|  |
| --- |
| Elmer Lukas, Heidt Christina, Treichler Delia  19. Dezember 2011 |

|  |
| --- |
| Studienarbeit |
| Entwurf |
|  |

****

# Dokumentinformationen

## Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Version | Änderung | Autor |
| 25.09.2011 | 1.0 | Erste Version des Dokuments | lelmer |
| 03.10.2011 | 1.1 | Review | dtreichl |
| 12.12.2011 | 1.2 | Schichtenarchitektur dokumentiert | lelmer |
| 15.12.2011 | 1.3 | Architektur angepasst, Sensitivitätsanalyse | lelmer |
| 16.12.2011 | 1.4 | Review | cheidt |
| 19.12.2011 | 1.5 | Review | dtreichl |

## Inhaltsverzeichnis

[1 Dokumentinformationen 1](#_Toc312076316)

[1.1 Änderungsgeschichte 1](#_Toc312076317)

[1.2 Inhaltsverzeichnis 1](#_Toc312076318)

[1.3 Abbildungsverzeichnis 2](#_Toc312076319)

[1.4 Tabellenverzeichnis 2](#_Toc312076320)

[2 Design Entscheide 3](#_Toc312076321)

[2.1 PDF auf Surface 2 Darstellen 3](#_Toc312076322)

[2.1.1 Varianten 3](#_Toc312076323)

[2.1.2 Nutzwertanalyse 3](#_Toc312076324)

[2.1.3 Sensitivitätsanalyse 4](#_Toc312076325)

[3 Architektur 5](#_Toc312076326)

[3.1 Systemübersicht 5](#_Toc312076327)

[3.1.1 Sharepoint Server 5](#_Toc312076328)

[3.1.2 Webserver der Zühlke Engineering AG 6](#_Toc312076329)

[3.1.3 Microsoft Surface 2 6](#_Toc312076330)

[3.2 Assemblies und Namespaces 6](#_Toc312076331)

[3.2.1 Interfaces 9](#_Toc312076332)

[3.2.2 Implementation 11](#_Toc312076333)

[3.3 UI Design 16](#_Toc312076334)

[3.3.1 Allgemein 16](#_Toc312076335)

[3.3.2 Unterteilung der Views 16](#_Toc312076336)

[3.4 Prozesse und Threads 19](#_Toc312076337)

[3.4.1 Konversation PDF zu XPS 20](#_Toc312076338)

[3.4.2 Parallel Preparer 20](#_Toc312076339)

[3.4.3 Asynchrones und verzögertes Laden des XPS Dokuments 20](#_Toc312076340)

[4 Patterns 21](#_Toc312076341)

[4.1 MVVM 21](#_Toc312076342)

[4.2 Flyweight 21](#_Toc312076343)

[4.3 Dependency Injection mit Microsoft Unity 21](#_Toc312076344)

## Abbildungsverzeichnis

[Abbildung 1 - Systemübersicht 5](#_Toc312074293)

[Abbildung 2 - Generierung Dependency Graph 6](#_Toc312074294)

[Abbildung 3 - Übersicht Abhängigkeiten 7](#_Toc312074295)

[Abbildung 5 - Übersicht Abhängigkeiten mit Details 8](#_Toc312074296)

[Abbildung 4 - Details Abhängigkeiten 8](file:///D:\flip_svn.elmermx.ch\doc\03_Technischer_Bericht_Teil_2\04_Entwurf\Entwurf.docx#_Toc312074297)

[Abbildung 6 - NDepend Dependency Graph 9](#_Toc312074298)

[Abbildung 8 - Übersicht Interfaces 9](#_Toc312074299)

[Abbildung 9 - Converter.Interfaces 10](#_Toc312074300)

[Abbildung 10 - Services.Interfaces 10](#_Toc312074301)

[Abbildung 11 - Loader.Interfaces 11](#_Toc312074302)

[Abbildung 12 - Implementation Übersicht 11](#_Toc312074303)

[Abbildung 13 - Services.Loader 11](#_Toc312074304)

[Abbildung 14 - Converter.Pdf 12](#_Toc312074305)

[Abbildung 15 - Preparer 12](#_Toc312074306)

[Abbildung 16 - Sequenzdiagramm Preparer 13](#_Toc312074307)

[Abbildung 17 - Services 14](#_Toc312074308)

[Abbildung 18 - UserInterface.Surface 15](#_Toc312074309)

[Abbildung 19 - ProjectFlip 16](#_Toc312074310)

[Abbildung 20 - Übersicht 17](#_Toc312074311)

[Abbildung 21 - Filter View 17](#_Toc312074312)

[Abbildung 22 - Gefilterte Übersicht 18](#_Toc312074313)

[Abbildung 23 - Info View 18](#_Toc312074314)

[Abbildung 24 - Detail View 19](#_Toc312074315)

[Abbildung 25 - Detail View mit Zoom 19](#_Toc312074316)

## Tabellenverzeichnis

[Tabelle 1 - Nutzwertanalyse 4](#_Toc312074317)

# Design Entscheide

## PDF auf Surface 2 Darstellen

Die primäre Aufgabe der Software ist es, PDF Dokumente auf dem Surface 2 darzustellen. Dazu gibt es verschiedene Möglichkeiten, die genauer analysiert wurden. Zur Entscheidung wurde schliesslich eine Nutzwertanalyse durchgeführt.

### Varianten

#### Variante 1: PDF direkt darstellen

Eine naheliegende Lösung ist natürlich, das PDF direkt auf dem Surface darzustellen. Leider unterstützt das Surface 2 Framework diese Darstellung nicht direkt; es müsste noch eine PDF Library zur Darstellung zur Laufzeit geladen werden. Dadurch wird die Laufzeitumgebung von einer zusätzlichen Library abhängig, was die Installation erschwert oder gar verunmöglicht. Dies kann zurzeit nicht genauer beurteilt werden, da die zu testende Hardware noch nicht verfügbar ist.

#### Variante 2: Umwandlung zu XPS

Das Surface 2 SDK erlaubt es, ohne weitere Libraries XPS Dokumente anzuzeigen. Dies geschieht mit Hilfe der DocumentViewer Klasse. Das XPS Dokument ähnelt im Aufbau einem PDF Dokument, so können Schriften und Vektorgrafiken beliebig vergrössert werden, ohne dass dabei ein Qualitätsverlust entsteht.

#### Variante 3: Umwandlung zu Bild

Die einfachste Möglichkeit, um ein PDF auf dem Surface 2 darzustellen, ist eine Umwandlung des PDF Dokuments in eine Rastergrafik. Diese können ohne Probleme auf dem Surface 2 dargestellt werden und auch die Umwandlung eines PDFs zu einem Bild ist mit Open Source Libraries problemlos möglich (z.B. mit Image Magick oder GhostScript/GhostPDF/GhostXPS). Der grosse Nachteil besteht darin, dass die Bilder sehr viel Speicherplatz verbrauchen würden. Eine Rastergrafik verfügt zudem über eine Maximalgrösse, wird diese bei einer Vergrösserung überschritten, kann das Bild nicht mehr verlustfrei angezeigt werden.

### Nutzwertanalyse

In der Nutzwertanalyse (siehe Tabelle 1 - Nutzwertanalyse) geht die **Variante 2: Umwandlung zu XPS** als Sieger vor der Variante 3: Umwandlung zu Bild hervor.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nutzwertanalyse: PDF auf Surface 2 darstellen | | | | | | | | | |
|  |  | **Variante 1** | | | **Variante 2** | | | **Variante 3** | |
|  |  | **PDF direkt darstellen** | | **Umwandlung zu XPS** | | | | **Umwandlung zu Bild** | |
| Kriterium | **Gewichtung** | **Bewertung** | **Total** | **Bewertung** | | **Total** | | **Bewertung** | **Total** |
|  | **1 bis 5** | **1 bis 5** |  | **1 bis 5** | |  | | **1 bis 5** |  |
| Programmieraufwand | 2 | 1 | 2 | 3 | | 6 | | 5 | 10 |
| Kopplung Laufzeitumgebung | 5 | 1 | 5 | 5 | | 25 | | 5 | 25 |
| Testbarkeit | 4 | 1 | 4 | 5 | | 20 | | 5 | 20 |
| Darstellungsqualität | 4 | 5 | 20 | 4 | | 16 | | 1 | 4 |
| Total Punkte |  |  | **31** |  | | **67** | |  | **59** |
| Rang |  |  | **3** |  | | **1** | |  | **2** |
|  |  |  |  |  | | |  |  |  |
| Bemerkung: Höhere Gewichtungen / Bewertungen sind besser. | | | | | | | | | |

Tabelle 1 - Nutzwertanalyse

### Sensitivitätsanalyse

In der Sensitivitätsanalyse wird untersucht, wie stark sich eine Änderung auf das Gesamtergebnis auswirken würde.

