Der Hintergrund wird abgedunkelt. Abbildung 11 zeigt die geöffnete Maske für Verbraucher. In Abbildung 12 sind die weiteren Masken für Verteiler und Verbrauchergruppen zu sehen. Die Detailmasken sind möglichst ähnlich aufgebaut, um dem Benutzer eine schnellere Orientierung zu ermöglichen. Zum Beispiel sind die Aktionsbuttons oder das Steuerelement zur Eingabe von Messungen immer gleich aufgebaut.

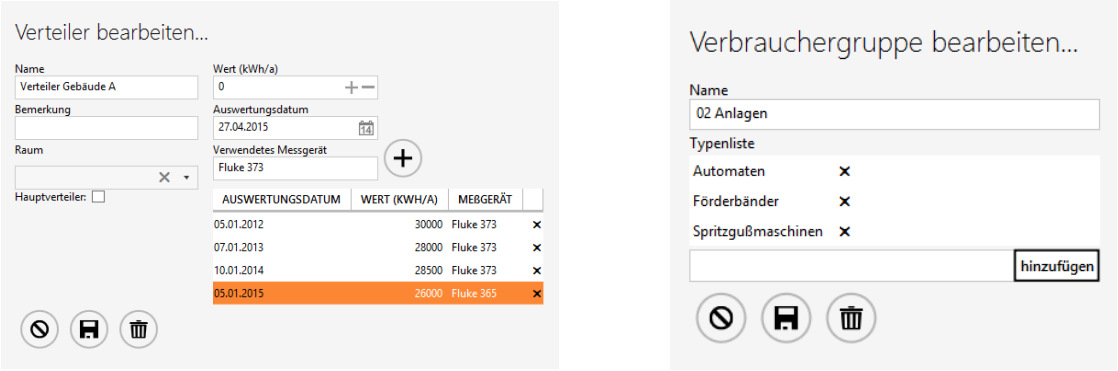

Abbildung 12: Screenshots der Detailmasken für Verteiler und Verbrauchergruppen[116]

Umfangreiche Änderungen wurden im Bereich Maßnahmenmanagement vorgenommen. In der Übersicht (Abbildung 13) wurden einige Bezeichnungen geändert und kleinere Fehler behoben. Die Gruppierung in Kataloge war redundant und wurde durch die Verbrauchergruppen ersetzt. Die Such- und Filterfunktionen wurden entsprechend angepasst. In die Detailmaske Abbildung 14 fließen die Ergebnisse des Integrationsprozesses ein. Zu jeder Maßnahme muss jetzt der Verbraucher eingetragen werden, auf den sich diese Maßnahme bezieht. Damit werden auch die Rauminformationen und Herstellerangaben angezeigt. Die Messwerte aus dem Verbrauchermanagement sind in Dropdown-Listen ersichtlich. Der passende Verbrauchsmesswert beim Anlegen und bei Abschluss der Maßnahme kann aus dieser Liste ausgewählt werden. Außerdem werden die kalkulierten und prognostizierten Einsparwerte nun direkt berechnet und müssen nicht mehr manuell ermittelt und eingegeben werden. Die Oberfläche der Maske wurde komplett überarbeitet und verkleinert, obwohl Eingabefelder dazugekommen

[116]Screenshot der Anwendung

sind. Damit wird die Maske ab einer SXGA-Bildschirmauflösung[117] dargestellt, ohne scrollen zu müssen.

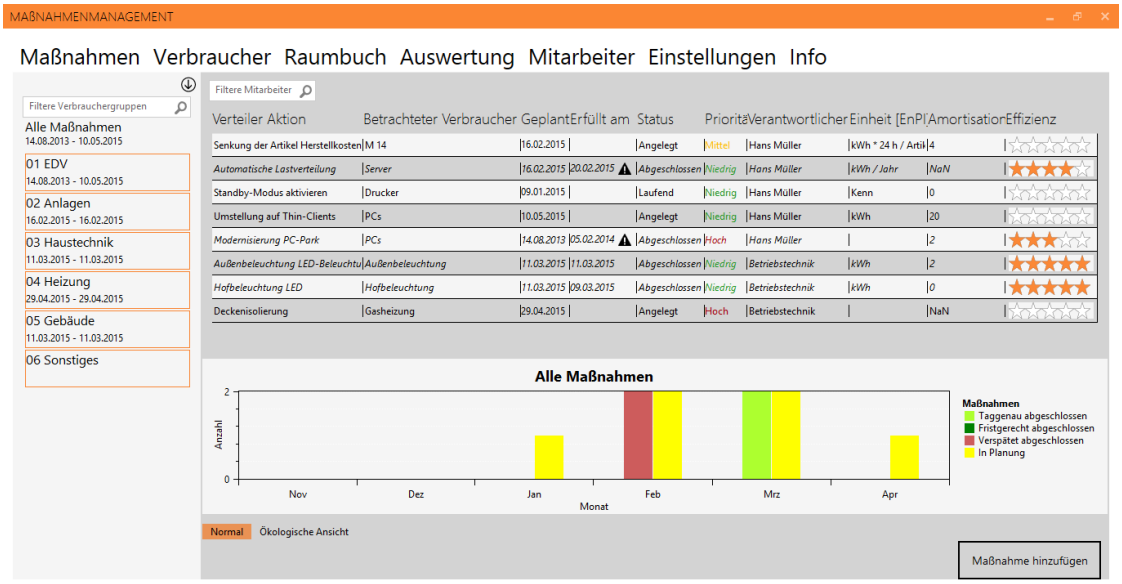

Abbildung 13: Screenshot der Maßnahmenübersicht[118]

Als neue Funktion ist auch eine grafische Auswertung der Energieverbräuche hinzugekommen. In dieser Auswertung kann der Anwender über einen Zeitverlauf die tatsächlichen Messwerte eines Verteilers mit den dazugehörigen Verbrauchern und den prognostizierten Verbrauchswerten vergleichen. Dadurch können einige qualitative Aussagen getroffen werden. Beispielsweise kann bestimmt werden, wie groß der Anteil der betrachteten Verbraucher am gesamten Energieverbrauch ist und wie gut die prognostizierten Einsparungen mit den tatsächlichen Einsparungen übereinstimmen. Mit einem Klick auf die Grafik werden die exakten Werte angezeigt und dadurch auch quantitative Aussagen ermöglicht.

