|  |
| --- |
| Elmer Lukas, Heidt Christina, Treichler Delia  15. 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 |

## Inhaltsverzeichnis

[1 Dokumentinformationen 1](#_Toc311701117)

[1.1 Änderungsgeschichte 1](#_Toc311701118)

[1.2 Inhaltsverzeichnis 1](#_Toc311701119)

[2 Design Entscheide 2](#_Toc311701120)

[2.1 PDF auf Surface 2 Darstellen 2](#_Toc311701121)

[2.1.1 Varianten 2](#_Toc311701122)

[2.1.2 Nutzwertanalyse 2](#_Toc311701123)

[2.1.3 Sensitivitätsanalyse 3](#_Toc311701124)

[3 Architektur 3](#_Toc311701125)

[3.1 Interfaces 6](#_Toc311701126)

[3.1.1 Converter.Interfaces 6](#_Toc311701127)

[3.1.2 Services.Interfaces 7](#_Toc311701128)

[3.1.3 Loader.Interfaces 7](#_Toc311701129)

[3.2 Implementation 8](#_Toc311701130)

[3.3 Services.Loader 8](#_Toc311701131)

[3.4 Converter.Pdf 8](#_Toc311701132)

[3.5 Preparer 9](#_Toc311701133)

[3.6 Services 11](#_Toc311701134)

[3.7 UserInterface.Surface 12](#_Toc311701135)

[3.8 ProjectFlip 13](#_Toc311701136)

[4 Patterns 13](#_Toc311701137)

[4.1 MVVM 13](#_Toc311701138)

[4.2 Flyweight 13](#_Toc311701139)

[4.3 Dependency Injection mit Unity 14](#_Toc311701140)

# Design Entscheide

## PDF auf Surface 2 Darstellen

Eine sehr wichtige 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

Die 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 und vielleicht gar nicht möglich ist (kann zurzeit nicht genauer beurteilt werden, da die zu testende Hardware noch fehlt).

#### Variante 2: Umwandlung zu XPS

Das Surface 2 SDK erlaubt es, ohne weitere Libraries XPS Dokumente anzuzeigen (DocumentViewer, <http://msdn.microsoft.com/en-us/library/system.windows.controls.documentviewer.aspx>). Da ein XPS Dokument ähnlich wie ein PDF Dokument ist, können Textinhalte als Text gespeichert werden, wodurch beliebig weit herangezoomt werden kann.

#### Variante 3: Umwandlung zu Bild

Die einfachste Möglichkeit, um ein PDF auf dem Surface 2 darzustellen, ist eine Umwandlung des PDF Dokuments in ein Bild (z.B. PNG). Natürlich können Bilder 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 ist, dass die Bilder sehr gross sein würden, da der Text als Bild dargestellt werden müsste. Dadurch wird auch der maximale Zoom für die Bilder festgelegt.

### Nutzwertanalyse

In der Nutzwertanalyse geht die **Variante 2: Umwandlung zu XPS** als knapper 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. | | | | | | | | | |

### Sensitivitätsanalyse

In der Sensitivitätsanalyse wird untersucht, wie stark sich eine kleine Änderung auf das Ergebnis auswirken würde. Auf jeden Fall ist klar, dass die Variante 1 nicht als Sieger abschneiden kann. Zwischen der Variante 2 und 3 liegt schliesslich nur das Kriterium Darstellungsqualität, welches bei einer anderen Bewertung das Ergebnis stark beeinflussen würde. Da dieses Kriterium jedoch für Zühlke sehr wichtig ist und die Variante 3 in Project Flip 1.0 verfolgt wurde mit einem nicht vollständig zufriedenstellenden Ergebnis endete, ist die Wahl mit der **Variante 2: Umwandlung zu XPS**, doch 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 eingeteilt. Die drei Hauptkomponenten sind:

* Sharepoint Server
* Webserver <http://zuehlke.com>
* Microsoft Surface 2

Hier ein Überblick über die Komponenten und deren Rollen:

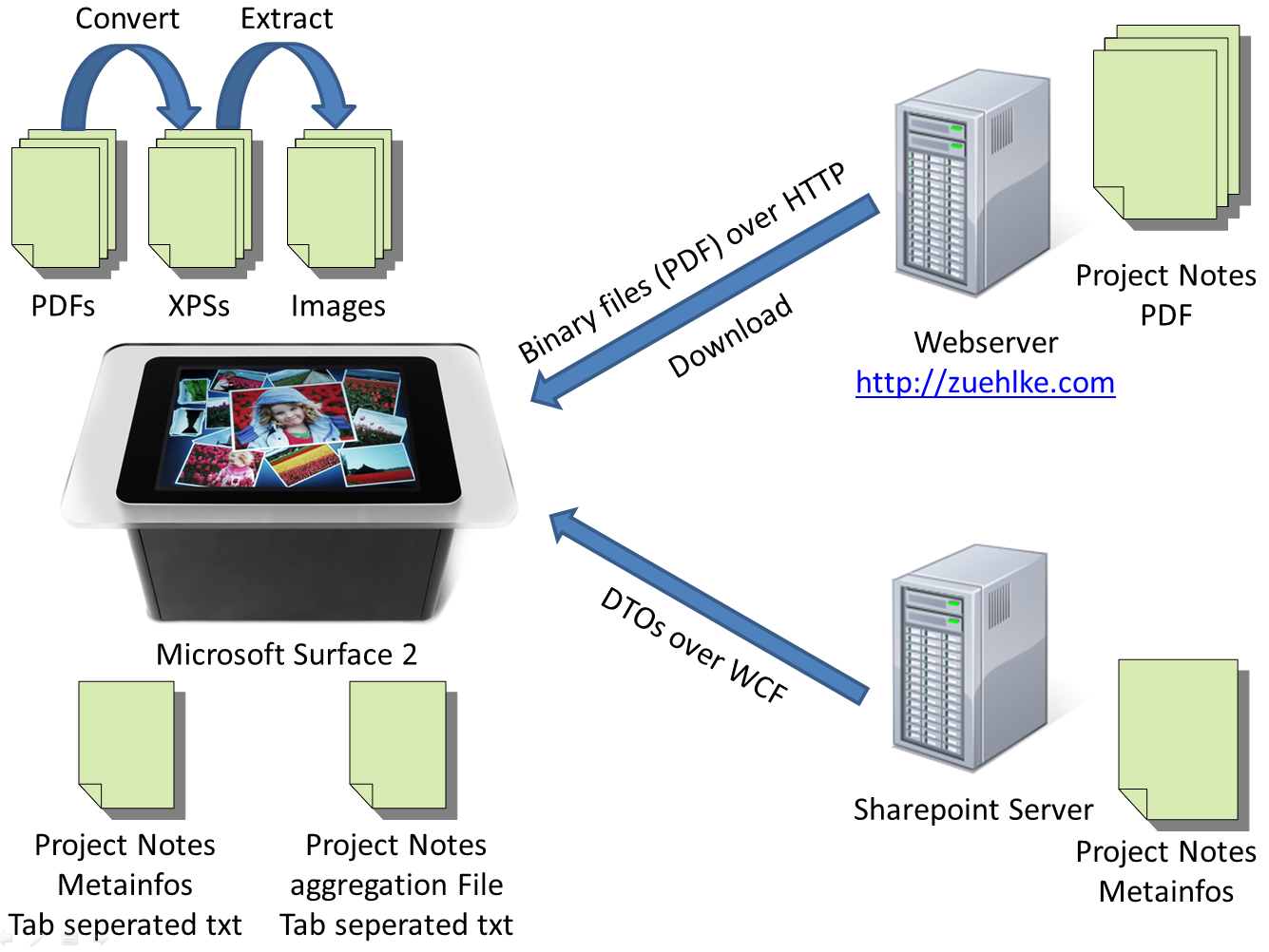


Abbildung 1 - Systemübersicht

### Sharepoint Server

Auf dem internen Sharepoint Server von Zühlke werden verschiedene Informationen verwaltet. Unter Anderem gehören hier auch die Project Notes und deren Metainformationen dazu. Nachdem eine neue Project Note publiziert wurde, werden dazugehörige Metainformationen generiert und auf dem Sharepoint Server abgelegt. So können verschiedene Personen und Applikationen diese Daten abrufen und verarbeiten.

In der aktuellen Implementation wird der Sharepoint Server nicht eingebunden, da dies den Zeitrahmen des Projektes gesprengt hätte.

### Webserver <http://zuehlke.com>

Der Webserver von Zühlke hostet natürlich die Internetseite von Zühlke. Interessant ist allerdings auch, dass dort die Projektreferenzen heruntergeladen werden können.

So wurde zu Beginn der Arbeit von der Zühlke ein statisches Textdokument bereitgestellt, in dem die Metainformationen vorhanden waren. Über diese Informationen können dann die PDF Dokumente direkt vom Webserver heruntergeladen werden, die dann auf dem Microsoft Surface 2 gespeichert werden.

### Microsoft Surface 2

Der Microsoft Surface 2 wird in der Eingangshalle von Zühlke stehen und direkt von den Kunden bedient werden. Um die Verfügbarkeit des Services sicherzustellen, muss der Surface möglichst unabhängig von den anderen Komponenten sein. Deshalb werden periodisch die Informationen auf den Surface heruntergeladen und verarbeitet. Dieser Vorgang ist natürlich getrennt von der Ausführung und wird noch genauer im Betriebskonzept (TODO: ref) beschrieben.

Wie im vorherigen Absatz beschrieben wurde, wurde von Zühlke ein statisches Texdokument mit den Metainformationen der Project Notes bereitgestellt. Dieses Dokument enthält alle wichtigen Informationen zu dem Project Notes.

Nachdem die PDF Dokumente heruntergeladen und lokal auf dem Microsoft Surface gespeichert wurden, werden die PDF Dokumente weiter verarbeitet. Als erstes werden die PDF Dokumente in XPS Dokumente konvertiert, in einem zweiten Schritt werden dann Bilder aus der XPS Datei extrahiert.

Eine wichtige Anforderung war es, dass die Applikation benutzerfreundlich ist. Deshalb musste sichergestellt werden, dass nicht zu viele Informationen auf einmal dargestellt werden. Aus diesem Grund müssen die Metadaten, die in einer sehr vielfältigen Ausprägungen in den Metadaten beschrieben sind (z.B. Java, JBoss, JavaBeans, JPA, usw.), auf einen gemeinsamen Nenner (z.B. Java) gemappt bzw. aggregiert werden. Aus diesem Grund liegt auf dem Surface 2 eine zweite Tab-getrennte Textdatei, die mit Excel bearbeitet werden kann, um die Metadaten zu aggregieren.

## Assemblies und Namespaces

Für so wenige Zeilen Code, wie im Projekt enthalten sind, könnte man schon fast sagen, dass der Code ein wenig overengineered ist. Dies wurde aber explizit von Zühlke gewünscht, da das Projekt auch noch weiterentwickelt werden soll.

Zum Beweis der Schichtenarchitektur wurde im Visual Studio 2010 ein Dependency Diagramm erstellt:



Abbildung 2 - Generierung Dependency Graph

In einem zweiten Schritt wurden dann die „Generics“ und die „Externals“ entfernt, damit das Ganze übersichtlicher wird. Anschliessend wurden die Tests, Implementation und Interfaces gruppiert und eingefärbt, damit das Diagramm übersichtlicher wird. Das sieht dann folgendermassen aus:



Abbildung 3 - Übersicht Abhängigkeiten

Auf der nächsten Seite sind weitere Details zu den Abhängigkeiten ersichtlich:

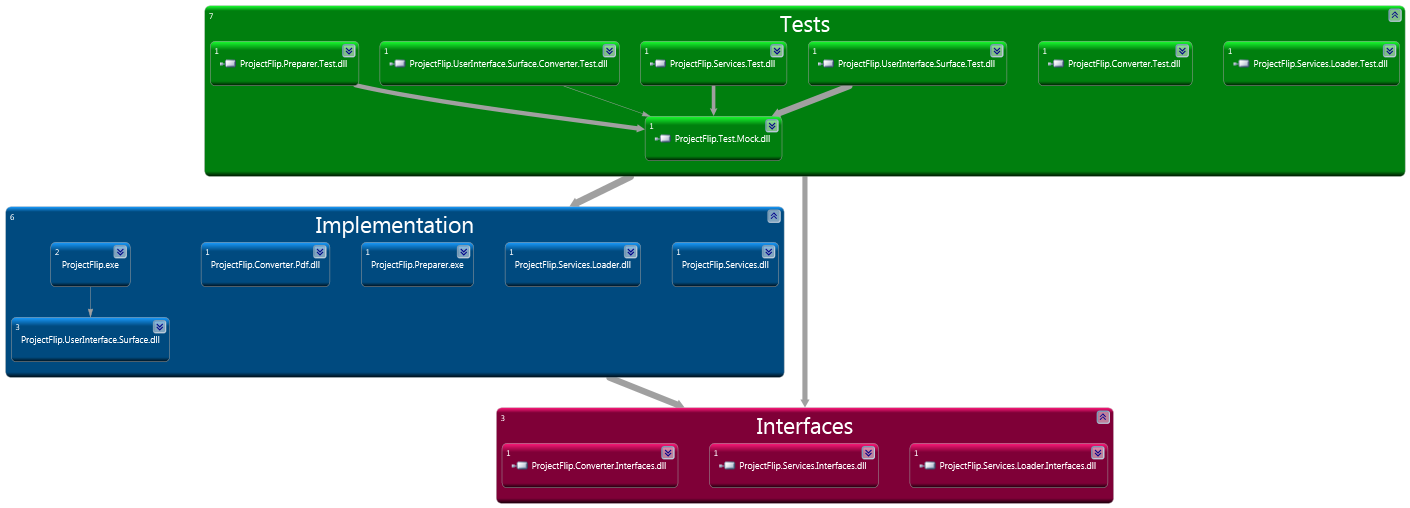
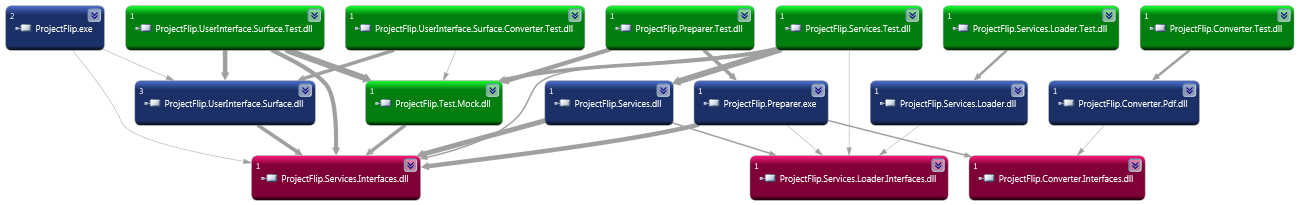