Die Variante 1 wird auch bei Änderungen der Bewertung nicht als Sieger hervorgehen.   
Zwischen der Variante 2 und 3 ist die Darstellungsqualität das entscheidende Kriterium. Eine andere Bewertung dieses Kriteriums würde das Ergebnis so stark beeinflussen, dass die Variante 3 zum Gewinner würde. Dieses Kriterium ist jedoch für die Zühlke Engineering AG sehr wichtig. Zudem wurde die Variante 3 bereits im Vorgängerprojekt Project Flip 1.0 verfolgt, leider mit einem nicht vollständig zufriedenstellenden Ergebnis. Somit ist die Wahl der **Variante 2: Umwandlung zu XPS** klar begründet.

# Architektur

Bei der Architektur wurde darauf geachtet, dass die einzelnen Komponenten möglichst unabhängig sind und so einfach ausgetauscht werden können.

## Systemübersicht

Das System ist in mehrere Komponenten unterteilt. Die drei Hauptkomponenten sind:

* Sharepoint Server
* Webserver der Zühlke Engineering AG
* Microsoft Surface 2

Hier ein Überblick über die Komponenten und deren Rollen:

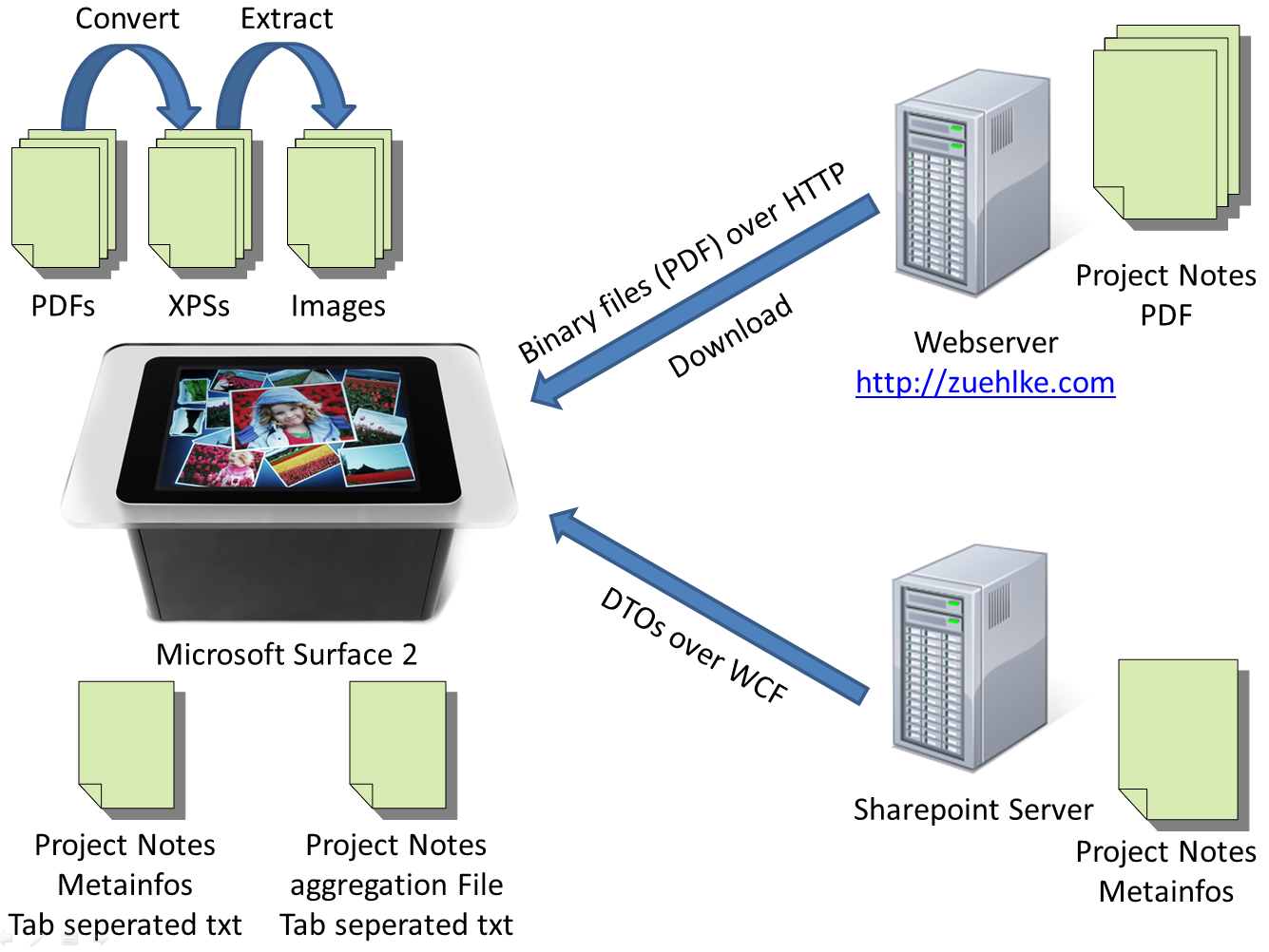


Abbildung 1 - Systemübersicht

### Sharepoint Server

Auf dem internen Sharepoint Server von Zühlke Engineering AG werden die Project Notes und deren Metainformationen verwaltet. Nachdem eine neue Project Note publiziert wurde, werden die Metadaten erfasst und auf dem Sharepoint Server abgelegt. So können verschiedene Personen und Applikationen diese Daten abrufen und verarbeiten.

In der aktuellen Implementation besteht keine Anbindung an den Sharepoint Server, da die Hardware zum Zeitpunkt des Projektes noch nicht zur Verfügung stand und von Extern kein Zugriff bestand. Des Weiteren hätte dies auch den Zeitrahmen des Projektes gesprengt.

### Webserver der Zühlke Engineering AG

Der Webserver der Zühlke Engineering AG hostet die Internetseite von Zühlke. Zudem stehen dort auch diverse Project Notes zum Download zur Verfügung.

Zu Beginn der Arbeit wurde von der Zühlke Engineering AG ein statisches Textdokument bereitgestellt, in welchem die Metainformationen aller Project Notes vorhanden sind. Mit Hilfe dieser Informationen können die PDF Dokumente direkt vom Webserver heruntergeladen und gespeichert werden. So stehen sie für die Applikation zur Verfügung.

### Microsoft Surface 2

Der Microsoft Surface 2 wird in der Eingangshalle der Zühlke Engineering AG stehen, damit wartende Kunden sich mit den Project Notes auseinander setzen können. Um die Verfügbarkeit des Services sicherzustellen, muss der Surface möglichst unabhängig von den anderen Komponenten sein. Deshalb werden die für die Applikation benötigten Informationen periodisch heruntergeladen und verarbeitet. Dieser Vorgang ist von der Programm-Ausführung getrennt und wird im Betriebskonzept (TODO: ref) genauer beschrieben.

Bevor die Applikation gestartet werden kann, müssen die gespeicherten PDF Dokumente weiter verarbeitet werden. Als erstes werden die PDF Dokumente in XPS Dokumente konvertiert. In einem zweiten Schritt werden Bilder aus der XPS Datei extrahiert.

Eine wichtige Anforderung an das Projekt war die Benutzerfreundlichkeit der Applikation. Deshalb wurde sichergestellt, dass nicht zu viele Informationen auf einmal dargestellt werden. Die Metadaten der Project Notes enthalten Bezeichnungen, welche sehr vielfältig sind (z.B. Java, JBoss, JavaBeans, JPA). Aus diesem Grund besteht die Möglichkeit, verschiedene Namen auf einen gemeinsamen Begriff (z.B. Java) zu mappen. Um die Aggregation einfach anpassen und erweitern zu können, steht eine weitere, Tab-getrennte Textdatei zur Verfügung, welche mit Microsoft Excel bearbeitet werden kann.