6.2 Organisatorische Herausforderungen

Eine große Herausforderung war der Durchführung als Einzelprojekt. Der Bearbeiter muss eine Vielzahl an Rollen übernehmen, die im Praxisalltag eigentlich zurecht gegen-

[117] 1280 mal 1024 Bildpunkte
[118] Screenshot der Anwendung
[119] Screenshot der Anwendung
[120] Screenshot der Anwendung

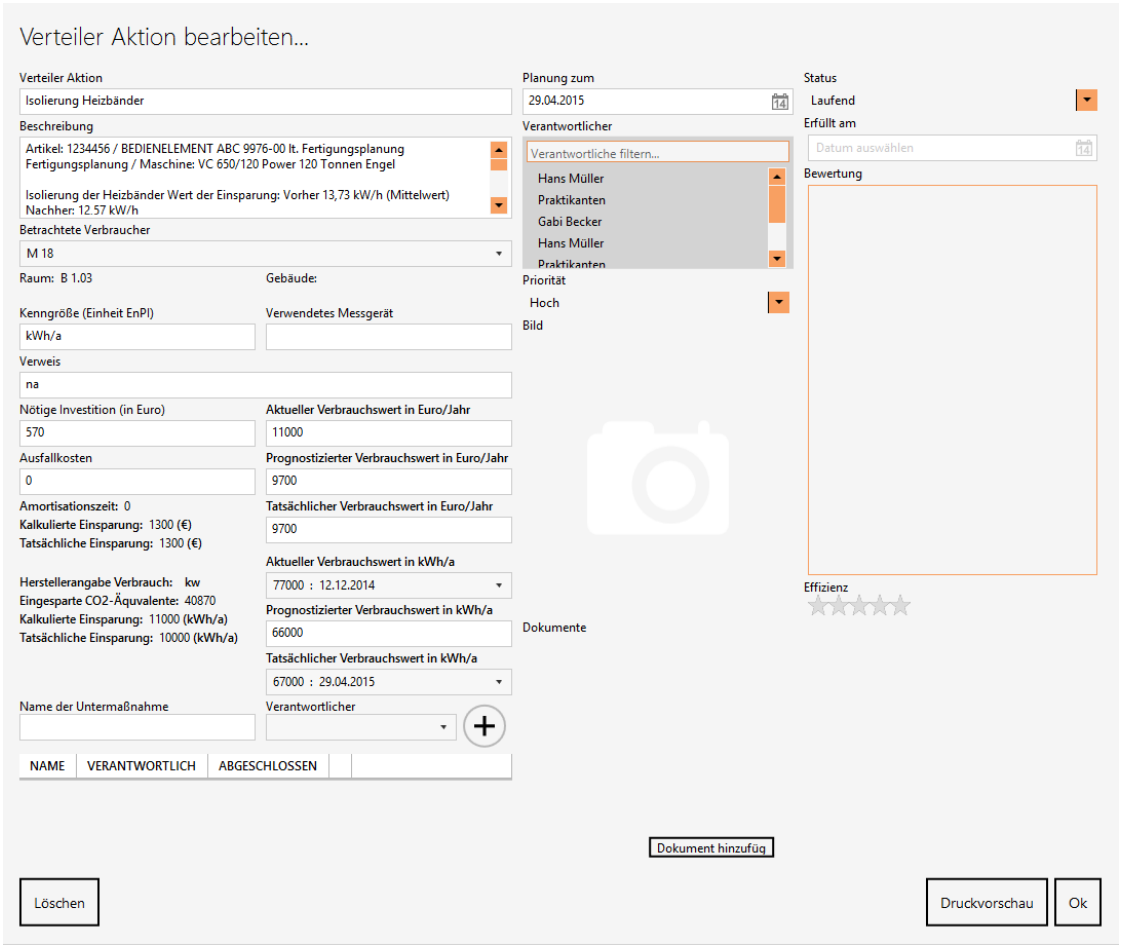

Abbildung 14: Screenshot der Maske um Maßnahmen zu bearbeiten[119]

einander abgegrenzt sind. Die Anforderungserhebung, die Entwicklung und das Testen sind üblicherweise personell voneinander getrennt. Auch bewährte Praktiken wie Pair Programming oder Code Reviews können nicht angewendet werden und gefährden die Softwarequalität. Auch bei einer hohen Selbstdisziplin, kann nicht gewährleistet werden, dass andere Entwickler sich in einer angemessenen Zeit in den Quellcode einarbeiten können. Es wurde darauf geachtet, selbst ein Sprint Review beziehungsweise eine Sprint Retrospektive durchzuführen. Damit kann der Entwicklungsprozess verbessert werden, indem Gutes beibehalten und Probleme beseitigt werden. Auch hierfür ist ein Team zur besseren Reflexion geeigneter. Die gewählte agile Vorgehensweise hat sich im Projektverlauf trotzdem bewährt. Viele Anforderungen wurden durch die Benutzer erst mit der Auslieferung einer neuen Softwareversion gestellt. Darauf konnte schnell reagiert werden. Es ist allerdings sinnvoll, stark auf die Einhaltung von der Sprint-Zyklen zu achten. Das heißt, dass vor einem neuen Zyklus gemeinsam mit dem