Abbildung 5 - Übersicht Abhängigkeiten mit Details

Abbildung 4 - Details Abhängigkeiten

Um auch noch mit einem alternativen Tool die Schichtung zu beweisen, anbei ein Diagramm, das von NDepend generiert wurde:

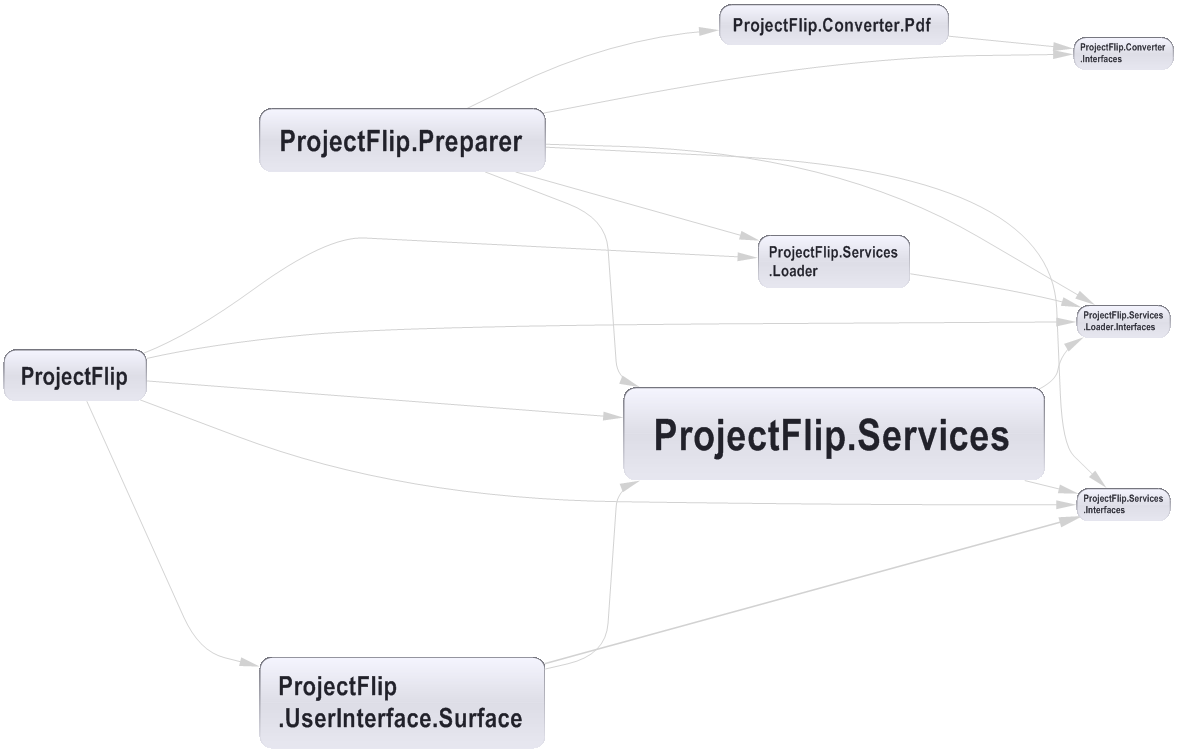


Abbildung 6 - NDepend Dependency Graph

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

Nun werden die einzelnen Projekte genauer beschrieben. Die Test Projekte werden nicht weiter dokumentiert, da ihre Namen schon aussagen, welche Projekte sie testen.

### Interfaces

Um die Implementation untereinander zu entkoppeln, wurden Interfaces eingesetzt:

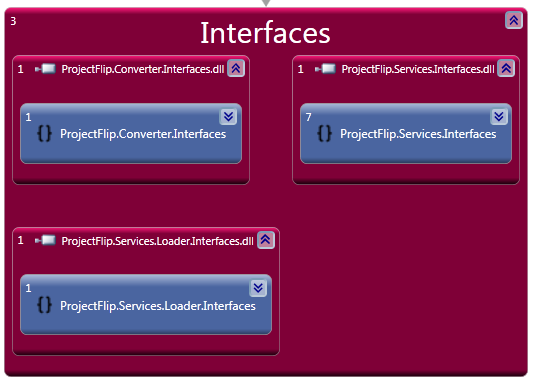


Abbildung 7 - Übersicht Interfaces

#### Converter.Interfaces

IConverter: Dieses Interface wird für die Konvertierung vom PDF Dokumenten in XPS Dokumente benutzt. So ist es einfach, diese Implementation auszutauschen. Dies wird in Zukunft, sobald mehr solche Konverter gratis zur Verfügung stehen und sich das XPS Dateiformat mehr durchgesetzt hat, auch gut möglich sein.

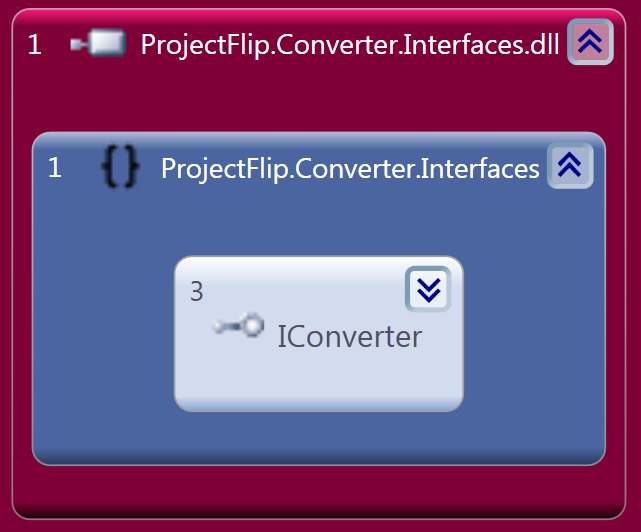


Abbildung 8 - Converter.Interfaces

#### Services.Interfaces

IAggregator, IMetadata, IMetadataType: Diese Interfaces sind für die Metadaten, die an den ProjectNotes hängen, verantwortlich. Es werden als Beispiel Dinge wie „C++“ als IMetadata und „Technologie“ als IMedatataType gespeichert. Der IAggregator ist für das Mapping verantwortlich, also wenn z.B. „JavaBeans“ nach „Java“ gemappt werden soll.

