Praxisbericht

3. Semester

Entwicklung einer Lösung zur Verbesserung der Verwaltung von Change Requests

Studiengang Angewandte Informatik

Duale Hochschule Baden-Württemberg

Stuttgart

Von

Felix Seidel

03.03.2016

Matrikelnummer:	1612461
Ausbildungsfirma:	Hewlett Packard GmbH
Ausbildungsort:	München
Betreuer der Ausbildungsfirma:	Sven Henselmann
	sven.henselmann@hpe.com

Abstract

This paper in German language describes the difficulties of the current billing process regarding an ITIL-based service and makes several approaches to solve this problems with C# and the Microsoft SharePoint API CSOM.

First there is a short introduction into the current situation. Second the use cases are analyzed to create a new solution. Then the planning process starts with a discussion of the platform and the design of the database and the class diagram as well.

Furthermore this paper points out some interesting facts about the implementation of the solution. Finally this work ends with the testing phase, a short conclusion and an outlook into the future.

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Aus den benutzten
Quellen, direkt oder indirekt, übernommene Gedanken habe ich als solche kenntlich gemacht.
Diese Arbeit wurde bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

03.03.2016, München _____

Datum, Ort Unterschrift

Inhaltsverzeichnis

Abkürzungen

API	Application Programming Interface
CAML	Collaborative Application Markup Language
CSOM	C-Sharp Object Model
PEPC	Project-Employee-Price-Category (Zuweisungstabelle in der Datenbank)
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language

1 Einleitung

Hewlett Packard Enterprise (HPE) ist als Konzern unter anderem auf das Outsourcing von IT-Infrastruktur spezialisiert. Einer dieser Kunden, der Teile seine Infrastruktur an HPE outgesourct hat, ist die Airbus Group. Im Rahmen der Leistungserbringung müssen natürlich auch Leistungsnachweise (ein Excel-Sheet, das die geleisteten Stunden dokumentiert) erstellt werden, mithilfe derer der Abrechnungsprozess gestartet werden kann. Die Erstellung der Leistungsnachweise scheint im ersten Moment trivial, aufgrund der großen Mitarbeiterzahl müssen aber auch viele Leistungsnachweise erstellt werden. Dies stellt personellen Aufwand dar und hat damit natürlich auch wirtschaftliche Auswirkungen, außerdem steigt das Fehlerpotential.

1.1 Zielsetzung dieser Arbeit

Aufgrund der wirtschaftlichen Auswirkungen und des Fehlerpotentials besteht natürlich ein Interesse, den Abrechnungsprozess beziehungsweise Teile davon zu automatisieren. Dafür soll zunächst der Status Quo durchleuchtet werden, bevor die potentiellen Anwendungsfälle einer neuen Lösung für dieses Problem analysiert werden. Darauf folgt die technische Planung, die die Wahl der Plattform, das Design der Datenbank und die Modellierung der Klassendiagramme untersucht.
Bei der anschließenden Implementierung sollen Besonderheiten aufgezeigt und diskutiert werden, bevor diese Arbeit klassisch mit einer Zusammenfassung und einem Ausblick abgeschlossen wird.

1.2 Einordnung in übergeordnete Geschäftsprozesse

Im Rahmen des Outsourcings hat der Kunde mit HPE mehrere Verträge abgeschlossen. Diese Verträge laufen über mehrere Jahre und sind dementsprechend eher statisch. Da die IT-Branche aber einem schnellen Wandel unterliegt und dadurch dynamisch ist, wird es mit diesen statischen Verträgen schwierig, auf Änderungswünsche des Kunden einzugehen. Im Rahmen von ITIL existieren daher die so genannten Change Requests

(kurz CR), mithilfe derer relativ schnell Änderungswünsche umgesetzt werden können. Diese Arbeit behandelt den Prozess der Zeiterfassung für die Leistungserbringung im Rahmen dieser Change Requests.

2 Analyse des Status Quo

Derzeit existiert ein SharePoint in der Abteilung, in denen die Mitarbeiter, die die Leistung erbringen (im Folgenden Tracker genannt), ihre Stunden eintragen. Dafür existiert in dem SharePoint eine Liste, die die entsprechenden Spalten für die reguläre Arbeitszeit, Stunden außerhalb der regulären Arbeitszeit und Beschreibung der Angelegenheit vorhält. Einmal im Monat, immer am Ende des Abrechnungszeitraums, werden von einer administrativen Mitarbeiterin mithilfe eines Excel-Makros die Leistungsnachweise daraus erstellt. Das Excel-Makro lädt alle Daten herunter und gibt der Mitarbeiterin dann die Möglichkeit, die Mitarbeiter auszuwählen, zu denen die Leistungsnachweise erstellt werden sollen.
Manche Daten können mit diesem Tool noch nicht geliefert werden, beispielsweise müssen für Reisekosten immer auch Belege vorliegen. Diese werden mit dem SharePoint nicht erfasst und müssen daher von der Mitarbeiterin manuell zu den Leistungsnachweisen hinzugefügt werden.

Die fertigen Leistungsnachweise sendet die Mitarbeiterin per Mail an den Kunden, damit dieser sie abnehmen kann. Erst dann ist die eigentliche Abrechnung möglich. Da in diesem Prozess viele manuelle Schritte notwendig sind, ist er umfangreich und fehleranfällig. Sinnvoll wäre also eine Lösung, die einige Schritte automatisiert und damit beschleunigt.

Im aktuellen Prozess ist es für die Projekt-Owner auch schwierig, einen schnellen Überblick über das Budget eines Projektes zu erhalten. Dies liegt daran, dass in der SharePoint-Liste nur die Stunden gespeichert werden, ohne diese mit konkreten Stundensätzen zu verknüpfen.

3 Analyse der Anwendungsfälle

Das neue Zeiterfassungstool soll möglichst optimal gestaltet werden und den Nutzern den Workflow vereinfachen. Zu diesem Zweck ist es notwendig, das Anwenderverhalten zu studieren und auszuwerten.
Der erste Schritt in diesem Fall ist, die offensichtlichen und von den Stakeholdern ausdrücklich geforderten Funktionen festzuhalten. Gewünscht ist eine Funktion, um Projekte (in diesem Fall eben die Change Requests) und die entsprechenden Ansprechpartner beim Kunden verwalten zu können. Bisher existiert eine Datenbank, in der die relevanten Daten für jedes Projekt gespeichert sind – diese Datenbank steht aber in keinem Zusammenhang mit der derzeitigen Time-&Material-Datenbank. Angedacht ist daher eine Integration, um den Projekten Mitarbeiter zuzuweisen und einen Überblick zu erhalten, wer seine Zeiten zu diesem Projekt erfassen muss.

Dies führt direkt zum nächsten Punkt: Gewünscht werden nämlich außerdem umfassende Zeiterfassungsfeatures, die nicht nur den Tracker in seiner täglichen Arbeit unterstützen, sondern auch den Projektmanagern eine Übersicht über das verbrauchte Budget ermöglichen. Zu diesem Zweck sollen den Mitarbeitern für die Projekte Stundensätze (das, was der Kunde an HPE zahlt) und Kosten (das, was die Mitarbeiter tatsächlich kosten) zugeordnet werden – die Mitarbeiter dürfen diese Zuordnung aber natürlich nicht sehen. Der Tracker soll regelmäßig per Mail daran erinnert werden, seine Zeiten zu den entsprechenden Projekten sorgfältig zu erfassen. Derzeit wird dies manuell und unvollständig durchgeführt, indem verglichen wird, wer in der Vergangenheit auf welches Projekt getrackt hat – einmalige Zuweisungen und kurzfristige Änderungen fallen dabei häufig unter den Tisch.

Die dritte und letzte von den Stakeholdern gewünschte Funktion ist eine automatische Erstellung der Leistungsnachweise. Der Abrechnungsprozess erfordert eine Überprüfung der Zeiterfassung durch den Kunden, dafür existieren Excel-Sheets, die mit den Zeitdaten

befüllt und dann an den Kunden geschickt werden. In einem vorausgegangenen Versuch, den Zeiterfassungsprozess zu vereinfachen, wurde ein Excel-Tool entwickelt, das die Daten aus dem SharePoint lädt und dann in das Formular überträgt. Dieses Tool soll nach Möglichkeit integriert und automatisiert werden.

Die Software wird später hauptsächlich von Trackern genutzt. Daher ist es natürlich sinnvoll, die Bedürfnisse der Tracker möglichst gut abzubilden und zu erfüllen. Ein Tracker wird sehr wahrscheinlich seine Stunden in Intervallen blockweise erfassen (je kürzer das Intervall ist, desto besser für die Budgetplanungen), was dazu führt, dass die Eingabe mehrerer Daten auf einmal vereinfacht werden sollte, beispielsweise ist an dieser Stelle die Erstellung eines Template für jeden Tracker für wiederkehrende Aktivitäten (und daraus resultierend ähnliche Trackings) denkbar.

Die finalen Nutzungsfälle sind in Abbildung 1 dargestellt. Anhand der Farben ist gut zu erkennen, dass das bisherige Tool beziehungsweise der bisherige Prozess um einige Funktionen und Aspekte erweitert werden soll.

