HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

UniversitätsSpital
Zürich

Operationsaufklärung mit Windows 8

Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Herbstsemester 2013/2014

Autor(en):	Oskar Knobel
	Reto Lämmler
Betreuer:	Simon Gubler
Verantwortlicher:	Prof. Hansjörg Huser
Projektpartner:	UniversitätsSpital Zürich, Klinik für Geburtshilfe
Experte:	Stefan Zettel, Ascentiv AG, Zürich
Gegenleser:	Prof. Dr. Josef Joller

Aufgabenstellung

Operationsaufklärung mit Windows 8

Aufgabenstellung für Oskar Knobel und Reto Lämmler

Ausgangslage:

Die Klinik für Geburtshilfe bereitet ihre Patienten jeweils mit einem Operationsaufklärungsprotokoll auf die bevorstehende Operation vor. Im Verlauf des Protokolls beschreibt der zuständige Arzt der Patientin den Vorgang der Operation und die möglichen Risiken auf einem Papierformular mit Hilfe von Skizzen. Wenn die Patientin mit den Operationsbedingungen einverstanden ist, unterschreibt sie daraufhin das Formular, das dann gescannt und archiviert wird. Die Unterschrift dient zur rechtlichen Absicherung des Spitals.

Ziel der Arbeit:

Das Ziel der Arbeit ist ein Windows 8-Prototyp, der das Formular auf einem Tablet implementiert. Analog der Papiervariante soll der Prototyp die Patientin über die Operation und die Risiken informieren. Die Applikation soll das PDF der Operationsbeschreibung sowie multimediale Elemente zeigen, z.B. beschreibende Texte, Skizzen, Videos der Operation oder Fotos. Falls dies rechtlich möglich ist, soll die Patientin zusätzlich direkt auf dem Tablet das Formular unterschreiben können. Um das Formular zu personalisieren, soll der Prototyp über einen WCF-Webservice Patientendaten vom KIS (Klinischen Informationssystem) abfragen können. Der Webservice soll auch eine Methode enthalten, die das unterschriebene Formular übernimmt und im KIS abspeichert, z.B. im PDF-Format.

Resultate

- Ein Windows 8-Prototyp für die Darstellung und Editierung des Formulars
- Eine Abklärung zur rechtlichen Situation von digitalen Unterschriften
- Ein nur über das interne Netz erreichbarer WCF-Webservice für das KIS zum Lesen von Patientendaten und Abspeichern des unterschriebenen Formulars oder ein Mock-Service

Auftraggeber

UniversitätsSpital Zürich, Klinik für Geburtshilfe, Prof. Dr. Juozas Kurmanavicius, Jihad Taktak

Projektteam

- Oskar Knobel (oknobel@hsr.ch)
- Reto Lämmler (r1laemml@hsr.ch)

Betreuung HSR

Betreuer: Simon Gubler
sgubler@hsr.ch

Verantwortlicher: Prof. Hansjörg Huser
hhuser@hsr.ch
Tel: 055 222 49 12
HSR Raum 6.010

Projektabwicklung

Termine

- Beginn der Arbeit: Mo., 16. Sept. 2013
- Zwischenpräsentation für Gegenleser: 6. Nov. 2013
- Abgabetermin Abstract für DA-Broschüre: 12.Dez.2013
- Abgabetermin Kurzfassung/Poster/Mgmt-Summary zum Review: 16. Dez. 2013
- Abgabetermin (inkl. Poster): 20. Dez. 2013, 12:00 Uhr
- Mündliche BA-Prüfung: 16.01.2014, 13:30 Uhr – 14:30 Uhr
- Zwischenbesprechung/Review mit Auftraggeber nach Projektplan

Arbeitsaufwand

Für die erfolgreich abgeschlossene Arbeit werden 12 ECTS angerechnet. Dies entspricht einer Arbeitsleistung von mind. 360 Stunden pro Student. (14 Wochen zu ca. 27h)

Hinweise für die Gliederung und Abwicklung des Projektes

Gliedern Sie Ihre Arbeit in 4 bis 5 Teilschritte. Schliessen Sie jeden Teilschritt mit einem Meilenstein ab. Definieren Sie für jeden Meilenstein, welche Resultate dann vorliegen müssen!

Folgende Teilschritte bzw. Meilensteine sollten Sie in Ihrer Planung vorsehen:

Schritt 1: Projektauftrag inkl. Projektplan (mit Meilensteinen),

- Meilenstein 1: Review des Projektauftrages abgeschlossen. Projektauftrag von Auftraggeber und Dozent genehmigt
  Letzter Meilenstein: Systemtest abgeschlossen
  Termin: ca. eine Woche vor Abgabe
- Entwickeln Sie Ihre SW in einem iterativen, inkrementellen Prozess: Planen Sie möglichst früh einen ersten lauffähigen Prototypen mit den wichtigsten und kritischsten Kernfunktionen. In die folgenden Phasen können Sie dieses Kernsystem schrittweise ausbauen und testen.
- Falls Sie in Ihrer Arbeit neue oder Ihnen unbekannte Technologien einsetzen, sollten Sie parallel zum Erarbeiten des Projektauftrages mit dem Technologiestudium beginnen.
- Setzen Sie konsequent Unit-Tests ein! Verwalten Sie ihre Software und Dokumente auf einem geeigneten Repository. Stellen Sie sicher, dass der/die Betreuer jederzeit Zugriff auf das

Repository haben und dass das Projekt anhand des Repositories jederzeit wiederhergestellt werden kann.

- Achten Sie auf die Einhaltung guter Programmier- und Designprinzipien
  - DRY, high cohesion, loose coupling, etc.
  - Clean Code!
- Halten Sie sich im Übrigen an die Vorgaben aus dem Modul SE-Projekt.

Projektadministration

- Führen Sie ein individuelles Projekttagebuch aus dem ersichtlich wird, welche Arbeiten Sie durchgeführt haben (inkl. Zeitaufwand). Diese Angaben sollten u.a. eine individuelle Beurteilung ermöglichen.
- Dokumentieren Sie Ihre Arbeiten laufend. Legen Sie Ihre Projektdokumentation mit der aktuellen Planung und den Beschreibungen der Arbeitsresultate elektronisch in einem Projektordner ab. Dieser Projektordner sollte jederzeit einsehbar sein (z.B svn-Server oder File-Share).

Inhalt der Dokumentation

Bei der Abgabe muss jede Arbeit folgende Inhalte haben:

- Dokumente gemäss Vorgabe: https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html
- Aufgabenstellung
- Technischer Bericht
- Projektdokumentation
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind.
- Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen.
- Projekttagebuch, Dokumentation des Projektverlaufes, Planung etc.

Fortschrittsbesprechung

Regelmässig findet zu einem fixen Zeitpunkt eine Fortschrittsbesprechung statt.

Teilnehmer: Dozent, Betreuer und Studenten, bei Bedarf auch Vertreter der Auftraggeber

Termin: jeweils Mittwoch, 13:00 Uhr, Raum 6.010 (Abweichungen werden rechtzeitig kommuniziert)

Traktanden:

- Was wurde erreicht, was ist geplant, welche Probleme stehen an
- Review von Code/Dokumentation (Abgabe jeweils einen Tag vor dem Meeting

Falls notwendig, können weitere Besprechungen / Diskussionen einberufen werden.

Sie erstellen zu jeder Besprechung ein Kurzprotokoll, welches Sie spätestens 2 Tage nach der Sitzung per Email an den Betreuer senden.

Rapperswil, 16. Dezember 2013

Hansjörg Huser

Abstract

Für die Klinik für Geburtshilfe am UniversitätsSpital Zürich soll eine Windows Applikation für Tablets entwickelt werden, welche das zur Operationsaufklärung eingesetzte Formular ersetzen soll. Dieses Formular wird bisher vor einem Eingriff vom Arzt mit der Patientin zusammen ausgefüllt. Die Patientin wird mittels Skizzen aufgeklärt und über die Risiken informiert.

Ziel der Bachelorarbeit ist es, diesen Prozess zu modernisieren und mit einer auf Tablets lauffähigen App durchzuführen. Dabei sollen multimediale Inhalte und Touch-Fähigkeit für die Neugestaltung dieses Prozesses genutzt werden. Abschliessend steht das ursprünglich genutzte Formular digital zur Verfügung und kann mit dem Stift von Arzt und Patientin unterschrieben und abgeschlossen werden, worauf es archiviert und als PDF abgelegt wird.

Im Rahmen dieser Bachelorarbeit wurde eine Windows Desktop App auf .NET Basis mit WPF entwickelt, welche im Windows 8 Modern Style für Tablets mit Touch-Input optimiert ist. Dazu gehört auch ein WCF Web Service, welcher nach Abschluss des Projektes mit PERINAT, dem bestehenden Informationssystem der Klinik für Geburtshilfe, verknüpft wird.

Die Hauptaufgabe der Applikation ist die Durchführung des oben beschriebenen Prozesses. Damit dieser Aufklärungsprozess simuliert werden konnte, wurden die benötigten Daten (Patienten, Operationstypen, Medien) auf einer lokalen Datenbank gemockt und dann via Web Service asynchron für die App bereitgestellt. Um den Prozess zu unterstützen und die Daten zu verwalteten, bietet die Applikation Master/Detail-Ansichten für Patienten, Operationstypen, Risiken und Multimediadateien, auf welchen Datensätze hinzugefügt, bearbeitet oder entfernt werden können.

Management Summary

Ausgangslage

In der Klinik für Geburtshilfe im Universitätsspital Zürich wird vor jedem operativen Eingriff ein Aufklärungsgespräch mit der Patientin geführt. Bei diesem Prozess füllt der Arzt ein vorgedrucktes Formular aus, wo er die geplanten Eingriffe und die damit verbundenen Risiken beschreibt. Er erklärt der Patientin mündlich und anhand von Skizzen, wie die Operation ablaufen wird.

Die geplanten Eingriffe und jedes während der Aufklärung angesprochene Risiko muss auf dem Formular aufgeführt werden. In der Regel gibt es für jeden Operationstyp einen Grundstock an Risiken, welche jedes Mal von Hand aufgeschrieben werden müssen. Weitere Angaben sind „Datum“ und „Dauer der Aufklärung“ sowie ein optionales Feld „Übersetzer“, welche von Hand eingetragen werden.

Falls die Patientin mit den bevorstehenden Eingriffen einverstanden ist, wird am Ende des Prozesses das Formular von den beiden Parteien „Arzt/Ärztin“ und der „Patientin“ unterschrieben. Alternativ kann es auch vorkommen, dass die Patientin nicht einverstanden ist und der Prozess darin resultiert, dass kein Eingriff vorgenommen werden darf.

Mit der Unterschrift wird der Eingriff rechtsgültig autorisiert. Danach muss das Dokument von Hand eingescannt und archiviert werden.

Ziel der Bachelorarbeit ist es, diesen Prozess zu modernisieren und mit einer auf Tablets lauffähigen App durchzuführen. Dabei sollen multimediale Inhalte und Touch-Fähigkeit für die Neugestaltung dieses Prozesses genutzt werden. Abschliessend steht das ursprünglich genutzte Formular digital zur Verfügung und kann mit dem Stift von Arzt und Patientin unterschrieben und abgeschlossen werden, worauf es archiviert und als PDF abgelegt wird.

Vorgehen und Technologien

In einem ersten Schritt wurden die Anforderungen an die Applikation beim Industriepartner abgeholt und in einer Anforderungsanalyse dokumentiert. Zudem musste abgeklärt werden, welche Art von Windows Applikation umgesetzt werden kann. Aufgrund hoher Lizenzkosten sowie allfälligen Einschränkungen wurde auf eine Umsetzung als Windows Store App verzichtet und stattdessen eine Windows Desktop Applikation geplant.

Im Anschluss wurde ein erster Prototyp erstellt, um den Prozess des Aufklärungsgespräches in der Applikation abzustecken und geplante Features, wie zum Beispiel das Unterschreiben auf dem Tablet oder das Exportieren von PDF-Dateien, zu testen. Da der Prozess eine Vielzahl an Informationen benötigt, wurde er in Form eines Wizards umgesetzt, in welchem der Benutzer Schritt für Schritt die benötigten Eingaben vornehmen kann und dabei entsprechend unterstützt wird. Dieser Prototyp wurde dann dem Industriepartner präsentiert und validiert.

Da die Buttons und Eingabefelder in der Applikation noch viel zu klein waren, war die Bedienung mittels Touch-Eingaben auf dem Tablet noch nicht optimal. Deshalb wurden Papierprototypen

erstellt um die Benutzbarkeit zu verbessern. Anhand dieser Skizzen wurde dann die Applikation umgestaltet und optimiert.

Schliesslich wurden die gewünschten Anforderungen des Industriepartners weiter umgesetzt und in Abständen von 2-3 Wochen im Universitätsspital präsentiert und besprochen.

Ergebnis

Das Ergebnis dieser Bachelorarbeit ist eine voll funktionsfähige Windows Applikation, welche von der Klinik für Geburtshilfe zu einem späteren Zeitpunkt eingesetzt werden kann. Die Applikation kann sowohl auf dem Tablet für das Aufklärungsgespräch als auch auf dem Desktop Computer für die Erfassung von Daten genutzt werden.

Um die Eingriffe der Patientin zu veranschaulichen, sind für die verschiedenen Operationstypen vordefinierte Skizzen hinterlegt, auf welchen mittels Stift zusätzliche Zeichnungen gemacht werden können. Zudem können auch Videos abgespielt werden. Damit der Arzt während der Aufklärung nicht vergisst über ein Risiko aufzuklären, müssen diese mittels Checkliste abgearbeitet werden. Zusätzliche Informationen wie Arzt oder Dauer der Aufklärung müssen nicht mehr manuell auf dem Formular eingetragen werden, sondern werden automatisch aufgeführt. Das Formular kann vom Arzt und der Patientin direkt auf dem Tablet unterschrieben und dann mittels PDF-Export auf eine Datenablage kopiert werden.

Damit die benötigten Daten verwaltet werden können, wurden in der Applikation Verwaltungsmasken für die Patientinnen, die Operationstypen sowie für die Operationsrisiken umgesetzt.

Ausblick

Die Applikation soll nach Abschluss des Projektes im Universitätsspital mit PERINAT, dem bestehenden Informationssystem der Klinik für Geburtshilfe, verknüpft und getestet werden. Die Applikation soll ab dem nächsten Jahr produktiv zum Einsatz kommen und nimmt damit eine Vorreiterrolle bei der Einführung neuer EDV-Lösungen im medizinischen Umfeld des Universitätsspitals Zürich ein. Bei erfolgreichem produktivem Betrieb in der Klinik für Geburtshilfe kann ein universitätsspitalweiter Einsatz in Betracht gezogen werden.

Eigenständigkeitserklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Rapperswil, 16. Dezember 2013

Oskar Knobel

Reto Lämmler

Inhaltsverzeichnis

1 Einleitung

Der Prozess der Operationsaufklärung verfolgt das Ziel, einem Patienten eine ausreichende Informationsgrundlage zu bieten, damit dieser eine vernünftig begründbare Entscheidung für oder gegen einen bevorstehenden Eingriff fällen kann. Der Arzt arbeitet im Lauf dieses Prozesses zusammen mit dem Patienten ein Formular aus, auf welchem der Eingriff und seine Risiken beschrieben werden. Dabei skizziert er bei Bedarf die Grundzüge des operativen Eingriffs oder betroffener Körperstellen. Erst, wenn dieses Formular von beiden Parteien unterschrieben worden ist, darf der eigentliche Eingriff durchgeführt werden.

An der Klinik für Geburtenhilfe am Universitätsspital Zürich wird dieser Prozess nach wie vor traditionell mit Stift und Papier durchgeführt. Diese Methode ist zweckgemäss, jedoch veraltet und nicht effizient, da für die meisten Eingriffe das Formular mit ähnlichen Inhalten beschrieben werden muss. Aus diesem Grund wurde der Entschluss gefasst, den gesamten Prozess zu modernisieren und mit Hilfe von Windows 8 Tablets umzusetzen.

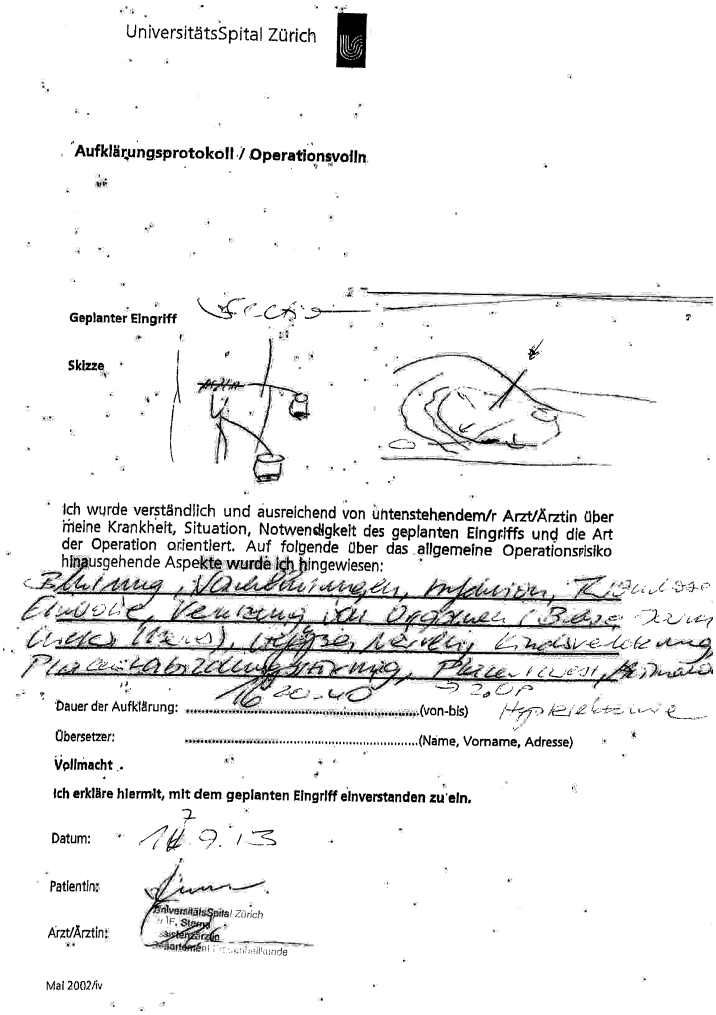
Im Rahmen dieser Bachelorarbeit wurde eine .NET Applikation mit WPF entwickelt, welche diese Modernisierung möglich macht und die Effizienz des Prozesses der Operationsaufklärung steigern soll.

2 Grundlagen

2.1 Ausgangslage und Ziel

2.1.1 Ist-Zustand

In der Klinik für Geburtshilfe im Universitätsspital Zürich wird vor jedem operativen Eingriff ein Aufklärungsgespräch mit der Patientin geführt. Bei diesem Prozess füllt der Arzt ein vorgedrucktes Formular aus, wo er die geplanten Eingriffe und die damit verbundenen Risiken beschreibt. Er erklärt der Patientin mündlich und anhand von Skizzen, wie die Operation ablaufen wird. Im Folgenden wird ein anonymisiertes Aufklärungsprotokoll gezeigt:



UniversitätsSpital Zürich

Aufklärungsprotokoll / Operationsvolln

Geplanter Eingriff

Skizze

Ich wurde verständlich und ausreichend von untenstehendem/r Arzt/Ärztin über meine Krankheit, Situation, Notwendigkeit des geplanten Eingriffs und die Art der Operation orientiert. Auf folgende über das allgemeine Operationsrisiko hinausgehende Aspekte wurde ich hingewiesen:

Dauer der Aufklärung: .....(von-bis)

Übersetzer: .....(Name, Vorname, Adresse)

Vollmacht .

Ich erkläre hiermit, mit dem geplanten Eingriff einverstanden zu ein.

Datum:

Patientin:

Arzt/Ärztin:

Mai 2002/iv

Abbildung 1: Aufklärungsprotokoll Ist-Zustand

Die geplanten Eingriffe und jedes während der Aufklärung angesprochenes Risiko muss auf dem Formular aufgeführt werden. In der Regel gibt es für jeden Operationstyp einen Grundstock an Risiken, welche jedes Mal von Hand aufgeschrieben werden müssen. Weitere Angaben sind „Datum“

und „Dauer der Aufklärung“ sowie ein optionales Feld „Übersetzer“, welche von Hand eingetragen werden.

Falls die Patientin mit den bevorstehenden Eingriffen einverstanden ist, wird am Ende des Prozesses das Formular von den beiden Parteien „Arzt/Ärztin“ und der „Patientin“ unterschrieben. Alternativ kann es auch vorkommen, dass die Patientin nicht einverstanden ist und der Prozess darin resultiert, dass kein Eingriff vorgenommen werden darf.

Mit der Unterschrift wird der Eingriff rechtsgültig autorisiert. Danach muss das Dokument von Hand eingescannt und archiviert werden.

2.1.2 Soll-Zustand

Mit der fertigen App für Operationsaufklärung mit Windows 8 erfolgt die Aufklärung der Patientin digital und multimedial gestützt. Der Arzt nimmt anstelle eines Papiers sein Tablet mit der installierten App ins Zimmer der Patientin mit und führt die Aufklärung interaktiv mit ihr durch.

Der Arzt sucht die Patientin, welche bereits im Klinischen Informationssystem erfasst und entsprechend vormarkiert ist und startet für sie den Aufklärungsprozess, welcher bereits zu diesem Zeitpunkt in Grundzügen personalisiert ist.

In einem ersten Schritt wählt er die gewünschten Operationstypen. Die Risiken der Eingriffe, auf welche die Patientin aufmerksam gemacht werden muss, werden von der App aus den verschiedenen vorliegenden Datenquellen zusammengeführt, als Checkliste bereitgestellt und müssen nur noch in speziellen Fällen manuell ergänzt werden. Bei der Aufklärung können die Risiken so Schritt für Schritt besprochen und abgehakt werden.

Vordefinierte Bilder und Videos stehen bei der Aufklärung einen Fingerwisch entfernt bereit, um der Patientin bei Bedarf mehr vom bevorstehenden Eingriff zu zeigen. Vordefinierte Operationsskizzen können während der Aufklärungssitzung mit dem Stift direkt auf dem Tablet ergänzt und markiert werden. Diese Skizzen werden zusammen mit dem digitalen Formular abgelegt und können jederzeit wieder aufgerufen werden.

Im Hintergrund werden die einzelnen Schritte der Besprechung minutiös protokolliert, damit später genau nachverfolgt werden kann, wie die Sitzung abgelaufen ist. Anfang und Ende der Aufklärungssitzung müssen nicht mehr von Hand erfasst werden, sondern werden automatisch nachgeführt.

Der letzte Schritt des Aufklärungsprozesses ist das eigentliche Formular, welches nun vollständig ausgefüllt angezeigt wird. Der Arzt übergibt das Tablet der Patientin, damit diese das Formular noch einmal anschauen kann. Ist sie mit den besprochenen Inhalten einverstanden, so unterschreibt sie das Formular direkt auf dem Tablet. Sobald auch der Arzt unterschrieben hat, kann das Formular abgeschlossen werden. Es wird dann in der Backend-Datenbank persistiert und ein PDF generiert.

3 Anforderungsanalyse

3.1 Anwendungsfälle

3.1.1 Übersicht über Use Cases

#	Titel	Beschreibung
1	Patientin verwalten (CRUD)	Die Patientin kann aus einer Übersicht von Patientinnen ausgewählt und in einer Detailansicht angezeigt werden. In der Detailansicht können Patientendaten und Schwangerschaftsinformationen angezeigt werden. Zudem wird eine Liste von Patientenrisiken angezeigt, welche bearbeitet werden kann.
2	Risiko verwalten (CRUD)	Ein Risiko kann aus einer Übersicht von Risiken ausgewählt und in einer Detailansicht angezeigt sowie bearbeitet werden. Zudem können Risiken erstellt und gelöscht werden.
3	Aufklärungsprozess durchführen	Bei der Durchführung eines Aufklärungsprozesses wird die betreffende Patientin im System ausgewählt und mittels Medien auf die bevorstehende Operation und die damit verbundenen Risiken aufgeklärt.
4	Aufklärungsprozess abschliessen bzw. unterschreiben	Mit Abschluss und Unterschrift des Formulars wird die Dauer des Aufklärungsprozesses festgehalten und eine druckfertige Version als PDF exportiert.
5	Aufklärungsprozess verwerfen	Wenn die Patientin mit dem Eingriff nicht einverstanden ist und nicht unterschreiben will, kann der Aufklärungsprozess verworfen werden.
6	Multimediainhalt verwalten (CRUD)	Auf dem Formular werden die zum Operationstyp zugehörigen Multimediainhalte verlinkt und können angezeigt werden.
7	Operationstyp verwalten (CRUD)	Ein Operationstyp kann aus einer Übersicht von Operationstypen ausgewählt und in einer Detailansicht angezeigt sowie bearbeitet werden. Zudem können Operationstypen erstellt und gelöscht werden.
8	Medien anzeigen und markieren	Medien können in einer Vollbild Ansicht angezeigt werden. Dabei kann mit einem Stift Skizzen auf dem Medium eingezeichnet und markiert werden.
9	Einstellungen verwalten	In den Einstellungen können Präferenzen zum Web Service, zur Darstellung der Applikationen sowie zur Datenhaltung der Medien vorgenommen werden.

Tabelle 1: Übersicht der Use Cases

3.1.2 Use Case Diagramm

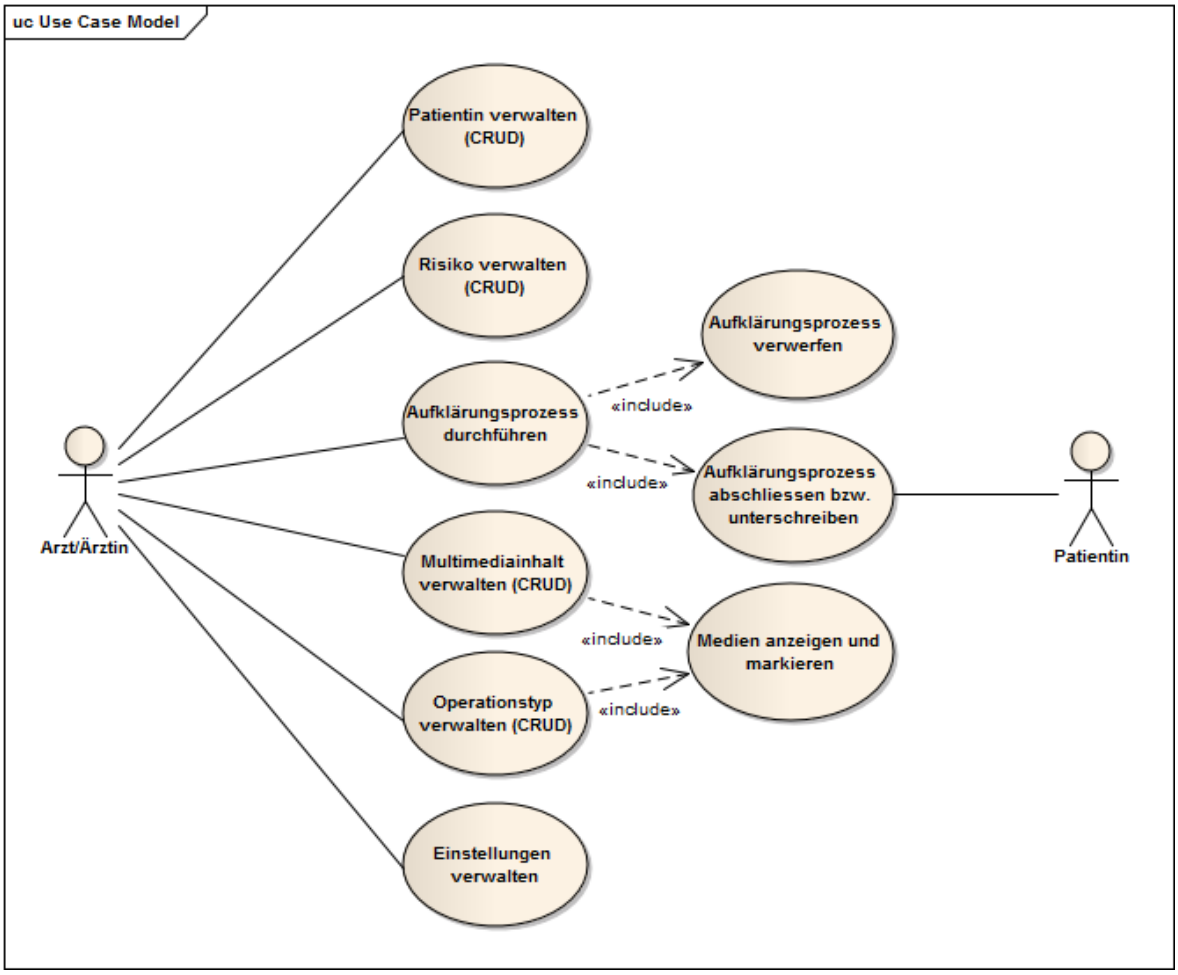


Abbildung 2: Use Case Diagramm

3.1.3 Beschreibungen Fully Dressed

3.1.3.1 Aufklärungsprozess durchführen

Title	Aufklärungsprozess durchführen
Primary Actor	Arzt/Ärztin
Stakeholders and Interests	• Arzt/Ärztin will Aufklärungsprozess mit Patientin durchführen
Preconditions	• Bestehende Verbindung zur Datenquelle • Datenquelle enthält Patientin • Arzt/Ärztin hat Patientin ausgewählt und Aufklärungswizard im System gestartet
Success Guarantees (Postconditions)	• Arzt/Ärztin hat Aufklärungsprozess mit Patientin durchführen können
Main Success Scenario	1. Arzt/Ärztin wählt ein oder mehrere Operationstypen aus 2. Arzt/Ärztin klärt über die mit den Operationstypen verbundenen Risiken auf und hakt diese im System ab. 3. Arzt/Ärztin klärt die Patientin mittel Bilder und Videos über den operativen Eingriff auf. 4. Arzt/Ärztin gibt den Namen des Übersetzers in System ein, falls einer während der Aufklärung anwesend war. 5. Arzt/Ärztin schliesst Wizard ab, worauf ein Dokument zur digitalen Unterschrift bereitgestellt wird.
Frequency of Occurrence:	Häufig

Tabelle 2: UC Aufklärungsprozess durchführen

3.1.3.2 Medien anzeigen und markieren

Title	Medien anzeigen und markieren
Primary Actor	Arzt/Ärztin
Stakeholders and Interests	• Arzt/Ärztin will zu Aufklärungszwecken Medien anzeigen und in diesen Skizzen und Markierungen anbringen.
Preconditions	• Medium ist auf Tablet gespeichert
Success Guarantees (Postconditions)	• Arzt/Ärztin hat Medium angeschaut und allenfalls Markierungen darauf angebracht
Main Success Scenario	1. Arzt/Ärztin öffnet Medium zur Vollbildanzeige 2. Arzt/Ärztin zeichnet Markierungen auf dem Medium ein
Frequency of Occurrence:	Häufig
Special Requirements	Der zum Tablet dazugehörige Stift.

Tabelle 3: UC Medien anzeigen und markieren

3.2 Funktionale Anforderungen

Im Folgenden werden Funktionale Anforderungen aufgelistet, welche aus der Aufgabenstellung sowie Gesprächen mit der Klinik für Geburtshilfe abgeleitet wurden.

3.2.1 Anbindung an KIS/PERINAT

Die App soll später in Echtzeit an das bestehende Klinische Informationssystem (KIS) namens PERINAT angebunden werden können. Von dort werden Patientendaten angeliefert, welche in der App angezeigt werden können, grösstenteils aber schreibgeschützt sind.

Die Patientendaten werden dazu genutzt, um das Aufklärungsformular auf die Patientin zu personalisieren. Nach Abschluss des Formulars wird eine druckbare Version wieder in das KIS exportiert.

3.2.2 Digitale Operationsaufklärung

Die App ermöglicht es den Ärztinnen und Ärzten, den Prozess der Operationsaufklärung digital mit einem Windows 8 Tablet durchzuführen.

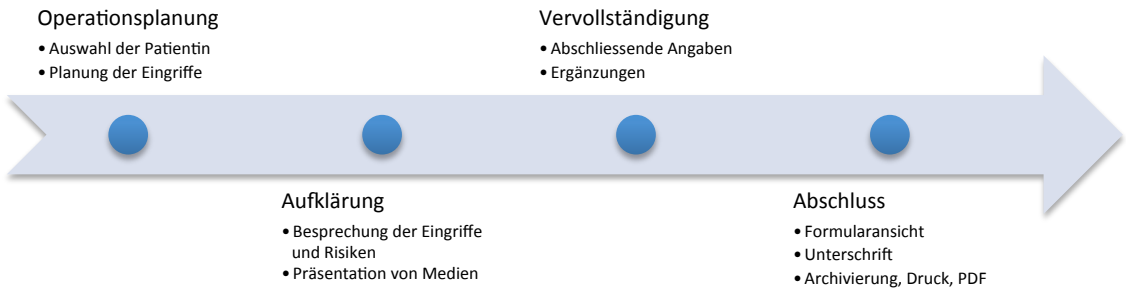


Abbildung 3: Prozess der Operationsaufklärung

Im Rahmen dieses Prozesses werden Arzt und Patientin von der App durch das Aufklärungsgespräch geführt, wobei die relevanten Inhalte jeweils automatisch bereitgestellt werden. Dies fängt bei der Operationsauswahl an, geht über die daraus abgeleitete Risikocheckliste bis hin zu den zugeordneten Medien. Es ist möglich, relevante Videos und Fotos zu präsentieren oder Operationsskizzen interaktiv mit dem Stift zu ergänzen, sollte der Arzt dies für notwendig halten. Am Ende dieses Aufklärungsprozesses steht wieder das ursprüngliche Formular, nun digital auf dem Tablet, vollständig und sauber ausgefüllt. Mit der Unterschrift kann das Formular abgeschlossen werden.

Beim Abschluss des Formulars wird dieses im Backend persistiert sowie ein PDF generiert und abgelegt. Es ist auch möglich, das Formular vor Abschluss zu drucken und von Hand zu unterschreiben.

3.2.3 Verwaltung von Operationstypen und Multimediainhalten

In der App gibt es die Möglichkeit, verschiedene Operationstypen zu wählen, anzuschauen, zu bearbeiten oder neue zu definieren. Ein Operationstyp bildet zusammen mit den Patientendaten die Grundlage für den Aufklärungsprozess.

Der Benutzer hat die Möglichkeit, Informationen zu einem ausgewählten Operationstyp unabhängig von einem konkreten Fall anzuzeigen. Dabei kann er bei Bedarf den Operationstypen mit weiteren Daten anreichern. Dazu gehören:

- Risiken, welche beim Aufklärungsprozess herangezogen werden.
- Multimediainhalte (Fotos und Videos), die beim Aufklärungsprozess gezeigt werden können.
- Zusätzlich gibt es auch die Möglichkeit, Grafiken als Operationsskizzen zur Verfügung zu stellen. Diese können vom Arzt auf dem Formular mit dem Stift ergänzt und markiert werden.

3.3 Nicht-Funktionale Anforderungen

3.3.1 Leistungs- und Effizienzanforderungen

Die App soll schnell auf Benutzereingaben reagieren, um eine angenehme Bedienung sicherzustellen. Wichtig ist dabei, dass der Umgang mit der App so reibungslos und angenehm ist, dass dies einen nennenswerten Mehrwert gegenüber der bisherigen Methode der Operationsaufklärung mit Papier und Stift darstellt.

Mögliche Wartezeiten sollten wie bei mobilen Apps üblich entsprechend deutlich visualisiert werden und der Benutzer sollte niemals den Eindruck erhalten, dass die App abgestürzt ist.

Prozesse, welche nicht direkt im Verlauf des Aufklärungsgesprächs ablaufen, können auch länger dauern, sollten aber nicht als lästig wahrgenommen werden.

3.3.2 Qualitätsanforderungen

Die hier aufgeführten Qualitätsanforderungen orientieren sich grob an den Standards von ISO/IEC 9126, gehen aber weniger ins Detail.

3.3.2.1 Funktionalität

Die beschriebenen Funktionalen Anforderungen müssen komplett umgesetzt werden, damit sich eine Verwendung der App wirklich lohnt.

3.3.2.2 Zuverlässigkeit

Alle im Verlauf der Operationsaufklärung notwendigen Tätigkeiten sollen mit der App abgewickelt werden können. Ärzte sollen möglichst angenehm durch den Prozess geführt und unterstützt, aber nicht eingeschränkt werden. Sollten bei der Bearbeitung des Formulars freie Bemerkungen notwendig sein, so können diese eingefügt werden.

Bezüglich der Stabilität gilt: Eine App, welche den Arzt in seiner Tätigkeit behindert oder sogar abstürzt, wird nicht benutzt. Eingaben, welche im Verlauf der Operationsaufklärung gemacht werden

dürfen auf keinen Fall verloren gehen, da eine Wiederholung des Prozesses auf keinen Fall notwendig werden darf.

3.3.2.3 Effizienz

Die Durchführung des Aufklärungsprozesses soll für Arzt und Patientin so angenehm und reibungslos wie möglich geschehen.

Zu berücksichtigen ist auch noch, dass es sich bei der App primär um eine mobile Applikation handelt, welche auf Tablets ohne permanente Energieversorgung zum Einsatz kommt. Es wird daher angestrebt, den Ressourcenverbrauch in akzeptablem Rahmen zu halten.

3.3.2.4 Änderbarkeit

Die im Rahmen des Projekts erzeugten Apps, Services und Artefakte sollen verständlich umgesetzt und für eine spätere Übernahme bereitgehalten werden. Dazu gehören eine saubere und verständliche Implementierung und entsprechend aktuell gehaltene Dokumentation.

Weiterhin soll bei der Umsetzung der App Wert darauf gelegt werden, dass man sie später bei Bedarf auch noch erweitern könnte.

3.3.2.5 Übertragbarkeit

Die App sollte so einfach wie möglich zu installieren sein. Angestrebt wird ein einfaches Copy Deployment ohne Installer. Die Konfiguration der App sollte nur einmalig notwendig sein und beinhaltet die Festlegung des zu nutzenden Backends.

3.3.2.6 Testbarkeit

Damit die Funktionalität der App vollumfänglich gewährleistet ist, soll die App mittels Unit-Tests getestet werden. So können während der Entwicklung Fehler entdeckt und behoben werden.

3.3.3 Anforderungen an Benutzbarkeit

Die App ist auf die Bedienung mit einem Windows 8 Tablet mit Touchscreen ausgelegt. Die Benutzeroberfläche widerspiegelt diese Tatsache dadurch, dass das Bedienkonzept und die genutzten Steuerelemente sich an gängigen Touch-Standards orientieren, wie man sie bereits von Windows 8 oder Windows Phone kennt.

Anforderungen zu den Reaktionszeiten der Benutzeroberfläche wurden bereits im obigen Abschnitt 3.3.1 zu den Leistungs- und Effizienzanforderungen genannt.

3.3.4 Anforderungen an das Zielsystem

Für den Betrieb der App soll aktuelle und zukünftige Standardhardware mehr als ausreichend sein. Weiterhin soll keine Bindung an einen speziellen Hardwarehersteller eingegangen werden, wodurch Anschaffungs- und Betriebskosten auch langfristig niedrig gehalten werden können.

Zur Ausführung der App ist ein Windows 8 PC oder Tablet notwendig. Die App ist für die Bedienung mit Touchscreen optimiert, erlaubt aber auch die Benutzung ohne entsprechende Hardware. Es ist auch möglich, die App auf Notebooks oder stationären PCs zu benutzen.

Für Datentransfers wird auf WCF Web Services zugegriffen, was eine bestehende Netzwerkverbindung zu diesen voraussetzt. Da die App ausschliesslich im Spital eingesetzt wird, wird davon ausgegangen, dass das Netzwerk mittels Wi-Fi sichergestellt wird.

3.4 External Design

In diesem Abschnitt wird das User Interface der Applikation beschrieben. Aufgrund der Tatsache, dass eine *Desktop App* entwickelt wird (siehe Abschnitt 4.1.1.3), welche annähernd das Aussehen einer Windows 8 App im *Modern Style* haben soll, müssen diverse Überlegungen gemacht werden.

3.4.1 Navigationskonzepte in Windows 8

Damit eine Desktop App auf einem Tablet vernünftig mit Touch bedient werden kann, sollten die Bedienelemente (Buttons, Eingabefelder, …) entsprechend so gross dargestellt werden, dass sie mit dem Finger oder Stift bedient werden können. Microsoft hat zu diesem Zweck eine neue UI Design Language entwickelt, welche für solche Nutzungsszenarien optimiert ist. In diesem Abschnitt werden die Aspekte, welche im Rahmen des Projekts ebenfalls genutzt werden, kurz vorgestellt.

3.4.1.1 Tile-View mit Filterung

In Windows 8 werden viele Datenübersichten in der sogenannten Tile-View dargestellt. Die Datenelemente werden dabei als Kacheln mit beliebigen, eventuell sogar aktiven Inhalten dargestellt. Die Navigation durch die Daten erfolgt horizontal mit Mausrad oder Touch. Darüber hinaus ist es üblich, Möglichkeiten zur Filterung der angezeigten Daten bereitzustellen. Seit Windows 8.1 erfolgt dies zumeist über eine Suchbox oben rechts in der jeweiligen Ansicht. So kann mit wenigen Touch-Inputs schnell das gewünschte Element gefunden werden.