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. Es 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.

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 involvierte Team dar, also die einzelnen IPersons.

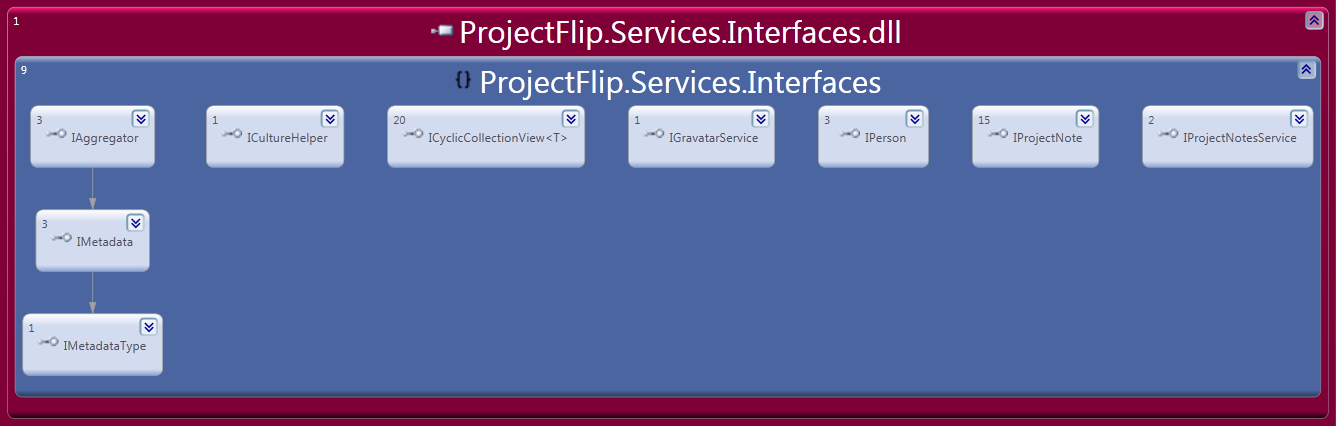


Abbildung 9 - Services.Interfaces

#### Loader.Interfaces

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

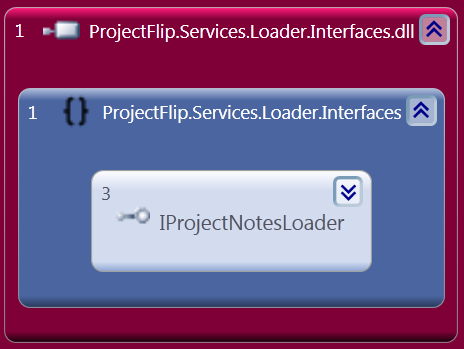


Abbildung 10 - Loader.Interfaces

### Implementation

Die Implementation ist natürlich von den Interfaces abhängig. Dadurch können die Implementationen voneinander entkoppelt werden. Hier die Übersicht:

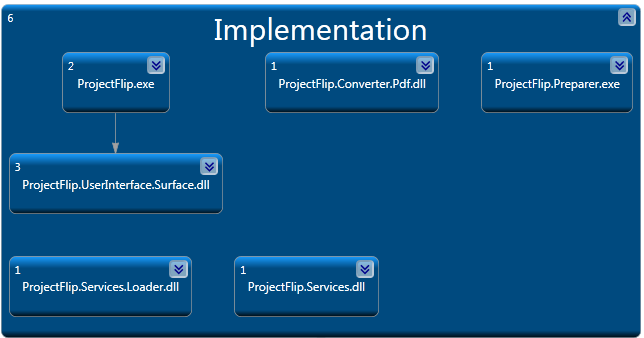


Abbildung 11 - Implementation Übersicht

#### Services.Loader

Der ProjectNotesLoader kann eine Liste von IProjectNotes laden. In der aktuellen Version werden die Daten von einem Tabulator getrenntem Textdokument gelesen, in Zukunft kann das aber auch von einer Datenbank oder vom SharePoint geladen werden.

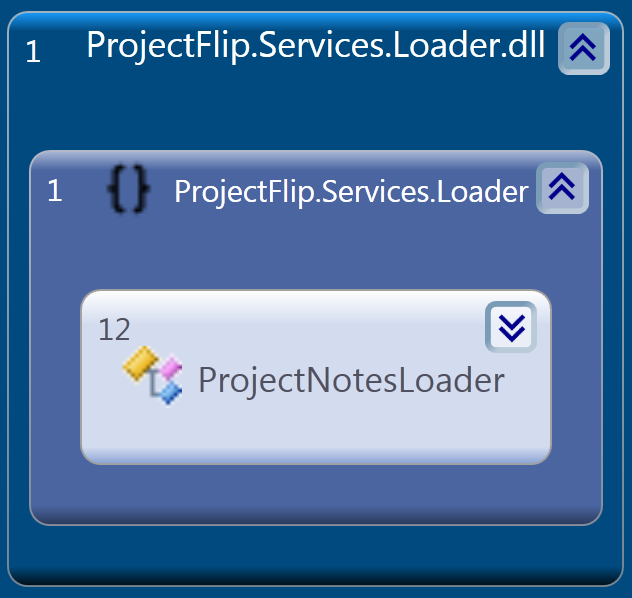


Abbildung 12 - Services.Loader

#### 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 der AcrobatLocation im Settings File konfiguriert werden. Stadardmässig ist der Pfad auf „C:\Program Files (x86)\Adobe\Reader 10.0\Reader\AcroRd32.exe“ eingestellt.

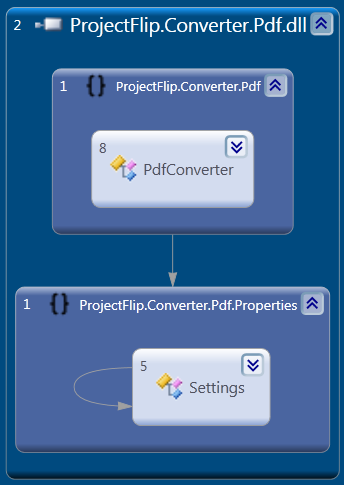


Abbildung 13 - Converter.Pdf

#### Preparer

Der Preparer ist dafür verantwortlich, dass die ProjectNotes 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 14 - 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 in ein XPS Dokument konvertiert. Der ImageExtractorProcessor extrahiert schliesslich noch ein Bild aus dem XPS Dokument. Sollte etwas schief gegangen sein und sich die Dateien in einem fehlerhaften Zustand befinden, räumt der CleanupProcessor schliesslich noch auf.