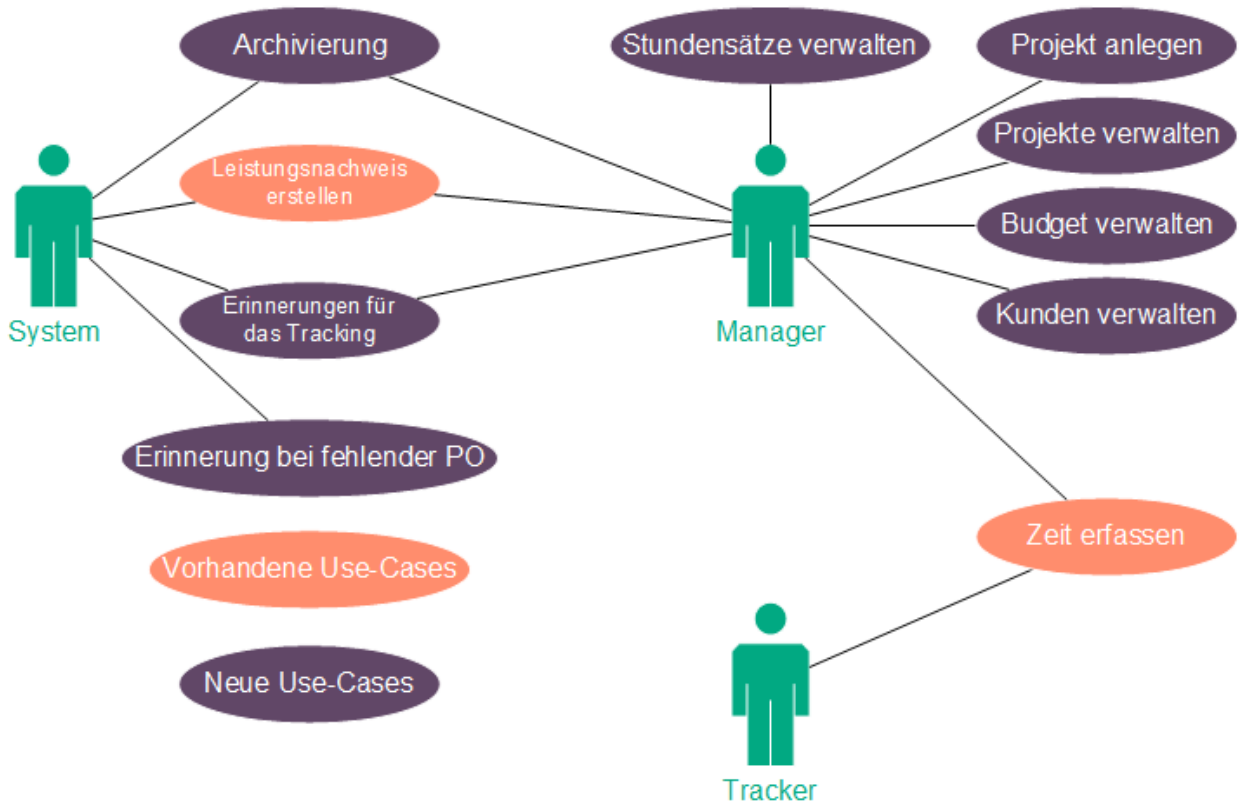


Abbildung 1: Endergebnis der Analyse der Nutzungsfälle in schematischer Darstellung.

4 Analyse der möglichen Plattformen

Hewlett Packard Enterprise hat verschiedene Mittel, mit denen die einzelnen Business Units Anwendungen bereitstellen und nutzen können. Eine Möglichkeit ist die Verwendung eines SharePoints, der in den meisten Fällen (wie beispielweise gemeinsames Bearbeiten von Excel-Files) ausreicht. Der SharePoint lässt sich aber auch mit benutzerdefinierten Anwendungen erweitern, sodass es prinzipiell nur wenige Grenzen gibt [1].

Eine andere Möglichkeit ist die Verwendung einer Private-Cloud von HPE. Der Beantragungs-Prozess ist aber umfangreich: Zum einen benötigt die Applikation, die in der Cloud gehostet werden soll, eine offizielle Genehmigung. Diese Genehmigungsphase kann mehrere Monate dauern und damit ist der Prozess leider noch nicht abgeschlossen, denn dann muss man immer noch die Cloud tatsächlich bestellen und konfigurieren.

Ist die Cloud bereitgestellt, sind Themen wie Benutzer- und Rechteverwaltung der Beginn der Entwicklung – etwas, das aufwendig und durch die Verwendung von SharePoints vermieden werden kann. Insgesamt reichen diese beiden Probleme – der Beantragungsprozess und die umfangreiche Entwicklung der Rechteverwaltung – aus, um diese Plattform für dieses Projekt ausscheiden zu lassen.

Da für die derzeit benutzte Version des Tools SharePoint 2010 im Einsatz ist und die Nutzer den Workflow über den Browser schätzen, sollte eine Möglichkeit gefunden werden, den Client in den Browser zu integrieren – oder aber die Berechnung auf den SharePoint-Server zu übertragen. Dies ist grundsätzlich mit den so genannten SharePoint-Web Parts möglich. Seit SharePoint 2013 existieren auch SharePoint Apps (beziehungsweise SharePoint AddIns). Der Unterschied liegt dabei im Detail: Beide Ansätze können mit C# .NET entwickelt werden, jedoch laufen die Berechnungen bei den Web Parts serverseitig, bei den Apps clientseitig. Diesen Punkt gilt es zwar zu beachten, er spielt aber in diesem Fall keine Rolle, da sowohl benutzerdefinierte Web Parts als auch Apps bei HPE auf den SharePoints deaktiviert sind. Dies wird dadurch ersichtlich, dass die entsprechenden Schaltflächen im SharePoint, mit denen sich Apps oder Web Parts hochladen beziehungsweise aktivieren lassen, ausgegraut sind.

Mit etwas Know-How rund um den SharePoint und JavaScript, ist es auch möglich, mithilfe von JavaScript und dem so genannten Content Editor Web Part clientseitig berechnete Applikationen in die SharePoint-Oberfläche zu integrieren [2]. Dabei platziert man den vorinstallierten Content Editor Web Part auf der entsprechenden SharePoint-Seite und fügt den JavaScript-Code entweder direkt ein oder lässt ihn aus einer Text-Datei laden. Grundsätzlich ist diese Vorgehensweise für ein komplexeres Projekt nicht elegant, sie sollte aber funktionieren. An dieser Stelle jedoch greifen wieder die firmenseitigen Restriktionen, die den Listenzugriff durch die JavaScript-API verhindern und damit durch einen „403: FORBIDDEN“-Code das Script vorzeitig beenden.

Wen man den Content Editor Web Part auf der Seite platziert, stößt man unter anderem auf den „Silverlight“ Web Part. Es ist also auch möglich, Silverlight-Anwendungen für

SharePoint zu entwickeln [3]. Da diese Plattform aber derzeit nach und nach an Bedeutung verliert und in Microsofts neuem Edge-Browser gar nicht mehr unterstützt wird (obwohl Silverlight von Microsoft stammt) [4], wird an dieser Stelle auf tiefergehende Überlegungen zu dieser sterbenden Technologie verzichtet.

Auf Grundlage der bisherigen Erkenntnisse bleibt also nur noch eine Möglichkeit übrig, im Rahmen der gegebenen Umstände eine verbesserte Zeit- und Kostenerfassung umzusetzen, nämlich eine klassische Desktop-Anwendung, die mit dem .NET-Framework entwickelt wird und auf dem SharePoint C# Object Model (CSOM) basiert.

5 Entwurf der Datenbank

Mit der Entscheidung für den SharePoint als Plattform wurde gleichzeitig die Frage geklärt, welches Datenbank-Management-System verwendet werden soll: SharePoint bietet mit seinen Listen grundsätzlich die Möglichkeit, datenbank-ähnliche Tabellen zu erstellen. Dadurch entfällt die Notwendigkeit, eine eigene Datenbank aufzusetzen (für welche man sowieso einen Server bräuchte, was wieder zu dem im vorherigen Kapitel beschriebenen Genehmigungsprozess führt). Es gibt zwar prinzipiell Vorbehalte gegen die Verwendung von SharePoint als Datenbank-Server[5], die Einfachheit der Nutzerverwaltung, des Rechtemanagements, der Genehmigungsprozess für eine Alternative zum SharePoint und der Artikel „SharePoint as a Relational Database“ [6] leisten aber genug Überzeugungsarbeit.

Die Datenbank selbst ist flach organisiert mit der Zuweisung von Projekt, Beschäftigungsart, Mitarbeiter und Stundensätzen im Mittelpunkt.
Für diese vier Punkte sollen eigene Tabellen (ich werde im weiteren Verlauf dieser Arbeit von Listen sprechen, um den SharePoint-spezifischen Begriff zu verwenden) erstellt werden, außerdem für den Kunden, die Zulieferer, die Projektkategorien und die Benutzerrollen.

In jeder Liste sollen die relevanten Daten gespeichert werden. Das fertige Datenbankmodell, wie es nach der Normalisierung [7] aussieht, ist im Anhang zu finden.

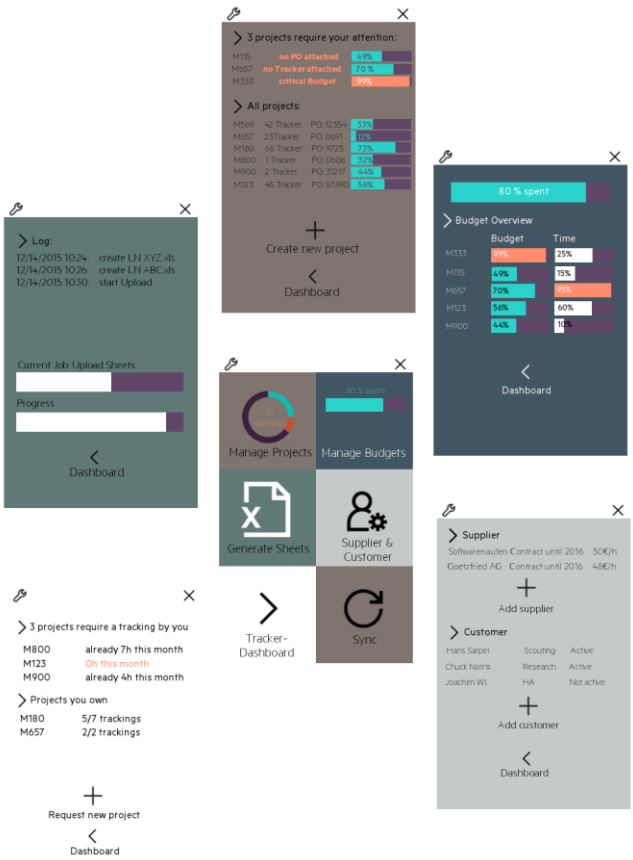
6 Entwurf der Benutzeroberfläche

Der bei der Analyse der Nutzungsfälle gesetzte Fokus auf die Arbeit der Tracker muss natürlich beim Entwurf der Benutzeroberfläche berücksichtigt werden. An dieser Stelle wird es interessant, weil die überwiegende Mehrheit der geplanten Funktionalität ausschließlich dem Manager beziehungsweise Projekt-Eigentümer zur Verfügung steht (beispielsweise Projekt- & Budgetverwaltung), diese Funktionalitäten aber dennoch gut präsentiert werden sollten.
Mit der Trennung von Hewlett Packard in zwei unabhängige Unternehmen zum November 2015 ist für Hewlett Packard Enterprise eine neue Corporate Identity erschaffen worden. Es bietet sich also an, die im Rahmen dieses Rebrandings gesetzten Richtlinien für das neue Tool zu berücksichtigen, sofern dies sinnvoll für die Funktionalität ist. Dies betrifft vor allem Farbgebung und Wahl der Icons.