## Assemblies und Namespaces

Das Projekt enthält, im Vergleich zu anderen, wenige Zeilen Code. Der Code könnte daher schon als overengineered bezeichnet werden. Dies wurde aber explizit von der Zühlke Engineering AG so gewünscht, da das Projekt noch weiterentwickelt werden soll.

Zum Beweis der Schichtenarchitektur wurde im Visual Studio 2010 ein Dependency Diagramm erstellt. Wie dabei vorgegangen wurde, ist in Abbildung 2 - Generierung Dependency Graph ersichtlich.



Abbildung 2 - Generierung Dependency Graph

Um das Diagramm übersichtlich zu halten, wurden in einem zweiten Schritt die „Generics“ und die „Externals“ entfernt, damit nur die eigens erstellten Klassen sichtbar bleiben. Anschliessend wurden die Tests, Implementation und Interfaces gruppiert und eingefärbt, damit das Diagramm überschaubarer wird (siehe Abbildung 3 - Übersicht Abhängigkeiten).



Abbildung 3 - Übersicht Abhängigkeiten

Nachfolgend sind die Klassen pro Gruppierung und weitere Details zu den Abhängigkeiten ersichtlich.

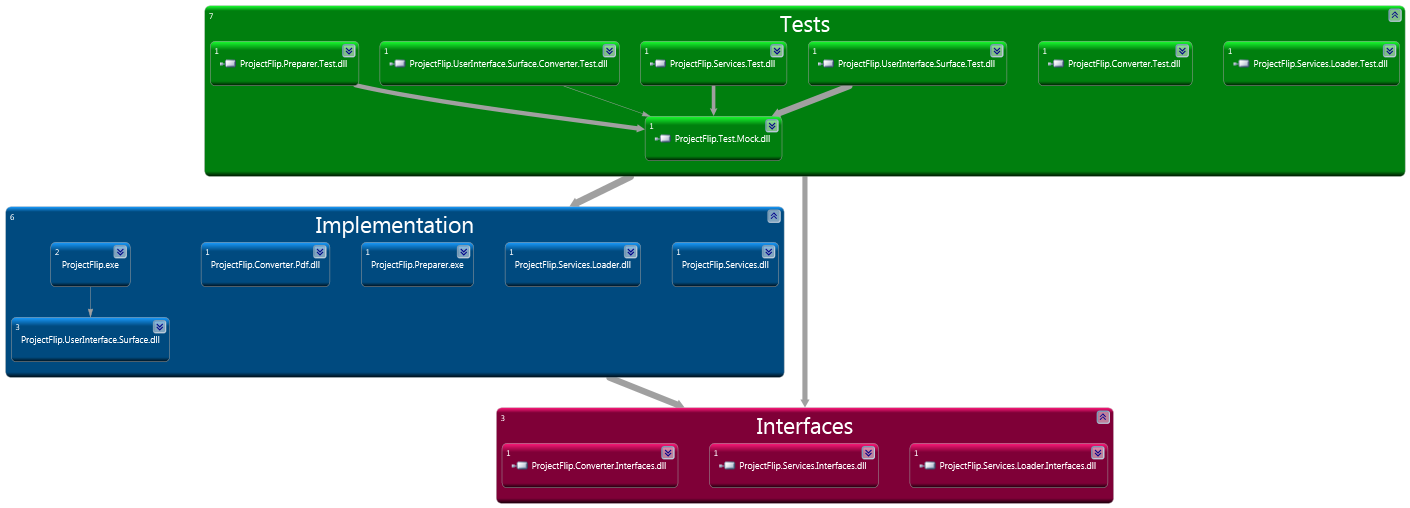
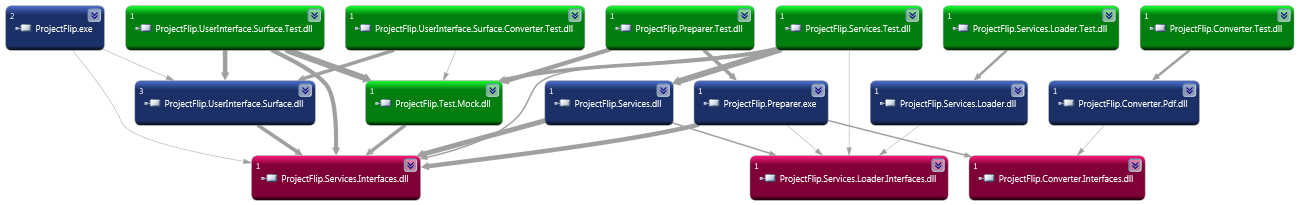


Abbildung 5 - Übersicht Abhängigkeiten mit Details

Abbildung 4 - Details Abhängigkeiten

Um die Schichtung mit einem alternativen Tool zu beweisen, wurde mit NDepend folgendes Diagramm generiert.

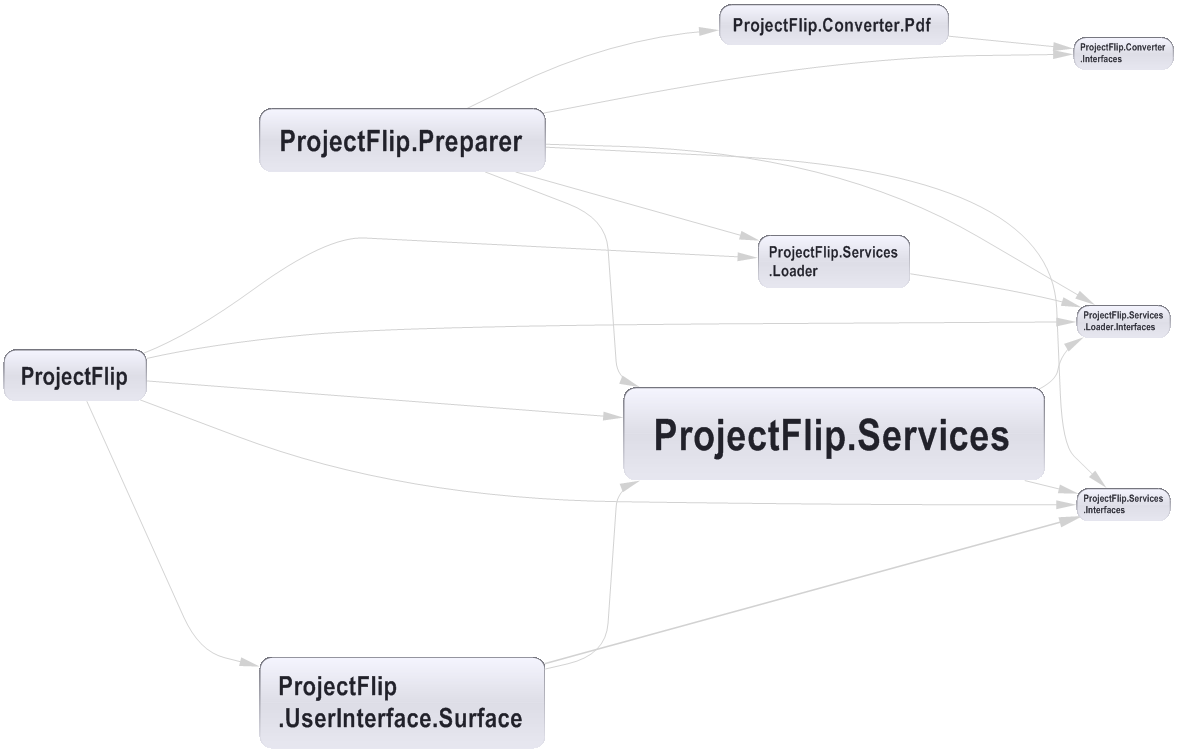


Abbildung 6 - NDepend Dependency Graph

Ganz links befindet sich das Startup Projekt, weiter rechts folgen das User Interface und der Preparer, dann die Services, Loader und Converter und ganz rechts schliesslich noch die Interfaces.

In den folgenden Unterkapiteln werden die einzelnen Projekte der Visual Studio Solution genauer beschrieben. Die Test Projekte jedoch werden nicht weiter dokumentiert, da ihre Namen ihren Zweck schon genügend vermitteln.

### Interfaces

Um die Implementation untereinander zu entkoppeln, wurden Interfaces eingesetzt. Die Abbildung 8 - Übersicht Interfaces gibt einen Überblick.