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



Abbildung 15 - Sequenzdiagramm Preparer

#### Services

Der Service ist dafür verantwortlich, die IProjectNotes und die aggregierten IMetadata Objekte zur Verfügung zu stellen. So erhält er zuerst die Rohdaten vom IProjectNotesLoader und baut dann aufgrund der Eingaben die Objektstruktur mit den IProjectNotes und IMetadatas auf. Die Metadaten werden gleich beim Import 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 Zühlke konfiguriert ist: http://www.zuehlke.com/uploads/tx\_zepublications/. Von diesem Verzeichnis aus werden die Project Notes heruntergeladen.

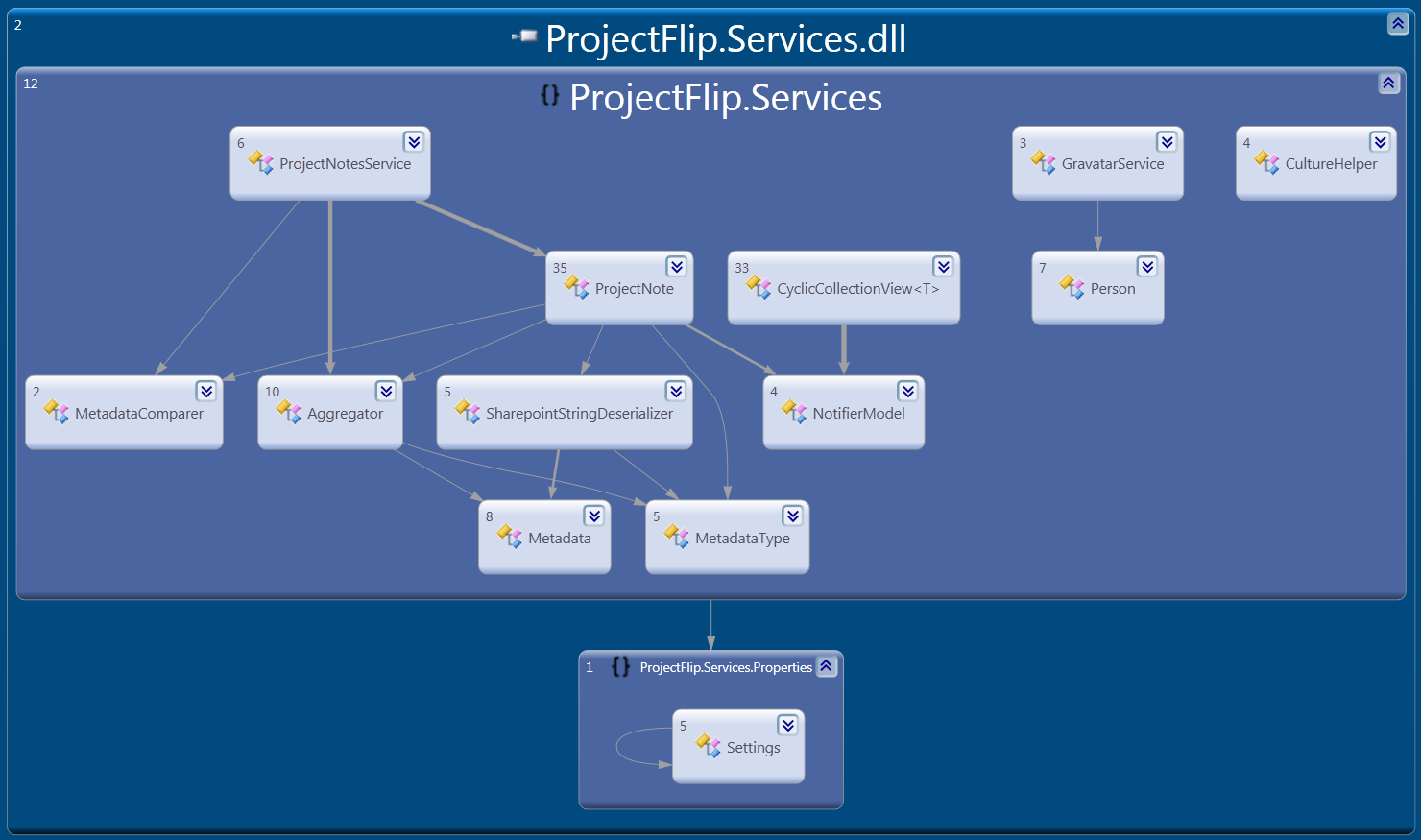


Abbildung 16 – Services

#### UserInterface.Surface

Das UserInterface.Surface Assembly ist für das GUI verantwortlich. Es benutzt zur Entkopplung der View und des Models das MVVM Pattern (siehe 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 notifizieren (Bsp. mithilfe von INotifyPropertyChanged). Das GravatarsViewModel verwaltet die Personen und stellt die Schnittstelle zwischen dem IGravatarService und dem Gravatars User Control dar.

Für die ICommand Implementation wird die Klasse Command implementiert, welche dann 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, während die Klassen AboutView, DetailView, FilterDetailView, FilterView, Gravatars, ZoomInIcon und ZoomOutIcon einzelne UserControls darstellen, die vom OverviewWindow benutzt werden.