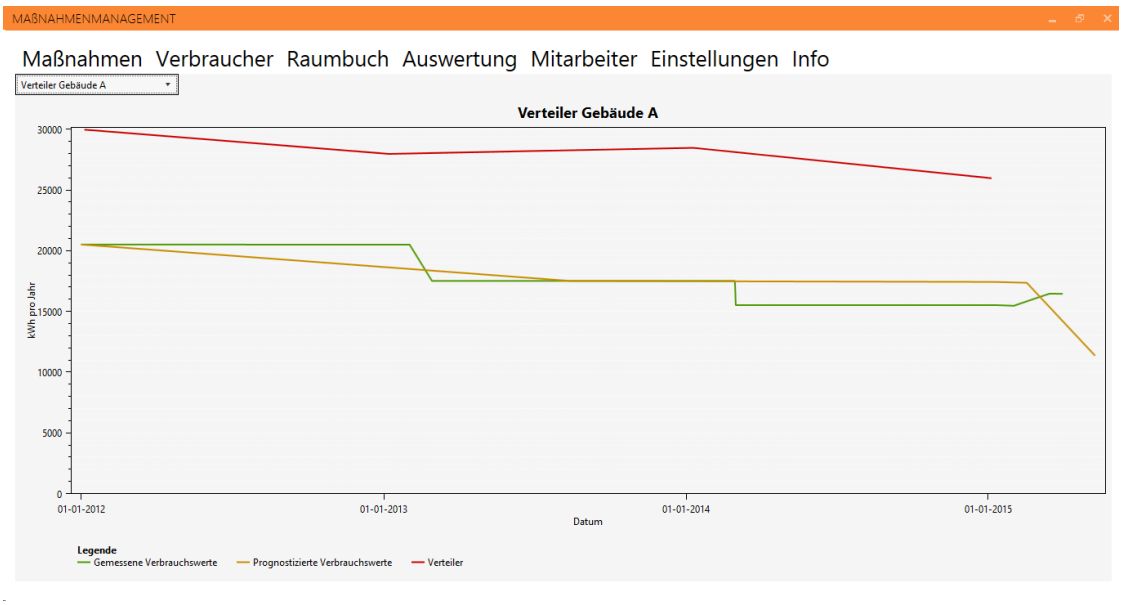

Abbildung 15: Screenshot der grafischen Auswertung[120]

Kunden festgelegt wird, welche Anforderungen umgesetzt werden und währenddessen keine neuen Anforderungen aufgenommen werden.

6.3 Verbesserungs- und Erweiterungsmöglichkeiten

Im gesamten Entwicklungsprozess kam es zu Schwierigkeiten aufgrund veralteter Softwareversionen und Bibliotheken. Die verfügbare Dokumentation der Hersteller bezieht sich nur auf den aktuellsten Entwicklungsstand. Zudem stellen alte Versionen ein Sicherheitsrisiko dar. Für das eingesetzte OpenResKit-Framework wird empfohlen, die wichtigsten Bibliotheken zu aktualisieren. Wie bereits in Unterabschnitt 3.2.1 beschrieben, bietet EF 6 zusätzliche Möglichkeiten zur Domänenerstellung. So kann aus bestehenden Datenbanken der Quellcode generiert werden, um den Code-First Ansatz zu verfolgen. Bei der durchgeführten Implementierung musste das Domänenmodell für die zu integrierende Datenbank erstellt werden und danach ein Integrationspaket mit SSIS erstellt werden. Mit dem Code-First Reverse Engineering Funktionen von EF 6 wird das Domänenmodell automatisch generiert, bleibt aber trotzdem noch anpassbar. Die eigentlichen Daten können mit den standardmäßig verfügbaren Importwerkzeugen in die SQL-Datenbank überführt werden. Für die Integrations- und Erweiterungsaufgaben vereinfacht dies die Erstellung des Domänenmodells enorm, bietet aber trotz-

dem die gleiche Flexibilität. Für EF liegt seit Oktober 2014 eine Betaversion der komplett neu aufgebauten Version 7 vor. Hierin gibt es auch Möglichkeit, Migrationsskripte aus verschiedenen Assemblies zu verwenden. Zudem wird die Datenbank nicht mehr mit dem Datenbankmodell verglichen.[121] Dadurch können codebasierte Migrationen verwendet werden, was die Weiterentwicklung der OpenResKit-Domänen erleichtert und kompliziertere Anpassungen ermöglicht.

Für das OpenResKit-Projekt ist die Entwicklung einer Wissensbasis vorgesehen.[122] Die Erweiterung ist wünschenswert, weil dadurch die erlangten Erkenntnisse auch an andere Entwickler weitergegeben werden können. Gegebenenfalls ist die Erstellung eines Tutorials und die Veröffentlichung auf der Projekthomepage möglich. Damit kann gezeigt werden, wie verschiedene andere Datenquellen im OpenResKit-Hub typischerweise genutzt werden können.

In der entwickelten Anwendung werden die Verbrauchsdaten für die Verteiler erfasst. Diese Verbrauchsdaten bilden die Datengrundlage für die Sankey-Auswertung. Einerseits könnte die Anwendung so erweitert werden, dass eine Excel-Datei exportiert wird, die die passenden Daten für die Sankey-Auswertung liefert. Interessanter wäre jedoch der umgekehrte Weg, bei dem in einer Excel-Datei die relevanten Daten aus dem OData-Dienst importiert werden. Dies wäre eine passende Erweiterung, um noch eine weitere Auswertemöglichkeit für die erfassten Energieverbräuche zu schaffen.