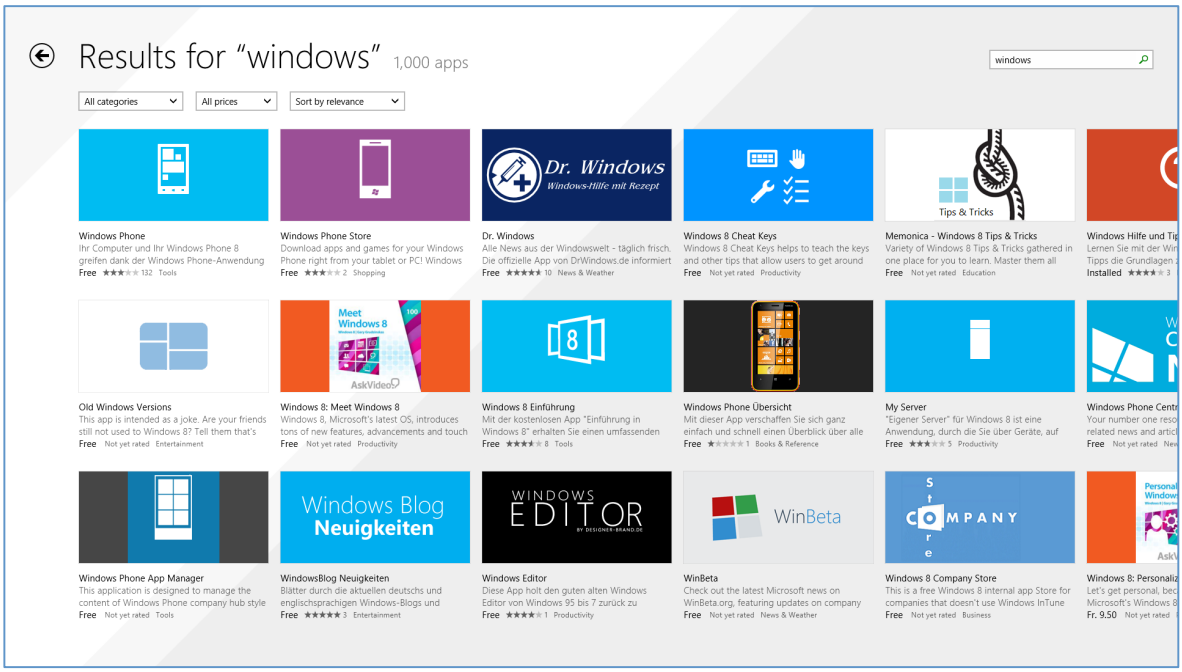


Abbildung 4: Windows 8 Tile-View mit Filterung

3.4.1.2 Panorama

Damit flexibel möglichst viele Informationen angezeigt werden können, werden Elemente in Windows 8 oft mit Hilfe eines so genannten Panoramas angezeigt und dabei in Kategorien, den Panorama Items, gegliedert. Ein anschauliches Beispiel hierfür ist der neue Start Screen von Windows 8, wo die Live Tiles der Apps in Panorama Items organisiert werden können.

Das Panorama bzw. die Panorama Items wachsen dabei vertikal bis die Maximalhöhe erreicht ist und dann horizontal von links nach rechts weiter. Innerhalb des Panoramas kann wieder mittels Mausrad oder Touch horizontal gescrollt werden.

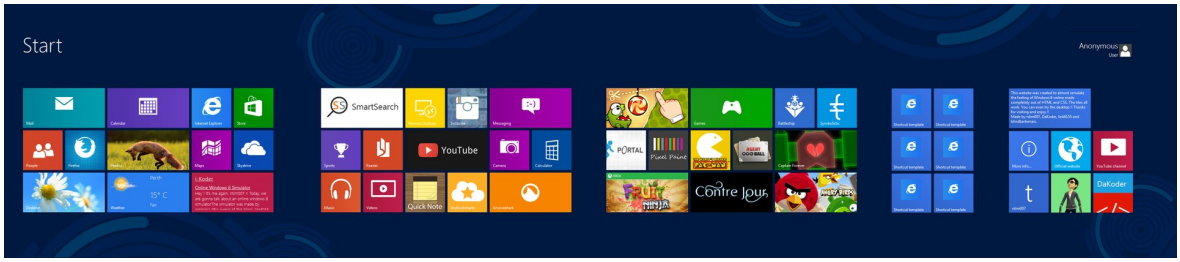


Abbildung 5: Windows 8 Panorama-View

3.4.2 Prototyping

Um die im Abschnitt Navigation (siehe Abschnitt 3.4.1) beschriebenen Konzepte umsetzen, hat das Projektteam einen Paper Prototype entworfen, welcher verschiedene Aspekte der geplanten App umsetzt und auch den Prozess des Aufklärungsgespräches einschliesst. In den folgenden Abschnitten werden die wichtigsten Masken des Prototyps vorgestellt und ihre Evolution durch den Projektverlauf erklärt. Die finale App hat nach eingegangenem Feedback diverse Optimierungen erfahren und sieht daher nicht mehr ganz gleich aus.

3.4.2.1 Start Screen

Auf der Startseite der Applikation sollen die vier Hauptbereiche „patientinnen“, „operationen“, „medien“ und „verwaltung“ möglichst gross als Kacheln dargestellt werden.

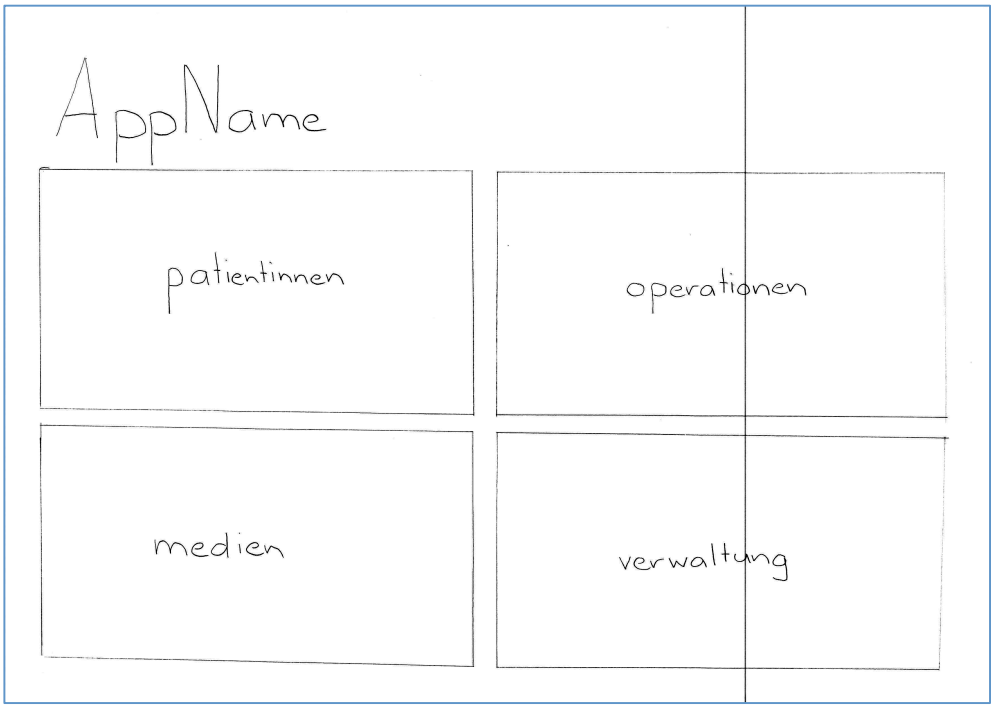


Abbildung 6: Wireframe Startscreen

3.4.2.2 Masteransicht mit Tiles und Filter

Wenn man beispielsweise zur Patientinnen-Ansicht navigiert, werden die Patientinnen wiederum als Kacheln dargestellt. Es besteht die Möglichkeit, die Anzeige mittels Textfilter einzuschränken und somit nach einer bestimmten Patientin zu suchen. Je nach Bedarf könnte dieses Filtersystem um weitere Kriterien erweitert werden.

Dank flexiblem Kachelinhalt ist es auch möglich, die Darstellung der Patientinnen bereits auf dieser Master-Übersicht mit nützlichen Informationen anzureichern. Angedacht war hier beispielsweise, dass man Hochrisikopatientinnen farblich kennzeichnet.

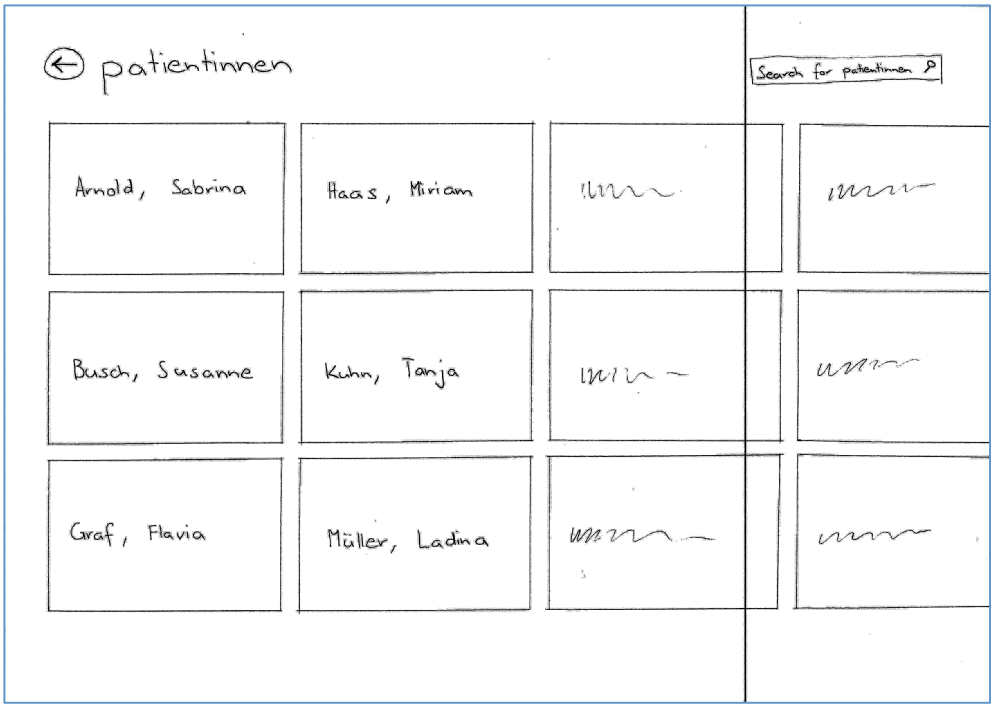


Abbildung 7: Wireframe Masteransicht der Patientinnen

Hat der Benutzer die gewünschte Patientin gefunden, so kann er einfach die entsprechende Kachel betätigen und landet in der Detailansicht (siehe Abschnitt 3.4.2.3).

Das oben dargestellte Wireframe der Patientenübersicht steht als erste Anwendung exemplarisch für diverse Masken, welche nach dem gleichen Schema umgesetzt wurden. So wurde dieses Konzept auch für Operationen, Medien oder Risiken umgesetzt, was in einer einheitlichen, intuitiven und bereits von Windows 8 her bekannten Benutzerführung resultiert.

3.4.2.3 Detailansicht mit Panorama und flexiblem Inhalt

Ausgehend von einer Masteransicht (siehe Abschnitt 3.4.2.2) navigiert der Benutzer in der Regel zur Detailansicht eines Datensatzes, hier gezeigt am Beispiel einer Patientin. Die verschiedenen Informationen werden hier zusammengeführt und dem Benutzer präsentiert. In der ersten Version des Paper Prototypes wurde diese Detailansicht mit einem fixen Layout umgesetzt, worin Listen mit entsprechenden Daten platziert waren (siehe Abbildung 8).

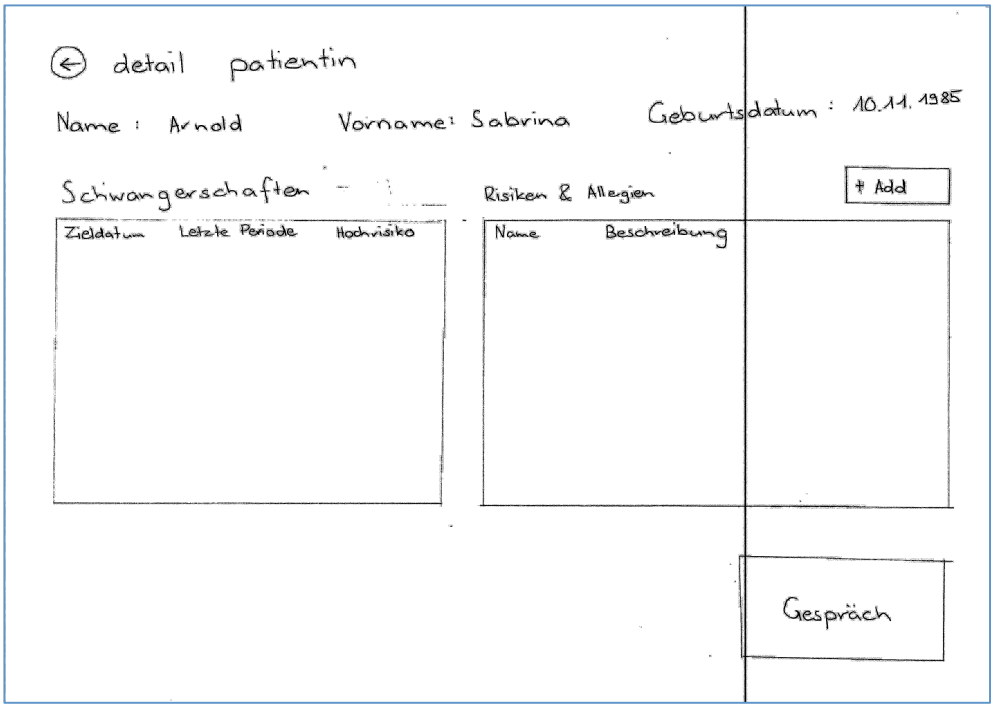


Abbildung 8: Wireframe Detailansicht einer Patientin

Das Feedback zu diesem ersten Entwurf der Detailansicht fiel aus verschiedenen Gründen mehrheitlich kritisch aus:

- Die Touchfunktionalität der Tablets kann so nicht ausgenutzt werden.
- Es werden zu viele nutzlose Informationen auf dem Screen angezeigt.
- Die genutzten Steuerelemente (Listen, Tabellen) fühlen sich in einem Touch-System wie Fremdkörper an.

Aus diesem Grund hat das Projektteam die Detailansicht umgestaltet und dabei anstelle eines fixen Layouts auf flexible Layouts mit Panoramas (siehe Abschnitt 3.4.1.2) anvisiert. Sämtliche kritisierten Punkte konnten damit behandelt werden, insbesondere auch die Frage der Informationsdichte.

3.4.2.4 Aufklärungsassistent

Der Hauptanwendungsfall der App ist die Operationsaufklärung. Der schlussendlich umgesetzte Assistent für den Aufklärungsprozess ist das Resultat eines längeren Designprozesses, an dessen Anfang das ursprünglich genutzte Formular stand. Zu Beginn war die Vorstellung, dass dieses Formular schlicht als interaktives Dokument direkt auf dem Tablet umgesetzt werden sollte.

Bereits beim Entwurf des Papier Prototypen stellte sich jedoch heraus, dass dieser Ansatz nicht brauchbar ist, da wiederum die Informationsdichte zu hoch und die Bedienung mittels Touch unbrauchbar sein würde. Nach Absprache mit allen beteiligten Parteien hat man sich schlussendlich auf die Umsetzung mittels eines Assistenten geeinigt, an dessen Ende das Formular wieder angezeigt wird.

Der erste Schritt im Prozess ist die Auswahl der geplanten Eingriffe, welche analog zur Masteransicht (siehe Abschnitt 3.4.2.2) wieder mit Kacheln dargestellt wird. Im Unterschied zur Masteransicht

erfolgt bei Betätigung der Kacheln jedoch kein Drilldown zur Detailansicht, sondern eine Selektion des entsprechenden Eingriffs.

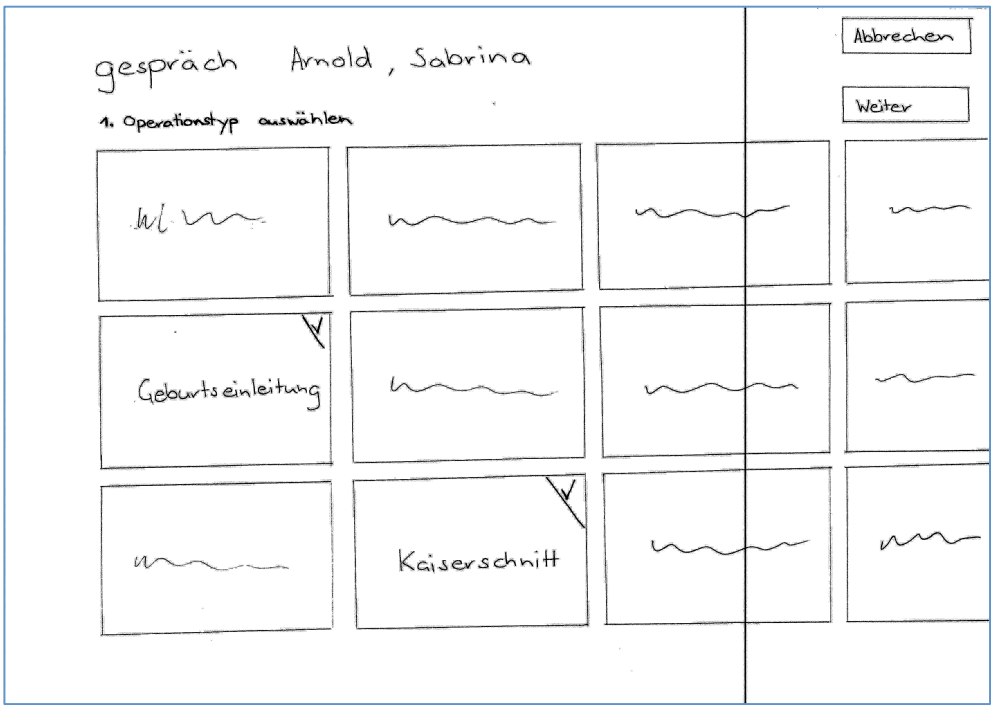


Abbildung 9: Wireframe Operationstyp auswählen

Mit der Auswahl der Eingriffe ist es der App möglich, die Risikocheckliste aus den Operationstypen und der Patientenvorgeschichte herzuleiten. Darüber hinaus stehen nun entsprechende Medien zur Verfügung. Alle diese Daten werden auf einem breiten Panorama für die Besprechung zusammengeführt.

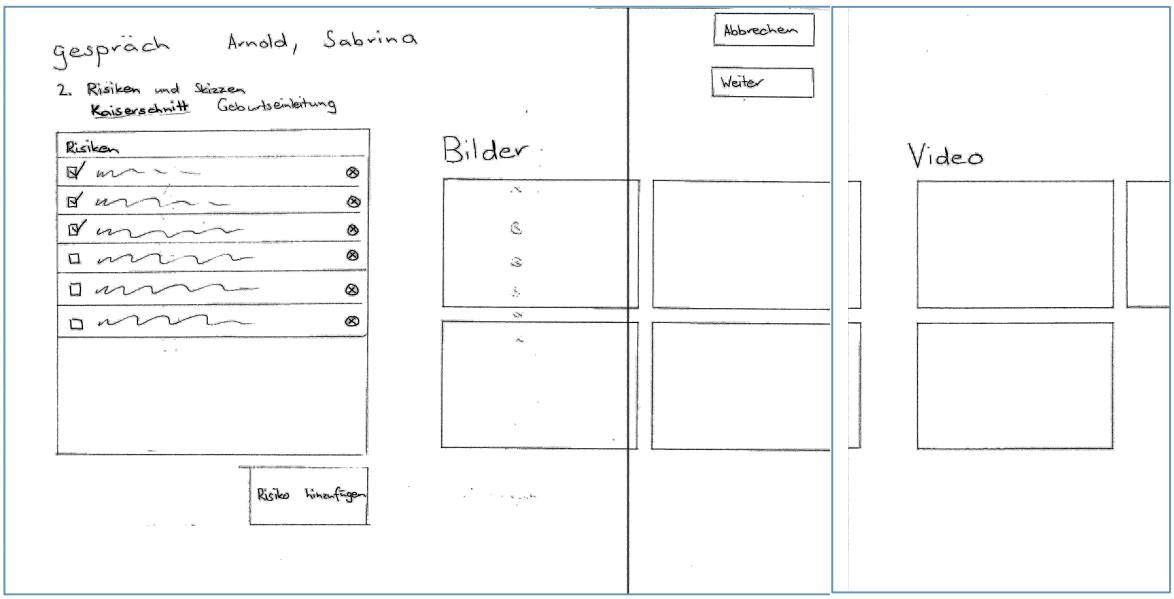


Abbildung 10: Wireframe Risiken und Skizzen

Die Medien und Skizzen für die Besprechung werden wiederum mit Kacheln direkt im Panorama angezeigt und können durch einfache Betätigung aufgerufen werden. Es öffnet sich dabei ein Overlay, wo die entsprechenden Inhalte angezeigt werden.

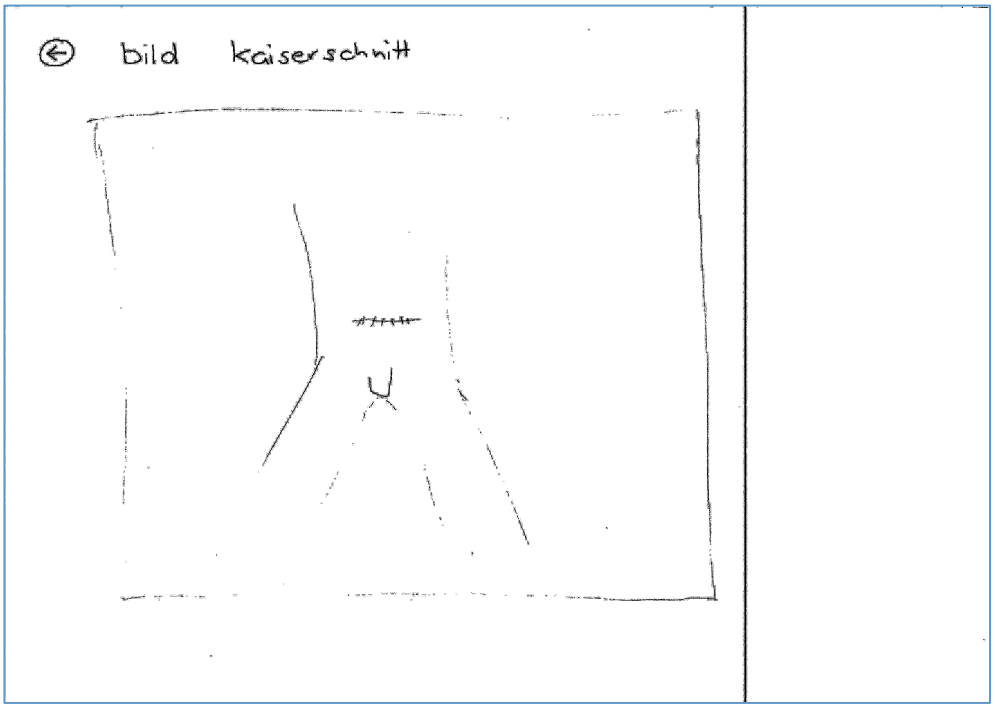


Abbildung 11: Wireframe Skizze

Nach Abschluss des Gesprächs führt der Assistent weiter auf eine Maske, auf welcher abschliessende Angaben zum Aufklärungsprozess ergänzt werden können, wie etwa dem Übersetzer.

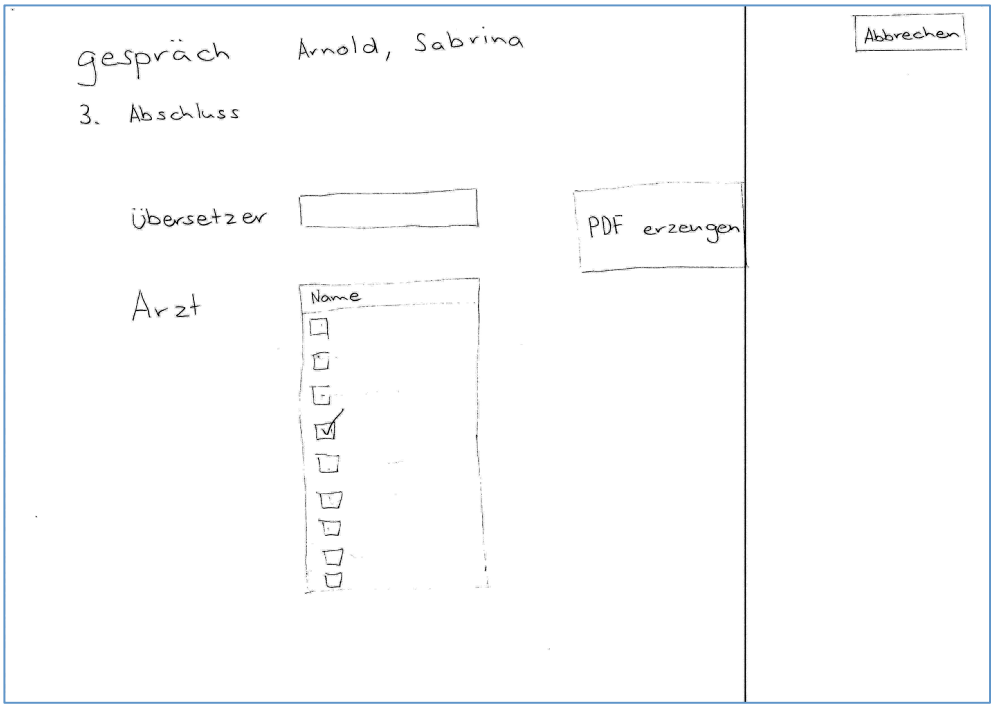


Abbildung 12: Wireframe Abschluss Wizard

In der ersten Version des Paper Prototypes war angedacht, dass der Arzt ebenfalls auf dieser Abschlussseite ausgewählt werden kann. Es hat sich jedoch herausgestellt, dass dies den Prozessfluss eher behindert und im Prinzip von vornherein klar sein sollte, welcher Arzt die App gerade bedient. Bei der späteren Implementierung wurde deshalb ein globaler Login implementiert.

Nach Abschluss dieses Schritts wird das voll parametrisierte Formular angezeigt, welches dann unterschrieben und so abgeschlossen werden kann.

4 Software Architektur

4.1 Technischer Hintergrund

An dieser Stelle wird kurz auf einige grundsätzliche Überlegungen und Entscheidungen zur Software Architektur eingegangen. Für eine detailliertere Beschreibung der angesprochenen Aspekte sei jeweils auf die entsprechenden Abschnitte verwiesen.

4.1.1 Wahl der Entwicklungsplattform

Auf Windows 8 und Windows RT stehen zwei grundsätzlich verschiedene Plattformen für die Entwicklung von Applikationen zur Auswahl. Mit Applikation gemeint ist hier vor allem derjenige Teil, mit welchem die Benutzer später direkt interagieren – also das Frontend. Das Backend selbst fällt gar nicht unter diese Diskussion, da dieses nochmals einem anderen Entwicklungsmodell folgt, welches aber in Windows 8 und Windows RT nicht entscheidend überarbeitet wurde.

Das in diesem Abschnitt behandelte Thema gehört im Prinzip zur Anforderungsanalyse (siehe Abschnitt 3) oder zumindest in den Bereich Research und Evaluation von Technologie. Der Grund weshalb es dennoch im Rahmen der Softwarearchitektur besprochen wird ist, dass die Wahl der Entwicklungsplattform weitreichende Konsequenzen auf die Architektur der App hat und zu Beginn des Projekts zu längeren Diskussionen geführt hat.

4.1.1.1 Desktop App oder Windows Store App

In diesem Abschnitt werden kurz die beiden zuvor erwähnten Entwicklungsmodelle und –Plattformen vorgestellt. Für eine detailliertere Beschreibung derselben sei auf das *Windows Whitepaper* (siehe Appendix Windows Whitepaper) verwiesen. Um die Diskussion besser zu veranschaulichen wird nochmals auf das dort bereits enthaltene Architekturdiagramm (Turner, 2012) zurückgegriffen.

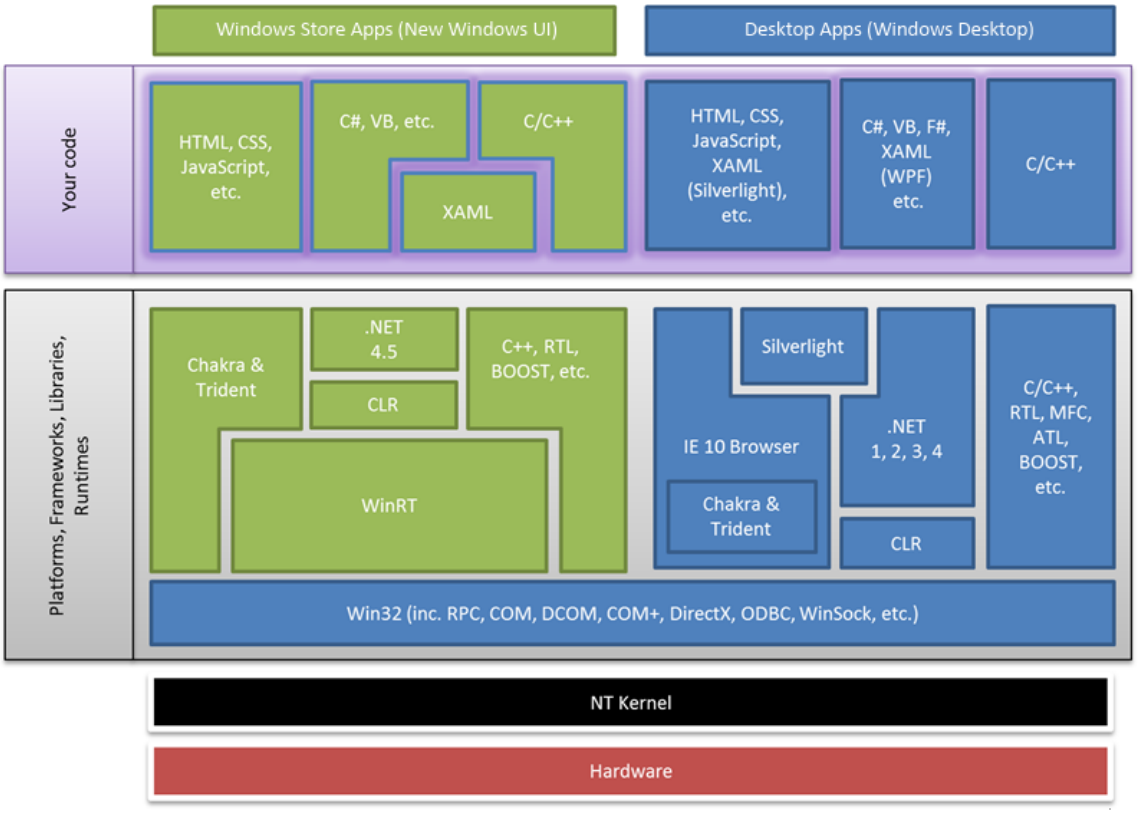


Abbildung 13: Windows 8 Architekturdiagramm von Rich Turner

Windows 8 und Windows RT erlauben die Ausführung zweier Arten von Apps bzw. Applikationen:

- *Desktop Apps* können nur auf dem *Windows Desktop* ausgeführt werden. *Desktop Apps* können nur für Windows 8, nicht aber für Windows RT entwickelt werden.
- *Windows Store Apps* können nur im *New Windows UI* ausgeführt werden. *Windows Store Apps* können für Windows 8 und Windows RT entwickelt werden.

Die beiden Arten von Apps unterscheiden sich in der Entwicklung grundlegend:

- Entwickelt man eine *Windows Store App*, so ist die neue Windows Runtime (WinRT) als Ganzes verfügbar, aber es kann nur ein Subset vom .NET Framework genutzt werden. Dieses Subset ist in etwa vergleichbar mit demjenigen von Silverlight für Windows Phone.
- Entwickelt man eine *Desktop App*, so ist das .NET Framework als Ganzes verfügbar, aber es kann nur ein Subset von WinRT genutzt werden. Zusätzlich zum ganzen .NET Framework können aber im Prinzip beliebige Softwarekomponenten genutzt werden, was eine enorme Flexibilität ermöglicht, da im traditionellen Umfeld von Windows Programmbibliotheken für fast alle erdenklichen Zwecke vorhanden sind.

Die Festlegung auf eine der beiden Plattformen stellt einen Grundsatzentscheid dar und hat grosse Auswirkungen auf die spätere Entwicklung und das Endresultat des Projekts, da es keine einfache Möglichkeit für einen späteren Wechsel gibt und die Kompatibilität dazwischen eingeschränkt ist.

4.1.1.2 Analyse der Anforderungen

Eine Entscheidung für oder gegen eines der beiden Modelle muss auf Basis der Anforderungen an das Projekt (siehe Abschnitt 3) gefällt werden, da diese von den Modellen jeweils unterschiedlich gut abgedeckt werden. In diesem Abschnitt wird kurz auf die Anforderungen eingegangen, welche für die Auswahl einer Plattform wichtig sind.

Bei den Funktionalen Anforderungen lässt sich die grosse Mehrheit der Wünsche problemlos umsetzen, doch gibt es gewisse Aspekte, welche im Rahmen einer *Windows Store App* schwierig sind. Der Grund hierfür liegt oft in sicherheitstechnischen Bedenken und dem daraus abgeleiteten Sandboxing der Apps. Die Verwendung gemeinsamer Dateien ist dabei auf ein absolutes Minimum eingeschränkt.

Anforderung	**Desktop App**	**Windows Store App**
Abbildung Formular in App	Problemlos.	Problematisch, da Windows Runtime FlowDocument nicht brauchbar unterstützt. Es muss eine Alternative gesucht oder von Hand gebaut werden.
Dateisystemzugriff für Ablage und Archivierung	Problemlos.	Zugriff ist nur auf isolierte Dateisystembereiche erlaubt. Zugriff auf Netzwerkshares ist nicht möglich. **Unmöglich**.
PDF-Generierung aus Formular	Problemlos.	Problematisch, mit Zusatzkosten verbunden.
Pen & Digital Ink für Markierung und Unterschrift	Problemlos.	Problematisch. Pen muss separat angeschafft werden und die Ink-Unterstützung ist schlecht.
Verwaltung von Multimediainhalten	Problemlos.	Eingeschränkt. Siehe auch Dateisystemzugriff. **Unmöglich**.

Tabelle 4: Funktionale Anforderungen und Umsetzbarkeit in Entwicklungsmodellen

Bei den Nichtfunktionalen Anforderungen sind die Einschränkungen von *Windows Store Apps* ebenfalls sehr problematisch und in ihren Auswirkungen noch härter zu spüren.

Anforderung	**Desktop App**	**Windows Store App**
Direktzugriff auf Backenddatenbank analog zu PERINAT	Problemlos.	EF und andere ORMs können nicht genutzt werden, da nur ein .NET Subset verfügbar ist. **Unmöglich**.
In-House Deployment und	Problemlos (von IT gehandhabt).	Problematisch, da mit massiven

Wartung		Zusatzkosten verbunden und lizenztechnisch fragwürdig. Windows Store Deployment und Zertifizierung zeitaufwendig und fragwürdig. **Unmöglich**.
Touch-First Bedienung	Mit Zusatzaufwand und evtl. Zusatzkosten verbunden, da externe Komponenten zugezogen werden müssen.	Problemlos.
Verwendung von Standardhardware	Problemlos. Es werden keine Windows RT Geräte unterstützt.	Problemlos. Alle Geräte können genutzt werden.

Tabelle 5: Nichtfunktionale Anforderungen und Entwicklungsmodell

Insbesondere der Punkt für In-House Deployment und Wartung ist kritisch, da dieser im Prinzip die Kontrolle des Kunden über das Deployment sicherstellt. Die Alternative zum In-House Deployment ist ein Deployment über den Windows Store, wo für jede Änderung an der App eine tagelange Zertifizierungs- und Testprozedur von Microsoft angestossen werden muss.

Für eine detaillierte Aufstellung der Anforderungen im Rahmen dieses Projekts sei auf die *Anforderungsanalyse* verwiesen.

4.1.1.3 Entscheidung für den Windows Desktop

Ausgehend von den vorhergehenden Ausführungen wurde die Entscheidung hin zu einer traditionellen *Desktop App* und gegen eine *Windows Store App* gefällt, da nur so die gewünschte Funktionalität optimal bzw. überhaupt umgesetzt werden kann.

Diese Entscheidung hat weitreichende Implikationen auf das Programmiermodell der App und den dazugehörigen Services. Diese Unterschiede werden im Rahmen dieses Dokuments beschrieben, so weit dies vom Projektteam als notwendig erachtet wird.

4.1.2 Architektonische Ziele

4.1.2.1 Erweiterbarkeit & Übertragbarkeit

Aus der Aufgabenstellung und den ersten Gesprächen mit dem Industriepartner an der Klinik für Geburtshilfe wurde schnell klar, dass es sich bei der zu entwickelnden App nur um eine erste Version bzw. einen Prototypen handelt, welcher später weiterentwickelt oder umgebaut werden sollte. Aus diesem Grund sollte die Gesamtarchitektur so sein, dass im Hinblick auf zukünftige Änderungswünsche folgende qualitative Kriterien erfüllt sind:

- Das Gesamtprojekt soll zum Übergabezeitpunkt einen sauberen, gut dokumentierten Zustand aufweisen, damit die neuen Entwickler sich so gut wie möglich zu Recht finden. Dies schliesst ein gut refaktorisiertes Design der Klassen- und Projektstruktur ein.
- Um spätere Änderungen so einfach wie möglich zu machen sollen Extension- und Variation-Points so offen wie möglich gehalten werden. Durch die Entkoppelung der einzelnen

Komponenten soll sichergestellt werden, dass diese ausgetauscht oder erweitert werden können.
- Soweit möglich soll auf frei verfügbare, weit verbreitete Technologien anstelle von proprietären Lösungen gesetzt werden.

4.1.2.2 Benutzbarkeit

Die Benutzbarkeit der App stellt einen Spagat zwischen zwei Welten dar (siehe Abschnitt 4.1.1.1). Auf der einen Seite befindet sich der Benutzer auf der Desktop-Umgebung und wird mit einer verhältnismässig grossen Datenmenge konfrontiert. Auf der anderen Seite soll die App touch-optimiert sein und sich beim Design an Windows Store Apps orientieren. Diese beiden Ziele stellen in gewisser Weise einen Zielkonflikt dar. Es muss daher ausbalanciert werden, welches Ziel im jeweiligen Anwendungsfall wichtiger ist.

- Bei Masken welche CRUD Use Cases umsetzen, stösst der Modern Style an seine Grenzen aufgrund der Datenmengen. Für die App heisst dies, dass die Masken eventuell reduziert werden müssen, damit nur die tatsächlich relevanten Daten erscheinen.
- Eingabeschemen mit Maus, Stylus und Touch müssen überall vorgesehen sein. Es dürfen keine Annahmen über den allfälligen Anschluss einer Tastatur gemacht werden.
- Die Steuerelemente müssen für die Touchbedienung gross genug sein.
- Es dürfen keine Annahmen zur Bildschirmauflösung, Bildschirmgrösse oder Orientierung der Tablets bei der Benutzung gemacht werden. Zum Zeitpunkt der Entwicklung liegen gängige Auflösungen zwischen 1366x768 und 1920x1080 Pixel bei 10“ Bildschirmdiagonale, jeweils mit Portrait und Landscape Orientierung.

4.1.2.3 Effizienz & Skalierbarkeit

Die Gesamtarchitektur soll keine einschränkenden Annahmen machen zur späteren Verwendung und Skalierung der App. Aus diesem Grund muss mit Ressourcen bewusst umgegangen werden. Weiterhin müssen beim Design folgende Gedanken eine Rolle spielen:

- Die Leistung der Tablets ist heute für eine solche Applikation mehr als ausreichend. Der limitierende Faktor ist der Energiebedarf bei der Ausführung.
- Für das Backend gilt, dass man nicht einschätzen kann, wie viele Instanzen der App später produktiv in Betrieb sein werden. Die Architektur sollte also skalierbar sein.
- Beim Datenaustausch muss davon ausgegangen werden, dass die Datenmenge kontinuierlich wächst. Die Service-Schnittstelle muss die Übertragung einer beliebigen Datenmenge unterstützen, z.B. mittels Streaming oder Paging (siehe Abschnitte 4.4.2.3 oder 4.4.2.4).

4.1.3 Einschränkungen

4.1.3.1 Systemanforderungen

Die im Rahmen des Projekts entwickelte App nutzt Features, welche erst ab Windows 8 verfügbar sind. Obwohl die zugrundeliegende Technologie mit Windows Presentation Foundation also hauptsächlich auch auf älteren Systemen wie Windows 7 problemlos lauffähig ist, ist Windows 8 bzw. Windows 8.1 als minimale Systemanforderung für das Frontend gegeben. Darüber hinaus muss nochmals die Tatsache betont werden, dass die Verwendung der Windows Presentation Foundation

und die Entwicklung einer Windows Desktop App einen Einsatz der App auf Windows RT verunmöglicht (siehe Abschnitt 4.1.1.1). Zusatzmodule, welche an das Backend entwickelt werden können, unterliegen nicht denselben Restriktionen.