Modern sind derzeit so genannte Dashboards, die möglichst viele Informationen möglichst übersichtlich auf einer Seite zusammenfassen und bei einem Klick auf Information direkt zur Quelle der Information weitereiten. Das Dashboard besteht also aus Widgets, die die Informationen angemessen präsentieren.
Optisch lehnen sich im Moment viele Programme an der Kachel-Oberfläche von Windows 8 an, einem „flat-look“, wobei der Trend zu leichten, minimalistischen 3D-Effekten zu gehen scheint. [8]

Die grundsätzliche Fahrtrichtung ist also gegeben: Farb- und Iconwahl gemäß der Corporate Identity von Hewlett Packard Enterprise, ein Dashboard-ähnliches Hauptmenü und nach Möglichkeit eine flach wirkende Optik.

Dies und die Gesetze zur Gestaltung von Oberflächen und Software Ergonomie [9] führen zu dem ersten Grobentwurf, der in Abbildung 3 zu sehen ist.



Die Farben in dieser Abbildung entsprechen den in der Corporate Identity festgelegten Farben und sollen die Präsentation der Daten sinnvoll unterstützen: Der Orange-Ton „HPE Orange“ wird bei kritischen Fehlern verwendet, ansonsten sorgen im Kontrast eher kalte Farben für eine neutrale Darstellung.
Erwähnenswert ist, dass an dieser Stelle noch keine Überlegungen zur Optimierung der Benutzeroberfläche für den Tracker eingeflossen sind. Dennoch soll die grundsätzliche Gestaltung in diesem Entwurf die Richtung für die weitere Entwicklung vorgeben.

Abbildung 3: Erster Grobentwurf der Benutzeroberfläche

Deutlich wird schon an dieser Stelle, dass sich die Benutzeroberfläche nicht mit den Hausmitteln von Windows-Forms-Anwendungen, die ja mit Visual Studio und C# ziemlich schnell gestaltet sind, umsetzen lassen.
Es wird die Verwendung der Windows Presentation Foundation notwendig, die umfassende Freiheiten bei der Gestaltung, Animation und beim Arrangement von Windows Anwendungen und deren Steuerelementen bietet.

7 Planung und Entwurf des Klassendiagramms

7.1 Funktionsweise der SharePoint CSOM API

Bevor die weiteren Planungen, insbesondere der Entwurf des Klassendiagramms fortgeführt werden können, muss man sich darüber im Klaren sein, wie die SharePoint CSOM API funktioniert.
Im Rahmen dieser Arbeit wird nur die Bearbeitung von Listen & Rechten auf dem SharePoint benötigt, weshalb auch nur diese Punkte der API erläutert werden sollen.
Die CSOM API handhabt Serveranfragen ähnlich wie etwaige SQL-Bibliotheken.
Für eine Serveranfrage benötigt man einen so genannten `ClientContext`, ein Objekt, das den SharePoint im Code repräsentiert und mit der URL zu eben diesem SharePoint instanziiert wird [10]. Innerhalb dieses Objektes findet sich ein `Web`-Objekt, mithilfe dessen die Listen abgerufen werden können. Dies funktioniert mit der `GetByTitle()`-Funktion des `Lists`-Objektes innerhalb von `Web` [11][12]. Die dadurch erhaltenen Informationen müssen in einem `List`-Objekt gespeichert werden – man kann aber keinesfalls schon damit arbeiten.

Die Liste selbst enthält `ListItems`, die in einer `ListItemCollection` zusammen gefasst sind. Mit der Methode `List.GetItems()` und einer so genannten CAML-Query als Parameter lassen sich bestimmte Items aus der Liste abrufen. Eine CAML-Query ist eine SQL-ähnliche Abfrage, mit der sich Filter beim Abrufen der Daten setzen lassen [13]. Die empfangenen Items werden wieder als `ListItemCollection` zurückgegeben.

Die Daten sind nach `GetItems()` aber immer noch nicht heruntergeladen. Versucht man an dieser Stelle, mit ihnen zu arbeiten, wird zur Laufzeit eine `PropertyOrFieldNotInitializedException` geworfen. Um dies zu verhindern, muss man den `ClientContext` konkret anweisen, die Datenabfrage bei seiner nächsten Verbindung zum SharePoint zu berücksichtigen. Dies geschieht mit der `ClientContext.Load()`-Methode, die als Parameter eben die `ListItemCollection` erwartet, die über die `GetItems()`-Methode spezifiziert wurde. Anschließend kann man

noch weitere Aktionen bündeln und mit der `Load()`-Methode zur nächsten Anfrage hinzufügen. Die Anfrage wird dann mit `ClientContext.ExecuteQuery()` gestartet, was die weitere Ausführung im aktuellen Thread so lange blockiert, bis die Anfrage abgeschlossen ist. Alternativ kann man auch `ClientContext.ExecuteQueryAsync()` verwenden, welche die weitere Ausführung nicht blockiert und nach Erfolg oder Misserfolg die als Parameter übergebenen Events auslöst [14].

Als Beispiel soll das Listing 1 die Verwendung der verschiedenen Methoden verdeutlichen.

```
using System;
using System.Linq;
using Microsoft.SharePoint.Client;

public class MySharePoint
{
        //Lege neuen ClientContext an und definiere die abzurufende Liste
        ClientContext ctx = new ClientContext(„http://mysharepoint“);
        Web web = ctx.Web;
        List list = web.Lists.GetByTitle(„myList“);

        CamlQuery query = new CamlQuery();
        //Diese Query lädt alle Elemente herunter
        query.ViewXml = „<Query />“ ;

        //Rufe die angefragten Elemente ab
        ListItemCollection result = list.GetItems(query);
        //Füge die Listenabfrage zur nächsten SharePoint-Verbindung hinzu
        ctx.Load(result);
        //Verbinde mit SharePoint und bearbeite Anfrage
        ctx.ExecuteQuery();
}
```

Listing 1: Eine beispielhafte Abfrage einer Liste von einem SharePoint

Analog funktioniert das Ändern und Hinzufügen von Listenelementen zu einer Liste: Die Anfrage wird mit den entsprechenden Methoden erstellt und mit `ExecuteQuery` durchgeführt.
Sinnvoll ist es, die SharePoint-Anfrage in ein try-catch-Konstrukt zu verschieben, da es durchaus passieren kann, dass der SharePoint nicht erreichbar ist, die Liste gelöscht wurde oder der Nutzer nicht die Rechte hat, um die entsprechenden Daten anzufragen.

Die Nutzer- und Rechteverwaltung läuft mit der CSOM API ziemlich einfach: Wenn sich der Nutzer im Intranet befindet, werden seine ActiveDirectory-Daten verwendet, um sich zu identifizieren. Dabei muss die Rechte-Verwaltung der SharePoint-Oberfläche berücksichtigt werden, denn der Nutzer kann über die CSOM API natürlich nicht mehr Rechte erhalten, als über den SharePoint definiert ist. Wenn der Nutzer nur Leseberechtigung auf eine Liste hat, kann er auch mit CSOM keine Änderungen an der Liste durchführen.

Auch wenn man sich um die Rechteverwaltung nicht kümmern muss, empfiehlt es sich aus Performancegründen, lokal irgendeine Form der Rechteverwaltung umzusetzen, um unnötige Anfragen an den Server zu verhindern. Im Fall der Time-&Material-Datenbank gibt es nur zwei definierte Rechtegruppen, weshalb eine eigene Tabelle sinnvoll ist, die dem Nutzer die entsprechende Gruppe zuordnet. Zu Beginn der Sitzung wird der Rechtelevel abgefragt und die Anwendung anhand des Levels konfiguriert.

7.2 Entwurf des Klassendiagramms

Das SharePoint CSOM bietet eine relativ komfortable API zur Arbeit mit dem SharePoint. Dennoch ist es nicht unbedingt sinnvoll, die API direkt mit der Benutzeroberfläche zu verknüpfen. Die Gründe dafür sind vielfältig:

1. Der Code wird unübersichtlich und schwer wartbar, wenn durch die direkte Verknüpfung das Model-View-Controller-Muster ignoriert wird.
2. Die Wartezeiten steigen drastisch an, da jeder SharePoint-Zugriff (abhängig von der Verbindungsqualität) mindestens 4 Sekunden benötigt. Durch die direkte

Verknüpfung von API und Benutzeroberfläche können die Zugriffe nicht sinnvoll zusammengefasst und Daten nur mühsam lokal gespeichert werden.

3. Das Tool wird sehr statisch. Wenn sich nur eine Komponente ändert (beispielsweise kann ja die Datenquelle von einem SharePoint auf eine SQL-Datenbank umziehen oder die Darstellung von WPF auf ein HTML basiertes Framework wechseln), muss quasi das ganze Tool neu geschrieben werden.

Sinnvoll ist daher ein modularer Aufbau, bei dem die Kommunikation mit dem SharePoint, die lokale Verwaltung der Daten, die Kommunikation mit etwaigen Export-Tools (beispielsweise für einen Excel-Export) sowie die Darstellung selbst in ein jeweiliges Modul ausgelagert werden.

Wenn sich jetzt ein Modul ändert, sind in den anderen Modulen gar keine oder nur geringfügige Änderungen notwendig: Wenn der SharePoint als Datenquelle ersetzt wird, muss das Kommunikations-Modul ausgetauscht werden und die lokale Verwaltung der Daten an das neue Modul angepasst werden, die übrigen Module bleiben davon aber mehr oder weniger unberührt.