[121] vgl. [Lam14]
[122] vgl. [WKZS14, Seite 4]

7 Fazit

Im Unterabschnitt 2.2.2 wurde auf die Vorteile einer IT-Konsolidierung hingewiesen. Durch die entstandene einfache Anwendung, die konsequent gemeinsam mit den Nutzern entwickelt wurde, konnte die bisherige Komplexität verringert werden. Arbeitsschritte, die bisher in vier verschiedenen Programmen ausgeführt wurden, können nun in einer Anwendung bearbeitet werden. Die Datenstruktur, sowie die darunter liegenden Datenbanken, wurde in einer zentralen Server-Komponente vereinheitlicht. Diese läuft auf einem redundanten System und wird regelmäßig gesichert. Damit wurden erhebliche Qualitätsverbesserungen aus Sicht des IT-Managements erreicht. Die größten Vorteile sind aber aus Nutzersicht gegeben. Alle relevanten Daten des Energiemanagements sind nun in einer Anwendung einseh- und bearbeitbar. Der bisherige manuelle Datenabgleich zwischen verschiedenen Anwendungen entfällt. Weiterhin konnten, aufgrund der zusätzlichen Möglichkeiten, weitere Anforderungen, wie eine Auswertungsfunktion, umgesetzt werden. Der gesamte Dokumentationsprozess der Norm ISO 50001 kann in einer einzigen Anwendung erfolgen.

Dennoch sind die Schwierigkeiten einer Weiterentwicklung im laufenden Betrieb nicht zu unterschätzen. Wegen des heterogenen Datenbestands, unterschiedlicher Entwickler, mangelnder Dokumentation und veralteter Software ist die Erweiterung sehr aufwändig. Bei der Weiterentwicklung waren die Migrationsmöglichkeiten der verwendeten OpenResKit-Datenbank begrenzt. Zudem war die Datenqualität der zu integrierenden Daten mangelhaft und basierte auf veralteten Technologien, was sich aber erst im Laufe des Projektes herausstellte. Aufgrund unterschiedlicher Softwareentwickler geht viel Wissen über die erstellte Software verloren. Sogar auf den ersten Blick nachrangige Probleme, zum Beispiel uneinheitliche Namenskonventionen, erschweren die Analyse der Software. Die festgestellten Mängel der vorherigen Version, sollen die Implementierung nicht abwerten. Vielmehr sinkt die Codequalität durch wechselnde Entwickler und fehlenden Review-Maßnahmen ganz automatisch, wor-

auf auch in Abschnitt 2.2.1 hingewiesen wurde. Eine dauerhafte Wartung der Software würde die Weiterentwicklung erleichtern und beschleunigen.

OpenResKit hat sich für die Umsetzung als geeignet erwiesen. Insbesondere der modulare Aufbau des Servers bietet sich für pragmatische Umsetzung der Erweiterungen und für Integrationsaufgaben an. Grundsätzlich ist durch die Client-Server-Architektur keine enge Verknüpfung zwischen der Datenbasis und der Benutzeranwendung vorhanden. Deshalb wäre es durchaus möglich zukünftig Teile der Datenpflege mit einem anderen Client auszuführen und die Datenbasis weiteren Systemen zur Verfügung zu stellen. Während bisherige OpenResKit-Projekte die Datenerfassungsclients immer mitgeliefert haben, zeigt dieses Projekt, dass die Daten auch aus externen Datenquellen stammen und direkt in dem OpenResKit-Server migriert werden können.

Literaturverzeichnis

[BAF15] BAFA - Bundesamt für Wirtschaft und Ausfuhrkontrolle: *Energiemanagementsysteme: Liste förderfähiger Energiemanagementsoftware.* http://www.bafa.de/bafa/de/energie/energiemanagementsysteme/publikationen/energiemanagementsoftware.pdf. Version: 30.012015, Zuletzt abgerufen am: 29.03.2015

[Bal11] Balzert, Helmut: *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb.* 3. Heidelberg : Spektrum Akademischer Verlag, 2011 (SpringerLink : Bücher). -- ISBN 3827422469

[Bau13] Baumgartner, Manfred: *Agile Testing: Der agile Weg zur Qualität.* München : Hanser, Carl, 2013. -- ISBN 3446431942

[BCF+14] Beck, Kent ; Cunningham, Ward ; Fowler, Martin ; Beedle, Mike ; van Bennekum, Arie ; Cockburn, Alistair ; Grenning, James ; Highsmith, Jim ; Hunt, Andrew ; Jeffries, Ron ; Kern, Jon ; Marick, Brian ; Mellor, Steve ; Schwaber, Ken ; Sutherland, Jeff ; Thomas, Dave: *Manifest für Agile Softwareentwicklung.* http://www.agilemanifesto.org/iso/de/. Version: 21.11.2014, Zuletzt abgerufen am: 08.02.2015

[Bec00] Beck, Kent: *Extreme programming eXplained: Embrace change.* Reading, MA : Addison-Wesley, 2000 (XP series). -- ISBN 0201616416

[BF14] Bourque, Pierre ; Fairley, R. E.: *Swebok: Guide to the software engineering body of knowledge.* Version 3.0. [Los Alamitos, CA] : IEEE Computer Society, 2014. -- ISBN 0769551661

[BRW02] Bennett, Keith H. ; Rajlich, Vaclav ; Wilde, Norman: Software evolution and the staged model of the software lifecycle. In: *Advances in computers* (2002), Nr. 56, S. 1--54. -- ISSN 0065--2458

[Bun12] Bund: *Stromsteuergesetz: StromStG.* http://www.gesetze-im-internet.de/bundesrecht/stromstg/gesamt.pdf. Version: 5.12.2012, Zuletzt abgerufen am: 12.04.2015

[Bun14] Bund: *Erneuerbare-Energien-Gesetz: EEG 2014.* 21.07.2014

[Coc04] Cockburn, Alistair: *Crystal Clear: A Human-Powered Methodology for Small Teams.* HARLOW : Addison Wesley, 2004. -- ISBN 0201699478

[Dev15] Devart: *dotConnect for MySQL.* https://www.devart.com/dotconnect/mysql/. Version: 2015, Zuletzt abgerufen am: 19.04.2015

[FB99] Fowler, Martin ; Beck, Kent: *Refactoring: Improving the design of existing code.* Reading, MA : Addison-Wesley, 1999 (The Addison-Wesley object technology series). -- ISBN 0--201--48567--2

[Fow15] Fowler, Martin: *P of EAA: Repository.* http://martinfowler.com/eaaCatalog/repository.html. Version: 10.04.2015, Zuletzt abgerufen am: 02.05.2015