Der hier beschriebene Umstand ist im Besonderen auch für die Mitarbeiter am Universitätsspital Zürich von Bedeutung, da diese zum Zeitpunkt der Projektarbeit mehrheitlich noch mit Windows XP arbeiten. Die Bereitstellung entsprechender Tablets und PCs mit Windows 8 oder neuerem Betriebssystem ist noch nicht gesichert und muss von den Mitarbeitern der Klinik für Geburtshilfe separat angegangen werden.

4.1.3.2 Entkoppelung von KIS/PERINAT

Die Patientendaten des Unispitals sind in einem eigenen klinischen Informationssystem und zugehörigen Datenbanksystem namens PERINAT gespeichert. Aufgrund des fehlenden Wissens über dieses Datenbanksystem und die genauen Anbindungsparameter, sowie zeitlichen Überlegungen hat das Projektteam mit dem Industriepartner vereinbart, dass ein eigenes Datenbanksystem entwickelt wird. Erst nach Abschluss des Projektes wird dann von der IT Abteilung des Universitätsspitals Zürich PERINAT mit den vom Projektteam im Rahmen des Projekts implementierten Web Services verbunden.

4.2 Systemaufbau

4.2.1 System-Übersicht

Das System kann primär in zwei separate, entkoppelte Bereiche unterteilt werden. Es handelt sich dabei um einen Client-Teil und einen Server-Teil. Die Frontend App repräsentiert dabei den Client; der WCF Service und die Microsoft SQL Server Datenbanken den Server.

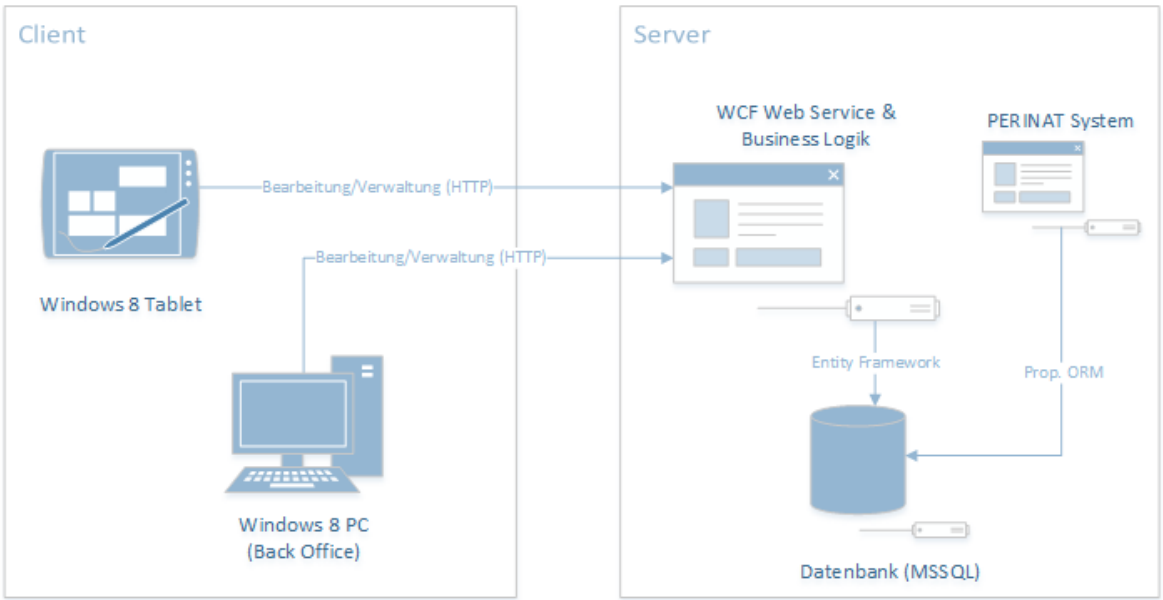


Abbildung 14: Deploymentdiagramm

Zu Testzwecken ist es auch möglich, sämtliche Teile der App auf demselben Gerät zu installieren, indem man Datenbank & Web Service, sowie auch die App selbst auf einem einzigen PC installiert und entsprechend konfiguriert.

Für jeden Bereich wurden in der Solution separate Projekte angelegt die mehr oder weniger unabhängig von einander entwickelt werden konnten, da von Anfang an auf ein entkoppeltes Design geachtet wurde. Die Hauptschnittstellen zwischen diesen Projekten werden in Abschnitt 4.2.6 besprochen. Eine Übersicht über die Assemblies wird in Abschnitt 4.3.5.1 gegeben.

4.2.2 Frontend App (FlexApp)

Die Frontend App dient dazu den Aufklärungsprozess der Klinik für Geburtshilfe für die Ärztinnen und Ärzte, sowie auch für die Patientinnen übersichtlich und unterstützend zu gestalten. Zum einen werden die vom Web Service bereitgestellten Daten den Benutzern klar strukturiert angezeigt und im Rahmen des Aufklärungsprozesses entsprechend aufbereitet. Zum anderen werden die gemachten Eingaben über den Web Service wieder in das System eingeleitet und verarbeitet.

Im Frontend sind zwei primäre Benutzungsvarianten angedacht:

1. Benutzung in der Operationsaufklärung mit Tablets.
2. Benutzung im Back Office, wo die Datenverwaltung gemacht werden kann. So werden die Operationen und Medien in der Regel vor den Gesprächen entsprechend parametrisiert.

4.2.3 Backend Service (BusinessService)

Der Backend Service setzt die benötigte Logik um, damit der Datenaustausch und die Kommunikation zwischen dem Frontend und der Datenbank transparent und reibungslos ablaufen kann. Dazu stellt der Service eine WCF Schnittstelle für das Frontend bereit.

4.2.4 Kommunikation (DataTransferInterface)

Die Kommunikation zwischen Client und Server erfolgt über WCF. Die hierfür benötigte Schnittstelle wird über eine gemeinsam genutzte Library zur Verfügung gestellt.

4.2.5 Datenbanken (Intern und PERINAT)

Auf der Datenbank werden die für den Aufklärungsprozess benötigten Daten konsistent gehalten und bereitgestellt. Diese werden dann von der Businesskomponente des Backend Services mittels Entity Framework abgeholt und für die App aufbereitet und bereitgestellt.

Zu Beginn der Entwicklung und bis nach der weiteren Parametrisierung und Modifikation der App durch Mitarbeiter vom Universitätsspital Zürich werden App-Datenbank und das klinische Informationssystem PERINAT separat existieren. Die Verbindung zwischen den beiden Datenbanken wird vom Projektpartner umgesetzt und getestet.

4.2.6 Hauptschnittstellen

Aus dem groben Systemaufbau, sowie dem Layering und den Erläuterungen dazu (siehe Abschnitt 4.3.1) werden zwei Hauptschnittstellen ersichtlich.

4.2.6.1 Schnittstelle zwischen Frontend App und Backend

Die Frontend App kommuniziert via HTTP mit dem Backend Service, der einen WCF Service Contract implementiert. Diese Kommunikation findet zumeist asynchron statt; es kann zu Konstellationen kommen, bei welchen mehrere App Instanzen gleichzeitig mit dem Backend arbeiten.

4.2.6.2 Schnittstelle zwischen Backend und Datenbank

Zwischen dem Backend Service und der internen Datenbank findet selbst wieder eine traditionelle Client/Server-Kommunikation statt, wobei der Backend Service den Client und die Microsoft SQL Server Datenbank den Server darstellt.

Die Datenbank befindet sich nicht unbedingt auf dem gleichen Host wie der Backend Service, wenn auch dies aus Performanceüberlegungen sinnvoll wäre.

4.2.6.3 Schnittstelle zwischen Datenbank und PERINAT

Zwischen der internen Datenbank und dem klinischen Informationssystem PERINAT wird zu einem späteren Zeitpunkt eine Verknüpfung aktiviert. Wie genau diese Verknüpfung umgesetzt wird, ist zum Projektende noch nicht definiert. Es sind verschiedene Varianten denkbar:

- Verknüpfung auf Basis von SQL Server, evtl. mit Triggern, Views und Procedures.
- Verknüpfung mit periodischem Dateiaustausch, z.B. CSV-I/O.
- Verknüpfung über erweiterte Web Services.

Die Umsetzung dieser Schnittstelle wird in der vorliegenden Arbeit nicht behandelt.

4.3 Komponentenübersicht

4.3.1 Layerdiagramm

Das gesamte Projekt ist in vier Layer unterteilt, welche sich auf Client und Server gemäss ihrer Funktionalität verteilen:

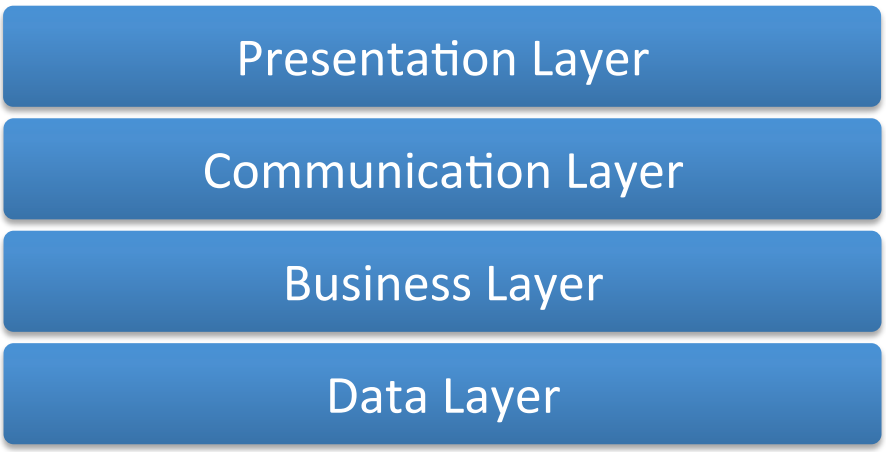


Abbildung 15: Layerdiagramm

Die Inhalte der einzelnen Layer lassen sich wie folgt zusammenfassen:

Schicht	Beschreibung	Hauptpackages
Presentation Layer	Die oberste Schicht ist das Presentation Layer und beinhaltet alle Komponenten, welche dem Benutzer angezeigt werden und mit denen interagiert werden kann.	FlexApp (4.3.2.1), ModernStyleComponents (4.3.2.5), WindowsRuntimeSupport (4.3.2.7)
Communication Layer	Das Communication Layer ist eine logische Schicht zwischen Client und Server und beinhaltet den WCF Service Contract. Dieser ist von der Service Implementierung abgetrennt und entkoppelt so Client und Server voneinander.	DataTransferInterface (4.3.2.4)
Business Layer	Im Business Layer sind alle inhaltlichen Aspekte abgehandelt. Die Kernkomponente der Applikation, welche die Backend-Funktionalität zur Verfügung stellt ist hier implementiert.	BusinessService (4.3.2.2)
Data Layer	Das Data Layer beinhaltet die Komponenten für den Datenbankzugriff.	DataPersistenceModels (4.3.2.3)

Tabelle 6: Beschreibung der Layer

4.3.2 Packages

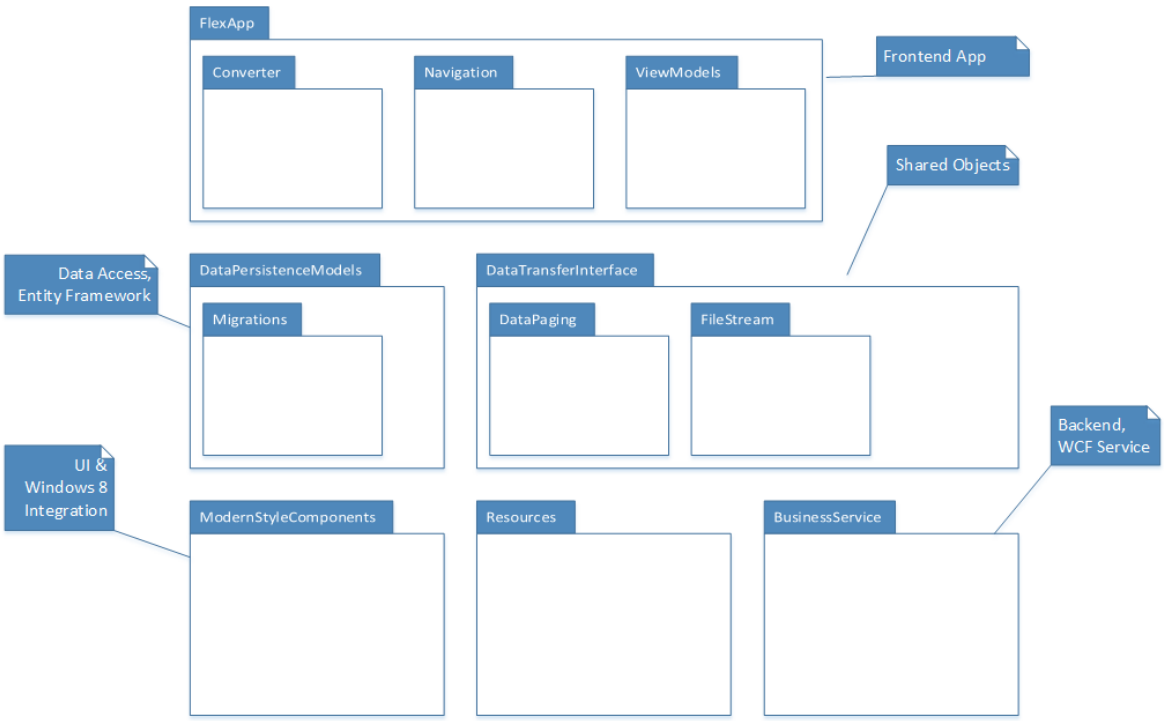


Abbildung 16: Packagestruktur

4.3.2.1 FlexApp Packages

Das *FlexApp* Package beinhaltet alle für die App relevanten Klassen und Ressourcen. Die App-Klasse und globale XAML-Ressourcen sind Bestandteil, wie auch der primäre App Container. Darüber hinaus sind noch weitere Bestandteile enthalten. Diese sind in drei Subpackages eingeteilt:

- Das *Converter* Subpackage enthält in der App genutzte Konverter. Konverter sind Funktionen, welche innerhalb der Bindings im UI genutzt werden, um Werte in Domain-Klassen für die Anzeige beliebig flexibel aufzubereiten.
- Das *Navigation* Subpackage enthält die in hauptsächlich XAML codierte Benutzeroberfläche der App. Die App folgt einem Navigationsmodell, welches auf Seiten und nicht auf Fenstern aufbaut. Der Hauptcontainer der App zeigt dabei wie ein Browser die Seiten an, welche gerade benötigt werden.
- Das *ViewModels* Subpackage enthält die View Models der App. Die im *Navigation* Subpackage definierten Views verwenden mittels MVVM (siehe Abschnitt 4.4.1.1) zur Verfügung gestellte Funktionalität. Die in diesem Subpackage implementierten Klassen bilden die Domain-Klassen aus dem DataTransferInterface Package (siehe Abschnitt 4.3.2.4) auf die hierfür nutzbare View Models ab und stellen passende Kommandos für die App bereit.

4.3.2.2 BusinessService Package

Im *BusinessService* Package ist der Windows Communication Foundation Service des App Backends implementiert.

4.3.2.3 DataPersistenceModels Package

Im *DataPersistenceModels* Package sind die service-seitig benötigten Klassen für die Persistenz der Daten enthalten. Dabei handelt es sich hauptsächlich um ORM Klassen, mit welchen der Zugriff auf die Datenquelle durchgeführt wird, sowie um eine zusammenfassende ORM Kontext Klasse.

Aufgrund der Verwendung von Entity Framework 6.0 mit Code First Ansatz gibt es noch ein Subpackage Migrations, welches Klassen für die Datenbank Migration und Seeds enthält.

4.3.2.4 DataTransferInterface Packages

Im *DataTransferInterface* Package enthalten sind Objekte, welchen allen Komponenten gemeinsam sind. Dazu gehören primär die Schnittstelle des Backend Services, welche als WCF Service Contract definiert ist.

Zu dieser Schnittstelle gehören auch die Domain-Objekte, welche für die App relevante Entitäten verknüpfen und für Transaktionen auf dem Service zur Verfügung stellen. Bei den Domain-Objekten ist die Mehrheit der Klassen ähnlich zu denen aus dem DataPersistenceModels Package (siehe Abschnitt 4.3.2.3), jedoch nicht exakt gleich. Es handelt sich um eine Anwendung des *Data Transfer Object* Anti-Patterns.

Unterstützende Komponenten sind in Subpackages enthalten:

- Das *DataPaging* Subpackage enthält die Klassen, welche für die Umsetzung der Paging- und Suchfunktionen gebraucht werden (siehe Abschnitt 4.4.2.3).
- Das *FileStream* Subpackage enthält die Klassen, welche für die Umsetzung des File Streaming gebraucht werden (siehe Abschnitt 4.4.2.4).

4.3.2.5 ModernStyleComponents Package

Das *ModernStyleComponents* Package beinhaltet spezielle UI Komponenten, welche in der App gebraucht werden. Diese Komponenten gliedern sich nahtlos in das bestehende System der Windows Presentation Foundation ein.

Zu den nennenswerten UI Komponenten gehören:

- Die SignatureBox Komponente, welche über das von der Windows Presentation Foundation bereitgestellte InkCanvas ein Eingabefeld für Unterschriften bereitstellt.
- Das Panorama Control, welches die für Windows 8 und Windows Phone typische Panorama-Navigation (siehe dazu *Windows Whitepaper*) innerhalb einer App ermöglicht (siehe Abschnitt 4.4.6).

Darüber hinaus sind auch diverse unterstützende Klassen in diesem Package definiert.

4.3.2.6 Resources Package

Das *Resources* Package enthält diverse in der App genutzte Ressourcen. Dabei handelt es sich um in XAML definierte Styles und Objekte, aber auch um statische Medien und die für die Internationalisierung benötigten String-Tabellen, welche als RESX abgelegt sind.

4.3.2.7 WindowsRuntimeSupport Package

Das *WindowsRuntimeSupport* Package beinhaltet Windows 8 bzw. Windows Runtime spezifische Klassen, welche in der App gebraucht werden.

Zu den nennenswerten Features dieses Packages gehören:

- Live Tile und App Shortcut Unterstützung (siehe Abschnitt 4.4.9.1).
- Toast Service und Schnittstelle (siehe Abschnitt 4.4.9.2).

4.3.3 Domainmodell

Das Domainmodell wurde auf Basis der für das Formular benötigten Daten aufgebaut. Der obere Bereich (Patient, Schwangerschaft) stellt die Patientendaten dar. Die Daten der Operationstypen sowie die dazugehörigen Operations- und Patientenrisiken werden im mittleren Bereich (Risiko, Operationstyp) aufgezeigt. Im rechten Bereich (Media, Foto, Video, Skizze, Markup) werden die Multimediadaten dargestellt. Das Formular sowie die Arztdaten sind im unteren Bereich erfasst.

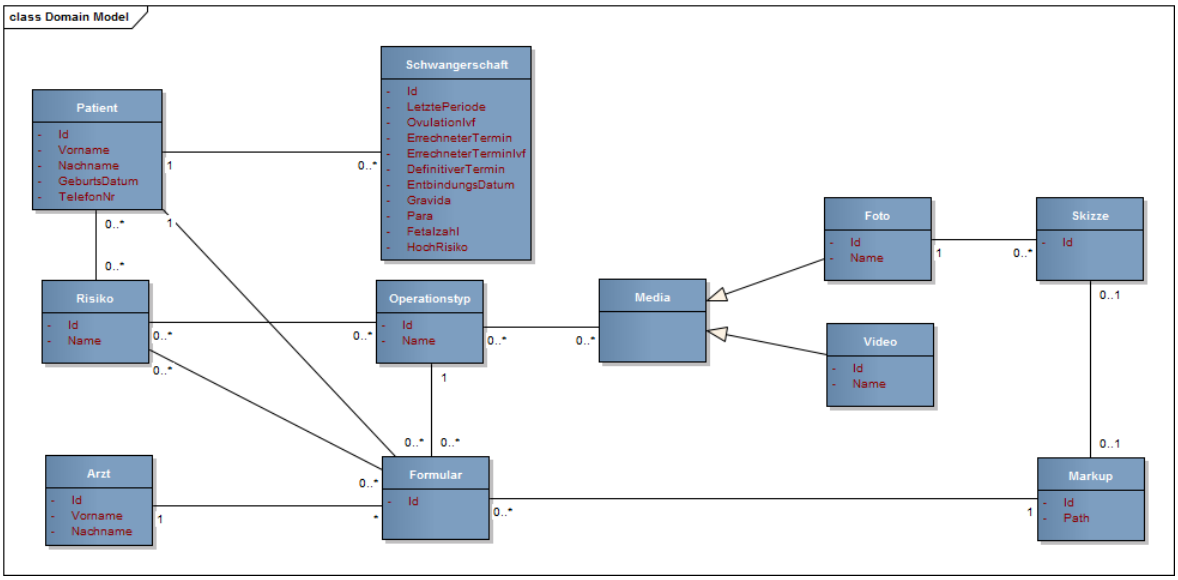


Abbildung 17: Domain Model

4.3.4 Datenhaltung

Für die Datenhaltung wird ein Microsoft SQL Server Express oder besser genutzt. Unterstützt werden alle Versionen ab 2008, wobei bei der Entwicklung zumeist Version 2012 genutzt wurde. Die Anbindung des Backends an die Datenbank erfolgt dabei über das Entity Framework (siehe Abschnitt 4.3.5.2) und nicht über das LePhone Framework (Liang, 2011), wie es an der Klinik für Geburtshilfe bei PERINAT genutzt wird.

4.3.4.1 Schnittstelle

Als Schnittstelle zur eigentlichen Datenbank fungiert wie oben beschrieben das Entity Framework, welches zu Beginn des Projekts auf Version 6.0 aktualisiert wurde. Mit diesem neuen Release im September 2013 haben leider einige neue Bugs und undokumentierte Breaking Changes ins Entity Framework Einzug gehalten. Dazu gehören:

- Undokumentierte Umstellung der Art und Weise, wie eine Datenbank initialisiert werden kann.
- Attribute für die Datenbank-seitige Berechnung von Columns funktionieren nicht mehr oder nicht wie erwartet; betrifft insbesondere auch Felder vom Datentyp DateTime, welche nicht mehr automatisch berechnet werden.

Grund für diese Probleme ist, dass das Entwicklungsteam hinter dem Entity Framework seine Lösung portabler hat gestalten wollen, wofür leider einige SQL-Server spezifische Funktionen entfernt werden mussten (Microsoft Open Technologies, Inc., 2013). Für den Grossteil dieser Probleme hat das Projektteam brauchbare Workarounds finden können.

4.3.4.2 Entity Framework Model

Zur Modellierung der Datenbank wurde vom Projektteam der Code-First Ansatz gewählt, da dies den Migrationsprozess mit existierenden Daten vereinfachen kann, da dieser in der Regel explizit ausprogrammiert wird (Microsoft Corp., 2013). Dies bedeutet, dass die benötigten Entitäten zuerst als .NET Klasse realisiert wurden, welche selbst anstelle eines eigentlichen Modells oder Schemas zur Erzeugung der Datenbank genutzt werden.

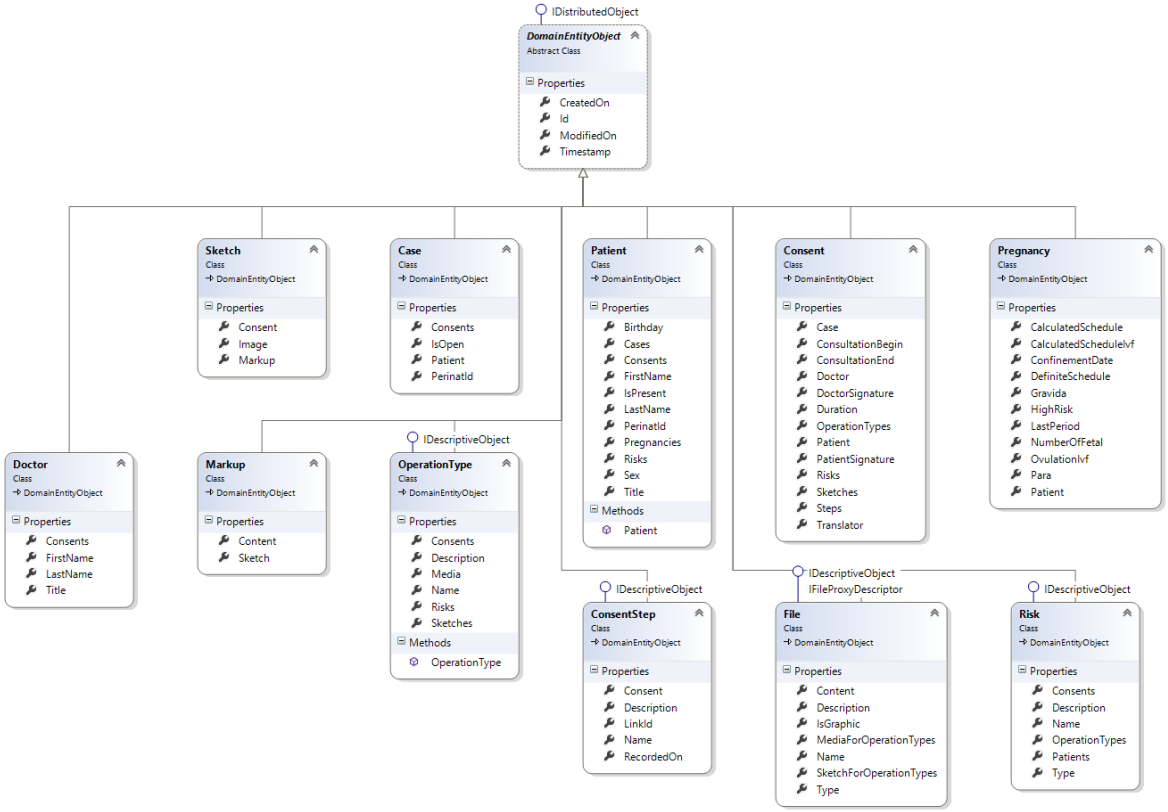


Abbildung 18: Klassendiagramm der Entitäten

Es ist durchaus wahrscheinlich, dass zu einem späteren Zeitpunkt vom Ansatz Code-First Abstand genommen werden soll. Die Umstellung von Code-First auf die Erzeugung des Codes für die Entitäten von einer existierenden Datenbank wäre mit Aufwand verbunden, aber nicht unmöglich, da die Entitäten selbst über keinerlei Logik implementieren und somit eigentlich nur neu erzeugt werden müssen. Sämtliche Funktionen, welche mit den Entitäten arbeiten wurden als Extensions realisiert und sind von einer allfälligen Neuerzeugung nicht betroffen.

4.3.4.3 Struktur der Datenbank

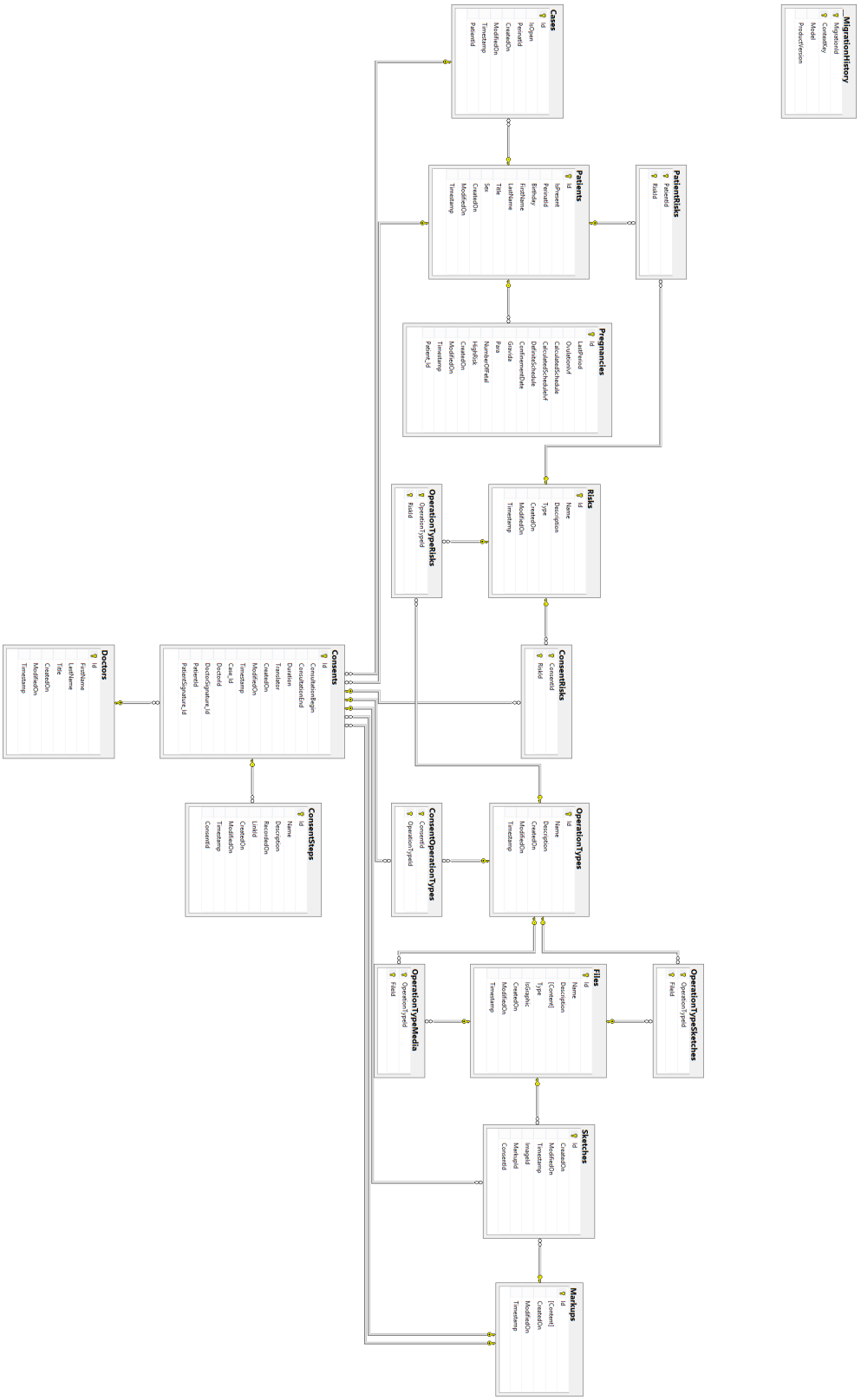


Abbildung 19: Tabellen in der Datenbank

4.3.5 Assemblies

4.3.5.1 Aufteilung in Assemblies

Dieser Abschnitt beschreibt die im Rahmen des Projekts entwickelten Assemblies.

Assembly	Beschreibung
Eyecon.BusinessService	Enthält die Implementierung des Backend WCF Services und der Datenbankanbindung.
Eyecon.BusinessService.Testing	Enthält Tests für die Business-Logik im Service.
Eyecon.DataTransferInterface	Enthält die Definition der verteilten Objekte, welche zwischen Backend und Frontend ausgetauscht werden. Enthält auch die Definition der Service Schnittstelle, welche im BusinessService implementiert wird.
Eyecon.DataTransferInterface.Testing	Enthält Tests für die verteilten Objekte.
Eyecon.FlexApp	Enthält die Frontend App.
Eyecon.FlexApp.Testing	Enthält Tests für Komponenten der Frontend App.
Eyecon.ModernStyleComponents	Enthält UI Komponenten für die Frontend App und Schnittstellen für die Integration mit Windows 8 Diensten wie Toasts.

Abbildung 20: Assemblies in der Solution

4.3.5.2 Externe Abhängigkeiten

Dieser Abschnitt beschreibt externe Abhängigkeiten des Projekts. Dabei handelt es sich vor allem um Programmierbibliotheken, welche für die Umsetzung spezieller Funktionalität eingebunden werden.

Abhängigkeit	Beschreibung	Einbindungsart
Entity Framework http://entityframework.codeplex.com/	"Entity Framework (EF) is an object-relational mapper that enables .NET developers to work with relational data using domain-specific objects." (Microsoft Open Technologies, Inc., 2013). Genutzt für das Design und die Anbindung an die Datenbank.	NuGet Package in Visual Studio
MahApps.Metro http://mahapps.com/MahApps.Metro/	"MahApps.Metro strives to make good looking metro interfaces easier" (Jenkins & Ginnivan, 2012). Genutzt für die erweiterte Umsetzung der App im Modern Style (siehe Abschnitt 4.4.4.2).	NuGet Package in Visual Studio
ModernUI for WPF http://mui.codeplex.com/	"A set of controls and styles converting your WPF application into a great looking Modern UI app." (Zwikstra, 2013). Genutzt für die primäre Umsetzung der App im Modern Style, sowie für das allgemeine	NuGet Package in Visual Studio

	Navigationsmodell der App (siehe Abschnitt 4.4.4.1).	
Ninject http://www.ninject.org/	"Ninject helps you use the technique of dependency injection to break your applications into loosely-coupled, highly-cohesive components, and then glue them back together in a flexible manner." (Enkari, Ltd., 2012) – Genutzt zur Umsetzung des Locator Services (siehe Abschnitt 4.4.1.3).	NuGet Package in Visual Studio
PDFSharp http://www.pdfsharp.net/	„PDFsharp is the Open Source .NET library that easily creates and processes PDF documents on the fly from any .NET language." (empira Software GmbH, 2009). Genutzt für die Umwandlung von XPS in PDF.	Einbindung einer Referenz auf DLL im Projekt
Extended WPF Toolkit Community Edition http://wpftoolkit.codeplex.com/	"Extended WPF Toolkit™ is the number one collection of WPF controls, components and utilities for creating next generation Windows applications." (Xceed, 2013) – Genutzt für die Bereitstellung spezieller Controls.	NuGet Package in Visual Studio
Windows API Code Pack	"Windows API Code Pack for Microsoft. NET Framework provides a source code library that can be used to access some features of Windows 7 and Windows Vista from managed code." (Khandelwal, 2011) – Genutzt für die Integration mit Windows 8.	NuGet Package in Visual Studio, Einbindung einer Referenz auf DLL in Projekt

Tabelle 7: Externe Abhängigkeiten

4.3.6 Unit Testing

Aus dem Systemaufbau nicht sofort ersichtlich ist die Existenz von Testprojekten, welche zu den jeweiligen Bereichen gehören. Die Testprojekte können am besten nach dem beschriebenen Layering (4.3.1) organisiert werden, was sich auch in der Solution so wiederspiegelt.

4.3.6.1 Presentation Layer Tests

Das systematische Testing im Presentation Layer deckt drei Hauptbereiche ab und betrifft praktisch ausschliesslich die Frontend App. Es handelt sich bei den Tests um:

- View Model Tests
- UI Komponenten Tests
- Konverter Tests

Die Unit Tests für die Konverter im Frontend sind trivial, da sie zumeist nur eine sehr lokale Funktionalität testen und keine Infrastruktur voraussetzen. Es handelt sich um einfache Wertumwandlungen, deren Resultate mit statischen Soll-Werten verglichen werden.

Eine komplexere Situation stellen die View Model Tests dar, da diese in vielen Fällen über interne Zustandswechsel ihr Verhalten anpassen und zugleich eine gewisse Infrastruktur voraussetzen. Dazu

gehört eine Unterstützung durch den Dispatcher von WPF, ohne welchen bereits die Grundkomponenten wie Commands in den Tests versagen. Ziel der View Model Tests ist es, die im UI interaktiv verursachten Zustände (also durch Benutzereingaben) der verschiedenen View Models systematisch zu erzeugen und die Commands darauf zu Testen. Hierdurch wird die Funktionalität *hinter* dem XAML getestet.

Schlussendlich gibt es noch einige UI Komponenten Tests. Diese Tests betrachten jeweils eine sehr spezifische Funktion eines Controls oder einer anderen Komponente.

4.3.6.2 Communication Layer Tests

Bei den Tests für das Communication Layer handelt es sich hauptsächlich um Tests für die Data Transfer Objects und die Beziehung unter ihnen. Diese Tests sind wichtig, da Fehler im Communication Layer zu einer falschen Datenverarbeitung führen können, aber meist erst sehr spät auffallen. Auch die Infrastruktur – z.B. die Funktion für das automatische Deep Cloning von Data Contracts – wird systematisch getestet.

4.3.6.3 Business Layer Tests

Die Tests im Business Layer beschäftigen sich mit den Businesstransaktionen und Service-Funktionen, welche vom Backend angeboten werden (siehe Abschnitt 4.4.2.1). Da diese Tests sich auf die darunterliegende Datenbank auswirken, greifen diese oft auch direkt darauf zu, wodurch das Layering ein wenig verwischt wird.

4.4 Konzepte und Vorgehensweisen

4.4.1 Allgemeine Konzepte

4.4.1.1 Model-View-ViewModel Pattern

Sowohl in der Windows Presentation Foundation, als auch der Windows Runtime werden Benutzeroberflächen mittels XAML Markup definiert. Ziel ist es, Benutzeroberfläche und Programmlogik so weit wie möglich voneinander zu entkoppeln. Hierfür wird oft ein Architekturpattern namens Model-View-ViewModel, kurz MVVM, verwendet. MVVM ist eine Variante des bekannten Model-View-Controller Patterns zur Trennung von Markup und Logik.

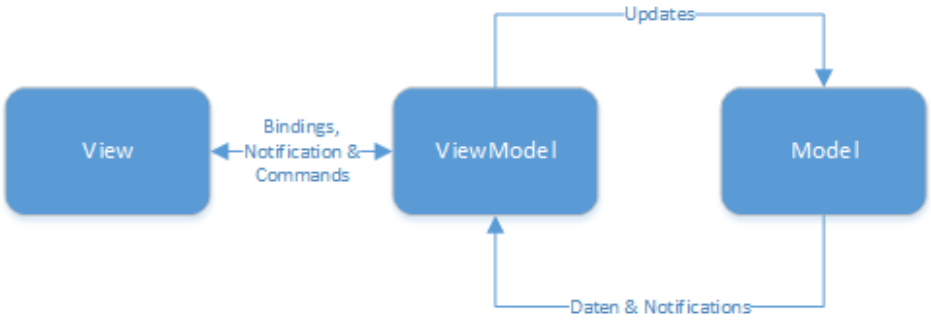


Abbildung 21: Model-View-ViewModel Pattern

Die drei Bestandteile innerhalb des Patterns haben eine klar definierte Bedeutung:

- Die View repräsentiert die Benutzeroberfläche oder zumindest einen Teil davon. In der Regel wird die View mit XAML Markups definiert, welche Bindings auf die Properties des ViewModels enthalten.
- Das ViewModel ist das Verbindungsstück zwischen View und Model. Es bietet der View alle benötigten Informationen über Properties an, welche mittels Bindings referenziert werden können. Darüber hinaus nimmt das ViewModel den Input der View entgegen und führt Operationen auf dem Model aus.
- Das Model ist die Datenquelle eines ViewModels.

Im Rahmen des Projekts wurde versucht, die Verwendung von Code-Behind bei der Umsetzung der gewünschten Features auf ein Minimum zu reduzieren. Code, welcher Logik der App implementiert wurde soweit möglich in Commands von ViewModels ausgelagert; Code welcher sich um die Darstellung der Inhalte kümmert soweit wie möglich in XAML umgesetzt oder in entkoppelte Komponenten ausgelagert, wenn dies sinnvoll erschien.