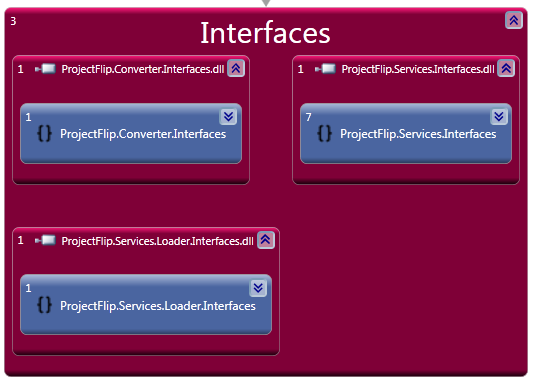


Abbildung 8 - Übersicht Interfaces

Die Interfaces werden nachfolgend genauer beschrieben.

#### Converter.Interfaces

Das Interface IConverter wird für die Konvertierung vom PDF Dokument in ein XPS Dokument eingesetzt. Das ermöglicht ein einfaches Austauschen der Implementation. Dies wird in Zukunft, sobald mehr solche Konverter gratis zur Verfügung stehen und sich das XPS Dateiformat mehr durchgesetzt hat, auch gut möglich sein.

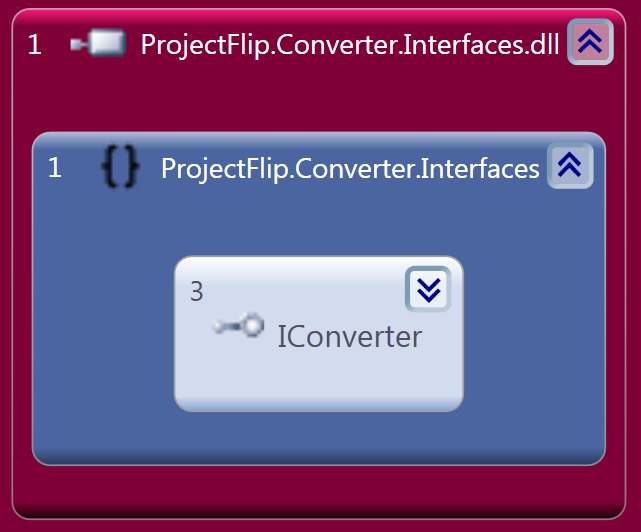


Abbildung 9 - Converter.Interfaces

#### Services.Interfaces

Die Interfaces IAggregator, IMetadata und IMetadataType repräsentieren die Metadaten der Project Notes. So wird als Beispiel der Begriff „C++“ als IMetadata und die Kategorie „Technologie“ als IMedatataType gespeichert. Der IAggregator ist für das Mapping verantwortlich, mit Hilfe dessen beispielsweise die Bezeichnungen „JavaBeans“ und „Java EE“ zum Begriff „Java“ zusammengefasst werden können.

Der ICultureHelper ist dafür verantwortlich, dass eine Sprache registriert werden kann. Dies wird zur Anzeige eines XPS benötigt.

Die ICyclicCollectionView<T> ist eine zyklische Liste, auf der gefiltert werden kann. Sie implementiert das INotifyPropertyChanged, damit sie die involvierten Komponenten benachrichtigen kann.

Die IProjectNote stellt eine ProjectNote dar. Neben den Properties (z.B. Title) stellt sie auch eine Preload-Methode zur Verfügung, um das XPS Dokument mittels Eager Loading vorzuladen.

Der IProjectNoteService stellt den Service für die Project Notes zur Verfügung. Es bietet zwei Property Getters – einer für die IProjectNotes und einer für die IMetadatas.

Ein Teammitglied wird durch die IPerson repräsentiert. Der IGravatarService stellt den Service für das Team dar, also die einzelnen IPersons.

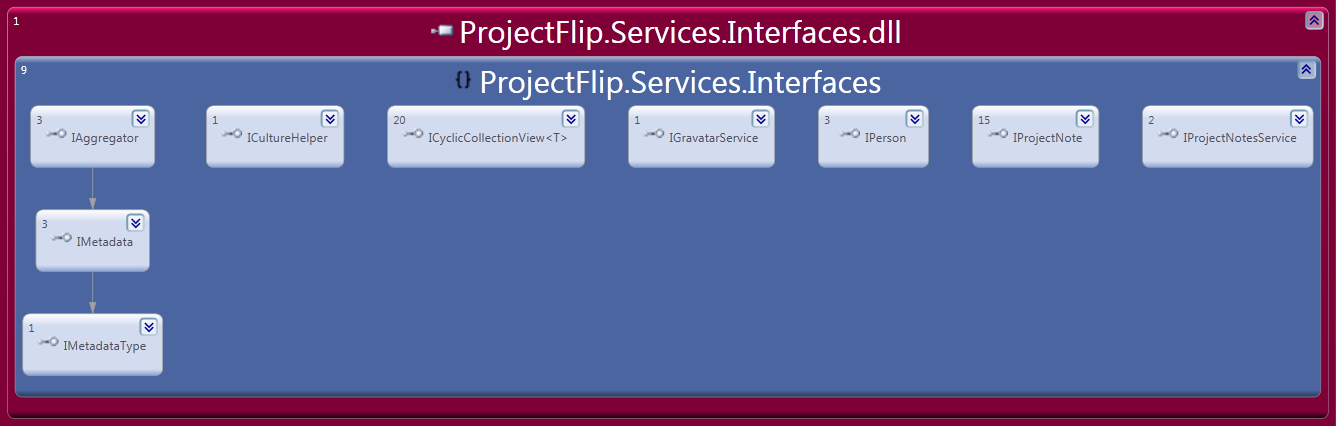


Abbildung 10 - Services.Interfaces

#### Loader.Interfaces

Der IProjectNotesLoader stellt die Möglichkeit, IProjectNotes zu laden, zur Verfügung. Das Interface ist deshalb in einem separaten Assembly, damit der Preparer und die eigentliche Applikation dieses gemeinsame Interface implementieren können.

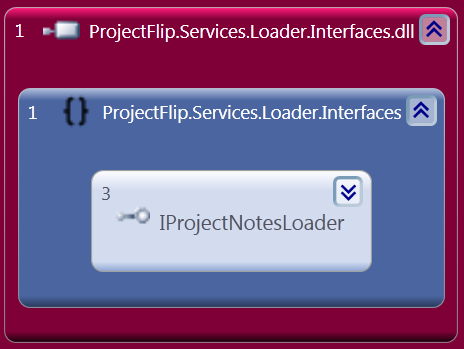


Abbildung 11 - Loader.Interfaces

### Implementation

Die Implementation ist von den Interfaces abhängig. Dadurch können die Implementationen voneinander entkoppelt werden. Dies ist in Abbildung 12 - Implementation Übersicht ersichtlich.

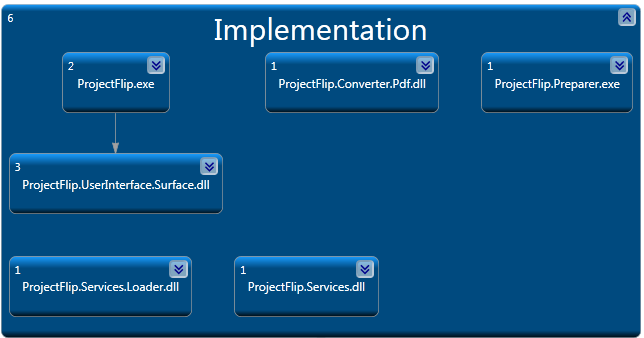


Abbildung 12 - Implementation Übersicht

#### Services.Loader

Der ProjectNotesLoader kann eine Liste von IProjectNotes laden. In der aktuellen Version werden die Daten von einer Tab-getrennten Textdatei gelesen, in Zukunft können diese aber auch von einer Datenbank oder vom Sharepoint geladen werden.