7.2.1 Planung des Moduls SharePointCommunication

Das SharePoint-Modul benötigt eine Klasse, die alle Standard-Aufgaben bündelt: Lese- und Schreibzugriffe auf Listen, Rechteverwaltung sowie das Abfangen von etwaigen Server-Fehlern. Diese Klasse heißt `SharePointConnector` und benötigt eine Schnittstelle zur lokalen Datenverwaltung, um die Daten entsprechend speichern beziehungsweise abrufen zu können. Diese Schnittstelle heißt `IListRepresentation` und stellt alle Methoden bereit, die der `SharePointConnector` für seine Arbeit benötigt. Damit ist das SharePoint-Modul schon abgeschlossen. Ursprünglich war noch eine Klasse `LocalListItem` vorgesehen, die einen Listeneintrag für die weitere Verarbeitung durch die anderen Module unabhängig von dem `ListItem` der SharePoint-API darstellt. Dieser Abstraktionsschritt erschwert allerdings die weitere Verarbeitung der Daten und führt zu unvorhergesehenen Problemen wie Zugriffsversuche auf nicht vorhandene

Spalten, weshalb die Klasse letztlich in der weiteren Planung keine Berücksichtigung mehr findet. Das komplette Modul kann in Abbildung 4 eingesehen werden.

Abbildung 4: Das Modul SharePointCommunication im Planungsstadium

7.2.2 Planung des Moduls DataOrganization

Das zweite Modul, `DataOrganzitaion`, kümmert sich um die lokale Verwaltung der Daten.

Es enthält eine eigene Klasse zu jedem Listentyp (beispielsweise eine Klasse `Customer` für die Customers-Liste), wobei jede Klasse das `IListRepresentation`-Interface implementiert. Diese Klassen stellen zunächst einmal Grundfunktionalitäten bereit, um die Daten aus den Tabellen in die Objekte zu konvertieren. Dafür besitzt jede Klasse

einen Konstruktor, der als Parameter unter anderem ein `ListItem` verlangt, welches ein Listenelement einer SharePoint-Liste repräsentiert. Zu sehen ist das in der beispielhaften Signatur in Listing 2.

```
public Category(ListItem item, DataOverhead overhead)
```

Listing 2: Die Signatur eines Konstruktors des DataOrganization-Moduls am Beispiel der Category-Klasse.

Dieser Konstruktor kennt die Spaltennamen der Liste, die er repräsentiert, und kann damit einfach auf die Spalten des Elements zugreifen und die Werte typsicher in lokale Variablen umzuwandeln. Die Klassen implementieren außerdem die Funktionen, die für das Umwandeln der lokalen Objekte in SharePoint-Listenelemente benötigt werden. Das ist einmal die Methode `GetKeys()`, die als `string`-Array die Spaltennamen zurückgibt. Damit kann das SharePoint-Modul dann die Spalten adressieren. Die `GetValues()`-Methode wiederum liefert die konkreten Werte der Spalten als `object`-Array zurück. Jede Klasse muss außerdem noch weitere Methoden anbieten, die spezifisch auf ihren Funktionen basieren: Für die Projekte beispielsweise muss das bereits verbrauchte Budget berechnet werden, also stellt das die `Project`-Klasse eine Methode `CalculateBudgets()` zur Verfügung.

Das Modul beinhaltet außerdem die Klasse `DataOverhead`, die den Dreh- und Angelpunkt des kompletten Moduls, wenn nicht sogar der ganzen Applikation darstellt. Die Klasse hat ein Member vom Typ `SharePointConnector`, um sich mit dem SharePoint zu verbinden, sowie die Funktionen `GetAllData()`, um alle Daten zu laden, `GetUserRole()`, um vom Server die Benutzerrolle abzufragen und die beiden Methoden `LoadOffline()` und `SaveOffline()`, um den aktuellen Zustand aller Objekte zu serialisieren. Um die Daten zu speichern waren ursprünglich Arrays der entsprechenden Klassen vorgesehen, da die Anwendung allerdings auch dynamisch neue Elemente (beispielsweise ein neues Projekt) erstellen und zum Overhead hinzufügen können soll, ist die Verwendung von linearen Listen an dieser Stelle besser. Im .NET-Framework

erstellt man diese Listen mit der Klasse `List<T>` aus dem Namespace `System.Collections.Generic`. De facto wird nur ein Objekt der Klasse DataOrganization existieren, womit sie per Defintion zum Singleton wird, auch wenn in der Implementierung kein Mechanismus vorgesehen ist, der die Einhaltung der Definition überprüft.

Abbildung 5 zeigt einen Ausschnitt des UML-Diagramms des DataOrganization-Moduls. Zu beachten ist, dass in dieser Version noch mit Arrays anstatt mit Listen geplant wird und die Projektdaten eine eigene Klasse (`Dates`) haben. Da diese Klasse aber eigentlich eine reine Speicherklasse ohne zusätzliche Funktionalität ist und die entsprechende Tabelle im Datenbank-Design aufgrund einer 1:1-Beziehung einfach in die Projects-Tabelle integriert werden kann, wurde auch im Klassendiagram zu einem späteren Zeitpunkt die Datumsklasse in die Klasse `Project` integriert. Eine finale Version des kompletten Klassendiagramms befindet sich im Anhang dieser Arbeit.

Für die Klassen der Oberfläche wurde in der Planungsphase kein UML-Diagramm angelegt, dies geschah aus verschiedenen Gründen. Zum einen sollte das Design der Oberfläche agil geschehen, also in ständiger Iteration. Zum anderen sollte erst das Backend funktionieren, bevor darauf passende Oberflächenklassen entwickelt werden.

Abbildung 5: Ein Ausschnitt aus einer frühen Version des UML-Diagramms für das `DataOrganization`-Modul.

8 Umsetzung

8.1 Implementierung des SharePoint-Moduls

In einem ersten, eher naiven Ansatz implementiert die Klasse `SharePointConnector` die verschiedenen Listenzugriffsmethoden (`GetList()`, `UpdateItem()`, `UpdateMultipleItems()`, `AddItems()`, `DeleteItems()`) wie folgt: Jeder Methode muss mindestens der Listenname übergeben werden. Je nachdem, ob die Methode liest oder schreibt, muss auch noch dieser Zugriff spezifiziert werden: Bei den Lesezugriffen mit einer CAML-Query, bei den Schreibzugriffen mit den entsprechend zu schreibenden Daten. Im Listing 3 ist die Implementierung der GetList()-Methode zu sehen.

```
public ListItemCollection GetList(string listName)
{
    ListItemCollection listItemCollection;

    //Try to get the List from the Server. Possible errors are unauthorized
      access and a wrong list title
    try
    {
        ClientContext contextToUse;
        if (listName == Properties.Resources.Employees)
            contextToUse = additionalClientContext;
        else
            contextToUse = databaseClientContext;

        Web web = contextToUse.Web;
        List list = web.Lists.GetByTitle(listName);

        CamlQuery query = new CamlQuery();
        query.ViewXml = "<View></View>";

        listItemCollection = list.GetItems(query);
        contextToUse.Load(listItemCollection);
        contextToUse.ExecuteQuery();
    }
    catch (ServerUnauthorizedAccessException e)
    {
        throw e;
    }
    catch
    {
        throw new ArgumentException("The list" + listName + "does not exist!");
    }


    return listItemCollection;
```

```
        }
```

Listing 3: Die Implementierung der `GetList()`-Methode

Zu beachten bei dieser Implementierung ist, dass bei einem Listennamen „Employees“ ein anderer ClientContext verwendet wird. Dies ist notwendig, weil die Employee-Daten auch auf einem anderen SharePoint liegen können müssen.

Dabei zeigt sich schnell, dass diese Implementierung aus Performancegründen ungenügend ist. In dem Moment, in dem mehrere Listen auf einmal geladen werden müssen (beispielsweise bei der Initialisierung bei Anwendungsstart), dauert es extrem lange, da für jede Liste `GetList()` einmal aufgerufen werden muss. Dies bedeutet gleichzeitig, dass für n Listen auch n-mal `ClientContext.ExecuteQuery()` aufgerufen werden muss, und das dauert.

Daher ist es sinnvoll, eine Methode zu implementieren, die mehrere Listen mit einer Query lädt. In Listing 4 kann der Teil des Codes eingesehen werden, der sich im Vergleich zu `GetList()` geändert hat.

```
foreach(string listName in listNames)
  {
      if (listName == Properties.Resources.Employees)
      {
          List list =
              additionalClientContext.Web.Lists.GetByTitle(listName);
          CamlQuery query = new CamlQuery();
          query.ViewXml = camlQueries[i];

          listItemCollection[i] = list.GetItems(query);
          additionalClientContext.Load(listItemCollection[i]);
          additionalClientContext.ExecuteQuery();
      }
      else
      {
          List list = web.Lists.GetByTitle(listName);

          CamlQuery query = new CamlQuery();
          query.ViewXml = camlQueries[i];

          listItemCollection[i] = list.GetItems(query);
          databaseClientContext.Load(listItemCollection[i]);
      }
```

```
            i++;
        }

        databaseClientContext.ExecuteQuery();
```

Listing 4: Die entscheidenden Unterschiede der neuen `GetLists()`-Methode im Vergleich zur alten `GetList()`-Methode.

Die Methode erwartet ein `string`-Array als Parameter und iteriert durch dieses Array. Nur wenn die „Employees“-Liste abgerufen wird, werden zwei Queries ausgeführt, ansonsten kann jede Liste mit nur einer Query geladen werden.

Da die anderen Methoden ähnlich aufgebaut sind, soll an dieser Stelle nicht tiefer darauf eingegangen werden. Die Methoden, die in die Listen schreiben, bedienen sich der `IListRepresentation`-Schnittstelle, die die Methoden `GetKeys()` und `GetValues()` anbietet, mit denen auf die Spaltennamen und die entsprechenden Werte zugegriffen werden kann.

8.2 Implementierung des DataOrganization-Moduls

Nach Abschluss der Arbeiten am SharePoint-Modul kann das `DataOrganization`-Modul implementiert werden. Dieses Modul stellt dem Frontend letztlich die Daten zur Verfügung. Alle Klassen bis auf `DataOverhead` müssen IListRepresentation implementieren, um vom `SharePointConnector` mit Daten gefüllt zu werden.
Exemplarisch soll diese Implementation am Beispiel der Klasse `Project` erfolgen.
Alle anderen Klassen (abgesehen von `DataOverhead`) werden dann analog implementiert.