4.4.1.2 ViewModel Basisklassen

Zur systematischen Gruppierung und Entwicklung der ViewModels wurde eine Reihe von Basisklassen im Projekt eingeführt. Diese Klassen befinden sich im ViewModels Subpackage der App (siehe Abschnitt 4.3.2.1):

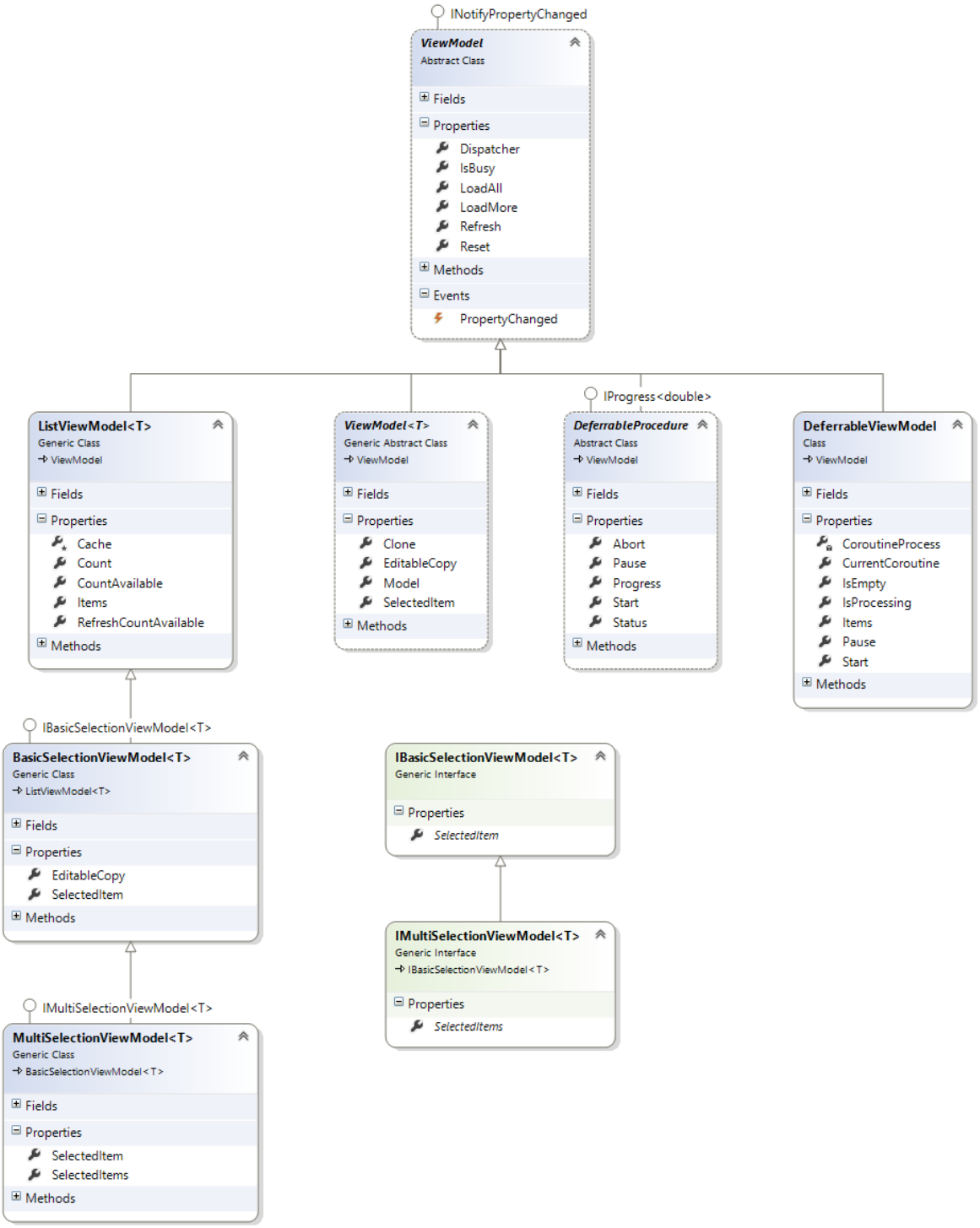


Abbildung 22: View Model Basisklassen

Diese Basisklassen für View Models bieten eine wiederverwendbare Infrastruktur für grob drei verschiedene Nutzungsszenarien:

Klassen	Kategorie	Beschreibung
ViewModel, ViewModel<T>	Allgemeine View Models und View Models für Detailansichten	Die Klasse ViewModel dient allen anderen View Models als Basisklasse. Die Spezialisierung mit dem generischen Parameter T ist Basisklasse der View Models mit Model vom Typ T, wie sie oft auf Detailansichten gebraucht werden.
ListViewModel<T>, BasicSelectionViewModel<T>, MultiSelectionViewModel<T>	View Models für Listen und Masteransichten, View Models mit Selektionsfeatures	Diese Basisklassen implementieren View Models, welche mit Listen von Models vom Typ T arbeiten. Je nach Wunsch wird auch eine einfache oder mehrfache Selektion von Models unterstützt.
DeferrableViewModel, DeferrableProcedure	View Models für globale, nebenläufige Prozesse	Die Klasse DeferrableViewModel implementiert eine logische Queue, welche Objekte vom Typ DeferrableProcedure entgegennimmt und sequentiell verarbeitet. Eine DeferrableProcedure ist eine Art nebenläufiger Task mit zusätzlicher Infrastruktur für Status- und Fortschrittsmeldungen.

Tabelle 8: View Model Basisklassen

4.4.1.3 ViewModel Locator und LocatorService

Mit der Komplexität des Projekts wächst auch die Anzahl der Views und View Models stetig an, bis zu einem Punkt, wo entweder der Überblick verloren geht, oder die Koppelung zwischen den verschiedenen Objekten zu gross wird, als dass sie vom Entwickler sinnvoll gehandhabt werden kann. Gängige Ansätze zur Lösung dieses Problems sind Dependency Injection oder Inversion of Control.

Zusätzlich zu diesem Umstand kommt erschwerend hinzu, dass die Verwendung des Navigationsmodells von *Modern UI for WPF* (siehe Abschnitt 4.4.4.1) diverse Einschränkungen bei der Übergabe von Parametern bei der Navigation zwischen verschiedenen Pages mit sich bringt, da diese nicht über Objekte, sondern mittels URIs adressiert werden. Das bedeutet, dass zwar zu einer Page navigiert werden kann, bei der Navigation aber höchstens ein URI Fragment spezifiziert werden kann, was für die Übergabe eines komplexen Objektes nicht ausreicht.

Im Rahmen des Projekts wurde daher ein entsprechende Klasse LocatorService mit Hilfe von Ninject (siehe Abschnitt 4.3.5.2) realisiert, welcher den Umgang mit der Masse an Views und View Models etwas erleichtert und zugleich die Übergabe der relevanten Daten an die jeweils genutzten View Models übernimmt.

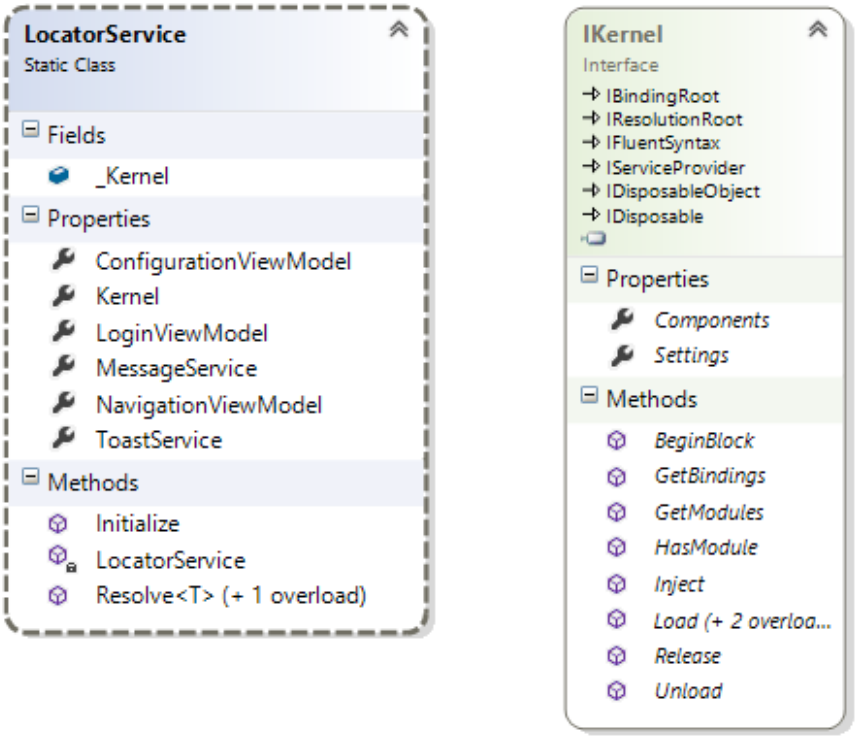


Abbildung 23: LocatorService mit IKernel von Ninject

Der LocatorService bietet mit der generischen Methode Resolve eine einfache Schnittstelle, um beliebige View Model Bindings zur Laufzeit auflösen zu können. Intern wird die Verwaltung der View Models und Services über die Schnittstelle IKernel von Ninject vorgenommen. Es besteht die Möglichkeit, zur Laufzeit neue Bindings zu definieren oder existierende Bindings zu ersetzen.

Die Bedeutung des Wortes Bindings ist hier eine andere als bei WPF. Es handelt sich um eine zur Laufzeit vorgenommene Zuordnung einer Schnittstelle zu einer Instanz einer Klasse. So wurde beispielsweise ein Toast Service realisiert (siehe Abschnitt 4.4.9.2) in Form der Klasse ToastService, welche die Schnittstelle IToastService implementiert. Innerhalb der App wird stets gegen IToastService entwickelt, um die Komponenten zu entkoppeln, wobei der LocatorService genutzt wird, um die eigentliche Instanz aufzulösen (da sie so oft genutzt wird, wurde ein statisches Property ToastService in der Klasse definiert, welches allerdings dasselbe tut):

```
LocatorService.Resolve<IToastService>().Show(
    string.Format("{0} ({1})", Strings.FileSynchronizationToastFoundFilesMessage, Syncing),
    Strings.FileSynchronizationToastFoundFilesToastTitle);
```

Der Grund, weshalb dies funktioniert ist, dass für die App beim Start eine Reihe von Standardbindings spezifiziert wird. Jede View, welche ein View Model oder einen Service benötigt, kann sich diese Objekte über den LocatorService auflösen lassen. In der folgenden Tabelle werden einige der wichtigsten Bindings aufgelistet:

Kategorie	Schnittstelle	Zuordnung
Service & Maintenance	IEyeconService	Methodenaufruf zur Konfiguration eines Service Proxys (siehe Abschnitte 4.2.3, 4.3.2.4 und 4.4.2)

	ConfigurationViewModel	ConfigurationViewModel als Singleton
Navigation	NavigationViewModel	NavigationViewModel als Singleton (siehe Abschnitt 4.4.5.1)
Operation	OperationMasterViewModel	OperationMasterViewModel als Singleton
	OperationDetailViewModel	Konstruktor von OperationDetailViewModel, welchem die aktuelle Selektion des OperationMasterViewModels mitgegeben wird
Patienten	PatientMasterViewModel	PatientMasterViewModel als Singleton
	PatientDetailViewModel	Konstruktor von PatientDetailViewModel, welchem die aktuelle Selektion des PatientMasterViewModels mitgegeben wird
UI Services	IMessageService	MessageService als Singleton (siehe Abschnitt 4.4.7)
	IMultiSelectionService<File>	MediaDialog
	IMultiSelectionService<Risk>	RiskDialog
	IToastService	ToastService Konstruktor mit aktueller APP_ID (siehe Abschnitt 4.4.9.2)

Tabelle 9: Auszug aus den LocatorService Bindings

In der obigen Tabelle fällt auf, dass die MasterViewModels bei der Zuordnung als Singleton aufgeführt werden. Singleton bezieht sich hierbei auf die Existenz innerhalb des Ninject Kernels, welcher zur Verwaltung genutzt wird. Es wird also bei jedem Aufruf von Resolve das gleiche View Model zurückgeliefert. Dies hat zwei positive Effekte:

1. Auf den Master-Ansichten wird jeweils dasselbe View Model genutzt. Auch wenn zwischen den Pages hin und her navigiert wird, bleibt deren Zustand im View Model erhalten.
2. Beim Drilldown vom Master- auf die Detail-Ansicht kann die Detail-Ansicht über den LocatorService das passende DetailViewModel anfordern, welches automatisch richtig initialisiert wurde (mit der Selektion des MasterViewModels).

Somit wird das bereits erwähnte Problem der Übergabe von Parametern zwischen den Pages (siehe Abschnitt 4.4.5.4) elegant und zentral gelöst. Selbstverständlich steht es den Views trotzdem frei, ihre eigenen View Models zu instanziieren, sofern dies Sinn macht.

Die Einführung des LocatorServices erlaubt auch die vereinfachte Verwendung und Zuordnungen von DataContext Objekten im Designer und zur Laufzeit der App, indem diese sogar per XAML assoziiert werden können.

```
DataContext="{x:Static view:LocatorService.LoginViewModel}"
```

Somit kann in vielen Fällen auch der Code-behind für das Setzen des DataContexts entfernt werden.

4.4.1.4 Asynchrone Tasks mit Await und Async Pattern

Mit dem .NET Framework 4.0 wurde die Unterstützung für nebenläufige Entwicklung neu betrachtet und ein neues, task-basiertes Modell in Form der Task Parallel Library (TPL) eingeführt (Bray, 2012). Ziel war es, die Responsiveness von .NET Applikationen zu verbessern, indem man die Entwickler mehr dazu motiviert, sich mit asynchroner Ausführung von Code zu beschäftigen. Mit .NET 4.5 wurde eine sprachliche Unterstützung dieser Features in C# eingeführt, wo neu mit den Schlüsselwörtern *await* und *async* Asynchronität sehr einfach modelliert werden kann. Darüber hinaus wurden mit dem neuen Release diverse Neuerungen im Framework eingeflochten, um asynchrone Methodenaufrufe bei altbekannten Klassen zu erlauben.

Um von dieser neuen Infrastruktur profitieren zu können, wurde in vielen View Models von diesem neuen Await-Async Pattern Gebrauch gemacht, was besonders auch bei der Kommunikation mit dem Backend viele Vorteile bringt und die Responsiveness spürbar verbessert. Der Service selbst unterstützt das Pattern ebenfalls nativ über Windows Communication Foundation (siehe Abschnitt 4.4.2.2 zur asynchronen Kommunikation); gleiches gilt auch für die zugrunde liegende Datenbankanbindung mittels Entity Framework 6.0 (siehe Abschnitt 4.4.3.2), wo das neue Task-Modell ebenfalls voll integriert ist.

4.4.2 Kommunikation zwischen Client und Server

Die Kommunikation zwischen Client und Backend Services wurde mit Windows Communication Foundation realisiert. Es handelt sich um einen normalen SOAP Service mit BasicHttpBinding und Konfiguration für erweiterte Features wie WCF Streaming (welches für die Übertragung der Medien genutzt wird).

Da zum Zeitpunkt der Abgabe die genaue Infrastruktur, auf welcher der Service betrieben werden soll, noch nicht feststeht, wurde vereinbart, dass zu Beginn Self-Hosting in Form einer Konsolenapplikation implementiert werden soll.

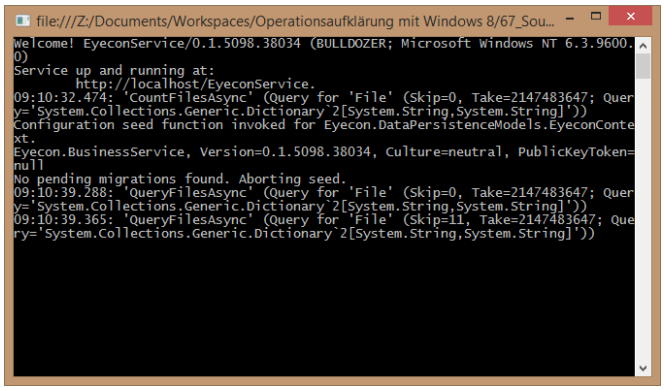


Abbildung 24: Self-hosted WCF Web Service

Eine spätere Umstellung von dieser Form des Hostings auf IIS oder Windows Services ist mit minimalem Aufwand verbunden, da Service und Schnittstelle nicht angepasst werden müssen.

4.4.2.1 Gemeinsamer Service Contract und DTOs

Die Service Contract Schnittstelle IEyeconService, sowie sämtliche für die Kommunikation genutzten Data Transfer Objects sind im Package DataTransferInterface (siehe Abschnitt 4.3.2.4) im

gleichnamigen Assembly definiert und werden von Client und Server gleichermassen referenziert. Beide konsumieren jeweils praktisch sämtliche Inhalte aus dem Package. Im Gegenzug wurden Client und Server *vollständig* entkoppelt, so dass diese einander nicht mehr kennen und in der Solution auch keine Referenzierung zwischen den jeweiligen Projekten existiert.

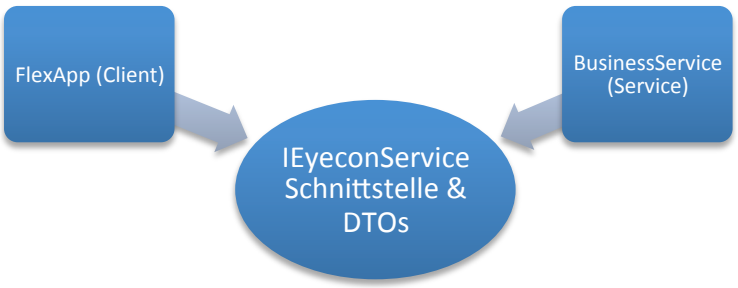


Abbildung 25: Kommunikation über gemeinsamen Service Contract

Die Entscheidung zu diesem Design kam zustande, weil es sich im vorliegenden Fall um eine Topologie handelt, bei der alle Parameter jederzeit bekannt sind, da alle Bestandteile der Infrastruktur unter der Kontrolle der Entwickler sind. Die Vorteile dieses Design lassen sich wie folgt zusammenfassen:

- Die Entwicklung wird vereinfacht, da sämtliche Bestandteile des Projekts (also Client, Schnittstelle und Server) in der gleichen Solution sind und die Gültigkeit aller Aufrufe bereits zum Build-Zeitpunkt statisch durch den Compiler geprüft werden kann.
- Refactoring wird vereinfacht, da bei einer Anpassung der Service-Schnittstelle die betroffenen Stellen im App- und Backend-Projekt automatisch angepasst werden.
- Es wird eine vollständige Entkoppelung von App und Backend erreicht, da ausschliesslich über den Service Contract und die DTOs miteinander kommunizieren.
- Aus der vollständigen Entkoppelung folgt die Möglichkeit, das Frontend oder das Backend komplett zu ersetzen. Dies ist insbesondere für das Frontend interessant, da dort prinzipiell nichts gegen die Koexistenz verschiedener Clients spricht.

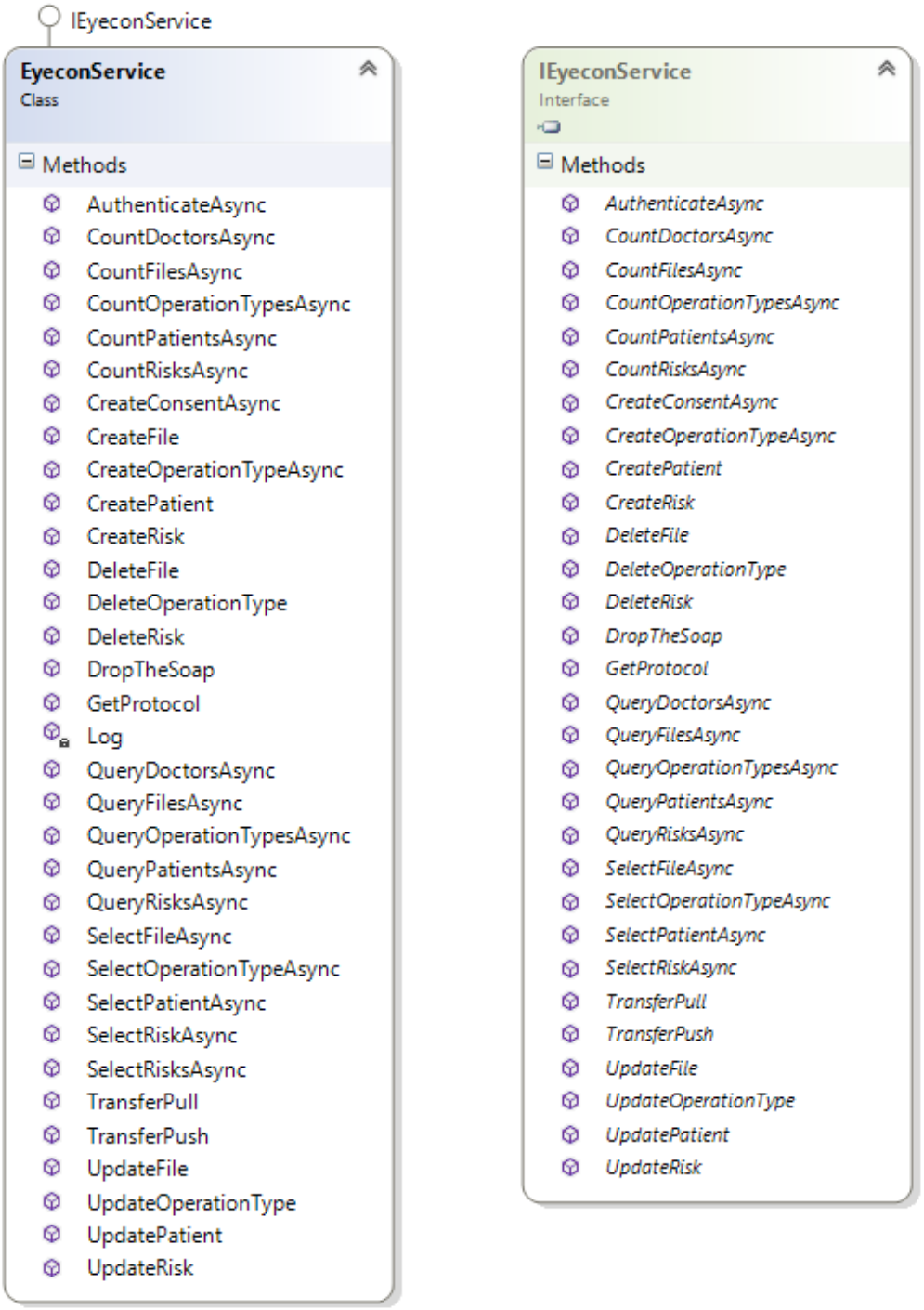


Abbildung 26: IEyeconService und EyeconService

Pro veröffentlichte Entität bietet die Service Schnittstelle jeweils mehr oder weniger ein Standardset an Methoden an, welche wie folgt erklärt werden können:

Methode	Bedeutung
CountTAsync	Zählt die durch den PagingContext gefilterten Resultate.

CreateT	Erzeugt einen neuen Datensatz.
DeleteT	Löscht den ausgewählten Datensatz.
QueryTAsync	Liefert die durch den PagingContext gefilterten Resultate.
SelectTAsync	Liefert den ausgewählten Datensatz.
UpdateT	Updatet den ausgewählten Datensatz.

Tabelle 10: Standardmethoden für Service-Calls

Selbstverständlich wurden nur die Operationen implementiert, welche für den jeweiligen Anwendungsfall sinnvoll sind und als Transaktion modelliert werden konnten. Unnötige CRUD Operationen wurden in der Service Schnittstelle weggelassen. Query- und Select-Calls liefern verknüpfte Entitäten gleich mit (Patientenrisiken werden mit dem Patienten geladen), Updates behandeln diese entsprechend (Patientenrisiken werden mit dem Patienten gespeichert).

4.4.2.2 Asynchrone Kommunikation

Für die Kommunikation über Windows Communication Foundation wurde eine mehrheitlich asynchrone Schnittstelle entworfen, was bereits mit dem Service Contract beginnt. Auf diese Weise kann vom sprachinternen *await-async* Pattern gebraucht gemacht werden, was die Entwicklung nebenläufiger Anwendungen stark vereinfacht und zu einem flüssigeren Laufzeitverhalten führen kann. Dies ist insbesondere auch darum interessant, weil das zur Datenverwaltung genutzte Entity Framework (siehe Abschnitte 4.3.4 und 4.4.3.2) das Pattern ebenfalls unterstützt. Wo sinnvoll wurden deshalb die Operation Contracts so definiert, dass sie anstelle der eigentlichen Resultate Task-Objekte zurückliefern.

```
[OperationContract]
Task<Risk[]> QueryRisksAsync(PagingContext<Risk> Cont);
```

Die Clients, welche gegen die Service Schnittstelle kommunizieren, interpretieren solche Definitionen automatisch als asynchrone Methoden und erlauben einen entsprechenden Aufruf.

```
await LocatorService.Resolve<IEyeconService>().QueryRisksAsync(this.Paginator)
```

Auf der Implementierungsseite wurden die entsprechenden Methoden mit dem Schlüsselwort *async* markiert, und, wo sinnvoll, entsprechende *awaits* gesetzt.

```
public async Task<DataTransferInterface.Risk[]>
QueryRisksAsync(PagingContext<DataTransferInterface.Risk> Cont)
{
    return await Task.Run(() =>
    {
        using (var Context = new EyeconContext())
        {
            return Context.Risks
                .Where(Risk => Risk.Name.Contains(Cont.QueryString))
                .OrderBy(Risk => Risk.Name).Skip(Cont.Skip).Take(Cont.Take).ToArray()
                .Select(Risk => Risk.ToTransferInterface()).ToArray();
        }
    }).Uncapture();

}
```

Durch konsequente Anwendung dieser asynchronen Kommunikation und Kapselung dieser Aufrufe in View Models und Commands konnte die Responsiveness der App spürbar verbessert werden.

4.4.2.3 Data Paging und Suche

Beim Design der Service Schnittstelle wurde der Frage nachgegangen, was passiert, wenn im Laufe der Lebenszeit der App irgendeinmal nicht mehr 100 Risiken (oder andere Datensätze), sondern 10000 oder noch mehr vorhanden sind. In diesem Fall kommt es zu langen Wartezeiten oder sogar Timeouts und die App sowie das Backend können nicht mehr genutzt werden. Die Lösung hierfür ist Paging, welches zusammen mit der sowieso benötigten Suchfunktion umgesetzt wurde. Es handelt sich um eine einfache, aber funktionsfähige und erweiterbare Implementierung auf Basis der Klasse PagingContext<T>, welche im DataTransferInterface Package (siehe Abschnitt 4.3.2.4) als DTO definiert ist und bei Service-Calls mitgegeben werden kann, um Paging- und Suchparameter zu spezifizieren.

Das Paging selbst wird direkt auf dem Entity Framework Context mit den LINQ Methoden Skip und Take realisiert, welche nach der konfigurierten Filterung aufgerufen werden um das jeweilige Resultset einzuschränken.

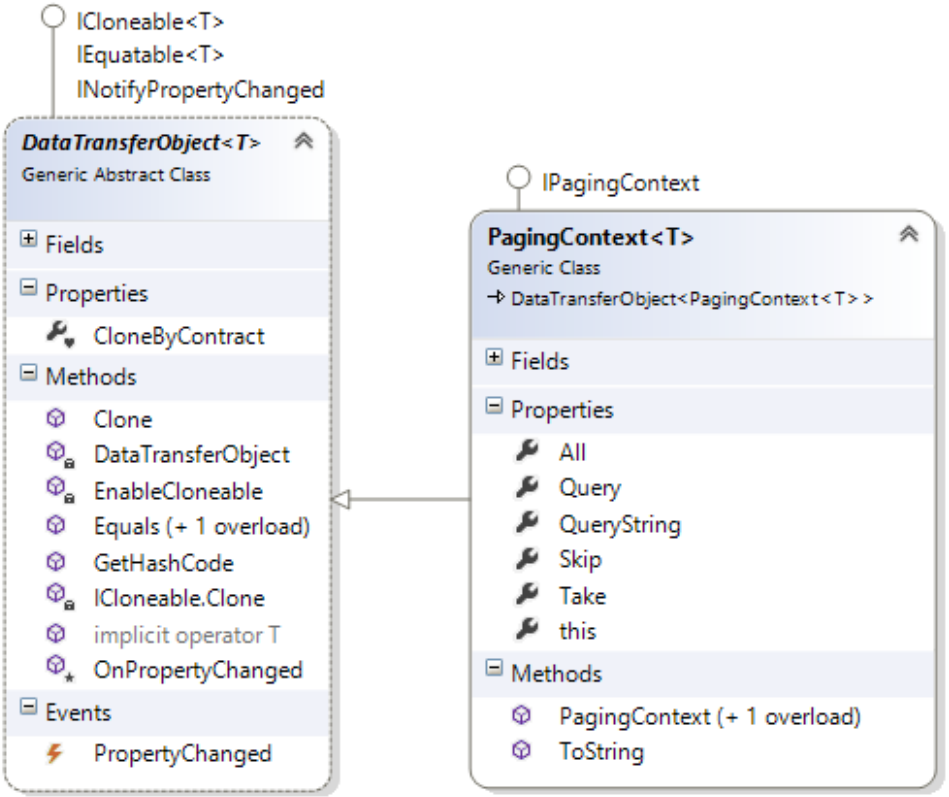


Abbildung 27: Klasse PagingContext<T>

Beim PagingContext handelt es sich also um nichts anderes als einen weiteren Data Contract mit einigen Properties, welche die ausgeführten LINQ Queries modifizieren.

Property	Bedeutung
All	Ein statisches Property, welches für den parametrisierten Typen PagingContext<T> einen Kontext liefert, welcher das Resultset nicht filtert und kein Paging durchführt. Nützlich, wenn alle Elemente abgefragt werden sollen.

Query	Ein Dictionary, welches die Spezifikation beliebiger (Such-)Parameter erlaubt. Die Bedeutung der Parameter wird durch die Service-Methode spezifiziert und die Werte werden nicht durch den PagingContext verifiziert.
QueryString	Ein Standard Parameter, welcher in der Regel für Bindings genutzt wird.
Skip	Beschreibt, wie viele Elemente ausgelassen werden sollen.
Take	Beschreibt, wie viele Elemente maximal zurückgeliefert werden dürfen.

Tabelle 11: Properties der PagingContext<T> Klasse

Hauptnutzer der PagingContext<T> Klasse sind ListViewModels (siehe Abschnitt 4.4.1.1) und Ableitungen derselben, welche das Paging und die Suchfunktionen beinahe transparent implementieren und den Pages (siehe Abschnitt 4.4.5.3) über Commands (wie LoadMore oder CountAvailable) zur Verfügung stellen.

4.4.2.4 File Streaming

Einen Sonderfall der Datenübertragung im Backend stellt das File Streaming dar, wo jeweils bereits eine einzelne Entität ein Vielfaches der normalerweise erlaubten Grösse für die Übertragung hat. Dieses Problem kann nicht mit Paging (siehe Abschnitt 4.4.2.3) gelöst werden und muss grundsätzlich anders angegangen werden.

Bevor man die Zusammenhänge genauer anschaut muss die Terminologie geklärt werden: Im Datenmodell respektive der Datenbank (siehe Abschnitt 4.3.4) existiert eine Entität File, welche ganze Dateien in einer VARBINARY Column aufnimmt. Bei den DTOs im DataTransferInterface Package (siehe Abschnitt 4.3.2.4) existiert eine Klasse File, welche als Proxy für eine solche Entität fungiert, die VARBINARY Column jedoch niemals mitführt, sonst aber über die gleichen Attribute verfügt. Diese Attribute sind in der Schnittstelle IFileProxyDescriptor zusammengefasst, welche allen File-Klassen gemeinsam ist.

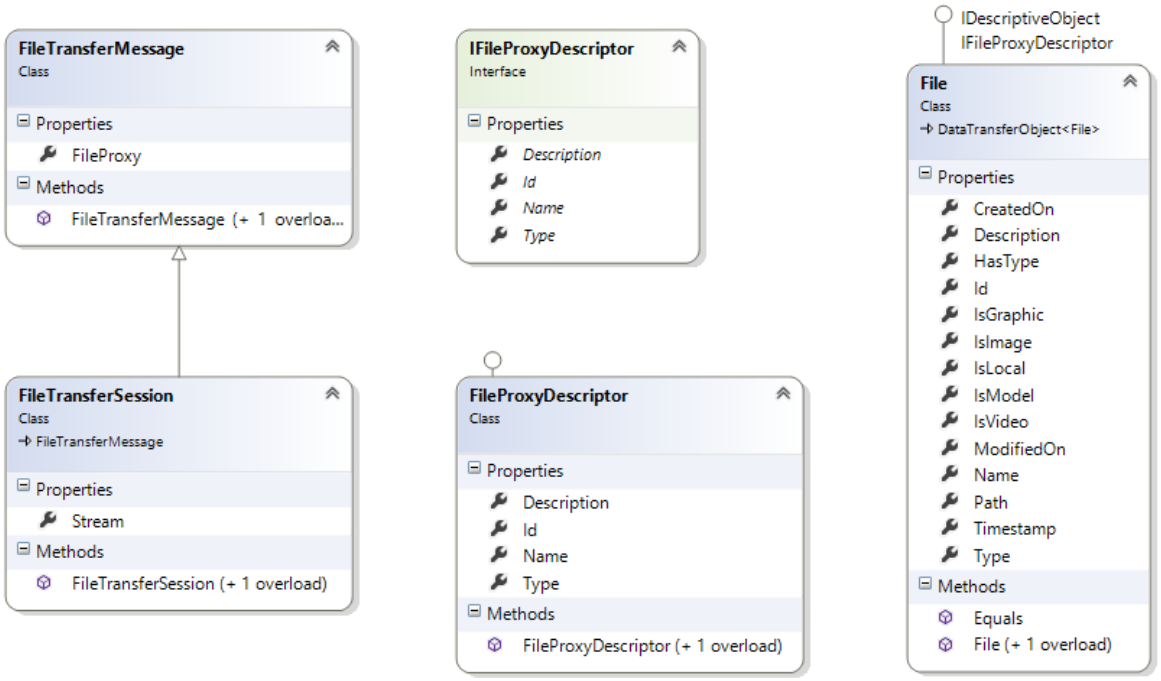


Abbildung 28: Klassen der File Streaming Infrastruktur

Der Web Service bietet nun für Files dieselben asynchronen, mit Paging-Unterstützung ausgestatteten Methoden an, die auch für andere Entitäten vorhanden sind, wobei der Dateiinhalt aber niemals mitgeliefert wird. Die Motivation hinter diesem Design ist, dass es so möglich wird, effizient tausende Files zu durchsuchen. Der eigentliche Inhalt der Files wird nur bei Bedarf vom Server geladen, was die Implementierung eines Client-seitigen File Caching ermöglicht und nahelegt.

Der Up- und Download von Files wird daher nicht wie eigentlich erwartet durch eine Methode wie QueryFilesAsync realisiert, sondern über die Methoden TransferPull und TransferPush.

```
[OperationContract, FaultContract(typeof(FileTransferException))]
Task<FileTransferMessage> TransferPush(FileTransferSession Request);

[OperationContract, FaultContract(typeof(FileTransferException))]
Task<FileTransferSession> TransferPull(FileTransferMessage Request);
```

Als Parameter wird dabei jeweils ein Objekt vom Typ FileTransferMessage bzw. FileTransferSession übergeben. Diese beiden Klassen sind als WCF Message Contracts (*nicht Data Contracts!*) definiert und enthalten im Header einen FileProxyDescriptor, welcher das gemeinte File identifiziert. Die Klasse FileTransferSession enthält als einzigen Body Member zusätzlich einen beliebigen .NET Stream (System.IO.Stream), welcher von WCF kontinuierlich über das Netzwerk gesendet wird. Die Kommunikation beim File Transfer erfolgt also übers Kreuz:

Transfermodus	Parameter beim Aufruf	Resultat des Aufrufs
Pull (Download)	FileTransferMessage mit Informationen über das zu herunterladende File.	FileTransferSession, vom Server eröffnet, mit Stream ins Memory des Servers.
Push (Upload)	FileTransferSession, vom Client eröffnet, mit Stream direkt ins Dateisystem des Clients.	FileTransferMessage mit Informationen über das hochgeladene File.

Abbildung 29: File Transfer Vorgänge

Die Übermittlung der Streams über das Netzwerk wird von WCF nach einmaliger Konfiguration automatisch gehandhabt und die Streams können im Code wie gewohnt anprogrammiert werden. Service-seitig ist der Upload eines Files daher als einfaches Lesen eines Streams umgesetzt:

```
var Source = Session.Stream;

// Load it all into memory.
using (var Target = new MemoryStream())
{
    await Source.CopyToAsync(Target);

    // ...
}
```

Die Konfiguration des Services für Streaming kann deklarativ oder per Code beim erzeugen des WCF Bindings erfolgen. In jedem Fall muss der TransferMode auf Streamed gesetzt werden:

```
var Binding = new BasicHttpBinding()
{
    TransferMode = TransferMode.Streamed,
    MaxReceivedMessageSize = 2147483647
};
```

Die Synchronhaltung und das Caching der Files auf der Client-seite ist über entsprechende View Models abgehandelt (siehe Abschnitt 4.4.1.1) und wird an dieser Stelle nicht besprochen.

4.4.2.5 Berechtigung für Self-Hosting

WCF Services, welche interaktiv in einer Konsolenapplikation gehostet werden, verfügen beim Start nicht über ausreichende Berechtigungen, um eine URL-Reservation in Windows tätigen zu können. Der BusinessService erkennt diesen Zustand und schreibt eine entsprechende Meldung – sowie einen Vorschlag zur Lösung – auf die Konsole.

Um dieses Problem zu lösen kann man den Service entweder als Administrator starten (ein Account mit Administratorberechtigungen reicht hierfür nicht aus, da User Account Control eine Reservierung dann interaktiv bestätigen lassen muss), oder einmalig eine Reservationserlaubnis für den Benutzer vergeben, welcher den Service startet (Microsoft Corp., 2012). Hierzu muss auf einer Kommandozeile im Administratormodus folgendes Kommando abgesetzt werden:

```
netsh http add urlacl url=http://+:10001/YourUri user=DOMAIN\\USER
```

Nach (korrekter) Eingabe dieser Zeile lässt sich der Service interaktiv vom angegebenen Benutzer hosten, was auch bei der Entwicklung interessant ist.

4.4.3 Spezielle Aspekte im Backend

4.4.3.1 Projektion von Entities in Data Transfer Objects

Die in der Datenbank vorhandenen Entitäten (siehe Abschnitt 4.3.4) werden vom Backend über den Service Contract nicht direkt zur Verfügung gestellt, da dies nicht nur eine unnötige Koppelung zwischen den verschiedenen Projekten (die Frontend App müsste damit die genauen Entitäten kennen) verursachen würde, sondern auch gefährlich wäre, da eventuell unerlaubte Operationen möglich würden.

Stattdessen werden im DataTransferInterface Package (siehe Abschnitt 4.3.2.4) Data Transfer Object Klassen definiert, welche im Rahmen des Service Contracts für die Kommunikation genutzt werden. Diese Klassen scheinen denjenigen aus dem DataPersistenceModels Package (siehe Abschnitt 4.3.2.3) sehr ähnlich zu sein, werden aber dazu genutzt, um im Service Contract sinnvolle Transaktionen anbieten zu können.

Damit das Backend fliessend zwischen Entities und DTOs konvertieren kann, wurde eine Klasse DomainMapping definiert (diese ist als partial deklariert und über die Dateien der jeweiligen Entity Klassen verstreut). Die Klasse beinhaltet Extension-Methoden, welche Entities und deren Navigation Properties in DTOs projizieren und umgekehrt.

Extension	Beschreibung
ToTransferInterface(Entity)	Konvertiert einen Entity Objekt-Graphen in einen DTO Objekt-Graphen für die Übermittlung via WCF.
ToPersistenceModels(DTO)	Konvertiert einen DTO Objekt-Graphen in einen Entity Objekt-Graphen für das Entity Framework.

Tabelle 12: Extensionmethoden der DomainMapping Klasse

4.4.3.2 Asynchrones LINQ to Entities

Das Entity Framework 6.0 unterstützt das neue Task-Modell von .NET 4.5 (siehe Abschnitt 4.4.1.4) und damit das sprachinterne *await-async* Pattern für die vereinfachte Umsetzung nebenläufiger Datenbankabfragen (Microsoft Corp., 2013). Obwohl die Datenabfragen selbst dadurch kaum spürbare Performancegewinne erfahren, macht die Benutzung dieses Features Sinn, da damit die Responsiveness der Clients bei lang laufenden Abfragen verbessert werden kann.

Diese Überführung asynchroner Aufrufe vom Client über den Web Service in die Datenschicht wird dadurch ermöglicht, dass Windows Communication Foundation das neue Task-Pattern ebenfalls intrinsisch unterstützt (siehe Abschnitt 4.4.2.2 zur asynchronen Kommunikation). Service-Methoden, bei denen sich ein asynchroner Aufruf anbietet, eignen sich auch für die asynchrone Kommunikation mit der Datenbank:

```
public async Task<Guid> CreateOperationTypeAsync(DataTransferInterface.OperationType Original)
{
    return await Task.Run(async () =>
    {
        using (var Context = new EyeconContext())
        {
            var Target = Original.ToPersistenceModels();
            Context.OperationTypes.Add(Target);
            await Context.SaveChangesAsync();
            return Target.Id;
        }
    }).Uncapture();
}
```

Aufrufe dieser Art verhalten sich also in mehrfacher Hinsicht asynchron, da bereits das aufrufende View Model beim Client mit dem Service asynchron kommuniziert. Der Service selbst kommuniziert mit der Datenbank ebenfalls asynchron.

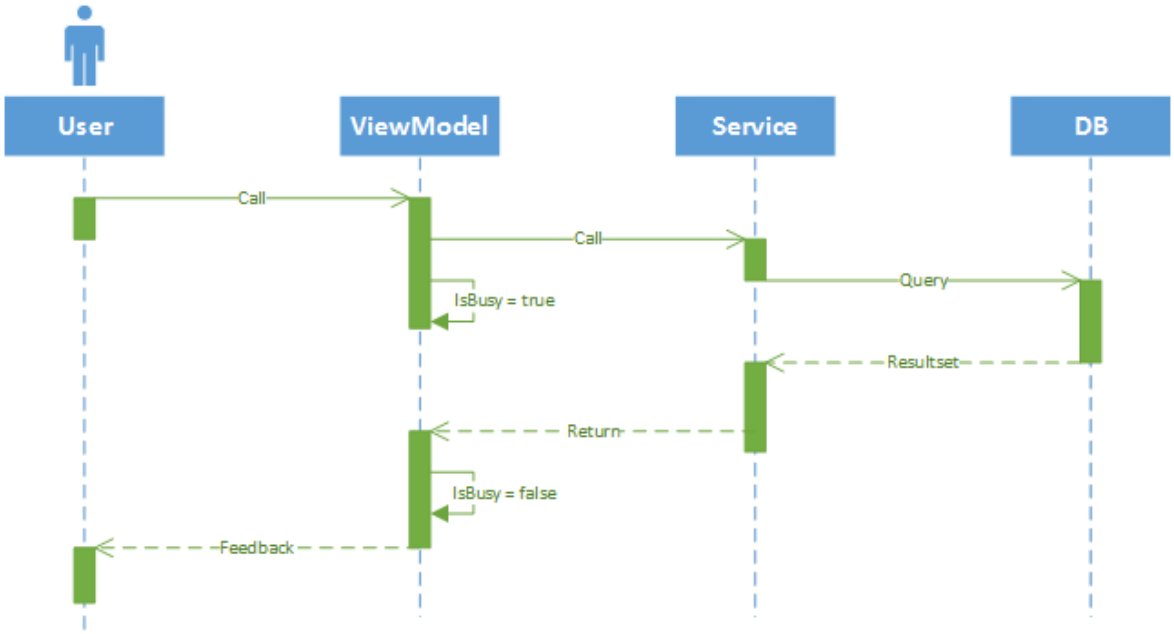


Abbildung 30: Asynchrone Kommunikation zwischen Client, Service und Datenbank (ohne Controls)

4.4.3.3 Exception Handling mit WCF

Das Exception Handling im Backend bedarf einer separaten Besprechung, da die Exceptions aus Sicht der Benutzer der Frontend App auf einem entfernten System (dem Server, wo der Web Service ausgeführt wird) auftreten. Bei der Kommunikation mit einem WCF Service können Exceptions jedoch nicht einfach vom Server zum Client weitergereicht werden, da in der Regel über ein SOAP-Protokoll kommuniziert wird. Stattdessen müssen Exceptions in Faults verpackt werden (dies ist nur möglich, wenn die Exceptions serialisierbar sind), welche dann serialisiert, übermittelt und schliesslich wieder deserialisiert werden können, worauf das eigentliche Exception Handling auf der anderen Maschine (dem Client) weitergehen kann.