[Hru14] Hruschka, Peter: *Business Analysis und Requirements Engineering: Prozesse und Produkte nachhaltig verbessern.* München : Hanser, 2014. -- ISBN 3446438629

[HTW13] HTW Berlin: *Quellcode DomainModelContextFactory.cs.* https://github.com/htw-bui/OrkHub/blob/master/OpenResKit.DomainModel/DomainModelContextFactory.cs. Version: 04.09.2013, Zuletzt abgerufen am: 18.04.2015

[HTW15] HTW Berlin: *Maßnahmenmanagement | OpenResKit.* http://openreskit.htw-berlin.de/node/37. Version: 05.04.2015, Zuletzt abgerufen am: 06.04.2015

[Hub14] Huber, Sebastian: *Informationsintegration in dynamischen Unternehmensnetzwerken: Architektur, Methode und Anwendung.* Wiesbaden : Gabler, 2014 (SpringerLink : Bücher). -- ISBN 3658077476

[ISO11] ISO: *ISO 50001 - Energiemanagementsysteme.* 2011. Berlin, 2011

[ISO14] ISO: *The ISO Survey of Management System Standard Certifications 2013: Executive Summary.* http://www.iso.org/iso/iso_survey_executive-summary.pdf?v2013. Version: 2014, Zuletzt abgerufen am: 26.03.2015

[Jam09] James, Alex D.: *Tip 12 - How to choose an Inheritance Strategy.* http://blogs.msdn.com/b/alexj/archive/2009/04/15/tip-12-choosing-an-inheritance-strategy.aspx. Version: 14.04.2009, Zuletzt abgerufen am: 10.04.2015

[Jun06] Jung, Reinhard: *Architekturen zur Datenintegration: Gestaltungsempfehlungen auf*

der Basis fachkonzeptueller Anforderungen. Wiesbaden : Deutscher Universitäts-Verlag/GWV-Fachverlage GmbH, Wiesbaden, 2006 (Wirtschaftsinformatik). -- ISBN 3835090739

[Lac15] Lackes, Richard: *Definition Software Engineering.* http://wirtschaftslexikon.gabler.de/Definition/software-engineering.html. Version: 17.03.2015, Zuletzt abgerufen am: 17.03.2015

[Lam14] Lambson, Brice: *EF7 Migrations: Design-time.* http://www.bricelam.net/2014/12/16/ef7-migrations-designtime.html. Version: 16.12.2014, Zuletzt abgerufen am: 02.05.2015

[Leh80] Lehman, M. M.: Programs, life cycles, and laws of software evolution. In: *Proceedings of the IEEE* 68 (1980), Nr. 9, S. 1060--1076. http://dx.doi.org/10.1109/PROC.1980.11805. -- DOI 10.1109/PROC.1980.11805. -- ISSN 0018--9219

[Lig09] Liggesmeyer, Peter: *Software-Qualität: Testen, Analysieren und Verifizieren von Software.* 2. Heidelberg : Spektrum Akademischer Verlag, 2009. -- ISBN 978--3--8274--2203--3

[LK13] Leopold, Klaus ; Kaltenecker, Siegfried: *Kanban in der IT: Eine Kultur der kontinuierlichen Verbesserung schaffen.* 2., überarb. Aufl. München : Hanser, 2013. -- ISBN 3446438300

[LN07] Leser, Ulf ; Naumann, Felix: *Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen.* 1. Aufl. Heidelberg : dpunkt-Verl, 2007 http://deposit.ddb.de/cgi-bin/dokserv?id=2798715&prov=M&dok_var=1&dok_ext=htm. -- ISBN 978 3 89864 400 6

[LNMG+11] Loos, Peter ; Nebel, Wolfgang ; Marx Gómez, Jorge ; Hasan, Helen ; Watson, Richard T. ; Vom Brocke, Jan ; Seidel, Stefan ; Recker, Jan: Green IT: Ein Thema für die Wirtschaftsinformatik? In: *WIRTSCHAFTSINFORMATIK* 53 (2011), Nr. 4, S. 239--247. http://dx.doi.org/10.1007/s11576-011-0278-y. -- DOI 10.1007/s11576--011--0278--y. -- ISSN 0937--6429

[LRW+97] Lehman, M. M. ; Ramil, J. F. ; Wernick, P. D. ; Perry, D. E. ; Turski, W. M.: Metrics and laws of software evolution-the nineties view. In: *Fourth International Software Metrics Symposium*, 5-7 Nov. 1997, S. 20--32

[Mar09] Martin, Robert C.: *Clean-Code: Refactoring, Patterns, Testen und Techniken für*

sauberen Code. 1. Aufl. Heidelberg, München, Landsberg, Frechen : mitp, 2009. -- ISBN 3826655486

[MBK+12] Mertens, Peter ; Bodendorf, Freimut ; König, Wolfgang ; Picot, Arnold ; Schumann, Matthias ; Hess, Thomas: *Grundzüge der Wirtschaftsinformatik*. 11. Aufl. Berlin : Springer, 2012. -- ISBN 3642305148

[Mic14] Microsoft: *Windows 8 Guidlines*. http://go.microsoft.com/fwlink/p/?linkid=258743. Version: 2014, Zuletzt abgerufen am: 19.04.2014

[Mic15a] Microsoft: *Entity Framework-Fluent-API*. https://msdn.microsoft.com/de-de/data/jj591617.aspx. Version: 2015, Zuletzt abgerufen am: 10.04.2015

[Mic15b] Microsoft: *Entity Framework Version History*. https://msdn.microsoft.com/en-us/data/jj574253.aspx. Version: 2015, Zuletzt abgerufen am: 10.04.2015

[Mic15c] Microsoft: *Kompositionsanalysetool (Mefx)*. https://msdn.microsoft.com/de-de/library/ff576068(v=vs.110).aspx. Version: 2015, Zuletzt abgerufen am: 20.04.2015