Zunächst einmal besitzt die Klasse öffentliche Properties zu jeder Spalte. Für die Spalten, in denen „echte“ Werte gespeichert werden, existieren entsprechende Strings, Integer oder andere Datentypen. Die Spalten, die mit IDs auf andere Listen verweisen, sind in der Klasse als Objekt der entsprechenden Klasse der anderen Liste vorhanden.

Der Konstruktor von `Project` erwartet, wie bereits im Kapitel über die Planung des Moduls beschrieben, den Overhead und das Item, aus dem das Objekt generiert werden soll. Das Listing 5 zeigt zwei Zeilen aus dem Konstruktor. Analog dazu werden alle anderen Properties Werten zugewiesen, nur die Objekte der anderen Listen werden zu einem späteren Zeitpunkt zugeordnet.

```
Name = (string)Item["projectName"];
FormerNumber = (string)Item["formerNumber"];
```

Listing 5: Beispielhafte Zuweisung aus dem Konstruktor von `Project`

Außerdem existiert noch ein leerer Konstruktor, um die Erzeugung von neuen Projekten zu ermöglichen. Die von der Schnittstelle definierten Methoden `GetID()` und `SetID()` sind einfache Getter & Setter, aus diesem Grund soll an dieser Stelle auf eine Diskussion mit Quellcode verzichtet werden.

Die Methoden `GetKeys()` und `GetValues()` geben die Schlüssel beziehungsweise die Werte zurück. Dies kann in Listing 6 nachvollzogen werden.

```
public string[] GetKeys()
{
       return new string[] {"projectArea", "projectOwner", "projectCustomer",
       "projectName", "formerNumber", "projectStatus", "projectDescription",
       "contracturalConsequences", "orderNumber", "revenueBudget", "isArchived",
       "costBudget", "WBSElement", "allowTravel", "dailyTravelRate" , "dateRecieved",
       "dateAnsweredAirbus", "dateAnsweredHPE", "targetDate", "approvalDate",
       "validUntil", "intervallStart"
        };
}
public object[] GetValues()
{
       return new object[] {Area.GetID(), OwnerHPE.GetID(), OwnerAirbus.GetID(), Name,
       FormerNumber, Status, Description,
       ContracturalConsequences, PurchaseOrder, RevenueBudget, isArchived, CostBudget,
       WbsElement, AllowTravel, DailyTravelRate, DateRecieved,
       DateAnsweredAirbus, DateAnsweredHPE, TargetDate, ApprovalDate, ValidUntil,
       IntervallStart
       } ;
}
```

Listing 6: Die Funktionen `GetKeys()` und `GetValues()` in ihrer Implementierung

Damit die Projekte später sortiert werden können, implementiert die Klasse, genau wie alle anderen Klassen des Moduls, `IComparable`. Daher implementiert die Klasse auch `CompareTo()`.

Nicht alle Klassen des Moduls, aber außer Project auch noch `PEPC`, `Employee` und `AdditionalCosts` implementieren die Methode `SetObjects()`.
Diese durchsucht alle erzeugten Objekte und verknüpft diese gemäß den Relationen in den Listen. Da diese Methode je nach Rechtelevel des Nutzers unterschiedlich arbeitet, ist sie entsprechend lang. Es soll an dieser Stelle daher nur auf zwei Besonderheiten eingegangen werden:

1. Wenn eine eins-zu-viele-Relation umgesetzt werden soll, beispielweise kann ein Kunde mehreren Projekten zugewiesen werden, werden innerhalb der SetObjects-Methode in Project immer alle Kunden durchsucht. Dies ist mit einer foreach-Schleife umgesetzt und kann in Listing 7 nachvollzogen werden.
2. Wenn eine viele-zu-eins-Relation umgesetzt werden soll, beispielsweise können mehrere Trackings einem Projekt zugeordnet werden, ein Tracking kann aber immer nur zu einem Projekt gehören, ist es sinnvoll, die Liste der zu durchsuchenden Objekte nach jedem Treffer um den Treffer zu reduzieren. Dies ist nicht mit einer foreach-Schleife möglich, da damit die Collection verändert wird. Daher muss an dieser Stelle auf eine auf das letzte Element der Collection initialisierte und bis null dekrementierende for-Schleife zurückgegriffen werden [15]. Dies kann auch in Listing 7 nachvollzogen werden.

```
foreach(Customer customer in overhead.Customers)
{
	if(customer.ID == Convert.ToInt32(localListItem["projectCustomer"]))
	{
		OwnerAirbus = customer;
		break;
	}
}
for (int i = overhead.ListPepc.Count - 1; i >= 0; i--)
{
	if (Convert.ToInt32(overhead.ListPepc.ElementAt(i).localListItem["projectID"]) ==
			ID)
```

```
        {
                overhead.ListPepc.ElementAt(i).ProjectObject = this;
                Trackings.Add(overhead.ListPepc.ElementAt(i));
                overhead.ListPepc.RemoveAt(i);
        }
}
```

Listing 7: Die beiden verschiedenen Möglichkeiten, die Zuweisung zu implementieren.

Auf die in Listing 7 enthaltenen Listen des `DataOverhead`-Objektes `overhead` wird im nächsten Kapitel eingegangen.

Neben diesen Methoden implementiert `Project` auch noch zwei Methoden, die für die Funktionalität des Projektes selbst wichtig sind. Dies sind einmal `CalculateBudgets()`, welche aus allen Trackings und den damit verknüpften Stundensätzen das verbrauchte Budget einmal auf Kosten- und einmal auf Einnahmen-Seite berechnet, und `HasError()`, welche prüft, ob dem Projekt noch wichtige Daten fehlen (beispielsweise eine Purchase-Order-Nummer, die zur Abrechnung benötigt wird) oder ob sonst Fehler im Projekt vorliegen (beispielweise das Erreichen der Budget-Obergrenze).

Die zentrale Klasse des Moduls ist `DataOverhead`. Diese stellt, beinahe schon in Manier von globalen Variablen, alle SharePoint-Daten in Form von `IListRepresentation` implementierenden Klassen zur Verfügung. Gleichzeitig stellt `DataOverhead` diesen Klassen eine Instanz von `SharePointConnector` für Listenzugriffe bereit. Diese wird direkt im Konstruktor von DataOverhead initialisiert (der Konstruktor von `SharePointConnector` wiederum verbindet sich mit dem SharePoint, um Konfigurationsdaten aus einer „Setup“-Liste zu laden).

Die Klasse stellt mit `GetAllData()` eine Methode zur Verfügung, die alle bisher lokal vorhandenen Daten löscht, neu herunterlädt, in Objekte konvertiert und in den entsprechenden (lokalen) Listen speichert sowie diese verknüpft. Diese Methode ist mit über 300 Zeilen die umfangreichste des gesamten Projektes, ist aber dennoch leicht verständlich. Je nach Rechtelevel (dies muss vorher mit `GetUserRole()` abgefragt werden!) werden die CAML-Queries definiert. In der Regel werden alle (nicht archivierten)

Daten abgefragt, außer der Nutzer ist Tracker, dann werden die CAML-Queries so definiert, dass nur die den Nutzer betreffenden Daten geladen werden.
Manche Queries sind abhängig von anderen Daten: Der Tracker lädt beispielsweise nur die Projekte herunter, denen er auch als Tracker zugeordnet ist. Dementsprechend müssen zunächst die Zuordnungen geladen werden (PEPC), bevor daraus dann die Query für die Projekte gebildet wird.

8.3 Implementierung der Oberfläche

Die GUI besteht in WPF aus zwei Teilen: dem XAML-Code, der HTML/CSS-ähnlich die Ansicht strukturiert und gestaltet, und damit verknüpfter C#-Code, der die dahinterstehende Logik festlegt.
Aufgrund der fehlenden Planung der Oberfläche hat die Implementierung an dieser Stelle einen experimentellen Charakter. Es soll dennoch ein Überblick über den groben Ablauf gegeben werden.

Beim Start der Anwendung öffnet sich der Ladebildschirm, der ein neues `DataOverhead`-Objekt erzeugt. Die damit einhergehende Initialisierung des `SharePointConnectors` lässt die GUI einfrieren - so lange, bis die Setup-Liste geladen ist. Wenn dann noch die `GetAllData()`-Methode ausgeführt wird, friert die GUI erneut ein. Daher ist es unerlässlich, die Initialisierungen und den Daten-Download in einen eigenen Thread auszulagern [16][17]. Die Syntax dafür kann in Listing 8 eingesehen werden.

```
new Thread(AllTheWork).Start();
```

Listing 8: Exemplarische Auslagerung der Methode `AllTheWork()` in einen neuen Thread.

Wenn man in diesem Thread auf die GUI zugreifen will, beispielsweise nach Abschluss aller Arbeiten, wenn das Dashboard geladen werden soll, muss dies mit dem Dispatcher geschehen. Die Syntax hierfür findet sich im Listing 9.

```
Application.Current.Dispatcher.BeginInvoke(
        System.Windows.Threading.DispatcherPriority.Normal, new Action(StartDashboard));
```

Listing 9: Zugriff auf die GUI aus einem anderen Thread.

Beim Starten des Dashboards werden die Listen des `DataOverhead` nach fehlerhaften Projekten durchsucht und diese dem Nutzer angezeigt. Dafür besitzt `Dashboard`, wie die meisten anderen Fenster, eine Methode `SetItems()`, welche nach der Initialisierung aufgerufen wird und die Steuerelemente im Fenster platziert.

Zum Thema Steuerelemente: WPF bietet umfangreiche Möglichkeiten, Steuerelemente zu individualisieren und eigene hinzuzufügen. Diese sollen hier kurz aufgeführt werden:

1) Anpassung des Aussehens mit Templates [18]. Dafür wird mit XAML-Code Farbe, Form und optisches Verhalten (beispielsweise bei einem `MouseOver`-Event) festgelegt.
2) Bündelung vorhandener Steuerelemente zu einem sogenannten `UserControl`. Damit lassen sich immer wiederkehrende Blöcke von Steuerelementen zu einem neuen Element zusammenfassen (beispielsweise ein „Adress“-Block, bestehend aus mehreren Eingabefeldern für Straße, Hausnummer, Postleitzahl und Stadt). Diesem `UserControl` kann auch noch mit C#-Code logisches Verhalten zugeordnet werden.
3) Erstellung komplett eigener Steuerelemente, sogenannte `CustomControls`. Dies stellt das umfangreichste Verfahren dar und kommt daher auch nur zur Anwendung, wenn eine gewünschte Funktionalität nicht von einem bereits vorhandenen Steuerelement unterstützt wird. Im Rahmen dieser Arbeit wird auf diese Methode nicht zurückgegriffen.