Im Zusammenhang mit der Datenverwaltung im Backend wurden zwei zusätzliche Exceptionklassen definiert und als mögliche Faults im Service Contract spezifiziert. Bei beiden Exceptiontypen handelt es sich um serialisierbare Data Contracts, welche im DataTransferInterface Package (siehe Abschnitt 4.3.2.4) definiert sind.

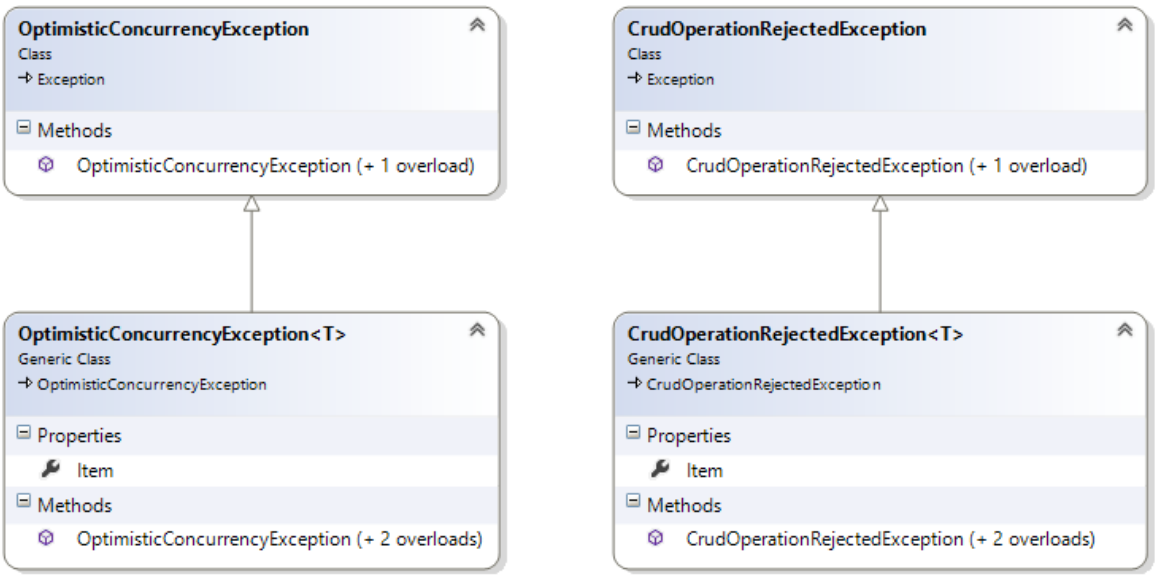


Abbildung 31: Exceptionklassen im DataTransferInterface

Beide Exceptionklassen repräsentieren Fehler in der Datenverwaltung und erlauben in der abgeleiteten Variante die Angabe des betroffenen Objektes. In der Regel wird das übermittelte DTO angegeben, welches den Fehler verursacht hat. Tritt ein Fehler bei einem Patient-Objekt auf, so wird dieses also zu Informationszwecken wieder übermittelt.

Exception	Beschreibung
CrudOperationRejectedException	Repräsentiert einen Fehler, bei welchem eine unerlaubte CRUD Operation ausgeführt werden sollte. Unerlaubt heisst hier, dass die Operation für die Integrität der Daten nicht förderlich wäre.
OptimisticConcurrencyException	Repräsentiert einen Concurrency Fehler (siehe Abschnitt 4.4.3.4 zum Thema Optimistic Concurrency).

Tabelle 13: Exceptionklassen im DataTransferInterface

4.4.3.4 Optimistic Concurrency

Das Backend implementiert eine Absicherung gegen Inkonsistenzen durch Fehler wegen nebenläufigen Aufrufen mehrerer Clients auf Basis eines *Optimistic Concurrency Control* (OCC) Mechanismus (Microsoft Corp., 2013). OCC gewährt in einer Umgebung mit potentiell mehreren gleichzeitig aktiven Benutzern allen den Parallelzugriff mit Leserechten für alle Datensätze in der Annahme, dass diese meistens nur gelesen und nur sehr selten geschrieben werden müssen.

Wird ein Datensatz von jemandem geändert, erhalten alle anderen Benutzer, welche dies kurz danach ebenfalls tun wollen, eine entsprechende Fehlermeldung, dass ihre gewünschte Änderung auf Basis veralteter Daten begründet ist – dem Änderungswunsch wird in der Folge nicht Rechnung getragen. Der Benutzer, dessen Änderung tatsächlich durchgeführt wurde, ist also *privilegiert*.

Der OCC-Ansatz minimiert Konflikte und Verwaltungsaufwand, da nur gleichzeitige Schreibzugriffe zu Fehlern führen (und damit geprüft werden müssen), Lesezugriffe hingegen per Definition sicher sind. Ein eigentliches Locking von Datensätzen ist nicht nötig.

Das Backend implementiert OOC auf der Ebene der Datenhaltung mit dem Entity Framework, indem bei Update-Methoden sowohl das originale, als auch das modifizierte Objekt übermittelt werden muss. Dies ist bereits im Service Contract (siehe Abschnitt 4.4.2.1) ersichtlich:

```
[OperationContract, FaultContract(typeof(OptimisticConcurrencyException<OperationType>))]
void UpdateOperationType(OperationType Original, OperationType Modified);
```

Beim Update vergleicht das Entity Framework das Original mit dem aktuell in der Datenbank vorhandenen Stand und löst eine System.Data.OptimisticConcurrencyException aus, sollten diese nicht miteinander übereinstimmen. Tritt dieser Fall ein, so wird vom Backend eine OptimisticConcurrencyException (siehe Abschnitt 4.4.3.3) geworfen, welche via WCF an den Client übermittelt und dort behandelt wird (typischerweise durch eine einfache Fehlermeldung).

Der Ansatz OOC wurde gewählt, weil die Daten für die Operationsaufklärung nur sehr selten geändert werden müssen – in vielen Fällen bietet der Service Contract nicht einmal eine entsprechende Update-Methode an, da diese in keinem realistischen Use Case gebraucht wird – und die Wahrscheinlichkeit einer parallelen Änderung sehr gering ist. Der Aufwand für das Concurrency Control konnte daher so minimiert werden.

4.4.4 Umsetzung Modern Style auf dem Desktop

Damit eine Windows Desktop App auf Basis der Windows Presentation Foundation wie eine Windows Store App gestaltet werden kann braucht es wesentlich mehr als nur entsprechende Styles. Einige der wichtigsten Steuerelemente und Verhaltensweisen der Windows Store Umgebung haben auf dem Desktop keine Entsprechung und müssen daher separat nachgerüstet oder selbst entwickelt werden. Für die Umsetzung der vorliegenden Arbeit wurde daher auf zwei frei verfügbare UI Libraries (siehe Abschnitte 4.3.5.2, 4.4.4.1 und 4.4.4.2) genutzt, welche bereits einen grossen Teil der Anforderungen bereits umsetzen und nur noch in geringem Masse angepasst werden mussten.

Beim Thema Touch sieht die Situation ganz ähnlich aus, da WPF von Anfang an mit einer recht gut dokumentierten Touchunterstützung entwickelt wurde (Petzold, 2010), welche aber über Jahre hinweg nirgendwo wirklich genutzt wurde. Es waren daher an verschiedenen Stellen Anpassungen notwendig (siehe Abschnitt 4.4.4.3), welche zumeist mit Styles oder Behaviors umgesetzt wurden.

4.4.4.1 Modern UI for WPF

Modern UI for WPF, im Code und Dokumentation zumeist nur MUI genannt implementiert eine Sammlung von Controls und Styles für WPF (Zwikstra, 2013), welche im Design stark an die Guidelines von Microsoft für Modern Style bzw. dem Vorgänger Zune angelehnt ist. MUI geht dabei weit über die Bereitstellung von Controls und Styles hinaus und bietet ein eigenes Konzept für die Navigation innerhalb einer App und das dynamische Nachladen von Inhalten.

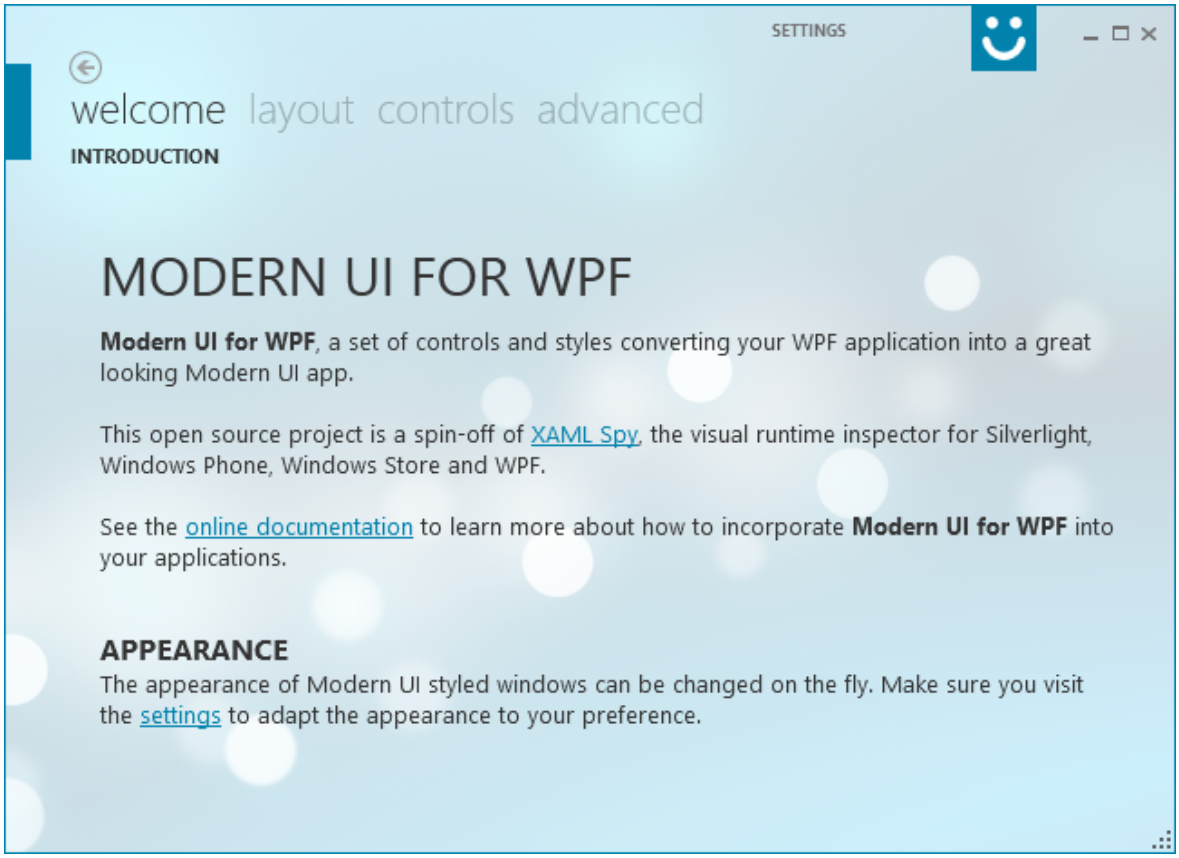


Abbildung 32: Modern UI Snowflake Sample

Für die vorliegende Arbeit wurden diverse Bestandteile von MUI genutzt.

MUI Klasse	Beschreibung	Nutzungsart
IContent	Die IContent Schnittstelle stellt die Verbindung zwischen Daten und Pages und erlaubt die Behandlung von Events wie OnNavigatedTo oder OnNavigatedFrom.	Alle Pages im Projekt wurden als UserControls von ModernPage abgeleitet, welches IContent implementiert.
Link	Implementiert einen Link für ein ModernMenu (siehe Abbildung 32 das Menuelement *INTRODUCTION*). Ein Link ist ein Datenelement mit einem vordefinierten, impliziten Style, welcher über die Attribute DisplayName und Source verfügt. Das Attribut Source verweist dabei auf eine XAML-Datei im Projekt, welche bei Auswahl des Links neu geladen wird und in einem ModernFrame angezeigt werden kann.	Genutzt in View Models für Navigation.
LinkGroup	Implementiert eine Obergruppe für Links (siehe Abbildung 32 im Menu die Hauptelemente *welcome, layout, controls und advanced*) für ModernMenus.	Genutzt in View Models für Navigation.
ModernFrame	Implementiert einen Content-Bereich mit Modern Style Transition Effekt (siehe Abbildung 32 Fensterinhalt), wo	Die Content-Bereiche der Hauptansicht, sowie auf dem

	beliebige User Controls angezeigt werden können. *Gemäss MUI Konvention ist jeweils die Rede von Pages, tatsächlich funktioniert die Darstellung von Pages, wie man sie aus WPF kennt, nicht in ModernFrames. Alle Pages werden als UserControls realisiert.*	Assistenten zur Aufklärung wurden mit ModernFrames realisiert. Die Auswahl der aktuellen Page erfolgt über ein ModernMenu, Links oder Buttons.
ModernMenu	Implementiert ein Modern Style bzw. Zune Style Menu (siehe Abbildung 32 oberer Fensterrand). Die Bezeichnung Menu ist irreführend, da es sich tatsächlich eher um eine Art mehrstufiges Pivot (Microsoft Corp., 2013) bzw. Tabs handelt.	Die Navigation auf der Hauptansicht der App erfolgt mit einem ModernMenu. Für die Verwendung mit Touch wurde der Default-Style von ModernMenu umgeschrieben und die Elemente vergrössert.
ModernWindow	Implementiert ein Modern Style bzw. Zune Style Window (siehe Abbildung 32) mit zusätzlichen Features wie Chrome Settings und Icons.	Hauptansicht und Assistent wurden mit einem modifizierten ModernWindow umgesetzt.

Tabelle 14: Genutzte MUI Klassen

4.4.4.2 MahApps.Metro

MahApps.Metro ist ein Open Source Projekt welches seit dem Jahr 2011 existiert und zum Ziel hat, Metro bzw. Modern Style UIs in WPF zu realisieren. Anders als bei Modern UI (siehe Abschnitt 4.4.4.1) wird dieses Ziel fast ausschliesslich über Styles erreicht, so dass nur die Controls neu implementiert wurden, welche in WPF noch nicht existieren.

Abbildung 33: MahApps.Metro Window Chrome

Für die Umsetzung des vorliegenden Projekts wurde nur ein kleiner Teil von MahApps.Metro genutzt. Es handelt sich dabei hauptsächlich um einzelne implizite Styles, welche besser für Touch optimiert sind als bei MUI (siehe Abschnitt 4.4.4.1), sowie um das ProgressRing Control, welches Windows 8 typisch einen kontinuierlichen Fortschritt symbolisiert.

MahApps.Metro implementiert ein Panorama Control sowie Tiles. Zum Zeitpunkt der Projektumsetzung war dieses Panorama jedoch sehr instabil und konnte nur sehr umständlich mit Tiles – und keinen anderen Inhalten – gefüttert werden, weshalb das Projektteam von einer Verwendung weggekommen ist. Stattdessen wurde eine eigene Lösung umgesetzt (siehe Abschnitt 4.4.6).

4.4.4.3 Unterstützung von Touch in WPF

Sowohl für die nativen Controls von WPF, als auch für diejenigen, welche von MUI oder MahApps.Metro (siehe Abschnitte 4.4.4.1 und 4.4.4.2) referenziert werden, musste die Touch-Fähigkeit geprüft werden. In fast allen Fällen waren dabei Defizite bemerkbar, welche sich grob in drei Kategorien aufteilen lassen:

#	Problemart	Beschreibung

1.	Design	Grösse und Design eines Controls sind nicht für Touch optimiert. In der Regel sind die Controls zu klein.
2.	Verwendung	Die Verwendungsart eines Controls passt nicht zu Toucheingaben.
3.	Kombination	Die Kombination von Controls führt bei Toucheingaben zu Problemen, was oft mit der Fokussteuerung von Windows oder WPF zu tun hat.

Tabelle 15: Standardprobleme von WPF Controls bei Toucheingaben

Die Probleme der Kategorien 1 und 2 sind in den meisten Fällen mit Styles (siehe Abschnitt 4.4.4.5) oder der Wahl eines alternativen Steuerelements lösbar. Dieser Ansatz wurde daher auch so oft wie möglich genutzt.

Schwieriger ist es, wenn man mit der Kategorie 3 kämpft, da diese Fehler nicht auf den ersten Blick offensichtlich sind und ihre Behebung oft ungeahnte Seiteneffekte hat, weil die Event Handler nicht sauber ausprogrammiert wurden.

4.4.4.4 Tastaturen und andere virtuelle Geräte

Eine spezielle Rolle bei der Umsetzung eines UIs für Touch kommt der Tastatur zu. Für den Formfaktor von Tablets (oder auch bei Smartphones) ist der Anschluss einer physischen Tastatur (oder auch einer physischen Maus) an das jeweilige Gerät eher die Ausnahme als die Regel. Dies hat Konsequenzen auf das Design einer App, da diese auf eine Bedienung ohne eben diese Eingabegeräte vorbereitet sein muss.

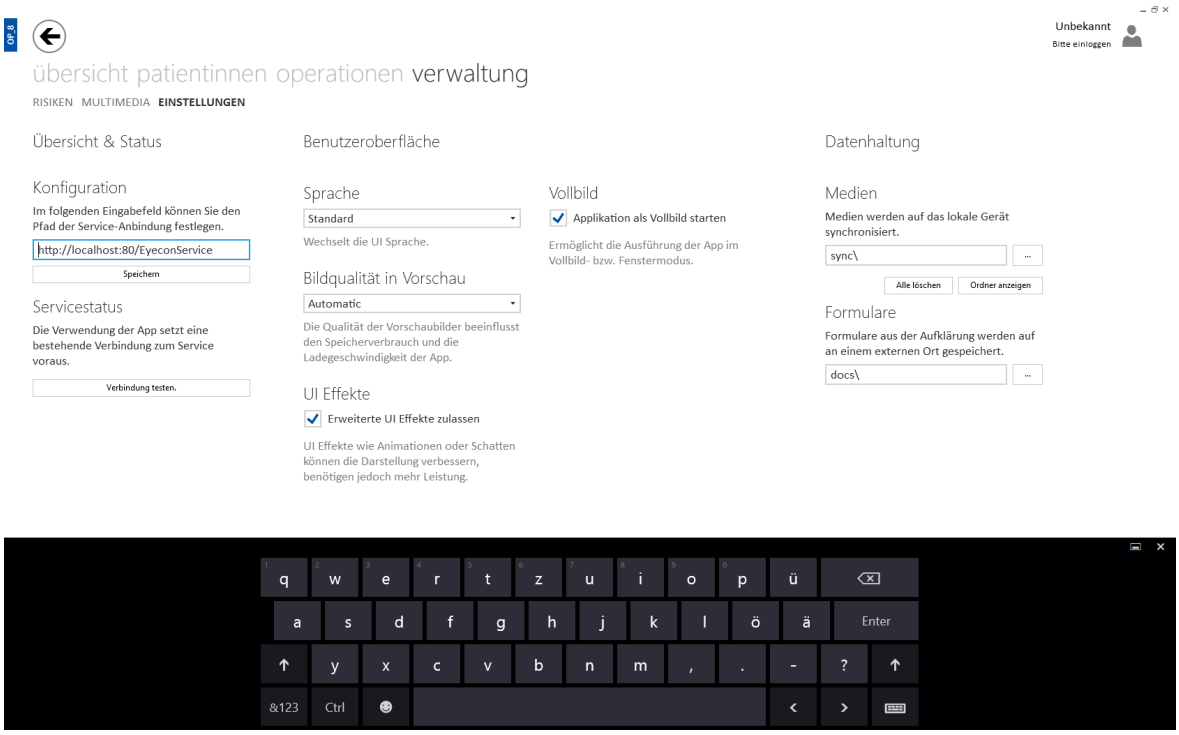


Abbildung 34: Unterstützung des Windows 8 On-screen Keyboards mit dynamischem Layout

Da beim Industriepartner an der Klinik für Geburtshilfe die Absicht besteht, die App auf Microsoft Surface Tablets – entsprechende Geräte wurden dem Projektteam zur Verfügung gestellt – zu

benutzen, welche standardmässig ohne zusätzliche Eingabegeräte geliefert werden, wurde eine umfassende Unterstützung für das On-Screen Keyboard von Windows 8 umgesetzt (siehe Abschnitt 4.4.9.3 für genauere Erläuterungen zu diesem Thema).

Anzumerken ist hierbei jedoch, dass die Unterstützung der virtuellen Tastatur leider weder von Windows Presentation Foundation, noch vom Windows Runtime Subset für Desktop Apps gegeben ist und daher selbst entwickelt werden musste. Die Anbindung dieser Unterstützung erfolgt über Styles für die relevanten Steuerelemente. Beim Layout der App Pages (siehe Abschnitt 4.4.5.3) wurde darauf geachtet, dass dieses keine fixen Grössen vorgibt und sich der vorhandenen Bildschirmgrösse anpassen kann. So wurde sichergestellt, dass die App auch mit angezeigter virtueller Tastatur bedienbar bleibt.

4.4.4.5 Styles und Templates

In der Frontend App wird eine grosse Anzahl an verschiedenen Styles und Templates genutzt; die Benutzung erstreckt sich dabei jeweils über diverse Pages (siehe Abschnitt 4.4.5.3). Aus diesem Grund wurden die genutzten Styles aus der Definition der Pages herausfaktorisiert und in separaten XAML-Ressourcen abgelegt. Diese befinden sich im Resources Package (siehe Abschnitt 4.3.2.6) und werden per MergedDictionary in die globale App Klasse eingebunden, damit sie allen Pages implizit zur Verfügung stehen.

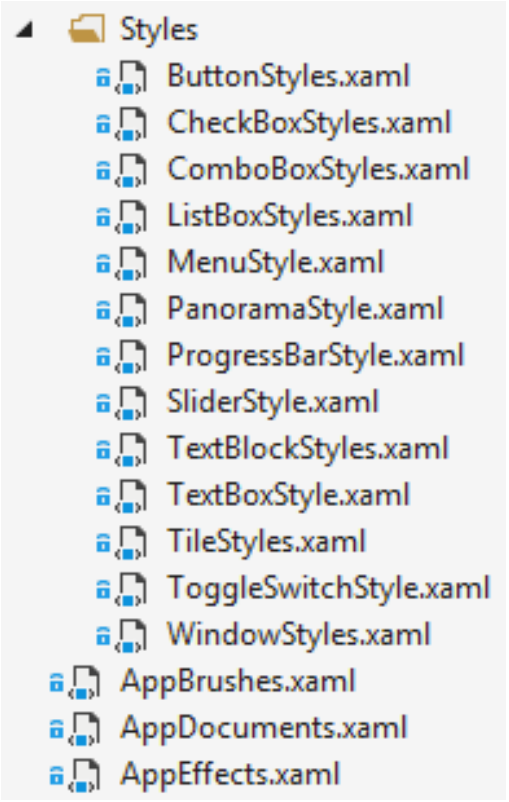


Abbildung 35: Styles und Templates in den App-Ressourcen

Um das Styling in der App flexibler und systematischer zu gestalten, wurden auch bei den Styles selbst bis zu einem gewissen Grad Gemeinsamkeiten herausfaktorisiert und wieder in separaten XAML-Ressourcen abgelegt. So wurden die genutzten Brushes in die Datei AppBrushes.xaml

ausgelagert. Andere Styles, etwa aus TextBlockStyles.xaml, referenzieren diese Brushes und werden so automatisch mitangepasst, sollte eine Änderung vorgenommen werden.

Wo sinnvoll wurden die Styles mit systematisch vergebenen Namen gruppiert. Das Projektteam hat sich dabei an einem von Microsoft bereits seit Windows Phone 7 genutzten System orientiert (Microsoft Corp., 2013).

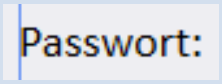
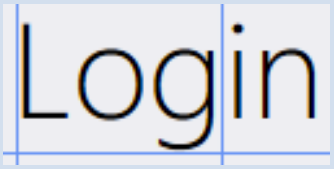
Name	Beschreibung	Bild
AppTextNormalStyle	Normaler Text	
AppTextSubtleStyle	Text im Hintergrund	
AppTextTitle1Style	Grosse Überschrift	

Tabelle 16: Gruppierung von Styles für TextBlock-Elemente

Es wurde darauf geachtet, dass in den Definitionen der Pages stets diese vordefinierten Styles genutzt werden, damit ein einheitliches Design in der ganzen App genutzt wird.

4.4.5 Navigation mit Pages

In Anlehnung an bestehende Projekte, welche mit MUI implementiert wurden (siehe Abschnitt 4.4.4.1), wurde für die Frontend App eine Page-basierte Navigationsstruktur umgesetzt. Dies gilt sowohl für die Hauptansicht, als auch für den Assistenten zur Operationsaufklärung.

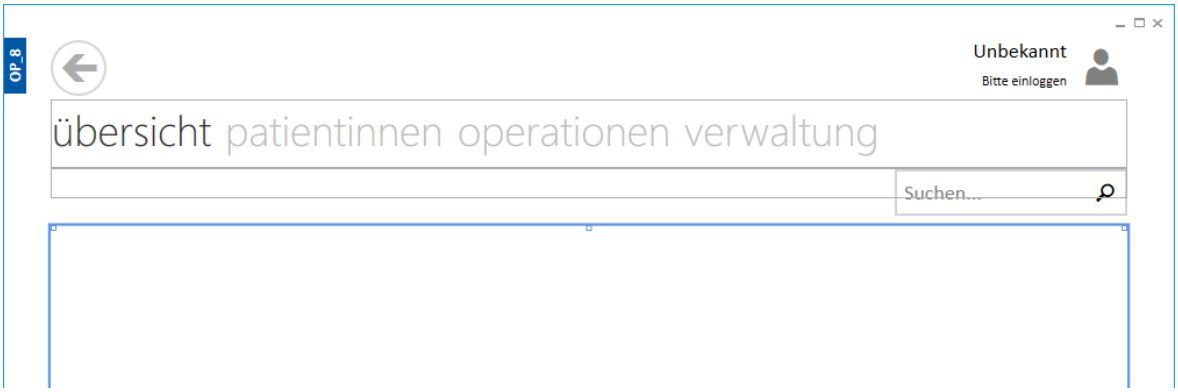


Abbildung 36: Navigationsstruktur im Hauptbereich (Ansicht im Visual Studio Designer)

In Abbildung 36 werden grob die wichtigsten UI Elemente für die Navigation nach dem Modell von MUI gezeigt, wie sie für das Projekt implementiert wurden:

- In der oberen Hälfte der Abbildung befindet sich das ModernMenu mit entsprechenden Links und LinkGroups.

- Die blau umrandete Fläche beinhaltet ein ModernFrame, in welchem die eigentlichen Pages dargestellt werden.
- Beim Fenster handelt es sich um ein ModernWindow mit angepasstem Style.

Die angezeigten Inhalte, sowie die gesamte Navigationsstruktur wurde dabei nicht hart codiert, sondern werden gemäss MVVM Pattern (siehe Abschnitte 4.4.1.1 und 4.4.5.1) über ein View Model zur Verfügung gestellt. Die Navigation zwischen den einzelnen Pages erfolgt über Commands oder im Fall der Hauptansicht über das ModernMenu mit Links und LinkGroups. Das zentrale ModernFrame ist jeweils mit einem Binding auf die ausgewählte Page – definiert als Property des View Models – verknüpft, welche bei der Navigation jeweils neu gesetzt wird.

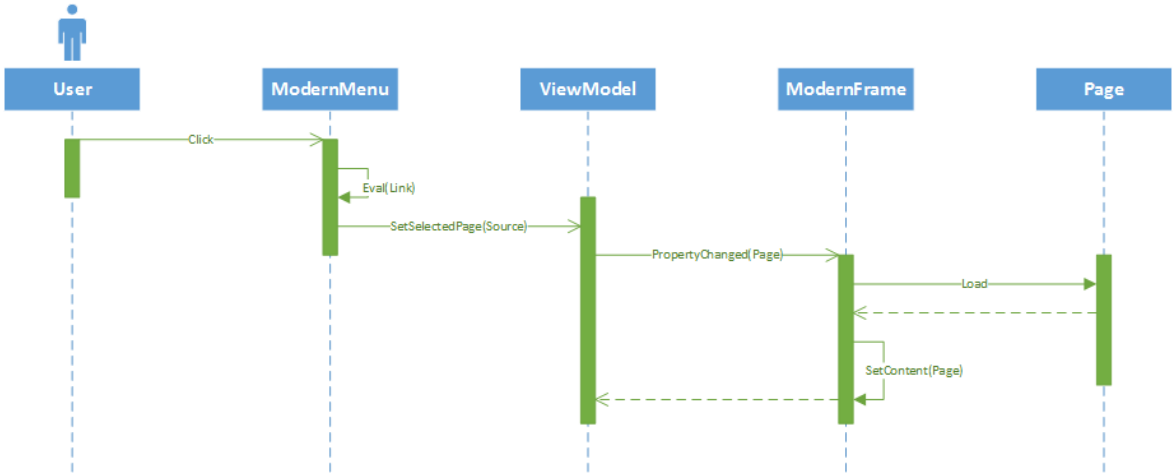


Abbildung 37: Ablauf einer Page-Navigation mit ModernMenu und ModernFrame

Das gleiche System für die Navigation kommt auch im Assistenten für die Operationsaufklärung zur Anwendung, mit dem Unterschied, dass dort kein ModernMenu angezeigt wird und die Navigation sequentiell durch die Schritte mit Buttons erfolgt.

4.4.5.1 Navigation View Model

Die Navigation zwischen den einzelnen Pages erfolgt ausschliesslich mit View Models. Für den Hauptbereich der App steht hierfür die Klasse NavigationViewModel zur Verfügung, welche als Dreh- und Angelpunkt fungiert.

Grundsätzlich unterscheidet sich diese Klasse nicht von anderen ViewModel, welche die Selektion eines einzelnen Elements aus einer Datenstruktur erlauben. Aus diesem Grund wird dieses View Model von den gleichen Basisklassen abgeleitet wie alle anderen. Bei den Elementen, welche ausgewählt werden können, handelt es sich um MUI Links (siehe Abschnitt 4.4.4.1).

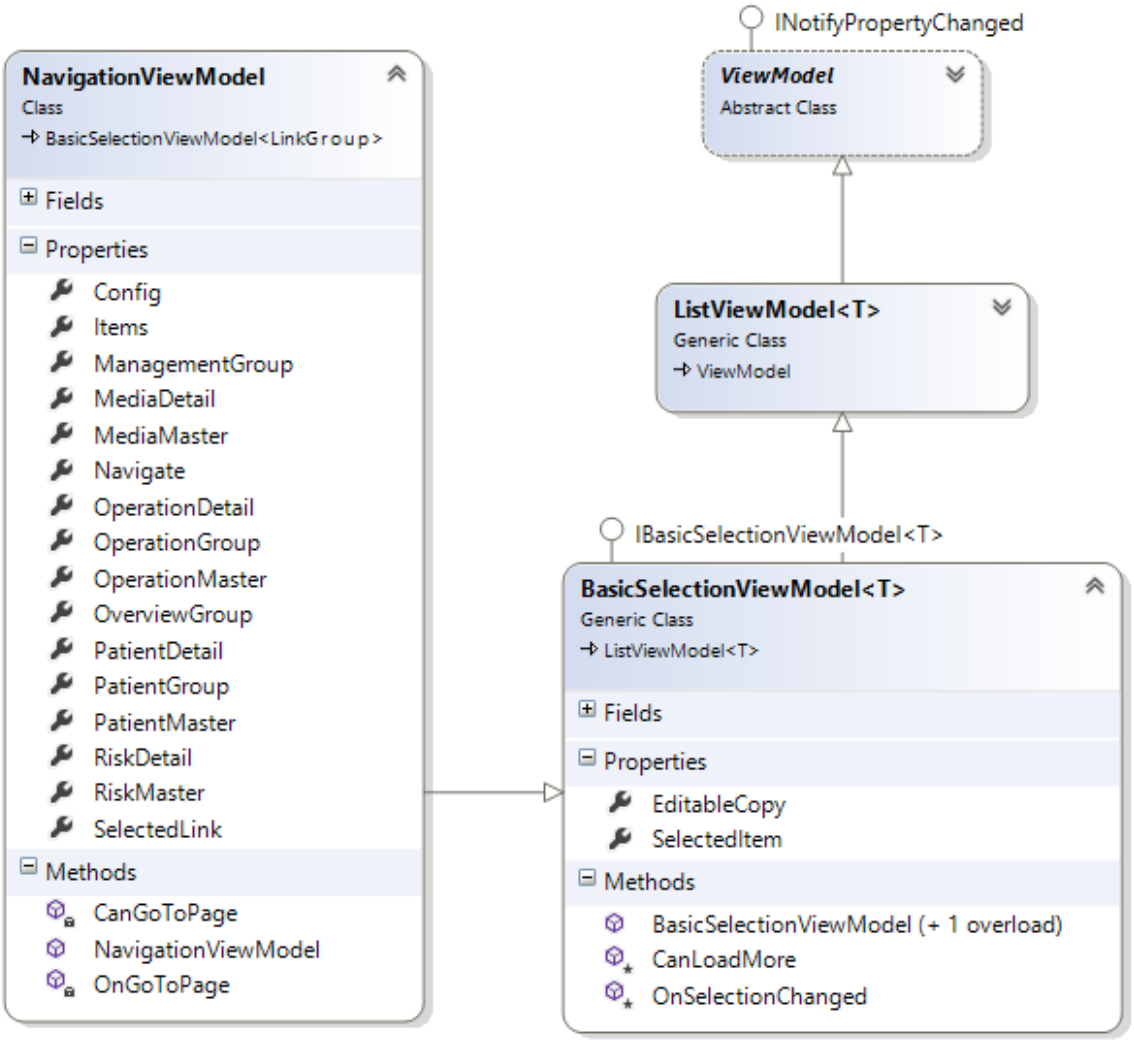


Abbildung 38: Klasse NavigationViewModel

Die Navigation zwischen den einzelnen Pages erfolgt indem das Property SelectedLink bzw. SelectedItem auf die gewünschte Page gesetzt wird. Das zur Anzeige der Inhalte genutzte ModernFrame ist per Binding mit diesem Property verknüpft (siehe Abschnitt 0) und aktualisiert seinen Inhalt automatisch auf die ausgewählte Seite. Im XAML kann eine solche Navigation bequem gebunden werden, indem das Command Navigate genutzt wird. Als Parameter wird dabei der gewünschte MUI Link bzw. die LinkGroup angegeben, welcher auf die XAML-Ressource mit der Definition der gewünschten Page verweist (siehe Abschnitt 4.4.5.3):

```
<Button Command="{Binding Path=Navigate}" CommandParameter="{Binding Path=PatientMaster}"
Grid.Column="0">
```

Für den Assistenten zur Operationsaufklärung ist eine vergleichbare Funktionalität direkt in der Klasse ConsentViewModel eingebunden, da die Navigation dort nicht frei, sondern in Abhängigkeit des Fortschritts des Aufklärungsprozesses erfolgt.

4.4.5.2 Container

Im MUI Sprachgebrauch und im Rahmen diese Projekts werden die Fenster, welche ein solches Navigation View Model für mehrere Pages und ein ModernFrame zur Verfügung stellen, als Container bezeichnet. Zum Zeitpunkt der Abgabe wurden im Projekt zwei Container implementiert:

- Der AppContainer stellt den Haupbereich zur Verfügung, wo alle Standardeingaben gemacht werden können.
- Der WizardContainer stellt den Aufklärungsassistenten zur Verfügung.

Bei allen anderen Fenstern handelt es sich nicht um Container im gleichen Sinn.

4.4.5.3 Umsetzung der Pages

Gemäss MUI Konvention werden alle Pages, welche innerhalb eines ModernFrames angezeigt werden sollen, nicht als WPF Pages, sondern als normale UserControls umgesetzt. Um besser mit den Navigationsevents umgehen zu können, hat sich das Projektteam dazu entschlossen, sämtlichen Pages die IContent-Schnittstelle einzuverleiben (siehe Abschnitt 4.4.4.1) und diese Grundkonfiguration als Basisklasse *ModernPage* für alle Pages zur Verfügung zu stellen.

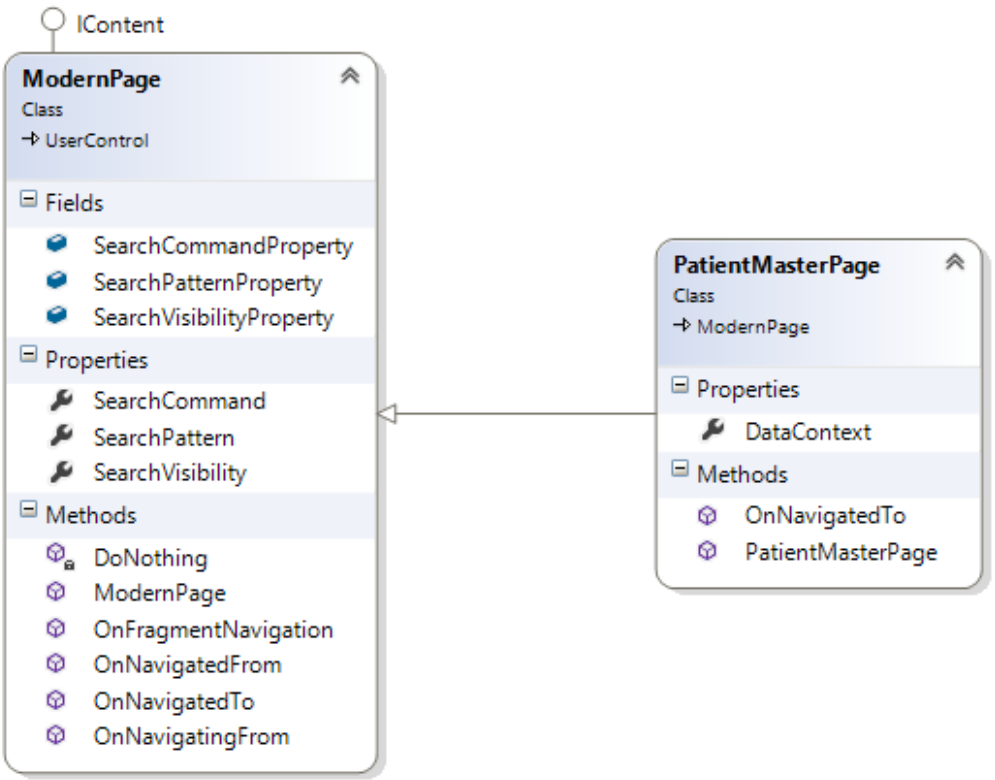


Abbildung 39: Klassen ModernPage und PatientMasterPage

Zusätzlich zu den vererbten Properties bietet ModernPage weitere Dependency Properties für eine mit den View Models integrierte Suchfunktion an, welche optional genutzt werden kann. Die Idee dahinter ist, dass der Container (siehe Abschnitt 4.4.5.2), welcher die Page enthält, eine Suchbox zur Verfügung stellt, welche auf diese Properties gebunden wird.

Dependency Property	Beschreibung
SearchCommandProperty	Publiziert das ICommand für die Suche für allfällige Bindings. Typischerweise wird das SearchCommand auf das Reset Command eines View Models gebunden.
SearchPatternProperty	Publiziert das Pattern für die Suche für allfällige Bindings. Typischerweise wird das SearchPattern auf das QueryString Property eines PagingContexts gebunden (siehe Abschnitt 4.4.2.3).
SearchVisibilityProperty	Publiziert die Steuerung, ob die Suche für die aktuelle Page für allfällige Bindings angezeigt werden soll.

Tabelle 17: Dependency Properties für Suche

Die Page selbst hat die Möglichkeit, die oben beschriebenen Properties auf das genutzte View Model zu binden, wodurch ein Hochreichen der entsprechenden Funktionalität an den Container ermöglicht wird. Benutzt man also die globale Suchbox, so wird jeweils automatisch die Suche der gerade aktiven Page bemüht, ohne dass die Page diese noch explizit ausprogrammieren muss.