[Mic15d] Microsoft: *Marlett*. https://www.microsoft.com/typography/fonts/family.aspx?FID=14. Version: 2015, Zuletzt abgerufen am: 26.04.2015

[Mil14] Miller, Rowan: *EF7 - What Does ``Code First Only'' Really Mean - ADO.NET Blog - Site Home - MSDN Blogs*. http://blogs.msdn.com/b/adonet/archive/2014/10/21/ef7-what-does-code-first-only-really-mean.aspx. Version: 21.10.2014, Zuletzt abgerufen am: 14.04.2015

[MSDa] MSDN: *Entity Framework -- Automatische Code First-Migrationen*. https://msdn.microsoft.com/de-de/data/jj554735.aspx, Zuletzt abgerufen am: 13.04.2015

[MSDb] MSDN: *Entity Framework Code First Migrations in Team Environments*. https://msdn.microsoft.com/en-US/data/dn481501, Zuletzt abgerufen am: 14.04.2015

[MyS] MySQL: *C.1.8 Changes in MySQL 4.1.18 (2006-01-27)*. http://dev.mysql.com/doc/refman/4.1/en/news-4-1-18.html, Zuletzt abgerufen am: 11.04.2015

[ODa14] OData ; Microsoft (Hrsg.): *Open Data Protocol.* http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-ODATA].pdf. Version: 15.05.2014, Zuletzt abgerufen am: 24.04.2015

[ODa15] OData: *URL Conventions.* http://www.odata.org/documentation/odata-version-3-0/url-conventions/. Version: 2015, Zuletzt abgerufen am: 10.04.2015

[PB06] Pellegrini, Tassilo ; Blumauer, Andreas: *Semantic Web: Wege zur vernetzten Wissensgesellschaft.* 1. Aufl. Berlin : Springer, 2006 (X.media.press). -- ISBN 978--3--540--29324--8

[Rau99] Rautenstrauch, Claus: *Betriebliche Umweltinformationssysteme : Grundlagen, Konzepte und Systeme.* Berlin [u.a.] : Springer, 1999 (Springer-Lehrbuch). -- ISBN 9783540661832

[Ros13] Rossak, Ines (Hrsg.): *Datenintegration: Integrationsansätze, Beispielszenarien, Problemlösungen.* München : Hanser, 2013. -- ISBN 3446432213

[RS14] Rupp, Chris ; SOPHISTen: *Requirements-Engineering und -Management: Aus der Praxis von klassisch bis agil.* 6., aktualisierte und erweiterte Auflage. München : Hanser, Carl, 2014. -- ISBN 9783446438934

[RSP09] Rupp, Chris ; Schüpferling, Dirk ; Pikalek, Christian: Deltarequirements -- Auf den Spuren der Projektrealität. In: *Informatik-Spektrum* 32 (2009), Nr. 2, S. 110--117. http://dx.doi.org/10.1007/s00287-008-0312-7. -- DOI 10.1007/s00287--008--0312--7. -- ISSN 0170--6012

[Sch08] Schmidt, Mario: The Sankey Diagram in Energy and Material Flow Management. In: *Journal of Industrial Ecology* 12 (2008), Nr. 1, S. 82--94. http://dx.doi.org/10.1111/j.1530-9290.2008.00004.x. -- DOI 10.1111/j.1530--9290.2008.00004.x. -- ISSN 10881980

[Sne12] Sneed, Harry M.: Nachhaltigkeit durch gesteuerte Software-Evolution. In: Brandt-Pook, Hans (Hrsg.): *Nachhaltiges Software Management* Bd. 209. Bonn : Ges. für Informatik, 2012 (GI-Edition : Proceedings). -- ISBN 978--3--88579--603--9, S. 50--68

[Som11] Sommerville, Ian: *Software engineering.* 9th ed. Boston : Pearson, 2011. -- ISBN 978--0--13--703515--1

[SS13] Sneed, Harry M. ; Seidl, Richard: *Softwareevolution: Erhaltung und Fortschreibung bestehender Softwaresysteme.* 1. Aufl. s.l. : dpunkt.verlag, 2013 http:

//ebooks.ciando.com/book/index.cfm/bok_id/1003044. -- ISBN 978 3 86490 041 9

[SS14] Schwaber, Ken ; Sutherland, Jeff: *Der Scrum Guide: Der gültige Leitfaden für Scrum.* http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-DE.pdf. Version: 24.11.2014, Zuletzt abgerufen am: 08.02.2015

[Sta12a] Stackoverflow: *ListBox with single select and also unselect on click...?* http://stackoverflow.com/questions/5139956/listbox-with-single-select-and-also-unselect-on-click. Version: 22.10.2012, Zuletzt abgerufen am: 05.05.2015

[Sta12b] Stackoverflow: *.net - WPF Printing to fit page - Stack Overflow.* http://stackoverflow.com/questions/7931961/wpf-printing-to-fit-page. Version: 24.01.2012, Zuletzt abgerufen am: 20.04.2015

[Tie13] Tiemeyer, Ernst (Hrsg.): *Handbuch IT-Management: Konzepte, Methoden, Lösungen und Arbeitshilfen für die Praxis.* 5., überarb. und erw. Aufl. München : Hanser, 2013. -- ISBN 3446435573

[Umw12] Umweltbundesamt: *Energiemanagementsysteme in der Praxis: ISO 50001: Leitfaden für Unternehmen und Organisationen.* 1. 2012

[VDI11] VDI Zentrum Ressourceneffizienz GmbH: *Umsetzung von Ressourceneffizienz-Maßnahmen in KMU und ihre Treiber.* Berlin, 2011

[Weg13] Wegener, Jörg: *WPF 4.5 und XAML: Grafische Benutzeroberflächen für Windows.* München : Hanser, 2013 (Net-Bibliothek). -- ISBN 978--3--446--43541--4