Mit der ersten Methode können die Buttons so konfiguriert werden, dass sie den im Kapitel über die Oberflächengestaltung beschriebenen Anforderungen genügen.
Mit der zweiten Methode können Labels und TextBoxen so gruppiert werden, dass sie einer Zeile einer Liste aus dem SharePoint entsprechen. Dann lassen sich diese Zeilen mittels C# Code ziemlich einfach der Oberfläche hinzufügen.

Die Oberfläche benötigt dafür nur ein `StackPanel` mit einem eindeutig zugewiesenen Namen. Das `StackPanel` ist ein Steuerelement, das andere Steuerelemente untereinander oder nebeneinander anordnet. Dabei wird jeder Inhalt „abgeschnitten“, der über die Grenzen des `StackPanels` hinausreicht. Im Code muss dann eine neue Instanz des zu platzierenden Elements erstellt und dem `StackPanel` zugewiesen werden (siehe Listing 10).

```
allProjects.Children.Clear();
foreach (Project project in overhead.Projects)
{

       ProjectListItem pli = new ProjectListItem(this);
            //…
       allProjects.Children.Add(pli);
}
```

Listing 10: Hinzufügen eines neuen Elementes. Zunächst werden alle bisher platzierten Elemente gelöscht, um Kopien zu vermeiden. Zu beachten ist, dass zwischen Erzeugen der Instanz und Hinzufügen zum StackPanel „allProjects“ die Eigenschaften des Objektes noch festgelegt werden können.

Dieses Vorgehen kann exemplarisch für alle Fenster übernommen werden. Unterschiede gibt es nur in den Konstruktoren der Steuerelemente, diese sind abhängig von den darzustellenden Daten.

8.4 Implementierung der Excel-Funktionalitäten

Auch für die Implementierung der Excel-Funktionalitäten wurde im Vorfeld kein Plan erstellt, weshalb auch diese einen experimentellen Charakter aufweist.
Das in der vorhandenen Lösung verwendete Excel-Makro soll komplett abgelöst werden, da dies sehr langsam und statisch geworden ist. Grundsätzlich wird von Microsoft keine API angeboten, um Excel-Dateien mit C# zu erstellen, glücklicherweise gibt es aber eine GPL-lizenzierte API von EPPlus [19]. Der Umgang mit dieser API ist, bis auf syntaktische

Unterschiede, die aus den verschiedenen Sprachen resultieren, identisch mit der Verwendung von VBA in Excel.
Die Vorbereitung der Daten für die Erstellung der Leistungsnachweise geschieht in der Oberfläche. Der Nutzer wählt Projekt, Mitarbeiter und Intervall aus und anhand dieser Daten wird eine Liste aller für den Leistungsnachweis relevanten Trackings erstellt und an das Excel-Modul übergeben.

Die Klasse `CreateAssessment` ist für die Erstellung des Leisungsnachweises zuständig und bietet nach außen eine statische Methode `SaveToFile()` an. Die Implementierung dieser Methode ruft zunächst eine Methode `PrepareDates()` auf, welche anhand des Abrechnungsintervalls die darin enthaltenen Daten (im Sinne von Zeit) in dem Template vorbereitet. Dann werden alle entsprechenden Tracking-Daten eingefügt und mit dem `Calculate()`-Befehl alle vorhandenen Formeln des Sheets berechnet.

Zu beachten ist, dass die API mit der TEXT-Formel von Excel nicht umgehen kann. Diese wird aber (unnötigerweise) im Template des Leistungsnachweises verwendet, um aus der aktuellen Systemzeit eine eindeutige Buchstabenkombination zu berechnen, die für die Rechnungserstellung gebraucht wird. Die Formel hat aber nur optische Auswirkungen und kann daher gestrichen werden.

Neben der Erstellung der Leistungsnachweise kann das Excel-Modul auch Projekt-Daten und Tracking-Daten exportieren. Dies geschieht eher formlos und soll allen Nutzern die Möglichkeit geben, für sie relevante Informationen dauerhaft lokal zu speichern. Die dafür benötigten Klassen `ExportProjects` und `ExportTrackings` sind analog zu `CreateAssessment` aufgebaut und sollen daher an dieser Stelle nicht weiter diskutiert werden.

9 Änderungen am Plan und den Anforderungen während der Entwicklung

Natürlich sind in einem agilen Prozess, was die Entwicklung von moderner Software in der Regel immer ist, Änderungen an der Planung auch zur Implementierungszeit erforderlich. Im Rahmen dieses Projektes betrifft dies die Module `SharePointCommunication` und `DataOrganization`. Die Änderungen in `SharePointCommunication` wurden in den vorangegangenen Kapiteln bereits beschrieben (Wegfall von `LocalListItem`, Optimierung der Abfrage mehrere Listen auf einmal), die der `DataOrganization` noch nicht.

Innerhalb dieses Moduls sind weitere Klassen notwendig geworden, die im Planungsstadium noch nicht enthalten waren. Dies ist einmal eine Klasse „`AdditionalCosts`“, die es den Projekt-Ownern ermöglicht, Korrekturbeträge sowohl auf Kosten- als auch auf Einnahmenseite zu den Projekten hinzuzufügen. Die Notwendigkeit der Klassen `ArchivedProject` und `ArchivedProjectPEPC` wurde im Planungsstadium schlicht übersehen. Diese beiden Klassen erben von `Project` beziehungsweise von `PEPC` und repräsentieren, wie der Name vermuten lässt, archivierte Projekte und deren `PEPC`s.
Bei der Planung wurde auch nicht berücksichtigt, dass archivierte Projekte und deren `PEPC`s in der Implementierung nicht automatisch geladen werden, sondern explizit vom Nutzer angefragt werden müssen. Um dann die Daten richtig zuzuordnen, kann nicht die normale `SetObjects()`-Methode verwendet werden, weil diese sich auf die automatisch geladenen Daten bezieht. Dies generiert die Notwendigkeit der neuen Klassen, um die Methode `SetObjects()` überschreiben zu können. Außerdem wurde die Zuordnungsklasse PEPC geändert. Da es sinnlos ist, einem Tracker zu einer Kategorie im selben Projekt verschiedene Stundensätze zuzuordnen (eine Kategorie repräsentiert immer ähnliche Arbeit und wird de facto auch immer gleich berechnet), ist die Relation Price-Category jetzt eine eins-zu-eins-Relation. Lediglich aus Datenschutzgründen (der Tracker darf den Stundensatz nicht sehen, braucht aber die Kategorie für seine Zeiterfassung) sind die entsprechenden Felder nicht in eine einzige Liste geflossen.

10 Testphase und Release

Natürlich ist die Anwendung während der Implementierung schon diversen Alpha-Tests unterzogen worden. Dennoch ist es sinnvoll, eine Beta-Testphase zu starten, um eben möglichst alle Fehlerfälle zu überprüfen und gegebenenfalls auszubessern. Während dieser Testphase sind diverse Verbesserungsmöglichkeiten offengelegt worden, die zum Teil noch im Rahmen dieser Arbeit umgesetzt werden können. Da diese Verbesserungsmöglichkeiten das Layout einzelner Fenster und verschiedene kleinere Funktionen betreffen, die nur geringen Einfluss auf die in den vorangegangenen Kapiteln beschriebenen Aspekte haben, sollen diese im Folgenden nicht weiter ausgeführt und diskutiert werden.

Der andere Teil der Verbesserungsvorschläge ist sehr umfangreich und kann aus zeitlichen Gründen im Rahmen dieser Arbeit nicht behandelt werden. Diese Verbesserungsvorschläge beinhalten zum Großteil Komfortfunktionen, welche die Eingabe und die Verwaltung der Daten erleichtern sollen, wie beispielsweise eine Suchfunktion.

Mit dem Ende der Entwicklungsphase stellt sich die Frage, wie das Tool am besten veröffentlicht wird. Zur Diskussion stehen hierbei zwei Möglichkeiten:

1. Im neuen Tool werden nur neue Projekte berücksichtigt. Dies führt dazu, dass für die Verwaltung der alten Projekte noch die bisher gegebenen Werkzeuge verwendet werden müssen, gibt aber gleichzeitig eine größere Kontrolle über die Verteilung des Tools, vor allem bei möglichen Fehlerfällen nach der Testphase.
2. Alle alten Daten werden in das neue Tool übernommen. Dies stellt einen großen, einmaligen Aufwand dar, bedeutet aber gleichzeitig, dass in Zukunft nur noch dieses eine Tool zur Verwaltung der Daten verwendet werden muss. Nachteilig ist zu beurteilen, dass für die alten Projekte zur korrekten Budgetkalkulation die bis

zum Tage der Migration erzeugten Einnahmen und Kosten manuell als Korrekturbetrag (`AdditionalCosts`) ergänzt werden müssen.

Die Entscheidung hierüber liegt nicht im Rahmen dieser Arbeit. Da beide Punkte sowohl Vor- als auch Nachteile bringen und es ganz offensichtlich keine „richtige“ Entscheidung gibt, sollte der Leser die Frage nach der besseren Option für sich persönlich beantworten.

11 Zusammenfassung und Ausblick

Die Entwicklung eines Tools dieser Größe ist komplex und bringt viele verschiedene und auch interessante Facetten mit sich. Während der Entwicklungszeit wurden über 6000 Zeilen C#-Code geschrieben, zusammengefasst (also auch der von Visual Studio generierte Code beziehungsweise der XAML-Code für die Darstellung) umfasst das Projekt über 37000 Zeilen Code. Dafür war zu Beginn eine Auseinandersetzung mit den Use-Cases und den möglichen Plattformen notwendig, ebenso wie die Planung der Datenbank sowie der wichtigen Klassen des Backends. Für die Implementierung spielt die CSOM-API eine wichtige Rolle. Nur mit ihr sind Listenzugriffe mit einer Client-Applikation überhaupt möglich.