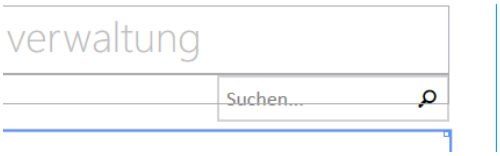


Abbildung 40: Globale Suchbox im Container

Die Verwendung von ModernPage als Basisklasse für UserControls wird vom Visual Studio Designer nicht direkt unterstützt und muss manuell eingetragen werden, wenn man eine neue Page definieren möchte. Die Schritte zur Erzeugung einer neuen Page sind wie folgt:

1. Hinzufügen eines neuen UserControls wie gewohnt.
2. Anpassung des Root-Elements im XAML-Code, so dass nicht ein generisches UserControl, sondern eine ModernPage definiert werden soll:

```
<nav:ModernPage x:Class="Eyecon.FlexApp.Navigation.Patient.PatientMasterPage"
```

3. Entfernen der expliziten Vererbung der generierten C# Klasse im Code-behind:

```
/// <summary>
/// Interaction logic for PatientMasterPage.xaml
/// </summary>
public partial class PatientMasterPage // : UserControl (entfernt)
```

Nach dieser einmaligen Umstellung funktioniert der Designer wie gewohnt und die Page kann wie ein normales UserControl weiterentwickelt werden.

4.4.5.4 Übergabe von Parametern

Das Navigationsmodell von MUI erlaubt nur eine eingeschränkte Form von Parameterübergaben zwischen den genutzten Pages unter Verwendung von URI Fragmenten, welche an die URI der jeweiligen XAML-Ressourcen angehängt und dort dann wieder ausgelesen werden können (über die Methode OnFragmentNavigation der Schnittstelle IContent). Dies stellt ein Problem dar für diverse

Navigationsmuster wie etwa dem Master-Detail-Drilldown. Die Übergabe des in der Master-Ansicht selektierten Datensatzes in die Detail-Ansicht kann so beispielsweise nicht umgesetzt werden.

Für die Übergabe von Parametern wurde deshalb überall ein LocatorService genutzt (siehe Abschnitt 4.4.1.3), welcher diese Verknüpfungen automatisch vornimmt.

4.4.6 Panorama und PanoramaItem

Für die Darstellung verschiedenster Inhalte in der App wird auf ein Panorama Control (Microsoft Corp., 2013) zurückgegriffen, welches Inhalte in PanoramaItems gruppiert und horizontal anordnet. Da WPF im Gegensatz zu WinRT (oder Silverlight auf Windows Phone 7) kein solches Control zur Verfügung stellt, musste das Projektteam nach einer Alternative suchen. Diverse Implementierung von Panorama Controls sind als Open Source verfügbar, unter anderem auch von MahApps.Metro, welches im Rahmen des Projekts bereits für andere Zwecke zum Einsatz kommt (siehe Abschnitt 4.4.4.2). Bei der Evaluation der Optionen für ein Panorama Control musste das Projektteam aber leider feststellen, dass keine der frei verfügbaren Varianten für eine Nutzung im Rahmen dieses Projekts geeignet ist und nicht dem Verhalten demjenigen der offiziellen Variante entspricht.

4.4.6.1 Eigenimplementierung

Aus diesem Grund wurde eine eigene Implementierung für das Panorama gebaut, wobei folgende Anforderungen massgeblich waren:

- Die Lösung sollte sich für den Entwickler so verhalten, wie die offiziellen Implementierungen für WinRT und Windows Phone. Zudem sollte sie auch vom Designer unterstützt werden.
- Der Inhalt des Panoramas sollte flexibel gestaltet werden können. Die meisten Open Source Lösungen unterstützen nur (Live) Tiles als Inhalt. Der Inhalt sollte sowohl statisch im Designer definiert, als auch mittels Bindings dazu geschaltet werden können.
- Das Panorama muss mit Touch bedienbar sein.
- Die Implementierung sollte so simpel wie möglich sein.

Herausgekommen ist ein Duo von Controls, wie sie auch bei WinRT und Windows Phone genutzt werden. Die Klassen Panorama und PanoramaItem stehen für das Panorama, respektive eine Gruppe innerhalb eines Panoramas.

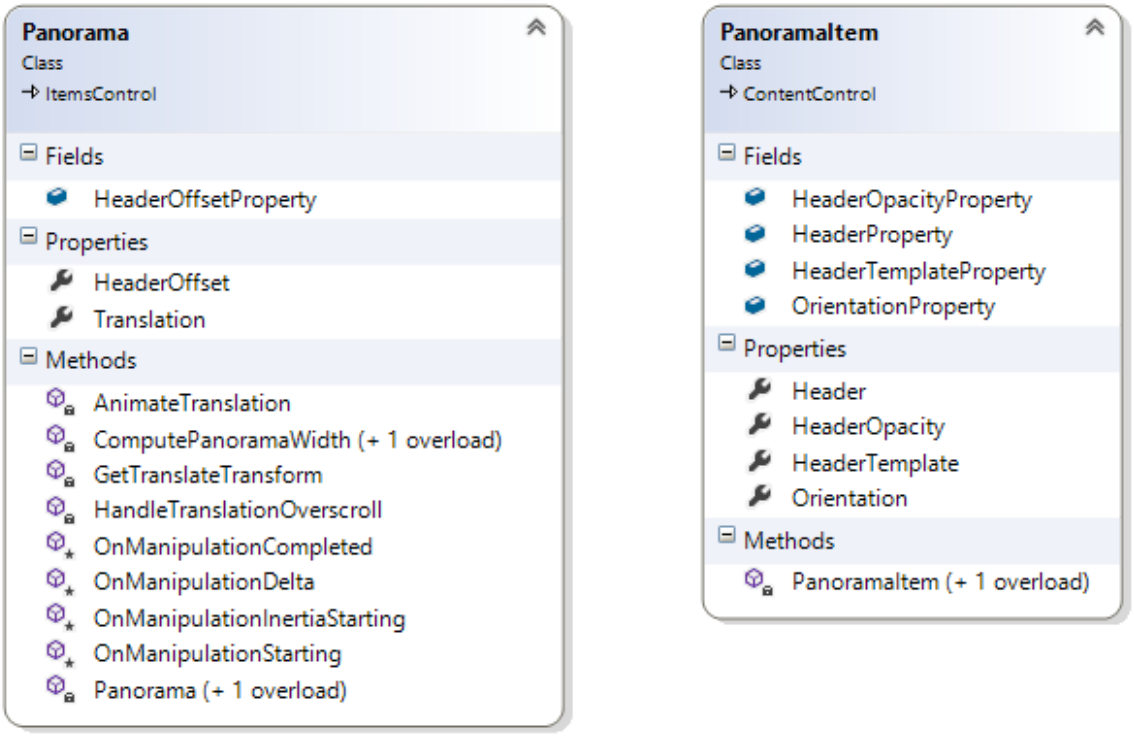


Abbildung 41: Klassen Panorama und PanoramaItem

Bei der Klasse Panorama handelt es sich um ein simples ItemsControl mit implizitem Styling (bekannte ItemsControls sind die ListBox, ComboBox oder das TabControl), dessen Items PanoramaItems sind, welche wiederum auf ein StackPanel mit horizontaler Orientierung abgebildet werden. Tatsächlich ist am Panorama Control nicht viel mehr dran, wie aus dem Default-Style ersichtlich wird:

```
    <Style TargetType="{x:Type local:Panorama}">
        <Setter Property="Background" Value="Transparent"/>
        <Setter Property="ItemsPanel">
            <Setter.Value>
                <ItemsPanelTemplate>
                    <StackPanel Orientation="Horizontal"/>
                </ItemsPanelTemplate>
            </Setter.Value>
        </Setter>
        <Setter Property="IsManipulationEnabled" Value="True"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type local:Panorama}">
                    <Border Margin="{Binding RelativeSource={RelativeSource TemplatedParent},
Path=Margin}"
                            Padding="{Binding RelativeSource={RelativeSource TemplatedParent},
Path=Padding}">
                        <ItemsPresenter/>
                    </Border>
                </ControlTemplate>
            </Setter.Value>
        </Setter>

    </Style>
```

Von besonderer Bedeutung im Default-Style ist das Property IsManipulationEnabled, welches für die intuitive Touchbehandlung gebraucht wird (siehe Abschnitt 4.4.6.3).

Das PanoramaItem, welches standardmässig innerhalb eines Panoramas zum Einsatz kommt, ist selbst ebenfalls nicht mehr als ein ContentControl (bekannte ContentControls sind der Button, beliebige UserControls oder die Klasse Window) mit einem impliziten Style und einem zusätzlichen Property Header, welches den Titel des Items speichert. Die Properties Title und Content (geerbt von ContentControl) können mit beliebigem Inhalt gefüllt werden, da die Darstellung über WPF ContentPresenter erfolgt, wodurch die Verwendung von Styles und Templates für diese Inhalte ermöglich wird. Typischerweise wird daher als Content ein Grid mit beliebigem Inhalt platziert.

Auf Basis dieser Grundkonfiguration wurde das typische Verhalten eines Panorama Controls nachgebaut.

4.4.6.2 Translation

Das horizontale Scrolling im Panorama wird durch das Property Translation realisiert, welches den entsprechenden Wert in der zugrundeliegenden Matrix verändert. Translation bedeutet dabei in etwa das, was man als relative X-Position *aller* PanoramaItems bezeichnen würde. Der Wert des Properties wird mit anderen Worten zur X-Koordinate der PanoramaItems addiert (tatsächlich wird dies über Matrizen realisiert) bevor man mit diesen interagieren kann. Um die Bedeutung der Translation zu veranschaulichen sei folgende Beschreibung gegeben:

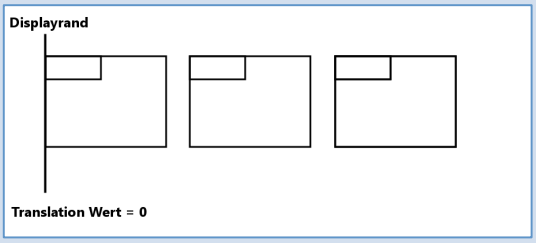
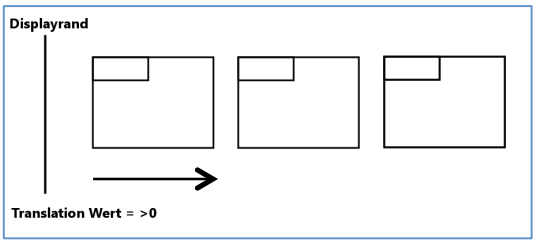
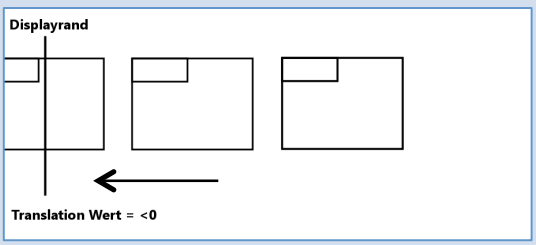
Wert der Translation	**Bedeutung**	**Bild**
0 (Standardwert)	Das Panorama befindet sich im Normalzustand. Das erste PanoramaItem befindet sich an Position 0.	
>0	Das Panorama verschiebt nach rechts und befindet sich im Overscrolling-Modus, da das erste PanoramaItem nicht mehr bei 0 steht und links davon keine weiteren PanoramaItems sind.	
<0	Das Panorama verschiebt sich nach links. Das erste PanoramaItem ist eventuell nicht mehr sichtbar, da es im negativen Bereich anfängt. Fängt auch das letzte PanoramaItem absolut im negativen Bereich an, so befindet sich das Panorama im Overscrolling-Modus.	

Tabelle 18: Bedeutung der Translation im Panorama

In der obigen Beschreibung ist die Rede vom Overscrolling-Modus. Dies bedeutet nichts anderes als, dass das Panorama über seine Items hinaus scrollt. Ist dies der Fall, so wird nach einem bestimmten

Zeitraum, in welchem keine weiteren Usereingaben getätigt wurden, automatisch zum letzten sinnvollen Wert für das Translation Property zurückgescrollt. So wird verhindert, dass die Translation aus Versehen zu grosse oder zu kleine Werte annimmt beim Scrolling durch den User. Dies hätte zur Folge, dass nichts mehr angezeigt würde und auch nicht mehr zurückgescrollt werden könnte.

4.4.6.3 Touch-Unterstützung mit Manipulation

Die Touch-Unterstützung für das Panorama wurde mit diversen Ansätzen ausgetestet und es hat sich herausgestellt, dass die Verwendung von Manipulation die intuitivste Variante darstellt. Mit WPF Manipulation stehen Entwicklern diverse high-level Events zur Verfügung, so dass diese sich nicht mehr um die einzelnen Touchpoints oder das Timing der Eingaben kümmern müssen, sondern direkt gegen die bekannten Gesten (z.B. Flick oder Pinch) programmieren können.

Charles Petzold fasst die wichtigsten Aspekte von Manipulation sehr gut zusammen (Petzold, 2010): WPF UI-Elemente, für welche das Property IsManipulationEnabled aktiviert wurde, erhalten automatisch eine erweiterte, intuitive Interpretation für Toucheingaben durch WPF selbst, welche dann entsprechende Events auf den jeweiligen Controls mit Bubbling auslöst. Dabei handelt es sich um folgende Events:

Event	Bedeutung
ManipulationStarting	Aufgerufen bevor eine logische Manipulation gestartet wird. Dieser Event ist von besonderer Bedeutung, da hiermit Manipulationen umgeleitet und parametrisiert werden können.
ManipulationStarted	Aufgerufen gleich nach dem Start der Manipulation.
ManipulationDelta	Aufgerufen jeweils für einen Manipulationsschritt durch den Dispatcher-Thread. Mit diesem Event wird die eigentlich Manipulation schrittweise auf ein Objekt angewendet. Dies gilt auch für Inertialmanipulation (siehe unten).
ManipulationInertiaStarted	Aufgerufen, wenn die Manipulation durch den User beendet wird, dass manipulierte Objekt aber aufgrund physikalischer Eigenschaften noch weiter manipuliert werden soll. Gängige Anwendung sind Effekte wie das langsame weiterrollen und drehen von Objekten nachdem der Benutzer diese losgelassen hat, wobei diese Effekte kontinuierlich abnehmen.
ManipulationBoundaryFeedback	Aufgerufen, wenn das manipulierte Objekt die Grenze eines anderen Visuals übertritt. Erlaubt die Implementierung von Visual Feedback.
ManipulationCompleted	Aufgerufen, nachdem eine Manipulation abgeschlossen wurde.

Tabelle 19: WPF Touch Manipulation Events

Für die vorliegende Panorama Implementierung wurde die Standardbehandlung der Events ManipulationStarting, ManipulationDelta, ManipulationInertiaStarted und ManipulationCompleted überschrieben und so angepasst, dass das Translation Property (siehe Abschnitt 4.4.6.2) anhand der Benutzereingaben manipuliert wird.

- Bewegt der Benutzer seine Touchpoints horizontal, so wird im ManipulationDelta Event ein entsprechender Wert addiert oder subtrahiert.
- Lässt der Benutzer seine Touchpoints los, so wird im ManipulationInertiaStarted Event eine Inertiakonfiguration vorgenommen, welche das ManipulationDelta Event anhand physikalischer Parameter über einen bestimmten Zeitraum weiter auslöst, was den bekannten Effekt des Weiterscrollens umsetzt.

Die Darstellung des Panorama Controls und der Translation wird mittels Bindings automatisch übernommen und muss nicht separat ausgelöst werden.

4.4.7 Message Service

Die App verwendet eine eigene MessageBox Variante, welche im Design dem Modern Style von Windows 8 entspricht, aber mehr oder weniger die Schnittstelle der WPF (und der älteren Windows Forms) MessageBox benutzt. Die MessageBox ist dabei als normales Window MessageDialog im Fullscreen-Modus umgesetzt, auf dem zusätzliche Dependency Properties auf entsprechende Controls gebunden wurden.

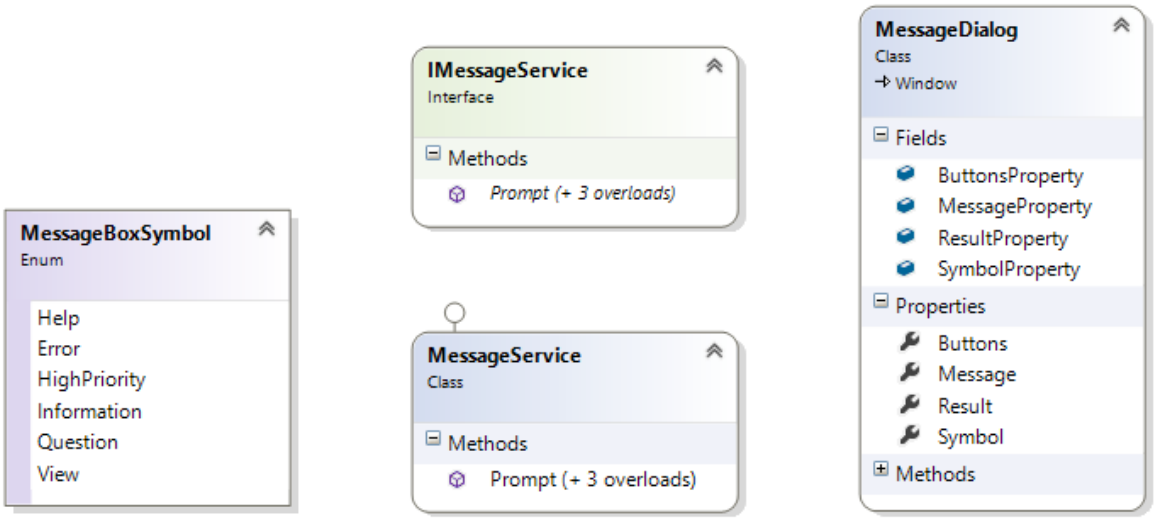


Abbildung 42: IMessageService, MessageService und MessageDialog

Anders als die traditionelle MessageBox Klasse unterstützt der MessageDialog keine MessageBoxImages, sondern setzt stattdessen auf MessageBoxSymbols. Ein entsprechender Enum MessageBoxSymbol stellt entsprechende Symbole zur Verfügung. Diese Symbole sind als Integer-Konstanten definiert, welche in ihrem Wert jeweils einem Zeichen aus dem Font Segoe UI Symbol entspricht (Microsoft Corp., 2013), welcher zur Darstellung im MessageDialog genutzt wird. Hierdurch wird erreicht, dass die verwendeten Symbole – die im Gegensatz zu den früheren Images vektorbasiert sind – stets scharf dargestellt werden und zum Modern Style passen (siehe Abschnitt 4.4.2).

Um den restlichen Code von der eigentlichen MessageBox zu entkoppeln, erfolgt der Aufruf der MessageBox über die Schnittstelle IMessageService, welche nur noch Überladungen der Methode Prompt anbietet. Die konkrete Implementierung in Form der Klasse MessageService leitet diese Aufrufe um auf eine jeweils neu erzeugte Instanz des MessageBox Windows, welches modal

angezeigt wird. Typischerweise werden interaktive Messages über den LocatorService (siehe Abschnitt 4.4.1.3) getätigt:

```
if (LocatorService.Resolve<IMessageService>().Prompt(
    string.Format("Sind Sie sicher, dass Sie das Risiko '{0}' löschen möchten?", this.Model.Name),
    "Risiko löschen",
    MessageBoxButton.YesNo,

    MessageBoxSymbol.Question) == MessageBoxResult.Yes)
```

Auf diese Weise könnten andere Implementierungen für eine MessageBox genutzt werden, ohne dass der Code überall angepasst werden müsste. An dieser Stelle sei auf die Unterstützung von Toasts als Alternative zur MessageBox verwiesen (siehe Abschnitt 4.4.9.2).

4.4.8 Umsetzung des Formulars

Das im Rahmen des Aufklärungsprozesses erstellte Formular wurde als FlowDocument umgesetzt und steht als Template in der entsprechenden XAML-Ressource AppDocuments.xaml zur Verfügung (siehe Abschnitte 4.3.2.6 und 4.4.4.5).

In der Tat handelt es sich bei diesem Template um nichts anderes als ein FlowDocument, dessen Struktur dem ursprünglichen Formular nachempfunden ist. Die Inhalte, welche auf dem Formular angezeigt werden, werden mit exakt denselben WPF Controls dargestellt, die auch im Assistenten zur Operationsaufklärung selbst genutzt werden. Die Einbindung der benötigten Daten erfolgt durch das Setzen des DataContext Properties des FlowDocument Objektes (was vom entsprechenden View Model automatisch gemacht wird). Da die Inhalte mit Bindings verknüpft sind, werden diese praktisch in Echtzeit aktualisiert.

Abbildung 43: Formulartemplate im Visual Studio Designer

Möchte man den Formularinhalt oder das Layout anpassen, so ist nur eine einmalige Anpassung des zugrundeliegenden XAML-Templates notwendig und die Änderung wird sofort in Designer, Vorschau und Export umgesetzt.

Würde sich das Universitätsspital Zürich dazu entschliessen, ein neues Logo zu benutzen, so könnte dieses einfach im XAML angepasst werden:

```
<Paragraph TextAlignment="Right">
    <Image HorizontalAlignment="Right" Source="/Eyecon/Resources/USZ_Logo.png" Width="240"/>
</Paragraph>
```

Das gleiche System könnte auch zur Definition weiterer Formulare genutzt werden, sollte hierfür ein Bedarf bestehen, da nur Standardkomponenten von WPF genutzt werden.

4.4.8.1 Export als PDF

Der Export des Formulars als PDF wurde über einen Umweg über das XPS Format realisiert, was aus Perspektive des Projektteams und aller beteiligten Parteien mehrere Vorteile bietet. So ist die

Entwicklung sehr viel einfacher als bei einer direkten PDF-Generierung und darüber hinaus auch noch kostenfrei. Der Prozess der PDF-Generierung kann daher wie folgt dargestellt werden:

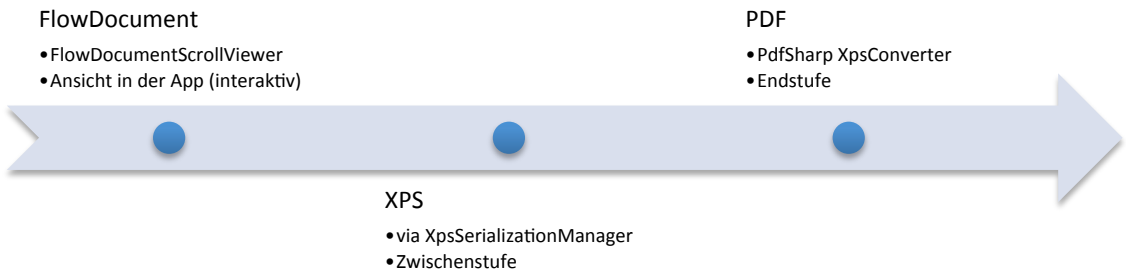


Abbildung 44: Prozess der Konvertierung von FlowDocument zu PDF

Windows Presentation Foundation unterstützt sowohl FlowDocument, als auch XPS nativ und kann daher FlowDocuments praktisch ohne Aufwand in XPS (de-)serialisieren. Die entsprechende Prozedur, welche ein FlowDocument im XPS-Format in einen Stream schreibt ist nur einige Zeilen lang:

```
        /// <summary>
        /// Converts the <see cref="FlowDocument"/> to an XPS stream.
        /// </summary>
        /// <param name="Source">The document.</param>
        /// <param name="Target">The stream.</param>
        private static void FlowDocumentToXPS(FlowDocument Source, MemoryStream Target)
        {
            using (var package = Package.Open(Target, FileMode.Create, FileAccess.ReadWrite))
            {
                using (var Document = new XpsDocument(package, CompressionOption.Normal))
                {
                    var Serializer = new XpsSerializationManager(new XpsPackagingPolicy(Document),
false);
                    var Paginator = ((IDocumentPaginatorSource)Source).DocumentPaginator;
                    Paginator.PageSize = new Size(Source.PageWidth, Source.PageHeight);
                    Serializer.SaveAsXaml(Paginator);
                    Serializer.Commit();
                }
            }
        }
```

Das Resultat, ein XPS Dokument, könnte theoretisch ebenfalls als Zielformat für den Export dienen, da es sich in der Verwendung nicht wesentlich von PDF unterscheidet und seit Windows Vista nativ unterstützt wird (und auf Windows XP kostenlos nachgerüstet werden kann).

Ausgehend vom in-memory XPS Dokument werden die Klassen XpsDocument und XpsConverter von PdfSharp (siehe Abschnitt 4.3.5.2) angesprochen, welche beliebige XPS Dokumente einlesen und in das PDF Format umwandeln können. Der entsprechende Code ist wiederum nur einige Zeilen lang:

```
        /// <summary>
        /// Saves the document as a PDF file to the specified location.
        /// </summary>
        /// <param name="FileName">The name of the file.</param>
        private void SaveAsPdf(string FileName)
        {
            using (var Stream = new MemoryStream())
            {
                FlowDocumentToXPS(this.GetDocument(), Stream);
                Stream.Position = 0;
                var Model = PdfSharp.Xps.XpsModel.XpsDocument.Open(
                    Stream);
```

```
            XpsConverter.Convert(Model, FileName, 0);
        }
    }
```

Die so entwickelte Umwandlung von FlowDocuments in das PDF Format unterstützt im Prinzip sämtliche in WPF definierten Visuals – also Controls, welche auf dem Bildschirm angezeigt werden können – weil diese vom XPS Serializer von .NET interpretiert werden können und ist somit extrem flexibel und qualitativ hochwertig.

Zur genutzten Library PdfSharp bleibt jedoch anzumerken, dass die neuen Versionen der verwendeten XpsConverter Klasse aus dem frei verfügbaren Paket entfernt wurden und nur noch gegen eine Lizenzierungsgebühr erhältlich sind. Die in diesem Projekt verwendete Version ist daher schon ein wenig älter, funktioniert aber problemlos.

4.4.8.2 Druck

Der Druck des FlowDocuments aus der App heraus erfolgt wie auch die PDF-Generierung (siehe Abschnitt 4.4.8.1) über eine Umwandlung in das XPS Format, wobei wieder die schon dort genutzte Funktion FlowDocumentToXPS genutzt wird. Das resultierende XPS Dokument kann mit dem normalen Druckdialog von WPF gedruckt werden.

4.4.9 Windows 8 Integration

Für die Integration in Windows 8 wurden das Projekt für die Frontend App gemäss Dokumentation parametrisiert, damit dieses neben der Unterstützung für das .NET Framework 4.5 auch Teile der Windows Runtime (siehe Abschnitt 4.1.1) konsumieren kann. Hierbei wurde das Build-Target auf Windows 8 gesetzt, was eine Unterstützung auf älteren Plattformen faktisch verunmöglicht.

Der genaue, offizielle Vorgang der Konfiguration eines Projekts in Visual Studio für die gleichzeitige Verwendung von .NET und WinRT beinhaltet zum Zeitpunkt der Entwicklung (wo bereits Visual Studio 2013 genutzt wird) leider noch immer die manuelle Editierung der Projektfiles (Marukovich, 2013), wo die Zielplattform als Erstes auf 8.0 hochgeschraubt werden muss (H., 2012).

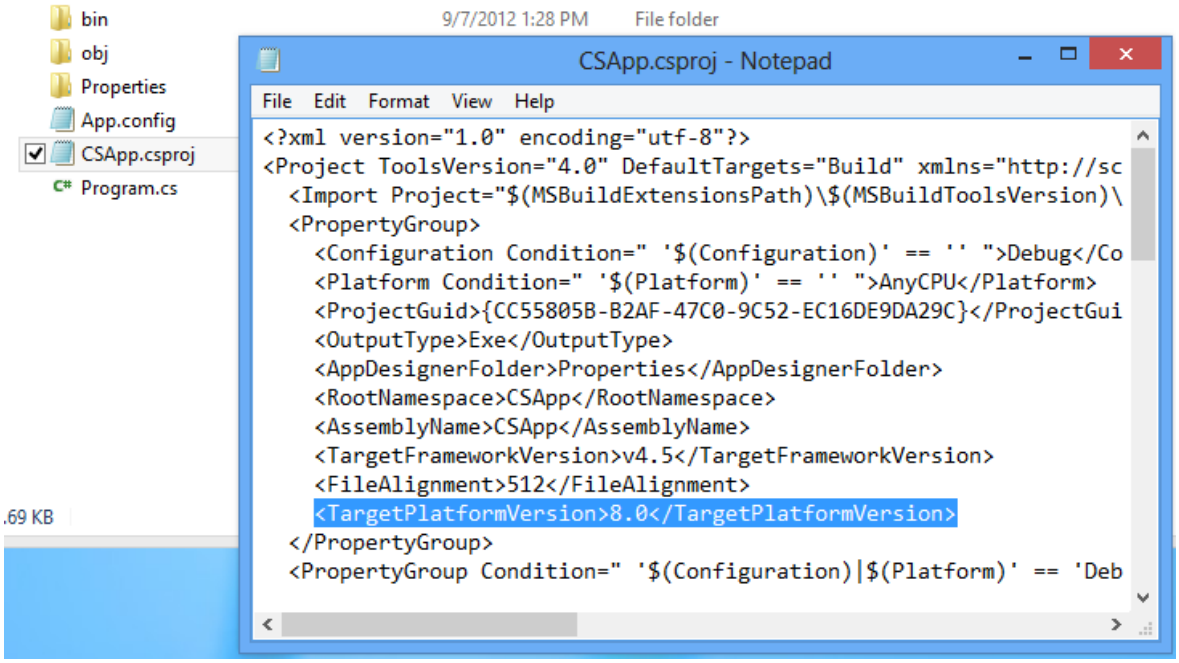


Abbildung 45: Manuelles Setzen der Zielplattform auf Windows 8 (H., 2012)

Nach dieser Umstellung stehen im Visual Studio für das Projekt auch die Windows Runtime Libraries zur Auswahl und können über den bekannten Dialog *Add Reference* zum Projekt hinzugefügt werden:

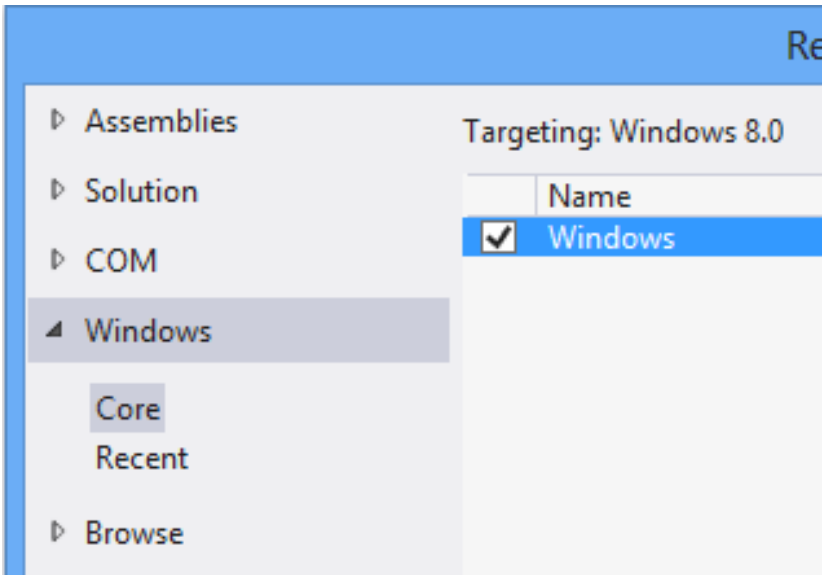


Abbildung 46: Referenzierung der Windows Runtime in Desktop Apps (H., 2012)

Zur Vereinfachung der Interoperabilität wurde auch noch ein Verweis auf die System.Runtime.WindowsRuntime.dll hinzugefügt, welche die nahtlose Verbindung zwischen den verschiedenen Typen aus dem .NET Framework und ihren Entsprechungen in der Windows Runtime sicherstellt (Microsoft Corp., 2013).

Zusätzlich zur Referenzierung der Windows Runtime wurde auch eine native Integration mit Hilfe des Windows API Code Packs eingebaut (siehe Abschnitt 4.3.5.2), da nicht alle Bestandteile von WinRT genutzt werden können.

4.4.9.1 Tile Unterstützung

Die Frontend App unterstützt die Platzierung eines Live Tiles auf dem Windows 8 Start Screen, sowie die Erzeugung eines App Shortcuts analog zu Windows Store Apps (siehe Abschnitt 4.1.1.1). Hierzu wurde im ModernStyleComponents die Klasse Deployment zur Verfügung gestellt, welche die entsprechenden Methoden TryCreateShortcut bzw. InstallShortcut anbietet.

4.4.9.2 Toast Service

Analog zum Message Service unterstützt die Frontend App auch die mit Windows 8 eingeführten Toasts über einen Toast Service.

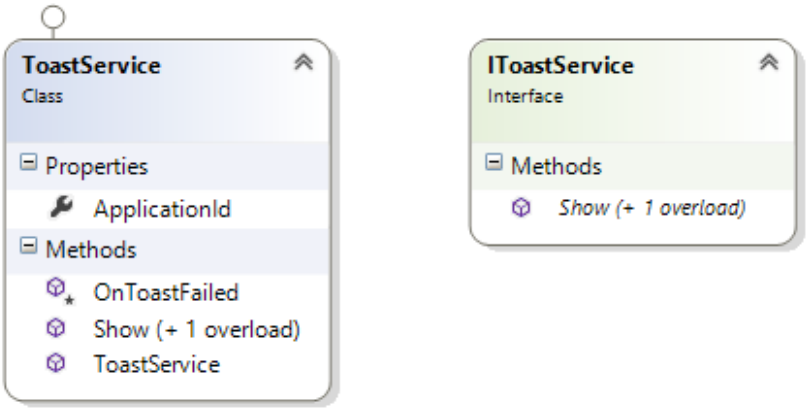


Abbildung 47: IToastService und ToastService

Um den Code vom eigentlichen Toast Framework von Windows 8 zu entkoppeln, erfolgen die Aufrufe stets über die Schnittstelle IToastService, welche nur noch Überladungen der Methode Show anbietet. Die konkrete Implementierung in Form der Klasse ToastService leitet diese Aufrufe um in das von WinRT zur Verfügung gestellt Toast Framework. Typischerweise werden Toasts über den LocatorService erzeugt:

```
LocatorService.Resolve<IToastService>().Show(
    string.Format("{0} ({1})", Strings.FileSynchronizationToastFoundFilesMessage, Syncing),
    Strings.FileSynchronizationToastFoundFilesToastTitle);
```

Auf diese Weise könnten andere Implementierungen für Toasts genutzt werden, ohne dass der Code überall angepasst werden müsste. Die Verwendung der Standardimplementierung der Toasts auf Windows 8 setzt die Existenz eines Live Tiles bzw. App Shortcuts voraus (siehe Abschnitt 4.4.9.1).

An dieser Stelle sei auch noch auf die interaktive MessageBox verwiesen, welche im Rahmen dieses Projekts implementiert wurde (siehe Abschnitt 4.4.7).

4.4.9.3 On-screen Keyboard

Windows 8 kommt mit einem neuen On-screen Keyboard, welches von allen Windows Store Apps implizit unterstützt wird und welches in der Frontend App genutzt wird (siehe Abschnitt 4.4.4.4).

Leider ist die Unterstützung für On-screen Keyboards sowohl in der Windows Presentation Foundation, als auch im Windows Runtime Subset für Desktop Apps ungenügend, weshalb eine eigene Lösung implementiert werden musste.

Gemäss Dokumentation von Microsoft (Microsoft Corp., 2013) kann der Anschlussstatus einer Tastatur mit der Windows Runtime Klasse KeyboardCapabilities über Property KeyboardPresent abgefragt werden. Diverse Quellen (Schulte, 2012) und auch die Erfahrungen bei der Entwicklung zeigen jedoch, dass dieses Property *immer* den Wert 1 zurückgibt und daher nutzlos ist. Aus zeitlichen Gründen und auch, weil die Auswirkungen auf die Benutzbarkeit der App minimal (wird eine physische Tastatur genutzt, so wird die virtuelle Tastatur automatisch sofort ausgeblendet) sind, wurde deshalb auf eine Tastaturdetektion verzichtet.

Für die Steuerung des On-screen Keyboards gibt es von Microsoft leider noch weniger Unterstützung als bei der Detektion; es gibt zurzeit keinen offiziell unterstützten Weg für eine Desktop App auf Basis von WPF die Kontrolle darüber zu erlangen, da das Focus-Tracking der Windows Runtime nur mit den nativen Controls von Windows 8 funktioniert (Microsoft Corp., 2013). WPF wiederum implementiert diese Controls selbst und bleibt damit aussen vor. Eine eventuelle (und eher theoretische) Möglichkeit zur Verknüpfung von WPF Controls mit dem nativen Focus-Tracking von Windows 8 wäre die Implementierung der IInputPanelConfiguration Schnittstelle (Microsoft Corp., 2013) mit C++ um das Tracking zu aktivieren. Im Anschluss müssten logische native Controls (Microsoft Corp., 2013) als Attached Properties (ebenfalls in C++ implementiert) an die WPF Steuerelemente angehängt und die Focus-Events darauf umgeleitet werden – eine sehr aufwendige Lösung für ein Problem, welches keines sein sollte. Die schlussendlich für das vorliegende Projekt umgesetzte Lösung erlaubt theoretisch die spätere Umstellung auf den oben beschriebenen Mechanismus, ist selbst jedoch wesentlich einfacher gestrickt und besteht aus zwei Klassen:

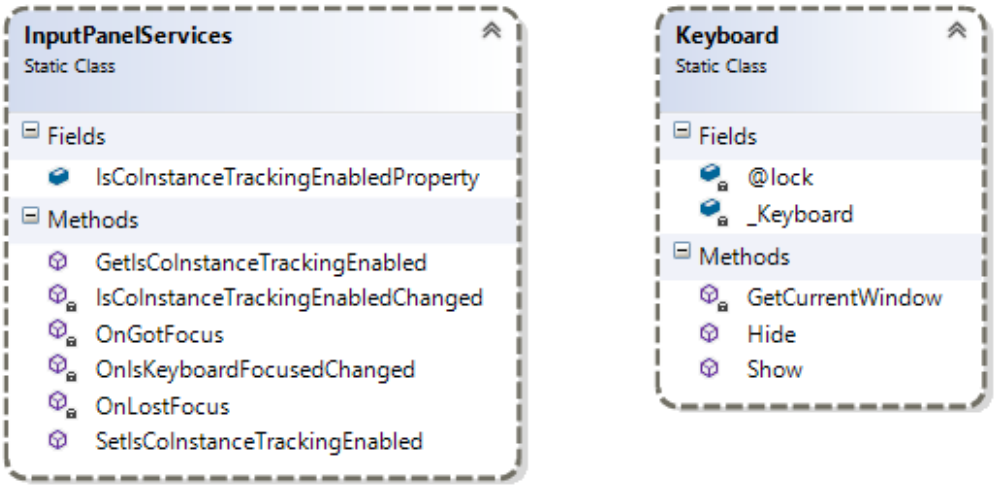


Abbildung 48: Klassen InputPanelServices und Keyboard

Die Klasse Keyboard bietet mit den Methoden Show und Hide die Möglichkeit an, das On-screen Keyboard anzuzeigen oder zu verstecken. Das Layouting der App wird dabei automatisch übernommen. Der Mechanismus zur Steuerung dieser Komponente ist unschön, funktioniert jedoch problemlos:

```
Process.Start(@"C:\Program Files\Common Files\Microsoft Shared\ink\TabTip.exe");
```

Der Start der virtuellen Tastatur erfolgt intern über den Start des TabTip-Prozesses. Die Beendigung erfolgt ganz ähnlich einfach durch Terminierung des Prozesses, in welchem die Tastatur gerade läuft.

Der automatische Aufruf dieser Methoden zum richtigen Zeitpunkt erfolgt mit Hilfe der Klasse InputPanelServices, die das Attached Property IsCoInstanceTrackingEnabled zur Verfügung stellt. Wird dieses für ein (beliebiges) WPF Steuerelement aktiv gesetzt, so wird ein erweitertes Focus-Tracking aktiviert, welches die Steuerung des On-screen Keyboards übernimmt:

```
<Setter Property="wrs:InputPanelServices.IsCoInstanceTrackingEnabled" Value="True"/>
```

Die Anwendung dieses Attached Properties erfolgt global durch implizite Styles (siehe Abschnitt 4.4.4.5) für alle relevanten Steuerelemente.

4.4.10 Internationalisierung

Die statischen Daten auf der Applikation werden mittels App Ressource Binding in Ressource Files abhängig nach Sprache in der Applikation lokal gespeichert.

Diese Ressourcen Files befinden sich im Ressourcen Folder „String“ des FlexApp Projekts.

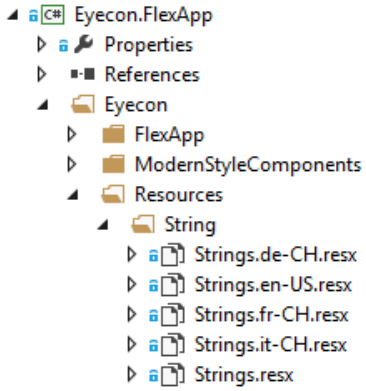


Abbildung 49: Ordnerstruktur Ressourcen Files

In diesem File wird jedem Ressourceneintrag ein Wert zugewiesen, welcher dann im UI angezeigt werden soll.