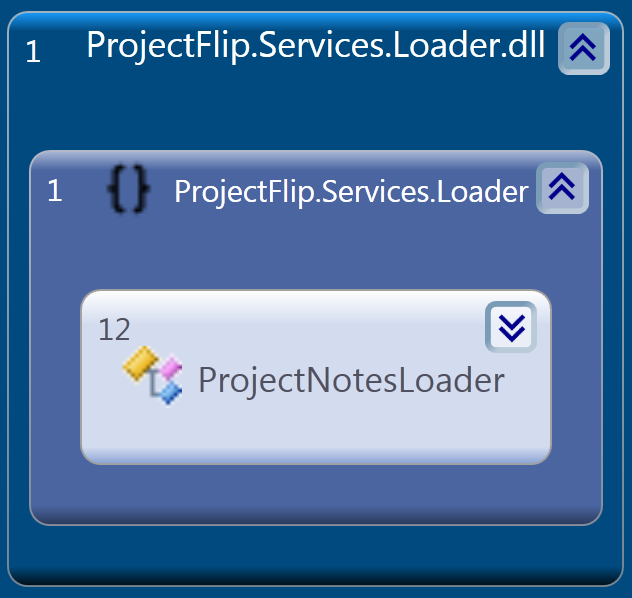


Abbildung 13 - Services.Loader

#### Converter.Pdf

Der PdfConverter ist dafür verantwortlich, das PDF Dokument in ein XPS Dokument zu konvertieren. Da die Konvertierung mithilfe des Adobe Readers geschieht, kann die AcrobatLocation im Settings File konfiguriert werden. Standardmässig ist der Pfad auf „C:\Program Files (x86)\Adobe\Reader 10.0\Reader\AcroRd32.exe“ gesetzt.

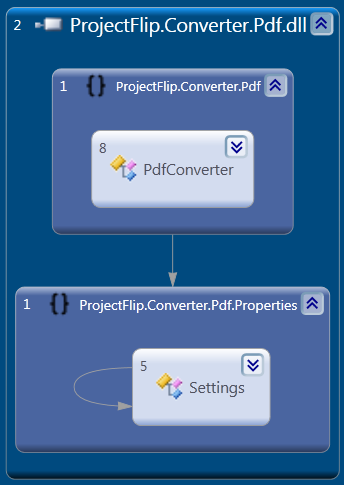


Abbildung 14 - Converter.Pdf

#### Preparer

Der Preparer ist dafür verantwortlich, dass die Project Notes der Applikation im entsprechenden Format zur Verfügung steht. Konkret heisst das: pro Project Note ein XPS Dokument und ein Bild (243x243 Pixel gross). Dazu verwendet der Preparer einen IProjectNotesLoader und einen IConverter.

Da der Preparer vor der Applikation und unabhängig ausgeführt wird, ist er als Executable (exe) kompiliert.



Abbildung 15 - Preparer

Beim Preparer sind die einzelnen IProcessors wichtig. Als erstes wird der DownloadProcessor verwendet, der das PDF Dokument herunterlädt und lokal speichert. In einem zweiten Schritt wird der ConverterProcessor verwendet, der das PDF Dokument in ein XPS Dokument konvertiert. Der ImageExtractorProcessor extrahiert schliesslich noch ein Bild aus dem XPS Dokument. Tritt dabei ein Fehler auf und befinden sich die Dateien dadurch in einem fehlerhaften Zustand, so werden diese vom CleanupProcessor entfernt.

Um die Ausführung zu beschleunigen, wurde die Verarbeitung der einzelnen Project Notes parallelisiert. Das folgende Sequenzdiagramm zeigt den Ablauf auf (in der Liste „processors“ sind Instanzen der einzelnen IProcessors enthalten):



Abbildung 16 - Sequenzdiagramm Preparer

#### Services

Der Service stellt die IProjectNotes und die aggregierten IMetadata Objekte zur Verfügung. Er erhält zuerst die Rohdaten vom IProjectNotesLoader und baut dann aufgrund der Eingaben die Objektstruktur mit den IProjectNotes und IMetadatas auf. Die Metadaten werden beim Import sogleich aggregiert. Um auch die Personen, die am Projekt mitgearbeitet haben, darzustellen, stellt der GravatarService eine Liste von Persons zur Verfügung. Weiter implementiert das Service Projekt bestimmte Hilfsklassen, wie zum Beispiel den MetadataComparer.

Damit nach Project Notes gefiltert werden kann, stellt der Service eine CyclicCollectionView<T> zur Verfügung. Diese implementiert das Interface ICyclicCollectionView<T>.

Der Service kann auch konfiguriert werden, so gibt es eine Einstellung ProjectNotesUrl, die zurzeit auf die HTTP URL von der Zühlke Engineering AG konfiguriert ist: „http://www.zuehlke.com/uploads/tx\_zepublications/“. Von diesem Verzeichnis aus werden die Project Notes heruntergeladen.

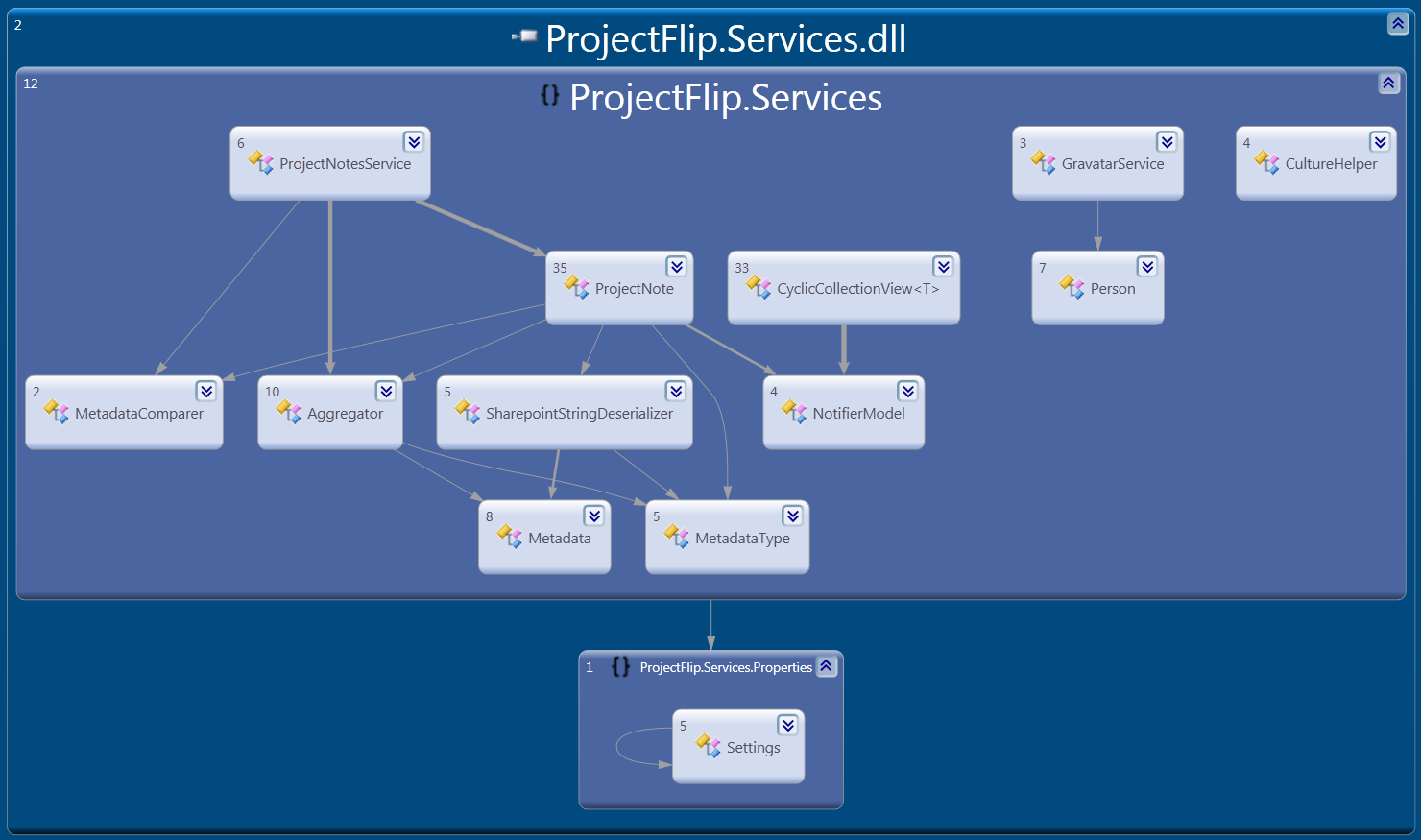


Abbildung 17 - Services

#### UserInterface.Surface

Das UserInterface.Surface Assembly ist für das GUI verantwortlich. Es benutzt zur Entkopplung der View und des Models das MVVM Pattern (siehe Unterkapitel 4.1 MVVM).