Das finale Tool ist einsatzbereit. Es bietet alle von den Stakeholdern gewünschten Funktionen und kann dementsprechend so verwendet werden, wie ursprünglich vorgesehen. Trotzdem sollte für zukünftige Projekte dieser Größe mindestens ein weiterer Entwickler zur Verfügung stehen, um die Komplexität herunter zu brechen, Fehler zu vermeiden und die Qualität des Codes und des Endergebnisses zu erhöhen. Deutlich wird dies an der Zahl der Codezeilen: Wikipedia rechnet mit einer durchschnittlichen Produktivität von 10 bis maximal 50 Zeilen Code pro Personenarbeitstag [20]. Geht man von diesem Wert aus, ergibt sich eine Projektdauer von (gerundet) 138 Personenarbeitstagen. Das Projekt selbst hatte aber eine Projektdauer von 53 Personenarbeitstagen, daraus folgt, dass eigentlich für ein Projekt

dieser Größe zwei, wenn nicht sogar drei Entwickler beziehungsweise Projektbeteiligte notwendig gewesen wären.

Mit einer größeren Mannstärke hätten sich auch alle Wünsche noch im Rahmen dieser Arbeit berücksichtigen lassen. Dafür ist nun ein Nachfolgeprojekt geplant, das das vorhandene Tool verbessert und erweitert. Langfristig steht damit dem Ziel, das Abrechnungsverfahren zu automatisieren, in technischer Hinsicht nichts mehr im Wege.

12 Literaturverzeichnis

[1] Microsoft (2015): "Ausführen grundlegender Vorgänge unter Verwendung von SharePoint 2013-Clientbibliothekscode" https://msdn.microsoft.com/de-de/library/office/fp179912.aspx Einsichtnahme: 01.12.2015

[2] Microsoft (2011): "Using the JavaScript Object Model in a Content Editor Web Part (Kumar Abhishek Verma)" https://blogs.msdn.microsoft.com/sharepointdev/2011/04/14/using-the-javascript-object-model-in-a-content-editor-web-part-kumar-abhishek-verma/ Einsichtnahme: 10.12.2015

[3] Microsoft (o. J.): "Walkthrough: Creating a Silverlight Web Part

that displays OData for SharePoint”
“https://msdn.microsoft.com/de-de/library/hh285093.aspx”
Einsichtnahme: 10.12.2015

[4] Breithut, Jörg (2015): “Microsoft beendet Silverlight: Watchever und Sky unter Druck – SPIEGEL ONLINE”
http://www.spiegel.de/netzwelt/web/microsoft-beendet-silverlight-watchever-und-sky-unter-druck-a-1041978.html
Einsichtnahme: 10.12.2015

[5] Wright, Chris (2013): „3 things not to do with SharePoint“
http://betanews.com/2012/09/05/3-things-not-to-do-with-sharepoint/
Einsichtnahme: 09.12.2015 & 20.01.2016

[6] Blodgett, Matt (2014): „SharePoint as a Relational Database“
http://www.spmeta.net/2014/02/sharepoint-as-relational-database.html
Einsichtnahme: 09.12.2015 & 20.01.2016

[7] Kersken, Sascha (2003): „Kompendium der Informationstechnik“
Bonn 2003

[8] Kennedy, Erik D. (2014): „7 Rules for Creating Gorgeous UI (Part 1)“
https://medium.com/@erikdkennedy/7-rules-for-creating-gorgeous-ui-part-1-559d4e805cda#.rzwj7tmeg
Einsichtnahme: 10.12.2016

[9] Lambertz, Bernd (o.J): „Software Engineering I“, Manuskript einer Vorlesung an der DHBW Stuttgart, Stuttgart 2015

[10] Microsoft (o.J.): „ClientContext class“ im MSDN https://msdn.microsoft.com/library/office/microsoft.sharepoint.client.clientcontext(v=office.15).aspx Einsichtnahme: 21.01.2016

[11] o.V. (2015): „2013 – Method ‚GetList‘ does not exist – SharePoint Stack Exchange“ http://sharepoint.stackexchange.com/questions/151354/method-getlist-does-not-exist Einsichtnahme: 01.12.2015

[12] Microsoft (o.J): „ListCollection.GetByTitle method“ im MSDN https://msdn.microsoft.com/DE-DE/library/office/microsoft.sharepoint.client.listcollection.getbytitle.aspx Einsichtnahme: 21.01.2016

[13] Fandrich, Andreas (2010): „CAML-Abfragen in SharePoint“ http://blogs.evocom.de/af/archive/2010/12/03/caml-anfragen-in-sharepoint.aspx Einsichtnahme: 01.12.2015

[14] Microsoft (o.J.): „Aufrufen von ‚Load‘ und ‚ExecuteQuery‘ vor dem Zugriff auf Werteigenschaft“ im MSDN

https://msdn.microsoft.com/de-de/library/office/ee535262(v=office.14).aspx
Einsichtnahme: 01.12.2015

[15] Mageed, Ahmad (2009): „How to remove elements from a generic list while iterating over it?“
http://stackoverflow.com/questions/1582285/how-to-remove-elements-from-a-generic-list-while-iterating-over-it
Einsichtnahme: 24.02.2016

[16] o.V. (2007): „[FAQ] Controls von Thread aktualisieren lassen (Control.Invoke/Dispatcher.Invoke)“
http://www.mycsharp.de/wbb2/thread.php?threadid=33113
Einsichtnahme: 25.02.2016

[17] Volz, Bernhard (2008): Einstieg in Visual C# 2008 (Galileo Computing) Bonn 2008

[18] Anderson, David (2014): „WPF: How to Template, Style, and Animate a WPF Button Control“
https://www.youtube.com/watch?v=W8GoWHSkT4g
Einsichtnahme: 25.02.2016

[19] o.V. (2016): „EPPlus-Create advanced Excel spreadsheets on the server“
http://epplus.codeplex.com/

Einsichtnahme: 25.02.2016

[20] Wikipedia (o.J.): „Lines of Code“
https://de.wikipedia.org/wiki/Lines_of_Code
Einsichtnahme: 02.03.2016

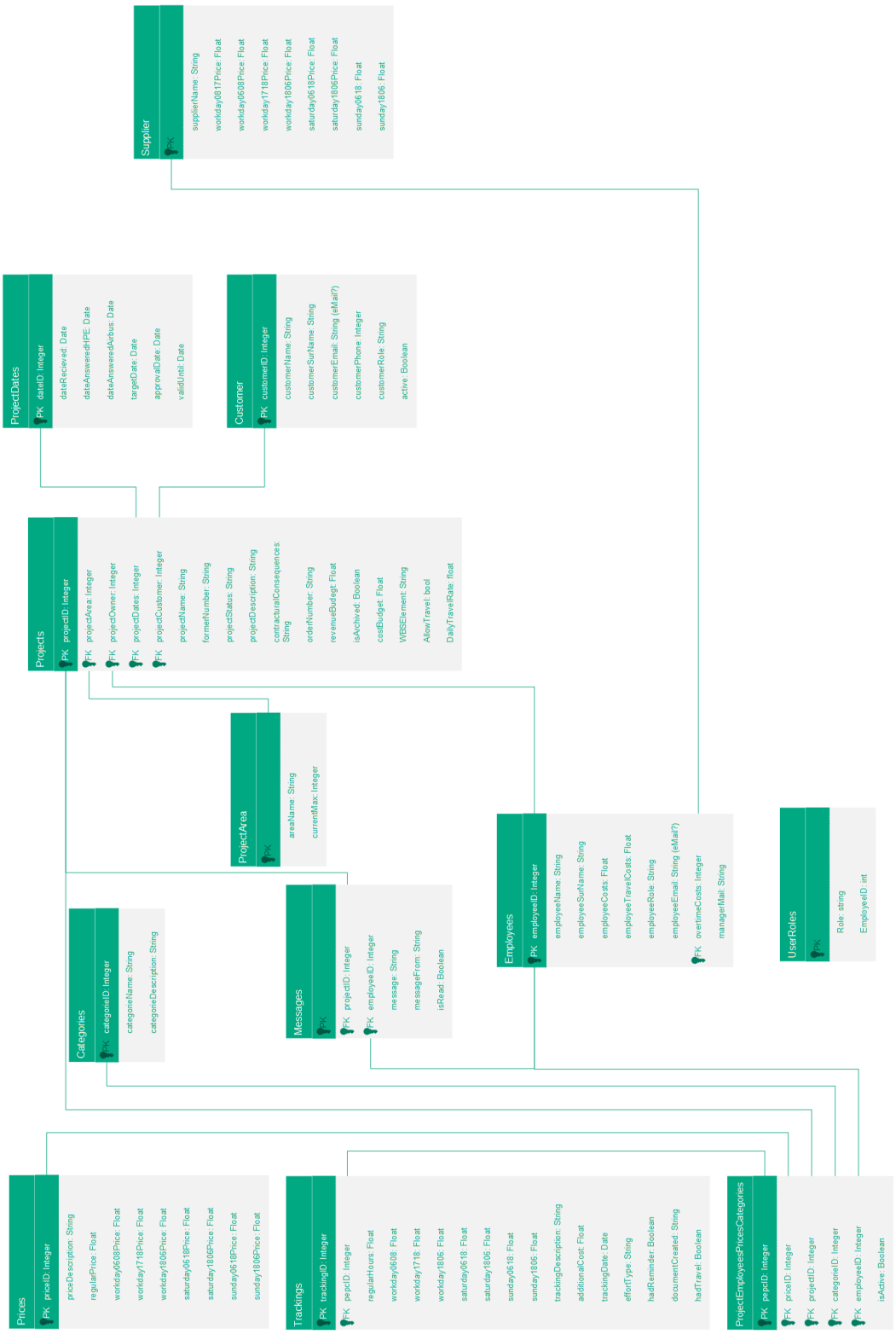
13 Glossar

Application Programming Interface (API)	Sammlung von Bibliotheken zur Schaffung einer Programmierschnittstelle
C#-Object Model (CSOM)	API von Microsoft zur Arbeit mit SharePoint in C#
CAML-Query	SQL-ähnliche Anfrage an einen SharePoint zur Abfrage von Listendaten
Change-Request	Ein im Rahmen von ITIL eingereichter Änderungswunsch am Bestand
ITIL	IT Infrastructure Library, Sammlung von Best Practises im Rahmen der Serviceerbringung
Leistungsnachweis	Excel-Dokument zur Dokumentation der geleisteten Stunden
Projekt-Owner	Der HPE-Mitarbeiter, der dem Kunden gegenüber für die Umsetzung des Change-Requests