PatientDetailPanoramaPatientDetailPersonalDataTitle	Personalien
PatientDetailPanoramaPatientDetailTitle	Patientendaten
PatientDetailPanoramaPatientInterviewTitleButtonLabel	Patientengespräch
PatientDetailPanoramaPregnanciesHighRiskPregnanciesCheckBoxLabel	Hochrisikoschwangerschaft(en)

Abbildung 50: Eintrag Ressourcenfile

Im XAML der View wird im Root-Node der Pfad des Ressourcenfiles in einem Namespace gespeichert.

```
xmlns:resx="clr-namespace:Eyecon.Resources.String"
```

So lässt sich im betreffenden Control die String Ressource des entsprechenden String-Eintrages aufrufen.

```
<mod:PanoramaItem Header="{x:Static resx:Strings.PatientDetailPanoramaPatientDetailTitle}">
```

5 Qualitätssicherung

5.1 Kennzahlen

Damit die Qualität des Codes beschrieben werden kann, sind nachfolgend einige Kennzahlen der Solution aufgeführt und zur Veranschaulichung mit grafischen Elementen dargestellt.

5.1.1 Lines of Code

Das ganze Projekt (ohne Tests) umfasst 4101 Zeilen Code (gemeint sind bereinigte C# Codezeilen; XAML und andere Dateitypen sind hier nicht berücksichtigt), welche folgendermassen auf die verschiedenen Projekte in der Solution aufgeteilt sind:

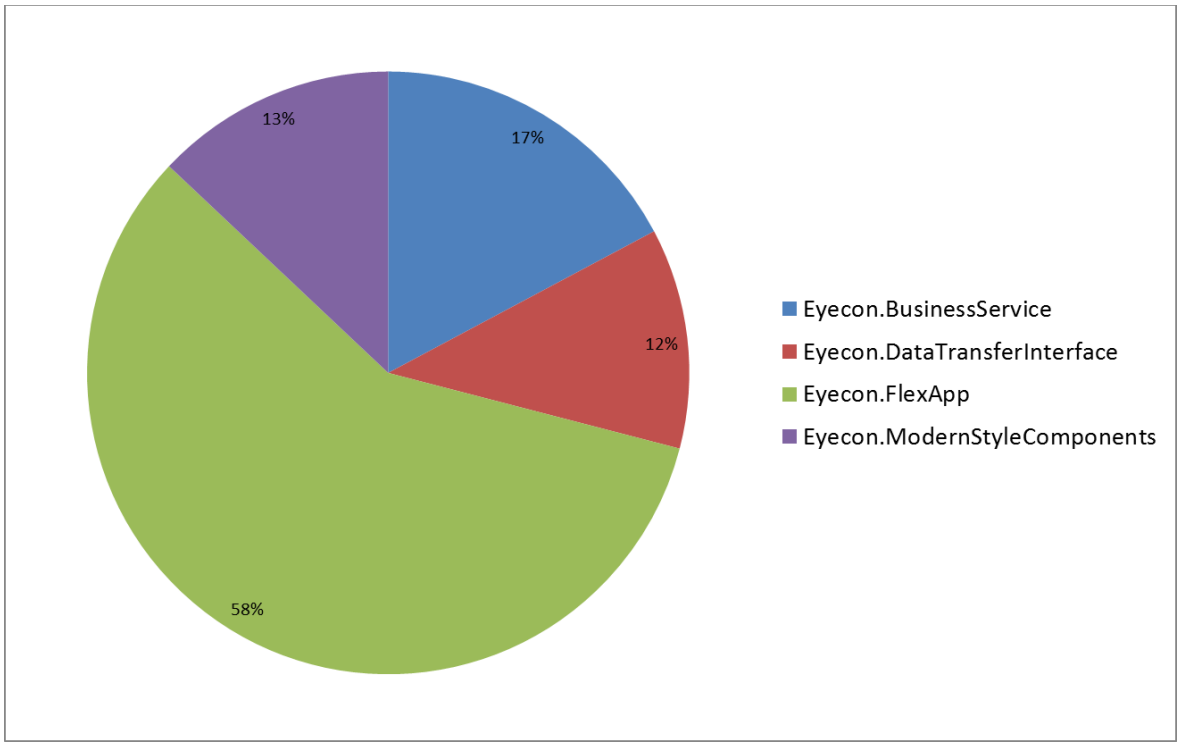


Abbildung 51: Diagramm Lines of Code

Das FlexApp Projekt ist mit 59 Prozent die zentrale Schicht der Applikation, da sie die ganze Logik in den verschiedenen ViewModels enthält.

5.1.2 Class Coupling

Misst die Kopplung an eindeutige Klassen durch Parameter, lokale Variablen, Rückgabetypen, Methodenaufrufe, generische oder Vorlageninstanziierungen, Basisklassen, Schnittstellenimplementierungen, für externe Typen definierte Felder sowie Attributdekorationen. Gute Softwareentwürfe zeichnen sich durch Typen und Methoden mit hoher Kohäsion und loser Kopplung aus. Eine enge Kopplung deutet auf einen Entwurf hin, der aufgrund zahlreicher gegenseitiger Abhängigkeiten zwischen anderen Typen nur schwer wiederzuverwenden und zu verwalten ist. (Microsoft Corp.)

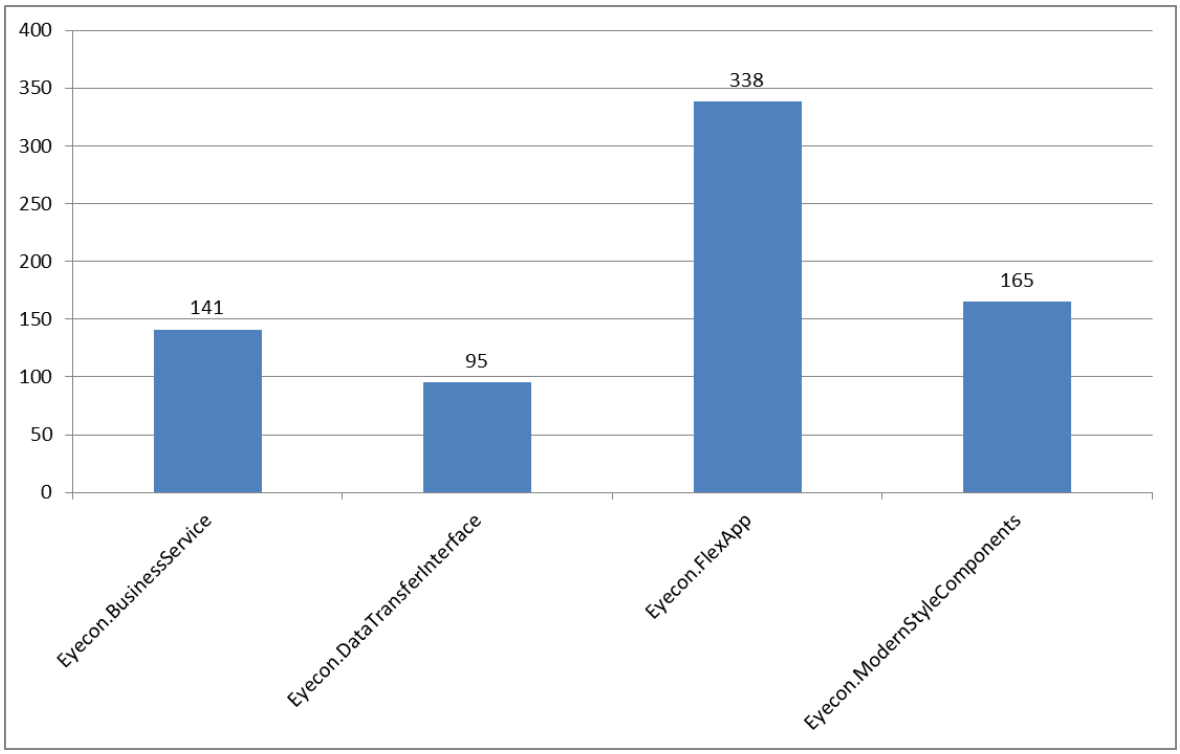


Abbildung 52: Diagramm Class Coupling

Aufgrund der Verwendung des ViewModel Locator konnte die ViewModels entkoppelt und somit eine losere Kopplung erzielt werden.

5.1.3 Cyclomatic Complexity

Misst die strukturelle Komplexität des Codes. Sie wird durch Berechnung der Anzahl unterschiedlicher Codepfade im Programmfluss erstellt. Für ein Programm mit komplexer Ablaufsteuerung sind mehr Komponententests erforderlich, wenn eine gute Codeabdeckung erzielt werden soll, zudem verschlechtert sich die Verwaltbarkeit. (Microsoft Corp.)

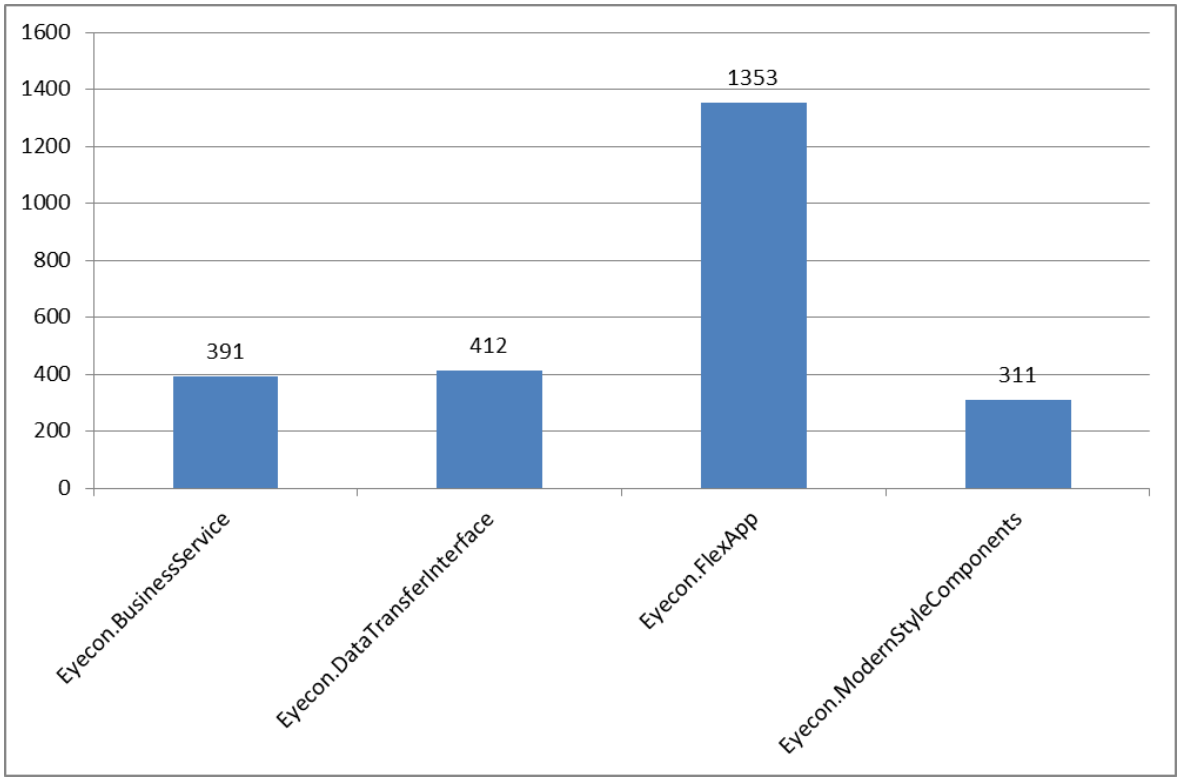


Abbildung 53: Diagramm Cyclomatic Complexity

5.1.4 Maintainability Index

Berechnet einen relativen Indexwert zwischen 0 und 100, der angibt, wie einfach der Code zu verwalten ist. Ein hoher Wert steht für bessere Verwaltbarkeit. Mit farbcodierten Bewertungen können problematische Stellen im Code schnell ermittelt werden. Eine grüne Bewertung liegt zwischen 20 und 100 und gibt an, dass der Code über eine gute Wartbarkeit verfügt. Eine gelbe Bewertung liegt zwischen 10 und 19 und gibt an, dass der Code über eine mäßige Wartbarkeit verfügt. Eine rote Bewertung liegt zwischen 0 und 9 und gibt eine niedrige Wartbarkeit an. (Microsoft Corp.)

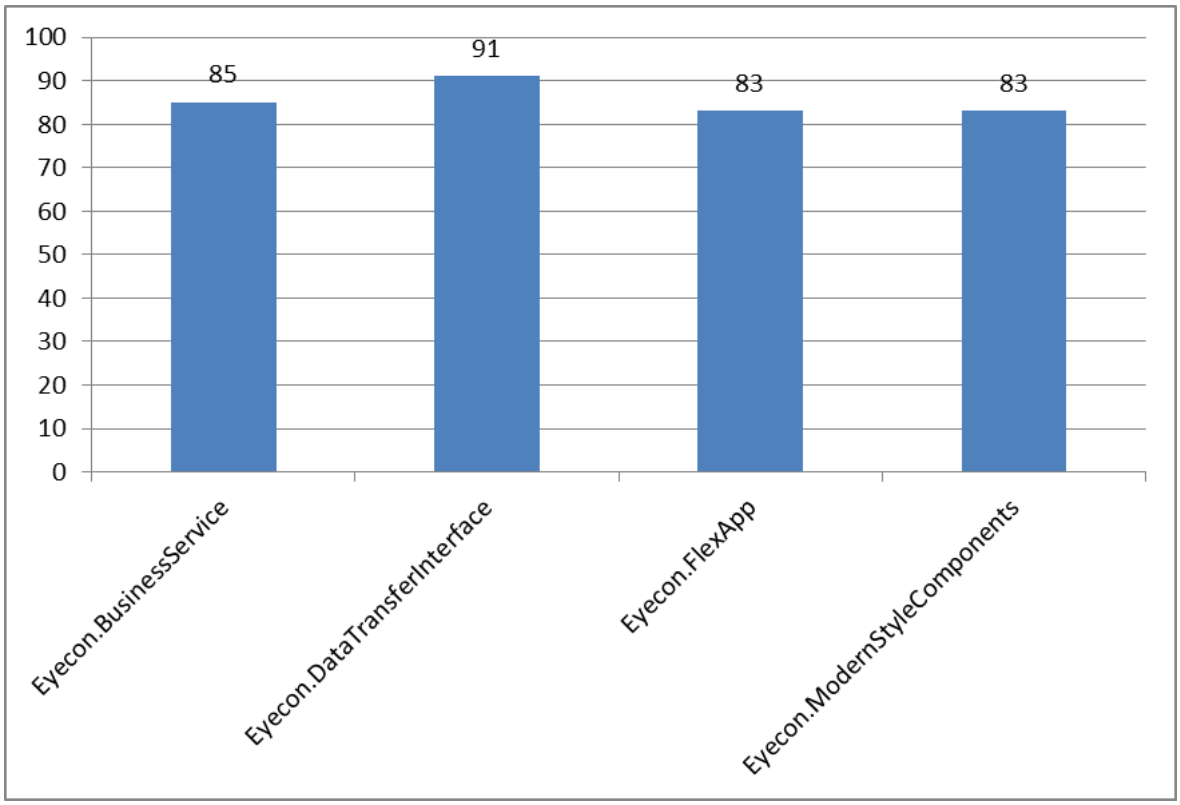
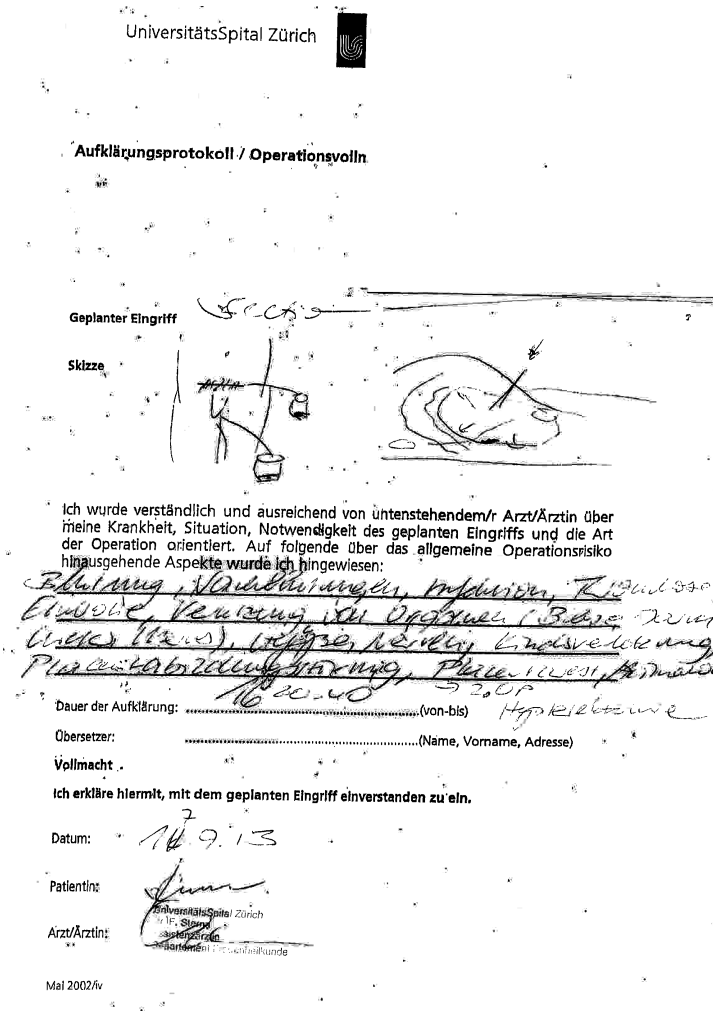


Abbildung 54: Diagramm Maintainability Index

6 Technischer Bericht

6.1 Ausgangslage

In der Klinik für Geburtshilfe im Universitätsspital Zürich wird vor jedem operativen Eingriff ein Aufklärungsgespräch mit der Patientin geführt. Bei diesem Prozess füllt der Arzt ein vorgedrucktes Formular aus, wo er die geplanten Eingriffe und die damit verbundenen Risiken beschreibt. Er erklärt der Patientin mündlich und anhand von Skizzen, wie die Operation ablaufen wird.



UniversitätsSpital Zürich

Aufklärungsprotokoll / Operationsvolln

Geplanter Eingriff

Skizze

Ich wurde verständlich und ausreichend von untenstehendem/r Arzt/Ärztin über meine Krankheit, Situation, Notwendigkeit des geplanten Eingriffs und die Art der Operation orientiert. Auf folgende über das allgemeine Operationsrisiko hinausgehende Aspekte wurde ich hingewiesen:

Dauer der Aufklärung: .....(von-bis)

Übersetzer: .....(Name, Vorname, Adresse)

Vollmacht .

Ich erkläre hiermit, mit dem geplanten Eingriff einverstanden zu ein.

Datum:

Patientin:

Arzt/Ärztin:

Mai 2002/iv

Abbildung 55: Papierformular als Ausgangslage

Die geplanten Eingriffe und jedes während der Aufklärung angesprochene Risiko muss auf dem Formular aufgeführt werden. In der Regel gibt es für jeden Operationstyp einen Grundstock an Risiken, welche jedes Mal von Hand aufgeschrieben werden müssen. Weitere Angaben sind „Datum“ und „Dauer der Aufklärung“ sowie ein optionales Feld „Übersetzer“, welche von Hand eingetragen werden.

Falls die Patientin mit den bevorstehenden Eingriffen einverstanden ist, wird am Ende des Prozesses das Formular von den beiden Parteien „Arzt/Ärztin“ und der „Patientin“ unterschrieben. Alternativ kann es auch vorkommen, dass die Patientin nicht einverstanden ist und der Prozess darin resultiert, dass kein Eingriff vorgenommen werden darf.

Mit der Unterschrift wird der Eingriff rechtsgültig autorisiert. Danach muss das Dokument von Hand eingescannt und archiviert werden.

Das Ziel der Bachelorarbeit ist die Erstellung eines Windows 8-Prototyp, der das Formular auf einem Tablet implementiert. Analog der Papiervariante soll der Prototyp die Patientin über die Operation und die Risiken informieren. Die Applikation soll das PDF der Operationsbeschreibung sowie multimediale Elemente zeigen, z.B. beschreibende Texte, Skizzen, Videos der Operation oder Fotos.

6.2 Lösungskonzept

Die mit dem Industriepartner ausgearbeitete Lösungsarchitektur teilt sich in zwei Bereiche auf:

- Client App für den Aufklärungsprozess und die Verwaltung der benötigten Daten
- Server Teil in Form eines Web Services

Die Umsetzung eines effektiven Assistenten zur Operationsaufklärung bedingt, dass die hierfür benötigten Daten (Operationstypen, Risiken, Medien) zum Zeitpunkt der Aufklärung bereits vorhanden sind, weshalb entsprechende Masken zur Verwaltung dieser Inhalte benötigt werden.

Abbildung 56: Patientenübersicht

Abbildung 57: Medienübersicht

Der Assistent soll dem Benutzer den Prozess der Operationsaufklärung so effizient und einfach wie möglich machen, indem jeder Schritt separat abgearbeitet werden kann, und die benötigten Informationen jeweils aus dem vorangegangenen Schritt hergeleitet werden. Aus der Wahl der Eingriffe, sowie der Patientin wird eine personalisierte Risikocheckliste hergeleitet, welche durch den Arzt abgearbeitet werden muss, bevor das Formular definitiv unterschrieben und abgeschlossen werden kann.

6.3 Umsetzung

In einem ersten Schritt wurden die Anforderungen an die Applikation beim Industriepartner abgeholt und in einer Anforderungsanalyse dokumentiert. Zudem musste abgeklärt werden, welche Art von Windows Applikation umgesetzt werden kann. Aufgrund hoher Lizenzkosten sowie allfälligen Einschränkungen wurde auf eine Umsetzung als Windows Store App verzichtet und stattdessen eine Windows Desktop Applikation geplant.

Im Anschluss wurde ein erster Prototyp erstellt, um den Prozess des Aufklärungsgespräches in der Applikation abzustecken und geplante Features, wie zum Beispiel das Unterschreiben auf dem Tablet oder das Exportieren von PDF-Dateien, zu testen. Da der Prozess eine Vielzahl an Informationen benötigt, wurde er in Form eines Assistenten umgesetzt, in welchem der Benutzer Schritt für Schritt die benötigten Eingaben vornehmen kann und dabei entsprechend unterstützt wird. Dieser Prototyp wurde dann dem Industriepartner präsentiert und validiert.

Da die Buttons und Eingabefelder in der Applikation noch viel zu klein waren, war die Bedienung mittels Touch-Eingaben auf dem Tablet noch nicht optimal. Deshalb wurden Papierprototypen erstellt um die Benutzbarkeit zu verbessern. Anhand dieser Skizzen wurde dann die Applikation umgestaltet und optimiert.

Schliesslich wurden die gewünschten Anforderungen des Industriepartners weiter umgesetzt und in Abständen von 2-3 Wochen im Universitätsspital präsentiert und besprochen.

6.4 Ergebnis

Das Ergebnis dieser Bachelorarbeit ist eine Windows Applikation, welche von der Klinik für Geburtshilfe zu einem späteren Zeitpunkt eingesetzt werden kann. Die Applikation kann sowohl auf dem Tablet für das Aufklärungsgespräch als auch auf dem Desktop Computer für die Erfassung von Daten genutzt werden.

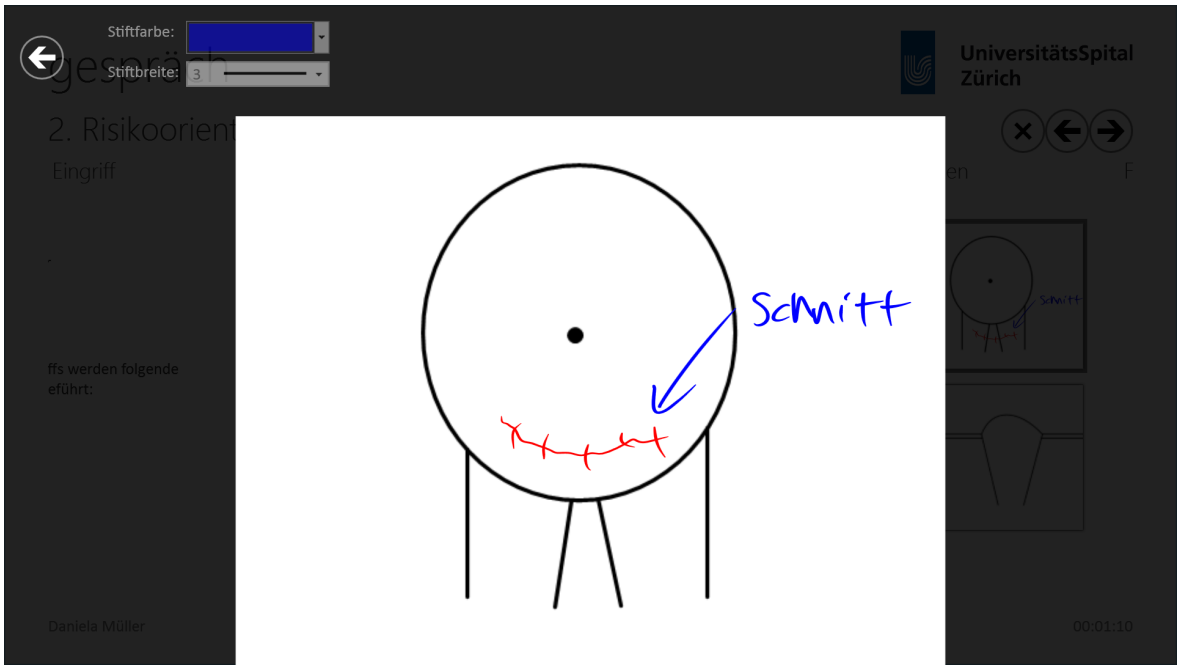
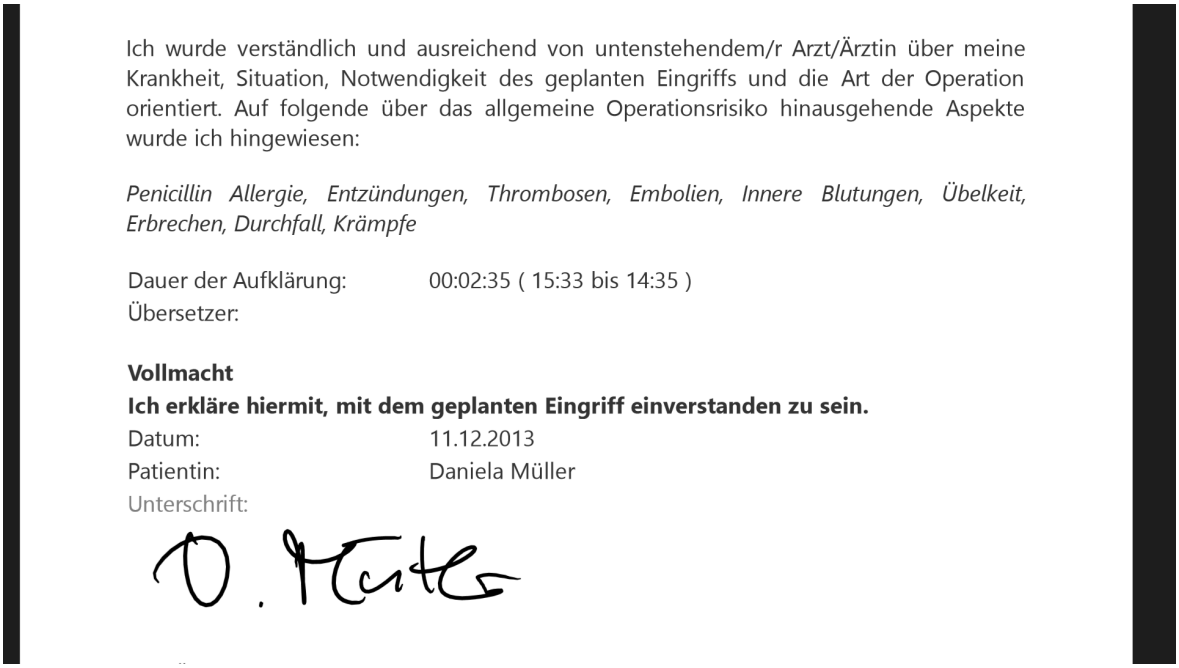


Abbildung 58: Skizzierfunktion

Um die Eingriffe der Patientin zu veranschaulichen, sind für die verschiedenen Operationstypen vordefinierte Skizzen hinterlegt, auf welchen mittels Stift zusätzliche Zeichnungen gemacht werden können. Zudem können auch Videos abgespielt werden. Damit der Arzt während der Aufklärung nicht vergisst über ein Risiko aufzuklären, müssen diese mittels Checkliste abgearbeitet werden. Zusätzliche Informationen wie Arzt oder Dauer der Aufklärung müssen nicht mehr manuell auf dem Formular eingetragen werden sondern werden automatisch aufgeführt. Das Formular kann vom Arzt und der Patientin direkt auf dem Tablet unterschrieben und dann mittels PDF-Export auf eine Datenablage kopiert werden.



Ich wurde verständlich und ausreichend von untenstehendem/r Arzt/Ärztin über meine Krankheit, Situation, Notwendigkeit des geplanten Eingriffs und die Art der Operation orientiert. Auf folgende über das allgemeine Operationsrisiko hinausgehende Aspekte wurde ich hingewiesen:

Penicillin Allergie, Entzündungen, Thrombosen, Embolien, Innere Blutungen, Übelkeit, Erbrechen, Durchfall, Krämpfe

Dauer der Aufklärung: 00:02:35 (15:33 bis 14:35)
Übersetzer:

Vollmacht
Ich erkläre hiermit, mit dem geplanten Eingriff einverstanden zu sein.
Datum: 11.12.2013
Patientin: Daniela Müller
Unterschrift:

Abbildung 59: Unterschriebenes Formular auf dem Tablet

Damit die benötigten Daten verwaltet werden können, wurden in der Applikation Verwaltungsmasken für die Patientinnen, die Operationstypen sowie für die Operationsrisiken umgesetzt.

6.5 Ausblick

Die Applikation soll nach Abschluss des Projektes im Universitätsspital mit PERINAT, dem bestehenden Informationssystem der Klinik für Geburtshilfe, verknüpft und getestet werden. Die Applikation soll ab dem nächsten Jahr produktiv zum Einsatz kommen und nimmt damit eine Vorreiterrolle bei der Einführung neuer EDV-Lösungen im medizinischen Umfeld des Universitätsspitals Zürich ein. Bei erfolgreichem Betrieb im produktiven Umfeld in der Klinik für Geburtshilfe kann ein universitätsspitalweiter Einsatz in Betracht gezogen werden.

Appendix A: Windows Whitepaper

A.1 Evolution von Windows

A.1.1 Windows 7 Editions und Architekturen

Windows 7 erreichte am 22. September 2009 den RTM-Status (LeBlanc, 2009) und wurde innert kurzer Zeit zum meistverbreiteten Windowssystem. Windows 7 wurde in sechs verschiedenen Editions veröffentlicht, von welchen nur vier für die kommenden Betrachtungen von Bedeutung sind: Home Premium, Professional, Enterprise und Ultimate. Enterprise und Ultimate sind von den Features her äquivalent und unterscheiden sich eigentlich nur noch im Lizenzierungsmodell. Alle Editions sind zu einander kompatibel indem sie auf der gleichen Zielarchitektur lauffähig sind – gemeint sind x86 und x64, wo eine weitgehende Aufwärtskompatibilität seit Jahren gewährleistet ist – und funktional jeweils eine Teilmenge der grösseren Editions bieten. Weiterhin gilt, dass alle Editions auf dem gleichen Kernel Windows NT 6.1 aufsetzen.

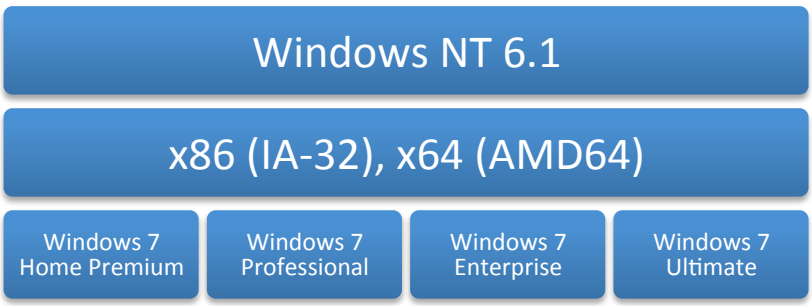


Abbildung 60: Windows 7 Editions und Architekturen

Windows Phone 7 basiert nicht auf Windows NT sondern auf einer speziellen Variante von Windows Embedded CE und ist daher in Abbildung 60 nicht aufgeführt.

A.1.2 Windows 8 Editions und Architekturen

Mit der Vorstellung von Windows 8 bzw. Windows NT 6.2 veränderte sich dieses einfache Bild, da mit ARM eine dritte Zielarchitektur Einzug hielt. ARM ist eine von ARM Holdings entwickelte und an diverse Halbleiterhersteller lizenzierte RISC Architektur.

Obwohl bereits seit Mitte der 1980er Jahre in vielen Kleinst- und Mobilgeräten eingesetzt, hat die zunehmende Verbreitung von PDAs, Mobiltelefonen die Bedeutung von ARM nochmals enorm gesteigert. Nach Jahren überproportionaler Entwicklungssprünge sind die neuesten ARM Chips heute leistungsmässig auf ähnlichem Niveau wie ultraportable Intel Designs, verbrauchen dabei aber oft weniger Energie, was sie für mobile Geräte optimal positioniert.

Anders als etwa zwischen x86 und x64 ist bei ARM *keinerlei Kompatibilität mit bisherigen Produkten gewährleistet*. Als Konsequenz davon sind nicht mehr alle Windows Editions auf allen Architekturen lauffähig. Sämtlicher Code für die neue Plattform muss von Grund auf neu geschrieben oder zumindest neu kompiliert werden. Dies gilt sogar für den Betriebssystemkern Windows NT 6.2/6.3.

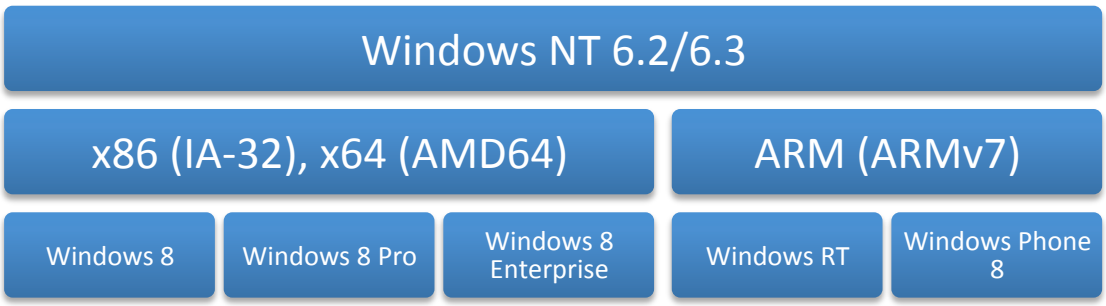


Abbildung 61: Windows NT 6.2/6.3 und Architekturen

Offensichtlich neu in Abbildung 61 erscheint daher die Architektur ARM, wo zwei neue Windows NT basierte Betriebssysteme ihren Platz einnehmen. Windows RT, welches mit diesem Namen wie Fremdkörper in der Windows 8 Welt wirkt, ist effektiv Windows 8 für ARM, wobei bei der Funktionalität einige signifikante Abstriche gemacht wurden (siehe Abschnitt A.2.1.2). Der Vollständigkeit halber ist auch Windows Phone 8 aufgelistet.

Zur ARM Architektur sollte noch angemerkt werden, dass dort mittelfristig die Ausweitung auf 64-bit ansteht, was das Bild noch einmal ein wenig komplexer macht. Wie bei der Einführung von x64 ist bei 64-bit ARM jedoch eine Kompatibilität zu bestehenden ARM Designs gegeben.

A.1.3 Nachfolger und Neuentwicklungen

Es soll nun kurz gezeigt werden, welche Windows 8 Editions welchen Windows 7 Editions entsprechen. Ebenfalls gezeigt werden Neuentwicklungen ohne Entsprechung.

Windows NT 6.1	Windows NT 6.2/6.3
Windows 7 Home Premium	Windows 8
Windows 7 Professional	Windows 8 Pro
Windows 7 Enterprise	Windows 8 Enterprise
Windows 7 Ultimate	-
-	Windows RT
-	Windows Phone 8

Tabelle 20: Windows 7 und seine Nachfolger

Wichtig zu verstehen ist hier, dass zwar Windows 8 und Windows RT technisch gesehen das gleiche Betriebssystem sind, Windows RT aber keinen Vorgänger hat und auf einer komplett anderen Architektur als Windows 8 läuft. Auch das Distributionsmodell von Windows RT unterscheidet sich massiv von dem von Windows 8. Alle Windows RT Lizenzen sind strikt an ein Gerät gebunden (Microsoft Corporation, 2013) und können nicht separat erworben werden. Auf die weiteren technischen Unterschiede wird später eingegangen.

A.2 Windows Terminologie

Mit Windows 8 und Windows RT hat sich die Benutzeroberfläche von Windows so radikal verändert, dass die seit Jahrzehnten gängigen Metaphern und Begriffe nicht mehr ausreichen oder nicht mehr benutzt werden können, um das System zu beschreiben (Niehus, 2013). Mit Metaphern gemeint sind traditionelle Begriffe wie *Desktop*, *Ordner* etc. welche ursprünglich aus anderen Kontexten in die Betriebssysteme übernommen wurden um neuen Benutzern die Orientierung so einfach wie möglich zu machen.

Die Einführung einer komplett neuen und abgetrennten, *Touch-First* Benutzeroberfläche (Niehus, 2013) in Windows liess viele dieser Metaphern hinter sich und setzt stattdessen auf neutrale Begriffe. In vielen Fällen wurden Aspekte der alten Umgebung sogar bewusst ganz weggelassen um die Bedienung zu vereinfachen oder weil sie als nicht mehr notwendig empfunden wurden. Was die Diskussion hierbei schwierig macht sind zwei Dinge:

- Den meisten Leuten mangelt es an Begriffen, um die verschiedenen Umgebungen und Benutzeroberflächen beim richtigen Namen zu nennen und auseinander zu halten.
- Die neue *Touch-First* Benutzeroberfläche ersetzt den alten Desktop nicht. Vielmehr ist es so, dass nun beide Oberflächen parallel existieren.

Aus dem zweiten Punkt folgt, dass man mit der Nomenklatur sehr vorsichtig sein muss. Begriffe aus der bisherigen Desktop Umgebung von Windows 7 sind unter Windows 8 weiterhin gültig und haben dieselbe Bedeutung. Zusätzlich dazu müssen für die neue Benutzeroberfläche neue Begriffe gefunden werden, welche eindeutig sind.

Sehr wichtig für das Verständnis ist auch, dass beide Oberflächen sowohl in Windows 8, als auch in Windows RT zur Verfügung stehen und prinzipiell überall über die gleichen Features und Einschränkungen aufweisen.

A.2.1 Windows Desktop und Desktop Apps

Der traditionelle Desktop von Windows Betriebssystemen (und auch anderen Betriebssystemen) ist in Windows 8 und Windows RT weiterhin vorhanden und wird nun als *Windows Desktop* bezeichnet. Mit Ausnahme des Startbuttons und Startmenüs wurden funktional alle Aspekte von Windows 7 übernommen. Mit Windows 8.1 wurde der Startbutton optisch wieder eingeführt; das alte Startmenü hingegen bleibt dauerhaft abgeschafft. Der neue Startbutton Wechselt zwischen der aktuell geöffneten *Windows Store App* (bzw. dem *Windows Desktop* (siehe Abschnitt A.2.1.3)) und dem *Start Screen* hin und her.

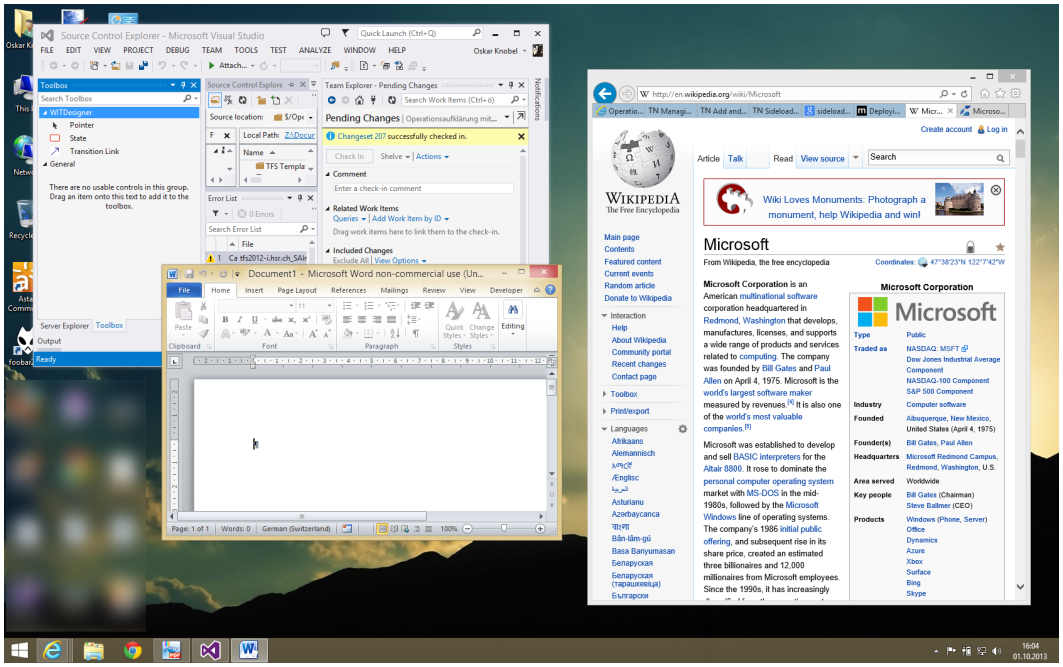


Abbildung 62: Windows 8.1 Desktop mit Desktop Apps

A.2.1.1 Programme und Applikationen als Desktop Apps

Alle gängigen Applikationen wie Windows Explorer, Internet Explorer, Microsoft Office oder das Control Panel sind auf dem *Windows Desktop* sowohl auf Windows 8 und Windows RT wie gewohnt verfügbar und können mit Tastatur und Maus oder Touch bedient werden. In der Terminologie hingegen wurden einige Anpassungen vorgenommen, welche kurz erklärt werden sollen.