[Wik15] Wikipedia ; Wikipedia (Hrsg.): *Open Data Protocol.* http://en.wikipedia.org/wiki/Open_Data_Protocol#Client_libraries. Version: 24.03.2015, Zuletzt abgerufen am: 07.04.2015

[WKZS14] Wohlgemuth, Volker ; Krehan, Peter ; Ziep, Tobias ; Schiemann, Lars: Entwicklung eines Open-Source basierten Baukastens zur Unterstützung und Etablierung der Ressourceneffizienz in produzierenden KMU. In: Wohlgemuth, Volker (Hrsg.) ; Lang, Corinna V. (Hrsg.) ; Marx Gómez, Jorge (Hrsg.): *Konzepte, Anwendungen und Entwicklungstendenzen von betrieblichen Umweltinformationssystemen (BUIS).* Aachen : Shaker, 2014 (Berichte aus der Umweltinformatik). -- ISBN 3844027416, S. 41--57

Anhang

Whiteboard Entwürfe

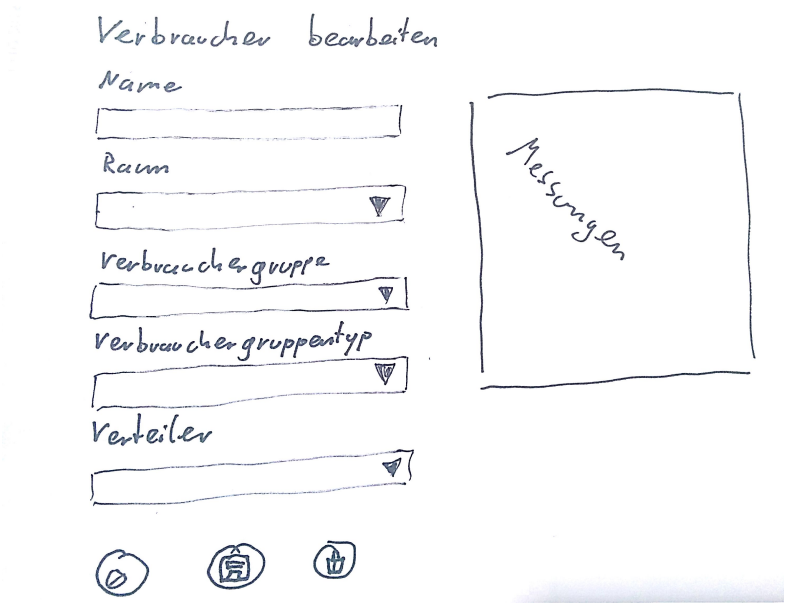

Abbildung 16: Mockup Detailmaske Verbraucher

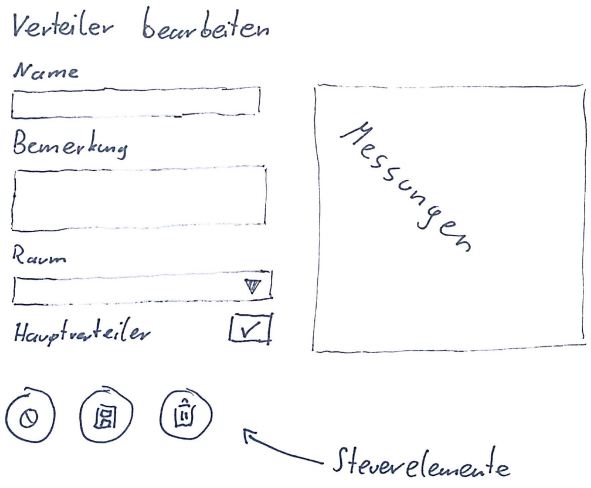

Abbildung 17: Mockup Detailmaske Verbraucher

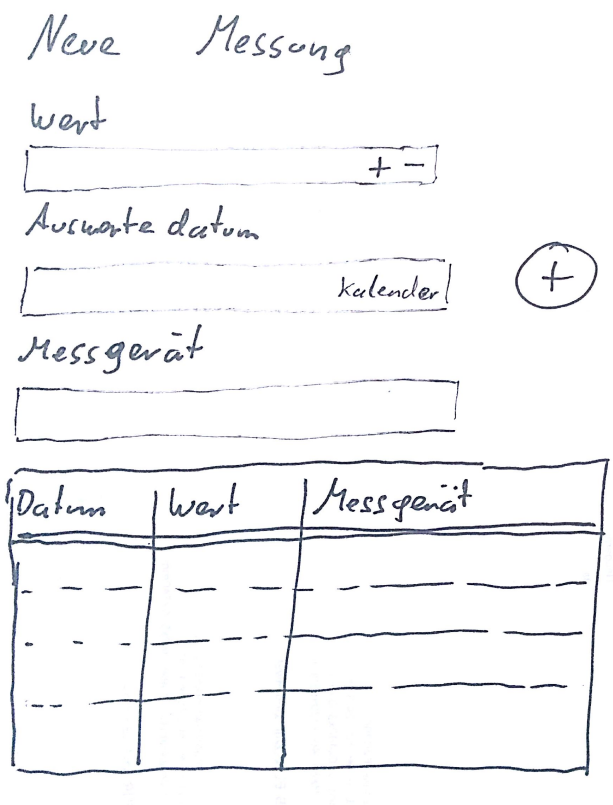

Abbildung 18: Mockup Detailmaske Verbraucher

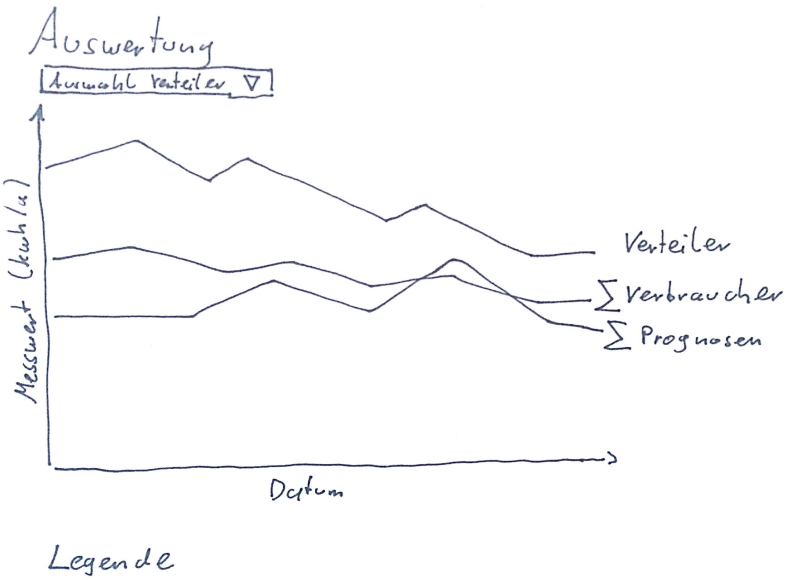

Abbildung 19: Mockup Detailmaske Verbraucher

Klassendiagramm

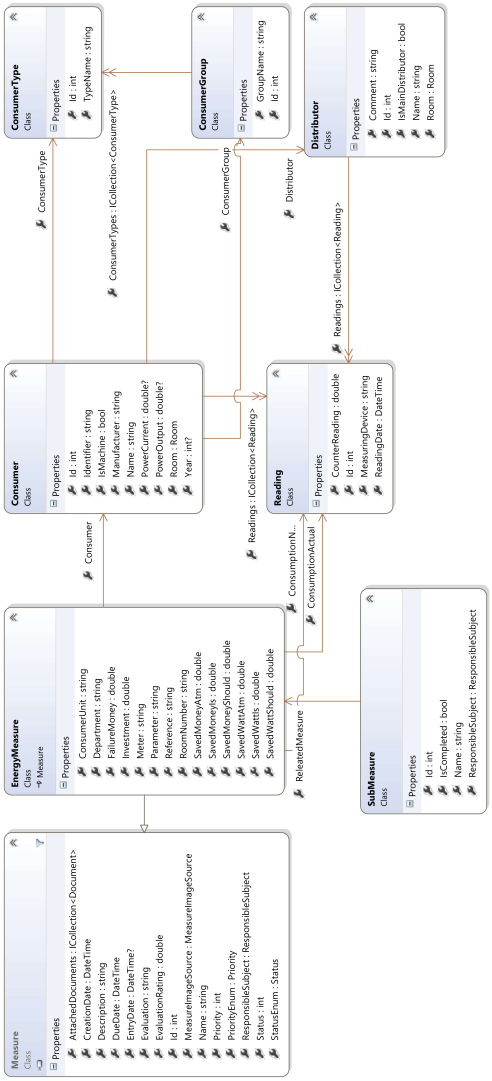

Abbildung 20: Klassendiagramm

Beispiel Klassenvererbung

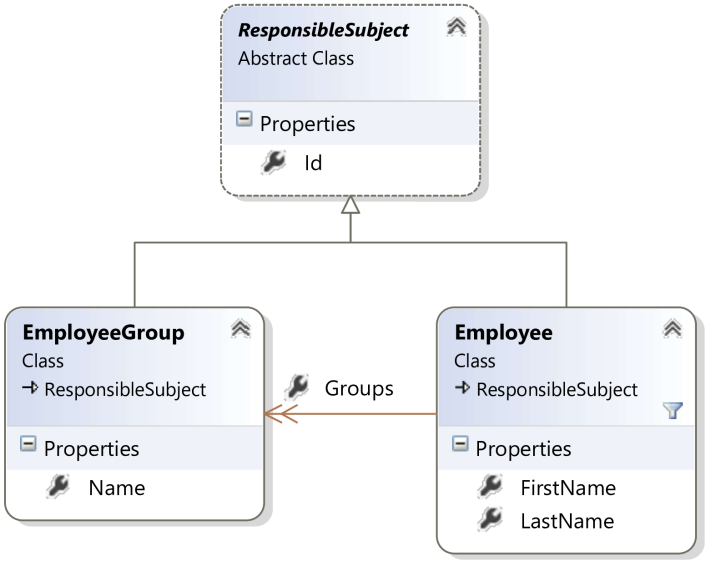

Abbildung 21: Umsetzung von Vererbungsbeziehungen im Entity Framework[123]