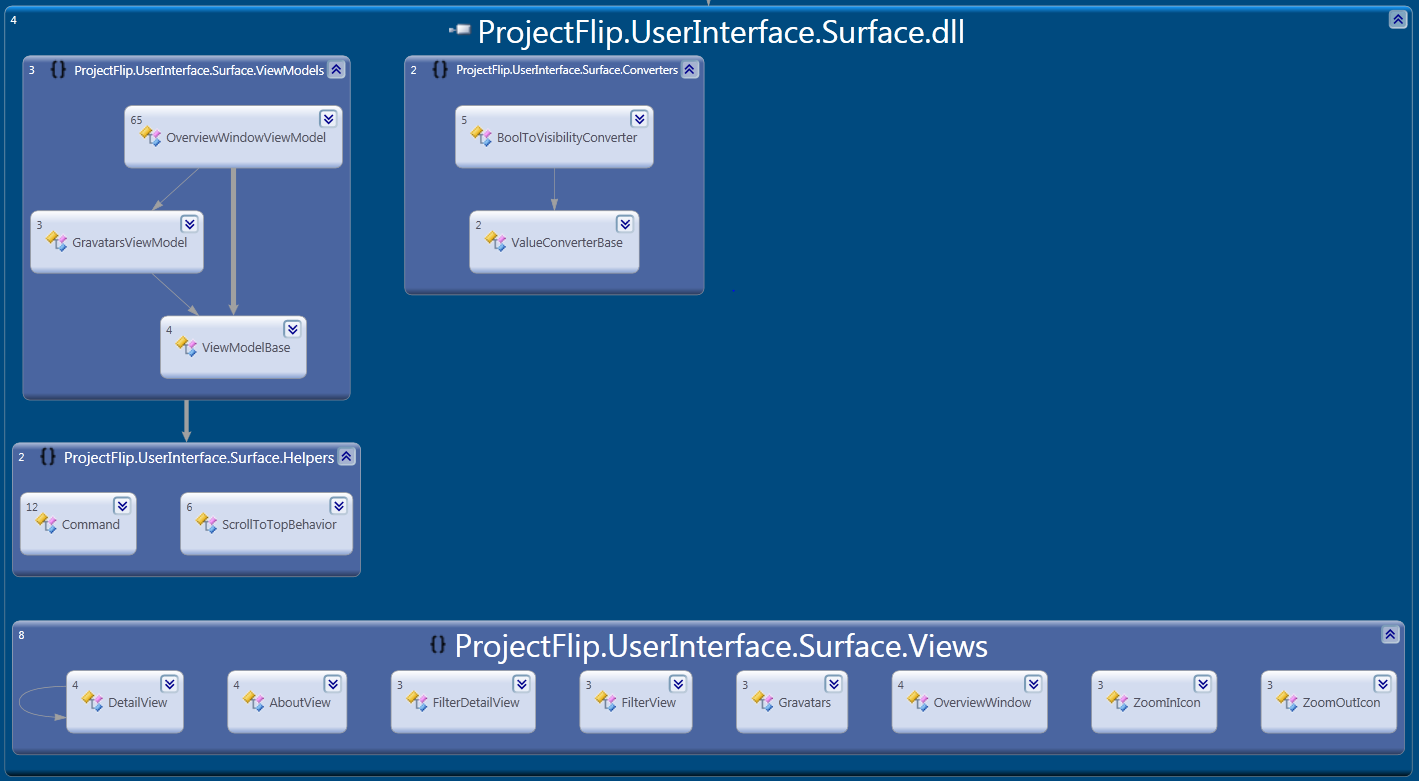


Abbildung 17 - UserInterface.Surface

#### ProjectFlip

Dieses Projekt ist das Startup Projekt. Es ist dafür verantwortlich, die richtigen Komponenten zusammenzufügen und schlussendlich das GUI zu starten. Hier wurde stark Dependency Injection eingesetzt, und zwar mit Unity (von Microsoft). So kann an einem Ort konfiguriert werden, welche Komponenten instanziiert werden sollen.

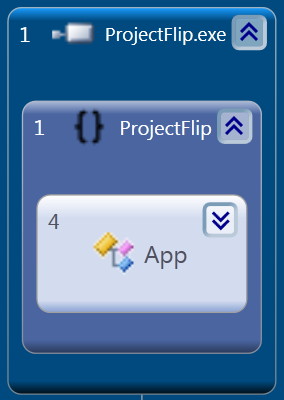


Abbildung 18 - ProjectFlip

## UI Design

### Allgemein

Für unsere Applikation wird WPF verwendet. Um die View vom Model zu trennen, wird MVVM (TODO: ref) eingesetzt. Durch den Einsatz vom MVVM konnte fast vollständig auf Code Behind verzichtet werden, wodurch die View Logik im ViewModel auch über Unit Tests getestet werden konnten.

### Unterteilung der Views

In vielen Applikationen kann zwischen verschiedenen Ansichten gewechselt werden. Bei Project Flip 2.0 ist das 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 abgedunkelten Hintergrund dargestellt, sodass der Hintergrund immer noch ersichtlich ist.

#### Overview

Die Übersicht ist in zwei Teile eingeteilt: die Filter Kriterien und die Listenansicht. Das sieht dann so aus:

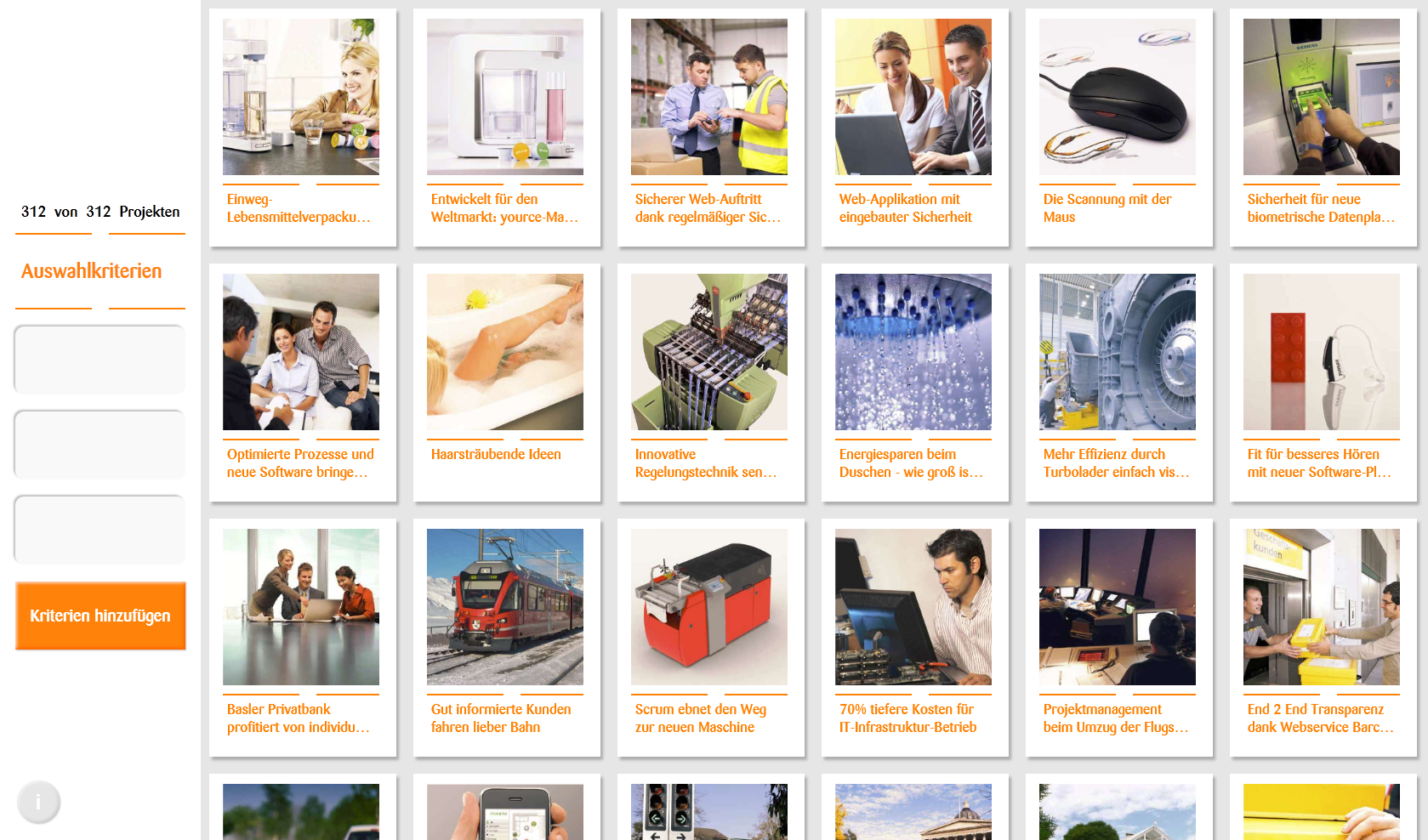


Abbildung 19 - Übersicht

Auf der linken Seite sind die Filterkriterien ersichtlich, während auf der rechten Seite die Project Notes aufgelistet werden.