Das OverviewWindowViewModel ist dafür verantwortlich, die Commands der View entgegenzunehmen, eine Aktion auszuführen (Bsp.: einen Filter setzen) und dann die View zu benachrichtigen (mithilfe von INotifyPropertyChanged). Das GravatarsViewModel verwaltet die Personen und stellt die Schnittstelle zwischen dem IGravatarService und dem Gravatars User Control dar.

Die Klasse Command implementiert das Interface ICommand, welches vom OverviewWindowViewModel benutzt wird.

Das ScrollToTopBehaviour wird benötigt, damit bei einer Änderung der Elemente im Container automatisch nach oben gescrollt wird.

Das OverviewWindow stellt das Hauptfenster dar. Die Klassen AboutView, DetailView, FilterDetailView, FilterView, Gravatars, ZoomInIcon und ZoomOutIcon stellen einzelne UserControls dar, die vom OverviewWindow benutzt werden.

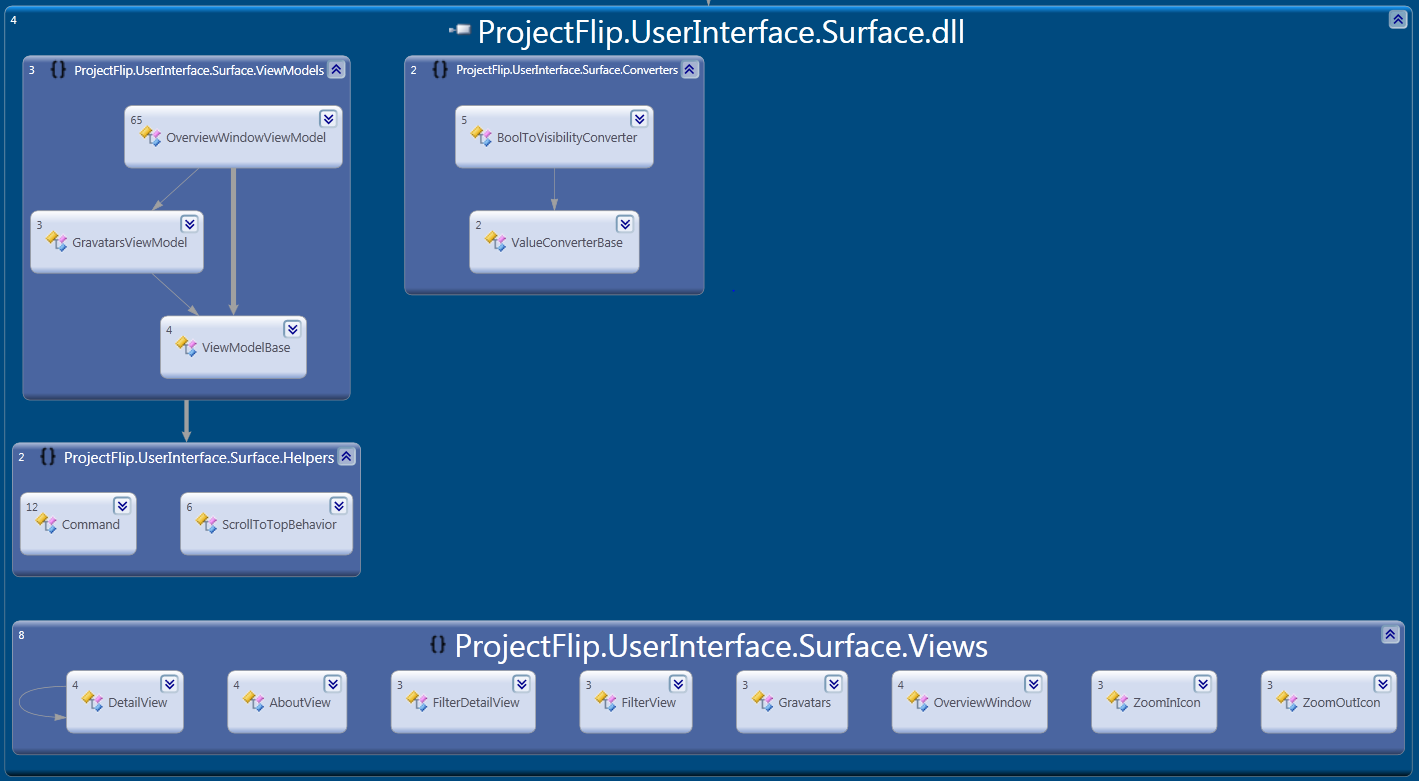


Abbildung 18 - UserInterface.Surface

#### ProjectFlip

Bei diesem Projekt handelt es sich das Startup Projekt. Es ist verantwortlich für die Zusammenfügung der richtigen Komponenten und schliesslich für das Starten des GUIs. Hier wurde Dependency Injection eingesetzt, und zwar mittels Microsoft Unity (siehe Unterkapitel 4.3 Dependency Injection mit Unity). So kann an einem zentralen Ort konfiguriert werden, welche Komponenten instanziiert werden sollen.

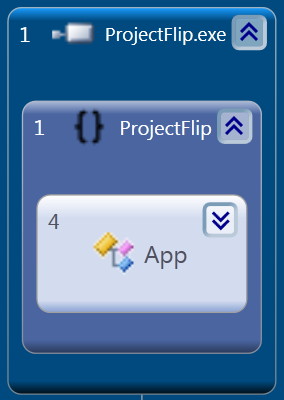


Abbildung 19 - ProjectFlip

## UI Design

### Allgemein

Für diese Applikation wurde WPF verwendet. Um die View vom Model zu trennen, wurde MVVM (siehe Unterkapitel 4.1 MVVM) eingesetzt. Durch die Verwendung dieses Patterns konnte fast vollständig auf Code Behind verzichtet werden, wodurch es möglich war, die Logik der View im ViewModel über Unit Tests zu prüfen.

### Unterteilung der Views

In vielen Applikationen kann zwischen verschiedenen Ansichten gewechselt werden. Bei Project Flip 2.0 ist dies jedoch so geregelt, dass der Startscreen immer im Hintergrund ersichtlich bleiben muss. Die anderen Komponenten (Detailansicht, Filter View, Info View) werden überlagert und mit einem halbtransparenten, abgedunkelten Hintergrund dargestellt.

#### Overview

Die Übersicht ist in zwei Teile eingeteilt: den Filter und die Listenansicht.