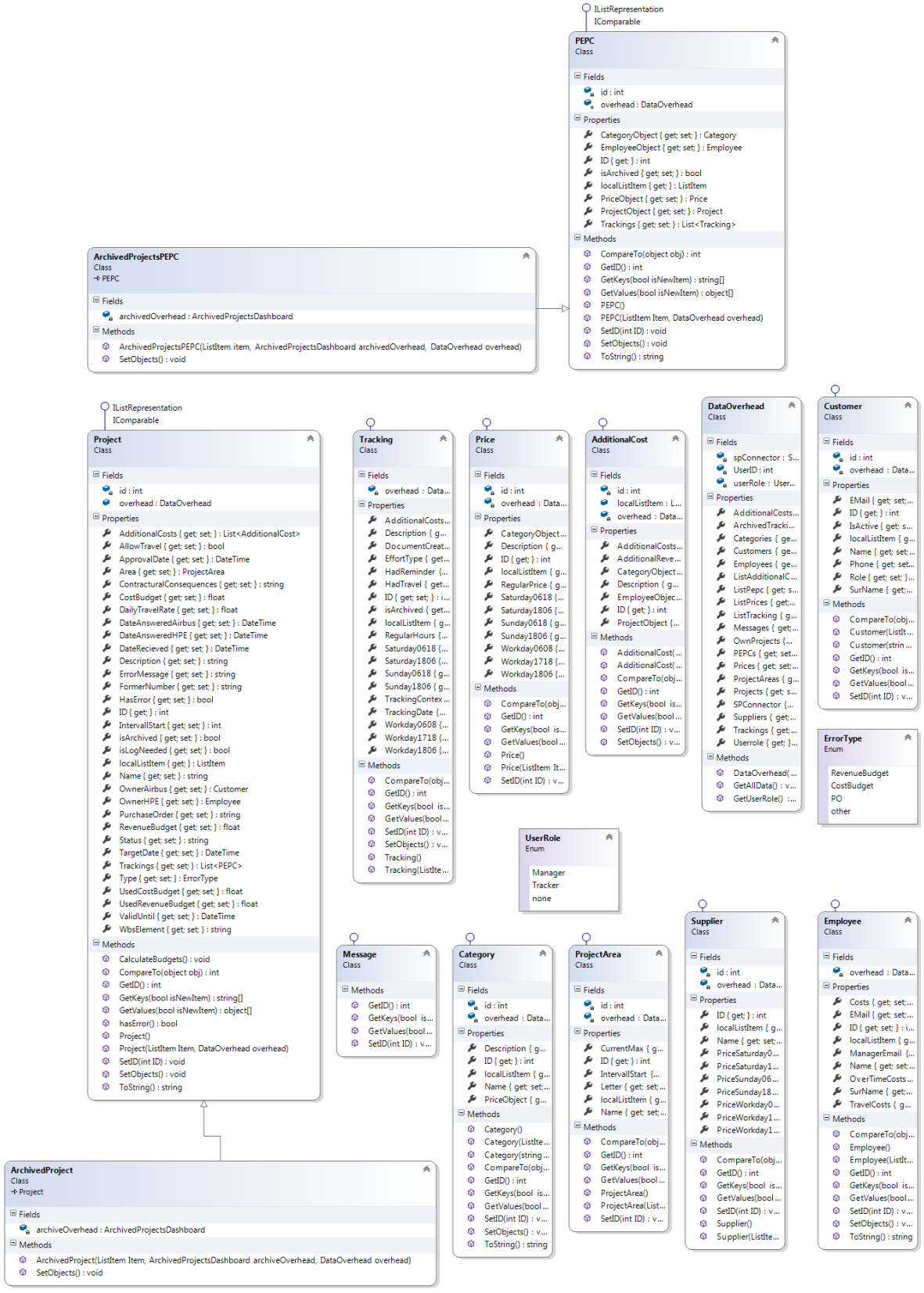
SharePoint	Tool von Microsoft für Firmen zur gemeinsamen Verwaltung von Daten, Dokumenten und Aufgaben
Tracker	Der Mitarbeiter, der Leistung gegenüber dem Kunden erbracht hat und dafür seine Arbeitszeit erfasst
Visual Basic for Applications (VBA)	Programmiersprache zur Erstellung von Makros in Office-Anwendungen
Windows Presentation Foundation (WPF)	Framework von Microsoft zur Gestaltung von Benutzeroberflächen
XAML	Beschreibungssprache im Rahmen von WPF

14 Anhang

Modell der Datenbank am Ende der Planungsphase

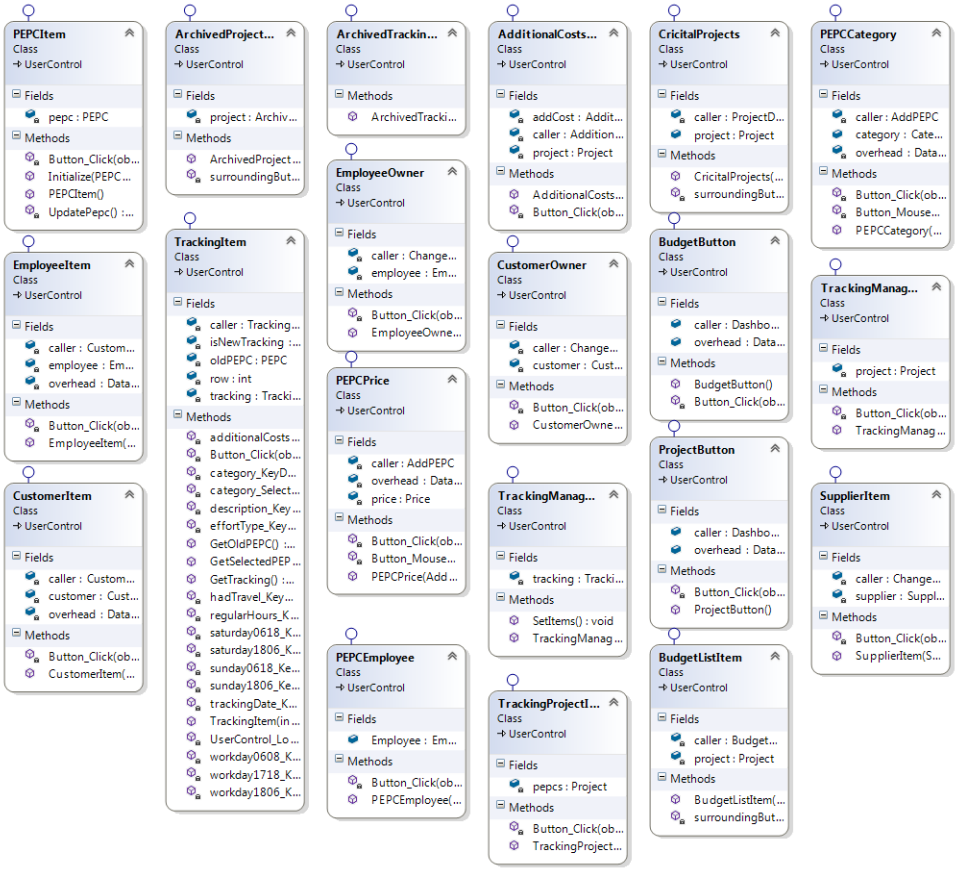

Supplier
supplierName: String
workday0817Price: Float
workday0608Price: Float
workday1718Price: Float
workday1806Price: Float
saturday0618Price: Float
saturday1806Price: Float
sunday0618: Float
sunday1806: Float
ProjectDates
dateID: Integer
dateRecieved: Date
dataAnsweredHPE: Date
dataAnsweredAirbus: Date
targetDate: Date
approvalDate: Date
validUntil: Date
Customer
customerID: Integer
customerName: String
customerSurName: String
customerEmail: String (eMail?)
customerPhone: Integer
customerRole: String
active: Boolean
Projects
projectID: Integer
projectArea: Integer
projectOwner: Integer
projectDates: Integer
projectCustomer: Integer
projectName: String
formerNumber: String
projectStatus: String
projectDescription: String
contractualConsequences: String
orderNumber: String
revenueBudegt: Float
isArchived: Boolean
costBudget: Float
WBSElement: String
AllowTravel: bool
DailyTravelRate: float
ProjectArea
areaName: String
currentMax: Integer
Employees
employeeID: Integer
employeeName: String
employeeSurName: String
employeeCosts: Float
employeeTravelCosts: Float
employeeRole: String
employeeEmail: String (eMail?)
overtimeCosts: Integer
managerMail: String
UserRoles
Role: string
EmployeeID: int
Categories
categoneID: Integer
categoneName: String
categoneDescription: String
Messages
projectID: Integer
employeeID: Integer
message: String
messageFrom: String
isRead: Boolean
Prices
priceID: Integer
priceDescription: String
regularPrice: Float
workday0608Price: Float
workday1718Price: Float
workday1806Price: Float
saturday0618Price: Float
saturday1806Price: Float
sunday0618Price: Float
sunday1806Price: Float
Trackings
trackingID: Integer
pepcID: Integer
regularHours: Float
workday0608: Float
workday1718: Float
workday1806: Float
saturday0618: Float
saturday1806: Float
sunday0618: Float
sunday1806: Float
trackingDescription: String
additionalCost: Float
trackingDate: Date
effortType: String
hadReminder: Boolean
documentCreated: String
hadTravel: Boolean
ProjectEmployeesPricesCategories
pepcID: Integer
priceID: Integer
projectID: Integer
categoneID: Integer
employeeID: Integer
isActive: Boolean

Modell des Moduls DataOrganization



Modell des Moduls DataVisualization: Windows

Modell des Moduls DataVisualization: Custom Controls



Modell der Module SharePointCommunication & ExcelCommunication