#### Filter View

Wird auf der Übersicht auf dem Button „Kriterien hinzufügen“ klickt, so wird die Filter View geöffnet:

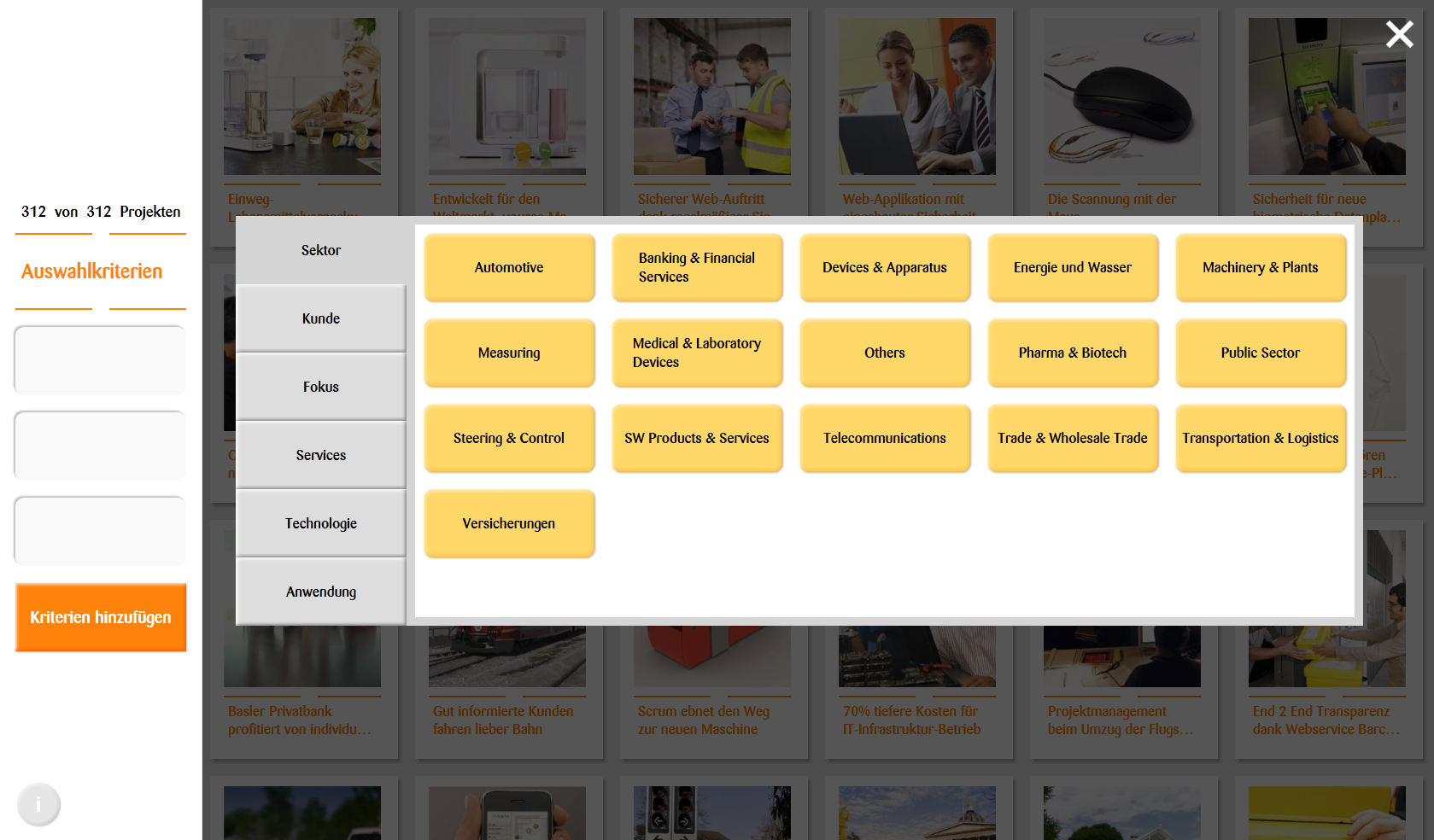


Abbildung 20 - Filter View

Wenn nun ein Filterkriterium hinzugefügt wird, so wird dieses Kriterium in der Übersicht eingefügt und die Liste wird entsprechend gefiltert:

Abbildung 21 - Gefilterte Übersicht



Abbildung 22 - Gefilterte Übersicht

#### Info View

Wird in der Übersicht auf den Info Button gedrückt, so wird die Info View eingeblendet:

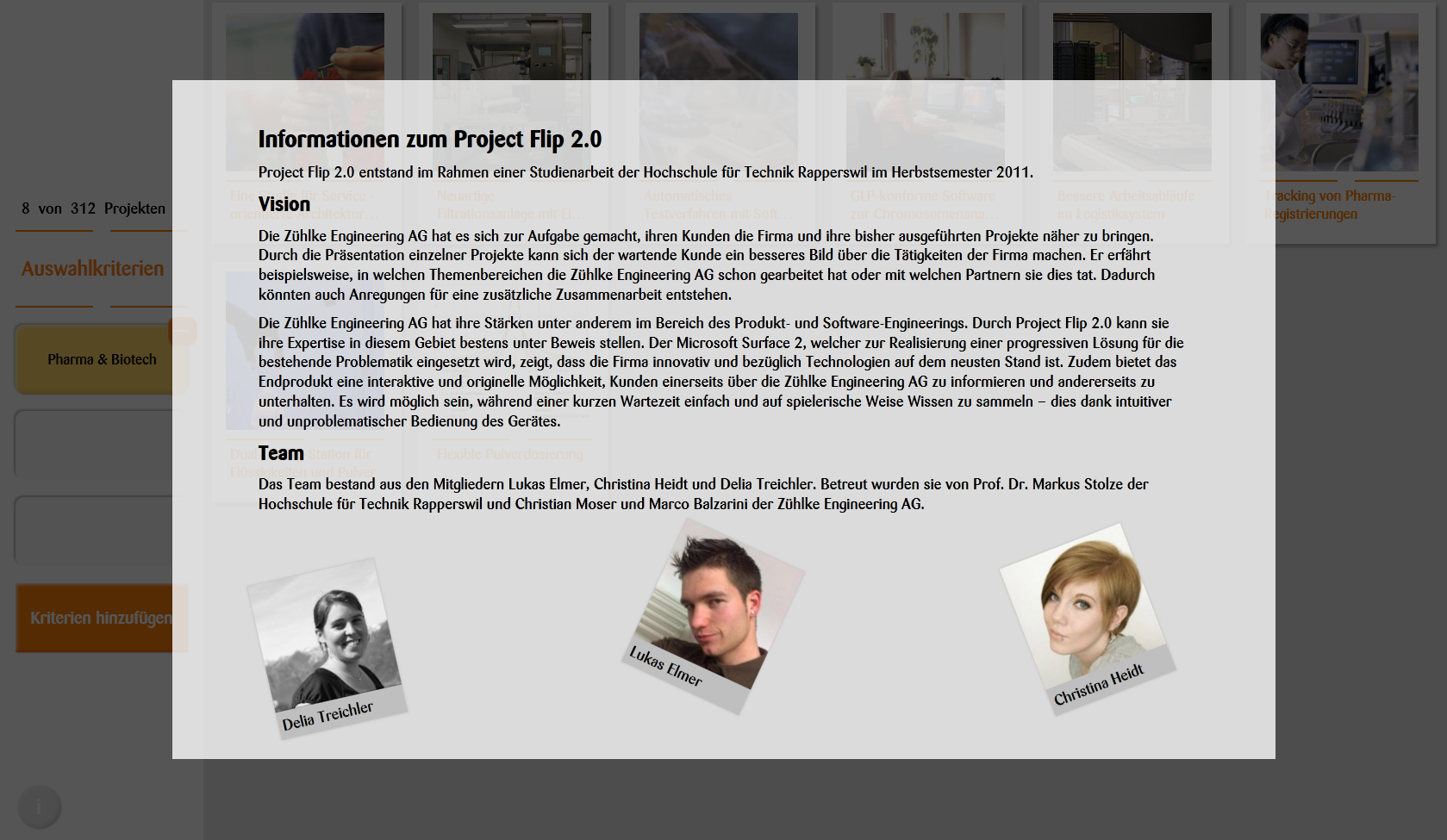


Abbildung 23 - Info View

#### Detail View

Die Detailansicht wird beim Drücken auf eine Project Note geöffnet:



Abbildung 24 - Detail View

In dieser Ansicht kann, wie in der Filter View, nach einem weiteren Kriterium gefiltert werden. Mit den Pfeilen links und rechts kann zum nächsten bzw. vorherigem Project Note navigiert werden. Ausserdem kann, um den Text zu lesen, mit der Lupe gezoomt werden:



Abbildung 25 - Detail View mit Zoom

## Prozesse und Threads

Die drei Tiers (Webserver, Sharepoint Server, Surface 2) laufen grundsätzlich unabhängig. Da weder am Webserver noch am Sharepoint Server etwas angepasst wurde, werden diese zwei Tiers nicht weiter beschrieben. Stattdessen wird auf den Surface 2 genauer eingegangen.

Als Grundmodell wird das .NET Framework mit WPF verwendet. Auf diese Konzepte wird nicht genauer eingegangen, da diese zum Grundstudium eines .NET / WPF Entwickler gehören. Dafür werden hier einige Spezialfälle, die in der Applikation programmiert wurden, genauer betrachtet.

### Konversation PDF zu XPS

Um das PDF zum XPS zu transformieren, 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, Konversion PDF zu XPS, Extraktion eines Bildes) der Project Notes verantwortlich ist, macht es Sinn, die Verarbeitung zu parallelisieren. Da pro Project Note die Verarbeitung seriell erfolgen muss, muss pro Project ein eigener Thread gestartet werden. Dieser kann dann unabhängig arbeiten und wegen der grossen Verzögerung z.B. beim Download oder beim Lesen von der Festplatte ist es parallel performanter.

Das Schöne an der Sache ist die, dass die einzelnen Project Notes völlig unabhängig voneinander sind und deshalb nicht speziell synchronisiert werden müssen. Ausserdem wird der interne ThreadPool von .NET benutzt, was weiter Performancevorteile bringt.

### Asynchrones und verzögertes Laden des XPS Dokumentes

In der Detailansicht der Project Note muss das XPS Dokument angezeigt werden. Zu Beginn des Projektes war es so, dass das XPS Dokument sofort geladen wurde, wenn die Detailansicht geöffnet wurde. Dies führte leider dazu, dass der PC dem Laden des XPS mehr Priorität gab als der Animation, die beim Öffnen des XPS ausgeführt werden sollte.

Die Idee war dann, dass wir das XPS erst nach der Animation laden, also um etwa 500ms verzögert. Also könnte man sich vorstellen, einfach einen neuen Thread zu erstellen, der zuerst 500ms schläft. Da gibt es allerdings ein weiteres Problem beim Laden des XPS Dokumentes, und zwar kann nur der GUI Thread das XPS Dokument öffnen und das Property im ViewModel setzten, da nur dieser Thread dazu berechtigt ist weil WPF nicht Multithreading Safe ist.

Deshalb wurde uns von Christian Moser (TODO: ref Sitzungsprotokoll) empfohlen, dass wir den Dispatcher mit einem Timer benutzen, um so das Dokument verzögert zu laden. Natürlich muss das Ganze noch synchronisiert werden, was mit einem Lock auf das aktuelle Objekt realisiert ist.

# Patterns

Die drei wichtigsten Patterns, die wir angewendet haben und hier beschreiben möchten, sind MVVM, Flyweight und Dependency Injection mit Unity.

## MVVM

Das MVVM („Model“, „View“, „ViewModel) Pattern wird benötigt, um die View von dem 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 Idee dahinter ist, dass sich das GUI schneller ändert als die Businesslogik und deshalb macht es Sinn, diese zwei Komponenten möglichst stark abzutrennen. Ein weiterer wichtiger Punkt ist, dass das ViewModel mit Unit Tests getestet werden kann.

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

## Flyweight

Flyweight (Deutsch: Fliegengewicht) wird für die Metadaten (Metadata) eingesetzt. Die Idee hinter dem Flyweight ist, dass viele Objekte andere gleiche Objekte benötigen, zum Beispiel ein Metatag mit dem Namen „Java“. Damit nicht sehr viele Objekte erstellt werden, referenzieren verschiedene ProjectNotes einen gleichen Metatag.

So werden das Starten und auch der Betrieb der Applikation performanter, weil nicht so viele Objekte erstellt werden müssen. Zusätzlich wird auch die Suche effizienter.

Der Nachteil an der Sache ist, dass die Flyweight Objekte immutable Objects, also unveränderlich, sein müssen. Deshalb hat das Metadata Model auch keine Setter.

## Dependency Injection mit 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 dass ein Overhead sein, der nicht unbedingt nötig wäre, aber speziell in grösseren Projekten ist Dependency Injection sehr wichtig. Denn so können z.B. auf eine einfache Art und Weise verschiedene Software Versionen ausgeliefert werden, indem die Container ausgetauscht werden. Dies kann auch beim Testen helfen, sodass man bei genau diesem Container die Mock Objekte registriert.