Programme und Applikationen, welche auf dem traditionellen *Windows Desktop* (gezeigt in Abbildung 62) ausgeführt werden, werden von Microsoft neu als *Desktop Apps* (Microsoft Corporation, 2013) bezeichnet. So sind etwa Visual Studio oder Microsoft Word nach der neuen Terminologie *Desktop Apps*. Alle *Desktop Apps*, welche gerade ausgeführt werden, erscheinen in der *Windows Taskbar* (in Abbildung 62 am unteren Bildschirmrand sichtbar). Internet Explorer ist ebenfalls eine *Desktop App*, hat aber auch eine Entsprechung in der neuen Benutzeroberfläche, wo dann von einer *Windows Store App* (siehe Abschnitt A.2.2) gesprochen wird.

A.2.1.2 Einschränkungen auf Windows RT

Unter Windows RT unterliegen der *Windows Desktop* und die darin laufenden *Desktop Apps* strengen Restriktionen.

- Die Hardwarearchitektur auf welcher Windows RT betrieben wird basiert auf ARM (siehe Abschnitt A.1.2). Sämtliche Software, welche ausgeführt werden soll, muss komplett neu für ARM kompiliert werden. Diese Einschränkung gilt auch für neue *Windows Store Apps* (siehe Abschnitt A.2.2), wo allerdings mit .NET bzw. WinRT (Achtung: WinRT ist nicht Windows RT. Siehe Abschnitt A.3) eine Zwischenschicht für Portabilität sorgt, solange nicht mit C++ entwickelt wird.

- Viel signifikanter ist die Tatsache, dass nur Microsoft selbst die Werkzeuge zur Entwicklung von ARM *Desktop Apps* besitzt und nutzen kann. Es gibt für Drittentwickler keine offizielle Möglichkeit, für Windows RT solche Apps zu entwickeln oder zu verteilen.

Zusammenfassend kann gesagt werden, dass sämtliche *Desktop Apps* auf Windows RT von Microsoft selbst entwickelt werden müssen. Es gibt keine Möglichkeit zusätzliche *Desktop Apps* zu installieren ausser wenn diese über Windows Update verteilt werden.

A.2.1.3 Der Desktop als Windows Store App

Logisch gesehen ist der *Windows Desktop* seit Windows 8 bzw. Windows RT selbst nur noch eine *Windows Store App* (siehe Abschnitt A.2.2.3), in welcher die bereits seit Jahrzehnten etablierten Applikationen, die nun *Desktop Apps* genannt werden, ausgeführt werden können.

A.2.2 New Windows UI und Windows Store Apps

Parallel zum traditionellen *Windows Desktop* gibt es seit Windows 8 bzw. Windows RT eine neu eingeführte Benutzeroberfläche mit komplett neuem und inkompatiblem Entwicklungsmodell. Nach dem Start von Windows 8 bzw. Windows RT findet man sich neu nicht mehr auf dem *Windows Desktop* (siehe Abbildung 62), sondern auf dem *Start Screen* des *New Windows UI* wieder.

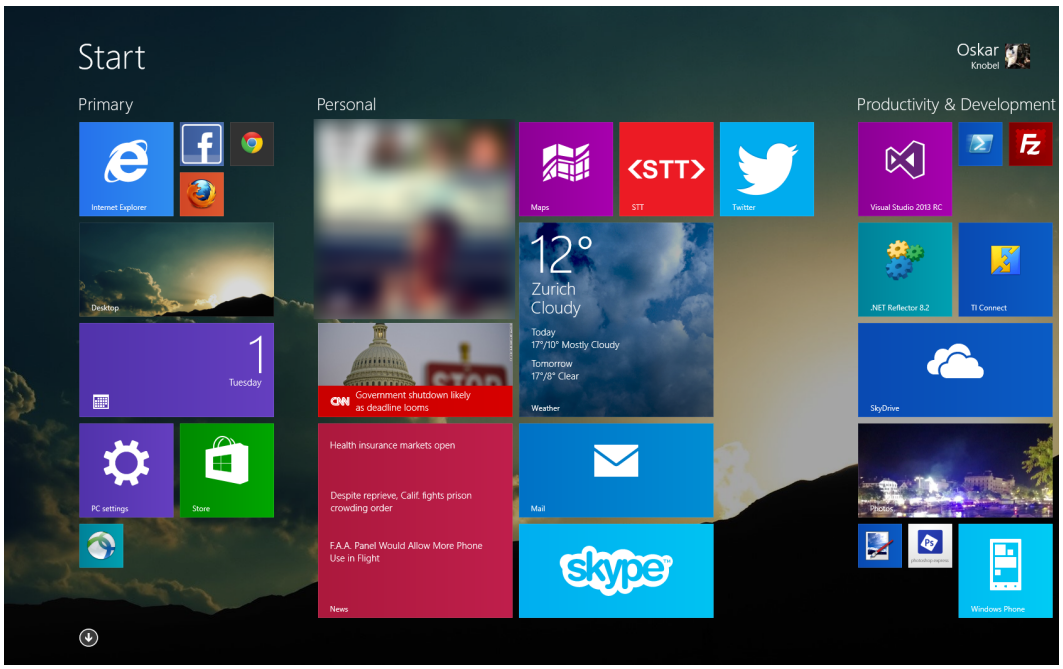


Abbildung 63: Windows 8.1 Start Screen mit Live Tiles

Der *Start Screen* ist nicht der *Windows Desktop* und ersetzt diesen auch nicht. Der *Start Screen* ersetzt stattdessen das alte Startmenü von Windows 7 durch eine neue Vollbildansicht mit aktiven Inhalten in einem Kachelsystem, den so genannten *Live Tiles*.

A.2.2.1 Start Screen und Live Tiles

Sowohl *Desktop Apps*, als auch *Windows Store Apps* können ihre eigenen Tiles auf den *Start Screen pinnen*. In Abbildung 63 sind beispielsweise die *Windows Store Apps* für News und Wetter gross

sichtbar in der Mitte. Ein Beispiel für ein Tile eines *Desktop Apps* ist Visual Studio 2013 am oberen rechten Bildrand. Auf dem *Start Screen* ebenfalls sichtbar ist ein *Live Tile* für den *Windows Desktop*. Dieser Umstand kommt daher, dass Windows 8 bzw. Windows RT den Windows Desktop als logische *Windows Store App* behandeln (siehe Abschnitt A.2.1.3).

A.2.2.2 Metro und seine Namenswechsel

Problematisch bei der Diskussion dieser Benutzeroberfläche ist, dass Microsoft im Verlauf der Entwicklung den Namen mehrmals geändert hat und selbst führende Microsoft Evangelisten die Begriffe immer wieder durcheinander bringen. Zu Beginn der Entwicklung von Windows 8 war überall die Rede von *Windows Metro style UI* (Niehus, 2013), wenn die neue Benutzeroberfläche gemeint war. *Metro* selbst war die Bezeichnung für die Design Language von Microsoft, welche einen besonderen Fokus auf klare Schriftsetzung und einfache grafische Formen legt. Die Bezeichnung wurde zuvor schon mehrere Jahre lang immer wieder im Zusammenhang mit Windows Phone und dem in Europa weitgehend unbekannten Zune gebraucht.

Ursprüngliche Bezeichnung	Aktuelle Bezeichnung	Bedeutung
Metro Design Language, kurz oft einfach *Metro Style* genannt	*Microsoft Design Language, Microsoft Design Style*	Eine Ansammlung von Richtlinien und Idealen für Design (Microsoft Corporation, 2013). Bekannt geworden mit Windows Phone 7.
Metro Desktop, Metro UI, Metro Windows, Windows Metro Style UI, ...	*New Windows UI, New Windows Experience*	Die Benutzeroberfläche, in welcher der Start Screen und die Windows Store Apps ablaufen. Explizit unterschieden wird hier vom *Windows Desktop*.
Metro App, Windows Metro Style App, ...	*Windows Store App* (Abschnitt A.2.2.3)	Apps, welche in der neuen Benutzeroberfläche ablaufen. Explizit davon ausgeschlossen sind traditionell Applikationen; diese werden als *Desktop Apps* bezeichnet.

Tabelle 21: Ursprüngliche und aktuelle Bezeichnungen für 'Metro'-artige Aspekte in Windows.

Aus rechtlichen Gründen war Microsoft kurz vor dem Release von Windows 8 dazu gezwungen, die Bezeichnung Metro komplett aus ihrem Repertoire zu streichen (Foley, 2012). Alle Referenzen dazu in Code, Dokumentation und den offiziellen Websites wurden hastig korrigiert und sprechen nun von *New Windows UI* oder *New Windows Experience* – Bezeichnungen, welche leider weit weniger einprägsam sind als das kurze *Metro*. Auch die Design Language wurde kurzerhand umbenannt und heisst seither offiziell *Microsoft Design Style*. In Quellen, welche nicht direkt von Microsoft stammen ist allerdings weiterhin oft von *Metro* die Rede, oder, was noch viel verwirrender ist, von unsinnigen Eigenkreationen wie *Windows 8 Metro Desktop* (svick, 2012) – im konkreten Fall war schlicht und ergreifend der *Start Screen* gemeint. *Metro* bzw. *New Windows UI* hat nichts mit dem *Windows Desktop* zu tun und läuft tatsächlich sogar komplett davon voneinander isoliert.

A.2.2.3 Windows Store Apps und Windows Store

New Windows UI auf Windows 8 bzw. Windows RT kommt einer eigenen Kategorie von Apps daher. Diese Apps werden als in Abgrenzung zu den *Desktop Apps* (siehe Abschnitt A.2.1.1) als *Windows Store Apps* bezeichnet und laufen in einem so genannten *App Container*, welcher nur innerhalb des *New Windows UI* sichtbar ist und die Ausführung für jede App komplett isoliert.

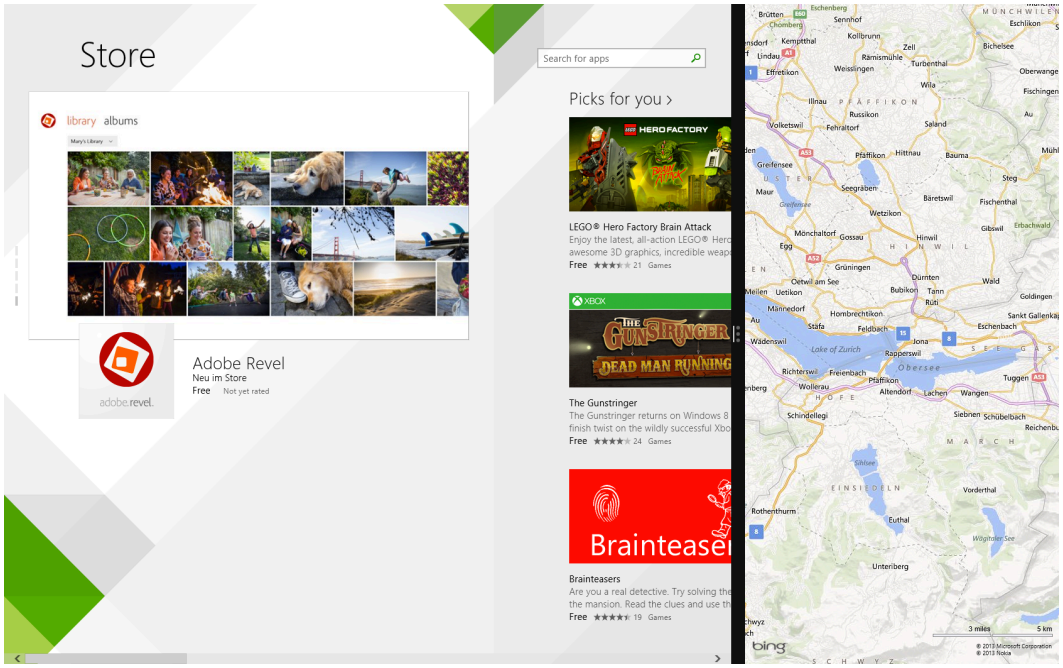


Abbildung 64: Windows 8.1 Store App und Maps App

In Abbildung 64 werden zwei *Windows Store Apps* parallel ausgeführt. Es handelt sich dabei um den *Windows Store* (der *Windows Store* ist eine *Windows Store App*, wo man andere *Windows Store Apps* suchen, downloaden oder bewerten kann) links und Maps rechts. Apps wie diese sind ausschliesslich im *New Windows UI* ausführbar.

Die Bezeichnung *Windows Store App* ist verwirrend, weil sie suggeriert, dass solche Apps zwingend etwas mit dem *Windows Store* zu tun haben müssten. Dem ist nicht so, obwohl die Standardverteilungsmethode für *Windows Store Apps* der *Windows Store* ist. *Windows Store Apps* können aber auch über andere Wege installiert werden (siehe Abschnitt A.4.1).

A.3 Entwicklung auf Windows 8/RT

In Abschnitt A.2.2 wurde angetönt, dass mit Windows 8 bzw. Windows RT ein neues Entwicklungsmodell eingeführt wurde, welches mit dem bisherigen Modell inkompatibel ist. Um noch einmal die Quintessenz dieser Diskussion zu rekapitulieren:

- *Desktop Apps* können nur auf dem *Windows Desktop* ausgeführt werden. *Desktop Apps* können nur für Windows 8, nicht aber für Windows RT oder Windows Phone 8 entwickelt werden.
- *Windows Store Apps* können nur im *New Windows UI* ausgeführt werden. Windows Store Apps können für Windows 8 und Windows RT entwickelt werden.

An dieser Stelle soll nun ein entwicklungstechnischer Überblick über die beiden oben genannten Modelle und ihren jeweiligen Einsatzort gegeben werden. Im vorliegenden Dokument wird dabei nicht auf die Entwicklung an sich eingegangen (hierfür sei auf die MSDN verwiesen), sondern vielmehr ein Blick aus der Vogelperspektive auf die genutzte Architektur darunter geworfen.

Das einzige öffentlich verfügbare offizielle Architekturdiagramm für Windows 8 stammt aus der BUILD Konferenz 2011 (Turner, 2012), lange bevor Windows 8 und Windows RT den RTM-Status erreicht hatten. Das damals vorgestellt Diagramm ist für eine ernsthafte Diskussion nicht geeignet, weil zu viele wichtige Aspekte nicht berücksichtigt wurden (z.B. wurde .NET schlicht vergessen) – vermutlich auch, weil sie damals noch gar nicht finalisiert waren. Anstelle von diesem Diagramm wird hier deshalb eine neuere, leider inoffizielle Variante von Rich Turner gezeigt (Turner, 2012). Diese Version ist ausführlicher und verdeutlicht die Zusammenhänge zwischen den verschiedenen Schichten im System viel besser.

Es sei angemerkt, dass das Diagramm nicht einfach 1:1 übernommen wurde, sondern noch an die neue Nomenklatur angepasst werden musste (siehe Abschnitt A.2.2.2).

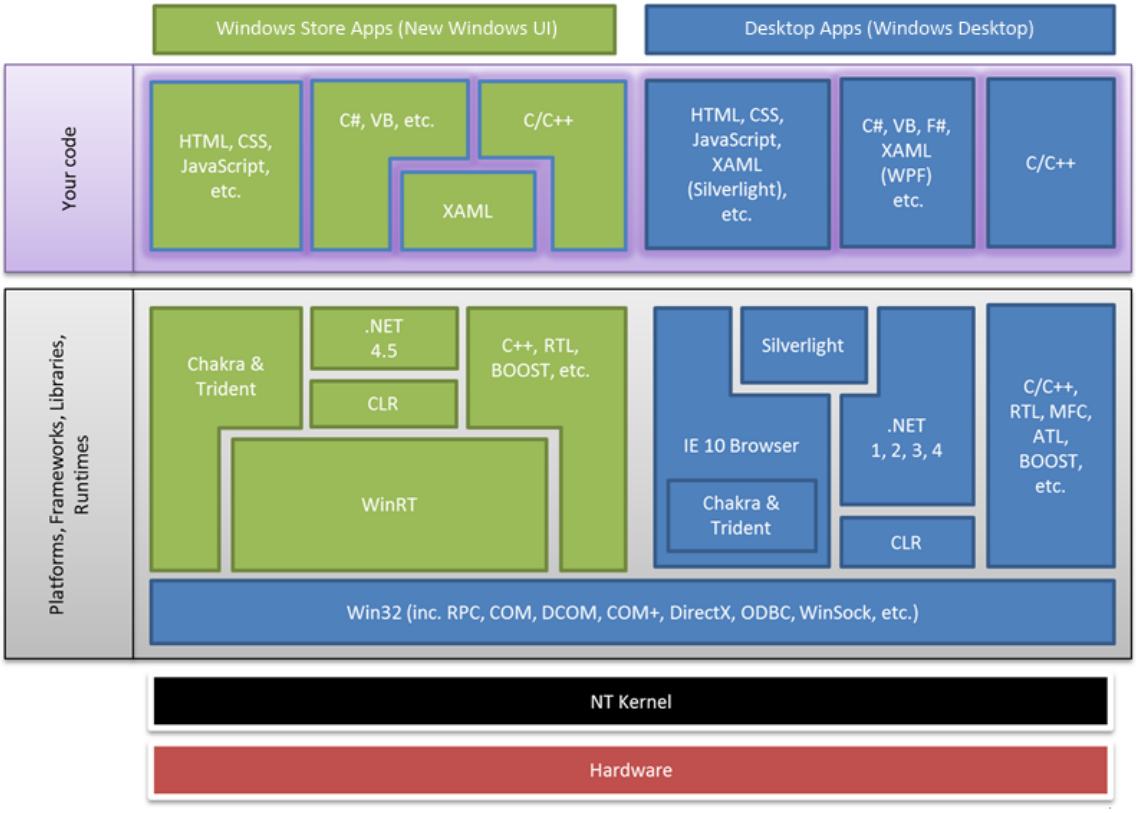


Abbildung 65: Windows 8 Architekturdiagramm von Rich Turner

Das vorliegende Diagramm ist sowohl für Windows 8, als auch für Windows RT und prinzipiell sogar für Windows Phone 8 gültig, wobei es in gewissen Schichten massive Unterschiede gibt. Die drittunterste Schicht mit der Bezeichnung Win32 fasst diverse Grundtechnologien zusammen und hat nicht unbedingt etwas mit 32-bit Windows oder der früheren WinAPI zu tun, kann diese aber sehr wohl beinhalten. Auf Windows RT (und Windows Phone 8) steht ein eingeschränkteres Set an Komponenten zur Verfügung als unter Windows 8.

Aus Entwickler- und Benutzersicht sind Windows 8 und Windows RT in zwei getrennte Systeme gespalten. Dabei handelt es sich um den traditionellen *Windows Desktop* mit den *Desktop Apps* (siehe Abschnitt A.2.1) und das *New Windows UI* mit den *Windows Store Apps* (siehe Abschnitt A.2.2). Dies spiegelt sich auch in Abbildung 65 wieder:

- Der linke, grüne Bereich beinhaltet die Architektur der *Windows Store Apps*.
- Der rechte, blaue Bereich bezieht sich auf die traditionellen *Desktop Apps*.
- Die untersten drei Schichten Hardware, NT Kernel und Win32 sind beiden Entwicklungsmodellen gemeinsam.

Die beiden Umgebungen bieten auf den ersten Blick scheinbar viele Gemeinsamkeiten. Befasst man sich aber genauer damit stellt man bald fest, dass sie jeweils auf einem komplett anderen Software Stack aufbauen – mit WinRT als Fundament oder eben nicht. Gerade bei Themen wie der UI Entwicklung findet man sich schnell zurecht und die Methodik ist die gleiche, doch die genutzten Frameworks sind jeweils nur auf einer der beiden Benutzeroberflächen lauffähig.

A.3.1 Windows Runtime und .NET Framework

Mit Windows 8 und Windows RT wurde ein neues Framework für die Entwicklung von *Windows Store Apps* eingeführt: Windows Runtime, kurz oft als WinRT bezeichnet. WinRT ist nicht dasselbe wie Windows RT. WinRT ist ein neuartiges Framework, welches auf x86/x64 und ARM Architekturen gleichermassen genutzt werden kann und damit die Portabilität von *Windows Store Apps* zwischen Windows 8 und Windows RT herstellt. Auch Windows Phone 8 nutzt eine Variante der Windows Runtime, welche hier aber nicht weiter betrachtet wird.

Anders als z.B. Silverlight oder das .NET Framework basiert WinRT nicht auf der *Common Language Runtime* und den Spezifikationen von .NET (in Abbildung 65 steht WinRT *unterhalb* der CLR und dem .NET Framework), sondern ist effektiv eine native, unverwaltete COM API (Icaza, 2011), welche sich aber designmässig stark an .NET orientiert. Darüber hinaus exportiert WinRT seine Metadaten nach dem ECMA 335 Standard (Icaza, 2011), welches auch von .NET genutzt wird. Als Konsequenz daraus wird WinRT von .NET Sprachen wie C# oder Visual Basic direkt nutzbar und muss nicht wie klassische COM/ActiveX Komponenten über einen Wrapper angesprochen werden. Auch der umgekehrte Fall, wo eine WinRT-Schnittstelle mit .NET Komponenten gefüttert wird setzt keine Wrapper mehr voraus.

Für den Entwickler stellt sich die Frage, mit was er nun wirklich entwickelt: .NET oder WinRT? Die Antwort hierauf ist nicht trivial. Tatsächlich werden beide Welten genutzt, wobei aber jeweils nur ein Teil davon verfügbar ist.

- Entwickelt man eine *Windows Store App*, so ist WinRT als Ganzes verfügbar, aber es kann nur ein Subset vom .NET Framework genutzt werden. Dieses Subset ist in etwa vergleichbar mit demjenigen von Silverlight.
- Entwickelt man eine *Desktop App*, so ist das .NET Framework als Ganzes verfügbar, aber es kann nur ein Subset von WinRT genutzt werden. Zusätzlich zu .NET können im Prinzip beliebige Softwarekomponenten genutzt werden, was eine enorme Flexibilität ermöglicht.

A.3.2 Entwicklung von Windows Store Apps

Bei der Entwicklung von Windows Store Apps kann auf die vollständige Windows Runtime zugegriffen werden. Vom .NET Framework ist hingegen nur ein Subset verfügbar, welches vom Umfang her ein wenig grösser ist als bei Silverlight für Windows Phone.

A.3.3 Entwicklung von Desktop Apps

Bei der Entwicklung von *Desktop Apps* auf Windows 8 kann grundsätzlich auf die gleichen Komponenten zugegriffen werden wie zuvor auf Windows 7. Darüber hinaus besteht aber auch die Möglichkeit, auf ein Subset der Komponenten der *Windows Runtime* zuzugreifen, um beispielsweise einen Sensor des Zielgeräts auszulesen.

Abschliessend bleibt anzumerken, dass Windows RT keine *Desktop Apps* von Drittentwicklern unterstützt (siehe Abschnitt A.2.1.2).

A.4 Windows Store App Deployment auf Windows 8 und Windows RT

A.4.1 Deployment von Windows Store Apps

Die Verteilung von *Windows Store Apps* auf Windows 8 und Windows RT unterliegt der Kontrolle und Lizenzierungsbedingungen von Microsoft. Diese Lizenzierungsbedingungen sind im *Volume Licensing reference guide Windows 8 and Windows RT* (Microsoft Corporation, 2013) beschrieben.

A.4.1.1 Veröffentlichung im Windows Store

Windows Store Apps (siehe Abschnitt A.2.2.3) werden hauptsächlich über den *Windows Store* veröffentlicht. Der Windows Store ist der standardmässige Kanal, über den Apps global oder regional allen Windows 8 und Windows RT Nutzern verfügbar gemacht werden können.

Möchte man als Entwickler seine Windows Store App über den Windows Store verfügbar machen, so muss man diese bei Microsoft zu einem vordefinierten Publishing-Prozess einreichen. Die App wird dann auf verschiedene Arte getestet, was in der Regel eine Woche dauert. Der gleiche Prozess gilt auch für App Updates, welche wieder neu getestet werden.

Wurden alle Tests erfolgreich durchlaufen, kann der Entwickler seine *Windows Store App* bzw. eine Update für ein solches veröffentlichen. Die Veröffentlichung erscheint dann innerhalb eines Tages im *Windows Store* und kann von potentiellen Kunden konsumiert werden.

A.4.1.2 Enterprise Sideloading

Für mittelgrosse und grosse Unternehmen bietet Microsoft die Möglichkeit des *Enterprise Sideloading* von *Windows Store Apps* an (Microsoft Corporation, 2013). Wie der Name bereits impliziert ist dieser Mechanismus nicht für kleine Unternehmen oder Einzelentwicklungen gedacht und sollte auch nicht zur Verteilung von Apps an externe Kunden genutzt werden.

Der Sideloading Mechanismus ist von der Windows Lizenz abhängig. Windows RT, Windows 8 und Windows 8 Pro müssen mit einem zusätzlich zu erwerbenden Product Key ausgestattet werden,

damit Sideloading von *Windows Store Apps* möglich wird (Microsoft Corporation, 2013). Bei Windows 8 Enterprise ist das Sideloading Teil des standardmässig enthaltenen Features, wobei die Geräte allerdings in einer Domäne registriert und mit einem speziell konfigurierten Windows Server 2012 administriert werden müssen.

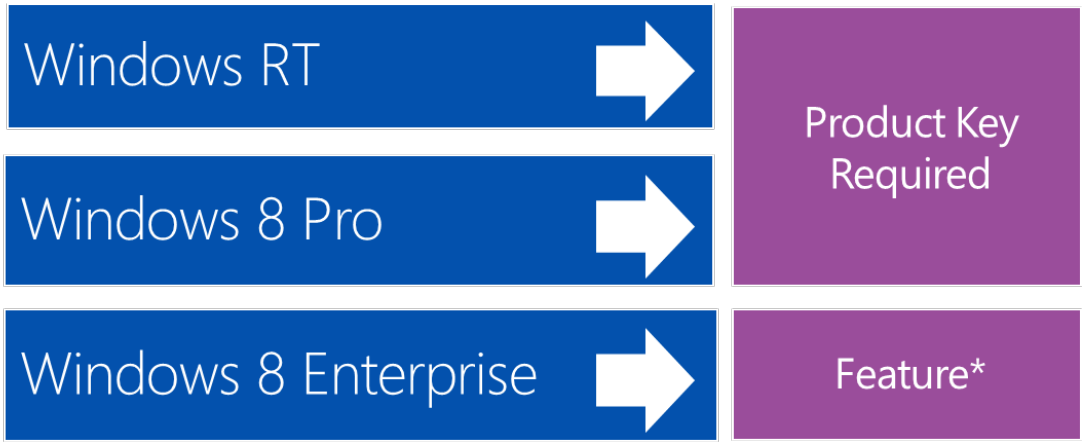


Abbildung 66: Lizenzierung von Sideloading auf Windows 8 und Windows RT

Es ist anzumerken, dass Windows 8 Enterprise nur für Grosskunden verfügbar ist und nicht regulär erworben werden kann. Für die weitere Betrachtung von Windows 8 und Windows RT ist dieses daher nicht mehr relevant.

A.4.2 Deployment von Desktop Apps

Die Verteilung von traditionellen *Desktop Apps* unterliegt keinen Restriktionen und erfolgt auf Windows 8 wie gewohnt durch einen Installer oder durch Kopieren. Touch-Support ist durch .NET 4.5 bzw. WPF gegeben; es stellt sich jedoch die Frage nach geeigneten Controls und Styles.

Diese Variante funktioniert nur auf Windows 8; Windows RT erlaubt die Installation von normalen Applikationen aus diversen Gründen nicht (siehe Abschnitte A.2.1.2 und A.3.3).

Appendix B: Glossar

Begriff	Erklärung
.NET Framework	Das .NET Framework ist eine von Microsoft entwickelte Plattform zur Entwicklung und Ausführung von Software.
ARM	Die ARM-Architektur ist ein 1983 vom britischen Computerunternehmen Acorn entwickeltes 32-Bit-Chip-Design, das seit 1990 von der aus Acorn ausgelagerten Firma ARM Limited weiterentwickelt wird. ARM steht für Advanced RISC Machines.
Binding	Bindings sind .NET Objekte, welche Verknüpfungen zwischen Properties schaffen. Gängigste Anwendung von Bindings sind Dependency Property Bindings in WPF, wo Control Properties auf Properties von Datenobjekten gebunden werden, um den jeweiligen Inhalt anzuzeigen oder zu verändern.
CLR	Abkürzung von Common Language Runtime.
COM	Abkürzung von Component Object Model.
CRUD	Abkürzung von Create, Read, Update, Delete – Dateneingabe und Modifikation.
Common Language Runtime	Die Common Language Runtime stellt eine gemeinsame Laufzeitumgebung für .NET Softwarekomponenten zur Verfügung. Es spielt dabei keine Rolle mehr, in welcher Sprache eine Komponente entwickelt wurde.
Component Object Model	Das Component Object Model ist eine von Microsoft entwickelte Technologie zur Erstellung von Softwarekomponenten, die unabhängig von der genutzten Programmiersprache eingesetzt werden können sollen.
Dependency Property	Stellt ein Property dar, welches durch Styling, Bindings, Animationen oder Vererbung gesetzt oder gelesen werden kann.
Desktop App	Desktop App ist die neue Bezeichnung für bisherige Windows Applikationen, welche auch unter Windows 8 und Windows RT auf dem Windows Desktop ausgeführt werden.
DTO	Abkürzung für Data Transfer Object. DTOs sind Objekte, welche Daten zwischen Prozessen verschieben. DTOs verfügen anders als Business Objekte über keine eigene Logik und werden nur zum Austausch von Daten genutzt.
Entity Framework	Das Entity Framework ist ein ORM Framework und stellt die Verbindung zwischen relationalen Datenbanken und den Domain-Objekten einer Applikation her.
Metro, Metro Style	Metro ist die alte Bezeichnung für die Design Sprache von Microsoft, welche mit dem Zune Interface eingeführt und mit Windows Phone Verbreitung gefunden hat. Metro wurde später aus rechtlichen Gründen umbenannt
Live Tile	Live Tiles sind aktive Kacheln von Apps (sowohl Windows Store Apps und Desktop Apps), welche auf den Start Screen gepinnt werden können. Neben der Möglichkeit, eine App zu starten können Live Tiles auch Inhalte darstellen.
POCO, PONO	Abkürzung für Plain Old CLR Object bzw. Plain Old .NET Object. Gemeint sind einfach Objekte bzw. Klassen, welche in der Regel von einem Framework konsumiert werden,

	welche aber nicht speziell dafür modelliert werden.
Start Screen	Der Start Screen ersetzt seit Windows 8 das Startmenü von Windows. Der Start Screen wird nicht mit Icons, sondern mit Live Tiles gefüllt.
Style	WPF Styles erlauben die Trennung der Konfiguration der Properties eines .NET Objekts von den jeweiligen Instanzen. Hauptnutzen hiervon ist die entkoppelte Definition des Aussehens von der App Struktur.
Task Parallel Library, TPL	Eine mit .NET 4.0 eingeführte Library im Namensraum System.Threading.Tasks, welche die Entwicklung nebenläufiger Routinen vereinfacht. Mit .NET 4.5 wurden in C# die Schlüsselwörter await und async eingeführt, welche sprachliche Unterstützung für dieses Feature anbieten.
Win32	Win32 beinhaltet die Programmierschnittstellen von Windows. Umgangssprachlich sind damit auch weitere Technologien gemeint wie COM oder DirectX. 64-bit Windows Varianten bieten nativ kein Win32 mehr an, sondern verfügen nur noch über eine dazu kompatible Schnittstelle namens WOW64.
Win64	Siehe Win32, aber für 64-bit Windows.
Windows 8, Windows 8.1	Windows 8 ist der Nachfolger von Windows 7. Windows 8 bietet eine neue, auf Touch-Bedienung optimierte Benutzeroberfläche.
Windows Desktop	Windows Desktop ist die neue Bezeichnung für den Desktop von Windows.
Windows NT	Windows NT ist eine Betriebssystemfamilie von Windows. Seit Windows 2000 sind fast alle bedeutenden Windows Varianten – einschliesslich Windows 8 und Windows RT – NT-basiert.
Windows Presentation Foundation	Windows Presentation Foundation (kurz: WPF) ist ein umfangreiches Grafik- und UI-Framework und Teil des .NET Frameworks 3.0 und höher.
Windows Runtime	Windows Runtime (kurz: WinRT) ist die Laufzeitumgebung für Windows Store Apps auf Windows 8, Windows RT und Windows Phone 8.
Windows RT	Windows RT ist eine Version von Windows für Geräte basierend auf der ARM Architektur. Windows RT und Windows 8 bieten das gleiche Bedienkonzept.
WinRT	Abkürzung von Windows Runtime. Achtung: Nicht dasselbe wie Windows RT!
Windows Store	Der Windows Store ist die primäre Bezugsquelle von Windows Store Apps auf Windows 8 und Windows RT.
Windows Store App	Eine neue Art von Apps, welche WinRT als Laufzeitumgebung nutzt und auf Windows 8 und Windows RT lauffähig ist.
WOW64	WOW64 implementiert Win32 als Wrapper für Win64 und stellt so die gleiche Funktionalität transparent zur Verfügung.
WPF	Abkürzung von Windows Presentation Foundation. Die Windows Presentation Foundation

ist das neue UI Framework, welches seit .NET 3.0 die alten Windows Forms ablöst.

Abbildung 67: Begriffe und Definitionen

Appendix C: Tabellen und Abbildungen

Abbildungsverzeichnis

Tabellenverzeichnis

Appendix D: Referenzen

[1] Bray, B. (3. April 2012). *Async in 4.5: Worth the Await*. Retrieved 16. December 2013 from .NET Framework Blog: http://blogs.msdn.com/b/dotnet/archive/2012/04/03/async-in-4-5-worth-the-await.aspx

[2] empira Software GmbH. (11. August 2009). *Home of PDFSharp and MigraDoc Foundation*. Retrieved 9. October 2013 from http://www.pdfsharp.net/

[3] Enkari, Ltd. (2012). *Open source dependency injector for .NET*. Retrieved 12. November 2013 from Ninject: http://www.ninject.org

[4] Foley, M. J. (9. August 2012). *Microsoft: Don't call it Metro. Call it 'Windows 8'.* Retrieved 1. October 2013 from Technology News - CNET News: http://news.cnet.com/8301-10805_3-57490379-75/microsoft-dont-call-it-metro-call-it-windows-8/

[5] H., X. (7. September 2012). Using Windows 8* WinRT API from desktop applications. *Intel Developer Zone Content Library* .

[6] Icaza, M. d. (15. September 2011). *WinRT demystified*. Retrieved 1. October 2013 from Miguel de Icaza's Blog: http://tirania.org/blog/archive/2011/Sep-15.html

[7] Jenkins, P., & Ginnivan, J. (18. October 2012). *MahApps.Metro Documentation*. Retrieved 26. October 2013 from http://mahapps.com/MahApps.Metro/

[8] Khandelwal, P. (15. November 2011). *Windows 7 API Code Pack*. Retrieved 15. December 2013 from CodePlex: http://www.nuget.org/packages/Windows7APICodePack/

[9] LeBlanc, B. (22. July 2009). *Windows 7 Team Blog*. Retrieved 30. September 2013 from Windows 7 Has Been Released to Manufacturing: http://blogs.windows.com/windows/archive/b/windows7/archive/2009/07/22/windows-7-has-been-released-to-manufacturing.aspx

[10] Liang, L. (28. December 2011). *DbEntry.Net (Lephone Framework)*. Retrieved 21. October 2013 from CodePlex: http://dbentry.codeplex.com

[11] Marukovich, A. (2. October 2013). *Using Windows 8 WinRT APIs in .NET Desktop Applications*. Retrieved 18. December 2013 from MSDN Blogs: Canadian Developer Connection: http://blogs.msdn.com/b/cdndevs/archive/2013/10/02/using-windows-8-winrt-apis-in-net-desktop-applications.aspx

[12] Microsoft Corp. (n.d.). From http://msdn.microsoft.com/de-de/library/bb385914.aspx

[13] Microsoft Corp. (2013). *.NET Framework mappings of Windows Runtime types*. Retrieved 15. December 2013 from Dev Center - Windows Store Apps: http://msdn.microsoft.com/en-us/library/windows/apps/hh995050(v=vs.85).aspx

[14] Microsoft Corp. (2013). *Code First Migrations*. Retrieved 21. October 2013 from Data Developer Center: http://msdn.microsoft.com/en-us/data/jj591621.aspx

[15] Microsoft Corp. (2. August 2012). *Configuring HTTP and HTTPS*. Retrieved 11. October 2013 from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/ms733768(v=vs.110).aspx

[16] Microsoft Corp. (2013). *Entity Framework Async Query & Save*. Retrieved 16. December 2013 from Data Developer Center: http://msdn.microsoft.com/en-us/data/jj819165.aspx

[17] Microsoft Corp. (16. November 2013). *Guidelines for Segoe UI Symbol icons*. Retrieved 21. November 2013 from Dev Center - Windows Store Apps: http://msdn.microsoft.com/en-us/library/windows/apps/jj841126.aspx

[18] Microsoft Corp. (16. November 2013). *IInputObject interface* . Retrieved 2. October 2013 from Dev Center - Desktop: http://msdn.microsoft.com/en-us/library/windows/desktop/bb761804(v=vs.85).aspx

[19] Microsoft Corp. (16. November 2013). *IInputPanelConfiguration interface*. Retrieved 2. October 2013 from Dev Center - Desktop: http://msdn.microsoft.com/en-us/library/windows/desktop/jj126267(v=vs.85).aspx

[20] Microsoft Corp. (16. November 2013). *KeyboardCapabilities class*. Retrieved 15. December 2013 from Dev Center - Windows Store Apps: http://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.input.keyboardcapabilities.aspx

[21] Microsoft Corp. (2013). *Optimistic Concurrency*. Retrieved 16. December 2013 from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/aa0416cz(v=vs.110).aspx

[22] Microsoft Corp. (4. December 2013). *Panorama control for Windows Phone*. Retrieved 10. December 2013 from Windows Phone Dev Center: http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff941104(v=vs.105).aspx

[23] Microsoft Corp. (4. December 2013). *Pivot control for Windows Phone*. Retrieved 14. December 2013 from MSDN: http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff941098(v=vs.105).aspx

[24] Microsoft Corp. (4. December 2013). *Theme resources for Windows Phone*. Retrieved 15. December 2013 from Windows Phone Dev Center: http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769552(v=vs.105).aspx

[25] Microsoft Corporation. (2. September 2013). *Index of UX guidelines for Windows Store apps*. Retrieved 1. October 2013 from Windows Dev Center: http://msdn.microsoft.com/en-us/library/windows/apps/hh465424.aspx

[26] Microsoft Corporation. (2013). *Volume Licensing reference guide for Windows 8 and Windows RT.*

[27] Microsoft Open Technologies, Inc. (1. November 2013). *Entity Framework*. Retrieved 2. November 2013 from CodePlex: http://entityframework.codeplex.com

[28] Niehus, O. (21. June 2013). *The new Windows UI*. Retrieved 1. October 2013 from MSDN Blogs: Oliver's Blog: http://blogs.msdn.com/b/olivnie/archive/2013/06/21/the-new-windows-ui.aspx

[29] Petzold, C. (August 2010). UI Frontiers: Multi-Touch Manipulation Events in WPF. *MSDN Magazine* .

[30] Schulte, R. (11. October 2012). *Windows 8 WinRT KeyboardCapabilities.KeyboardPresent is always true*. Retrieved 2. October 2013 from http://social.msdn.microsoft.com/Forums/windowsapps/en-US/5be268ba-b662-4a0e-ac82-ac1e90c31b56/windows-8-winrt-keyboardcapabilitieskeyboardpresent-is-always-true?forum=winappswithcsharp

[31] svick. (28. February 2012). *How to deploy a Metro App to the Desktop?* Retrieved 1. October 2013 from Stack Overflow: http://stackoverflow.com/questions/7451536/how-to-deploy-a-metro-app-to-the-desktop

[32] Turner, R. (27. January 2012). *An Accurate Windows 8 Platform Diagram?* Retrieved 1. October 2013 from BitCrazed: http://www.bitcrazed.com/post/2012/01/27/An-Accurate-Windows-8-Platform-Architecture-Diagram.aspx

[33] Xceed. (21. May 2013). *Extended WPF Toolkit Community Edition*. Retrieved 10. December 2013 from CodePlex: http://wpftoolkit.codeplex.com/

[34] Zwikstra, K. (9. April 2013). *Modern UI for WPF*. Retrieved 8. October 2013 from CodePlex: http://mui.codeplex.com