[123] eigene Darstellung

Tabellenschema nach Table-per-Hierarchy Ansatz

ResponsibleSubject				
ID	**Name**	**FirstName**	**LastName**	**Discriminator**
1	Entwickler	---	---	EmployeeGroup
2	---	Max	Mustermann	Employee
3	---	John	Doe	Employee

EmployeeEmployeeGroups	
Employee_ID	**EmployeeGroup_ID**
2	1
3	1

Tabellenschema nach Table-per-Tye Ansatz

ResponsibleSubject
ID
1
2
3

EmployeeGroup	
ResponsibleSubject_ID	**Name**
1	Entwickler

Employee			
ResponsibleSubject_ID	**FirstName**	**LastName**	**EmployeeGroup_ResponsibleSubject_ID**
2	Max	Mustermann	1
3	John	Doe	1

Tabellenschema nach Table-per-Concrete-Type Ansatz

EmployeeGroup	
ID	**Name**
1	Entwickler

Employee			
ID	**FirstName**	**LastName**	**EmployeeGroup_ID**
1	Max	Mustermann	1
2	John	Doe	1

Eidesstattliche Erklärung:

Ich versichere, dass ich die vorliegende Abschlussarbeit selbstständig und ohne unerlaubter Hilfe Dritter verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die inhaltlich oder wörtlich aus Veröffentlichungen stammen, sind als solche kenntlich gemacht. Diese Arbeit lag in gleicher oder ähnlicher Weise noch keiner Prüfungsbehörde vor und wurde bisher nicht veröffentlicht.

Berlin, 11. Mai 2015

Ort, Datum

Unterschrift