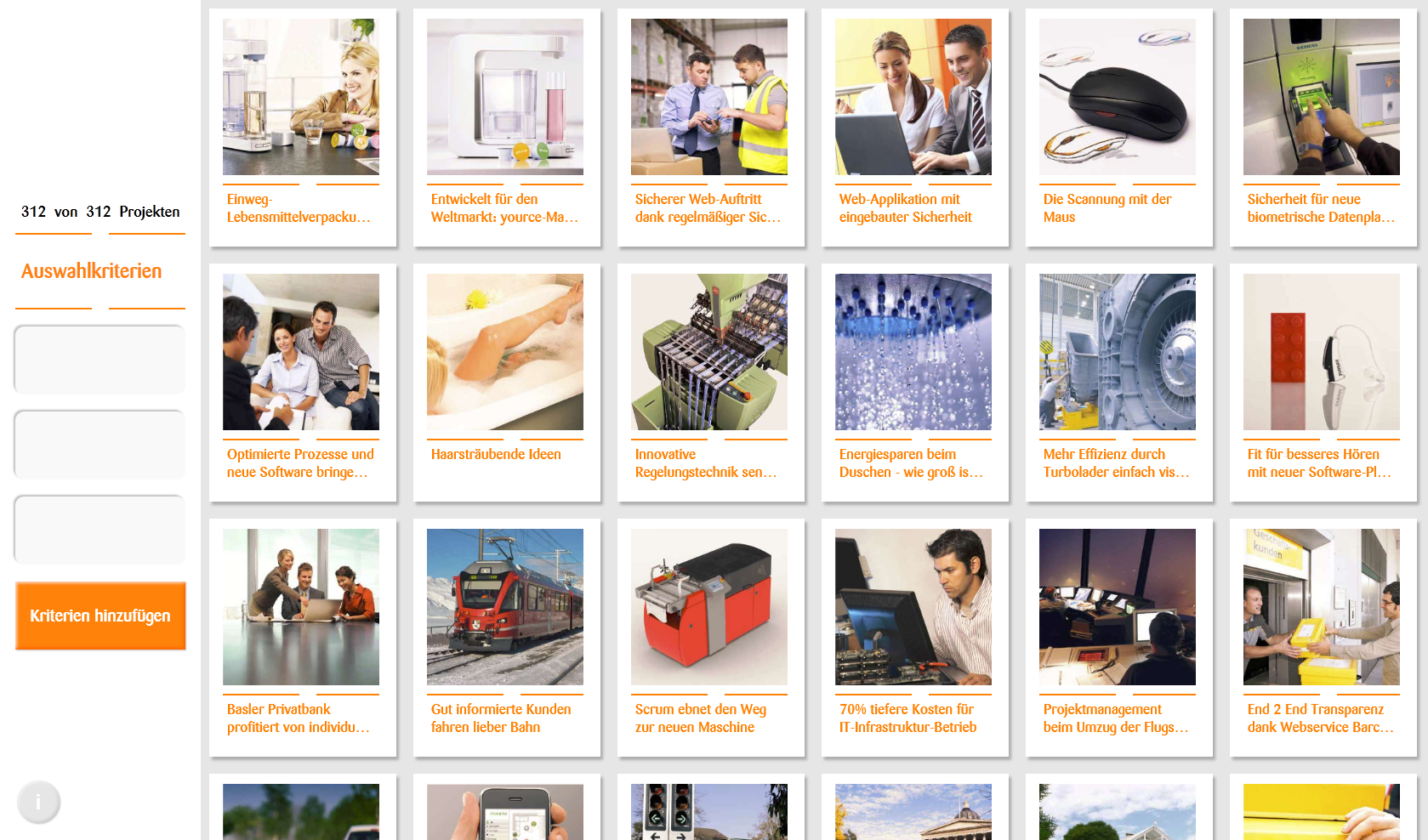


Abbildung 20 - Übersicht

Auf der linken Seite befindet sich der Filter. Auf der rechten Seite die Project Notes aufgelistet werden.

#### Filter View

Wird in der Übersicht auf den Knopf „Kriterien hinzufügen“ gedrückt, so öffnet sich die Filter View.

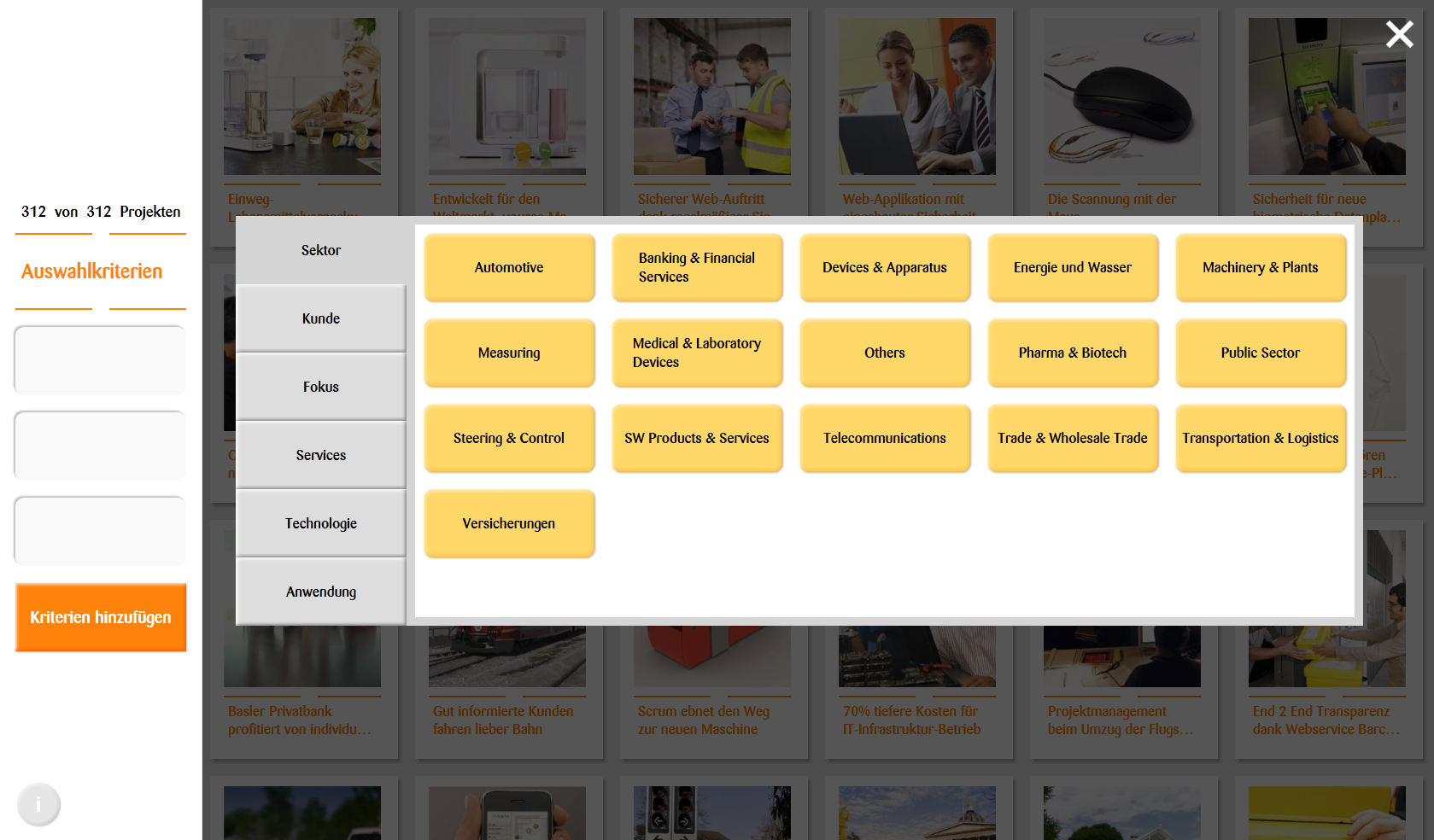


Abbildung 21 - Filter View

Wird nun ein Filterkriterium durch Antippen hinzugefügt, so wird dieses Kriterium in den Filter eingefügt, die Liste der Project Notes wird entsprechend gefiltert und die Ansicht wechselt zurück zur Übersicht (siehe Abbildung 22 - Gefilterte Übersicht).



Abbildung 22 - Gefilterte Übersicht

#### Info View

Wird in der Übersicht auf den Infoknopf gedrückt, so öffnet sich die Info View.

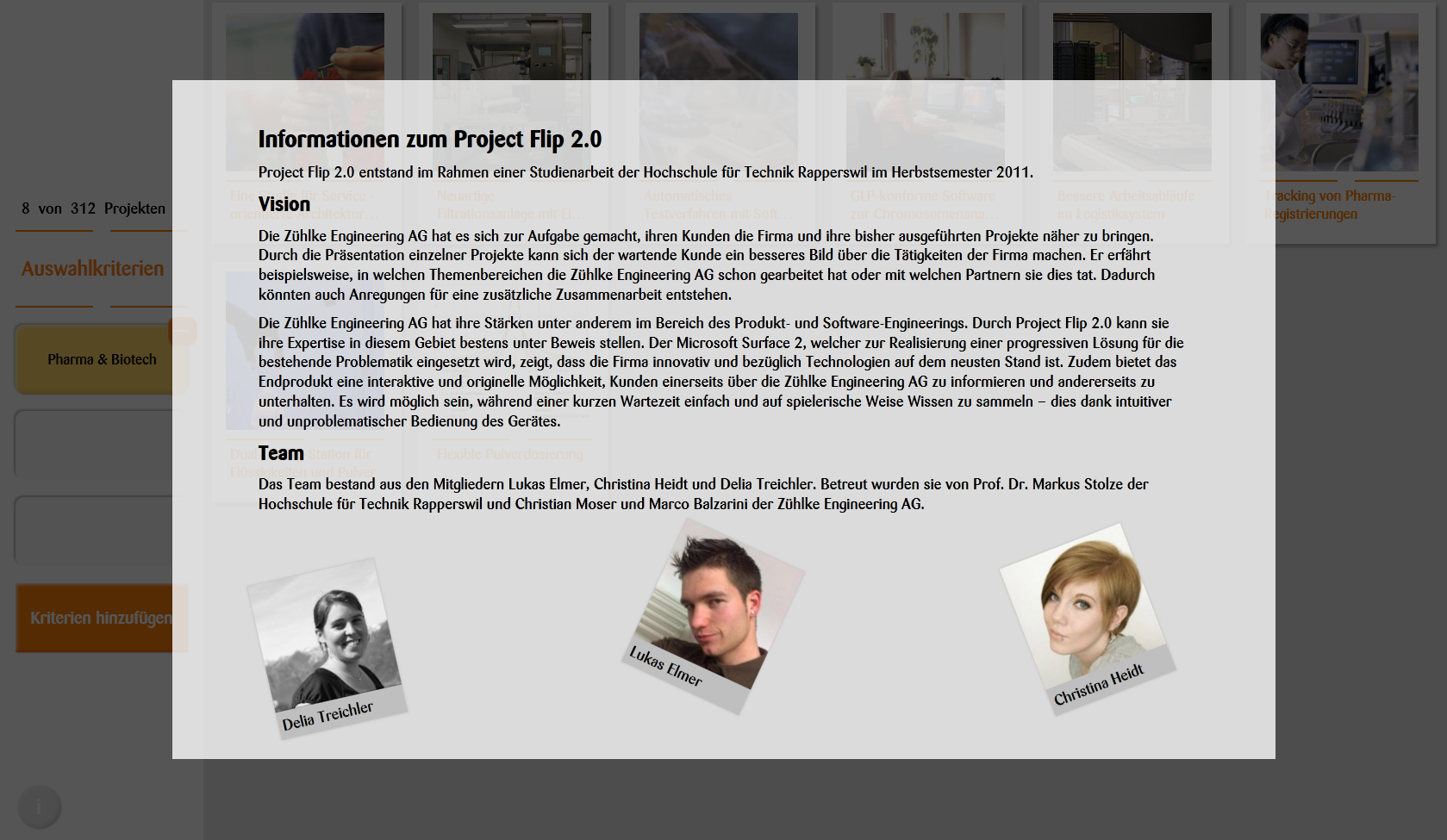


Abbildung 23 - Info View

#### Detail View

Wird in der Übersicht auf eine Project Note gedrückt, so öffnet sich diese in der Detailansicht.



Abbildung 24 - Detail View

Aus den Projektdaten kann ebenfalls ein Filterkriterium ausgewählt und dem Filter hinzugefügt werden. Mit den Pfeilen kann zur nächsten (nach rechts) beziehungsweise vorherigen (nach links) Project Note navigiert werden. Ausserdem kann, um den Text besser lesen zu können, die Project Note durch Antippen der Lupe vergrössert werden (siehe Abbildung 25 - Detail View mit Zoom).



Abbildung 25 - Detail View mit Zoom

## Prozesse und Threads

Die drei Tiers (Webserver, Sharepoint Server, Surface 2) agieren unabhängig. Da weder am Webserver noch am Sharepoint Server Anpassungen vorgenommen wurden, werden diese zwei Tiers nicht weiter beschrieben. Stattdessen wird auf den Surface 2 genauer eingegangen.

Als Grundmodell wird das .NET Framework mit WPF verwendet. Nachfolgend werden einige Spezialfälle, die in der Applikation programmiert wurden, genauer betrachtet.

### Konversation PDF zu XPS

Um das PDF in ein XPS zu konvertieren, wird der Adobe Reader und der XPS Dokument Writer (Drucker) benutzt. So muss für den Adobe Reader ein eigener Prozess gestartet werden, der dann nicht weiter kontrolliert werden kann aufgrund einer sauberen Schnittstellendefinition des Adobe Readers. Deshalb kann in der Konfiguration die Wartezeit („SecondsToWait“), bis der Adobe Prozess beendet wird, angegeben werden. Standardmässig ist dieser Wert mit 30 belegt, das heisst, der Adobe Reader hat maximal 30 Sekunden Zeit, bis er beendet wird.

### Parallel Preparer

Beim Preparer, also der Komponente, die für das Vorbereiten (Download PDF, Konvertierung PDF in XPS, Extraktion eines Bildes) der Project Notes verantwortlich ist, ist es sinnvoll, die Verarbeitung zu parallelisieren. Es wird für jede Project Note ein eigener Thread gestartet, da die Verarbeitung pro Project Note seriell erfolgen muss. Diese gestarteten Threads können dann unabhängig arbeiten. Wegen der grossen Verzögerung, welche beim Download oder beim Lesen von der Festplatte entstehen, ist eine parallele Verarbeitung performanter.

Vorteilhaft ist, dass die einzelnen Project Notes völlig unabhängig voneinander sind und deshalb nicht speziell synchronisiert werden müssen. Ausserdem wird der interne ThreadPool von .NET benutzt, welcher weitere Performancevorteile mit sich bringt.

### Asynchrones und verzögertes Laden des XPS Dokuments

In der Detailansicht wird das XPS Dokument angezeigt. Zu Beginn des Projektes wurde das XPS Dokument sofort geladen, sobald die Detailansicht geöffnet wurde. Dies führte dazu, dass der PC dem Laden des XPS Dokuments mehr Priorität gab als der Animation, die beim Öffnen der Detailansicht ausgeführt werden sollte. Daher war diese Animation kaum sichtbar.

Aus diesem Grund wird das Laden des XPS Dokuments erst nach Ablauf der Animation ausgeführt, also um etwa 500ms verzögert. Eine mögliche Umsetzung der Ladeverzögerung wäre das Erstellen eines neuen Threads, welcher zuerst 500ms schläft und danach das XPS Dokument lädt. Nun stellt sich aber das Problem, dass nur der GUI Thread das XPS Dokument laden und das Property im ViewModel setzen kann. Diese Einschränkung besteht, weil WPF nicht multithreading safe ist.

Deshalb wurde von der Betreuungsperson der Zühlke Engineering AG, Christian Moser, empfohlen, den Dispatcher mit einem Timer zu benutzen, um so das Dokument verzögert zu laden (TODO: ref Sitzungsprotokoll). Zusätzlich wird eine Synchronisation benötigt, welche mit einem Lock auf das aktuelle Objekt realisiert wird.

# Patterns

Die drei wichtigsten Patterns, die angewendet wurden, sind MVVM, Flyweight und Dependency Injection mit Microsoft Unity.

## MVVM

Das MVVM („Model“, „View“, „ViewModel) Pattern wird benötigt, um die View vom Model zu entkoppeln. Deshalb wird als Zwischenglied ein ViewModel erzeugt, das die Commands des GUIs abarbeitet und die verfügbaren Elemente dem GUI zur Verfügung stellt.

Die grundlegende Idee dahinter ist, dass sich das GUI schneller ändert als die Businesslogik und deshalb die zwei Komponenten möglichst stark abzutrennen sind. Zusätzlich kann das das ViewModel dadurch mit Unit Tests geprüft werden.

Weitere Informationen sind unter (TODO: ref) [<http://msdn.microsoft.com/en-us/magazine/dd419663.aspx>] zu finden.

## Flyweight

Flyweight wird für die Metadaten eingesetzt. Die Idee dabei ist, dass viele verschiedene Objekte die gleichen Objekte benötigen. Zum Beispiel benötigen mehrere Project Notes ein Metatag mit dem Namen „Java“. Damit dieses Objekt nicht immer wieder neu erstellt wird, referenzieren die verschiedene Project Notes das gleiche Metatag-Objekt.

Durch die Nutzung von Flyweight werden das Starten und der Betrieb der Applikation performanter, da einzelne Objekte wieder verwendet werden und nicht immer wieder neu erstellt werden müssen. Zusätzlich wird auch die Suche effizienter.

Der Nachteil besteht jedoch darin, dass die Flyweight Objekte immutable Objects, also unveränderlich, sein müssen. Deshalb verfügt das Metadata Model auch über keine Setter.

## Dependency Injection mit Microsoft Unity

Damit die Komponenten jederzeit und einfach ausgetauscht werden können, wurde mit Unity Containern (TODO: reference) gearbeitet, um Dependency Injection zu ermöglichen. Für dieses Projekt mag dies ein Overhead sein, aber speziell in grösseren Projekten ist Dependency Injection sehr wichtig. Denn so können beispielsweise, auf eine einfache Art und Weise, verschiedene Software Versionen ausgeliefert werden, indem die Container ausgetauscht werden. Dependency Injection kann auch beim Testen helfen, indem man beim Unity Container die Mock Objekte registriert.